
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

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 "presense of car parks and fitness centers". But price determinants could be different in case of different types of accomodations, such as the ones offered on Airbnb. That's why Wang and Nicolau (2017) also derived the drivers of price for Airbnb listings from 33 cities.

What is observed is that the dimension as well as the type of the accomodation positively influences the price. However, the location has almost no impact on the price, except for the case of being inside the circulare 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 determinans 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 develop 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|**

```
# Load shapefiles
berlin = sf::st_read(file.path(getwd(), "Data",
                               "Berlin-Ortsteile-polygon.shp", fsep="/"))

# Object with the neightbourhoods (and respective district)
berlin_neighbourhood_sf = berlin %>%
    dplyr::rename(id    = Name,
```

```
                    group = BEZNAME) %>%
    dplyr::select(id, group, geometry) %>%
    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 `dyplr`.

**Listing 2: |berlin_districts_neighbourhoods.R|**

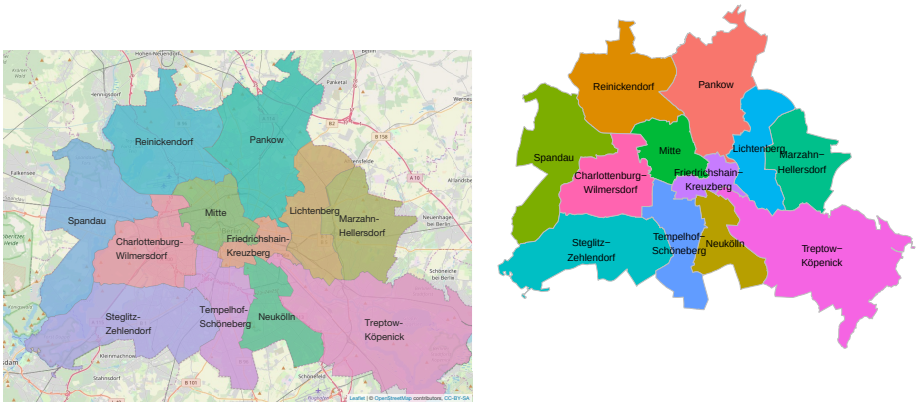
```
berlin_neighbourhood_singlebuckow_sf = berlin_neighbourhood_sf %>%
    dplyr::group_by(id, group) %>%
    dplyr::summarize(do_union = TRUE)
```

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|**

```
berlin_district_sf = berlin_neighbourhood_sf %>%
    dplyr::group_by(group) %>%
    dplyr::summarize(do_union = TRUE) %>%
    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.




**(a)** With `leaflet` **(b)** With `ggplot2`

**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.

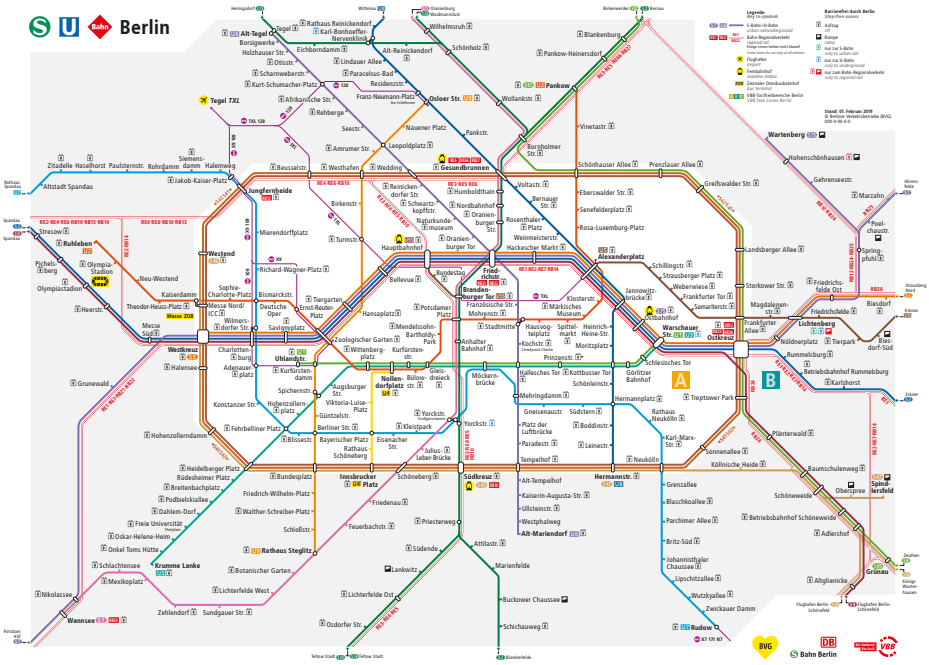



**Figure 2:** Network Map of Berlin Areas A and B (Source: Verkehrsverbund Berlin-Brandenburg (2019))

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|**

```
ringbahn_names_df = base::data.frame(
      id = c("Südkreuz", "Schöneberg", "Innsbrucker Platz", "Bundesplatz",
          "Heidelberger Platz", "Hohenzollerndamm", "Halensee", "Westkreuz",
          "Messe Nord/ICC", "Westend", "Jungfernheide", "Beusselstraße",
          "Westhafen", "Wedding", "Gesundbrunnen", "Schönhauser Allee",
          "Prenzlauer Allee", "Greifswalder Straße", "Landsberger Allee",
          "Storkower Straße", "Frankfurter Allee", "Ostkreuz",
          "Treptower Park", "Sonnenallee", "Neukölln", "Hermannstraße",
          "Tempelhof", "Südkreuz"),
      stringsAsFactors = FALSE) %>%
    tibble::rownames_to_column(var = "order") %>%
    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|**

```
berlin_vbb_A_sf = stations %>%
    dplyr::filter(fclass %like% "railway") %>%
    dplyr::rename(id = name) %>%
    dplyr::mutate(id = gsub("Berlin ", "", id),
                  id = gsub("Berlin-", "", id),
                  id = gsub(" *\\(.*?\\) *", "", id),
                  id = gsub("S ", "", id),
                  id = gsub("U ", "", id)) %>%
    points_midpoint() %>%
    dplyr::right_join(ringbahn_names_df, by = "id") %>%
    dplyr::arrange(order) %>%
    dplyr::select(long, lat) %>%
    base::as.matrix() %>%
    base::list() %>%
    sf::st_polygon() %>%
    sf::st_sfc() %>%
    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)
```

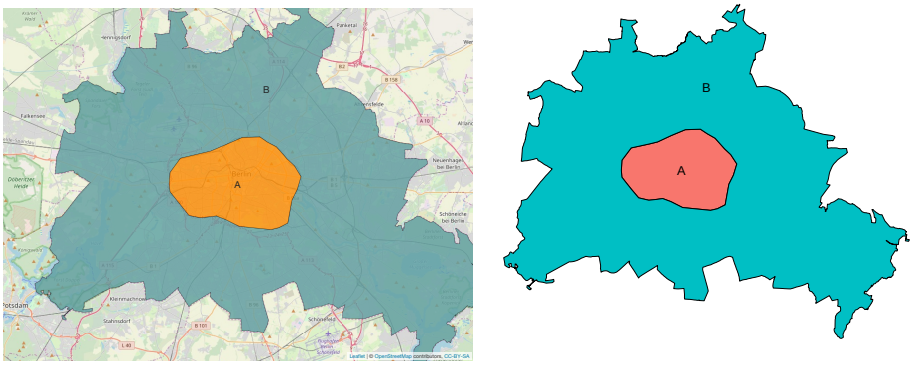
Finally, we bind the two objects by row and calculate their intersections thanks to the function `st_intersection` from the package `sf`. We then define the area names according to how many times the two previous polygons intersect:

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

**Listing 7: |berlin_vbb_zones.R|**

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

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




**(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|**

```
point_in_polygons <- function(points_df, polys_sf,
                              var_name, join_var = "id")  {
  # Create empty dataframe
  is_in <- data.frame(matrix(ncol = nrow(polys_sf), nrow = nrow(points_df)))
  is_in[,var_name] <- NA
  # Extract coordinates of the polygons
  coordinates <- as.data.frame(st_coordinates(polys_sf))
  # Extract names of the polygons
  name <- as.character(polys_sf$id)
  # For all polygons check if the points are inside of them
  for (k in 1:nrow(polys_sf)) {
    is_in[,k] <- sp::point.in.polygon(point.x = points_df$long,
                                      point.y = points_df$lat,
                                      pol.x   = coordinates$X
                                                [coordinates$L2 == k],
                                      pol.y   = coordinates$Y
                                                [coordinates$L2 == k])
    # Get the names of the polygons where the points are in one column
    is_in[,var_name][is_in[,k] == 1] <- name[k]
  }
  # Keep only summary column and add points' names
  is_in <- is_in %>%
    dplyr::select(var_name) %>%
    dplyr::mutate(id = points_df$id)
  # Add the summary column to the points dataframe
  points_df <- dplyr::full_join(points_df, is_in, by = join_var)
  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:

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

where $\phi$ corresponds to the latitude and $\psi$ 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|**

```
distance_count = function(main, reference, var_name, distance) {
    # Create variable names
    var_name_count = paste(var_name, "count", sep = "_")
    var_name_dist = paste(var_name, "dist", sep = "_")
    # Calculate distance for each listing to each station
    point_distance = geosphere::distm(x = main %>%
                                          dplyr::select(long, lat),
                                      y = reference %>%
                                          dplyr::select(long, lat),
                                      fun = distHaversine) %>%
        as.data.frame() %>%
        data.table::setnames(as.character(reference$id))
    # Calculate how many "reference" are within "distance"
    point_distance[,var_name_count] = rowSums(point_distance <= distance)
      %>%
                                        as.factor()

```

```
    # Calculate the distance to the nearest "reference"
    point_distance[,var_name_dist]  = apply(point_distance[,-ncol(point_
      distance)],
                                            MARGIN = 1,
                                            FUN = min) %>%
                                        round(0)
    # Insert this information into the main DF
    main = point_distance %>%
        dplyr::mutate(id = main$id) %>%
        dplyr::select(id, var_name_count, var_name_dist) %>%
        dplyr::right_join(main, by = "id")

    return(main)
}
```

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

**Listing 10: |data_preparation.R|**

```
listings_season_availability = listings_calendar %>%
  dplyr::group_by(id, year_season) %>%
  dplyr::summarize(count       = sum(available == TRUE),
                   season_days = n(),
                   proportion  = round(count/season_days*100, 4)) %>%
  dplyr::arrange(id, year_season) %>%
  dplyr::ungroup() %>%
  dplyr::mutate(season_av =  paste(year_season, "availability",
                                   sep = "_") %>%
                             tolower() %>%
                             factor(levels = paste(year_season,
                                                   "availability",
                                                   sep = "_") %>%
                                             tolower() %>%
                                             unique())) %>%
  dplyr::select(-count, -year_season, -season_days) %>%
  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 probabilites in our case.

**Listing 11: |descriptive_statistics.R|**

```
descriptive_statistics = function(df) {

  # Descriptive statistics for numeric variables
  if (unique(apply(df, 2, function(x) is.numeric(x)))) {

    summary = data.frame(
                variable = names(df),
                min      = apply(df, 2, min),
                `1Q`     = apply(df, 2, quantile, probs = 0.25),
                median   = apply(df, 2, median),
                `3Q`     = apply(df, 2, quantile, probs = 0.75),
```

```
                max       = apply(df, 2, max),
                iqr       = apply(df, 2, IQR),
                mean      = apply(df, 2, mean),
                sd        = apply(df, 2, sd),
        check.names = FALSE) %>%
      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|**

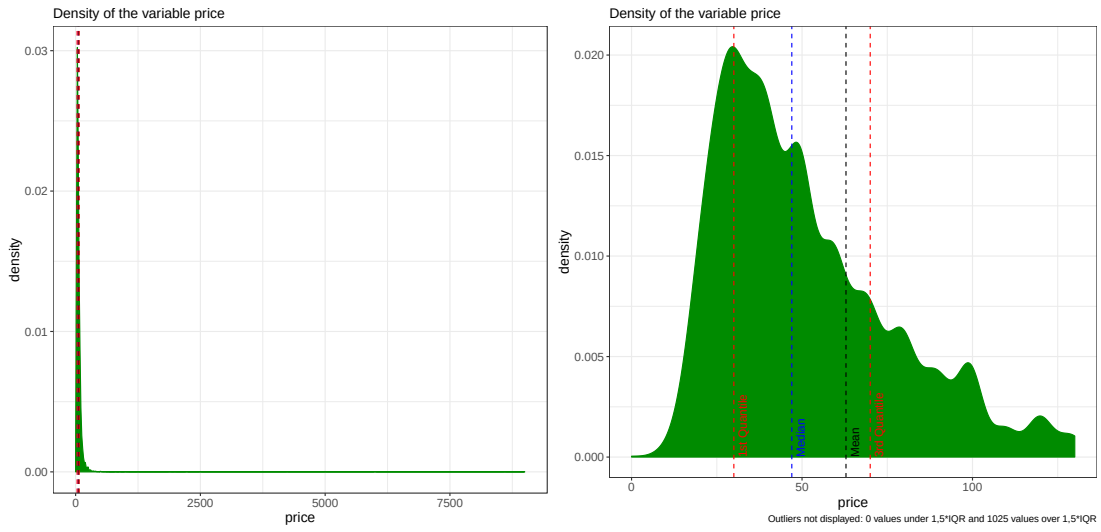
```
} else if (unique(apply(df, 2, function(x) is.character(x) | is.factor(x)
  | is.logical(x)))) {

  # Frequency
  frequency_list = apply(df, 2, table)

  frequency_df = data.frame(frequency = unlist(frequency_list)) %>%
    tibble::rownames_to_column(var = "var_fact")

  freq_var = colsplit(string = frequency_df$var_fact, pattern = "\\.",
    names = c("variable", "factor"))

  frequency = freq_var %>%
    cbind(frequency_df) %>%
    dplyr::select(-var_fact)

  # Proportion
  proportion_list = apply(df, 2, function(x) prop.table(table(x)))

  proportion_df = data.frame(proportion = unlist(proportion_list)) %>%
    tibble::rownames_to_column(var = "var_fact")
```

