



HUMBOLDT-UNIVERSITY OF BERLIN
SCHOOL OF BUSINESS AND ECONOMICS
LADISLAUS VON BORTKIEWICZ CHAIR OF STATISTICS

STATISTICAL PROGRAMMING LANGUAGES
SEMINAR PAPER

Exploratory Data Analysis of Airbnb properties in Berlin

Silvia Ventoruzzo
(592252)

submitted to

Alla PETUKHINA

March 15, 2019

Contents

1	Introduction	1
2	Data Preparation	2
2.1	Berlin neighbourhoods and districts	2
2.2	Berlin VBB Zones	4
2.3	Airbnb listings' attributes	7
3	Exploratory Data Analysis	10
3.1	Descriptive statistics	10
3.2	Distribution plots	12
4	Price analysis	13
4.1	Correlation with price	14
4.2	Linear regression on price	15
5	Clustering	17
5.1	Choosing the number of clusters	17
5.2	k-means clustering	20
6	ShinyApp	20
7	Results and Conclusions	22
References		23
Appendix		25
A	Code	25
A.1	Data and analysis	25
A.2	Functions	49
A.3	ShinyApp	66

1 Introduction

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 (Salter, Jessica (*The Telegraph*), 2012). Now, they are a multibillion-dollar company where properties from all over the world are on rent (Bort, Julie (*Business Insider*), 2018).

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 Wang and Nicolau (2017) itself, but none so far specific for the city of Berlin. We therefore attempted an exploratory analysis of Airbnb listings in Berlin and, in particular, of their price. For this purpose linear regression on price was also run and properties were clustered.

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 "presence of car parks and fitness centers". But price determinants could be different in case of different types of accommodations, 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.

What is observed is that the dimension as well as the type of the accommodation positively influences the price. However, the location has almost no impact on the price, except for the case of being inside the circular with respect to the opposite. Clustering on the other hand did not deliver usable results, because of the absence of easily recognizable patterns. Further analysis of the price determinants and the clusters may shed more light into the topic.

The paper is divided into 7 sections. Section 2 will present the data used for this study and explained how it was prepared. Consecutively, 3 will run exploratory data analysis, both in form of tables and plots, on this data. Because of our interest in explaining property price, 4 will focus on this feature and show correlation and regression with respect to it. Furthermore, an attempt at clustering the Airbnb properties will be done in 5. After that it will be explained in 6 why a ShinyApp was developed to present this research. Finally, 7 we will present the results and draw some conclusions.

2 Data Preparation

For the analysis explained in this paper data was downloaded for a website independent from airbnb itself. Insideairbnb (Insideairbnb, 2019) 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 (Amt für Statistik Berlin-Brandenburg, 2019) and Geofabrik (Geofabrik, 2019) 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 Amt für Statistik Berlin-Brandenburg (2019) contain one polygon for each neighbourhood and additional information about them, out of which we only kept the district.

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",
3                         "Berlin-Ortsteile-polygon.shp", fsep="/"))
4
5 # Object with the neighbourhoods (and respective district)
6 berlin_neighbourhood_sf = berlin %>%
7   dplyr::rename(id      = Name,
```

```

8     group = BEZNAME) %>%
9     dplyr::select(id, group, geometry) %>%
10    dplyr::arrange(group)

```

However, one of the neighbourhoods, 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 `dplyr`.

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.

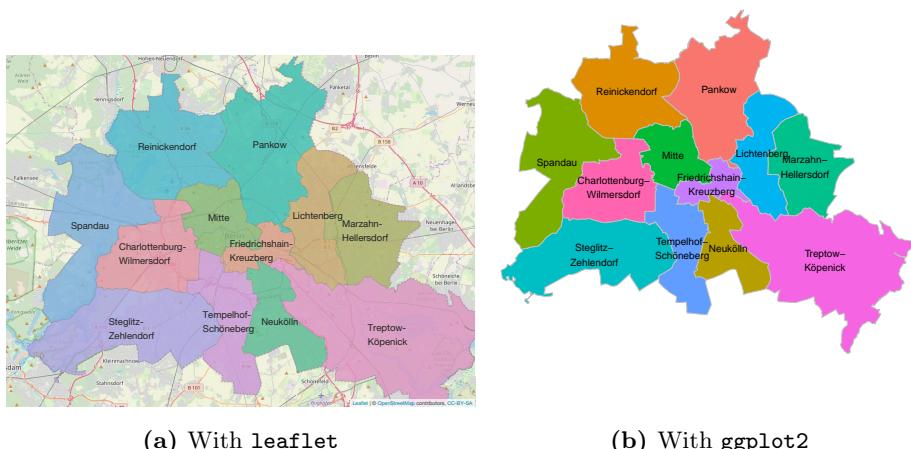


Figure 1: Maps of the Berlin Districts `berlin_districts_neighbourhoods_maps.R`

2.2 Berlin VBB Zones

The VBB (Verkehrsverbund Berlin-Brandenburg) is "the public transport authority covering the federal states of Berlin and Brandenburg" (Verkehrsverbund Berlin-Brandenburg, 2019). 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.

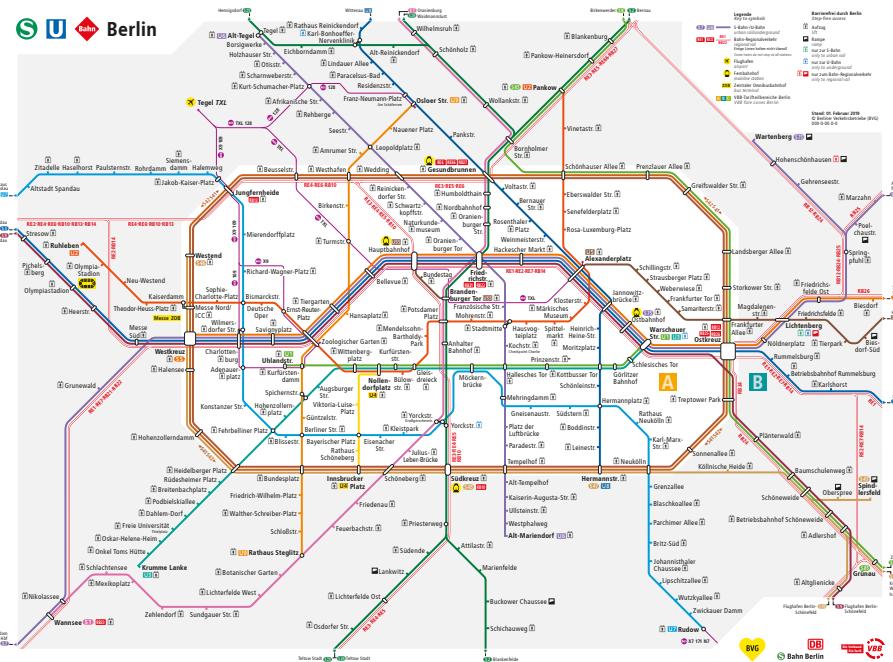


Figure 2: Network Map of Berlin Areas A and B (Source: Verkehrsverbund Berlin-Brandenburg (2019))

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",
8   "Treptower Park", "Sonnenallee", "Neukölln", "Hermannstraße",
9   "Tempelhof", "Südkreuz"),
10  stringsAsFactors = FALSE) %>%
11  tibble::rownames_to_column(var = "order") %>%
12  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 af 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)
```

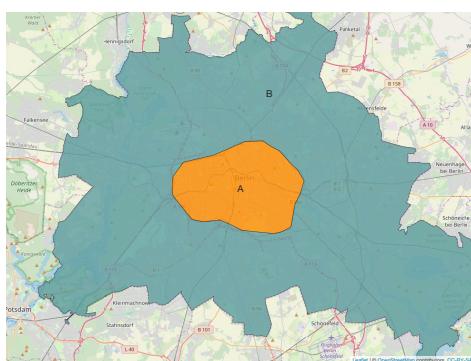
Finally, we bind the two objects by row and calculate their intersections thanks to the function `st_intersection` from the package `sf`. We then define the area names according to how many times the two previous polygons intersect:

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

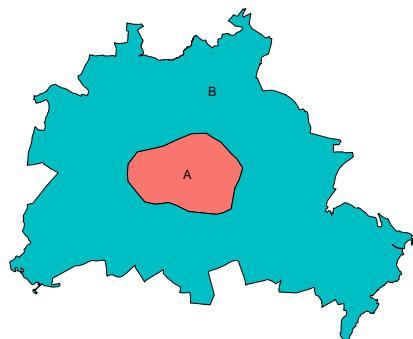
Listing 7: |berlin_vbb_zones.R|

```
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`.



(a) With leaflet



(b) With ggplot

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)
```

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:

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

where ϕ corresponds to the latitude and ψ to the longitude of the two points.

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    as.data.frame() %>%
12    data.table::setnames(as.character(reference$id))
13   # Calculate how many "reference" are within "distance"
14   point_distance[,var_name_count] = rowSums(point_distance <= distance)
15   %>%
16   as.factor()

```

```

17 # Calculate the distance to the nearest "reference"
18 point_distance[,var_name_dist] = apply(point_distance[,-ncol(point_
19   distance)], ,
20     MARGIN = 1,
21     FUN = min) %>%
22     round(0)
23 
24 # Insert this information into the main DF
25 main = point_distance %>%
26   dplyr::mutate(id = main$id) %>%
27   dplyr::select(id, var_name_count, var_name_dist) %>%
28   dplyr::right_join(main, by = "id")
29
30 return(main)
31 }
```

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

Listing 10: |data_preparation.R|

```

1 listings_season_availability = listings_calendar %>%
2   dplyr::group_by(id, year_season) %>%
3   dplyr::summarize(count      = sum(available == TRUE),
4                     season_days = n(),
5                     proportion = round(count/season_days*100, 4)) %>%
6   dplyr::arrange(id, year_season) %>%
7   dplyr::ungroup() %>%
8   dplyr::mutate(season_av = paste(year_season, "availability",
9                                     sep = "_") %>%
10                         tolower() %>%
11                         factor(levels = paste(year_season,
12                                         "availability",
13                                         sep = "_") %>%
14                                         tolower() %>%
15                                         unique())))
16   dplyr::select(-count, -year_season, -season_days) %>%
17   tidyr::spread(season_av, proportion)
```

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 1.

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 1: 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")

```

```

38
39   prop_var = colsplit(string = proportion_df$var_fact, pattern = "\\\.", 
40   names = c("variable", "factor"))
41
42   proportion = prop_var %>%
43     cbind(proportion_df) %>%
44     dplyr::select(-var_fact) %>%
45     dplyr::mutate(proportion = round(proportion, 4)*100,
46                   proportion = as.character(proportion) %>% paste("%"))
47
48   summary = frequency %>%
49     dplyr::inner_join(proportion, by = c("variable", "factor")) %>%
      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.

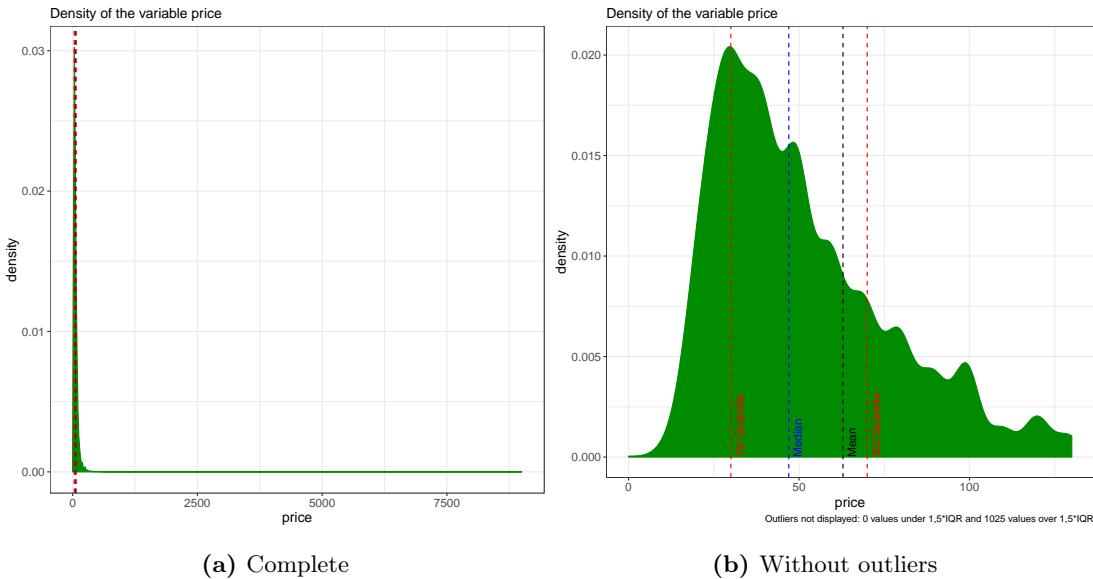


Figure 4: Distribution of the variable price distribution_plot.R

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.

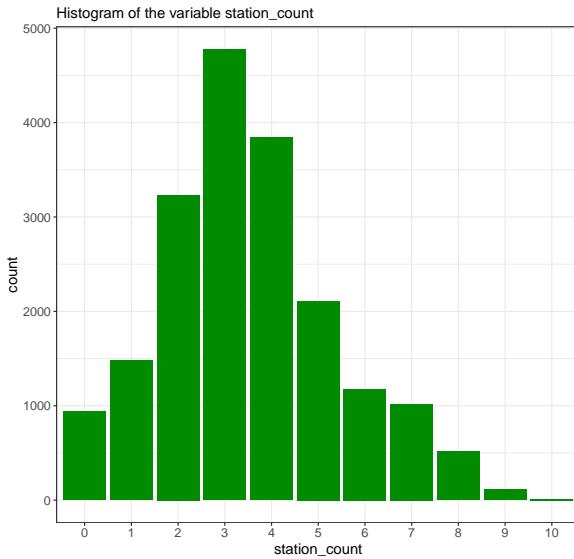


Figure 5: Distribution of the variable station_count distribution_plot.R

Having successfully 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.

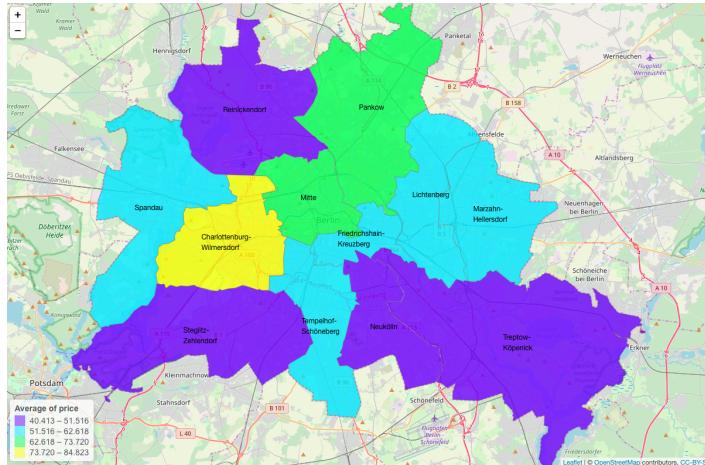


Figure 6: Distribution of the average of price across Berlin's districts var_avg_map.R

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 produced by `corrplot` from the package `corrplot` may not be the most easily readable in this case, especially because of the large number of variables.

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 }
```

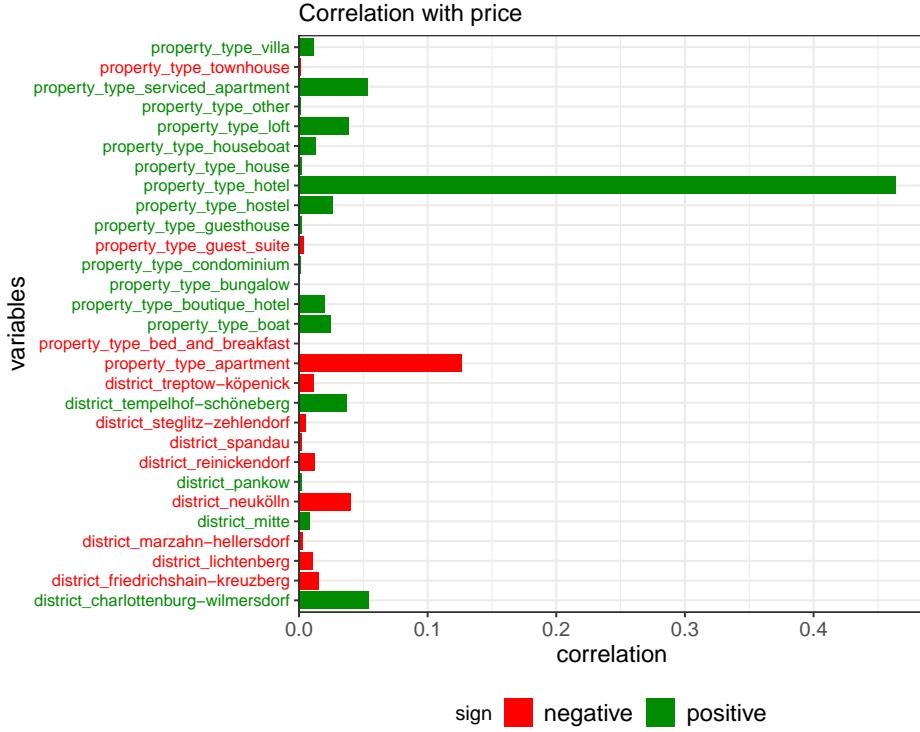


Figure 7: Sample of plot of correlation with price correlation_plot.R

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. For this purpose we used again the data with the categorical variables transformed to dummies. The linear regression was then run using all other variables, except *id*, *long*, *lat*, *neighbourhood* and *listing_url*, as regressors.

On table 2 you can see a sample of the results. A positive coefficient means that the variable has a positive impact on the dependent variable, the higher this regressor is, the higher *price* will be. On the other side, a negative coefficient suggests that a higher value of the independent variable will result in a lower value of *price*. Moreover, a variable is considered significant if the p-value is lower than the significance level, set at 5% in this case, since it rejects the null hypothesis for that regressor coefficient to be equal to zero (Moyé, 2006).

variable	coefficient	significant
(Intercept)	12.69	FALSE
host_listings_count	-0.73	TRUE
accommodates	16.31	TRUE

Table 2: Sample of linear regression results

Furthermore, to check the validity of the model one should also look at the distribution of the residuals, which should be normal. One can see from table 3 and picture 8 that this is here not the case.

min	1Q	median	3Q	max	iqr	mean	sd
-947.97	-18.72	-3.31	12.32	8625.31	31.04	0.00	138.02

Table 3: Descriptive statistics of regression residuals

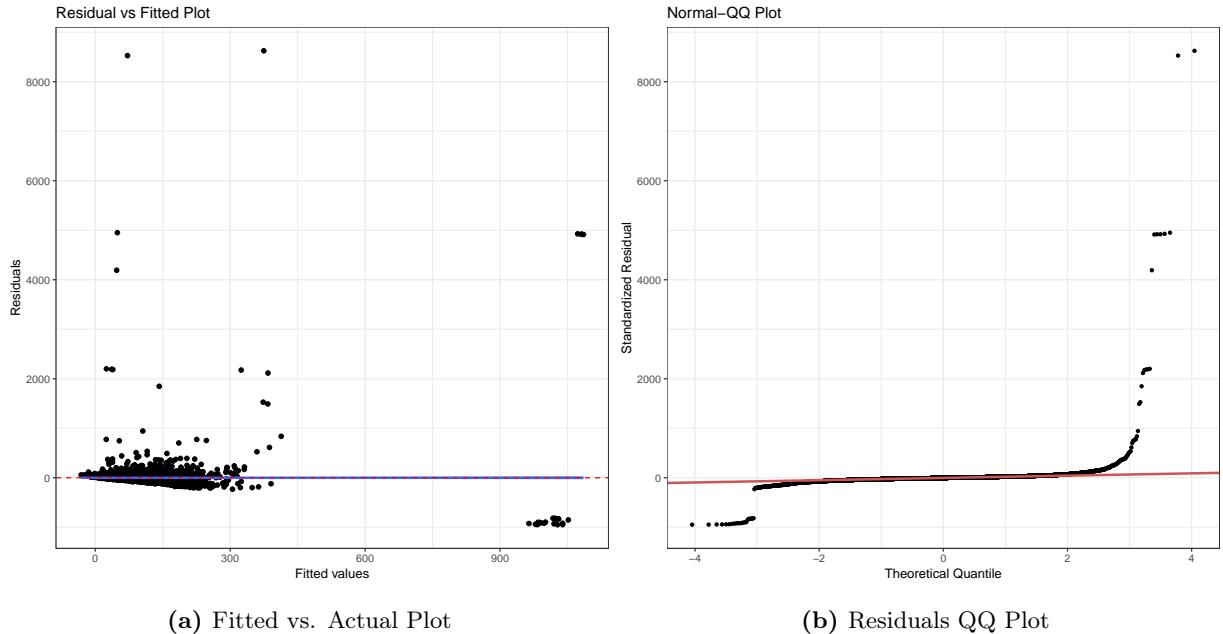


Figure 8: Plotting the clusters

Finally, the R^2 for this model, that is the proportion of the variance of *price* captured by the regressors, is only 0.1298, meaning that there is much room for improvement. One can test linear regression with other variables using the ShinyApp described in 6.

