



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 9, 2019

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Data Preparation</b>	<b>2</b>
2.1	Berlin neighbourhoods and districts . . . . .	2
2.2	Berlin VBB Areas . . . . .	4
2.3	Airbnb listings' attributes . . . . .	7
<b>3</b>	<b>Exploratory Data Analysis</b>	<b>10</b>
3.1	Descriptive statistics . . . . .	10
3.2	Distribution plots . . . . .	11
<b>4</b>	<b>Price analysis</b>	<b>14</b>
4.1	Correlation with price . . . . .	14
4.2	Linear regression on price . . . . .	15
<b>5</b>	<b>Clustering</b>	<b>16</b>
<b>6</b>	<b>ShinyApp</b>	<b>17</b>
<b>7</b>	<b>Results and Conclusions</b>	<b>19</b>
	<b>References</b>	<b>20</b>

# 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 (insideairbnb.com) scrapes (???) airbnb to get its data and 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. The variables that are being kept for this analysis are listing on table 1. (HOW TO MAKE IT CHANGE NUMBER???)

Even though the scraped data has already been partially .... from ....., not all information is necessary for our analysis and some feature engineering is needed.

Moreover, the spatial data from the downloaded shapefiles also requires to be reprocessed.

### 2.1 Berlin neighbourhoods and districts

Berlin consists of 96 neighbourhoods (Ortsteile), which are grouped into 12 districts (Bezirke).

The polygons to plot them are extracted from the relative shapefile which is loaded with the function `st_read` from the `sf` package to have it already as a `sf` polygon.

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

```
1 # This works when sourced
```

The types of objects used by and created with this package come in very handy since they look like data frames and many functions for data frames can be used on them.

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:** Summary of Berlin's `sf` object

Since the polygons represent the neighbourhoods, we do not need to perform any transformation on this object. Here we keep only the variables of interest, rename them and reorder

the rows.

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

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

However, we have the problem with the neighbourhood Buckow, is composed of two separate parts. Therefore we need to unite the neighbourhoods according to their name. In this way we obtain an sf object with 96 polygons, the one of Buckow being a list of polygons.

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

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

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



(a) With leaflet

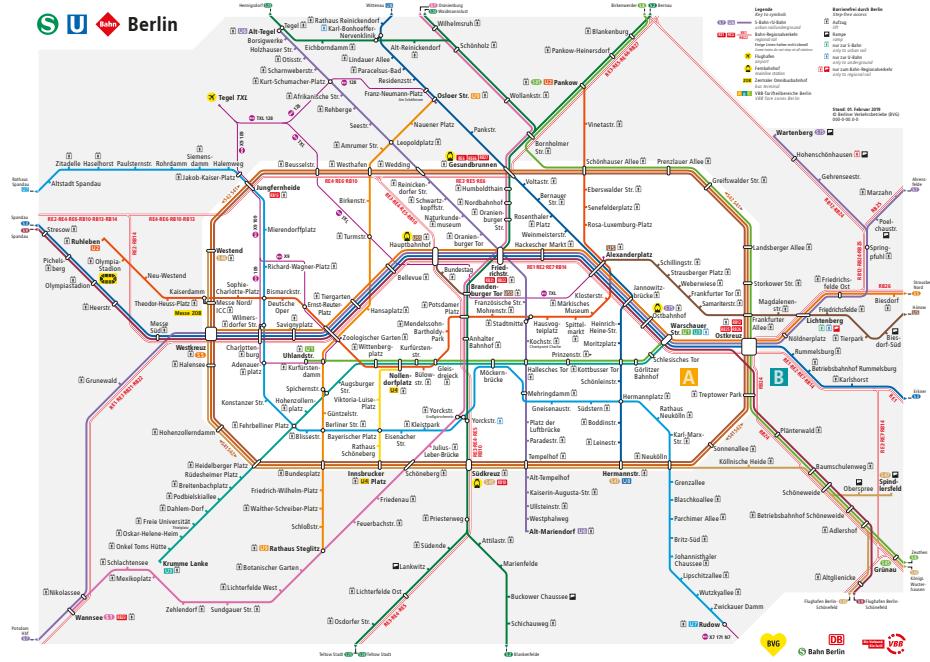


(b) With ggplot

Figure 1: Maps of the Berlin Districts  berlin\_districts\_neighbourhoods\_maps.R

## 2.2 Berlin VBB Areas

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 Ringbahn (circular line), 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 using the Berlin polygons and the stations points, which are again loaded using `st_read` from the `sf` package.

