Humboldt-Universität zu Berlin

School of Business and Economics

Ladislaus von Bortkiewicz Chair of Statistics

Statistical Programming Languages

Seminar paper

# Exploratory Data Analysis of airbnb listings in Berlin

*Silvia Ventoruzzo*

*(592252)*

submitted to

Alla Petukhina

March 11, 2019

# Contents

**1 Introduction** **1**

**2 Data Preparation** **2**

2.1 Berlin neighbourhoods and districts . . . . . 2

2.2 Berlin VBB Zones . . . . . 4

2.3 Airbnb listings' attributes . . . . . 7

**3 Exploratory Data Analysis** **10**

3.1 Descriptive statistics . . . . . 10

3.2 Distribution plots . . . . . 12

**4 Price analysis** **14**

4.1 Correlation with price . . . . . 14

4.2 Linear regression on price . . . . . 15

**5 Clustering** **16**

**6 ShinyApp** **16**

**7 Results and Conclusions** **18**

**References** **19**

## 1 Introduction

- What is the subject of the study? Describe the economic/econometric problem.
- What is the purpose of the study (working hypothesis)?
- What do we already know about the subject (literature review)? Use citations: *Gallant (1987) shows that... Alternative Forms of the Wald test are considered (Breusch and Schmidt, 1988).*
- What is the innovation of the study?
- Provide an overview of your results.
- Outline of the paper:

  *The paper is organized as follows. The next section describes the model under investigation. Section* **??** *describes the data set and Section* **??** *presents the results. Finally, Section 7 concludes.*
- The introduction should not be longer than 4 pages.

## 2 Data Preparation

For the analysis explained in this paper data was downloaded for a website independent from airbnb itself. insideairbnb (?) scrapes airbnb to get its data, posts it online for the public to use on own analysis, while also providing some analysis of its own.
The data is divided according to cities and for each there is general information about the city's properties and their availability for the next year. For this analysis not all variables are being kept, the focus is indeed on the ones that might have the most affect on price. Moreover, feature engineering will also be performed to extract even more useful information. More details in subsection 2.1.
Since we are dealing with data with coordinates, spatial data is also needed to produce a map of the location. For this purpose further data has been downloaded from the Statistics Office of Berlin-Brandenburg (?) and Geofabrik (?) and further processed, as explained in subsections 2.1 and 2.2.
To handle data will be make use of functions from the different packages in the `tidyverse`, which allows for easy manipulation of the data and flexible plotting, see Ross et al. (2017). For spatial data the package `sf` has been chosen, since it works well with the `tidyverse` packages and for all the other reasons listed in Pebesma (2018).

### 2.1 Berlin neighbourhoods and districts

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

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

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

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

**Listing 1: |berlin_districts_neighbourhoods.R|**

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

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

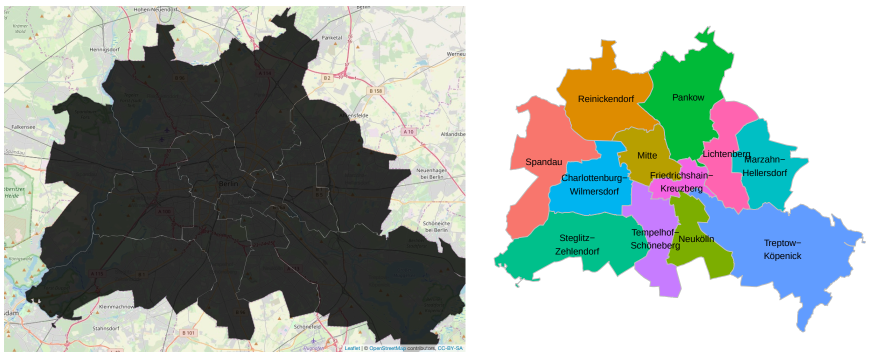
```r
berlin_neighbourhood_singlebuckow_sf = berlin_neighbourhood_sf %>%
    dplyr::group_by(id, group) %>%
    dplyr::summarize(do_union = TRUE)
```