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 20th 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)

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

where $dist(\mathbf{c}_i, \mathbf{x})$ is the Euclidean distance between the centroid of the i^{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 5.1 will deal with the process of choosing the appropriate number of clusters to be used in the clustering algorithm, while subsection 5.2 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 not 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 plots 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  # Plot tve against k values
26  plot = ggplot(data = tve, aes(x = clusters, y = tve)) +
27    geom_line(color = "grey") +
28    geom_point(color = "red") +

```

```

29   scale_x_continuous(breaks = plot_breaks) +
30 
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 
36   theme_bw() +
37 
38   theme(axis.text.x = element_text(size = rel(1.2)),
39         axis.text.y = element_text(size = rel(1.2)),
40         axis.title.x = element_text(size = rel(1.2)),
41         axis.title.y = element_text(size = rel(1.2)))
42 
43 
44   return(plot)
45 }

```

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.

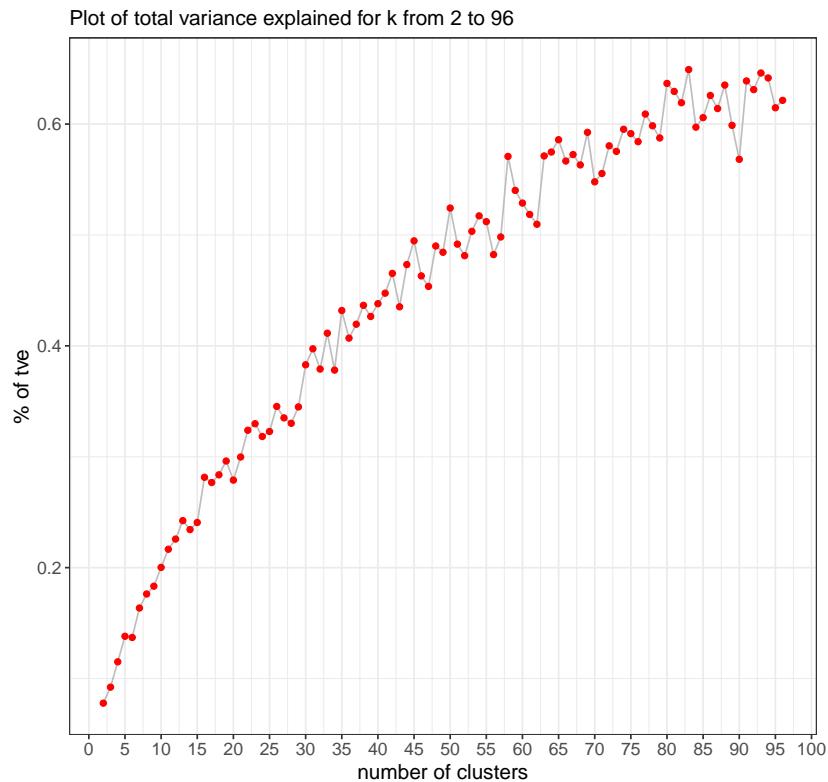
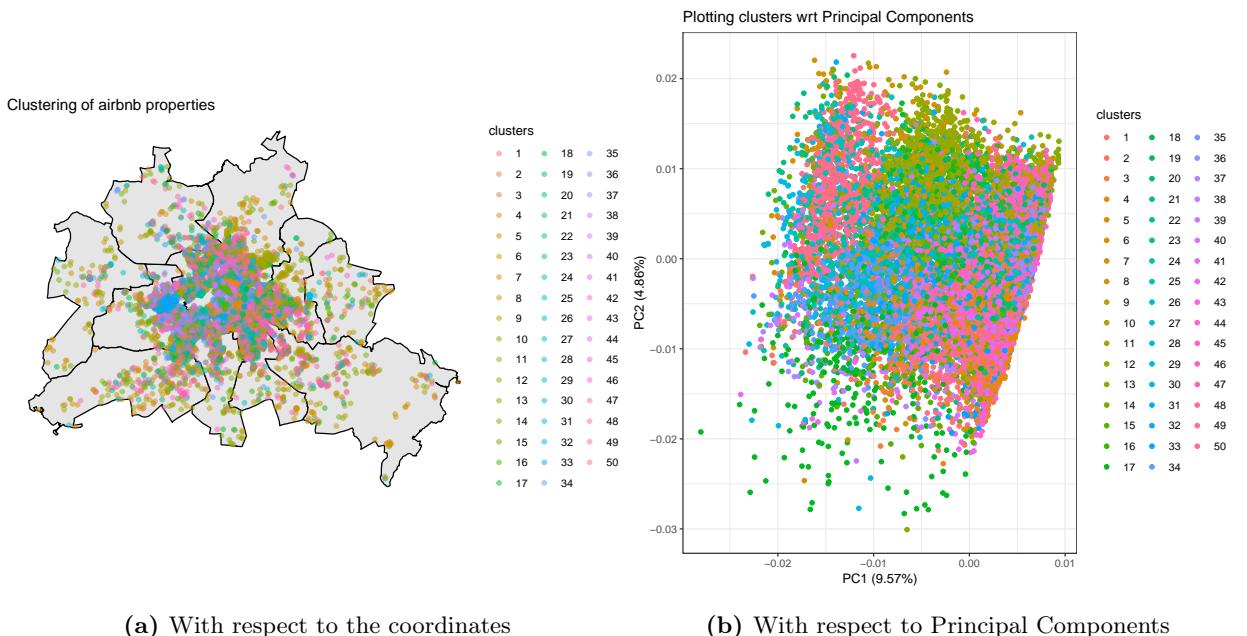


Figure 9: Plot of the total variance explained against different values of k number_of_clusters.R

5.2 k-means clustering

After selecting the number of clusters to use, one can apply it 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 10a. Another one can be to plot them with respect to the first two principal components, as in figure 10b.



(a) With respect to the coordinates

(b) With respect to Principal Components

Figure 10: Plotting the clusters

Because of the large amount of variables, points and clusters, it is not easy to visually display the groups. This task can be left for further studies.

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-friedly 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)`.

Because of its interactivity 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 code for the App can be found in the Appendix at subsection A.3. It is divided into four tabs: the first one is just introductory, presenting the project and the App itself; the second display Berlin maps and descriptive statistics according to section 3; the third relates to price analysis explained in section 4; the fourth and final one enables the user to perform clustering, like illustrated in section 5. The ShinyApp is reachable at this link: <https://silviav.shinyapps.io/airbnb-berlin/>.

7 Results and Conclusions

Airbnb is a relevant subject in this day and age where sharing economy has affected most business fields. Therefore there have been many studies about Airbnb, in relation to hotels (Wang and Nicolau, 2017), and to the rental market (Coles et al., 2017).

In this study we focused on the properties based in Berlin, the capital of Germany. We looked at how the properties are distributed across the cities, noting that most of them are in the central part of town, the one inside the circular line. We further explored the distribution of different attributes, identifying differences across the multiple the districts and neighbourhoods.

Location is just relatively a good indicator of price. Being inside the circular line leads to higher prices, but the districts are not as strong of a guide. The amount of stations within 1km is more related to price than the amount of attractios wihin 2km, probably because of the well functioning transportation system in Berlin. Trustworthiness of the host is also not strongly correlated with price, only not having a profile pics has a high regression coefficient. Finally, the type of accomodation is probably the information that mostly predicts price. Hotels lead to much higher prices, as well as booking a whole apartent compared to just a room, or part of it.

Unfortunately, clustering did not give us much insight into the distribution of the properties and their common attributs. We will leave it to future research to try different clustering methods and maybe test other variables.

The same can be said for linear regression, which did not deliver such reliable results, since the model did not respect some of the assumptions. One can try running the model with fewer or other variables, or perform Quantile Regression as in Wang and Nicolau (2017).

One can however already try different variables and number of clusters in the web interface developed for this study.

References

- AMT FÜR STATISTIK BERLIN-BRANDENBURG (2019): “Statistik Berlin-Brandenburg BerlinOpenData Geometrien,” <https://www.statistik-berlin-brandenburg.de/produkte/opendata/geometrienOD.asp?Kat=6301>, online; last accessed on 12. November 2018.
- BOCK, H.-H. (2007): “Clustering methods: a history of k-means algorithms,” in *Selected contributions in data analysis and classification*, Springer, 161–172.
- BORT, JULIE (BUSINESS INSIDER) (2018): “Insideairbnb Get the Data,” <https://www.businessinsider.de/airbnb-profit-revenue-2018-2?r=US&IR=T>, online; last accessed on 22. February 2019.
- COLES, P. A., M. EGESDAL, I. G. ELLEN, X. LI, AND A. SUNDARARAJAN (2017): “Airbnb usage across new York City neighborhoods: Geographic patterns and regulatory implications,” *Forthcoming, Cambridge Handbook on the Law of the Sharing Economy*.
- GEOFABRIK (2019): “Geofabrik Downloads: Europe, Germany, Berlin,” <http://download.geofabrik.de/europe/germany/berlin.html>, online; last accessed on 20. November 2018.
- 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.
- INSIDEAIRBNB (2019): “Insideairbnb Get the Data,” <http://insideairbnb.com/get-the-data.html>, online; last accessed on 04. March 2019.
- 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*.

MOYÉ, L. A. (2006): *Statistical reasoning in medicine: the intuitive P-value primer*, Springer Science & Business Media.

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.

SALTER, JESSICA (THE TELEGRAPH) (2012): “Airbnb: The story behind the /1.3bnroom – lettingwebsite,” online; last accessed on 15. March 2019.

TAN, P.-N. (2018): *Introduction to data mining*, Pearson Education India, chap. 8.

VERKEHRSVERBUND BERLIN-BRANDENBURG (2019): The company VBB, <https://www.vbb.de/en/about-us/the-company-vbb>, online; last accessed on 14. March 2018.

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.

A Code

A.1 Data and analysis

Listing 17: |berlin_districts_neighbourhoods.R|

```
1 # Load packages
2 needed_packages = c("dplyr", "data.table", "sf", "tibble")
3 for (package in needed_packages) {
4   if (!require(package, character.only=TRUE))
5     {install.packages(package, character.only=TRUE)}
6   library(package, character.only=TRUE)
7 }
8 rm("needed_packages", "package")
9
10 # Set working directory to the one where the file is located
11
12 # This works when run directly
13 # setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
14
15 # This works when sourced
16 setwd(dirname(sys.frame(1)$ofile))
17
18
19 # Load shapefiles
20 berlin = sf::st_read(file.path(getwd(), "Data",
21                       "Berlin-Ortsteile-polygon.shp", fsep="/"))
22
23 # Object with the neighbourhoods (and respective district)
24 berlin_neighbourhood_sf = berlin %>%
25   dplyr::rename(id      = Name,
26                 group = BEZNAME) %>%
27   dplyr::select(id, group, geometry) %>%
28   dplyr::arrange(group)
29
30 # Buckow is composed of two separate parts, so we need to join them
31 berlin_neighbourhood_singlebuckow_sf = berlin_neighbourhood_sf %>%
32   dplyr::group_by(id, group) %>%
33   dplyr::summarize(do_union = TRUE)
34
35 # Object with the districts
36 berlin_district_sf = berlin_neighbourhood_sf %>%
```

```

37   dplyr::group_by(group) %>%
38   dplyr::summarize(do_union = TRUE) %>%
39   dplyr::mutate(id = group)
40
41 # Create dataframes with the names for plotting
42 # Neighbourhoods
43 berlin_neighbourhoods_names = berlin_neighbourhood_sf %>%
44   sf::st_centroid() %>%
45   sf::st_coordinates() %>%
46   base::as.data.frame() %>%
47   dplyr::rename(long = X,
48                 lat = Y) %>%
49   dplyr::mutate(id = berlin_neighbourhood_sf$id,
50                 group = berlin_neighbourhood_sf$group,
51                 name = gsub("-", "-<br>", berlin_neighbourhood_sf$id))
52
53 # Districts
54 berlin_districts_names = berlin_district_sf %>%
55   sf::st_centroid() %>%
56   sf::st_coordinates() %>%
57   base::as.data.frame() %>%
58   dplyr::rename(long = X,
59                 lat = Y) %>%
60   dplyr::mutate(id = berlin_district_sf$id,
61                 group = berlin_district_sf$group,
62                 name = gsub("-", "-<br>", berlin_district_sf$id))
63
64 # Remove not needed data
65 rm("berlin")

```

Listing 18: |berlin_districts_neighbourhoods_maps.R|

```

1 # Load packages
2 needed_packages = c("leaflet", "ggplot2", "htmltools", "mapview")
3 for (package in needed_packages) {
4   if (!require(package, character.only=TRUE))
5     {install.packages(package, character.only=TRUE)}
6   library(package, character.only=TRUE)
7 }
8 rm("needed_packages")
9
10 # Set working directory to the one where the file is located

```

```

11
12 # This works when run directly
13 setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
14
15 # This works when sourced
16 # setwd(dirname(sys.frame(1)$ofile))
17
18 # Load file where the sf object is created
19 source("berlin_districts_neighbourhoods.R", chdir = TRUE)
20
21 # Colors for polygons
22 district_colors = colorspace::rainbow_hcl(n = 12)
23
24 # Leaflet map of the districts
25 berlin_district_leaflet = leaflet() %>%
26   addTiles() %>%
27   addPolygons(data = berlin_district_sf,
28     weight = 1, smoothFactor = 1, fillOpacity = 0.8,
29     fillColor = district_colors, color = "grey") %>%
30   addLabelOnlyMarkers(data = berlin_districts_names,
31     lng = ~long, lat = ~lat,
32     label = ~lapply(name, htmltools::HTML),
33     labelOptions = labelOptions(noHide = TRUE,
34       direction = 'center',
35       textOnly = TRUE,
36       textSize = "20px")) %>%
37   setView(lng = berlin_neighbourhoods_names$long
38         [berlin_neighbourhoods_names$id == "Mitte"],
39         lat = berlin_neighbourhoods_names$lat
40         [berlin_neighbourhoods_names$id == "Mitte"],
41         zoom = 11)
42 berlin_district_leaflet
43
44 # mapview::mapshot(berlin_district_leaflet,
45 #   file = "berlin_district_leaflet.pdf",
46 #   remove_controls = c("zoomControl", "layersControl",
47 #     "homeButton", "scaleBar"))
48
49
50 # ggplot map
51 ggplot() +

```

```

52   geom_sf(data = berlin_district_sf,
53           show.legend = FALSE, color = "grey") +
54   coord_sf(datum = NA) +
55   aes(fill = district_colors) +
56   geom_text(data = berlin_districts_names,
57             aes(x = long, y = lat,
58                  label = gsub("<br>", "\n", name),
59                  hjust = "center")) +
60   theme_classic() +
61   theme(plot.title = element_text(hjust = 0.5),
62         axis.title.x = element_blank(),
63         axis.title.y = element_blank())
64
65 # dev.copy2pdf(file = "berlin_district_ggplot.pdf")
66 # dev.copy2pdf(file = "../SeminarPaper/berlin_district_ggplot.pdf")

```

Listing 19: |berlin_vbb_zones.R|

```

1 # Load packages
2 needed_packages = c("dplyr", "data.table", "sf", "tibble")
3 for (package in needed_packages) {
4   if (!require(package, character.only=TRUE))
5     {install.packages(package, character.only=TRUE)}
6   library(package, character.only=TRUE)
7 }
8 rm("needed_packages", "package")
9
10 # Set working directory to the one where the file is located
11
12 # This works when run directly
13 # setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
14
15 # This works when sourced
16 setwd(dirname(sys.frame(1)$ofile))
17
18
19 # Load helpers
20 source(file.path(getwd(), "Helpers", "points_midpoint.R", fsep="/"))
21
22 # Load shapefiles
23 berlin = sf::st_read(file.path(getwd(), "Data",
24                       "Berlin-Ortsteile-polygon.shp", fsep="/"))

```

```

25 stations = sf::st_read(file.path(getwd(), "Data",
26                         "gis_osm_transport_free_1.shp", fsep="/"))
27
28 # Create dataframe with names of stations on the Ringbahn (delimits Area A)
29 ringbahn_names_df = base::data.frame(
30   id = c("Südkreuz", "Schöneberg", "Innsbrucker Platz", "Bundesplatz",
31         "Heidelberger Platz", "Hohenzollerndamm", "Halensee", "Westkreuz",
32         "Messe Nord/ICC", "Westend", "Jungfernheide", "Beusselstraße",
33         "Westhafen", "Wedding", "Gesundbrunnen", "Schönhauser Allee",
34         "Prenzlauer Allee", "Greifswalder Straße", "Landsberger Allee",
35         "Storkower Straße", "Frankfurter Allee", "Ostkreuz",
36         "Treptower Park", "Sonnenallee", "Neukölln", "Hermannstraße",
37         "Tempelhof", "Südkreuz"),
38   stringsAsFactors = FALSE) %>%
39   tibble::rownames_to_column(var = "order") %>%
40   dplyr::mutate(order = as.numeric(order))
41
42 # Create sf object of Area A
43 berlin_vbb_A_sf = stations %>%
44   dplyr::filter(fclass %like% "railway") %>%
45   dplyr::rename(id = name) %>%
46   dplyr::mutate(id = gsub("Berlin ", "", id),
47                 id = gsub("Berlin-", "", id),
48                 id = gsub(" *\\(.*)?\\) *", "", id),
49                 id = gsub("S ", "", id),
50                 id = gsub("U ", "", id)) %>%
51   points_midpoint() %>%
52   dplyr::right_join(ringbahn_names_df, by = "id") %>%
53   dplyr::arrange(order) %>%
54   dplyr::select(long, lat) %>%
55   base::as.matrix() %>%
56   base::list() %>%
57   sf::st_polygon() %>%
58   sf::st_sfc() %>%
59   sf::st_sf(crs = sf::st_crs(berlin))
60
61 # Create sf object of entire Berlin
62 berlin_sf = berlin %>%
63   dplyr::summarize(do_union = TRUE)
64
65 # Bind and intersect to create sf object with both VBB Zones (A and B)

```

```

66 berlin_vbb_AB_sf = berlin_vbb_A_sf %>%
67   base::rbind(berlin_sf) %>%
68   sf::st_intersection() %>%
69   dplyr::mutate(id = ifelse(n.overlaps > 1, "A", "B")) %>%
70   dplyr::select(-n.overlaps, -origins) %>%
71   dplyr::arrange(desc(id))
72
73 # Create dataframes with coordinates where to show zones' names on map
74 berlin_vbb_A_names = berlin_vbb_A_sf %>%
75   sf::st_centroid() %>%
76   sf::st_coordinates() %>%
77   base::as.data.frame() %>%
78   dplyr::rename(long = X,
79                 lat = Y) %>%
80   dplyr::mutate(id = "A")
81 berlin_vbb_B_names = berlin_sf %>%
82   sf::st_bbox() %>%
83   base::as.matrix() %>%
84   base::t() %>%
85   as.data.frame() %>%
86   dplyr::transmute(long = (xmax + xmin)/2,
87                     lat = (3*ymax + ymin)/4) %>%
88   dplyr::mutate(id = "B")
89 berlin_vbb_AB_names = berlin_vbb_A_names %>%
90   rbind(berlin_vbb_B_names) %>%
91   dplyr::mutate(name = gsub("-", "-<br>", id))
92
93 # Remove not needed data
94 rm("berlin", "berlin_sf", "ringbahn_names_df", "stations",
95    "berlin_vbb_A_sf", "berlin_vbb_A_names", "berlin_vbb_B_names")
96 rm(list=lsf.str()) # All functions

```

Listing 20: |berlin_vbb_zones_maps.R|