**Listing 4:** `|berlin_vbb_areas.R|`

```
1
2 # Load helpers
```

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

Firstly, we filter the stations that belong to the Ringbahn (border of area A). Since the shapefile does not include the line name for the stations, we need to create our own vector of names. We also add the order in which they need to be connected. The first and last station

are the same since the circle need to close.

Listing 5: |berlin\_vbb\_areas.R|

```
1 # Load shapefiles
2 berlin = sf::st_read(file.path(getwd(), "Data", "Berlin-Ortsteile-polygon.
   shp", fsep="/"))
3 stations = sf::st_read(file.path(getwd(), "Data", "gis_osm_transport_free_1.
   shp", fsep="/"))
4
5 # Create dataframe with names of stations on the Ringbahn (delimits Area A)
6 ringbahn_names_df = base::data.frame(
7     id = c("Südkreuz", "Schöneberg", "Innsbrucker Platz", "Bundesplatz",
8           "Heidelberger Platz", "Hohenzollerndamm", "Halensee", "Westkreuz",
9           "Messe Nord/ICC", "Westend", "Jungfernheide", "Beusselstraße",
10          "Westhafen", "Wedding", "Gesundbrunnen", "Schönhauser Allee",
11          "Prenzlauer Allee", "Greifswalder Straße", "Landsberger Allee",
12          "Storkower Straße", "Frankfurter Allee", "Ostkreuz", "Treptower
       Park",
```

Since some stations appear multiple time, being both subway and lightrail stations (and maybe even bus and tram stops), we 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.

Listing 6: |berlin\_vbb\_areas.R|

```
1     stringsAsFactors = FALSE) %>%
2     tibble::rownames_to_column(var = "order") %>%
3     dplyr::mutate(order = as.numeric(order))
4
5 # Create sf object of Area A
6 berlin_vbb_A_sf = stations %>%
7     dplyr::filter(fclass %like% "railway") %>%
8     dplyr::rename(id = name) %>%
9     dplyr::mutate(id = gsub("Berlin ", "", id),
```

By performing a right join with the dataframe with the Ringbahn station names, we only keep these stations. After performing some preparation steps, we use the function `st_polygon` from the `sf` package, which creates a polygon out of a list of points (CHECK!!!).

Listing 7: |berlin\_vbb\_areas.R|

```
11     id = gsub("Berlin-", "", id),
```

```

12         id = gsub(" *\\(.*?\\) *", "", id),
13         id = gsub("S ", "", id),
14         id = gsub("U ", "", id)) %>%
15     points_midpoint() %>%
16     dplyr::right_join(ringbahn_names_df, by = "id") %>%
17     dplyr::arrange(order) %>%
18     dplyr::select(long, lat) %>%

```

Secondly we need to create a polygon for entire Berlin. This is done by simply uniting all the neighbourhoods.

**Listing 8: |berlin\_vbb\_areas.R|**

```

19     sf::st_polygon() %>%
20     sf::st_sfc() %>%

```

Finally, we bind the two objects by rows and calculate their intersections thanks to the function `st_intersection`. 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 9: |berlin\_vbb\_areas.R|**

```

21 # Create sf object of entire Berlin
22 berlin_sf = berlin %>%
23     dplyr::summarize(do_union = TRUE)
24
25 # Bind and intersect to create sf object with both VBB Zones (A and B)
26 berlin_vbb_AB_sf = berlin_vbb_A_sf %>%

```

In this code the custom-function `points_midpoint` was used to calculate the middle point among many. It extracts the coordinates of the points thanks to `st_coordinates` and then calculates the mean of latitude and longitude.