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

## 3.2 Distribution plots

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

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




**(a)** Complete **(b)** Without outliers

**Figure 4:** Distribution of the variable price distribution_plot.R

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

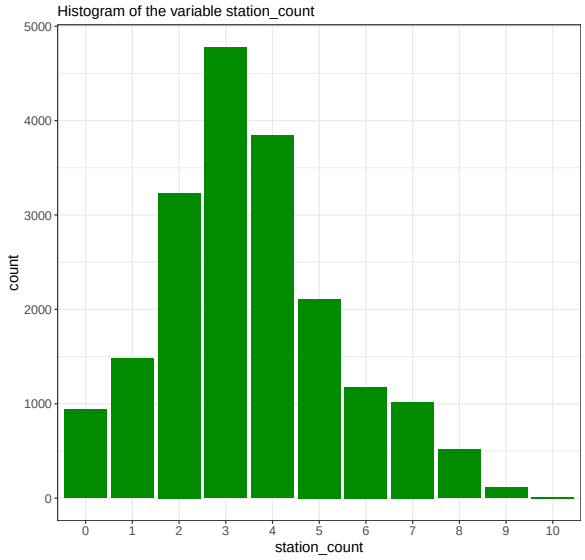



**Figure 5:** Distribution of the variable station_count distribution_plot.R

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

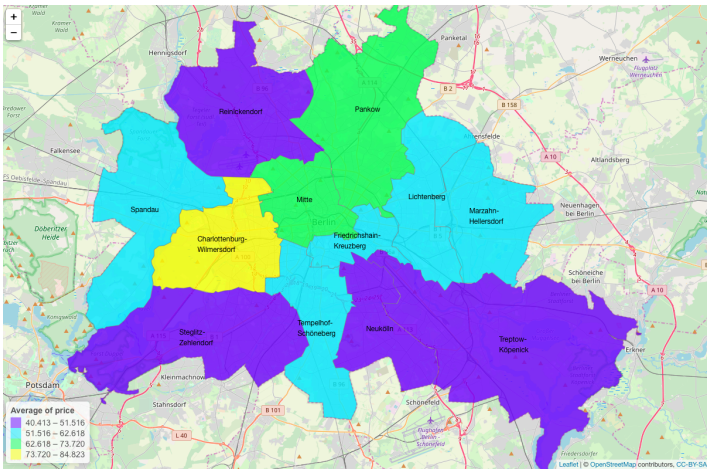



**Figure 6:** Distribution of the average of price across Berlin's districts var_avg_map.R

# 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|**

```
from_row_to_col = function(df, variable) {
    df = df %>%
      dplyr::mutate(yesno = 1) %>%
      dplyr::distinct() %>%
      tidyr::spread(variable, yesno, fill = 0)
    return(df)
}
```

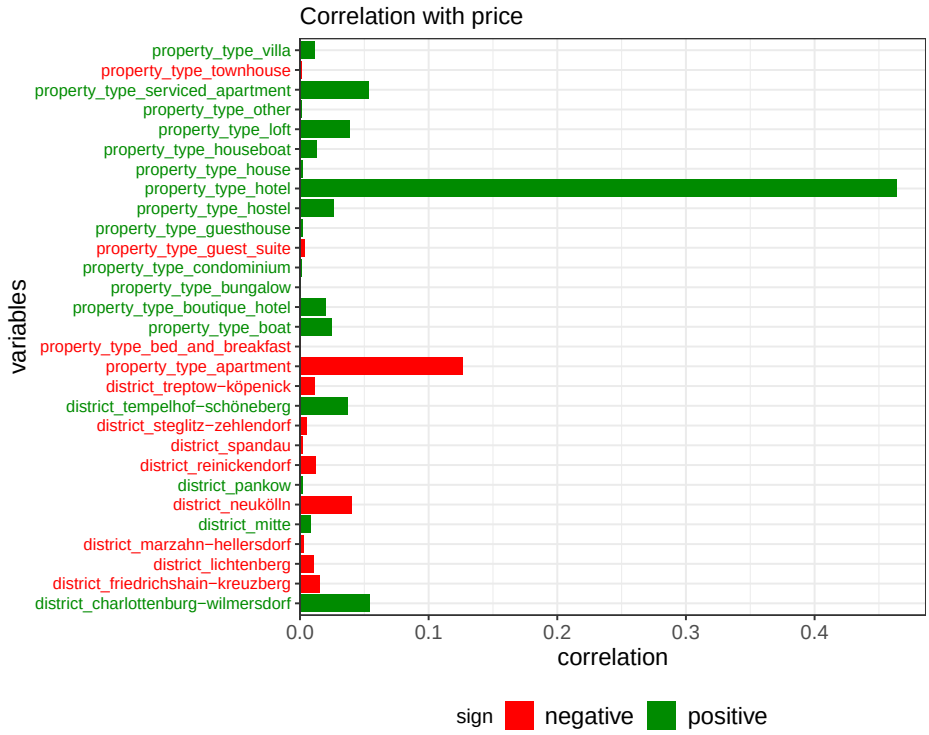



**Figure 7:** Sample of plot of correlation with price correlation_plot.R

## 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).

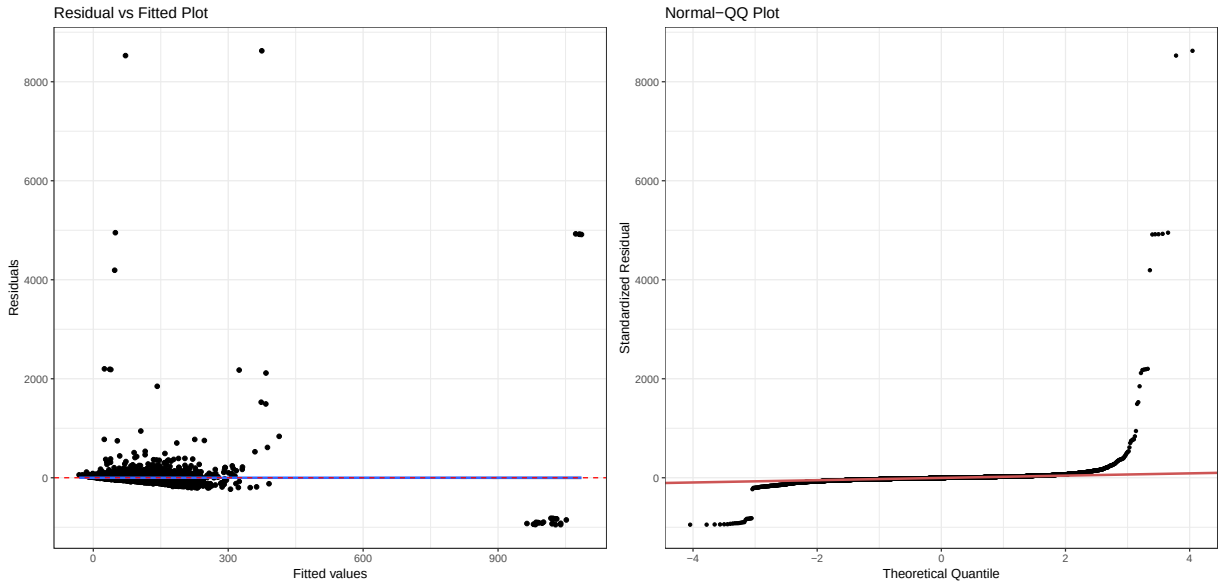
| variable | coefficient | significant |
|---|---|---|
| (Intercept) | 12.69 | FALSE |
| host_listings_count | -0.73 | TRUE |
| accommodates | 16.31 | TRUE |

**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




**(a)** Fitted vs. Actual Plot

**(b)** Residuals QQ Plot

**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 paritioning methods.

The basic ideas of this method were developed at the beginning of the second half of the $20^{th}$ 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)

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

   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 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|**

```
number_of_clusters = function(scaled_df, min = 2, max,
                              iter_max = 10, plot_breaks) {
  # Set seed for reproducibility
  set.seed(900114)
  # Values to test
  k_values = min:max
  # Empty dataframe for the total variance explained
  tve = data.frame(clusters = k_values,
                   tve      = rep(NA, length(k_values)))
  # Empty list for the kmeans objects
  clk = list()
  # Loop through the possible values of k
  for (k in k_values) {
    # Calculate k-means for each k
    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|**

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

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

**Listing 16: |number_of_clusters.R|**

```
  # Plot tve against k values
  plot = ggplot(data = tve, aes(x = clusters, y = tve)) +
    geom_line(color = "grey") +
    geom_point(color = "red") +
```

```
    scale_x_continuous(breaks = plot_breaks) +
    labs(x     = "number of clusters",
         y     = "% of tve",
         title = paste("Plot of total variance explained for k from", min, "
           to", max, sep = " ")) +
    theme_bw() +
    theme(axis.text.x = element_text(size = rel(1.2)),
          axis.text.y = element_text(size = rel(1.2)),
          axis.title.x = element_text(size = rel(1.2)),
          axis.title.y = element_text(size = rel(1.2)))

  return(plot)
}
```

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.

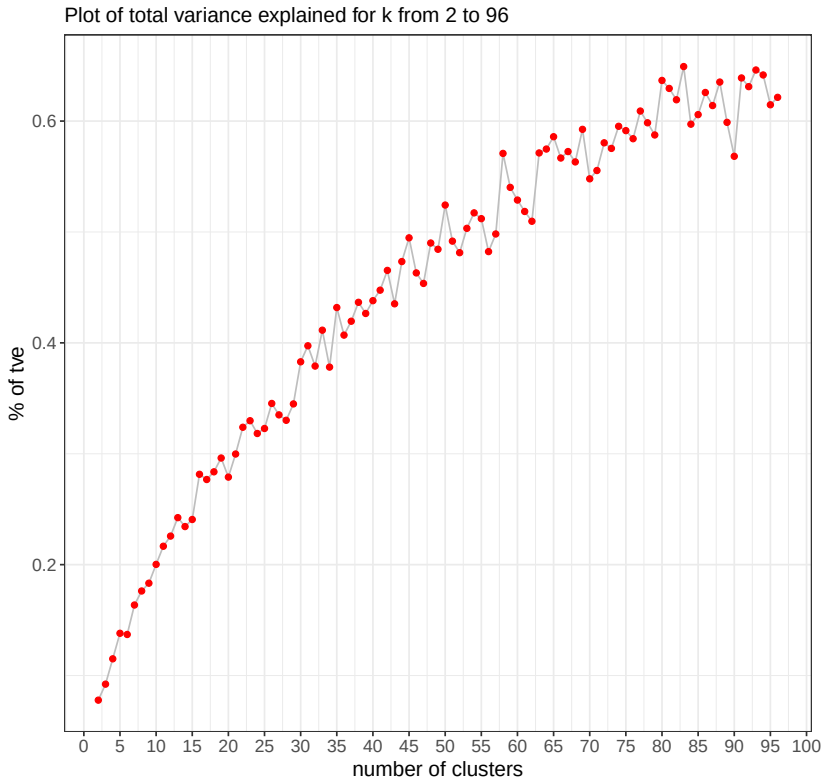
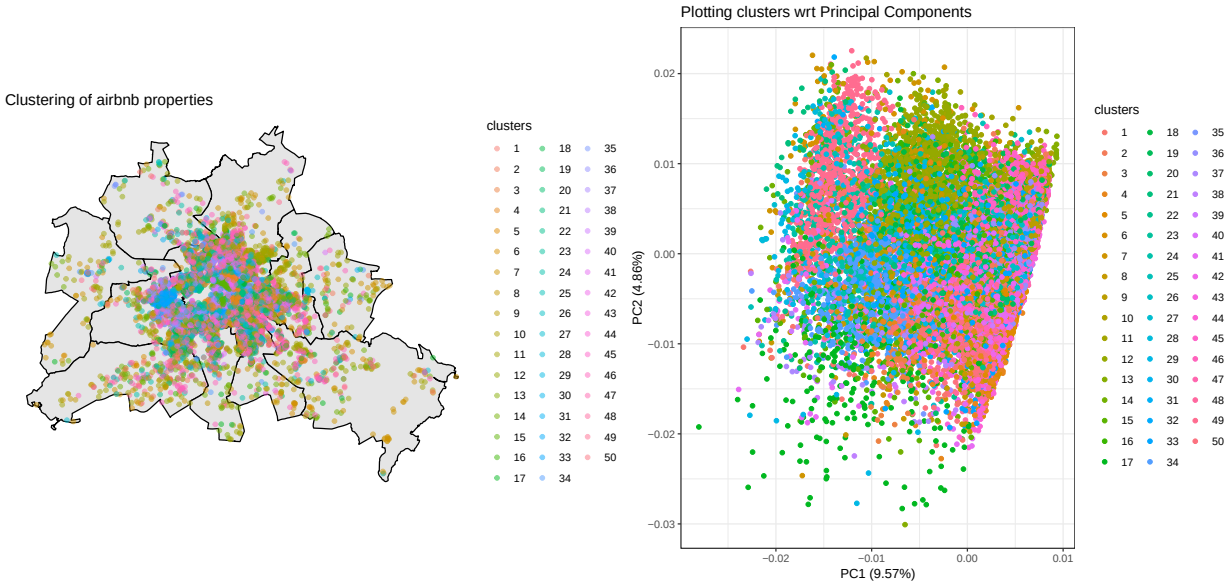



**Figure 9:** Plot of the total variance explained against different values of $k$ number_of_clusters.R

### 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 ouputs 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 interactiveness 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 Dijkestra 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 /1.3*bnroom* − *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|**