```

1 # Load packages
2 needed_packages = c("leaflet", "ggplot2", "htmltools", "mapview")
3 for (package in needed_packages) {
4   if (!require(package, character.only=TRUE))
5     {install.packages(package, character.only=TRUE)}
6   library(package, character.only=TRUE)
7 }
8 rm("needed_packages")

```

```

9
10 # Set working directory to the one where the file is located
11
12 # This works when run directly
13 setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
14
15 # This works when sourced
16 # setwd(dirname(sys.frame(1)$ofile))
17
18
19 # Load file where the sf object is created
20 source("berlin_vbb_zones.R", chdir = TRUE)
21
22 # Leaflet map
23 berlin_vbb_zones_leaflet = leaflet() %>%
24   addTiles() %>%
25   addPolygons(data = berlin_vbb_AB_sf,
26               weight = 1, smoothFactor = 1, fillOpacity = 0.8,
27               fillColor = c("cadetblue", "darkorange"),
28               color = "black") %>%
29   addLabelOnlyMarkers(data = berlin_vbb_AB_names,
30                       lng = ~long, lat = ~lat,
31                       label = ~lapply(name, htmltools::HTML),
32                       labelOptions = labelOptions(noHide = TRUE,
33                                         direction = 'center',
34                                         textOnly = TRUE,
35                                         textSize = "20px")) %>%
36   setView(lng = berlin_vbb_AB_names$long
37           [berlin_vbb_AB_names$id == "A"],
38           lat = berlin_vbb_AB_names$lat
39           [berlin_vbb_AB_names$id == "A"],
40           zoom = 11)
41 berlin_vbb_zones_leaflet
42
43 # mapview::mapshot(berlin_vbb_zones_leaflet,
44 #                     file = "berlin_vbb_zones_leaflet.pdf",
45 #                     remove_controls = c("zoomControl", "layersControl",
46 #                                         "homeButton", "scaleBar"))
47 # mapview::mapshot(berlin_vbb_zones_leaflet,
48 #                     file = "../SeminarPaper/berlin_vbb_zones_leaflet.pdf",
49 #                     remove_controls = c("zoomControl", "layersControl",

```

```

50 #                                         "homeButton", "scaleBar"))
51
52
53 # ggplot map
54 ggplot() +
55   geom_sf(data = berlin_vbb_AB_sf, show.legend = FALSE, color = "black") +
56   coord_sf(datum = NA) +
57   aes(fill = c("darkorange", "cadetblue")) +
58   geom_text(data = berlin_vbb_AB_names,
59             aes(x = long, y = lat, label = id, hjust = "center"),
60             size = 5) +
61   theme_classic() +
62   theme(plot.title = element_text(hjust = 0.5),
63         axis.title.x = element_blank(),
64         axis.title.y = element_blank())
65
66 # dev.copy2pdf(file = "berlin_vbb_zones_ggplot.pdf")
67 # dev.copy2pdf(file = "../SeminarPaper/berlin_vbb_zones_ggplot.pdf")

```

Listing 21: |data_preparation.R|

```

1 rm(list=ls(all = TRUE))
2
3 # Install and load needed packages
4 needed_packages = c("tidyverse",
5                      "geosphere",
6                      "readr",
7                      "rstudioapi",
8                      "Jmisc",
9                      "sp",
10                     "sf")
11 for (package in needed_packages) {
12   if (!require(package, character.only=TRUE)) {
13     install.packages(package, character.only=TRUE)}
14   library(package, character.only=TRUE)
15 }
16 rm("needed_packages", "package")
17
18 # Set working directory to the one where the file is located
19
20 # This works when run directly
21 # setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

```

```

22
23 # This works when sourced
24 setwd(dirname(sys.frame(1)$ofile))
25
26 # Load helper functions and other scripts
27 source(file.path(getwd(), "SPL_Berlin_VBB_Zones",
28                 "berlin_vbb_zones.R", fsep="/"), chdir = TRUE)
29 source(file.path(getwd(), "SPL_Berlin_Districts_Neighbourhoods",
30                 "berlin_districts_neighbourhoods.R", fsep="/"), chdir =
31                 TRUE)
32 Jmisc::sourceAll(file.path(getwd(), "Helpers", fsep="/"))
33
34 # Load shapefile
35 stations = sf::st_read(file.path(getwd(), "Data", "spatial_data",
36                         "gis_osm_transport_free_1.shp", fsep="/"))
37
38 # Print code development
39 print("Spatial data loaded.")
40
41 # Load dataframes
42 listings_summarized = read_csv(file.path(getwd(), "Data", "airbnb_data",
43                                 "listings.csv", fsep="/"),
44                                 na = c("NA", ""),
45                                 locale = locale(encoding = "UTF-8"))
46 listings_detailed = read_csv(file.path(getwd(), "Data", "airbnb_data",
47                               "listings.csv.gz", fsep="/"),
48                               na = c("NA", ""),
49                               locale = locale(encoding = "UTF-8"))
50 listings_calendar = read_csv(file.path(getwd(), "Data", "airbnb_data",
51                               "calendar.csv.gz", fsep="/"),
52                               na = c("NA", ""),
53                               locale = locale(encoding = "UTF-8"))
54
55 # Print code development
56 print("Airbnb datasets uploaded.")
57
58 # Join dataframes, first clean and keep only variables of interest
59 listings = df_join_clean(df1 = listings_detailed, df2 = listings_summarized)
60
61 # Check if there are missing values

```

```

62 apply(listings, 2, function(x) any(is.na(x)))
63
64 # host_is_superhost, # host_has_profile_pic, host_identity_verified
65 listings = listings %>%
66   dplyr::mutate(host_is_superhost      = ifelse(is.na(host_is_superhost),
67                           FALSE, host_is_superhost),
68   host_has_profile_pic    = ifelse(is.na(host_has_profile_pic
69                           ),
70                           FALSE, host_has_profile_
71                           pic),
72   host_identity_verified = ifelse(is.na(host_identity_
73                           verified),
74                           FALSE, host_identity_
75                           verified))
76
76 # review_scores_rating
77 # We insert the average of the review_scores_rating
78 listings %>%
79   dplyr::summarize(mean = mean(review_scores_rating, na.rm = TRUE), # 77
80                     median = median(review_scores_rating, na.rm = TRUE), # 95
81                     mode = getmode(review_scores_rating)) %>% # 100
82   round(0)
83 # mean might be influenced by outliers, being it so different from median
84 # and mode, and mode is too high, so I will substitute the missing values
85 # with the median
86 listings = listings %>%
87   dplyr::mutate(review_scores_rating = ifelse(is.na(review_scores_rating),
88                           median(review_scores_rating,
89                           na.rm = TRUE),
90                           review_scores_rating))
91 # bedrooms and beds
92 # Being the amount of missing values in these features relative small, we
93 # can derive their value from the other variables:
94 listings = listings %>%
95   # If beds is NA, but bedrooms has a valid value, we set beds with the
96   # number of bedrooms
97   dplyr::mutate(beds      = ifelse(is.na(beds) & !is.na(bedrooms),
98                           bedrooms, beds),
99   # If bedrooms is NA, but beds has a valid value, we set
100  # bedrooms with the number of beds

```

```

92     bedrooms = ifelse(is.na(bedrooms) & !is.na(beds), beds,
93                         bedrooms),
94                         # If both beds and bedrooms are NA: 1
95                         # Since in this case the property can accomodate only one
96                         # person, we assume that it has 1 bed and 1
97                         # bedroom (even if it might not
98                         # be separate)
99     beds      = ifelse(is.na(beds) & is.na(bedrooms), 1, beds),
100    bedrooms = ifelse(beds == 1 & is.na(bedrooms), 1, bedrooms)

101
102
103 # Check if there are missing values
104 apply(listings, 2, function(x) any(is.na(x)))

105
106 # Listings selection
107 # Keep only listings with at least one customer review, as per Wang and
108 # Nicolau (2017)
109
110 listings %>%
111   dplyr::filter(number_of_reviews == 0) %>%
112   dplyr::summarize(count = n())
113
114 listings = listings %>%
115   dplyr::filter(number_of_reviews > 0)

116
117 # Print code development
118 print("Missing values cleaned.")

119
120 ## New variables
121
122 # Areas: Not doing this with berlin_sf because the polys_sf need to have
123 # geometry sfc_POLYGON
124 # district
125
126 listings = point_in_polygons(points_df = listings,
127                               polys_sf    = berlin_district_sf,
128                               var_name    = "district")

129 # vbb_zone
130
131 listings = point_in_polygons(points_df = listings,
132                               polys_sf    = berlin_vbb_AB_sf,
133                               var_name    = "vbb_zone")

134 # neighbourhood
135
136 listings = point_in_polygons(points_df = listings,
137                               polys_sf    = berlin_neighbourhood_sf,
138                               var_name    = "neighbourhood")

```

```

127 # For the sake of consistency, we will delete these listings
128 listings = listings %>%
129   filter(!is.na(district) & !is.na(vbb_zone))
130
131 # Stations
132 railway_stations_df = stations %>%
133   dplyr::filter(fclass %like% "railway") %>%
134   dplyr::rename(id      = name) %>%
135   dplyr::mutate(id      = gsub("Berlin ", "", id),
136                  id      = gsub("Berlin-", "", id),
137                  id      = gsub(" *\\(.*)\\*", "", id),
138                  s_bahn = ifelse(startsWith(id, "S "), TRUE, FALSE),
139                  u_bahn = ifelse(startsWith(id, "U "), TRUE, FALSE),
140                  id      = gsub("S ", "", id),
141                  id      = gsub("U ", "", id)) %>%
142   points_midpoint()
143 railway_stations_df = point_in_polygons(points_df = railway_stations_df,
144                                         polys_sf   = berlin_district_sf,
145                                         var_name   = "district")
146 railway_stations_df = railway_stations_df %>%
147   filter(!is.na(district))
148 listings = distance_count(main = listings, reference = railway_stations_df,
149                           var_name = "station", distance = 1000)
150
151 # (Top 10) attractions
152 attractions_df = data.frame(
153   id    = c("Reichstag", "Brandenburger Tor", "Fernsehturm",
154           "Gendarmenmarkt", "Berliner Dom", "Kurfürstendamm",
155           "Schloss Charlottenburg", "Museuminsel",
156           "Gedenkstätte Berliner Mauer", "Potsdamer Platz"),
157   lat   = c(52.518611, 52.516389, 52.520803, 52.513333, 52.519167,
158           52.500833, 52.521111, 52.520556, 52.535, 52.509444),
159   long  = c(13.376111, 13.377778, 13.40945, 13.393056, 13.401111,
160           13.312778, 13.295833, 13.397222, 13.389722, 13.375833))
161 attractions_df = point_in_polygons(points_df = attractions_df,
162                                     polys_sf   = berlin_district_sf,
163                                     var_name   = "district")
164 listings = distance_count(main = listings, reference = attractions_df,
165                           var_name = "attraction", distance = 2000)
166
167 # season availability

```

```

168 listings_calendar = listings_calendar %>%
169   dplyr::rename(id = listing_id) %>%
170   dplyr::mutate(date = as.Date(date),
171                 year = lubridate::year(date),
172                 month = lubridate::month(date, label = TRUE),
173                 day = lubridate::day(date),
174                 season = ifelse((month == "Mar" & day >= 21) |
175                               (month == "Apr") |
176                               (month == "May") |
177                               (month == "Jun" & day < 21), "Spring",
178                               ifelse((month == "Jun" & day >= 21) |
179                                     (month == "Jul") |
180                                     (month == "Aug") |
181                                     (month == "Sep" & day < 21), "Summer",
182                               ifelse((month == "Sep" & day >= 21) |
183                                     (month == "Oct") |
184                                     (month == "Nov") |
185                                     (month == "Dec" & day < 21), "Fall",
186                               "Winter")))) %>%
187   factor(levels = c("Spring", "Summer",
188                     "Fall", "Winter")),
189   year_season = paste(tolower(season), year, sep = "_")) %>%
190 dplyr::ungroup() %>%
191 dplyr::select(-price)

192
193 # Availability per year_season
194 listings_season_availability = listings_calendar %>%
195   dplyr::group_by(id, year_season) %>%
196   dplyr::summarize(count = sum(available == TRUE),
197                     season_days = n(),
198                     proportion = round(count/season_days*100, 4)) %>%
199   dplyr::arrange(id, year_season) %>%
200   dplyr::ungroup() %>%
201   dplyr::mutate(season_av = paste(year_season, "availability",
202                                     sep = "_")) %>%
203   tolower() %>%
204   factor(levels = paste(year_season,
205                         "availability",
206                         sep = "_")) %>%
207   tolower() %>%
208   unique()))

```

```

209 dplyr::select(-count, -year_season, -season_days) %>%
210   tidyr::spread(season_av, proportion)
211
212 # Print code development
213 print("New variables created.")
214
215 # Join to the listings
216 listings = listings %>%
217   left_join(listings_season_availability, by = "id") %>%
218   replace(is.na(.), 0)
219
220 # Remove unnecessary objects
221 rm("listings_calendar", "listings_season_availability", "stations",
222     "berlin_neighbourhood_sf", "listings_detailed", "listings_summarized")
223 rm(list=lsf.str()) # All functions

```

Listing 22: |exploratory_data_analysis.R|

```

1 rm(list=ls(all = TRUE))
2
3 # Install and load needed packages
4 needed_packages = c("tidyverse",
5                      "rstudioapi",
6                      "Jmisc",
7                      "reshape2",
8                      "leaflet",
9                      "mapview",
10                     "xtable")
11 for (package in needed_packages) {
12   if (!require(package, character.only=TRUE)) {install.packages(package,
13   character.only=TRUE)}
14   library(package, character.only=TRUE)
15 }
16 rm("needed_packages", "package")
17
18 # Set working directory to the one where the file is located
19
20 # This works when run directly
21 # setwd(dirname(rstudioapi:::getActiveDocumentContext()$path))
22
23 # This works when sourced
24 setwd(dirname(sys.frame(1)$ofile))

```

```

24
25 # Load helper functions and other scripts
26 source("data_preparation.R", chdir = TRUE)
27 Jmisc::sourceAll(file.path(getwd(), "Helpers", fsep="/"))
28
29 ## SUMMARY STATISTICS
30
31 # Numeric variables
32 statistics_num = listings %>%
33   dplyr::select(-id, -listing_url, -long, -lat) %>%
34   dplyr::select_if(is.numeric) %>%
35   descriptive_statistics()
36
37 # Categorical variables
38 statistics_cat = listings %>%
39   dplyr::select(-id, -listing_url, -long, -lat) %>%
40   dplyr::select_if(function(x) is.character(x) | is.factor(x)
41                   | is.logical(x)) %>%
42   descriptive_statistics()
43
44 # Latex tables
45 # statistics_num %>%
46 #   filter(variable %in% c("price", "review_scores_rating")) %>%
47 #   xtable::xtable() %>%
48 #   print(include.rownames = FALSE)
49 # statistics_cat %>%
50 #   filter(variable == "room_type") %>%
51 #   xtable::xtable() %>%
52 #   print(include.rownames = FALSE)
53
54 ## SUMMARY DATAFRAMES
55
56 # Select numerical and categorical variables
57 num_var = listings %>%
58   dplyr::select(-id, -listing_url, -long, -lat,
59                 -vbb_zone, -district, -neighbourhood) %>%
60   dplyr::select_if(is.numeric) %>%
61   names()
62
63 fact_var = listings %>%
64   dplyr::select(-id, -listing_url, -long, -lat,

```

```

65     -vbb_zone, -district, -neighbourhood) %>%
66
67   dplyr::select_if(function(x) is.character(x) | is.factor(x)
68
69   | is.logical(x)) %>
70
71 names()
72
73
74 # For all Berlin
75
76 listings_summary = summarize_df(df = listings,
77
78                               vars_mean = num_var,
79
80                               vars_count = fact_var)
81
82
83 # For each VBB Zone
84
85 listings_vbb_summary = summarize_df(df = listings,
86
87                               wrt = "vbb_zone",
88
89                               vars_mean = num_var,
90
91                               vars_count = fact_var)
92
93
94 listings_vbb_summary = listings_vbb_summary %>%
95
96   rename(id      = vbb_zone) %>%
97
98   mutate(view    = "VBB Zones",
99
100  group     = id)
101
102
103 # For each district
104
105 listings_district_summary = summarize_df(df = listings,
106
107                               wrt = "district",
108
109                               vars_mean = num_var,
110
111                               vars_count = fact_var)
112
113
114 listings_district_summary = listings_district_summary %>%
115
116   rename(id      = district) %>%
117
118   mutate(view    = "Districts",
119
120  group     = id)
121
122
123 # For each neighbourhood
124
125 listings_neighbourhood_summary = summarize_df(df = listings,
126
127                               wrt = "neighbourhood",
128
129                               vars_mean = num_var,
130
131                               vars_count = fact_var)
132
133
134 listings_neighbourhood_summary = berlin_neighbourhoods_names %>%
135
136   dplyr::select(id, group) %>%
137
138   dplyr::left_join(listings_neighbourhood_summary,
139
140                     by = c("id" = "neighbourhood")) %>%
141
142   dplyr::mutate(view = "Neighbourhoods") %>%
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1489
1490
1491
1492
1493
1494
1495
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1587
1588
1589
1590
1591
1592
1593
1594
1595
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1686
1687
1688
1689
1690
1691
1692
1693
1694
1694
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1786
1787
1788
1789
1790
1791
1792
1793
1794
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1887
1888
1889
1890
1891
1892
1893
1894
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2
```

```

106     replace(is.na(.), 0)

107

108 ## PLOTS

109

110 # Numeric variables

111

112 # Complete distribution

113 distribution_plot(df = listings, var_name = "price")

114

115 # dev.copy2pdf(file = "./SeminarPaper/price_distribution_complete.pdf")
116 # dev.off()

117

118 # Without outliers

119 distribution_plot(df = listings, var_name = "price", hide_outliers = TRUE)

120

121 # dev.copy2pdf(file = "./SeminarPaper/price_distribution_nooutliers.pdf")
122 # dev.off()

123

124 # Plot factor variables

125 distribution_plot(df = listings, var_name = "station_count")

126

127 # dev.copy2pdf(file = "./SeminarPaper/room_type_distribution.pdf")
128 # dev.off()

129

130 # Maps of variable distribution

131 price_map_distr = var_avg_map(summary_df           = listings_district_summary,
132                                 var_name          = "price",
133                                 polygon_sf        = berlin_district_sf,
134                                 polygon_names_df = berlin_districts_names,
135                                 polygon_names_var= "name")

136 price_map_distr

137

138 # mapview::mapshot(price_map_distr, file = "./SeminarPaper/price_map_distr.
139 #                         pdf",
140 #                         remove_controls = c("zoomControl", "layersControl", "
141 #                                   homeButton", "scaleBar"))

142

143 print("Exploratory Data Analysis done.")