**Listing 10: |points\_midpoint.R|**

```

1 points_midpoint <- function(points_sf, point_name = "id") {
2
3     # Create symbol version of point_name
4     point_name <- sym(point_name)
5

```

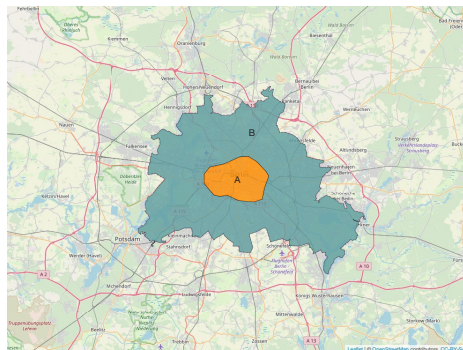


```

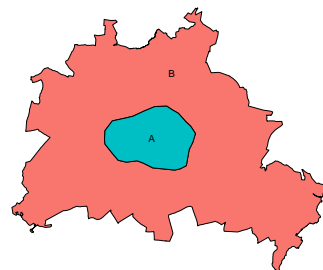
6  # Get coordinates from sf point objects
7  coordinates <- sf::st_coordinates(points_sf) %>%
8    base::as.data.frame() %>%
9    dplyr::rename(lat = Y,
10                 long = X)
11
12 # Bind coordinates with sf point object and calculate mean of lat and long
13 midpoints_df <- points_sf %>%
14   base::as.data.frame() %>%
15   dplyr::select(-geometry) %>%
16   base::cbind(coordinates) %>%
17   dplyr::group_by(!point_name) %>%
18   dplyr::summarize(lat = mean(lat),
19                   long = mean(long))
20
21 return(midpoints_df)
22
23 }

```

This 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 datasets consists in joining the two 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 proceed to feature engineering. We firstly derive the areas where they are located thanks to the spatial polygons created before and the function

point\_in\_polygons.

Listing 11: |point\_in\_polygons.R|

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

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.

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)

Listing 12: `|distance_count.R|`

```

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

```

This function firstly calculates the distance between all properties and all reference points, railway stations or tourist attractions, using the Haversine Formula, which "gives minimum distance between any two points on spherical body by using latitude and longitude" (?).

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

### 3 Exploratory Data Analysis

The exploratory data analysis will be divided in two parts: firstly, summary statistics are calculated for both numeric and categorical variables; secondly, density plots are produced for numerical variables and bar plots for categorical variables. Correlation will also be calculated, but only with respect to price. This subject will be dealt with in subsection ??.

#### 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 (?).

As in ? explained there are the general types of descriptive statistics:

- Measures of central tendency
- Measures of spread
- Graphical displays

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

Most of these statistics are usually applied to continuous data and even to numerical discrete data. For this type of data the following statistics of central tendency were calculated:

- $mean = \frac{1}{N} \sum_{i=1}^N x_i$
- 

The median is also the 2nd quantile.

As for spread, we can derive the range from the minimum and maximum  $range = max - min$  and we calculate the 1st and 3rd quantile (the 2nd ist just the median) and their difference  $IQR = 3Q - 1Q$ . They correspond to the 25% and 75% of the values. Finally we also derive the standard deviation, which is the squared root of the variance  $variance = .....$  (FORMULA OF VARIANCE)