```r
# Load packages
needed_packages = c("dplyr", "data.table", "sf", "tibble")
for (package in needed_packages) {
    if (!require(package, character.only=TRUE))
      {install.packages(package, character.only=TRUE)}
    library(package, character.only=TRUE)
}
rm("needed_packages", "package")

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

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

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


# Load shapefiles
berlin = sf::st_read(file.path(getwd(), "Data",
                               "Berlin-Ortsteile-polygon.shp", fsep="/"))

# Object with the neightbourhoods (and respective district)
berlin_neighbourhood_sf = berlin %>%
    dplyr::rename(id    = Name,
                  group = BEZNAME) %>%
    dplyr::select(id, group, geometry) %>%
    dplyr::arrange(group)

# Buckow is composed of two separate parts, so we need to join them
berlin_neighbourhood_singlebuckow_sf = berlin_neighbourhood_sf %>%
    dplyr::group_by(id, group) %>%
    dplyr::summarize(do_union = TRUE)

# Object with the districts
berlin_district_sf = berlin_neighbourhood_sf %>%
```

```
    dplyr::group_by(group) %>%
    dplyr::summarize(do_union = TRUE) %>%
    dplyr::mutate(id = group)

# Create dataframes with the names for plotting
# Neighbourhoods
berlin_neighbourhoods_names = berlin_neighbourhood_sf %>%
    sf::st_centroid() %>%
    sf::st_coordinates() %>%
    base::as.data.frame() %>%
    dplyr::rename(long = X,
                  lat  = Y) %>%
    dplyr::mutate(id    = berlin_neighbourhood_sf$id,
                  group = berlin_neighbourhood_sf$group,
                  name  = gsub("-", "-<br>", berlin_neighbourhood_sf$id))

# Districts
berlin_districts_names = berlin_district_sf %>%
    sf::st_centroid() %>%
    sf::st_coordinates() %>%
    base::as.data.frame() %>%
    dplyr::rename(long = X,
                  lat  = Y) %>%
    dplyr::mutate(id    = berlin_district_sf$id,
                  group = berlin_district_sf$group,
                  name  = gsub("-", "-<br>", berlin_district_sf$id))

# Remove not needed data
rm("berlin")
```

**Listing 18: |berlin_districts_neighbourhoods_maps.R|**

```
# Load packages
needed_packages = c("leaflet", "ggplot2", "htmltools", "mapview")
for (package in needed_packages) {
  if (!require(package, character.only=TRUE))
      {install.packages(package, character.only=TRUE)}
  library(package, character.only=TRUE)
}
rm("needed_packages")

# 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() +
```

```
    geom_sf(data = berlin_district_sf,
            show.legend = FALSE, color = "grey") +
    coord_sf(datum = NA) +
    aes(fill = district_colors) +
    geom_text(data = berlin_districts_names,
              aes(x = long, y = lat,
                  label = gsub("<br>", "\n", name),
                  hjust = "center")) +
    theme_classic() +
    theme(plot.title   = element_text(hjust = 0.5),
          axis.title.x = element_blank(),
          axis.title.y = element_blank())

# dev.copy2pdf(file = "berlin_district_ggplot.pdf")
# dev.copy2pdf(file = "../SeminarPaper/berlin_district_ggplot.pdf")
```

**Listing 19: |berlin_vbb_zones.R|**

```
# Load packages
needed_packages = c("dplyr", "data.table", "sf", "tibble")
for (package in needed_packages) {
    if (!require(package, character.only=TRUE))
        {install.packages(package, character.only=TRUE)}
    library(package, character.only=TRUE)
}
rm("needed_packages", "package")

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

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

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


# Load helpers
source(file.path(getwd(), "Helpers", "points_midpoint.R", fsep="/"))

# Load shapefiles
berlin = sf::st_read(file.path(getwd(), "Data",
                               "Berlin-Ortsteile-polygon.shp", fsep="/"))
```

```r
stations = sf::st_read(file.path(getwd(), "Data",
                                 "gis_osm_transport_free_1.shp", fsep="/"))

# Create dataframe with names of stations on the Ringbahn (delimits Area A)
ringbahn_names_df = base::data.frame(
      id = c("Südkreuz", "Schöneberg", "Innsbrucker Platz", "Bundesplatz",
          "Heidelberger Platz", "Hohenzollerndamm", "Halensee", "Westkreuz",
          "Messe Nord/ICC", "Westend", "Jungfernheide", "Beusselstraße",
          "Westhafen", "Wedding", "Gesundbrunnen", "Schönhauser Allee",
          "Prenzlauer Allee", "Greifswalder Straße", "Landsberger Allee",
          "Storkower Straße", "Frankfurter Allee", "Ostkreuz",
          "Treptower Park", "Sonnenallee", "Neukölln", "Hermannstraße",
          "Tempelhof", "Südkreuz"),
      stringsAsFactors = FALSE) %>%
    tibble::rownames_to_column(var = "order") %>%
    dplyr::mutate(order = as.numeric(order))

# Create sf object of Area A
berlin_vbb_A_sf = stations %>%
    dplyr::filter(fclass %like% "railway") %>%
    dplyr::rename(id = name) %>%
    dplyr::mutate(id = gsub("Berlin ", "", id),
                  id = gsub("Berlin-", "", id),
                  id = gsub(" *\\(.*?\\) *", "", id),
                  id = gsub("S ", "", id),
                  id = gsub("U ", "", id)) %>%
    points_midpoint() %>%
    dplyr::right_join(ringbahn_names_df, by = "id") %>%
    dplyr::arrange(order) %>%
    dplyr::select(long, lat) %>%
    base::as.matrix() %>%
    base::list() %>%
    sf::st_polygon() %>%
    sf::st_sfc() %>%
    sf::st_sf(crs = sf::st_crs(berlin))

# Create sf object of entire Berlin
berlin_sf = berlin %>%
    dplyr::summarize(do_union = TRUE)

# Bind and intersect to create sf object with both VBB Zones (A and B)
```

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

# Create dataframes with coordinates where to show zones' names on map
berlin_vbb_A_names = berlin_vbb_A_sf %>%
    sf::st_centroid() %>%
    sf::st_coordinates() %>%
    base::as.data.frame() %>%
    dplyr::rename(long = X,
                  lat  = Y) %>%
    dplyr::mutate(id = "A")
berlin_vbb_B_names = berlin_sf %>%
    sf::st_bbox() %>%
    base::as.matrix() %>%
    base::t() %>%
    as.data.frame() %>%
    dplyr::transmute(long = (xmax + xmin)/2,
                     lat  = (3*ymax + ymin)/4) %>%
    dplyr::mutate(id = "B")
berlin_vbb_AB_names = berlin_vbb_A_names %>%
    rbind(berlin_vbb_B_names) %>%
    dplyr::mutate(name = gsub("-", "-<br>", id))

# Remove not needed data
rm("berlin", "berlin_sf", "ringbahn_names_df", "stations",
   "berlin_vbb_A_sf", "berlin_vbb_A_names", "berlin_vbb_B_names")
rm(list=lsf.str()) # All functions
```

**Listing 20: |berlin_vbb_zones_maps.R|**

```
# Load packages
needed_packages = c("leaflet", "ggplot2", "htmltools", "mapview")
for (package in needed_packages) {
  if (!require(package, character.only=TRUE))
      {install.packages(package, character.only=TRUE)}
  library(package, character.only=TRUE)
}
rm("needed_packages")
```

```r

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

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

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


# Load file where the sf object is created
source("berlin_vbb_zones.R", chdir = TRUE)

# Leaflet map
berlin_vbb_zones_leaflet = leaflet() %>%
    addTiles() %>%
    addPolygons(data = berlin_vbb_AB_sf,
                weight = 1, smoothFactor = 1, fillOpacity = 0.8,
                fillColor = c("cadetblue", "darkorange"),
                color = "black") %>%
    addLabelOnlyMarkers(data = berlin_vbb_AB_names,
                        lng = ~long, lat = ~lat,
                        label = ~lapply(name, htmltools::HTML),
                        labelOptions = labelOptions(noHide = TRUE,
                                                    direction = 'center',
                                                    textOnly = TRUE,
                                                    textsize = "20px")) %>%
    setView(lng  = berlin_vbb_AB_names$long
                        [berlin_vbb_AB_names$id == "A"],
            lat  = berlin_vbb_AB_names$lat
                        [berlin_vbb_AB_names$id == "A"],
            zoom = 11)
berlin_vbb_zones_leaflet

# mapview::mapshot(berlin_vbb_zones_leaflet,
#                  file = "berlin_vbb_zones_leaflet.pdf",
#                  remove_controls = c("zoomControl", "layersControl",
#                                      "homeButton","scaleBar"))
# mapview::mapshot(berlin_vbb_zones_leaflet,
#                  file = "../SeminarPaper/berlin_vbb_zones_leaflet.pdf",
#                  remove_controls = c("zoomControl", "layersControl",
```

```
#                                          "homeButton","scaleBar"))


# ggplot map
ggplot() +
    geom_sf(data = berlin_vbb_AB_sf, show.legend = FALSE, color = "black") +
    coord_sf(datum = NA) +
    aes(fill = c("darkorange", "cadetblue")) +
    geom_text(data = berlin_vbb_AB_names,
              aes(x = long, y = lat, label = id, hjust = "center"),
              size = 5) +
    theme_classic() +
    theme(plot.title   = element_text(hjust = 0.5),
          axis.title.x = element_blank(),
          axis.title.y = element_blank())

# dev.copy2pdf(file = "berlin_vbb_zones_ggplot.pdf")
# dev.copy2pdf(file = "../SeminarPaper/berlin_vbb_zones_ggplot.pdf")
```

**Listing 21: |data_preparation.R|**

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

# Install and load needed packages
needed_packages = c("tidyverse",
                    "geosphere",
                    "readr",
                    "rstudioapi",
                    "Jmisc",
                    "sp",
                    "sf")
for (package in needed_packages) {
  if (!require(package, character.only=TRUE)) {
    install.packages(package, character.only=TRUE)}
  library(package, character.only=TRUE)
}
rm("needed_packages", "package")

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

  # This works when run directly
  # 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 =
                       TRUE)
31  Jmisc::sourceAll(file.path(getwd(), "Helpers", fsep="/"))
32
33  # Load shapefile
34  stations = sf::st_read(file.path(getwd(), "Data", "spatial_data",
35                                   "gis_osm_transport_free_1.shp", fsep="/"))
36
37  # Print code development
38  print("Spatial data loaded.")
39
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
```

```r
apply(listings, 2, function(x) any(is.na(x)))

# host_is_superhost, # host_has_profile_pic, host_identity_verified
listings = listings %>%
    dplyr::mutate(host_is_superhost      = ifelse(is.na(host_is_superhost),
                                                  FALSE, host_is_superhost),
                  host_has_profile_pic   = ifelse(is.na(host_has_profile_pic),
                                                  FALSE, host_has_profile_pic),
                  host_identity_verified = ifelse(is.na(host_identity_verified),
                                                  FALSE, host_identity_verified))