144 rm("price_map_distr", "fact_var", "num_var")

```

```
145 rm(list=lsf.str()) # All functions
```

Listing 23: |price_analysis.R|

```
1 rm(list=ls(all = TRUE))

2

3 # Install and load needed packages
4 needed_packages = c("tidyverse",
5                      "rstudioapi",
6                      "corrplot",
7                      "Jmisc",
8                      "xtable",
9                      "lindia")
10 for (package in needed_packages) {
11   if (!require(package, character.only=TRUE)) {
12     install.packages(package, character.only=TRUE)}
13   library(package, character.only=TRUE)
14 }
15 rm("needed_packages", "package")

16

17 # Set working directory to the one where the file is located
18

19 # This works when run directly
20 # setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

21

22 # This works when sourced
23 setwd(dirname(sys.frame(1)$ofile))

24

25 # Load helper functions and other scripts
26 source("exploratory_data_analysis.R", chdir = TRUE)
27 Jmisc::sourceAll(file.path(getwd(), "Helpers", fsep="/"))

28

29 ## Correlation between different variables and price
30

31 # Transform categorical variables from rows to column
32 listings_price = listings %>%
33   recode_all("room_type") %>% from_row_to_col("room_type") %>%
34   recode_all("property_type") %>% from_row_to_col("property_type") %>%
35   recode_all("cancellation_policy") %>%
36   from_row_to_col("cancellation_policy") %>%
37   recode_all("security_deposit") %>% from_row_to_col("security_deposit") %>%
38   recode_all("cleaning_fee") %>% from_row_to_col("cleaning_fee") %>%
```

```

39   recode_all("station_count") %>% from_row_to_col("station_count") %>%
40   recode_all("attraction_count") %>% from_row_to_col("attraction_count") %>%
41   recode_all("instant_bookable") %>% from_row_to_col("instant_bookable") %>%
42   recode_all("host_is_superhost") %>%
43   from_row_to_col("host_is_superhost") %>%
44   recode_all("host_has_profile_pic") %>%
45   from_row_to_col("host_has_profile_pic") %>%
46   recode_all("host_identity_verified") %>%
47   from_row_to_col("host_identity_verified") %>%
48   recode_all("district") %>% from_row_to_col("district") %>%
49   recode_all("vbb_zone") %>% from_row_to_col("vbb_zone") %>%
50   dplyr::select(-id, -lat, -long, -listing_url, -neighbourhood)

51
52 # Create dataframe with price correlations
53 listings_price_correlation = cor(listings_price$price, listings_price) %>%
54   as.data.frame() %>%
55   tidyr::gather() %>%
56   dplyr::rename(variable      = key,
57                 correlation = value) %>%
58   dplyr::filter(variable != "price")

59
60 print("Price correlation calculated.")

61
62 # Plot correlation
63 # Example corrplot
64 # listings %>%
65 #   dplyr::select(price, accommodates, bedrooms, beds,
66 #                  minimum_nights, station_dist) %>%
67 #   cor() %>%
68 #   corrplot(method = "circle", type = "upper")
69 # dev.copy2pdf(file = "./SeminarPaper/corrplot.pdf")
70 # dev.off()

71
72 # Correlation with price plot
73 correlation_plot(corr_df      = listings_price_correlation)

74
75 # dev.copy2pdf(file = "./SeminarPaper/price_correlation.pdf")
76 # dev.off()

77
78 ## Linear regression
79

```

```

80 # Fit linear model
81 listings_price_lm = lm(price ~ ., data = listings_price)
82
83 # Extract coefficients and p-values
84 listings_price_lm_results = data.frame(
85   variable      = names(listings_price_lm$coefficients
86                           [!is.na(listings_price_lm$coefficients)]),
87   coefficient   = listings_price_lm$coefficients
88                           [!is.na(listings_price_lm$coefficients)],
89   significant   = ifelse(summary(listings_price_lm)
90                           $coefficients[,4] < 0.5, TRUE, FALSE),
91   model_r_squared = summary(listings_price_lm)$r.squared) %>%
92   dplyr::mutate_if(is.numeric, function(x) round(x, 4))
93
94 # listings_price_lm_results %>%
95 #   dplyr::filter(variable %in% c("(Intercept)", "accommodates", "host_
96 #   listings_count")) %>%
97 #   dplyr::select(-model_r_squared) %>%
98 #   xtable::xtable() %>%
99 #   print(include.rownames = FALSE)
100
101 # Descriptive statistics of residuals
102 listings_price_lm_residuals = data.frame(
103   residuals = listings_price_lm$residuals
104 )
105
106 # descriptive_statistics(listings_price_lm_residuals) %>%
107 #   dplyr::select(-variable) %>%
108 #   xtable::xtable() %>%
109 #   print(include.rownames = FALSE)
110
111 # Fitted vs actual plot
112 ggplot(listings_price_lm, aes(x = .fitted, y = .resid)) +
113   geom_point() +
114   geom_smooth(method = lm) +
115   geom_hline(yintercept=0, col="red", linetype="dashed") +
116   xlab("Fitted values") +
117   ylab("Residuals") +
118   ggtitle("Residual vs Fitted Plot") +
119   theme_bw()
120 # dev.copy2pdf(file = "./SeminarPaper/fittedactual.pdf")

```

```

120 # dev.off()
121
122 # Residuals QQ Plot
123 lindia::gg_qqplot(listings_price_lm) +
124   theme_bw()
125 # dev.copy2pdf(file = "./SeminarPaper/qqplot.pdf")
126 # dev.off()
127
128 print("Linear regression on price calculated.")
129
130 # Remove objects not further needed
131 rm("listings_price_lm", "listings_price_lm_results")
132 rm(list=lsf.str()) # All functions

```

Listing 24: |clustering.R|

```

1 rm(list=ls(all = TRUE))
2
3 # Install and load needed packages
4 needed_packages = c("tidyverse",
5                      "rstudioapi",
6                      "Jmisc",
7                      "ggfortify")
8 for (package in needed_packages) {
9   if (!require(package, character.only=TRUE)) {
10     install.packages(package, character.only=TRUE)}
11   library(package, character.only=TRUE)
12 }
13 rm("needed_packages", "package")
14
15 # Set working directory to the one where the file is located
16
17 # This works when run directly
18 # setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
19
20 # This works when sourced
21 setwd(dirname(sys.frame(1)$ofile))
22
23 # Load helper functions and other scripts
24 source("price_analysis.R", chdir = TRUE)
25 Jmisc::sourceAll(file.path(getwd(), "Helpers", fsep="/"))
26

```

```

27 ## Calculate number of clusters
28
29 # Scale dataframe
30 listings_scaled = listings %>%
31   recode_all("room_type") %>% from_row_to_col("room_type") %>%
32   recode_all("property_type") %>% from_row_to_col("property_type") %>%
33   recode_all("cancellation_policy") %>%
34   from_row_to_col("cancellation_policy") %>%
35   recode_all("security_deposit") %>% from_row_to_col("security_deposit") %>%
36   recode_all("cleaning_fee") %>% from_row_to_col("cleaning_fee") %>%
37   recode_all("station_count") %>% from_row_to_col("station_count") %>%
38   recode_all("attraction_count") %>% from_row_to_col("attraction_count") %>%
39   recode_all("instant_bookable") %>% from_row_to_col("instant_bookable") %>%
40   recode_all("host_is_superhost") %>%
41   from_row_to_col("host_is_superhost") %>%
42   recode_all("host_has_profile_pic") %>%
43   from_row_to_col("host_has_profile_pic") %>%
44   recode_all("host_identity_verified") %>%
45   from_row_to_col("host_identity_verified") %>%
46   dplyr::select(-id, -lat, -long, -listing_url,
47                 -district, -vbb_zone, -neighbourhood) %>%
48   scale()
49
50 # Total variance explained
51 number_of_clusters(scaled_df      = listings_scaled,
52                     max          = 96,
53                     iter_max     = 20,
54                     plot_breaks = seq(0, 100, by = 5))
55
56 # dev.copy2pdf(file = "./SeminarPaper/numclusters.pdf")
57 # dev.off()
58
59 ## Clustering
60 n_clusters = 50
61 listings_kmeans = kmeans(listings_scaled,
62                         centers = n_clusters, iter.max = 20)
63
64 listings_clusters = listings_kmeans$cluster %>% as.factor()
65
66 # Plot clusters in map
67 ggplot() +

```

```

68 geom_sf(data = berlin_district_sf,
69           show.legend = FALSE, color = "black") +
70 coord_sf(datum = NA) +
71 geom_point(data = listings,
72             aes(x = long, y = lat, color = listings_clusters),
73             alpha = 0.5) +
74 theme(plot.title = element_text(hjust = 0.5)) +
75 ggtitle("Clustering of airbnb properties") +
76 labs(color = "clusters") +
77 theme_void()
78
79 # dev.copy2pdf(file = "./SeminarPaper/cluster_map.pdf")
80 # dev.off()
81
82 # Principal components
83 listings_pc = prcomp(x = listings_scaled, center = FALSE, scale = FALSE)
84
85 ggplot2::autoplot(listings_pc, data = listings_kmeans, colour = "cluster") +
86   ggtitle("Plotting clusters wrt Principal Components") +
87   labs(color = "clusters") +
88   theme_bw()
89
90 # dev.copy2pdf(file = "./SeminarPaper/cluster_plot.pdf")
91 # dev.off()
92
93 # Remove objects not further needed
94 rm("n_clusters")
95 rm(list=lsf.str()) # All functions

```

Listing 25: |shiny_preparation.R|

```

1 rm(list=ls(all = TRUE))
2
3 # Install and load needed packages
4 needed_packages = c("tidyverse",
5                      "rstudioapi",
6                      "sf")
7 for (package in needed_packages) {
8   if (!require(package, character.only=TRUE)) {
9     install.packages(package, character.only=TRUE)}
10  library(package, character.only=TRUE)
11 }

```

```

12 rm("needed_packages", "package")

13

14 # Set working directory to the one where the file is located

15

16 # This works when run directly
17 setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

18

19 # This works when sourced
20 # setwd(dirname(sys.frame(1)$ofile))

21

22 # Load scripts
23 source("clustering.R", chdir = TRUE)

24

25 # Calculate the center of Berlin polygon
26 berlin_center = berlin_neighbourhood_singlebuckow_sf %>%
27   dplyr::group_by() %>%
28   dplyr::summarize(do_union = TRUE) %>%
29   sf::st_centroid() %>%
30   sf::st_coordinates() %>%
31   base::as.data.frame() %>%
32   dplyr::rename(long = X,
33                 lat = Y) %>%
34   dplyr::select(lat, long)

35

36 # Join area summaries
37 listings_area_summary = rbind(listings_district_summary, listings_vbb_
38                               summary, listings_neighbourhood_summary)

39

40 # Join sf objects
41 berlin_district_sf = berlin_district_sf %>%
42   dplyr::mutate(view = "Districts")
43 berlin_neighbourhood_singlebuckow_sf =
44   berlin_neighbourhood_singlebuckow_sf %>%
45   dplyr::mutate(view = "Neighbourhoods")
46 berlin_vbb_AB_sf = berlin_vbb_AB_sf %>%
47   dplyr::mutate(view = "VBB Zones",
48                 group = id)
49 berlin_sf = rbind(berlin_district_sf,
50                   berlin_neighbourhood_singlebuckow_sf,
51                   berlin_vbb_AB_sf)

```

```

52 # Join names dataframes
53 berlin_districts_names = berlin_districts_names %>%
54   dplyr::mutate(view = "Districts")
55 berlin_neighbourhoods_names = berlin_neighbourhoods_names %>%
56   dplyr::mutate(view = "Neighbourhoods")
57 berlin_vbb_AB_names = berlin_vbb_AB_names %>%
58   dplyr::mutate(view = "VBB Zones",
59                 group = id)
60 berlin_names = rbind(berlin_districts_names,
61                      berlin_neighbourhoods_names,
62                      berlin_vbb_AB_names)
63
64 # Recode and transform from row to columns factor variables
65 listings_recoded = listings %>%
66   recode_all("room_type") %>% from_row_to_col("room_type") %>%
67   recode_all("property_type") %>% from_row_to_col("property_type") %>%
68   recode_all("cancellation_policy") %>%
69   from_row_to_col("cancellation_policy")
70
71 # Remove objects not further needed
72 rm("listings_district_summary", "listings_vbb_summary",
73    "listings_neighbourhood_summary", "berlin_district_sf",
74    "berlin_neighbourhood_singlebuckow_sf", "berlin_vbb_AB_sf",
75    "berlin_districts_names", "berlin_neighbourhoods_names",
76    "berlin_vbb_AB_names")
77 rm(list=lsf.str()) # All functions
78
79 # Save workspace
80 save.image(file = file.path(getwd(), "ShinyApp", "workspace_for_shinyapp.RData", fsep="/"))

```

A.2 Functions

Listing 26: |capwords.R|

```

1 # Capitalize the first letter of every word
2 # from the help for toupper()
3 capwords <- function(s, strict = FALSE) {
4   cap <- function(s) paste(toupper(substring(s, 1, 1)),
5                           {s <- substring(s, 2);
6                            if(strict) tolower(s) else s},

```

```

7           sep = "", collapse = " " )
8 sapply(strsplit(s, split = " "), cap, USE.NAMES = !is.null(names(s)))
9 }

```

Listing 27: |correlation_plot.R|

```

1 correlation_plot = function(corr_df, variables = corr_df$variable,
2                             variables_original = TRUE) {
3
4   correlation = corr_df %>%
5     dplyr::mutate(sign      = ifelse(correlation > 0, "positive",
6                   ifelse(correlation < 0, "negative",
7                         "null")),
8                   correlation = abs(correlation)) %>%
9     dplyr::arrange(variable)
10
11  if (all(correlation$variable == corr_df$variable)) {
12
13    # If variables are listed with their names from the original dataframe
14    # (before dummies)
15    if (variables_original) {
16
17      correlation = correlation %>%
18        dplyr::filter(grepl(paste(variables, collapse = " | "),
19                      corr_df$variable))
20
21    # If variables are listed with their names from corr_df
22  } else {
23
24    correlation = correlation %>%
25      dplyr::filter(variable %in% variables)
26
27  }
28
29
30  corr_plot = correlation %>%
31    ggplot(aes(x = variable, y = correlation, fill = sign)) +
32    geom_bar(stat="identity") +
33    scale_fill_manual("sign",
34                      values = c("positive" = "green4",
35                                "negative" = "red1",

```

```
36                                     "null" = "blue3")) +
37
38     coord_flip() +
39
40     labs(x = "variables", y = "correlation",
41           title = "Correlation with price") +
42
43     theme_bw() +
44
45     theme(axis.text.y      = element_text(
46
47             color = ifelse(
48
49                 correlation$sign == "positive",
50
51                 "green4",
52
53                 ifelse(
54
55                     correlation$sign == "negative",
56
57                     "red1",
58
59                     "blue3"))),
60
61
62     axis.text.x      = element_text(size = rel(1.2)),
63
64     axis.title.x    = element_text(size = rel(1.2)),
65
66     axis.title.y    = element_text(size = rel(1.2)),
67
68     legend.text     = element_text(size = rel(1.2)),
69
70     legend.position = "bottom") +
71
72     scale_y_continuous(expand = expand_scale(mult = c(0, 0.05)))
73
74
75
76     return(corr_plot)
77
78 }
```

Listing 28: |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))

18

19 # Descriptive statistics for categorical variables
20 } else if (unique(apply(df, 2, function(x) is.character(x) | is.factor(x)
| 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,
48                 proportion = as.character(proportion) %>% paste("%"))

49

50 summary = frequency %>%
51   dplyr::inner_join(proportion, by = c("variable", "factor")) %>%
52   dplyr::arrange(variable, desc(frequency))

53

54 } else {
55
56   stop("Check that all variables are either numeric or categorical.")

```

```

55
56     }
57
58     return(summary)
59
60 }
```

Listing 29: |df_join_clean.R|

```

1 # Join dataframes and cleaning
2 df_join_clean = function(df1, df2) {
3
4   df1 = df1 %>%
5     # price and fee columns
6     dplyr::mutate_if(grepl("price|cleaning|deposit|people" ,
7                           names(.)), funs(gsub("\$\$", "", .))) %>%
8     dplyr::mutate_if(grepl("price|cleaning|deposit|people" ,
9                           names(.)), funs(gsub("\\\\,", "", .))) %>%
10    dplyr::mutate_if(grepl("price|cleaning|deposit|people" ,
11                        names(.)), funs(as.numeric(.)))
12
13   df2 = df2 %>%
14     # price and fee columns
15     dplyr::mutate_if(grepl("price|cleaning|deposit|people" ,
16                           names(.)), funs(gsub("\$\$", "", .))) %>%
17     dplyr::mutate_if(grepl("price|cleaning|deposit|people" ,
18                           names(.)), funs(gsub("\\\\,", "", .))) %>%
19     dplyr::mutate_if(grepl("price|cleaning|deposit|people" ,
20                           names(.)), funs(as.numeric(.)))
21
22   df_joined = full_join(df1, df2, by = c("id", "host_id", "room_type", "price",
23                         "minimum_nights",
24                         "number_of_reviews",
25                         "latitude", "longitude",
26                         "availability_365", "calculated_host_listings_count")) %>%
27
28   #First selection
29   dplyr::select(id, longitude, latitude, price, property_type, room_type,
30                 instant_bookable, listing_url, host_is_superhost,
31                 security_deposit, cleaning_fee,
32                 calculated_host_listings_count, host_has_profile_pic, host
```

```

32     _identity_verified,
33     accommodates, bedrooms, beds, minimum_nights,
34     review_scores_rating, number_of_reviews, cancellation_
35         policy,
36     availability_30, availability_60, availability_90,
37         availability_365) %>%
38 
39     dplyr::rename(lat = latitude,
40                 long = longitude,
41                 host_listings_count = calculated_host_listings_count) %>%
42 
43 # property_type
44 
45     dplyr::group_by(property_type) %>%
46     dplyr::mutate(property_type_count = n()) %>%
47     dplyr::ungroup() %>%
48     dplyr::mutate(property_type = as.factor(ifelse(property_type_count
49         <= 10,
50             "Other", property_type)),
51 
52 # security_deposit
53 
54         security_deposit = ifelse(security_deposit == 0 |
55             is.na(security_deposit), FALSE,
56             TRUE),
57 
58 # cleaning_fee
59 
60         cleaning_fee = ifelse(cleaning_fee == 0 |
61             is.na(cleaning_fee), FALSE,
62             TRUE),
63 
64 # availabilities
65 
66         availability_30 = round(availability_30/30*100, 4),
67         availability_60 = round(availability_60/60*100, 4),
68         availability_90 = round(availability_90/90*100, 4),
69         availability_365 = round(availability_365/365*100, 4))
70 
71     %>%
72 
73     dplyr::select(-property_type_count)
74 
75 
76     return(df_joined)
77 }

```

Listing 30: |distance_count.R|

```

1 # Calculate the distance from the nearest reference point (from reference df
2 
3 # and the number of reference points within a certain distance
4 distance_count = function(main, reference, var_name, distance) {
5     # Create variable names

```

```

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

```

Listing 31: |distribution_plot.R|

```

1 distribution_plot = function(df, var_name, hide_outliers = FALSE, tilt_text_
2   x = FALSE) {
3
4   variable = sym(var_name)
5
6   # Plot for numeric variables
7   if (is.numeric(df[, var_name])) {
8
9     numvar_plot = df %>%
      dplyr::transmute(

```

```

10      var          = !variable,
11      median       = median(var),
12      mean         = mean(var),
13      '1Q'         = quantile(var, probs = 0.25),
14      '3Q'         = quantile(var, probs = 0.75),
15      IQR          = IQR(var),
16      upper        = (IQR*1.5) + '3Q',
17      lower        = '1Q' - (IQR*1.5),
18      outlier_upper = ifelse(var > unique(upper), 1, 0),
19      outlier_lower = ifelse(var < unique(lower), 1, 0),
20      outlier      = ifelse(outlier_upper == 1 | outlier_lower
21                      == 1, 1, 0),
22      count_upper   = sum(outlier_upper),
23      count_lower   = sum(outlier_lower)
24
25
26  if (hide_outliers) {
27
28
29    plot = numvar_plot %>%
30      dplyr::filter(outlier == 0) %>%
31      ggplot() +
32      geom_density(aes(x = var), fill = "green4", color = "green4") +
33      geom_vline(aes(xintercept = unique(median)),
34                 color = "blue", linetype = "dashed") +
35      annotate(geom = "text", x = (unique(numvar_plot$median) + 2.6), y =
36                0,
37                 label = "Median",
38                 colour = "blue",
39                 angle = 90, hjust = 0) +
39      geom_vline(aes(xintercept = unique(mean)),
40                 color = "black", linetype = "dashed") +
41      annotate(geom = "text", x = (unique(numvar_plot$mean) + 2.6), y = 0,
42                label = "Mean",
43                colour = "black",
44                angle = 90, hjust = 0) +
44      geom_vline(aes(xintercept = '1Q'),
45                 color = "red", linetype = "dashed") +
46      annotate(geom = "text", x = (unique(numvar_plot$'1Q') + 2.6), y = 0,
47                label = "1st Quantile",
48                colour = "red",
49                angle = 90, hjust = 0) +

