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 5, 2019

Contents

List of Abbreviations

CPI	Consumer Price Index	ETF	Equity Traded Funds
ETH	Eat the Horse	XLM	Xetra Liquidity

List of Figures

List of Tables

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 2 describes the data set and Section 5 presents the results. Finally, Section 6 concludes.
- The introduction should not be longer than 4 pages.

2 Data

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???).

3 Data Preparation

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.

3.1 Berlin neighbourhoods and districts

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

Neighbourhood	District
Charlottenburg	Charlottenburg-Wilmersdorf
Wilmersdorf	Charlottenburg-Wilmersdorf
Grunewald	Charlottenburg-Wilmersdorf
Westend	Charlottenburg-Wilmersdorf
Schmargendorf	Charlottenburg-Wilmersdorf
Charlottenburg-Nord	Charlottenburg-Wilmersdorf
Halensee	Charlottenburg-Wilmersdorf
Friedrichshain	Friedrichshain-Kreuzberg
Kreuzberg	Friedrichshain-Kreuzberg
Friedrichsfelde	Lichtenberg
Karlshorst	Lichtenberg
Malchow	Lichtenberg
Wartenberg	Lichtenberg
Falkenberg	Lichtenberg
Fennpfuhl	Lichtenberg
Lichtenberg	Lichtenberg
Neu-Hohenschönhausen	Lichtenberg
Alt-Hohenschönhausen	Lichtenberg
Rummelsburg	Lichtenberg
Marzahn	Marzahn-Hellersdorf
Biesdorf	Marzahn-Hellersdorf
Kaulsdorf	Marzahn-Hellersdorf
Mahlsdorf	Marzahn-Hellersdorf
Hellersdorf	Marzahn-Hellersdorf
Mitte	Mitte
Moabit	Mitte
Hansaviertel	Mitte
Gesundbrunnen	Mitte
Tiergarten	Mitte
Wedding	Mitte
Buckow	Neukölln
Buckow	Neukölln
Gropiusstadt	Neukölln
Neukölln	Neukölln
Britz	Neukölln
Rudow	Neukölln

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|

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

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	

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|

```
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)
```

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|

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

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.

3.2 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.

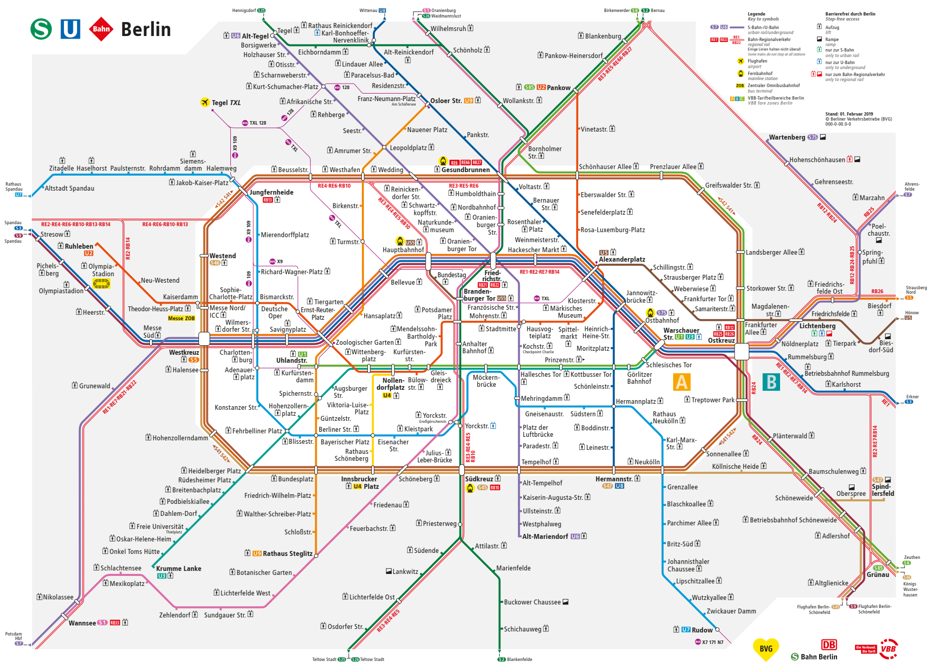

Figure 1: Network Map of Berlin Areas A and B (from VBB Website)

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|

```
berlin = sf::st_read(file.path(getwd(), "spatial_data", "Berlin-Ortsteile-
  polygon.shp", fsep="/"))
stations = sf::st_read(file.path(getwd(), "spatial_data", "gis_osm_transport
  _free_1.shp", fsep="/"))
```

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|

```
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 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|

```
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() %>%
```

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|

```
    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))
```

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|

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

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 ≤ 1)

Listing 9: |berlin_vbb_areas.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))
```

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|

```r
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))

  return(midpoints_df)

}
```

This sf object can be used to map the VBB zones using `leaflet` or `ggplot2`.

something something

3.3 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|

```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)
}
```

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|

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

    return(main)
}
```

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" (?).

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

4 Functions

- From package `sf`:

- st_read
- st_coordinates
- st_intersection
-
-

-
-
-
-

5 Results

- Organize material and present results.
- Use tables, figures (but prefer visual presentation):
  - Tables and figures should supplement (and not duplicate) the text.
  - Tables and figures should be provided with legends.

    Figure 2 shows how to include and reference graphics. The graphic must be labelled before. Files must be in `.eps` *format.*

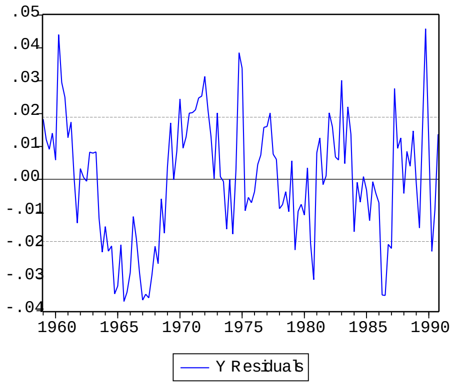

Figure 2: Estimated residuals from model XXX. ...

  - Tables and graphics may appear in the text or in the appendix, especially if there are many simulation results tabulated, but is also depends on the study and number of tables resp. figures. The key graphs and tables must appear in the text!

- Latex is really good at rendering formulas:

  Equation (2) represents the ACs of a stationary stochastic process:

$$f_y(\lambda) = (2\pi)^{-1} \sum_{j=-\infty}^{\infty} \gamma_j e^{-i\lambda j} = (2\pi)^{-1} \left(\gamma_0 + 2 \sum_{j=1}^{\infty} \gamma_j \cos(\lambda j) \right) \tag{2}$$

  where $i = \sqrt{-1}$ is the imaginary unit, $\lambda \in [-\pi, \pi]$ is the frequency and the γ_j are the autocovariances of y_t.

- Discuss results:
  - Do the results support or do they contradict economic theory ?
  - What does the reader learn from the results?
  - Try to give an intuition for your results.
  - Provide robustness checks.
  - Compare to previous research.

6 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)