HAW HAMBURG

Bachelor Thesis

Ivan Havrylenko

Validation of Crowdsourced Apartment Data for Data Veracity

Faculty of Computer Science and Engineering
Department of Information and Electrical Engineering

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Ivan Havrylenko

Validation of Crowdsourced Apartment Data for Data Veracity

Bachelor Thesis based on the examination and study regulations
for the Bachelor of Engineering degree programme
Bachelor of Science Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg
Supervising examiner: Prof. Dr. Heike Neumann
Second examiner: Prof. Dr. Annabella Rauscher-Scheibe

Day of delivery: 10. November 2022

Ivan Havrylenko

Title of Thesis

Validation of Crowdsourced Apartment Data for Data Veracity

Keywords

OpenStreetMap, Geodata, Nominatim, Validation, Crowdsourcing, Supervised Learning, Unsupervised Learning, K-Means, Multivariate Normal Distribution

Abstract

A problem of crowdsourcing data lies in assuring that anonymous users do not accidentally or intentionally provide false information. Therefore, a well-performing method for validating the users' information is vital for every purpose for which the data was originally collected. This thesis investigates different data validation techniques for crowdsourced data about the housing situation of residents of Hamburg in Germany. The results of this thesis are of special interest to the local initiative of "Hamburg Enteignet" who are, among other things, aiming to shed light on Hamburg's current housing market. Various data validation approaches were implemented to make sure that provided apartment data (address, cold rent and area) is likely to originate from an actual apartment in Hamburg in contrast to being fabricated. Apartment addresses were validated by cross-referencing the user's provided data with the open APIs Nominatim and OpenStreetMap. For cold rent and apartment area we considered K-Means clustering and threshold based approaches using standard deviation as well as a multivariate normal distribution. The combination of Nominatim API and OpenStreetMap for cross-referencing addresses, correctly determined the validity of about 91% of 2049 test addresses. The threshold based multivariate normal distribution approach performed best for classifying apartments based on cold rent and area with about 90% and 87% of 1143 test apartments correctly determined to be real and fake, respectively.

Ivan Havrylenko

Thema der Arbeit

Validierung von Crowdsourcing-Wohnungsdaten auf Datenrichtigkeit

Stichworte

OpenStreetMap, Geodata, Nominatim, Validierung, Crowdsourcing, Überwachtes Lernen, Unüberwachtes Lernen, K-Means, Mehrdimensionale Normalverteilung

Kurzzusammenfassung

Ein Problem beim Crowdsourcing von Daten besteht darin, sicherzustellen, dass anonyme Nutzer nicht versehentlich oder absichtlich falsche Angaben machen. Für die Vorhaben, für die die Daten ursprünglich erhoben wurden, ist daher eine gut funktionierende Methode, zur Validierung der Nutzerinformationen, unerlässlich. In dieser Arbeit werden verschiedene Datenvalidierungstechniken für Crowdsourced-Daten über die Wohnsituation von Bürgern der Stadt Hamburg, Deutschland, untersucht. Die Ergebnisse dieser Arbeit sind von besonderem Interesse für die lokale Initiative "Hamburg Enteignet", die u.a. Tranzparenz für den aktuellen Hamburger Wohnungsmarkt schaffen will. Um sicherzustellen, dass die bereitgestellten Wohnungsdaten (Adresse, Kaltmiete und Fläche) mit hoher Wahrscheinlichkeit aus einer realen Wohnung in Hamburg stammen und nicht gefälscht sind, wurden verschiedene Ansätze zur Datenvalidierung eingesetzt. Die Wohnungsadressen wurden durch Abgleich der vom Nutzer bereitgestellten Daten mit den offenen APIs Nominatim und OpenStreetMap validiert. Für Kaltmiete und Wohnungsfläche wurden K-Means-Clustering und schwellenwertbasierte Ansätze unter Verwendung der Standardabweichung sowie einer multivariaten Normalverteilung berücksichtigt. Durch die Kombination von Nominatim API und OpenStreetMap zum Abgleich von Adressen wurde die Gültigkeit von etwa 91% der 2049 Testadressen korrekt ermittelt. Der auf Schwellenwerten basierende Ansatz der multivariaten Normalverteilung schnitt bei der Klassifizierung von Wohnungen auf der Grundlage von Kaltmiete und Fläche am besten ab, wobei etwa 90% bzw. 87% von 1143 Testwohnungen korrekt als echt bzw. gefälscht eingestuft wurden. Insgesamt wurde festgestellt, dass der Adressabgleich mit Nominatim API und OpenStreetMap zusammen mit einem schwellenwertbasierten multivariaten Normalverteilungsansatz für Kaltmiete und Fläche angemessen ist, um im Kontext von Wohnungsmarktanalysen die Verlässlichkeit anonym bereitgestellter Wohnungsdaten zu erhöhen.