```

```

49   geom_vline(aes(xintercept = '3Q'),
50             color = "red", linetype = "dashed") +
51   annotate(geom = "text", x = (unique(numvar_plot$'3Q') + 2.6), y = 0,
52             label = "3rd Quantile",
53             colour = "red",
54             angle = 90, hjust = 0) +
55   labs(caption = paste("Outliers not displayed:",
56                         numvar_plot$count_lower, "values under 1,5*IQR
57                         and",
58                         numvar_plot$count_upper, "values over 1,5*IQR",
59                         sep = " "),
60             title = paste("Density of the variable", variable, sep = " "),
61             x = var_name) +
62   theme_bw() +
63   theme(axis.text.x = element_text(size = rel(1.2)),
64         axis.text.y = element_text(size = rel(1.2)),
65         axis.title.x = element_text(size = rel(1.2)),
66         axis.title.y = element_text(size = rel(1.2)))
67 } else {
68
69   plot = numvar_plot %>%
70     ggplot() +
71     geom_density(aes(x = var), fill = "green4", color = "green4") +
72     geom_vline(aes(xintercept = unique(median)),
73                color = "blue", linetype = "dashed") +
74     geom_vline(aes(xintercept = unique(mean)),
75                color = "black", linetype = "dashed") +
76     geom_vline(aes(xintercept = '1Q'),
77                color = "red", linetype = "dashed") +
78     geom_vline(aes(xintercept = '3Q'),
79                color = "red", linetype = "dashed") +
80     labs(x = var_name,
81           title = paste("Density of the variable", variable, sep = " "))
82     +
83     theme_bw() +
84     theme(axis.text.x = element_text(size = rel(1.2)),
85           axis.text.y = element_text(size = rel(1.2)),
86           axis.title.x = element_text(size = rel(1.2)),
87           axis.title.y = element_text(size = rel(1.2)))
88 }
```

```

88
89   if (tilt_text_x) {
90
91     plot = plot +
92       theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5)
93             )
94   }
95
96   # Plot for categorical variables
97 } else if (is.character(df[, var_name]) |
98           is.factor(df[, var_name]) |
99           is.logical(df[, var_name])) {
100
101   factvar_plot = listings %>%
102     dplyr::rename(var = !!variable) %>%
103     dplyr::group_by(var) %>%
104     dplyr::summarize(count = n())
105
106   plot = ggplot(factvar_plot) +
107     geom_bar(aes(x = var, y = count), fill = "green4", stat = "identity")
108     +
109     labs(x      = variable,
110          title = paste("Histogram of the variable", variable, sep = " "))
111     +
112     theme_bw() +
113       theme(axis.text.x = element_text(size = rel(1.2)),
114             axis.text.y = element_text(size = rel(1.2)),
115             axis.title.x = element_text(size = rel(1.2)),
116             axis.title.y = element_text(size = rel(1.2)))
117
118   if (tilt_text_x) {
119
120     plot = plot +
121       theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5)
122             )
123   }
124
125   if (hide_outliers) {

```

```

125
126     stop("hide_outliers cannot be TRUE for categorical variables.")
127
128 }
129
130 } else {
131
132     stop("Check what type of variable you area dealing with.")
133
134 }
135
136 plot
137
138 }
```

Listing 32: |from_row_to_col.R|

```

1 # Spread rows into columns using just one variable
2 from_row_to_col = function(df, variable) {
3     df = df %>%
4         dplyr::mutate(yesno = 1) %>%
5         dplyr::distinct() %>%
6         tidyr::spread(variable, yesno, fill = 0)
7     return(df)
8 }
```

Listing 33: |getmode.R|

```

1 # getmode function
2 # from: https://www.tutorialspoint.com/r/r\_mean\_median\_mode.htm
3 getmode <- function(v) {
4     uniqv <- unique(v)
5     uniqv[which.max(tabulate(match(v, uniqv)))]
6 }
```

Listing 34: |number_of_clusters.R|

```

1 # Calculate plot of tve to choose number of clusters for k-means clustering
2 number_of_clusters = function(scaled_df, min = 2, max,
3                               iter_max = 10, plot_breaks) {
4     # Set seed for reproducibility
5     set.seed(900114)
6     # Values to test
```

```

7 k_values = min:max
8 # Empty dataframe for the total variance explained
9 tve = data.frame(clusters = k_values,
10                  tve      = rep(NA, length(k_values)))
11 # Empty list for the kmeans objects
12 clk = list()
13 # Loop through the possible values of k
14 for (k in k_values) {
15   # Calculate k-means for each k
16   clk[[k-1]] = kmeans(scaled_df, centers = k, iter.max = iter_max)
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 }
24
25 # Plot tve against k values
26 plot = ggplot(data = tve, aes(x = clusters, y = tve)) +
27   geom_line(color = "grey") +
28   geom_point(color = "red") +
29   scale_x_continuous(breaks = plot_breaks) +
30   labs(x      = "number of clusters",
31        y      = "% of tve",
32        title = paste("Plot of total variance explained for k from", min,
33                      "to", max, sep = " ")) +
34   theme_bw() +
35   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 }
```

Listing 35: |point_in_polygons.R|

```

1 # Tells in which polygon a point is
2 point_in_polygons <- function(points_df, polys_sf,
3                                var_name, join_var = "id") {
4   # Create empty dataframe
```

```

5  is_in <- data.frame(matrix(ncol = nrow(polys_sf), nrow = nrow(points_df)))
6  is_in[,var_name] <- NA
7  # Extract coordinates of the polygons
8  coordinates <- as.data.frame(st_coordinates(polys_sf))
9  # Extract names of the polygons
10 name <- as.character(polys_sf$id)
11 # For all polygons check if the points are inside of them
12 for (k in 1:nrow(polys_sf)) {
13   is_in[,k] <- sp::point.in.polygon(point.x = points_df$long,
14                                     point.y = points_df$lat,
15                                     pol.x = coordinates$X
16                                     [coordinates$L2 == k],
17                                     pol.y = coordinates$Y
18                                     [coordinates$L2 == k])
19   # Get the names of the polygons where the points are in one column
20   is_in[,var_name][is_in[,k] == 1] <- name[k]
21 }
22 # Keep only summary column and add points' names
23 is_in <- is_in %>%
24   dplyr::select(var_name) %>%
25   dplyr::mutate(id = points_df$id)
26 # Add the summary column to the points dataframe
27 points_df <- dplyr::full_join(points_df, is_in, by = join_var)
28 return(points_df)
29 }
```

Listing 36: |points_midpoint.R|

```

1 # Calculate midpoint from a sf point object
2 points_midpoint <- function(points_sf, point_name = "id") {
3   # Create symbol version of point_name
4   point_name <- sym(point_name)
5   # Get coordinates from sf point objects
6   coordinates <- sf::st_coordinates(points_sf) %>%
7     base::as.data.frame() %>%
8     dplyr::rename(lat = Y,
9                   long = X)
10  # Bind coordinates with sf point object and calculate mean of lat and long
11  midpoints_df <- points_sf %>%
12    base::as.data.frame() %>%
13    dplyr::select(-geometry) %>%
14    base::cbind(coordinates) %>%
```

```

15   dplyr::group_by(!point_name) %>%
16     dplyr::summarize(lat      = mean(lat),
17                       long     = mean(long),
18                       s_bahn  = any(s_bahn),
19                       u_bahn  = any(u_bahn))
20   return(midpoints_df)
21 }

```

Listing 37: |recode_all.R|

```

1 # Recode variable: rename all factor levels of that variable
2 recode_all = function(df, variable) {
3   variable_name = sym(variable)
4   df = df %>%
5     mutate(!variable_name := as.factor(!variable_name))
6   levels(df[,variable]) = gsub(" ", "_", levels(df[,variable])) %>%
7     tolower() %>%
8     paste(variable, ., sep = "_")
9   return(df)
10 }

```

Listing 38: |summarize_df.R|

```

1 # Summary information for numeric and factor variables
2 summarize_df <- function(df, wrt = NULL, vars_mean, vars_count) {
3   # Create empty vector
4   summary_df <- c()
5   # For the case in which the summary is general
6   if(is.null(wrt)) {
7     # Variables of which we calculate the mean
8     for (var in vars_mean) {
9       var_name <- sym(var)
10      mean_df <- df %>% dplyr::summarize(!var_name := mean(!var_name))
11      summary_df <- c(summary_df, mean_df)
12    }
13    # Variables of which we count the unique values of the factors
14    for (var in vars_count) {
15      var_name <- sym(var)
16      count_df <- df %>%
17        recode_all(var) %>%
18        group_by(!var_name) %>%
19        dplyr::summarize(count = n()) %>%

```

```

20     spread(key = !!var_name, value = count)
21 
22   }
23 
24 } else {
25 
26   wrt_name <- sym(wrt)
27 
28   # Variables of which we calculate the mean
29   for (var in vars_mean) {
30 
31     var_name <- sym(var)
32 
33     mean_df <- df %>%
34       group_by (!!wrt_name) %>%
35       dplyr::summarize (!!var_name := mean (!!var_name))
36 
37     summary_df <- c(summary_df, mean_df)
38 
39     summary_df[duplicated(summary_df)] <- NULL
40 
41   }
42 
43   # Variables of which we count the unique values of the factors
44   for (var in vars_count) {
45 
46     var_name <- sym(var)
47 
48     count_df <- df %>%
49       recode_all(var) %>%
50       group_by (!!var_name, !!wrt_name) %>%
51       dplyr::summarize(count = n()) %>%
52       spread(key = !!var_name, value = count)
53 
54     summary_df <- summary_df %>% as.data.frame() %>%
55       dplyr::mutate (!!wrt_name := as.character (!!wrt_name)) %>%
56       full_join(count_df, by = wrt) %>%
57       replace(is.na(.), 0)
58 
59   }
60 
61   # Percentage of the rows
62   percentage_df <- df %>%
63     group_by (!!wrt_name) %>%
64     dplyr::summarize(percentage = round(n() / nrow(.) * 100, 2))
65 
66   summary_df <- summary_df %>% full_join(percentage_df, by = wrt)
67 
68 }
69 
70 return(summary_df)
71 }
```

Listing 39: |summary_stat_fact.R|

```

1 summary_stat_fact = function(df, var, constraint_var = NULL, constraint_
2   value = NULL) {
```

```

3  if (is.null(constraint_var) & is.null(constraint_value)) {
4    factor = df[, var]
5  } else if (is.null(constraint_var) | is.null(constraint_value)) {
6    stop("constraint_var or constraint_value is not defined. Review function
7      values.")
8  } else {
9    constraint = sym(constraint_var)
10   df        = df %>%
11     dplyr::filter (!!constraint == constraint_value)
12   factor    = df[, var]
13 }
14
15 frequency_df = table(factor) %>%
16   as.data.frame() %>%
17   dplyr::rename(frequency = Freq)
18
19 proportion_df = prop.table(table(factor)) %>%
20   as.data.frame() %>%
21   dplyr::rename(proportion = Freq) %>%
22   dplyr::mutate(proportion = round(proportion, 4)*100,
23                 proportion = as.character(proportion) %>% paste("%"))
24
25 summary = frequency_df %>%
26   dplyr::inner_join(proportion_df, by = "factor") %>%
27   dplyr::arrange(desc(frequency))
28
29 return(summary)
30 }
```

Listing 40: |summary_stat_num.R|

```

1 summary_stat_num = function(df, var, constraint_var = NULL, constraint_value
2   = NULL) {
3
4  if (is.null(constraint_var) & is.null(constraint_value)) {
5    variable = df[, var]
6  } else if (is.null(constraint_var) | is.null(constraint_value)) {
7    stop("constraint_var or constraint_value is not defined. Review function
8      values.")
9  } else {
10    constraint = sym(constraint_var)
```

```

9   df      = df %>%
10
11   dplyr::filter (!!constraint == constraint_value)
12   variable = df[, var]
13 }
14
15 summary = data.frame(
16
17   min     = min(variable),
18   `1Q`    = quantile(variable, probs = 0.25),
19   median  = median(variable),
20   `3Q`    = quantile(variable, probs = 0.75),
21   max     = max(variable),
22   iqr     = IQR(variable),
23   mean    = mean(variable),
24   sd      = sd(variable),
25
26   check.names = FALSE) %>%
27
28   dplyr::mutate_if(is.numeric, function(x) round(x, 4))
29
30
31 return(summary)
32
33 }
```

Listing 41: |var_avg_map.R|

```

1 var_avg_map = function(summary_df, var_name,
2
3   polygon_sf, polygon_names_df, polygon_names_var,
4   bins = TRUE, nsplits = 4, pretty = FALSE, zoom_level
5   = 10,
6   title = paste("Average of", var_name, sep = " "))
7
8
9   if (bins) {
10
11   colors = colorBin(topo.colors(nsplits), summary_df[, var_name], bins =
12   nsplits, pretty = pretty)
13
14 } else {
15
16   if (pretty) {
17
18     stop("Parameter pretty not used in colorQuantile.")
19
20   }
21
22 }
```

```

18
19     colors = colorQuantile(topo.colors(nsplits), summary_df[, var_name],
20                             bins = nsplits)
21
22
23 map = leaflet() %>%
24   addTiles() %>%
25   addPolygons(data = polygon_sf, weight = 1, smoothFactor = 1, fillOpacity
26               = 0.8,
27               color = "grey",
28               fillColor = ~colors(summary_df[, var_name])) %>%
29   addLabelOnlyMarkers(data = polygon_names_df,
30                       lng = ~long, lat = ~lat, label = ~lapply(polygon_
31                                         names_df[, polygon_names_var], htmltools::HTML),
32                       labelOptions = labelOptions(noHide = TRUE, direction
33                                     = 'center',
34                                     textOnly = TRUE,
35                                     textSize = 20,
36                                     style = list("color" =
37                                     black))) %>%
38   addLegend(position = "bottomleft", pal = colors,
39             values = summary_df[, var_name],
40             labFormat = labelFormat(transform = function(x) sort(x,
41                                         decreasing = FALSE)),
42             title = title)
43
44
45 return(map)
46
47
48 }

```

A.3 ShinyApp

Listing 42: |ui.R|

```

1 ui = dashboardPage(
2   skin = "red",
3   dashboardHeader(title = "Airbnb in Berlin"),
4   dashboardSidebar(
5     sidebarMenu(
6       id = "tabs",

```

```

7   menuItem("Intro",
8     tabName = "intro",
9     icon = icon("play-circle")),
10    menuItem("Maps",
11      tabName = "maps",
12      icon = icon("map")),
13    menuItem("Price",
14      tabName = "price",
15      icon = icon("dollar")),
16    menuItem("Clustering",
17      tabName = "cluster",
18      icon = icon("object-group"))
19  )
20),
21 dashboardBody(
22   tabItems(
23
24     tabItem(tabName = "intro",
25       fluidRow(
26         box(
27           title = "Welcome to the ShinyApp on Airbnb properties in
28             Berlin!",
29           width = 12,
30           status = "danger",
31           uiOutput("intro_ui"),
32           br(),
33           column(width = 4,
34             h4("Maps Tab"),
35             textOutput("intro_maps_text"),
36             br(),
37             actionButton("switch_maps", "Go to Maps Tab")),
38           column(width = 4,
39             h4("Price Tab"),
40             textOutput("intro_price_text"),
41             br(),
42             actionButton("switch_price", "Go to Price Tab")),
43           column(width = 4,
44             h4("Clustering Tab"),
45             textOutput("intro_cluster_text"),
46             br(),
47             actionButton("switch_cluster",

```

```

47                               "Go to Clustering Tab")))
48
49 )
50
51 ### TAB2 (Maps)
52 tabItem(tabName = "maps",
53         fluidRow(
54             tabBox(id = "berlin",
55                   width = 12,
56                   tabPanel(
57                       title = "Berlin map",
58                       textOutput("berlin_text"),
59                       br(),
60                       box(
61                           width = 12,
62                           solidHeader=TRUE,
63                           column(
64                               width = 6,
65                               selectInput(
66                                   inputId = "view",
67                                   label = "Choose view",
68                                   choices = berlin_sf %>%
69                                       as.data.frame() %>%
70                                       select(view) %>%
71                                       unique() %>%
72                                       t() %>%
73                                       c() %>%
74                                       setdiff("Neighbourhoods"))),
75                           column(
76                               width = 6,
77                               selectInput(
78                                   inputId = "variable1",
79                                   label = "Choose variable",
80                                   choices = c("none", listings_area_summary %>%
81                                       colnames() %>%
82                                       setdiff(c("id", "view", "group")) %>%
83                                       gsub("_", " ", .)))),
84                           ),
85                           box(
86                               width = 12,

```

```

87     solidHeader=TRUE,
88
89     column(
90         width = 4,
91         checkboxInput(inputId = "main_properties",
92                         label = "Display properties"
93                         ,
94                         value = FALSE)),
95
96     column(
97         width = 4,
98         checkboxInput(inputId = "main_stations",
99                         label = "Display train/
100                         subway stations",
101                         value = FALSE)),
102
103     column(
104         width = 4,
105         checkboxInput(
106             inputId = "main_sights",
107             label = "Display top 10 attractions",
108             value = FALSE))
109
110     ),
111
112     box(
113         width = 12,
114         align = "center",
115         solidHeader = TRUE,
116         h4(textOutput("title_main_map")),
117         leafletOutput("main_map") %>% withSpinner()
118         ),
119
120     box(
121         width = 12,
122         align = "center",
123         solidHeader = TRUE,
124         column(
125             width = 4,
126             h4(textOutput("title_main_table")),
127             tableOutput("main_table1"),
128             tableOutput("main_table2")
129             ),
130
131         column(
132             width = 8,
133             h4(textOutput("title_main_plot")),
134             plotOutput("main_plot"))

```



```

159                               value    = FALSE)),  

160  

161             column(  

162               width  = 4,  

163               checkboxInput(inputId = "district_stations"  

164                 "",  

165                               label   = "Display train/  

166                               subway stations",  

167                               value   = FALSE)),  

168  

169             column(  

170               width  = 4,  

171               checkboxInput(inputId = "district_sights",  

172                               label   = "Display top 10  

173                               attractions",  

174                               value   = FALSE))  

175  

176           ),  

177  

178           box(width      = 12,  

179                 align      = "center",  

180                 solidHeader = TRUE,  

181                 h4(textOutput("title_secondary_map")),  

182                 leafletOutput("secondary_map") %>%  

183                 withSpinner()  

184           ),  

185  

186           box(width      = 12,  

187                 align      = "center",  

188                 solidHeader = TRUE,  

189                 column(  

190                   width  = 4,  

191                   h4(textOutput("title_secondary_table")),  

192                   tableOutput("secondary_table1"),  

193                   tableOutput("secondary_table2")  

194             ),  

195             column(  

196               width  = 8,  

197               h4(textOutput("title_secondary_plot")),  

198               plotOutput("secondary_plot")  

199             )  

200           )  

201         )

```

```

196     )
197   ),
198
199 ### TAB3 (Price)
200 tabItem(tabName = "price",
201   fluidRow(
202     box(
203       status = "danger",
204       width = 12,
205       column(
206         width = 6,
207         textOutput(outputId = "price_text")
208       ),
209       column(
210         width = 6,
211         pickerInput(inputId = "variable3", label = "Choose variable
(s)",
212           choices = listings_price_correlation %>%
213             select(variable) %>%
214             unique() %>%
215             t() %>%
216             c() %>%
217             gsub("_", " ", .),
218             multiple = TRUE,
219             selected = listings_price_correlation %>%
220               select(variable) %>%
221               unique() %>%
222               t() %>%
223               c() %>%
224               gsub("_", " ", .),
225             options = list('actions-box' = TRUE))
226       )
227
228     )
229   ),
230   fluidRow(
231     tabBox(id = "price_box",
232       width = 12,
233       tabPanel(
234         title = "Correlation with price",