CANNOT LOAD THE RIGHT CODE :(

Listing 13: |summary\_stat\_num.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 }

```

With this code one can calculate the descriptive statistics for one variable on the basis of all the data or, if both *constraint\_var* and *constraint\_var* are set, on the basis of some constraint on the data.

This function will then be applied to all the numerical variables to give a result such as in table2.

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.

CANNOT LOAD THE RIGHT CODE :(

**Listing 14:** |summary\_stat\_fact.R|

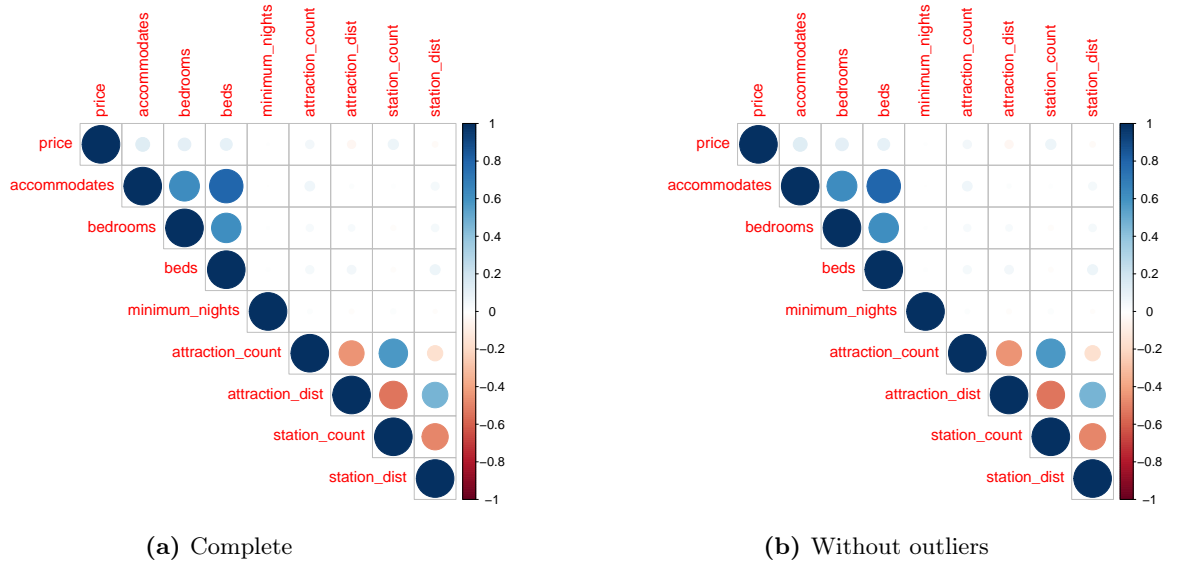
```