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|**

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

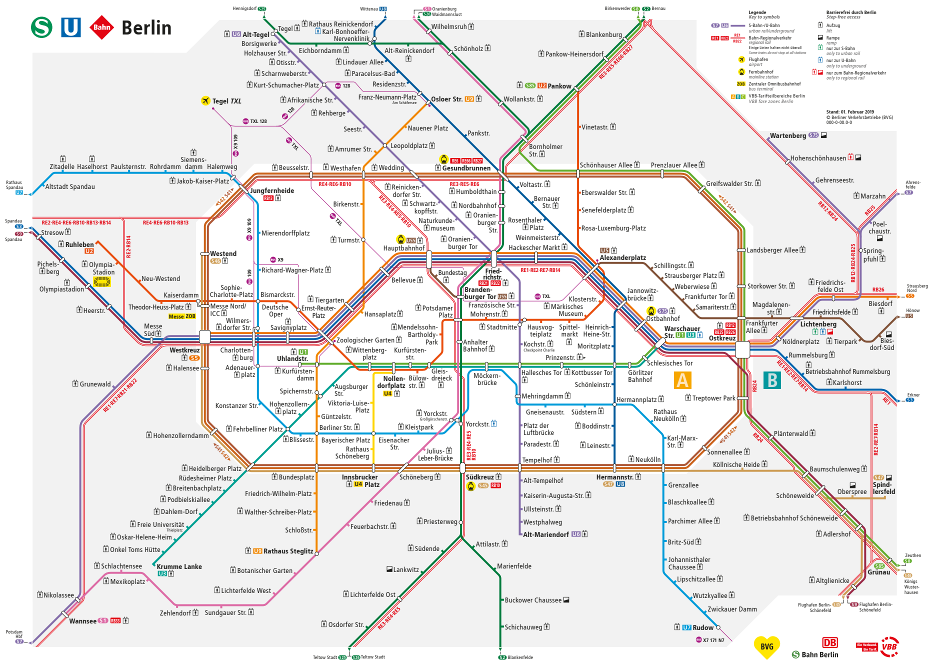


**Figure 2:** Network Map of Berlin Areas A and B (from VBB Website)

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

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

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

**Listing 4: |berlin_vbb_zones.R|**

```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|**

```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|**

```r
berlin_sf = berlin %>%
    dplyr::summarize(do_union = TRUE)
```

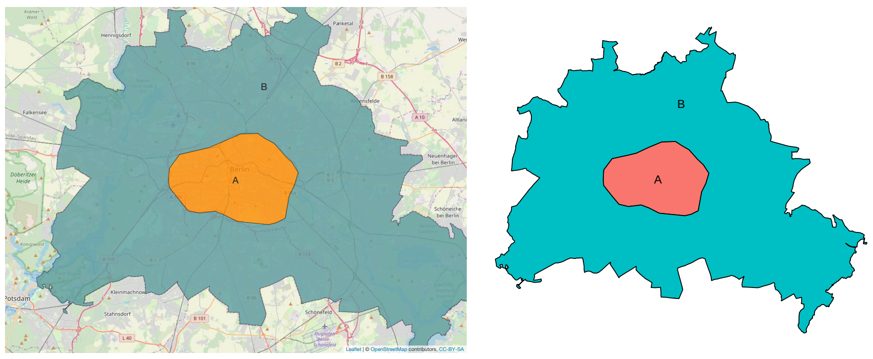
Finally, we bind the two objects by row and calculate their intersections thanks to the function `st_intersection` from the package `sf`. We then define the area names according to how many times the two previous polygons intersect:

- 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|**

```r
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))
```

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|**

```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:

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

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

**Listing 9: |distance_count.R|**