```

```

236     plotOutput(outputId = "corr_plot")
237   ),
238   tabPanel(
239     title = "Linear regression on price",
240     tabBox(id = "lm_model",
241       width = 12,
242       tabPanel(
243         title = "Model",
244         box(
245           width = 12,
246           align = "center",
247           solidHeader = TRUE,
248           column(
249             width = 3,
250             h4("R-Squared of the model"),
251             textOutput(outputId = "price_lm_r2")
252           ),
253           column(
254             width = 9,
255             h4("Coefficients of the linear regression
256               on price."),
257             tableOutput(outputId = "price_lm_
258               coefficients")
259           )
260         )
261       ),
262       tabPanel(
263         title = "Residuals",
264         box(width = 12,
265             solidHeader = TRUE,
266             align = "center",
267             h4("Summary statistics of the residuals"),
268             tableOutput(outputId = "price_lm_residuals
269             ")
270           ),
271         box(width = 12,
272             solidHeader = TRUE,
273             align = "center",
274             h4("Residuals plots"),
275             column(
276               width = 6,

```

```

274             plotOutput(outputId = "price_lm_resplot"
275                         ) %>% withSpinner()
276             ),
277             column(
278                 width = 6,
279                 plotOutput(outputId = "price_lm_qqplot")
280                         %>% withSpinner()
281             )
282         )
283     )
284   )
285   )
286   ),
287 ### TAB4 (Clustering)
288 tabItem(tabName = "cluster",
289   fluidRow(
290     box(status = "danger",
291         width = 12,
292         uiOutput("cluster_ui")),
293     tabBox(id = "cluster_test",
294         width = 12,
295         tabPanel(title = "Find number of clusters",
296                   textOutput("ctest_text"),
297                   br(),
298                   sliderInput(inputId = "ctest",
299                               label = "Choose which number of
300                                     clusters to test",
301                               min = 2, max = 96, value = c(2, 40),
302                               step = 1,
303                               width = "100%"),
304                   actionButton("ctest_start", "Calculate"),
305                   progressBar(id = "ctest_progress",
306                               status = "danger",
307                               value = 0, total = 100,
308                               display_pct = TRUE),
309                   plotOutput(outputId = "tve")
310             ),
311
312         tabPanel(title = "Cluster Airbnb properties",

```

```

311         uiOutput("cnum_ui"),
312
313         br(),
314
315         numericInput(inputId = "cnum",
316                         label = "Choose number of clusters for
317                         clustering",
318                         min = 2, max = 96, value = 12, step =
319                         1,
320                         width = "30%"),
321
322         actionButton("cnum_start", "Calculate"),
323
324         box(
325             width = 12,
326             align = "center",
327             solidHeader = TRUE,
328             column(
329                 width = 6,
330                 h4("Clusters on coordinates"),
331                 plotOutput(outputId = "clustercoord") %>%
332                     withSpinner()
333             ),
334             column(
335                 width = 6,
336                 h4("Clusters on Principal Components"),
337                 plotOutput(outputId = "clustercpc") %>%
338                     withSpinner()
339             )
340         )
341     )

```

Listing 43: |server.R|

```

1 server = function(input, output, session) {
2
3   # Tab 1
4   introduction_server(input, output, session)

```

```

5
6      # Tab 2
7
8      # Panel 1
9      maps_berlin_server(input, output, session)
10
11     # Panel 2
12     maps_district_server(input, output, session)
13
14     # Tab 3
15
16     # Panel 1
17     price_corr_server(input, output, session)
18
19     # Panel 2
20     price_lm_server(input, output, session)
21
22     # Tab 4
23     clustering_server(input, output, session)
24
25 }

```

Listing 44: |introduction_server.R|

```

1 ##########
2 # Introduction Server - file #####
3 # Author: Silvia Ventoruzzo #####
4 ##########
5
6 introduction_server = function(input, output, session) {
7
8     output$intro_ui = renderUI({
9
10     text1 = "This ShinyApp was developed by "
11     url1 = a("Silvia Ventoruzzo",
12             href="https://www.linkedin.com/in/silvia-ventoruzzo-1
13             a042110a/")
14
15     text2 = " for the university course 'Statistical Programming
16     Languages' at the "
17     url2 = a("Humboldt-University of Berlin",
18             href="https://www.hu-berlin.de/")
19
20     text3 = " in the Winter Term 2018/19. It shows some Exploratory Data
21
22 }

```

```

    Analysis of the Airbnb properties
17   in Berlin updated at the 14. January 2019."
18
19   part1 = paste(text1, url1, text2, url2, text3, sep = "") %>%
20     paste("<br/>", sep = "<br/>")
21   html1 = HTML(part1)
22
23   text4 = "The data for this project has been downloaded from the
24     following sources:"
25   url4 = a("Inside Airbnb",
26             href="http://insideairbnb.com/")
27   url5 = a("Statistics Office of Berlin-Brandenburg",
28             href="https://www.statistik-berlin-brandenburg.de/produkte/
29               opendata/geometrienOD.asp?Kat=6301")
30   url6 = a("Geofabrik",
31             href="http://download.geofabrik.de/europe/germany/berlin.
32               html")
33
34   part2 = paste(text4, "<ul><li>", sep = "<br/>") %>%
35     paste(url4, sep = "") %>%
36     paste(url5, sep = "</li><li>") %>%
37     paste(url6, sep = "</li><li>") %>%
38     paste("", sep = "</li></ul><br/>")
39   html2 = HTML(part2)
40
41   text5 = "The complete code for this project can be found on"
42   url7 = a("GitHub",
43             href="https://github.com/silvia-ventoruzzo/SPL-WISE-2018")
44   text6 = "."
45
46   part3 = paste(text5, url7, sep = " ") %>%
47     paste(text6, sep = "")
48   html3 = HTML(part3)
49
50   tagList(html1, html2, html3)
51
52 }
53
54 output$intro_maps_text = renderText({
55   "Map of Berlin or one of its district and variable distribution."

```

```

54 })
55
56 output$intro_price_text = renderText({
57   "Variable correlation with price and linear regression on price."
58 })
59
60 output$intro_cluster_text = renderText({
61   "Clustering of Airbnb properties."
62 })
63
64 # Connections to other tabs
65 observeEvent(input$switch_maps, {
66   updateTabItems(session, inputId = "tabs", selected = "maps")
67 })
68
69 observeEvent(input$switch_price, {
70   updateTabItems(session, inputId = "tabs", selected = "price")
71 })
72
73 observeEvent(input$switch_cluster, {
74   updateTabItems(session, inputId = "tabs", selected = "cluster")
75 })
76
77 }

```

Listing 45: |maps_berlin_server.R|

```

1 #####
2 # Maps Berlin Server - file #####
3 # Author: Silvia Ventoruzzo #####
4 #####
5
6 maps_berlin_server = function(input, output, session) {
7
8   output$berlin_text <- renderText({
9     "In this tab you can view a map of entire Berlin divided in its
10       different areas. You can choose which view
11       to show and also which variable to show the average in the different
12       areas. Moreover you can also display
13       the properties themselves, the railway stations and the top 10 tourist
14       attractions."
15   })

```

```

13
14     ### LEAFLET MAP
15
16     ## TITLE
17     title_main_map_change = reactive({
18
19         if (variable1() == "none") {
20
21             paste("Map of Berlin's", input$view, sep = " ")
22
23         } else if (grepl(paste(factor_vars(), collapse = "|"), variable1())) {
24
25             variable = stringr::str_match(variable1(), paste(factor_vars(),
26                                         collapse = "|")) %>%
27                                         c() %>% gsub("_", " ", .) %>% capwords()
28
29             variable %>%
30                 paste("Count of the variable", ., "in Berlin's", input$view, sep =
31                         " ")
32
33         } else {
34
35             variable = capwords(input$variable1)
36
37             variable %>%
38                 paste("Average of the variable", ., "in Berlin's", input$view, sep
39                         = " ")
40
41     })
42
43     observeEvent({c(input$view, input$variable1)},
44
45         {output$title_main_map = renderText({title_main_map_change
46                                         ()})},
47
48         ignoreNULL = FALSE)
49
50     ## REACTIVE ELEMENTS
51
52     # SF Dataframe to plot polygons
53     polygons = reactive({berlin_sf %>% filter(view == input$view)})

```

```

50
51 # Dataframe with coordinates of the areas' names
52 area_names = reactive({berlin_names %>% filter(view == input$view)})
53
54 # Number of colors depend on how many areas to display
55 colors = reactive({
56   colorspace::rainbow_hcl(n = berlin_names %>%
57     filter(view == input$view) %>%
58     dplyr::summarize(count = n()) %>%
59     as.numeric())
60 })
61
62 # Rename variable1
63 variable1 = reactive({
64
65   gsub(" ", "_", input$variable1)
66
67 })
68
69 var_table_plot = reactive({
70
71   if (variable1() == "none") { return() }
72
73   if (grepl(paste(factor_vars(), collapse = " | "), variable1())) {
74
75     variable = stringr::str_match(variable1(), paste(factor_vars(),
76       collapse = " | ")) %>% c()
77
78   } else {
79
80     variable = variable1()
81
82   }
83
84 })
85
86 # Number of areas in selected view
87 area_num = reactive({
88
89   berlin_names %>%
90     dplyr::filter(view == input$view) %>%

```

```

90     dplyr::summarize(count = n()) %>%
91     as.numeric()
92
93   })
94
95   # Colors for average maps
96   colors_var = reactive({
97     leaflet::colorNumeric(
98       palette = topo.colors(area_num()),
99       domain = listings_area_summary %>%
100         dplyr::filter(view == input$view) %>%
101         dplyr::select(variable1()),
102       n      = area_num(),
103       reverse = TRUE)})
104
105   # Text size depend on how many areas to display
106   text_size = reactive({ifelse(input$view == "Districts", "11px", "20px")
107 })
108
109   ## MAP
110
111   output$main_map = renderLeaflet({
112
113     main_icons = awesomeIconList(
114       "main_properties" = makeAwesomeIcon(icon = "home", markerColor = "red"),
115       "main_stations"   = makeAwesomeIcon(icon = "subway", library = "fa",
116                                             markerColor = "green", iconColor = "#FFFFFF"),
117       "main_sights"     = makeAwesomeIcon(icon = "eye-open", markerColor =
118                                             "blue"))
119
120     main_map = leaflet() %>%
121       addTiles() %>%
122       addLabelOnlyMarkers(data = area_names(),
123                             lng = ~long, lat = ~lat, label = ~lapply(name,
124                               htmltools::HTML),
125                             labelOptions = labelOptions(noHide = TRUE,
126                               direction = 'center',
127                               textOnly = TRUE,
128                               textsize = text_

```

```

124                                         size()))  %>%
125
126   addAwesomeMarkers(data = listings,
127
128     lng = ~long, lat = ~lat,
129     icon = main_icons[["main_properties"]],
130     group = "main_properties",
131     popup = ~lapply(paste("<b><a href=", listing_url,
132
133       "> ID: ", id, "</a></b>", sep = "") %>%
134
135       paste("Price: ", sep = "<br/>") %>%
136
137       paste(price, "$", sep = "") %>%
138       paste(property_type, sep = "<br/>") %>%
139
140       paste(room_type, sep = " - ") %>%
141
142       paste("Accomodates", sep = "<br/>") %>%
143
144       paste(accommodates, ifelse(
145
146         accommodates == 1, "person", "people"),
147         sep = " "),
148
149       htmltools::HTML),
150
151       clusterId = "1",
152
153       clusterOptions = markerClusterOptions(
154
155         iconCreateFunction=JS("function (cluster) {
156
157           var childCount = cluster.
158
159             getChildCount();
160
161           if (childCount > 1) {
162             c = 'rgba(235, 0, 0, 1);'
163
164             return new L.DivIcon({
165
166               html: '<div style='
167                 'background-color:' + c + ' \
168                 '><span>' + childCount +
169                 '</span></div>',
170
171               className: 'marker-
172
173                 cluster', iconSize: new
174                 L.Point(40, 40 )});}'
175
176           singleMarkerMode = FALSE)) %>%
177
178   addAwesomeMarkers(data = railway_stations_df,
179
180     lng = ~long, lat = ~lat,
181
182     icon = main_icons[["main_stations"]],
183
184     popup = ~htmltools::htmlEscape(
185
186       paste(ifelse(s_bahn == TRUE, "S", ""))

```

```

149          ifelse(s_bahn == TRUE & u_bahn == TRUE, "+"
150                  " , ""),
151          ifelse(u_bahn == TRUE, "U", ""),
152          sep = "") %>%
153          paste(id, sep = " "),
154          group = "main_stations",
155          clusterId = "2",
156          clusterOptions = markerClusterOptions(
157              iconCreateFunction = JS("function (cluster) {
158                  var childCount = cluster
159                      .getChildCount();
160                  if (childCount > 1) {
161                      c = 'rgba(7, 187, 31, 1)
162                      ;'}
163                  return new L.DivIcon({
164                      html: '<div style='
165                          background-color:' +c+
166                          '><span>' +
167                          childCount + '</span><
168                          /div>', className: '
169                          marker-cluster',
170                          iconSize: new L.Point
171                          (40, 40) });
172          ),
173          singleMarkerMode = FALSE)) %>%
174
175          addAwesomeMarkers(data = attractions_df,
176                  lng = ~long, lat = ~lat,
177                  icon = main_icons[["main_sights"]],
178                  popup = ~htmltools::htmlEscape(id),
179                  group = "main_sights",
180                  clusterId = "3",
181                  clusterOptions = markerClusterOptions(
182                      iconCreateFunction = JS("function (cluster) {
183                          var childCount = cluster
184                              .getChildCount();
185                          if (childCount > 1) {
186                              c = 'rgba(11, 200, 213,
187                              1);'}
188                          return new L.DivIcon({
189                              html: '<div style='
190                                  background-color:' +c+
191                                  '><span>' +

```

```

                                childCount + '</span><
                                /div>', className: 'marker-cluster',
                                iconSize: new L.Point
                                (40, 40) })};"),

174             singleMarkerMode = FALSE)) %>%
175             hideGroup(group = c("main_properties", "main_stations", "main_sights
176             ")) %>%
177             setView(lng = berlin_center$long, lat = berlin_center$lat, zoom =
178             10)

179             if (variable1() == "none") {

180                 main_map %>%
181                     addPolygons(data = polygons(), weight = 1, smoothFactor = 1,
182                     fillOpacity = 0.8,
183                     color = "grey", fillColor = colors())
184             } else {

185                 colors_vars = colors_var()

186                 main_map %>%
187                     addPolygons(data = polygons(), weight = 1, smoothFactor = 1,
188                     fillOpacity = 0.8,
189                     color = "grey",
190                     fillColor = ~colors_vars(listings_area_summary[
191                         listings_area_summary$view == input$view, variable1
192                         ()])) %>%
193                     addLegend(position = "bottomleft", pal = colors_var(),
194                     values = listings_area_summary[listings_area_summary$view ==
195                         input$view, variable1()],
196                     labFormat = labelFormat(transform = function(x) sort(x,
197                         decreasing = TRUE)),
198                     title = variable1())

199             }

200         }

201         # checkboxInputs
202         observeEvent({c(input$main_properties, input$main_stations, input$main_

```

```

sights)}, {

202
proxy = leafletProxy("main_map")

204
if (input$main_properties) {proxy %>% showGroup("main_properties")
} else {proxy %>% hideGroup("main_properties")}

207
if (input$main_stations) {proxy %>% showGroup("main_stations")
} else {proxy %>% hideGroup("main_stations")}

210
if (input$main_sights) {proxy %>% showGroup("main_sights")
} else {proxy %>% hideGroup("main_sights")}

213
})

215
216
### DESCRIPTIVE TABLE

218
## TITLE
title_main_table_change = reactive({

221
222
if (variable1() == "none") { return() }

223
if (grepl(paste(factor_vars(), collapse = "|"), variable1())) {

225
variable = stringr::str_match(variable1(), paste(factor_vars(),
collapse = "|")) %>%
227
c() %>% gsub("_", " ", .) %>% capwords()

228
} else {
229
230
variable = capwords(input$variable1)
231
}
232
233
234
}
235
236
variable %>%
paste("Descriptive statistics of the variable", ., sep = " ")
237
}
238
})

```

```

241 observeEvent({c(input$view, input$variable1)},
242               {output$title_main_table = renderText({title_main_table_
243                                         change()})},
244               ignoreNULL = FALSE)
245
246 ## REACTIVE ELEMENTS
247
248 # Categorical variables
249 factor_vars = reactive({
250
251   listings %>%
252     dplyr::select(-id, -listing_url, -long, -lat, -vbb_zone, -district,
253     -neighbourhood) %>%
254     dplyr::select_if(function(x) is.character(x) | is.factor(x) | is.
255     logical(x)) %>%
256     names()
257
258 })
259
260
261 var_table_plot = reactive({
262
263   if (variable1() == "none") { return() }
264
265   if (grepl(paste(factor_vars(), collapse = "|"), variable1())) {
266
267     variable = stringr::str_match(variable1(), paste(factor_vars(),
268       collapse = "|")) %>% c()
269
270   } else {
271
272     variable = variable1()
273
274   }
275
276   })
277
278 ## TABLES
279
280 output$main_table1 = renderTable(

```

```

278     if (variable1() == "none") {
279
280         return()
281
282     } else if (grepl(paste(factor_vars(), collapse = " | "), variable1())) {
283
284         main_df = statistics_cat %>%
285             dplyr::filter(variable == var_table_plot()) %>%
286             dplyr::select(-variable)
287
288     } else {
289
290         main_df = statistics_num %>%
291             dplyr::filter(variable == var_table_plot()) %>%
292             dplyr::select(min, '1Q', median, '3Q', max)
293     }
294
295     return(main_df)
296
297 })
298
299 output$main_table2 = renderTable({
300
301     if (variable1() == "none") {
302
303         return()
304
305     } else if (grepl(paste(factor_vars(), collapse = " | "), variable1())) {
306
307         return()
308
309     } else {
310
311         main_df = statistics_num %>%
312             dplyr::filter(variable == var_table_plot()) %>%
313             dplyr::select(mean, sd)
314     }
315
316     return(main_df)
317
318 })