# review_scores_rating
# We insert the average of the review_scores_rating
listings %>%
    dplyr::summarize(mean = mean(review_scores_rating, na.rm = TRUE), # 77
                     median = median(review_scores_rating, na.rm = TRUE), # 95
                     mode = getmode(review_scores_rating)) %>% # 100
    round(0)
# mean might be influenced by outliers, being it so different from median and mode, and mode is too high, so I will substitute the missing values with the median
listings = listings %>%
    dplyr::mutate(review_scores_rating = ifelse(is.na(review_scores_rating),
                                                median(review_scores_rating, na.rm = TRUE),
                                                review_scores_rating))
# bedrooms and beds
# Being the amount of missing values in these features relative small, we can derive their value from the other variables:
listings = listings %>%
  # If beds is NA, but bedrooms has a valid value, we set beds with the number of bedrooms
  dplyr::mutate(beds     = ifelse(is.na(beds) & !is.na(bedrooms),
                                  bedrooms, beds),
                # If bedrooms is NA, but beds has a valid value, we set bedrooms with the number of beds
```

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

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

# Listings selection
# Keep only listings with at least one customer review, as per Wang and
  Nicolau (2017)
listings %>%
  dplyr::filter(number_of_reviews == 0) %>%
  dplyr::summarize(count = n())
listings = listings %>%
  dplyr::filter(number_of_reviews > 0)

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

## New variables

# Areas: Not doing this with berlin_sf because the polys_sf need to have
  geometry sfc_POLYGON
# district
listings = point_in_polygons(points_df = listings,
                             polys_sf  = berlin_district_sf,
                             var_name  = "district")
# vbb_zone
listings = point_in_polygons(points_df = listings,
                             polys_sf  = berlin_vbb_AB_sf,
                             var_name  = "vbb_zone")
# neighbourhood
listings = point_in_polygons(points_df = listings,
                             polys_sf = berlin_neighbourhood_sf,
                             var_name = "neighbourhood")
```

```
# For the sake of consistency, we will delete these listings
listings = listings %>%
  filter(!is.na(district) & !is.na(vbb_zone))

# Stations
railway_stations_df = stations %>%
  dplyr::filter(fclass %like% "railway") %>%
  dplyr::rename(id      = name) %>%
  dplyr::mutate(id      = gsub("Berlin ", "", id),
                id      = gsub("Berlin-", "", id),
                id      = gsub(" *\\(.*?\\) *", "", id),
                s_bahn = ifelse(startsWith(id, "S "), TRUE, FALSE),
                u_bahn = ifelse(startsWith(id, "U "), TRUE, FALSE),
                id      = gsub("S ", "", id),
                id      = gsub("U ", "", id)) %>%
  points_midpoint()
railway_stations_df = point_in_polygons(points_df = railway_stations_df,
                                        polys_sf  = berlin_district_sf,
                                        var_name  = "district")
railway_stations_df = railway_stations_df %>%
  filter(!is.na(district))
listings = distance_count(main = listings, reference = railway_stations_df,
                          var_name = "station", distance = 1000)

# (Top 10) attractions
attractions_df = data.frame(
  id   = c("Reichstag", "Brandenburger Tor", "Fernsehturm",
           "Gendarmenmarkt", "Berliner Dom", "Kurfürstendamm",
           "Schloss Charlottenburg", "Museuminsel",
           "Gedenkstätte Berliner Mauer", "Potsdamer Platz"),
  lat  = c(52.518611, 52.516389, 52.520803, 52.513333, 52.519167,
           52.500833, 52.521111, 52.520556, 52.535, 52.509444),
  long = c(13.376111, 13.377778, 13.40945, 13.393056, 13.401111,
           13.312778, 13.295833, 13.397222, 13.389722, 13.375833))
attractions_df = point_in_polygons(points_df = attractions_df,
                                   polys_sf  = berlin_district_sf,
                                   var_name  = "district")
listings = distance_count(main = listings, reference = attractions_df,
                          var_name = "attraction", distance = 2000)

# season availability
```

```
listings_calendar = listings_calendar %>%
  dplyr::rename(id = listing_id) %>%
  dplyr::mutate(date         = as.Date(date),
                year         = lubridate::year(date),
                month        = lubridate::month(date, label = TRUE),
                day          = lubridate::day(date),
                season       = ifelse((month == "Mar" & day >= 21) |
                                        (month == "Apr") |
                                        (month == "May") |
                                        (month == "Jun" & day < 21), "Spring",
                                  ifelse((month == "Jun" & day >= 21) |
                                         (month == "Jul") |
                                        (month == "Aug") |
                                         (month == "Sep" & day < 21), "Summer",
                                  ifelse((month == "Sep" & day >= 21) |
                                         (month == "Oct") |
                                         (month == "Nov") |
                                         (month == "Dec" & day < 21), "Fall",
                                                                      "Winter"))) %>%
                                     factor(levels = c("Spring", "Summer",
                                                       "Fall", "Winter")),
                year_season = paste(tolower(season), year, sep = "_")) %>%
  dplyr::ungroup() %>%
  dplyr::select(-price)

# Availability per year_season
listings_season_availability = listings_calendar %>%
  dplyr::group_by(id, year_season) %>%
  dplyr::summarize(count       = sum(available == TRUE),
                   season_days = n(),
                   proportion  = round(count/season_days*100, 4)) %>%
  dplyr::arrange(id, year_season) %>%
  dplyr::ungroup() %>%
  dplyr::mutate(season_av =  paste(year_season, "availability",
                                   sep = "_") %>%
                             tolower() %>%
                             factor(levels = paste(year_season,
                                                   "availability",
                                                   sep = "_") %>%
                                             tolower() %>%
                                             unique())) %>%
```

```
  dplyr::select(-count, -year_season, -season_days) %>%
  tidyr::spread(season_av, proportion)

# Print code development
print("New variables created.")

# Join to the listings
listings = listings %>%
  left_join(listings_season_availability, by = "id") %>%
  replace(is.na(.), 0)

# Remove unnecessary objects
rm("listings_calendar", "listings_season_availability", "stations",
   "berlin_neighbourhood_sf", "listings_detailed", "listings_summarized")
rm(list=lsf.str()) # All functions
```

**Listing 22: |exploratory_data_analysis.R|**

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

# Install and load needed packages
needed_packages = c("tidyverse",
                    "rstudioapi",
                    "Jmisc",
                    "reshape2",
                    "leaflet",
                    "mapview",
                    "xtable")
for (package in needed_packages) {
    if (!require(package, character.only=TRUE)) {install.packages(package,
      character.only=TRUE)}
    library(package, character.only=TRUE)
}
rm("needed_packages", "package")

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

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

  # This works when sourced
  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,
```

```
                    -vbb_zone, -district, -neighbourhood) %>%
    dplyr::select_if(function(x) is.character(x) | is.factor(x)
                                                 | is.logical(x)) %>%
    names()

# For all Berlin
listings_summary = summarize_df(df = listings,
                                vars_mean = num_var,
                                vars_count = fact_var)

# For each VBB Zone
listings_vbb_summary = summarize_df(df = listings,
                                    wrt = "vbb_zone",
                                    vars_mean = num_var,
                                    vars_count = fact_var)

listings_vbb_summary = listings_vbb_summary %>%
    rename(id    = vbb_zone) %>%
    mutate(view  = "VBB Zones",
           group = id)

# For each district
listings_district_summary = summarize_df(df = listings,
                                         wrt = "district",
                                         vars_mean = num_var,
                                         vars_count = fact_var)
listings_district_summary = listings_district_summary %>%
    rename(id    = district) %>%
    mutate(view  = "Districts",
           group = id)

# For each neighbourhood
listings_neighbourhood_summary = summarize_df(df = listings,
                                              wrt = "neighbourhood",
                                              vars_mean = num_var,
                                              vars_count = fact_var)
listings_neighbourhood_summary = berlin_neighbourhoods_names %>%
    dplyr::select(id, group) %>%
    dplyr::left_join(listings_neighbourhood_summary,
                     by = c("id" = "neighbourhood")) %>%
    dplyr::mutate(view = "Neighbourhoods") %>%
```

```
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.
    pdf",
139 #                  remove_controls = c("zoomControl", "layersControl", "
    homeButton","scaleBar"))
140
141 print("Exploratory Data Analysis done.")
142
143 # Remove unnecessary objects
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") %>%
```

```
  recode_all("station_count") %>% from_row_to_col("station_count") %>%
  recode_all("attraction_count") %>% from_row_to_col("attraction_count") %>%
  recode_all("instant_bookable") %>% from_row_to_col("instant_bookable") %>%
  recode_all("host_is_superhost") %>%
  from_row_to_col("host_is_superhost") %>%
  recode_all("host_has_profile_pic") %>%
  from_row_to_col("host_has_profile_pic") %>%
  recode_all("host_identity_verified") %>%
  from_row_to_col("host_identity_verified") %>%
  recode_all("district") %>% from_row_to_col("district") %>%
  recode_all("vbb_zone") %>% from_row_to_col("vbb_zone") %>%
  dplyr::select(-id, -lat, -long, -listing_url, -neighbourhood)

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

print("Price correlation calculated.")

# Plot correlation
# Example corrplot
# listings %>%
#   dplyr::select(price, accommodates, bedrooms, beds,
#                 minimum_nights, station_dist) %>%
#   cor() %>%
#   corrplot(method = "circle", type = "upper")
# dev.copy2pdf(file = "./SeminarPaper/corrplot.pdf")
# dev.off()

# Correlation with price plot
correlation_plot(corr_df   = listings_price_correlation)

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

## Linear regression

```

```r
# Fit linear model
listings_price_lm = lm(price ~ ., data = listings_price)

# Extract coefficients and p-values
listings_price_lm_results = data.frame(
  variable        = names(listings_price_lm$coefficients
                          [!is.na(listings_price_lm$coefficients)]),
  coefficient     = listings_price_lm$coefficients
                          [!is.na(listings_price_lm$coefficients)],
  significant     = ifelse(summary(listings_price_lm)
                          $coefficients[,4] < 0.5, TRUE, FALSE),
  model_r_squared = summary(listings_price_lm)$r.squared) %>%
  dplyr::mutate_if(is.numeric, function(x) round(x, 4))

# listings_price_lm_results %>%
#     dplyr::filter(variable %in% c("(Intercept)", "accommodates", "host_
  listings_count")) %>%
#     dplyr::select(-model_r_squared) %>%
#     xtable::xtable() %>%
#     print(include.rownames = FALSE)

# Descriptive statistics of residuals
listings_price_lm_residuals = data.frame(
  residuals = listings_price_lm$residuals
  )

# descriptive_statistics(listings_price_lm_residuals) %>%
#   dplyr::select(-variable) %>%
#   xtable::xtable() %>%
#   print(include.rownames = FALSE)

# Fitted vs actual plot
ggplot(listings_price_lm, aes(x = .fitted, y = .resid)) +
  geom_point() +
  geom_smooth(method = lm) +
  geom_hline(yintercept=0, col="red", linetype="dashed") +
  xlab("Fitted values") +
  ylab("Residuals") +
  ggtitle("Residual vs Fitted Plot") +
  theme_bw()
# dev.copy2pdf(file = "./SeminarPaper/fittedactual.pdf")
```

```
# dev.off()

# Residuals QQ Plot
lindia::gg_qqplot(listings_price_lm) +
  theme_bw()
# dev.copy2pdf(file = "./SeminarPaper/qqplot.pdf")
# dev.off()

print("Linear regression on price calculated.")

# Remove objects not further needed
rm("listings_price_lm", "listings_price_lm_results")
rm(list=lsf.str()) # All functions
```

**Listing 24: |clustering.R|**

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

# Install and load needed packages
needed_packages = c("tidyverse",
                    "rstudioapi",
                    "Jmisc",
                    "ggfortify")
for (package in needed_packages) {
  if (!require(package, character.only=TRUE)) {
    install.packages(package, character.only=TRUE)}
  library(package, character.only=TRUE)
}
rm("needed_packages", "package")

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

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

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

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

```

```
## Calculate number of clusters

# Scale dataframe
listings_scaled = listings %>%
  recode_all("room_type") %>% from_row_to_col("room_type") %>%
  recode_all("property_type") %>% from_row_to_col("property_type") %>%
  recode_all("cancellation_policy") %>%
  from_row_to_col("cancellation_policy") %>%
  recode_all("security_deposit") %>% from_row_to_col("security_deposit") %>%
  recode_all("cleaning_fee") %>% from_row_to_col("cleaning_fee") %>%
  recode_all("station_count") %>% from_row_to_col("station_count") %>%
  recode_all("attraction_count") %>% from_row_to_col("attraction_count") %>%
  recode_all("instant_bookable") %>% from_row_to_col("instant_bookable") %>%
  recode_all("host_is_superhost") %>%
  from_row_to_col("host_is_superhost") %>%
  recode_all("host_has_profile_pic") %>%
  from_row_to_col("host_has_profile_pic") %>%
  recode_all("host_identity_verified") %>%
  from_row_to_col("host_identity_verified") %>%
  dplyr::select(-id, -lat, -long, -listing_url,
                -district, -vbb_zone, -neighbourhood) %>%
  scale()

# Total variance explained
number_of_clusters(scaled_df   = listings_scaled,
                   max         = 96,
                   iter_max    = 20,
                   plot_breaks = seq(0, 100, by = 5))

# dev.copy2pdf(file = "./SeminarPaper/numclusters.pdf")
# dev.off()

## Clustering
n_clusters = 50
listings_kmeans = kmeans(listings_scaled,
                         centers = n_clusters, iter.max = 20)

listings_clusters = listings_kmeans$cluster %>% as.factor()

# Plot clusters in map
ggplot() +
```

```
    geom_sf(data = berlin_district_sf,
            show.legend = FALSE, color = "black") +
    coord_sf(datum = NA) +
    geom_point(data = listings,
               aes(x = long, y = lat, color = listings_clusters),
               alpha = 0.5) +
    theme(plot.title = element_text(hjust = 0.5)) +
    ggtitle("Clustering of airbnb properties") +
    labs(color = "clusters") +
    theme_void()

# dev.copy2pdf(file = "./SeminarPaper/cluster_map.pdf")
# dev.off()

# Principal components
listings_pc = prcomp(x = listings_scaled, center = FALSE, scale = FALSE)

ggplot2::autoplot(listings_pc, data = listings_kmeans, colour = "cluster") +
    ggtitle("Plotting clusters wrt Principal Components") +
    labs(color = "clusters") +
    theme_bw()

# dev.copy2pdf(file = "./SeminarPaper/cluster_plot.pdf")
# dev.off()

# Remove objects not further needed
rm("n_clusters")
rm(list=lsf.str()) # All functions
```