```
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
```

```r
    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$id))
    # Calculate how many "reference" are within "distance"
    point_distance[,var_name_count] = rowSums(point_distance <= distance)
    # Calculate the distance to the nearest "reference"
    point_distance[,var_name_dist] = apply(point_distance
                                      [,-ncol(point_distance)],
                                      MARGIN = 1,
                                      FUN = min) %>%
                                  round(0)
    # Insert this information into the main DF
    main = point_distance %>%
        dplyr::mutate(id = main$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: |airbnb_listings.R|

```r
listings_season_availability = listings_calendar %>%
  dplyr::group_by(id, year_season) %>%
  # if loading the csv use sum(available)
  # dplyr::summarize(count = sum(available)) %>%
  # if loading the zip use sum(available == TRUE)
  dplyr::summarize(count = sum(available == TRUE)) %>%
  dplyr::arrange(id, year_season) %>%
  dplyr::ungroup() %>%
  dplyr::mutate(season_av =  paste(year_season, "availability", sep = "_")
    %>%
           tolower() %>%
           factor(levels = unique(tolower(paste(year_season, "availability",
              sep = "_"))))) %>%
  dplyr::select(-year_season) %>%
  tidyr::spread(season_av, count)
```

## 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 2.

| variable | min | 1Q | median | 3Q | max | iqr | mean | sd |
|---|---|---|---|---|---|---|---|---|
| price | 0.00 | 30.00 | 45.00 | 70.00 | 9000.00 | 40.00 | 67.14 | 220.28 |

**Table 2:** Sample of descriptive table for numeric variables

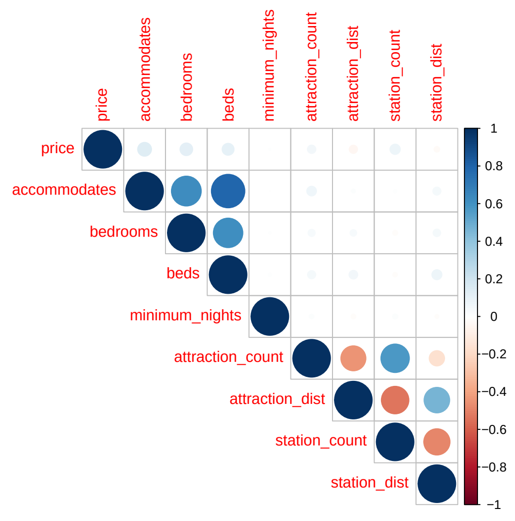
For categorical variables the explained statistics do not work, therefore frequencies and proportions of each factor were calculated.

**Listing 12: |descriptive_statistics.R|**

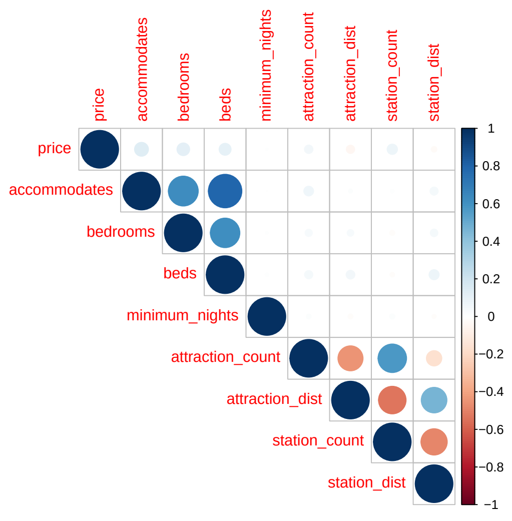
```
  } else if (unique(apply(df, 2, function(x) is.character(x) | is.factor(x)
    | is.logical(x)))) {

    # 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")

    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))
```

## 3.2 Distribution plots

Also for the distribution plots we distinguish between numerical and categorical variables. In the first case a density plot has been produced, where also mean, median, 1st and 3rd quantiles are visible. Unfortunately, many variables present outliers, which were in some cases excluded from the plots for better visualization.



**(a)** Complete



**(b)** Without outliers

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

For the categorical variables a bar plot is more appropriate, since the values that the variable can assume are discrete and usually also few, like in the case displayed in figure 8.