Contents

List of Figures

List of Tables

1 Introduction

An initiative in the city of Hamburg regarding the housing market aimed to evaluate whether it would be sensible to expropriate apartments from the largest real estate companies to battle the ever increasing rents for residents of Hamburg. This initiative is called "Hamburg Enteignet" inspired by a similar intiative from Berlin with the same goal "Deutsche Wohnen Enteignen" which already resulted in a "Beschlussvolksentscheid" which basically means that the Berlin senate now has to check whether the housing market expropriation is lawful [17] (which most people think it should be based on Article 15 of Basic Law for the Federal Republic of Germany [2]).

The "Hamburg Enteignet" initiative explained the publication of the Hamburg Rent Index 2021 by the Senate, which showed a rent increase of 7.3% in the last two years, where they underlined that the 7.3% increase represents thousands of residents, who can no longer afford living in Hamburg. This increase results in the displacement of residents to the outskirts of the city, to the countryside and accelerates the rise of homeless people, whilst the real estate agencies are receiving profits [19].

The "Hamburg Enteignet" initiative seeks a referendum for the expropriation and socialization of the large, profit-oriented real estate agencies on the basis of Article 15 of the Basic Law [2].

The project for "Hamburg Enteignet" is planning to have a user interface (UI), which gives the opportunity for all residents of Hamburg to provide the their apartment data like cold rent, apartment area, address in a form of crowd-sourcing. Here lies the problem of validating the provided data. How to prove that the provided data is legitimate? So the methodology for validation of user entered data is required. The project is aiming to find a way how to validate the apartment properties entered by a user. The first goal of the work is to find a way how to check, that the address, provided by a user, is actually real. This address may be real, but is it a residential address, or is it a part of an industrial or shopping area? Following this analysis, the project aims to find a suitable solution

to check the values of cold rent and apartment area, if they fit the trend in Hamburg city. A malicious actor could input wrong cold rent or area data which would skew the data foundation the analysis is based on. To prevent this, statistical methods together with supervised and unsupervised machine learning algorithms could be employed which would give the opportunity to generate distributions of the cold rent and area for analysis of how good the user entered data fits the aforementioned distributions.

2 Theoretical Basics

In this chapter we will discuss the most essential technical details, which are needed to implement a proper data validation approach. This chapter covers the theory behind OpenStreetMap project and its Application Programming Interfaces (API); various Goodness-of-Fit tests, which can be used to check the normality of data distribution; gradient ascend and descend as used in machine learning and K-Means clustering, which allows data partitioning into separate groups.

2.1 OpenStreetMap as a main data source

Throughout many various Volunteered Geographic Information (VGI) sources open for the public usage the most common one is considered to be OpenStreetMap (OSM). In nearly 10 years OSM has developed to be the leader amongst available VGIs comprising not only the geographical database but also software applications, APIs and information storage analogous to wikis, [9]. OSM is used in numerous applications starting from car navigation systems, bicycle maps and even digitalization of humanitarian aids, [5]. The OSM improving project's data and growing community have the opportunity to provide accurate and detailed geodata and maps from an image to a dynamic map of the world for everyone without any restrictions, [24]. In this chapter a general overview of OpenStreetMap is given as well as the reasons why this Volunteered Geographic Information source is considered to be up-to-date.