**Listing 25: |shiny_preparation.R|**

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

# Install and load needed packages
needed_packages = c("tidyverse",
                    "rstudioapi",
                    "sf")
for (package in needed_packages) {
  if (!require(package, character.only=TRUE)) {
    install.packages(package, character.only=TRUE)}
  library(package, character.only=TRUE)
}
```

```
rm("needed_packages", "package")

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

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

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

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

# Calculate the center of Berlin polygon
berlin_center = berlin_neighbourhood_singlebuckow_sf %>%
  dplyr::group_by() %>%
  dplyr::summarize(do_union = TRUE) %>%
  sf::st_centroid() %>%
  sf::st_coordinates() %>%
  base::as.data.frame() %>%
  dplyr::rename(long = X,
                lat  = Y) %>%
  dplyr::select(lat, long)

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

# Join sf objects
berlin_district_sf = berlin_district_sf %>%
  dplyr::mutate(view = "Districts")
berlin_neighbourhood_singlebuckow_sf =
  berlin_neighbourhood_singlebuckow_sf %>%
  dplyr::mutate(view = "Neighbourhoods")
berlin_vbb_AB_sf = berlin_vbb_AB_sf %>%
  dplyr::mutate(view  = "VBB Zones",
                group = id)
berlin_sf = rbind(berlin_district_sf,
                  berlin_neighbourhood_singlebuckow_sf,
                  berlin_vbb_AB_sf)

```

```r
# Join names dataframes
berlin_districts_names = berlin_districts_names %>%
  dplyr::mutate(view = "Districts")
berlin_neighbourhoods_names = berlin_neighbourhoods_names %>%
  dplyr::mutate(view = "Neighbourhoods")
berlin_vbb_AB_names = berlin_vbb_AB_names %>%
  dplyr::mutate(view  = "VBB Zones",
                group = id)
berlin_names = rbind(berlin_districts_names,
                     berlin_neighbourhoods_names,
                     berlin_vbb_AB_names)

# Recode and transform from row to columns factor variables
listings_recoded = listings %>%
  recode_all("room_type") %>% from_row_to_col("room_type") %>%
  recode_all("property_type") %>% from_row_to_col("property_type") %>%
  recode_all("cancellation_policy") %>%
  from_row_to_col("cancellation_policy")

# Remove objects not further needed
rm("listings_district_summary", "listings_vbb_summary",
   "listings_neighbourhood_summary", "berlin_district_sf",
   "berlin_neighbourhood_singlebuckow_sf", "berlin_vbb_AB_sf",
   "berlin_districts_names", "berlin_neighbourhoods_names",
   "berlin_vbb_AB_names")
rm(list=lsf.str()) # All functions

# Save workspace
# save.image(file = file.path(getwd(), "ShinyApp", "workspace_for_shinyapp.
  RData", fsep="/"))
```

## A.2 Functions

**Listing 26: |capwords.R|**

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

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

**Listing 27: |correlation_plot.R|**

```
correlation_plot = function(corr_df, variables = corr_df$variable,
                            variables_original = TRUE) {

    correlation = corr_df %>%
      dplyr::mutate(sign        = ifelse(correlation > 0, "positive",
                                  ifelse(correlation < 0, "negative",
                                                          "null")),
                    correlation = abs(correlation)) %>%
      dplyr::arrange(variable)

    if (all(correlation$variable == corr_df$variable)) {

      # If variables are listed with their names from the original dataframe
         (before dummies)
      if (variables_original) {

      correlation = correlation %>%
        dplyr::filter(grepl(paste(variables, collapse = "|"),
                            corr_df$variable))

      # If variables are listed with their names from corr_df
      } else {

        correlation = correlation %>%
          dplyr::filter(variable %in% variables)

      }

    }

    corr_plot = correlation %>%
      ggplot(aes(x = variable, y = correlation, fill = sign)) +
      geom_bar(stat="identity") +
      scale_fill_manual("sign",
                        values = c("positive" = "green4",
                                   "negative" = "red1",
```

```
                                    "null" = "blue3")) +
      coord_flip() +
      labs(x = "variables", y = "correlation",
           title = "Correlation with price") +
      theme_bw() +
      theme(axis.text.y     = element_text(
                                color = ifelse(
                                          correlation$sign == "positive",
                                               "green4",
                                        ifelse(
                                          correlation$sign == "negative",
                                               "red1",
                                              "blue3"))),
            axis.text.x     = element_text(size = rel(1.2)),
            axis.title.x    = element_text(size = rel(1.2)),
            axis.title.y    = element_text(size = rel(1.2)),
            legend.text     = element_text(size = rel(1.2)),
            legend.position = "bottom") +
      scale_y_continuous(expand = expand_scale(mult = c(0, 0.05)))

    return(corr_plot)

}
```

**Listing 28: |descriptive_statistics.R|**

```
descriptive_statistics = function(df) {

  # Descriptive statistics for numeric variables
  if (unique(apply(df, 2, function(x) is.numeric(x)))) {

    summary = data.frame(
                variable = names(df),
                min      = apply(df, 2, min),
                `1Q`     = apply(df, 2, quantile, probs = 0.25),
                median   = apply(df, 2, median),
                `3Q`     = apply(df, 2, quantile, probs = 0.75),
                max      = apply(df, 2, max),
                iqr      = apply(df, 2, IQR),
                mean     = apply(df, 2, mean),
                sd       = apply(df, 2, sd),
        check.names = FALSE) %>%
```

```
      dplyr::mutate_if(is.numeric, function(x) round(x, 4))

  # Descriptive statistics for categorical variables
  } else if (unique(apply(df, 2, function(x) is.character(x) | is.factor(x)
    | is.logical(x)))) {

    # Frequency
    frequency_list = apply(df, 2, table)

    frequency_df = data.frame(frequency = unlist(frequency_list)) %>%
      tibble::rownames_to_column(var = "var_fact")

    freq_var = colsplit(string = frequency_df$var_fact, pattern = "\\.",
      names = c("variable", "factor"))

    frequency = freq_var %>%
      cbind(frequency_df) %>%
      dplyr::select(-var_fact)

    # Proportion
    proportion_list = apply(df, 2, function(x) prop.table(table(x)))

    proportion_df = data.frame(proportion = unlist(proportion_list)) %>%
      tibble::rownames_to_column(var = "var_fact")

    prop_var = colsplit(string = proportion_df$var_fact, pattern = "\\.",
      names = c("variable", "factor"))

    proportion = prop_var %>%
      cbind(proportion_df) %>%
      dplyr::select(-var_fact) %>%
      dplyr::mutate(proportion = round(proportion, 4)*100,
                    proportion = as.character(proportion) %>% paste("%"))

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

  } else {

    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     #First selection
28     dplyr::select(id, longitude, latitude, price, property_type, room_type,
29                   instant_bookable, listing_url, host_is_superhost,
30                   security_deposit, cleaning_fee,
31                   calculated_host_listings_count, host_has_profile_pic, host
```

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

**Listing 30: |distance_count.R|**

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

```
    var_name_count = paste(var_name, "count", sep = "_")
    var_name_dist = paste(var_name, "dist", sep = "_")
    # Calculate distance for each listing to each station
    point_distance = geosphere::distm(x = main %>%
                                          dplyr::select(long, lat),
                                      y = reference %>%
                                          dplyr::select(long, lat),
                                      fun = distHaversine) %>%
        as.data.frame() %>%
        data.table::setnames(as.character(reference$id))
    # Calculate how many "reference" are within "distance"
    point_distance[,var_name_count] = rowSums(point_distance <= distance)
      %>%
                                        as.factor()

    # Calculate the distance to the nearest "reference"
    point_distance[,var_name_dist]  = apply(point_distance[,-ncol(point_
      distance)],
                                            MARGIN = 1,
                                            FUN = min) %>%
                                        round(0)
    # Insert this information into the main DF
    main = point_distance %>%
        dplyr::mutate(id = main$id) %>%
        dplyr::select(id, var_name_count, var_name_dist) %>%
        dplyr::right_join(main, by = "id")

    return(main)
}
```

**Listing 31: |distribution_plot.R|**

```
distribution_plot = function(df, var_name, hide_outliers = FALSE, tilt_text_
  x = FALSE) {

  variable = sym(var_name)

  # Plot for numeric variables
  if (is.numeric(df[, var_name])) {

    numvar_plot = df %>%
      dplyr::transmute(
```

```
                var             = !!variable,
                median          = median(var),
                mean            = mean(var),
                `1Q`            = quantile(var, probs = 0.25),
                `3Q`            = quantile(var, probs = 0.75),
                IQR             = IQR(var),
                upper           = (IQR*1.5) + `3Q`,
                lower           = `1Q` - (IQR*1.5),
                outlier_upper   = ifelse(var > unique(upper), 1, 0),
                outlier_lower   = ifelse(var < unique(lower), 1, 0),
                outlier         = ifelse(outlier_upper == 1 | outlier_lower
                  == 1, 1, 0),
                count_upper     = sum(outlier_upper),
                count_lower     = sum(outlier_lower)
                )

  if (hide_outliers) {

    plot = numvar_plot %>%
      dplyr::filter(outlier == 0) %>%
      ggplot() +
      geom_density(aes(x = var), fill = "green4", color = "green4") +
      geom_vline(aes(xintercept = unique(median)),
                 color = "blue", linetype = "dashed") +
      annotate(geom = "text", x = (unique(numvar_plot$median) + 2.6), y =
        0,
               label = "Median",
               colour = "blue",
               angle = 90, hjust = 0) +
      geom_vline(aes(xintercept = unique(mean)),
                 color = "black", linetype = "dashed") +
      annotate(geom = "text", x = (unique(numvar_plot$mean) + 2.6), y = 0,
               label = "Mean",
               colour = "black",
               angle = 90, hjust = 0) +
      geom_vline(aes(xintercept = `1Q`),
                 color = "red", linetype = "dashed") +
      annotate(geom = "text", x = (unique(numvar_plot$`1Q`) + 2.6), y = 0,
               label = "1st Quantile",
               colour = "red",
               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
                                   and",
57                               numvar_plot$count_upper, "values over 1,5*IQR",
58                               sep = " "),
59               title = paste("Density of the variable", variable, sep = " "),
60               x = var_name) +
61          theme_bw() +
62          theme(axis.text.x = element_text(size = rel(1.2)),
63                axis.text.y = element_text(size = rel(1.2)),
64                axis.title.x = element_text(size = rel(1.2)),
65                axis.title.y = element_text(size = rel(1.2)))
66      } else {
67
68        plot = numvar_plot %>%
69          ggplot() +
70          geom_density(aes(x = var), fill = "green4", color = "green4") +
71          geom_vline(aes(xintercept = unique(median)),
72                     color = "blue", linetype = "dashed") +
73          geom_vline(aes(xintercept = unique(mean)),
74                     color = "black", linetype = "dashed") +
75          geom_vline(aes(xintercept = `1Q`),
76                     color = "red", linetype = "dashed") +
77          geom_vline(aes(xintercept = `3Q`),
78                     color = "red", linetype = "dashed") +
79          labs(x     = var_name,
80               title = paste("Density of the variable", variable, sep = " "))
                +
81          theme_bw() +
82          theme(axis.text.x = element_text(size = rel(1.2)),
83                axis.text.y = element_text(size = rel(1.2)),
84                axis.title.x = element_text(size = rel(1.2)),
85                axis.title.y = element_text(size = rel(1.2)))
86
87      }
```

```
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       labs(x     = variable,
109            title = paste("Histogram of the variable", variable, sep = " "))
                +
110       theme_bw() +
111       theme(axis.text.x = element_text(size = rel(1.2)),
112             axis.text.y = element_text(size = rel(1.2)),
113             axis.title.x = element_text(size = rel(1.2)),
114             axis.title.y = element_text(size = rel(1.2)))
115
116     if (tilt_text_x) {
117
118       plot = plot +
119         theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5)
              )
120
121     }
122
123
124     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
```

```
  k_values = min:max
  # Empty dataframe for the total variance explained
  tve = data.frame(clusters = k_values,
                   tve      = rep(NA, length(k_values)))
  # Empty list for the kmeans objects
  clk = list()
  # Loop through the possible values of k
  for (k in k_values) {
    # Calculate k-means for each k
    clk[[k-1]] = kmeans(scaled_df, centers = k, iter.max = iter_max)
    # Save the number of clusters
    names(clk)[k-1] = paste(k, "clusters", sep = " ")
    # Calculate percentage of total variance explained
    tve$tve[k-1] = 1-clk[[k-1]]$tot.withinss/clk[[k-1]]$totss
    # Print process
    print(paste("k-means with", k, "clusters done", sep = " "))
  }