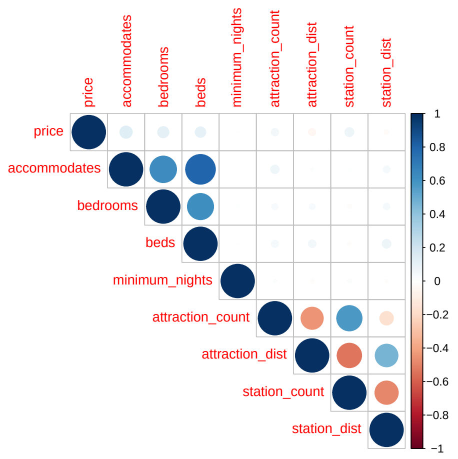


**Figure 5:** Distribution of the variable room_type exploratory_data_analysis.R

We also mapped the Berlin districts and the distribution of the average of a certain variable on them with the use of the package *leaflet*.

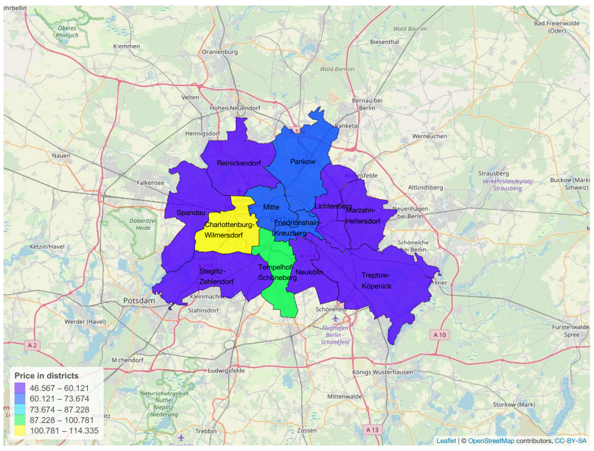


**Figure 6:** Distribution of the average of price across Berlin's districts

## 4 Price analysis

One of the mean factors in the choice of the property to book is its price. Therefore one may want to try and see what other 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 normal correlation plot like the one in figure 7 may not be the most easily readable in this case, especially because of the large number of variables.

– UNDERSTAND HOW TO PUT IMAGE embedded in text

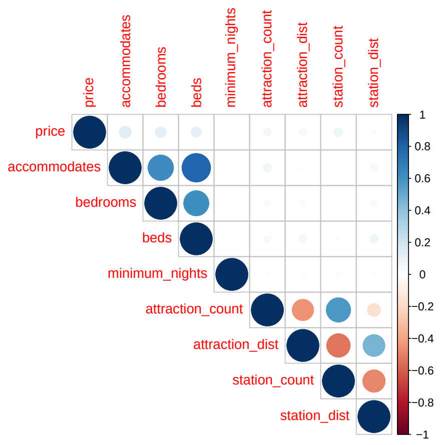


**Figure 7:** Correlation plot with the function *corrplot* from the package *corrplot*

In fact, categorical variables firstly need to be transformed into many dummy variables in order to calculate the correlation.