2.1.1 Overview of OpenStreetMap project

OpenStreetMap is a collaborative project aimed at collecting, storing and sharing publicly available geo-data, creating tools for working with them by the volunteer community. The project was founded in the UK in July 2004 by Steve Coast. In April 2006, the OpenStreetMap Foundation was registered as an international non-profit organization

created to support the development and dissemination of geospatial data, as well as to enable the use of geospatial data by anyone, [33]. The idea behind creating OSM was relatively straightforward and based on crowd-sourcing collaboration. This means that a user collects the local knowledge of the particular area where they live and share this information. Then the other user collects their local geographic data and combines it with the aforementioned one. This approach leads to a crowd data collection scaling which involves many users around the globe similarly to Wikipedia, [16]. The availability of Global Positioning System (GPS) has quite a solid impact on the OSM data contribution, since the registered OSM users can easily, using their smart devices as smartphones, collect geographic data which then can be uploaded to OSM. For instance major road data is usually obtained from tracks recorded by GPS receivers or GPS trackers. Such tracks are created by volunteers while moving around the study area. The tracks are then exported from the GPS device to the OSM map editor, [33].

2.1.2 OpenStreetMap data formats

A significant amount of data loaded into OSM is downloaded from portable satellite navigation or monitoring devices. The GPSBabel program can be used to convert coordinates from raw (NMEA) or proprietary formats to GPX (XML based). OpenStreetMap uses a topological data structure consisting of objects:

- node (point) - a point with the specified coordinates;
- way (line) - an ordered list of points that make up a line or polygon;
- relation (relation) - groups of points, lines and other relations to which certain properties are assigned;
- tag (tag) - key-value pairs, can be assigned to points, lines and relationships.

In this work we will primarily focus on nodes which are used to describe an OSM Type of searchable apartments. As of September 2020, OpenStreetMap contains over 7 billion nodes, [7].