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

  return(plot)
}
```

**Listing 35: |point_in_polygons.R|**

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

```r
  is_in <- data.frame(matrix(ncol = nrow(polys_sf), nrow = nrow(points_df)))
  is_in[,var_name] <- NA
  # Extract coordinates of the polygons
  coordinates <- as.data.frame(st_coordinates(polys_sf))
  # Extract names of the polygons
  name <- as.character(polys_sf$id)
  # For all polygons check if the points are inside of them
  for (k in 1:nrow(polys_sf)) {
    is_in[,k] <- sp::point.in.polygon(point.x = points_df$long,
                                      point.y = points_df$lat,
                                      pol.x   = coordinates$X
                                                [coordinates$L2 == k],
                                      pol.y   = coordinates$Y
                                                [coordinates$L2 == k])
    # Get the names of the polygons where the points are in one column
    is_in[,var_name][is_in[,k] == 1] <- name[k]
  }
  # Keep only summary column and add points' names
  is_in <- is_in %>%
    dplyr::select(var_name) %>%
    dplyr::mutate(id = points_df$id)
  # Add the summary column to the points dataframe
  points_df <- dplyr::full_join(points_df, is_in, by = join_var)
  return(points_df)
}
```

**Listing 36: |points_midpoint.R|**

```r
# Calculate midpoint from a sf point object
points_midpoint <- function(points_sf, point_name = "id") {
  # Create symbol version of point_name
  point_name <- sym(point_name)
  # Get coordinates from sf point objects
  coordinates <- sf::st_coordinates(points_sf) %>%
    base::as.data.frame() %>%
    dplyr::rename(lat  = Y,
                  long = X)
  # Bind coordinates with sf point object and calculate mean of lat and long
  midpoints_df <- points_sf %>%
    base::as.data.frame() %>%
    dplyr::select(-geometry) %>%
    base::cbind(coordinates) %>%
```

```
    dplyr::group_by(!!point_name) %>%
    dplyr::summarize(lat    = mean(lat),
                     long   = mean(long),
                     s_bahn = any(s_bahn),
                     u_bahn = any(u_bahn))
  return(midpoints_df)
}
```

**Listing 37: |recode_all.R|**

```
# Recode variable: rename all factor levels of that variable
recode_all = function(df, variable) {
    variable_name = sym(variable)
    df = df %>%
      mutate(!!variable_name := as.factor(!!variable_name))
    levels(df[,variable]) = gsub(" ", "_", levels(df[,variable])) %>%
      tolower() %>%
      paste(variable, ., sep = "_")
  return(df)
}
```

**Listing 38: |summarize_df.R|**

```
# Summary information for numeric and factor variables
summarize_df <- function(df, wrt = NULL, vars_mean, vars_count) {
  # Create empty vector
  summary_df <- c()
  # For the case in which the summary is general
  if(is.null(wrt)) {
    # Variables of which we calculate the mean
    for (var in vars_mean) {
      var_name <- sym(var)
      mean_df <- df %>% dplyr::summarize(!!var_name := mean(!!var_name))
      summary_df <- c(summary_df,mean_df)
    }
    # Variables of which we count the unique values of the factors
    for (var in vars_count) {
      var_name <- sym(var)
      count_df <- df %>%
        recode_all(var) %>%
        group_by(!!var_name) %>%
        dplyr::summarize(count = n()) %>%
```

```
        spread(key = !!var_name, value = count)
      summary_df <- summary_df %>% cbind(count_df)
    }
    # For the case in which the summary is done wrt a variable
  } else {
    wrt_name <- sym(wrt)
    # Variables of which we calculate the mean
    for (var in vars_mean) {
      var_name <- sym(var)
      mean_df <- df %>%
        group_by(!!wrt_name) %>%
        dplyr::summarize(!!var_name := mean(!!var_name))
      summary_df <- c(summary_df,mean_df)
      summary_df[duplicated(summary_df)] <- NULL
    }
    # Variables of which we count the unique values of the factors
    for (var in vars_count) {
      var_name <- sym(var)
      count_df <- df %>%
        recode_all(var) %>%
        group_by(!!var_name, !!wrt_name) %>%
        dplyr::summarize(count = n()) %>%
        spread(key = !!var_name, value = count)
      summary_df <- summary_df %>% as.data.frame() %>%
        dplyr::mutate(!!wrt_name := as.character(!!wrt_name)) %>%
        full_join(count_df, by = wrt) %>%
        replace(is.na(.), 0)
    }
    # Percentage of the rows
    percentage_df <- df %>%
      group_by(!!wrt_name) %>%
      dplyr::summarize(percentage = round(n()/nrow(.)*100, 2))
    summary_df <- summary_df %>% full_join(percentage_df, by = wrt)
  }
  return(summary_df)
}
```

**Listing 39: |summary_stat_fact.R|**

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

```

```
  if (is.null(constraint_var) & is.null(constraint_value)) {
    factor = df[, var]
  } else if (is.null(constraint_var) | is.null(constraint_value)) {
    stop("constraint_var or constraint_value is not defined. Review function
       values.")
  } else {
    constraint = sym(constraint_var)
    df         = df %>%
                   dplyr::filter(!!constraint == constraint_value)
    factor   = df[, var]
  }

  frequency_df = table(factor) %>%
    as.data.frame() %>%
    dplyr::rename(frequency = Freq)

  proportion_df = prop.table(table(factor)) %>%
    as.data.frame() %>%
    dplyr::rename(proportion = Freq) %>%
    dplyr::mutate(proportion = round(proportion, 4)*100,
                  proportion = as.character(proportion) %>% paste("%"))

  summary = frequency_df %>%
    dplyr::inner_join(proportion_df, by = "factor") %>%
    dplyr::arrange(desc(frequency))

  return(summary)

}
```

**Listing 40: |summary_stat_num.R|**

```
summary_stat_num = function(df, var, constraint_var = NULL, constraint_value
   = NULL) {

  if (is.null(constraint_var) & is.null(constraint_value)) {
    variable = df[, var]
  } else if (is.null(constraint_var) | is.null(constraint_value)) {
    stop("constraint_var or constraint_value is not defined. Review function
       values.")
  } else {
    constraint = sym(constraint_var)
```

```
    df          = df %>%
                  dplyr::filter(!!constraint == constraint_value)
    variable    = df[, var]
  }

  summary = data.frame(
              min    = min(variable),
              `1Q`   = quantile(variable, probs = 0.25),
              median = median(variable),
              `3Q`   = quantile(variable, probs = 0.75),
              max    = max(variable),
              iqr    = IQR(variable),
              mean   = mean(variable),
              sd     = sd(variable),
          check.names = FALSE) %>%
    dplyr::mutate_if(is.numeric, function(x) round(x, 4))

  return(summary)

}
```

**Listing 41: |var_avg_map.R|**

```
var_avg_map = function(summary_df, var_name,
                       polygon_sf, polygon_names_df, polygon_names_var,
                       bins = TRUE, nsplits = 4, pretty = FALSE, zoom_level
                         = 10,
                       title = paste("Average of", var_name, sep = " ")) {


  if (bins) {

    colors = colorBin(topo.colors(nsplits), summary_df[, var_name], bins =
      nsplits, pretty = pretty)

  } else {

    if (pretty) {

      stop("Parameter pretty not used in colorQuantile.")

    }
```

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

## 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
                Berlin!",
28              width  = 12,
29              status = "danger",
30              uiOutput("intro_ui"),
31              br(),
32              column(width = 4,
33                     h4("Maps Tab"),
34                     textOutput("intro_maps_text"),
35                     br(),
36                     actionButton("switch_maps", "Go to Maps Tab")),
37              column(width = 4,
38                     h4("Price Tab"),
39                     textOutput("intro_price_text"),
40                     br(),
41                     actionButton("switch_price", "Go to Price Tab")),
42              column(width = 4,
43                     h4("Clustering Tab"),
44                     textOutput("intro_cluster_text"),
45                     br(),
46                     actionButton("switch_cluster",
```

```
                                    "Go to Clustering Tab")))
        )
      ),

      ### TAB2 (Maps)
      tabItem(tabName = "maps",
          fluidRow(
            tabBox(id = "berlin",
                   width  = 12,
                   tabPanel(
                     title = "Berlin map",
                     textOutput("berlin_text"),
                     br(),
                     box(
                       width = 12,
                       solidHeader=TRUE,
                       column(
                         width = 6,
                         selectInput(
                           inputId = "view",
                           label   = "Choose view",
                           choices = berlin_sf %>%
                                       as.data.frame() %>%
                                       select(view) %>%
                                       unique() %>%
                                       t() %>%
                                       c() %>%
                                       setdiff("Neighbourhoods"))),
                       column(
                         width = 6,
                         selectInput(
                           inputId = "variable1",
                           label   = "Choose variable",
                           choices = c("none", listings_area_summary %>%
                                               colnames() %>%
                                               setdiff(c("id", "view", "
                                                 group")) %>%
                                               gsub("_", " ", .))))
                             ),
                          box(
                            width = 12,
```

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

```
                                )
                              )
                            ),
                    tabPanel(
                      title = "District map",
                      textOutput("district_text"),
                      br(),
                      box(
                        width = 12,
                        solidHeader=TRUE,
                        column(
                          width = 6,
                          selectInput(inputId  = "district",
                                      label    = "Choose district",
                                      choices  = berlin_names %>%
                                                   filter(view == "Districts
                                                     ") %>%
                                                   select(id) %>% unique()
                                                     %>% t() %>% c(),
                                      selected = "Mitte")),
                        column(
                          width = 6,
                          selectInput(inputId = "variable2",
                                      label   = "Choose variable",
                                      choices = c("none", listings_area_
                                        summary %>%
                                                              colnames() %>%
                                                              setdiff(c("id", "
                                                                view", "group")
                                                                ) %>%
                                                              gsub("_", " ", .)
                                                                )))
                        ),
                        box(
                          width = 12,
                          solidHeader=TRUE,
                          column(width = 4,
                                 checkboxInput(inputId = "district_
                                   properties",
                                               label   = "Display
                                                 properties",
```

```
                                                value    = FALSE)),
                        column(
                          width = 4,
                          checkboxInput(inputId = "district_stations
                            ",
                                        label   = "Display train/
                                          subway stations",
                                        value   = FALSE)),
                        column(
                          width = 4,
                          checkboxInput(inputId = "district_sights",
                                        label   = "Display top 10
                                          attractions",
                                        value   = FALSE))

                        ),
                      box(width       = 12,
                        align       = "center",
                        solidHeader = TRUE,
                        h4(textOutput("title_secondary_map")),
                        leafletOutput("secondary_map") %>%
                          withSpinner()
                        ),
                      box(width       = 12,
                        align       = "center",
                        solidHeader = TRUE,
                        column(
                          width = 4,
                          h4(textOutput("title_secondary_table")),
                          tableOutput("secondary_table1"),
                          tableOutput("secondary_table2")
                        ),
                        column(
                          width = 8,
                          h4(textOutput("title_secondary_plot")),
                          plotOutput("secondary_plot")
                        )
                      )
                      )

            )
```

```
          )
  ),

  ### TAB3 (Price)
  tabItem(tabName = "price",
          fluidRow(
            box(
              status = "danger",
              width  = 12,
              column(
                width = 6,
                textOutput(outputId = "price_text")
                ),
              column(
                width = 6,
                pickerInput(inputId  = "variable3", label = "Choose variable
                  (s)",
                            choices  = listings_price_correlation %>%
                              select(variable) %>%
                              unique() %>%
                              t() %>%
                              c() %>%
                              gsub("_", " ", .),
                            multiple = TRUE,
                            selected = listings_price_correlation %>%
                              select(variable) %>%
                              unique() %>%
                              t() %>%
                              c() %>%
                              gsub("_", " ", .),
                            options  = list(`actions-box` = TRUE))
                )



                )
            ),
          fluidRow(
            tabBox(id = "price_box",
                   width  = 12,
                   tabPanel(
                     title = "Correlation with price",
```

```
                        plotOutput(outputId = "corr_plot")
                        ),
                      tabPanel(
                        title = "Linear regression on price",
                        tabBox(id = "lm_model",
                               width = 12,
                               tabPanel(
                                 title = "Model",
                                 box(
                                   width = 12,
                                   align = "center",
                                   solidHeader = TRUE,
                                   column(
                                     width = 3,
                                     h4("R-Squared of the model"),
                                     textOutput(outputId = "price_lm_r2")
                                     ),
                                   column(
                                     width = 9,
                                     h4("Coefficients of the linear regression
                                       on price."),
                                     tableOutput(outputId = "price_lm_
                                       coefficients")
                                   )
                                 )
                               ),
                               tabPanel(
                                 title = "Residuals",
                                 box(width = 12,
                                     solidHeader = TRUE,
                                     align = "center",
                                     h4("Summary statistics of the residuals"),
                                     tableOutput(outputId = "price_lm_residuals
                                       ")
                                    ),
                                 box(width = 12,
                                     solidHeader = TRUE,
                                     align = "center",
                                     h4("Residuals plots"),
                                     column(
                                       width = 6,
```