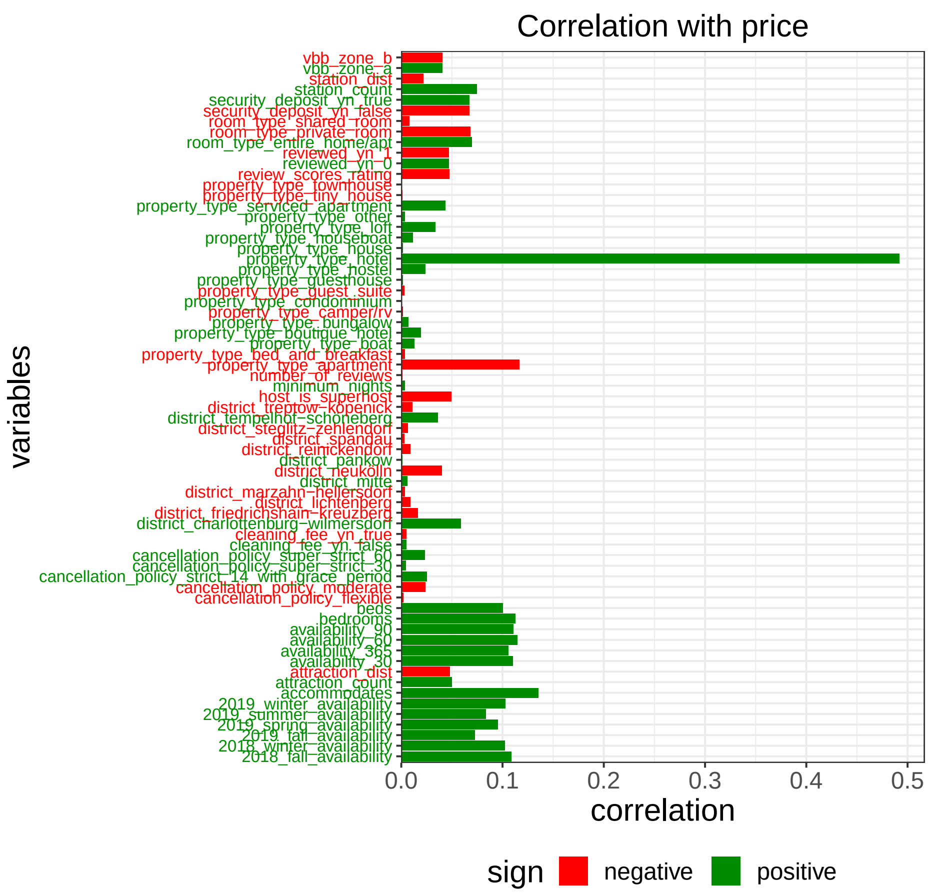


**Figure 8:** Plot of correlation with price

## 4.2 Linear regression on price

## 5 Clustering

- Organize material and present results.
- Use tables, figures (but prefer visual

## 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)`.

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

```r
ui = fluidPage(
    titlePanel("Hello Shiny!"),
    sidebarLayout(
      sidebarPanel(sliderInput(inputId = "bins",
                               label = "Number of bins:",
                               min = 1, max = 50, value = 30)),
      mainPanel(plotOutput(outputId = "distPlot")))
)

server = function(input, output) {
    output$distPlot = renderPlot({
      x     = faithful$waiting
      bins = seq(min(x), max(x), length.out = input$bins + 1)
      hist(x, breaks = bins, col = "#75AADB", border = "white",
           xlab = "Waiting time to next eruption (in mins)",
           main = "Histogram of waiting times")})
}

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

This simple App produces the result in figure 9.

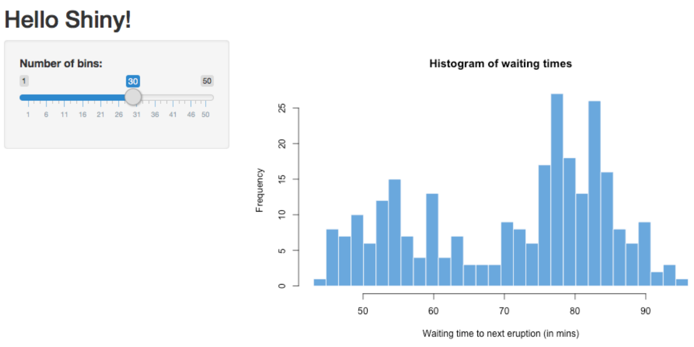


**Figure 9:** Example of simple ShinyApp from website

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 App is reachable at this link (INSERT SHINY APP LINK).

## 7 Results and Conclusions

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

## References

Breusch, T. S. and P. Schmidt (1988): “Alternative Forms of the Wald test: How Long is a Piece of String,” *Communications in Statistics, Theory and Methods*, 17, 2789–2795.

Gallant, A. R. (1987): *Nonlinear Statistical Models*, New York: John Wiley & Sons.

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.

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.

## Declaration of Authorship

I hereby confirm that I have authored this Bachelor's/Master's thesis 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, September 30, 2007

your name (and signature, of course)