```

```

319
320
321 ### DISTRIBUTION PLOT
322
323 ## TITLE
324 title_main_plot_change = reactive{
325
326   if (variable1() == "none") { return() }
327
328   if (grepl(paste(factor_vars()), collapse = " | ", variable1())) {
329
330     variable = stringr::str_match(variable1(), paste(factor_vars(),
331       collapse = " | ")) %>%
332       c() %>% gsub("_", " ", .) %>% capwords()
333
334   } else {
335
336     variable = capwords(variable1())
337
338   }
339
340   variable %>%
341     paste("Distribution of the variable", ., sep = " ")
342 )
343
344 observeEvent(c(input$view, input$variable1),
345   {output$title_main_plot = renderText({title_main_plot_
346     change())}),
347   ignoreNULL = FALSE)
348
349
350 ## REACTIVE ELEMENTS
351
352 ## PLOT
353
354 output$main_plot = renderPlot{
355
356   if (variable1() == "none") {
357
358     return()
359
360   }
361
362   ggplot(mpg, geom_bar(mapping(fill = discrete(cty)),
363     stat = "count"))
364
365   scale_x_discrete(labels = mpg)
366
367   scale_y_continuous(breaks = mpg)
368
369   theme_minimal()
370
371   ggplot2:::print(mpg)
372
373   grid(mpg)
374
375   grid(mpg)
376
377   grid(mpg)
378
379   grid(mpg)
380
381   grid(mpg)
382
383   grid(mpg)
384
385   grid(mpg)
386
387   grid(mpg)
388
389   grid(mpg)
390
391   grid(mpg)
392
393   grid(mpg)
394
395   grid(mpg)
396
397   grid(mpg)
398
399   grid(mpg)
400
401   grid(mpg)
402
403   grid(mpg)
404
405   grid(mpg)
406
407   grid(mpg)
408
409   grid(mpg)
410
411   grid(mpg)
412
413   grid(mpg)
414
415   grid(mpg)
416
417   grid(mpg)
418
419   grid(mpg)
420
421   grid(mpg)
422
423   grid(mpg)
424
425   grid(mpg)
426
427   grid(mpg)
428
429   grid(mpg)
430
431   grid(mpg)
432
433   grid(mpg)
434
435   grid(mpg)
436
437   grid(mpg)
438
439   grid(mpg)
440
441   grid(mpg)
442
443   grid(mpg)
444
445   grid(mpg)
446
447   grid(mpg)
448
449   grid(mpg)
450
451   grid(mpg)
452
453   grid(mpg)
454
455   grid(mpg)
456
457   grid(mpg)
458
459   grid(mpg)
460
461   grid(mpg)
462
463   grid(mpg)
464
465   grid(mpg)
466
467   grid(mpg)
468
469   grid(mpg)
470
471   grid(mpg)
472
473   grid(mpg)
474
475   grid(mpg)
476
477   grid(mpg)
478
479   grid(mpg)
480
481   grid(mpg)
482
483   grid(mpg)
484
485   grid(mpg)
486
487   grid(mpg)
488
489   grid(mpg)
490
491   grid(mpg)
492
493   grid(mpg)
494
495   grid(mpg)
496
497   grid(mpg)
498
499   grid(mpg)
500
501   grid(mpg)
502
503   grid(mpg)
504
505   grid(mpg)
506
507   grid(mpg)
508
509   grid(mpg)
510
511   grid(mpg)
512
513   grid(mpg)
514
515   grid(mpg)
516
517   grid(mpg)
518
519   grid(mpg)
520
521   grid(mpg)
522
523   grid(mpg)
524
525   grid(mpg)
526
527   grid(mpg)
528
529   grid(mpg)
530
531   grid(mpg)
532
533   grid(mpg)
534
535   grid(mpg)
536
537   grid(mpg)
538
539   grid(mpg)
540
541   grid(mpg)
542
543   grid(mpg)
544
545   grid(mpg)
546
547   grid(mpg)
548
549   grid(mpg)
550
551   grid(mpg)
552
553   grid(mpg)
554
555   grid(mpg)
556
557   grid(mpg)
558
559   grid(mpg)
560
561   grid(mpg)
562
563   grid(mpg)
564
565   grid(mpg)
566
567   grid(mpg)
568
569   grid(mpg)
570
571   grid(mpg)
572
573   grid(mpg)
574
575   grid(mpg)
576
577   grid(mpg)
578
579   grid(mpg)
580
581   grid(mpg)
582
583   grid(mpg)
584
585   grid(mpg)
586
587   grid(mpg)
588
589   grid(mpg)
590
591   grid(mpg)
592
593   grid(mpg)
594
595   grid(mpg)
596
597   grid(mpg)
598
599   grid(mpg)
600
601   grid(mpg)
602
603   grid(mpg)
604
605   grid(mpg)
606
607   grid(mpg)
608
609   grid(mpg)
610
611   grid(mpg)
612
613   grid(mpg)
614
615   grid(mpg)
616
617   grid(mpg)
618
619   grid(mpg)
620
621   grid(mpg)
622
623   grid(mpg)
624
625   grid(mpg)
626
627   grid(mpg)
628
629   grid(mpg)
630
631   grid(mpg)
632
633   grid(mpg)
634
635   grid(mpg)
636
637   grid(mpg)
638
639   grid(mpg)
640
641   grid(mpg)
642
643   grid(mpg)
644
645   grid(mpg)
646
647   grid(mpg)
648
649   grid(mpg)
650
651   grid(mpg)
652
653   grid(mpg)
654
655   grid(mpg)
656
657   grid(mpg)
658
659   grid(mpg)
660
661   grid(mpg)
662
663   grid(mpg)
664
665   grid(mpg)
666
667   grid(mpg)
668
669   grid(mpg)
670
671   grid(mpg)
672
673   grid(mpg)
674
675   grid(mpg)
676
677   grid(mpg)
678
679   grid(mpg)
680
681   grid(mpg)
682
683   grid(mpg)
684
685   grid(mpg)
686
687   grid(mpg)
688
689   grid(mpg)
690
691   grid(mpg)
692
693   grid(mpg)
694
695   grid(mpg)
696
697   grid(mpg)
698
699   grid(mpg)
700
701   grid(mpg)
702
703   grid(mpg)
704
705   grid(mpg)
706
707   grid(mpg)
708
709   grid(mpg)
710
711   grid(mpg)
712
713   grid(mpg)
714
715   grid(mpg)
716
717   grid(mpg)
718
719   grid(mpg)
720
721   grid(mpg)
722
723   grid(mpg)
724
725   grid(mpg)
726
727   grid(mpg)
728
729   grid(mpg)
730
731   grid(mpg)
732
733   grid(mpg)
734
735   grid(mpg)
736
737   grid(mpg)
738
739   grid(mpg)
740
741   grid(mpg)
742
743   grid(mpg)
744
745   grid(mpg)
746
747   grid(mpg)
748
749   grid(mpg)
750
751   grid(mpg)
752
753   grid(mpg)
754
755   grid(mpg)
756
757   grid(mpg)
758
759   grid(mpg)
760
761   grid(mpg)
762
763   grid(mpg)
764
765   grid(mpg)
766
767   grid(mpg)
768
769   grid(mpg)
770
771   grid(mpg)
772
773   grid(mpg)
774
775   grid(mpg)
776
777   grid(mpg)
778
779   grid(mpg)
780
781   grid(mpg)
782
783   grid(mpg)
784
785   grid(mpg)
786
787   grid(mpg)
788
789   grid(mpg)
790
791   grid(mpg)
792
793   grid(mpg)
794
795   grid(mpg)
796
797   grid(mpg)
798
799   grid(mpg)
800
801   grid(mpg)
802
803   grid(mpg)
804
805   grid(mpg)
806
807   grid(mpg)
808
809   grid(mpg)
810
811   grid(mpg)
812
813   grid(mpg)
814
815   grid(mpg)
816
817   grid(mpg)
818
819   grid(mpg)
820
821   grid(mpg)
822
823   grid(mpg)
824
825   grid(mpg)
826
827   grid(mpg)
828
829   grid(mpg)
830
831   grid(mpg)
832
833   grid(mpg)
834
835   grid(mpg)
836
837   grid(mpg)
838
839   grid(mpg)
840
841   grid(mpg)
842
843   grid(mpg)
844
845   grid(mpg)
846
847   grid(mpg)
848
849   grid(mpg)
850
851   grid(mpg)
852
853   grid(mpg)
854
855   grid(mpg)
856
857   grid(mpg)
858
859   grid(mpg)
860
861   grid(mpg)
862
863   grid(mpg)
864
865   grid(mpg)
866
867   grid(mpg)
868
869   grid(mpg)
870
871   grid(mpg)
872
873   grid(mpg)
874
875   grid(mpg)
876
877   grid(mpg)
878
879   grid(mpg)
880
881   grid(mpg)
882
883   grid(mpg)
884
885   grid(mpg)
886
887   grid(mpg)
888
889   grid(mpg)
890
891   grid(mpg)
892
893   grid(mpg)
894
895   grid(mpg)
896
897   grid(mpg)
898
899   grid(mpg)
900
901   grid(mpg)
902
903   grid(mpg)
904
905   grid(mpg)
906
907   grid(mpg)
908
909   grid(mpg)
910
911   grid(mpg)
912
913   grid(mpg)
914
915   grid(mpg)
916
917   grid(mpg)
918
919   grid(mpg)
920
921   grid(mpg)
922
923   grid(mpg)
924
925   grid(mpg)
926
927   grid(mpg)
928
929   grid(mpg)
930
931   grid(mpg)
932
933   grid(mpg)
934
935   grid(mpg)
936
937   grid(mpg)
938
939   grid(mpg)
940
941   grid(mpg)
942
943   grid(mpg)
944
945   grid(mpg)
946
947   grid(mpg)
948
949   grid(mpg)
950
951   grid(mpg)
952
953   grid(mpg)
954
955   grid(mpg)
956
957   grid(mpg)
958
959   grid(mpg)
960
961   grid(mpg)
962
963   grid(mpg)
964
965   grid(mpg)
966
967   grid(mpg)
968
969   grid(mpg)
970
971   grid(mpg)
972
973   grid(mpg)
974
975   grid(mpg)
976
977   grid(mpg)
978
979   grid(mpg)
980
981   grid(mpg)
982
983   grid(mpg)
984
985   grid(mpg)
986
987   grid(mpg)
988
989   grid(mpg)
990
991   grid(mpg)
992
993   grid(mpg)
994
995   grid(mpg)
996
997   grid(mpg)
998
999   grid(mpg)
1000
1001   grid(mpg)
1002
1003   grid(mpg)
1004
1005   grid(mpg)
1006
1007   grid(mpg)
1008
1009   grid(mpg)
1010
1011   grid(mpg)
1012
1013   grid(mpg)
1014
1015   grid(mpg)
1016
1017   grid(mpg)
1018
1019   grid(mpg)
1020
1021   grid(mpg)
1022
1023   grid(mpg)
1024
1025   grid(mpg)
1026
1027   grid(mpg)
1028
1029   grid(mpg)
1030
1031   grid(mpg)
1032
1033   grid(mpg)
1034
1035   grid(mpg)
1036
1037   grid(mpg)
1038
1039   grid(mpg)
1040
1041   grid(mpg)
1042
1043   grid(mpg)
1044
1045   grid(mpg)
1046
1047   grid(mpg)
1048
1049   grid(mpg)
1050
1051   grid(mpg)
1052
1053   grid(mpg)
1054
1055   grid(mpg)
1056
1057   grid(mpg)
1058
1059   grid(mpg)
1060
1061   grid(mpg)
1062
1063   grid(mpg)
1064
1065   grid(mpg)
1066
1067   grid(mpg)
1068
1069   grid(mpg)
1070
1071   grid(mpg)
1072
1073   grid(mpg)
1074
1075   grid(mpg)
1076
1077   grid(mpg)
1078
1079   grid(mpg)
1080
1081   grid(mpg)
1082
1083   grid(mpg)
1084
1085   grid(mpg)
1086
1087   grid(mpg)
1088
1089   grid(mpg)
1090
1091   grid(mpg)
1092
1093   grid(mpg)
1094
1095   grid(mpg)
1096
1097   grid(mpg)
1098
1099   grid(mpg)
1100
1101   grid(mpg)
1102
1103   grid(mpg)
1104
1105   grid(mpg)
1106
1107   grid(mpg)
1108
1109   grid(mpg)
1110
1111   grid(mpg)
1112
1113   grid(mpg)
1114
1115   grid(mpg)
1116
1117   grid(mpg)
1118
1119   grid(mpg)
1120
1121   grid(mpg)
1122
1123   grid(mpg)
1124
1125   grid(mpg)
1126
1127   grid(mpg)
1128
1129   grid(mpg)
1130
1131   grid(mpg)
1132
1133   grid(mpg)
1134
1135   grid(mpg)
1136
1137   grid(mpg)
1138
1139   grid(mpg)
1140
1141   grid(mpg)
1142
1143   grid(mpg)
1144
1145   grid(mpg)
1146
1147   grid(mpg)
1148
1149   grid(mpg)
1150
1151   grid(mpg)
1152
1153   grid(mpg)
1154
1155   grid(mpg)
1156
1157   grid(mpg)
1158
1159   grid(mpg)
1160
1161   grid(mpg)
1162
1163   grid(mpg)
1164
1165   grid(mpg)
1166
1167   grid(mpg)
1168
1169   grid(mpg)
1170
1171   grid(mpg)
1172
1173   grid(mpg)
1174
1175   grid(mpg)
1176
1177   grid(mpg)
1178
1179   grid(mpg)
1180
1181   grid(mpg)
1182
1183   grid(mpg)
1184
1185   grid(mpg)
1186
1187   grid(mpg)
1188
1189   grid(mpg)
1190
1191   grid(mpg)
1192
1193   grid(mpg)
1194
1195   grid(mpg)
1196
1197   grid(mpg)
1198
1199   grid(mpg)
1200
1201   grid(mpg)
1202
1203   grid(mpg)
1204
1205   grid(mpg)
1206
1207   grid(mpg)
1208
1209   grid(mpg)
1210
1211   grid(mpg)
1212
1213   grid(mpg)
1214
1215   grid(mpg)
1216
1217   grid(mpg)
1218
1219   grid(mpg)
1220
1221   grid(mpg)
1222
1223   grid(mpg)
1224
1225   grid(mpg)
1226
1227   grid(mpg)
1228
1229   grid(mpg)
1230
1231   grid(mpg)
1232
1233   grid(mpg)
1234
1235   grid(mpg)
1236
1237   grid(mpg)
1238
1239   grid(mpg)
1240
1241   grid(mpg)
1242
1243   grid(mpg)
1244
1245   grid(mpg)
1246
1247   grid(mpg)
1248
1249   grid(mpg)
1250
1251   grid(mpg)
1252
1253   grid(mpg)
1254
1255   grid(mpg)
1256
1257   grid(mpg)
1258
1259   grid(mpg)
1260
1261   grid(mpg)
1262
1263   grid(mpg)
1264
1265   grid(mpg)
1266
1267   grid(mpg)
1268
1269   grid(mpg)
1270
1271   grid(mpg)
1272
1273   grid(mpg)
1274
1275   grid(mpg)
1276
1277   grid(mpg)
1278
1279   grid(mpg)
1280
1281   grid(mpg)
1282
1283   grid(mpg)
1284
1285   grid(mpg)
1286
1287   grid(mpg)
1288
1289   grid(mpg)
1290
1291   grid(mpg)
1292
1293   grid(mpg)
1294
1295   grid(mpg)
1296
1297   grid(mpg)
1298
1299   grid(mpg)
1300
1301   grid(mpg)
1302
1303   grid(mpg)
1304
1305   grid(mpg)
1306
1307   grid(mpg)
1308
1309   grid(mpg)
1310
1311   grid(mpg)
1312
1313   grid(mpg)
1314
1315   grid(mpg)
1316
1317   grid(mpg)
1318
1319   grid(mpg)
1320
1321   grid(mpg)
1322
1323   grid(mpg)
1324
1325   grid(mpg)
1326
1327   grid(mpg)
1328
1329   grid(mpg)
1330
1331   grid(mpg)
1332
1333   grid(mpg)
1334
1335   grid(mpg)
1336
1337   grid(mpg)
1338
1339   grid(mpg)
1340
1341   grid(mpg)
1342
1343   grid(mpg)
1344
1345   grid(mpg)
1346
1347   grid(mpg)
134
```

```

358 } else if (grepl(paste(factor_vars(), collapse = " | "), variable1())) {
359
360   variable = stringr::str_match(variable1(), paste(factor_vars(),
361                                     collapse = " | ")) %>%
362     c() %>% gsub("_", " ", .) %>% capwords()
363
364   plot = distribution_plot(listings, var_table_plot(), tilt_text_x =
365                             TRUE)
366
367 } else {
368
369   plot = distribution_plot(listings, var_table_plot(), hide_outliers =
370                             TRUE)
371
372 }
373
374 return(plot)
375
376 }

```

Listing 46: |maps_district_server.R|

```

1 ##########
2 # Maps District Server - file #####
3 # Author: Silvia Ventoruzzo #####
4 ##########
5
6 maps_district_server = function(input, output, session) {
7
8   output$district_text <- renderText({
9     "This tab is similar to the previous one. You can still choose which
10       variable to show the average
11       in the different areas and display various properties, railway stations
12       and tourist attractions.
13
14       But here you see the city at the district's level, showing therefore the
15       neighbourhoods in the
16       different districts."
17   })
18
19   #### LEAFLET MAP

```

```

16
17 ## TITLE
18
19 title_secondary_map_change = reactive({
20
21   if (variable2() == "none") {
22
23     paste("Map of", input$district, sep = " ") %>%
24       paste('s', sep = "") %>%
25       paste("Neighbourhoods", sep = " ")
26
27 } else if (grepl(paste(factor_vars(), collapse = "|"), variable2())) {
28
29   variable = stringr::str_match(variable2(), paste(factor_vars(),
30                                     collapse = "|")) %>%
31     c() %>% gsub("_", " ", .) %>% capwords()
32
33   variable %>%
34     paste("Count of the variable", ., "in the district", input$district,
35           sep = " ")
36
37 } else {
38
39   variable = capwords(input$variable2)
40
41   variable %>%
42     paste("Average of the variable", ., "in the district", input$district,
43           sep = " ")
44
45 }
46
47 observeEvent({c(input$district, input$variable2)},
48               {output$title_secondary_map = renderText({title_secondary_map_change()})},
49               ignoreNULL = FALSE)
50
51 ## REACTIVE ELEMENTS
52
53 # SF Dataframe to plot polygons
54 neighbourhoods = reactive({berlin_sf %>%

```

```

53     filter(view == "Neighbourhoods",
54             group == input$district)})

55

56 # Dataframe with coordinates of the areas' names
57 neighbourhood_names = reactive({berlin_names %>%
58     filter(view == "Neighbourhoods",
59             group == input$district)})

60

61 # Number of colors depend on how many areas to display
62 neighbourhood_colors = reactive({
63     colorspace::rainbow_hcl(n = berlin_names %>%
64                             filter(view == "Neighbourhoods",
65                                     group == input$district) %>%
66                             dplyr::summarize(count = n()) %>%
67                             as.numeric())
68 })

69

70 # Rename variable2
71 variable2 = reactive({
72
73     gsub(" ", "_", input$variable2)
74
75 })

76

77 # Number of areas in selected view
78 neighbourhood_area_num = reactive({
79
80     berlin_names %>%
81         dplyr::filter(view == "Neighbourhoods",
82                         group == input$district) %>%
83         dplyr::summarize(count = n()) %>%
84         as.numeric()
85
86 })

87

88 # Colors for average maps
89 neighbourhood_colors_var = reactive({
90     leaflet::colorNumeric(
91         palette = topo.colors(area_num()),
92         domain = listings_area_summary %>%
93             dplyr::filter(view == input$view) %>%
```

```

94     dplyr::select(variable1()),
95     n       = area_num(),
96     reverse = TRUE)})

97

98

99 # When choosing a variable (input$variable2), the color should represent
100 # the variable's distribution
101 neighbourhood_colors_var = reactive({
102   leaflet::colorNumeric(
103     palette = topo.colors(neighbourhood_area_num()),
104     domain  = listings_area_summary %>%
105       dplyr::filter(view == "Neighbourhoods",
106                     group == input$district) %>%
107       dplyr::select(variable2()),
108     n       = neighbourhood_area_num(),
109     reverse = TRUE)
110   })

111

112

113 ## MAP
114

115 output$secondary_map = renderLeaflet({
116
117   district_icons = awesomeIconList(
118     "district_properties" = makeAwesomeIcon(icon = "home", markerColor = "red"),
119     "district_stations"   = makeAwesomeIcon(icon = "subway", library = "fa",
120                                               markerColor = "green", iconColor = "#FFFFFF"),
121     "district_sights"     = makeAwesomeIcon(icon = "eye-open", markerColor =
122                                               "blue")
123   )
124
125   secondary_map = leaflet() %>%
126     addTiles() %>%
127     addLabelOnlyMarkers(data = neighbourhood_names(),
128                           lng = ~long, lat = ~lat, label = ~lapply(name,
129                                             htmltools::HTML),
130                           labelOptions = labelOptions(noHide = TRUE,
131                                         direction = 'center',
132                                         textOnly = TRUE,