```
274                                   plotOutput(outputId = "price_lm_resplot"
                                        ) %>% withSpinner()
275                                   ),
276                                 column(
277                                   width = 6,
278                                   plotOutput(outputId = "price_lm_qqplot")
                                        %>% withSpinner()
279                                   )
280                                 )
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
                                              clusters to test",
300                                      min = 2, max = 96, value = c(2, 40),
                                              step = 1,
301                                      width = "100%"),
302                          actionButton("ctest_start", "Calculate"),
303                          progressBar(id = "ctest_progress",
304                                      status = "danger",
305                                      value = 0, total = 100,
306                                      display_pct = TRUE),
307                          plotOutput(outputId = "tve")
308                          ),
309
310                 tabPanel(title = "Cluster Airbnb properties",
```

```
                        uiOutput("cnum_ui"),
                        br(),
                        numericInput(inputId  = "cnum",
                                     label = "Choose number of clusters for
                                        clustering",
                                     min = 2, max = 96, value = 12, step =
                                       1,
                                     width = "30%"),
                        actionButton("cnum_start", "Calculate"),
                        box(
                          width       = 12,
                          align       = "center",
                          solidHeader = TRUE,
                          column(
                            width = 6,
                            h4("Clusters on coordinates"),
                            plotOutput(outputId = "clustercoord") %>%
                              withSpinner()
                           ),
                          column(
                            width = 6,
                            h4("Clusters on Principal Components"),
                            plotOutput(outputId = "clusterpc") %>%
                              withSpinner()
                           )
                        )
                    )
                )
        )


      )
    )
  )
)
```

**Listing 43: |server.R|**

```
server = function(input, output, session) {

  # Tab 1
  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
                     a042110a/")
13       text2 = " for the university course 'Statististical Programming
           Languages' at the "
14       url2  = a("Humboldt-University of Berlin",
15                 href="https://www.hu-berlin.de/")
16       text3 = " in the Winter Termin 2018/19. It shows some Exploratory Data
```

```
        Analysis of the Airbnb properties
      in Berlin updated at the 14. January 2019."

      part1 = paste(text1, url1, text2, url2, text3, sep = "") %>%
        paste("<br/>", sep = "<br/>")
      html1 = HTML(part1)

      text4 = "The data for this project has been downloaded from the
        following sources:"
      url4  = a("Inside Airbnb",
                href="http://insideairbnb.com/")
      url5  = a("Statistics Office of Berlin-Brandenburg",
                href="https://www.statistik-berlin-brandenburg.de/produkte/
                  opendata/geometrienOD.asp?Kat=6301")
      url6  = a("Geofabrik",
                href="http://download.geofabrik.de/europe/germany/berlin.
                  html")

      part2 = paste(text4, "<ul><li>", sep = "<br/>") %>%
        paste(url4, sep = "") %>%
        paste(url5, sep = "</li><li>") %>%
        paste(url6, sep = "</li><li>") %>%
        paste("",   sep = "</li></ul><br/>")
      html2 = HTML(part2)

      text5 = "The complete code for this project can be found on"
      url7  = a("GitHub",
                href="https://github.com/silvia-ventoruzzo/SPL-WISE-2018")
      text6 = "."

      part3 = paste(text5, url7, sep = " ") %>%
        paste(text6, sep = "")
      html3  = HTML(part3)

      tagList(html1, html2, html3)


    })

    output$intro_maps_text = renderText({
      "Map of Berlin or one of its district and variable distribution."
```

```
    })

    output$intro_price_text = renderText({
      "Variable correlation with price and linear regression on price."
    })

    output$intro_cluster_text = renderText({
      "Clustering of Airbnb properties."
    })

    # Connections to other tabs
    observeEvent(input$switch_maps, {
      updateTabItems(session, inputId = "tabs", selected = "maps")
    })

    observeEvent(input$switch_price, {
      updateTabItems(session, inputId = "tabs", selected = "price")
    })

    observeEvent(input$switch_cluster, {
      updateTabItems(session, inputId = "tabs", selected = "cluster")
    })

}
```

**Listing 45: |maps_berlin_server.R|**

```
##############################################################################
# Maps Berlin Server-file ####################################################
# Author: Silvia Ventoruzzo ##################################################
##############################################################################

maps_berlin_server = function(input, output, session) {

    output$berlin_text <- renderText({
      "In this tab you can view a map of entire Berlin divided in its
        different areas. You can choose which view
      to show and also which variable to show the average in the different
        areas. Moreover you can also display
      the properties themselves, the railway stations and the top 10 tourist
         attractions."
    })
```

```
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$view, sep = " ")
22
23        } else if (grepl(paste(factor_vars(), collapse = "|"), variable1())) {
24
25          variable = stringr::str_match(variable1(), paste(factor_vars(),
              collapse = "|")) %>%
26            c() %>% gsub("_", " ", .) %>% capwords()
27
28          variable %>%
29            paste("Count of the variable", ., "in Berlin's", input$view, sep =
                " ")
30
31        } else {
32
33          variable = capwords(input$variable1)
34
35          variable %>%
36            paste("Average of the variable", ., "in Berlin's", input$view, sep
                = " ")
37
38        }
39
40      })
41
42      observeEvent({c(input$view, input$variable1)},
43                   {output$title_main_map = renderText({title_main_map_change
                       ()})},
44                   ignoreNULL = FALSE)
45
46      ## REACTIVE ELEMENTS
47
48      # SF Dataframe to plot polygons
49      polygons = reactive({berlin_sf %>% filter(view == input$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$variable1)
66
67      })
68
69      var_table_plot = reactive({
70
71        if (variable1() == "none") { return() }
72
73        if (grepl(paste(factor_vars(), collapse = "|"), variable1())) {
74
75          variable = stringr::str_match(variable1(), paste(factor_vars(),
            collapse = "|")) %>% c()
76
77        } else {
78
79          variable = variable1()
80
81        }
82
83      })
84
85      # Number of areas in selected view
86      area_num = reactive({
87
88        berlin_names %>%
89          dplyr::filter(view == input$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     ## MAP
109
110     output$main_map = renderLeaflet({
111
112       main_icons = awesomeIconList(
113         "main_properties" = makeAwesomeIcon(icon = "home", markerColor = "
            red"),
114         "main_stations"   = makeAwesomeIcon(icon = "subway", library = "fa",
              markerColor = "green", iconColor = "#FFFFFF"),
115         "main_sights"     = makeAwesomeIcon(icon = "eye-open", markerColor =
              "blue")
116       )
117
118       main_map = leaflet() %>%
119         addTiles() %>%
120         addLabelOnlyMarkers(data = area_names(),
121                             lng = ~long, lat = ~lat, label = ~lapply(name,
                                  htmltools::HTML),
122                             labelOptions = labelOptions(noHide = TRUE,
                                  direction = 'center',
123                                                         textOnly = TRUE,
                                                           textsize = text_
```

```
                                                    size()))  %>%
124         addAwesomeMarkers(data = listings,
125                           lng = ~long, lat = ~lat,
126                           icon = main_icons[["main_properties"]],
127                           group = "main_properties",
128                           popup = ~lapply(paste("<b><a href=", listing_url,
                                "> ID: ", id, "</a></b>", sep = "") %>%
129                                           paste("Price: ", sep = "<br/>")
                                              %>%
130                                           paste(price, "$", sep = "") %>%
131                                           paste(property_type, sep = "<br/
                                              >") %>%
132                                           paste(room_type, sep = " - ")
                                              %>%
133                                           paste("Accomodates", sep = "<br/
                                              >") %>%
134                                           paste(accommodates, ifelse(
                                              accommodates == 1, "person", "
                                              people"), sep = " "),
135                                         htmltools::HTML),
136                           clusterId = "1",
137                           clusterOptions = markerClusterOptions(
138                             iconCreateFunction=JS("function (cluster) {
139                                                   var childCount = cluster.
                                                      getChildCount();
140                                                   if (childCount > 1) {
141                                                   c = 'rgba(235, 0, 0, 1);'}
142                                                   return new L.DivIcon({
                                                      html: '<div style=\"
                                                      background-color:'+c+'\"
                                                      ><span>' + childCount +
                                                      '</span></div>',
                                                      className: 'marker-
                                                      cluster', iconSize: new
                                                      L.Point(40, 40) });}"),
143                             singleMarkerMode = FALSE)) %>%
144         addAwesomeMarkers(data = railway_stations_df,
145                           lng = ~long, lat = ~lat,
146                           icon = main_icons[["main_stations"]],
147                           popup = ~htmltools::htmlEscape(
148                             paste(ifelse(s_bahn == TRUE, "S", ""),
```

```
149                                 ifelse(s_bahn == TRUE & u_bahn == TRUE, "+
                                      ", ""),
150                                 ifelse(u_bahn == TRUE, "U", ""),
151                                 sep = "") %>%
152                               paste(id, sep = " ")),
153                         group = "main_stations",
154                         clusterId = "2",
155                         clusterOptions = markerClusterOptions(
156                           iconCreateFunction = JS("function (cluster) {
157                                                   var childCount = cluster
                                                      .getChildCount();
158                                                   if (childCount > 1) {
159                                                   c = 'rgba(7, 187, 31, 1)
                                                      ;'}
160                                                   return new L.DivIcon({
                                                      html: '<div style=\"
                                                      background-color:'+c+'
                                                      \"><span>' +
                                                      childCount + '</span><
                                                      /div>', className: '
                                                      marker-cluster',
                                                      iconSize: new L.Point
                                                      (40, 40) });}"),
161                           singleMarkerMode = FALSE)) %>%
162       addAwesomeMarkers(data = attractions_df,
163                         lng = ~long, lat = ~lat,
164                         icon = main_icons[["main_sights"]],
165                         popup = ~htmltools::htmlEscape(id),
166                         group = "main_sights",
167                         clusterId = "3",
168                         clusterOptions = markerClusterOptions(
169                           iconCreateFunction = JS("function (cluster) {
170                                                   var childCount = cluster
                                                      .getChildCount();
171                                                   if (childCount > 1) {
172                                                   c = 'rgba(11, 200, 213,
                                                      1);'}
173                                                   return new L.DivIcon({
                                                      html: '<div style=\"
                                                      background-color:'+c+'
                                                      \"><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       setView(lng = berlin_center$long, lat = berlin_center$lat, zoom =
            10)
177
178     if (variable1() == "none") {
179
180       main_map %>%
181         addPolygons(data = polygons(), weight = 1, smoothFactor = 1,
            fillOpacity = 0.8,
182                     color = "grey", fillColor = colors())
183
184     } else {
185
186       colors_vars = colors_var()
187
188       main_map %>%
189         addPolygons(data = polygons(), weight = 1, smoothFactor = 1,
            fillOpacity = 0.8,
190                     color = "grey",
191                     fillColor = ~colors_vars(listings_area_summary[
                          listings_area_summary$view == input$view, variable1
                          ()])) %>%
192         addLegend(position = "bottomleft", pal = colors_var(),
193                   values = listings_area_summary[listings_area_summary$
                        view == input$view, variable1()],
194                   labFormat = labelFormat(transform = function(x) sort(x,
                        decreasing = TRUE)),
195                   title = variable1())
196
197     }
198     })
199
200   # checkboxInputs
201   observeEvent({c(input$main_properties, input$main_stations, input$main_
```

```
    sights)}, {
202
203     proxy = leafletProxy("main_map")
204
205     if (input$main_properties) {proxy %>% showGroup("main_properties")
206     } else {proxy %>% hideGroup("main_properties")}
207
208     if (input$main_stations) {proxy %>% showGroup("main_stations")
209     } else {proxy %>% hideGroup("main_stations")}
210
211     if (input$main_sights) {proxy %>% showGroup("main_sights")
212     } else {proxy %>% hideGroup("main_sights")}
213
214   })
215
216
217   ### DESCRIPTIVE TABLE
218
219   ## TITLE
220   title_main_table_change = reactive({
221
222
223     if (variable1() == "none") { return() }
224
225     if (grepl(paste(factor_vars(), collapse = "|"), variable1())) {
226
227       variable = stringr::str_match(variable1(), paste(factor_vars(),
          collapse = "|")) %>%
228         c() %>% gsub("_", " ", .) %>% capwords()
229
230     } else {
231
232       variable = capwords(input$variable1)
233
234     }
235
236     variable %>%
237       paste("Descriptive statistics of the variable", ., sep = " ")
238
239   })
240
```

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