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 }

```

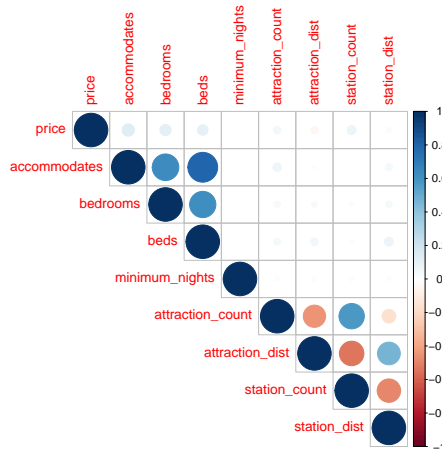
## 3.2 Distribution plots

Also for the distribution plots we distinguish between numerical and categorical variables. In the first case a frequency 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.



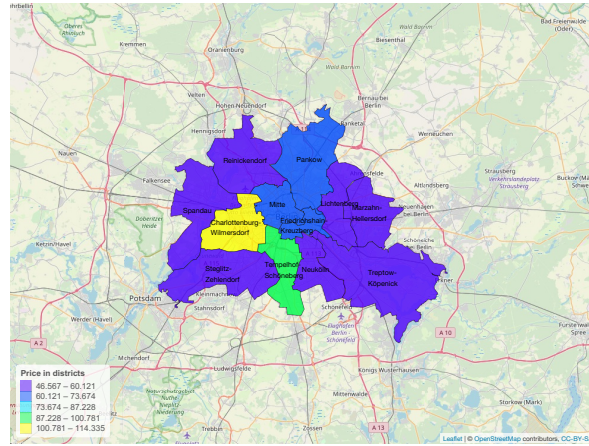
**Figure 4:** Distribution of the variable price  exploratory\_data\_analysis.R

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

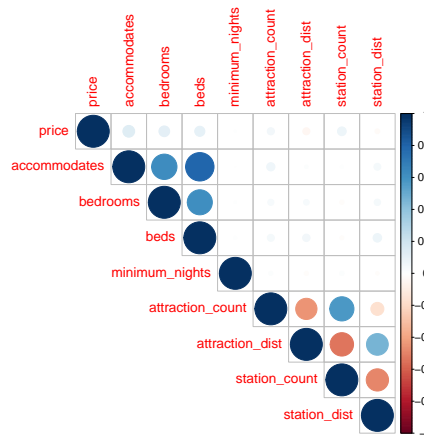
One of the main factors in the choice of the property to book is its price. Therefore one may want to try and see what other attributes influence its value.

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.



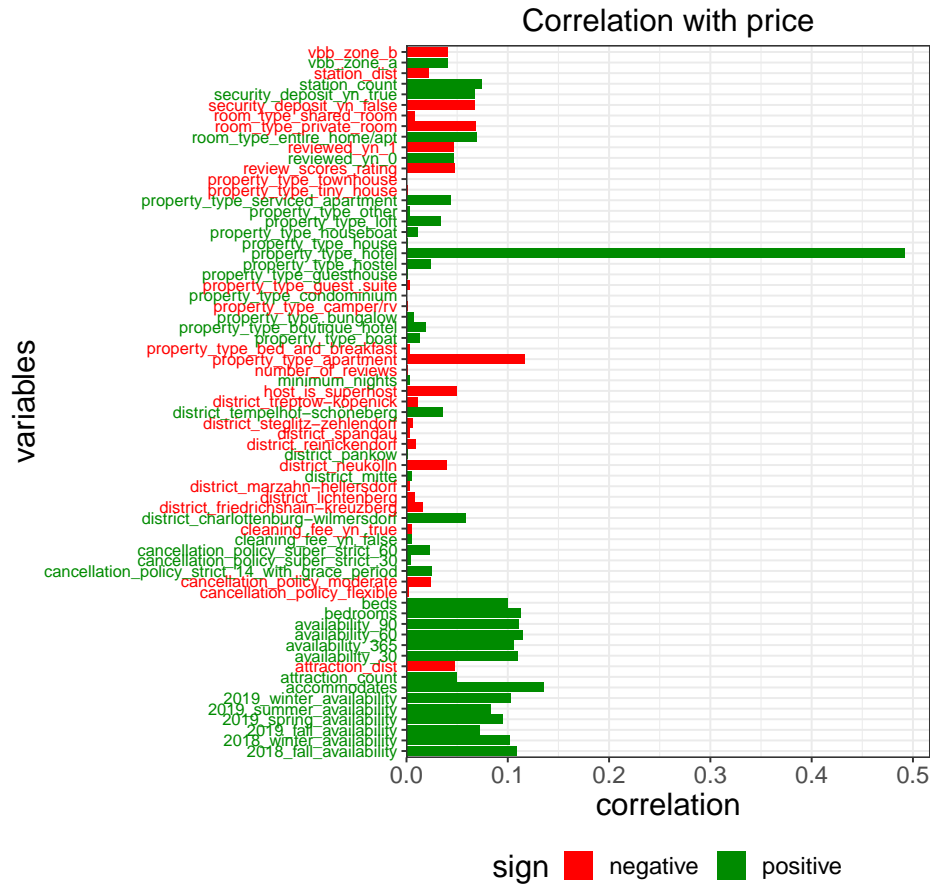


Figure 8: Plot of correlation with price

## 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-friendly way.

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

The App structure is divided into two main elements:

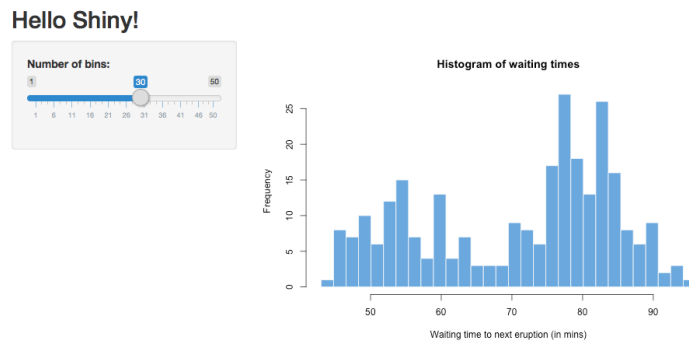
- *ui*: The user interface object determines the appearance of the App itself. Here the possible inputs and the outputs will be defined.
- *server*: The server function uses the input values chosen in the App to produce the outputs indicated in the user interface.

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

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

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

This simple App produces the result in figure 9.



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

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

## 7 Results and Conclusions

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

## References

- 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.

## **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)