```

```

                                         textsize = "11px"))
%>%
129 addAwesomeMarkers(data = listings %>% filter(district == input$district),
130   lng = ~long, lat = ~lat,
131   icon = district_icons[["district_properties"]],
132   group = "district_properties",
133   popup = ~lapply(paste("<b><a href=", listing_url, ">
ID: ", id, "</a></b>", sep = "") %>%
134     paste("Price: ", sep = "<br/>") %>%
135     paste(price, "$", sep = "") %>%
136     paste(property_type, sep = "<br/>") %>%
137     paste(room_type, sep = " - ") %>%
138     paste("Accomodates", sep = "<br/>") %>%
139     paste(accommodates, ifelse(
140       accommodates == 1, "person", "people"), sep = " "),
141     htmltools::HTML),
142   clusterId = "1",
143   clusterOptions = markerClusterOptions(
144     iconCreateFunction=JS("function (cluster) {
145       var childCount = cluster.
146         getChildCount();
147       if (childCount > 1) {
148         c = 'rgba(235, 0, 0, 1)';
149         return new L.DivIcon({ html:
150           '<div style=\"background-
151           color:' +c+ '\"><span>' +
152             childCount + '</span></div>',
153           className: 'marker-
154             cluster', iconSize: new L.
155             Point(40, 40) });
156       }
157     singleMarkerMode = FALSE));
158   ) %>%
159 addAwesomeMarkers(data = railway_stations_df %>% filter(district ==
160   input$district),
161   lng = ~long, lat = ~lat,
162   icon = district_icons[["district_stations"]],
163   popup = ~htmltools::htmlEscape(

```

```

153     paste(ifelse(s_bahn == TRUE, "S", ""),
154           ifelse(s_bahn == TRUE & u_bahn == TRUE, "+",
155                 ""),
156           ifelse(u_bahn == TRUE, "U", ""),
157             sep = "") %>%
158   paste(id, sep = " ")),
159   group = "district_stations",
160   clusterId = "2",
161   clusterOptions = markerClusterOptions(
162     iconCreateFunction = JS("function (cluster) {
163       var childCount = cluster.
164         getChildCount();
165       if (childCount > 1) {
166         c = 'rgba(7, 187, 31, 1)
167         ;'}
168       return new L.DivIcon({
169         html: '<div style="'
170           background-color:' + c + '"'
171         '><span>' + childCount +
172           '</span></div>',
173         className: 'marker-
174           cluster', iconSize: new
175             L.Point(40, 40) });
176     singleMarkerMode = FALSE)) %>%
177   addAwesomeMarkers(data = attractions_df %>% filter(district == input$district),
178     lng = ~long, lat = ~lat,
179     icon = district_icons[["district_sights"]],
180     popup = ~htmltools::htmlEscape(id),
181     group = "district_sights",
182     clusterId = "3",
183     clusterOptions = markerClusterOptions(
184       iconCreateFunction = JS("function (cluster) {
185         var childCount = cluster.
186           getChildCount();
187         if (childCount > 1) {
188           c = 'rgba(11, 200, 213, 1)
189           ;'}
190         return new L.DivIcon({
191           html: '<div style="'
192             background-color:' + c + '"'

```

```

179 ><span>' + childCount +
180   '</span></div>',
181   className: 'marker-
182   cluster', iconSize: new
183   L.Point(40, 40) };}"),
184   singleMarkerMode = FALSE)) %>%
185 hideGroup(group = c("district_properties", "district_stations", "
186 district_sights")) %>%
187 addMiniMap(centerFixed      = berlin_center %>% t() %>% c(),
188             zoomLevelFixed = 8,
189             zoomAnimation  = FALSE)
190
191
192 # if no variable is selected
193 if (variable2() == "none") {
194
195   secondary_map %>%
196     addPolygons(data = neighbourhoods(), weight = 1, smoothFactor = 1,
197                 fillOpacity = 0.8,
198                 color = "grey", fillColor = neighbourhood_colors())
199
200 # if variable is selected
201 } else {
202
203   neighbourhood_colors_vars = neighbourhood_colors_var()
204
205   secondary_map %>%
206     addPolygons(data = neighbourhoods(), weight = 1, smoothFactor = 1,
207                 fillOpacity = 0.8,
208                 color = "grey",
209                 fillColor = ~neighbourhood_colors_vars(
210                   listings_area_summary
211                     [listings_area_summary$view == "
212                       Neighbourhoods" &
213                         listings_area_summary$group ==
214                           input$district,
215                           variable2())]) %>%
216     addLegend(position = "bottomleft", pal = neighbourhood_colors_var(),
217               values = listings_area_summary[listings_area_summary$view
218                                         == "Neighbourhoods" &
219                                           listings_area_summary$group ==
220                                             input$district

```

```

208
209     variable2()]),
210     labFormat = labelFormat(transform = function(x) sort(x,
211         decreasing = TRUE)),
212     title = variable2()
213   }
214 }
215
# checkboxInputs
216 observeEvent({c(input$district_properties, input$district_stations, input$district_sights, input$district)}, {
217
218   proxy = leafletProxy("secondary_map")
219
220   if (input$district_properties) {proxy %>% showGroup("district_properties")}
221 } else {proxy %>% hideGroup("district_properties")}
222
223   if (input$district_stations) {proxy %>% showGroup("district_stations")}
224 } else {proxy %>% hideGroup("district_stations")}
225
226   if (input$district_sights) {proxy %>% showGroup("district_sights")}
227 } else {proxy %>% hideGroup("district_sights")}
228 }
229
230
# ## SUMMARY TABLE
231
232
## TITLE
233
234
235 title_secondary_table_change = reactive({
236
237   if (variable2() == "none") { return() }
238
239   if (grepl(paste(factor_vars(), collapse = "|"), variable2())) {
240
241     variable = stringr::str_match(variable2(), paste(factor_vars(),
242         collapse = "|")) %>%
243       c() %>% gsub("_", " ", .) %>% capwords()
244
245   }
246 }

```

```

244 } else {
245
246   variable = capwords(input$variable2)
247
248 }
249
250 variable %>%
251   paste("Summary statistics of the variable", ., "in the district",
252       input$district, sep = " ")
253
254 )
255
256 observeEvent({c(input$district, input$variable2)},
257               {output$title_secondary_table = renderText({title_secondary_
258                                         table_change()})}, ignoreNULL = FALSE)
259
260 ## REACTIVE ELEMENTS
261
262 # Categorical variables
263 factor_vars = reactive({
264
265   listings %>%
266     dplyr::select(-id, -listing_url, -long, -lat, -vbb_zone, -district, -
267       neighbourhood) %>%
268     dplyr::select_if(function(x) is.character(x) | is.factor(x) | is.
269       logical(x)) %>%
270     names()
271
272 })
273
274 var_table_plot2 = reactive({
275
276   if (variable2() == "none") { return() }
277
278   if (grepl(paste(factor_vars(), collapse = "|"), variable2())) {
279
280     variable = stringr::str_match(variable2(), paste(factor_vars(),
281       collapse = "|")) %>% c()
282
283   } else {

```

```

280     variable = variable2()
281 
282   }
283 
284 })
285 
286 
287 table_df = reactive({
288 
289   if (variable2() == "none") { return() }
290 
291   df = listings %>%
292     dplyr::filter(district == input$district) %>%
293     dplyr::select(var_table_plot2())
294 
295   return(df)
296 
297 })
298 
299 ## TABLE
300 
301 # Descriptive table
302 output$secondary_table1 = renderTable({
303 
304   if (variable2() == "none") {
305 
306     return()
307 
308   } else if (grepl(paste(factor_vars(), collapse = "|"), variable2())) {
309 
310     secondary_df = descriptive_statistics(table_df())
311 
312   } else {
313 
314     secondary_df = descriptive_statistics(table_df()) %>%
315       dplyr::select(min, '1Q', median, '3Q', max)
316 
317   }
318 
319   return(secondary_df)
320 }
```

```

321 })
322
323 output$secondary_table2 = renderTable({
324
325   if (variable2() == "none") {
326
327     return()
328
329   } else if (grepl(paste(factor_vars(), collapse = "|"), variable2())) {
330
331     return()
332
333   } else {
334
335     secondary_df = descriptive_statistics(table_df()) %>%
336       dplyr::select(mean, sd)
337
338   }
339
340   return(secondary_df)
341 }
342
343
344 ### DISTRIBUTION PLOT
345
346 ## TITLE
347 title_secondary_plot_change = reactive({
348
349   # factor_vars() <- c("room_type", "property_type", "room_type")
350
351   if (variable2() == "none") { return() }
352
353   if (grepl(paste(factor_vars(), collapse = "|"), variable2())) {
354
355     variable = stringr::str_match(variable2(), paste(factor_vars(),
356       collapse = "|")) %>%
357       c() %>% gsub("_", " ", .) %>% capwords()
358
359   } else {
360
361     variable = capwords(input$variable2)

```

```

361
362     }
363
364     variable %>%
365       paste("Distribution of the variable", ., "in the district", input$district, sep = " ")
366   })
367
368 observeEvent({c(input$view, input$variable2)},
369   {output$title_secondary_plot = renderText({title_secondary_
370     plot_change()})},
371   ignoreNULL = FALSE)
372
373 ## REACTIVE ELEMENTS
374
375 plot_df2 = reactive({
376
377   if (variable2() == "none") {
378
379     return()
380
381   } else if (grepl(paste(factor_vars(), collapse = "|"), variable2())) {
382
383     df = listings %>%
384       filter(district == input$district)
385
386   } else {
387
388     variable = var_table_plot2() %>% sym()
389
390     df = listings %>%
391       filter(district == input$district)
392
393   }
394
395   return(df)
396 }
397
398 ## PLOT
399

```

```

400 output$secondary_plot = renderPlot({
401
402   if (variable2() == "none") {
403
404     return()
405
406   } else if (grepl(paste(factor_vars(), collapse = "|"), variable2())) {
407
408     plot2 = distribution_plot(plot_df2(), var_table_plot2(), tilt_text_x =
409       TRUE)
410
411   } else {
412
413     plot2 = distribution_plot(plot_df2(), var_table_plot2(), hide_outliers
414       = TRUE)
415
416   }
417
418   return(plot2)
419
420 })

```

Listing 47: |price_corr_server.R|

```

1 ##########
2 # Price Correlation Server - file #####
3 # Author: Silvia Ventoruzzo #####
4 ##########
5
6 price_corr_server = function(input, output, session) {
7
8   ## TEXT
9
10  output$price_text = renderText({
11    "In this tab you can choose one or multiple variables and check their
12      correlation with price. Moreover,
13      a linear regression on price is run with the selected variable(s) as
14      regressors. You can also check the
15      coefficients and the residuals for this mode."
16  })

```

```

15
16 # For plot in tab3-panel1, check that at least one variable (input$variable3) is selected
17 # If not show an error message
18
19 ## REACTIVE ELEMENTS
20
21 price_correlation = reactive({
22
23   validate(
24     need(!is.null(input$variable3), "Please select variable(s)")
25   )
26
27   listings_price_correlation %>%
28     filter(variable %in% gsub(" ", "_", input$variable3))
29 })
30
31 ## PLOT
32
33 output$corr_plot = renderPlot({
34
35   correlation_plot(price_correlation())
36
37 })
38
39 }

```

Listing 48: |price_lm_server.R|

```

1 #####
2 # Price Server-file #####
3 # Author: Silvia Ventoruzzo #####
4 #####
5
6 price_lm_server = function(input, output, session) {
7
8   ## REACTIVE ELEMENTS
9
10  # For plot in tab3-panel1, check that at least one variable (input$variable3) is selected
11  # If not show an error message
12  price_lm_df = reactive({

```

```

13 validate(
14   need(!is.null(input$variable3), "Please select variable(s)")
15 )
16
17 listings_price %>%
18   dplyr::select(price, gsub(" ", "_", input$variable3))
19
20 })
21
22 price_lm = reactive({
23
24   lm(price ~ ., data = price_lm_df())
25
26 })
27
28 ## TEXT WITH R-SQUARED
29
30 output$price_lm_r2 = renderText({
31
32   summary(price_lm())$r.squared %>% round(4)
33
34 })
35
36 ## TABLE WITH COEFFICIENTS
37
38 output$price_lm_coefficients = renderTable({
39
40   coefficients = data.frame(
41     variable = c("(Coefficient)", gsub(" ", "_", input$variable3)),
42     coefficient = price_lm()$coefficients) %>%
43     tidyverse::drop_na() %>%
44     dplyr::mutate(num = 1:nrow(.)) %>%
45     dplyr::mutate_if(is.numeric, function(x) round(x, 4))
46
47   significance = data.frame(
48     significant = ifelse(summary(price_lm())$coefficients[,4] < 0.5,
49                           TRUE, FALSE)) %>%
50     dplyr::mutate(num = 1:nrow(.))
51
52   lm = coefficients %>%
53     dplyr::inner_join(significance, by = "num") %>%

```

```

54     dplyr::select(-num)
55
56     return(lm)
57
58   })
59
60   ## SUMMARY TABLE OF RESIDUALS
61
62   output$price_lm_residuals = renderTable({
63
64     lm = data.frame(
65       residuals = price_lm()$residuals
66     )
67
68     summary_stat_num(df = lm,
69                       var = "residuals")
70
71   })
72
73   ## FITTED VS ACTUAL PLOT
74
75   output$price_lm_resplot = renderPlot({
76
77     ggplot(price_lm(), aes(x = .fitted, y = .resid)) +
78       geom_point() +
79       geom_smooth(method = lm) +
80       geom_hline(yintercept=0, col="red", linetype="dashed") +
81       xlab("Fitted values") +
82       ylab("Residuals") +
83       ggtitle("Residual vs Fitted Plot") +
84       theme_bw()
85
86   })
87
88   ## QQ PLOT RESIDUALS
89
90   output$price_lm_qqplot = renderPlot({
91
92     gg_qqplot(price_lm()) +
93     theme_bw()
94

```

```

95     })
96
97 }
```

Listing 49: |clustering_server.R|

```

1 ##########
2 # Clustering Server - file #####
3 # Author: Silvia Ventoruzzo #####
4 #####
5
6 clustering_server = function(input, output, session) {
7
8   output$cluster_ui <- renderUI({
9
10   text1 = "In this tab you can cluster Airbnb properties in Berlin. The
11     process is the following:"
12
13   text2 = "Choose the number of the clusters according to the elbow-
14     method"
15
16   text2a = "Choose the number of clusters to test"
17
18   text2b = "Calculate for each number the percentage of the total variance
19     explained"
20
21   text2c = "Plot the total variance explained against the number of
22     clusters"
23
24   text2d = "Choose the number of clusters where adding another does not
25     add sufficient informatoin"
26
27   text3 = "Cluster the properties with the selected number of clusters"
28
29   text4 = "Plot the clusters"
30
31
32   intern = paste(text2, "<ul><li>", sep = "<br/>") %>%
33     paste(text2a, sep = "") %>%
34     paste(text2b, sep = "</li><li>") %>%
35     paste(text2c, sep = "</li><li>") %>%
36     paste(text2d, sep = "</li><li>") %>%
37     paste("", sep = "</li></ul>")
38
39
40   extern = paste(text1, "<ul><li>", sep = "<br/>") %>%
41     paste(intern, sep = "") %>%
42     paste(text3, sep = "</li><li>") %>%
43     paste(text4, sep = "</li><li>") %>%
44     paste("", sep = "</li></ul>")
```

```

32     html = HTML(extern)
33
34 })
35
36
37 ## Panel 1 (Find number of clusters)
38
39 output$ctest_text <- renderText({
40   "Before clustering the airbnb properties, we need to find out what is
41   the optimal number of clusters.
42   We use the elbow method, which means that from plot showing the
43   percentage of the total variance
44   explained for that number of clusters we choose the number before the
45   increase becomes too flat."
46 })
47
48 tot_var_explained <- eventReactive(input$ctest_start, {
49
50   clusters = seq(input$ctest[1], input$ctest[2], 1)
51
52   cluster_count = input$ctest[2] - input$ctest[1] + 1
53
54   tve = data.frame(clusters = clusters,
55                     tve      = rep(NA, cluster_count))
56
57   clk = list()
58
59   n = 1
60
61   for (k in seq(input$ctest[1], input$ctest[2], 1)) {
62
63     clk[[k-1]] = kmeans(listings_scaled, centers = k, iter.max = 20)
64     tve$tve[k-1] = 1 - clk[[k-1]]$tot.withinss / clk[[k-1]]$totss
65
66     updateProgressBar(session = session, id = "ctest_progress",
67                       value = (n / cluster_count * 100), total = 100,
68                       title = paste(n, "of", cluster_count))
69
70     n = n + 1
71   }
72
73
74   return(tve)

```

```

70 })
71
72
73 output$tve = renderPlot({
74
75   # Dependency on input$ctest_start
76   if (input$ctest_start == 0) return()
77
78   tot_var_explained() %>%
79     ggplot(aes(x = clusters, y = tve)) +
80     geom_line(color = "grey") +
81     geom_point(color = "red") +
82     scale_x_continuous(breaks = seq(input$ctest[1], input$ctest[2], 2))
83     +
84     theme_bw()
85 }
86
87 ## Panel 2 (Actual clustering)
88
89 output$cnum_ui <- renderUI({
90   text <- "Select the optimal number of clusters you found in the
91         previous tab and run the clustering
92         function. In this case we use k-means, which splits the data into the
93         given number of clusters
94         in such a way as to minimize the within-cluster sum of squares. For
95         detailed information on
96         this and other clustering algorithms you can read here:"
97   url <- a("Data Clustering: A Review",
98           href="http://delivery.acm.org/10.1145/340000/331504/p264-jain
99           .pdf?ip=141.20.217.40&id=331504&acc=PUBLIC&key=2
100          BA2C432AB83DA15%2E04C410D892D772C0%2E4D4702B0C3E38B35%2
101          E4D4702B0C3E38B35&__acm__=1551719946_9
102          d82c9a89fd8cd69cfa3d310ea27bc7f")
103
104   tagList(text, url)
105 }
106
107
108   # REACTIVE ELEMENTS
109
110   # k-means
111   k_means = eventReactive(input$cnum_start, {

```

```

103
104     kmeans(listings_scaled, centers = input$cnum, iter.max = 20)
105
106 })
107
108 # clusters
109 clusters <- eventReactive(input$cnum_start, {
110
111   k_means()$cluster %>% as.factor()
112
113 })
114
115 # Plot of cluster on coordinates
116 output$clustercoord = renderPlot({
117
118   # Dependency on input$ctest_start
119   if (input$cnum_start == 0) { return() }
120
121   ggplot() +
122     geom_sf(data = berlin_sf %>% dplyr::filter(view == "Districts"),
123             show.legend = FALSE, color = "black") +
124     coord_sf(datum = NA) +
125     geom_point(data = listings, aes(x = long, y = lat, color = clusters
126       (), alpha = 0.5) +
127       theme(plot.title = element_text(hjust = 0.5)) +
128       labs(color = "clusters") +
129       theme_void()
130
131 })
132
133 # Plot of clusters on Principal Components
134 output$clusterpc = renderPlot({
135
136   # Dependency on input$ctest_start
137   if (input$cnum_start == 0) { return() }
138
139   ggplot2::autoplot(listings_pc, data = k_means(), colour = "cluster") +
140     labs(color = "clusters") +
141     theme_bw()
142

```

```
143  
144 }
```

Listing 50: |app.R|

```
1 rm(list=ls(all = TRUE))  
2  
3 # Install and load needed packages  
4 needed_packages <- c("shiny",  
5                         "shinydashboard",  
6                         "leaflet",  
7                         "shinyWidgets",  
8                         "colorspace",  
9                         "htmltools",  
10                        "rstudioapi",  
11                        "devtools",  
12                        "lindia",  
13                        "tidyverse",  
14                        "Jmisc",  
15                        "stringr")  
16 for (package in needed_packages) {  
17   if (!require(package, character.only=TRUE)) {  
18     install.packages(package, character.only=TRUE)}  
19   library(package, character.only=TRUE)  
20 }  
21 devtools::install_github('andrewsali/shinycssloaders')  
22 library("shinycssloaders")  
23 rm("needed_packages", "package")  
24  
25 # Set working directory to the one where the file is located  
26 # setwd(dirname(sys.frame(1)$ofile)) # This works when sourcing  
27 setwd(dirname(rstudioapi::getActiveDocumentContext()$path)) # This works  
when running the code directly  
28  
29 # Load helper functions and other scripts  
30 load("workspace_for_shinyapp.RData")  
31 Jmisc::sourceAll(file.path(getwd(), "Helpers", fsep="/"))  
32 Jmisc::sourceAll(file.path(getwd(), "Server", fsep="/"))  
33 source("ui.R")  
34 source("server.R")  
35  
36 # Run app
```

```
37 shinyApp(ui      = ui,  
38           server = server)
```

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 15, 2019

Silvia Ventoruzzo