```
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(),
              collapse = "|")) %>%
331           c() %>% gsub("_", " ", .) %>% capwords()
332
333       } else {
334
335         variable = capwords(variable1())
336
337       }
338
339       variable %>%
340         paste("Distribution of the variable", ., sep = " ")
341     })
342
343     observeEvent({c(input$view, input$variable1)},
344                  {output$title_main_plot = renderText({title_main_plot_
                       change()})},
345                  ignoreNULL = FALSE)
346
347
348     ## REACTIVE ELEMENTS
349
350     ## PLOT
351
352     output$main_plot = renderPlot({
353
354       if (variable1() == "none") {
355
356         return()
357
```

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

**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
         variable to show the average
10     in the different areas and display various properties, railway stations
         and tourist attractions.
11     But here you see the city at the district's level, showing therefore the
          neighbourhoods in the
12     different districts."
13   })
14
15   ### LEAFLET MAP
```

```
16
17  ## TITLE
18
19  title_secondary_map_change = reactive({
20
21    if (variable2() == "none") {
22
23      paste("Map of", input$district, sep = " ") %>%
24        paste("'s", sep = "") %>%
25        paste("Neighbourhoods", sep = " ")
26
27    } else if (grepl(paste(factor_vars(), collapse = "|"), variable2())) {
28
29      variable = stringr::str_match(variable2(), paste(factor_vars(),
          collapse = "|")) %>%
30        c() %>% gsub("_", " ", .) %>% capwords()
31
32      variable %>%
33        paste("Count of the variable", ., "in the district", input$district,
             sep = " ")
34
35    } else {
36
37      variable = capwords(input$variable2)
38
39      variable %>%
40        paste("Average of the variable", ., "in the district", input$
            district, sep = " ")
41
42    }
43  })
44
45  observeEvent({c(input$district, input$variable2)},
46               {output$title_secondary_map = renderText({title_secondary_map
                   _change()})}, ignoreNULL = FALSE)
47
48
49  ## REACTIVE ELEMENTS
50
51  # SF Dataframe to plot polygons
52  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$variable2), the color should represent
        the variable's distribution
100   neighbourhood_colors_var = reactive({
101     leaflet::colorNumeric(
102       palette = topo.colors(neighbourhood_area_num()),
103       domain  = listings_area_summary %>%
104                   dplyr::filter(view == "Neighbourhoods",
105                                 group == input$district) %>%
106                   dplyr::select(variable2()),
107       n       = neighbourhood_area_num(),
108       reverse = TRUE)
109
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
            ", markerColor = "green", iconColor = "#FFFFFF"),
120       "district_sights"     = makeAwesomeIcon(icon = "eye-open", markerColor
             = "blue")
121     )
122
123     secondary_map = leaflet() %>%
124       addTiles() %>%
125       addLabelOnlyMarkers(data = neighbourhood_names(),
126                           lng = ~long, lat = ~lat, label = ~lapply(name,
                              htmltools::HTML),
127                           labelOptions = labelOptions(noHide = TRUE,
                              direction = 'center',
128                                                       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, "$", sep = "") %>%
136                                         paste(property_type, sep = "<br/>"
                                            ) %>%
137                                         paste(room_type, sep = " - ") %>%
138                                         paste("Accomodates", sep = "<br/>"
                                            ) %>%
139                                         paste(accommodates, ifelse(
                                            accommodates == 1, "person", "
                                            people"), sep = " "),
140                                       htmltools::HTML),
141                         clusterId = "1",
142                         clusterOptions = markerClusterOptions(
143                           iconCreateFunction=JS("function (cluster) {
144                                                 var childCount = cluster.
                                                    getChildCount();
145                                                 if (childCount > 1) {
146                                                 c = 'rgba(235, 0, 0, 1);'}
147                                                 return new L.DivIcon({ html:
                                                    '<div style=\"background-
                                                    color:'+c+'\"><span>' +
                                                    childCount + '</span></div
                                                    >', className: 'marker-
                                                    cluster', iconSize: new L.
                                                    Point(40, 40) });}"),
148                           singleMarkerMode = FALSE)) %>%
149       addAwesomeMarkers(data = railway_stations_df %>% filter(district ==
            input$district),
150                         lng = ~long, lat = ~lat,
151                         icon = district_icons[["district_stations"]],
152                         popup = ~htmltools::htmlEscape(
```

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

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

```
                                                    ,
208                                                 variable2()],
209                 labFormat = labelFormat(transform = function(x) sort(x,
                      decreasing = TRUE)),
210                 title = variable2())
211
212       }
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
231 ### SUMMARY TABLE
232
233 ## TITLE
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(),
        collapse = "|")) %>%
242       c() %>% gsub("_", " ", .) %>% capwords()
243
```

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

```
      variable = variable2()

    }

  })


  table_df = reactive({

    if (variable2() == "none") { return() }

    df = listings %>%
      dplyr::filter(district == input$district) %>%
      dplyr::select(var_table_plot2())

    return(df)

  })

  ## TABLE

  # Descriptive table
  output$secondary_table1 = renderTable({

    if (variable2() == "none") {

      return()

    } else if (grepl(paste(factor_vars(), collapse = "|"), variable2())) {

      secondary_df = descriptive_statistics(table_df())

    } else {

      secondary_df = descriptive_statistics(table_df()) %>%
        dplyr::select(min, `1Q`, median, `3Q`, max)

    }

    return(secondary_df)

```

```
321   })
322
323   output$secondary_table2 = renderTable({
324
325     if (variable2() == "none") {
326
327       return()
328
329     } else if (grepl(paste(factor_vars(), collapse = "|"), variable2())) {
330
331       return()
332
333     } else {
334
335       secondary_df = descriptive_statistics(table_df()) %>%
336         dplyr::select(mean, sd)
337
338     }
339
340     return(secondary_df)
341
342   })
343
344   ### DISTRIBUTION PLOT
345
346   ## TITLE
347   title_secondary_plot_change = reactive({
348
349     # factor_vars() <- c("room_type", "property_type", "room_type")
350
351     if (variable2() == "none") { return() }
352
353     if (grepl(paste(factor_vars(), collapse = "|"), variable2())) {
354
355       variable = stringr::str_match(variable2(), paste(factor_vars(),
          collapse = "|")) %>%
356         c() %>% gsub("_", " ", .) %>% capwords()
357
358     } else {
359
360       variable = capwords(input$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$title_secondary_plot = renderText({title_secondary_
                   plot_change()})},
370                ignoreNULL = FALSE)
371
372
373   ## REACTIVE ELEMENTS
374
375   plot_df2 = reactive({
376
377     if (variable2() == "none") {
378
379       return()
380
381     } else if (grepl(paste(factor_vars(), collapse = "|"), variable2())) {
382
383       df = listings %>%
384         filter(district == input$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
```

```
  output$secondary_plot = renderPlot({

    if (variable2() == "none") {

      return()

    } else if (grepl(paste(factor_vars(), collapse = "|"), variable2())) {

      plot2 = distribution_plot(plot_df2(), var_table_plot2(), tilt_text_x =
         TRUE)

    } else {

      plot2 = distribution_plot(plot_df2(), var_table_plot2(), hide_outliers
         = TRUE)

    }

    return(plot2)

  })

}
```

**Listing 47: |price_corr_server.R|**

```
################################################################################
# Price Correlation Server-file ################################################
# Author: Silvia Ventoruzzo ####################################################
################################################################################

price_corr_server = function(input, output, session) {

  ## TEXT

  output$price_text = renderText({
    "In this tab you can choose one or multiple variables and check their
      correlation with price. Moreover,
    a linear regression on price is run with the selected variable(s) as
      regressors. You can also check the
    coefficients and the residuals for this mode."
  })
```

```
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({
```

```
      validate(
        need(!is.null(input$variable3), "Please select variable(s)")
      )

      listings_price %>%
        dplyr::select(price, gsub(" ", "_", input$variable3))

    })

    price_lm = reactive({

      lm(price ~ ., data = price_lm_df())

    })

    ## TEXT WITH R-SQUARED

    output$price_lm_r2 = renderText({

      summary(price_lm())$r.squared %>% round(4)

    })

    ## TABLE WITH COEFFICIENTS

    output$price_lm_coefficients = renderTable({

      coefficients = data.frame(
        variable = c("(Coefficient)", gsub(" ", "_", input$variable3)),
        coefficient = price_lm()$coefficients) %>%
        tidyr::drop_na() %>%
        dplyr::mutate(num = 1:nrow(.)) %>%
        dplyr::mutate_if(is.numeric, function(x) round(x, 4))

      significance = data.frame(
        significant = ifelse(summary(price_lm())$coefficients[,4] < 0.5,
                             TRUE, FALSE)) %>%
        dplyr::mutate(num = 1:nrow(.))

      lm = coefficients %>%
        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
```

```
    })

}
```

**Listing 49: |clustering_server.R|**

```
################################################################################
# Clustering Server-file ######################################################
# Author: Silvia Ventoruzzo ###################################################
################################################################################

clustering_server = function(input, output, session) {

  output$cluster_ui <- renderUI({

    text1  = "In this tab you can cluster Airbnb properties in Berlin. The
      process is the following:"
    text2  = "Choose the number of the clusters according to the elbow-
      method"
    text2a = "Choose the number of clusters to test"
    text2b = "Calculate for each number the percentage of the total variance
       explained"
    text2c = "Plot the total variance explained against the number of
      clusters"
    text2d = "Choose the number of clusters where adding another does not
      add sufficient informatoin"
    text3  = "Cluster the properties with the selected number of clusters"
    text4  = "Plot the clusters"

    intern = paste(text2, "<ul><li>", sep = "<br/>") %>%
      paste(text2a, sep = "") %>%
      paste(text2b, sep = "</li><li>") %>%
      paste(text2c, sep = "</li><li>") %>%
      paste(text2d, sep = "</li><li>") %>%
      paste("",   sep = "</li></ul>")

    extern = paste(text1, "<ul><li>", sep = "<br/>") %>%
      paste(intern, sep = "") %>%
      paste(text3, sep = "</li><li>") %>%
      paste(text4, sep = "</li><li>") %>%
      paste("",   sep = "</li></ul>")

```

```
32      html = HTML(extern)
33
34    })
35
36
37      ## Panel 1 (Find number of clusters)
38
39      output$ctest_text <- renderText({
40        "Before clustering the airbnb properties, we need to find out what is
            the optimal number of clusters.
41        We use the elbow method, which means that from plot showing the
            percentage of the total variance
42        explained for that number of clusters we choose the number before the
            increase becomes too flat."
43      })
44
45      tot_var_explained <- eventReactive(input$ctest_start, {
46
47        clusters = seq(input$ctest[1], input$ctest[2], 1)
48
49        cluster_count = input$ctest[2] - input$ctest[1] + 1
50
51        tve = data.frame(clusters = clusters,
52                         tve      = rep(NA, cluster_count))
53
54        clk = list()
55
56        n = 1
57
58        for (k in seq(input$ctest[1], input$ctest[2], 1)) {
59
60          clk[[k-1]] = kmeans(listings_scaled, centers = k, iter.max = 20)
61          tve$tve[k-1] = 1-clk[[k-1]]$tot.withinss/clk[[k-1]]$totss
62
63          updateProgressBar(session = session, id = "ctest_progress",
64                            value = (n/cluster_count*100), total = 100,
65                            title = paste(n, "of", cluster_count))
66          n = n + 1
67        }
68
69        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$ctest[2], 2))
              +
83          theme_bw()
84
85      })
86
87      ## Panel 2 (Actual clustering)
88
89      output$cnum_ui <- renderUI({
90        text <- "Select the optimal number of clusters you found in the
            previous tab and run the clustering
91        function. In this case we use k-means, which splits the data into the
            given number of clusters
92        in such a way as to minimize the within-cluster sum of squares. For
            detailed information on
93        this and other clustering algorithms you can read here:"
94        url <- a("Data Clustering: A Review",
95                 href="http://delivery.acm.org/10.1145/340000/331504/p264-jain
                   .pdf?ip=141.20.217.40&id=331504&acc=PUBLIC&key=2
                   BA2C432AB83DA15%2E04C410D892D772C0%2E4D4702B0C3E38B35%2
                   E4D4702B0C3E38B35&__acm__=1551719946_9
                   d82c9a89fd8cd69cfa3d310ea27bc7f")
96        tagList(text, url)
97      })
98
99      # REACTIVE ELEMENTS
100
101     # k-means
102     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
            ()), alpha = 0.5) +
126         theme(plot.title = element_text(hjust = 0.5)) +
127         labs(color = "clusters") +
128         theme_void()
129
130     })
131
132     # Plot of clusters on Principal Components
133     output$clusterpc = renderPlot({
134
135       # Dependency on input$ctest_start
136       if (input$cnum_start == 0) { return() }
137
138       ggplot2::autoplot(listings_pc, data = k_means(), colour = "cluster") +
139         labs(color = "clusters") +
140         theme_bw()
141
142     })
```

```

}
```

**Listing 50: |app.R|**

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

# Install and load needed packages
needed_packages <- c("shiny",
                     "shinydashboard",
                     "leaflet",
                     "shinyWidgets",
                     "colorspace",
                     "htmltools",
                     "rstudioapi",
                     "devtools",
                     "lindia",
                     "tidyverse",
                     "Jmisc",
                     "stringr")
for (package in needed_packages) {
  if (!require(package, character.only=TRUE)) {
    install.packages(package, character.only=TRUE)}
  library(package, character.only=TRUE)
}
devtools::install_github('andrewsali/shinycssloaders')
library("shinycssloaders")
rm("needed_packages", "package")

# Set working directory to the one where the file is located
# setwd(dirname(sys.frame(1)$ofile)) # This works when sourcing
setwd(dirname(rstudioapi::getActiveDocumentContext()$path)) # This works
  when running the code directly

# Load helper functions and other scripts
load("workspace_for_shinyapp.RData")
Jmisc::sourceAll(file.path(getwd(), "Helpers", fsep="/"))
Jmisc::sourceAll(file.path(getwd(), "Server", fsep="/"))
source("ui.R")
source("server.R")

# 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