The result of the data query from OSM database is returned in XML format as shown in figure 2.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.0.2">
 <bounds minlat="54.0889580" minlon="12.2487570" maxlat="54.0913900" maxlon="12.2524800"/>
 <node id="298884269" lat="54.0901746" lon="12.2482632" user="SvenHRO" uid="46882" visible="true" version="1" changeset="676636" timestamp="2008-09-21T21:37:45Z"/>
 <node id="261728686" lat="54.0906309" lon="12.2441924" user="PikoWinter" uid="36744" visible="true" version="1" changeset="323878"
timestamp="2008-05-03T13:39:23Z"/>
 <node id="1831881213" version="1" changeset="12370172" lat="54.0900666" lon="12.2539381" user="lafkor" uid="75625" visible="true" timestamp="2012-07-20T09:43:19Z">
  <tag k="name" v="Neu Broderstorf"/>
  <tag k="traffic_sign" v="city_limit"/>
 </node>
 ...
 <node id="298884272" lat="54.0901447" lon="12.2516513" user="SvenHRO" uid="46882" visible="true" version="1" changeset="676636" timestamp="2008-09-21T21:37:45Z"/>
 <way id="26659127" user="Masch" uid="55988" visible="true" version="5" changeset="4142606" timestamp="2010-03-16T11:47:08Z">
  <nd ref="292403538"/>
  <nd ref="298884289"/>
  ...
  <nd ref="261728686"/>
  <tag k="highway" v="unclassified"/>
  <tag k="name" v="Pastower Straße"/>
 </way>
 <relation id="56688" user="kmvar" uid="56190" visible="true" version="28" changeset="6947637" timestamp="2011-01-12T14:23:49Z">
  <member type="node" ref="294942404" role=""/>
  ...
  <member type="node" ref="364933006" role=""/>
  <member type="way" ref="4579143" role=""/>
  ...
  <member type="node" ref="249673494" role=""/>
  <tag k="name" v="Küstenbus Linie 123"/>
  <tag k="network" v="VVW"/>
  <tag k="operator" v="Regionalverkehr Küste"/>
  <tag k="ref" v="123"/>
  <tag k="route" v="bus"/>
  <tag k="type" v="route"/>
 </relation>
 ...
</osm>
```

Figure 2.1: The XML sample of OSM data query.

Various OSM XML processing APIs, which will be discussed in the next chapters, provide the functionality to parse returned OSM XML into JSON format which can be further used for custom applications, [11].

2.1.3 OpenStreetMap data quality

Two major concepts, i.e. accuracy and precision, are often used for addressing the quality of data provided by geodatabases like OSM. The term accuracy is described as the degree to which map data matches with the reference values, on the contrary, precision measures the correctness of information in a geodatabase, [32]. There are several organizations i.e. ISO, ICA and CEN, which define internal quality aspects based on five points, [36]:

- attribute accuracy;
- positional accuracy;
- temporal accuracy;
- logical consistency;
- completeness.

Conversely, external quality considers whether a dataset is suitable for a particular task, so the user expects a dataset to meet some expectations, hence, each internal quality point does not interest them very much. According to, [22], the biggest percentage of OSM database contributors in Heidelberg, Germany, were "beginner mappers" comprising 74.3% of the total amount of contributors (346 in the study). This might lead to the conclusion that the data entered openly to OSM database might not be as precise as it could have been if a professional geographic agency would map the interested area.

Whilst using OSM database and developing corresponding software the quality question arises if this database can be treated as up-to-date. According to, [40], the data in OSM can be described into two different ways:

- OSM objects that do not change frequently i.e. house numbers;
- points of interest like commercial activities.

Those OSM objects like house numbers or street names are not changed frequently, which leads to the persistence of data quality that can be safely used for application development. On the contrary, commercial activities like company locations, shops etc. may change over time and they should not be considered as a reliable source for geolocation. The quantitative study taken in Germany, [50], shows that there is strong heterogeneity of the OSM data in terms of the regional completeness. However, most of the OSM data is collected by volunteers with probably moderate equipment the accuracy of the street data was good in major cities and acceptable in mid-sized towns, [50]. According to the aforementioned facts, OSM database can be used as a proven source for fetching house addresses, which are believed to be of reasonable quality and up-to-date, in Hamburg.

2.2 Goodness-of-Fit tests

The purpose of Goodness-of-Fit tests is to test the suitability of a random sample with a theoretical probability distribution function [20]. Strictly speaking, in the Goodness-of-Fit test analysis, the hypothesis if the random sample follows a specific discrete or continuous distribution is tested. The approach is as follows: firstly determine a test statistic which is defined as a function of the data measuring the distance between the hypothesis and the data. Secondly, assuming that the hypothesis is true, a probability value of obtaining data, with a larger value of the test statistic than the value observed, is determined. It is known to be a p value. Values of p, which are, for instance, smaller

than 0.01 are considered to be a poor fit of the distribution. On the contrary, p values closer to 1.0 depict a good fit of the distribution [20].

2.2.1 Normal distribution

The normal distribution depicts a set of continuous probability distributions, which have the same shape but differ in their position i.e. mean and scale parameters i.e. standard deviation [20]. The probability density function graph has a symmetric and bell-shaped form. According to the definition of standard deviation, provided in [20], a continuous random variable X has a normal distribution, with variance σ^2 and mean μ if its probability density function (PDF) $f_X(x)$ and cumulative distribution function (CDF) $F_X(x)$ are as follows

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}, -\infty < x < \infty,$$

and

$$F_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-(y-\mu)^2/2\sigma^2} dy = \frac{1}{2}\left[1 + erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right], -\infty < x < \infty, -\infty < \mu < \infty, \sigma > 0,$$

where $erf(.)$ is Gauss error function [48]. Figure 2.2 depicts a normal curve with a shaded area as a proportion of scores between $\mu - \sigma$ and $\mu + \sigma$. The value of the mean sets the position of the normal curve center. In all normal curves half of the scores lie to the left of the mean and half to the right; the standard deviation value determines the spread i.e. the bigger σ, the more spread out or flat the curve is [34].

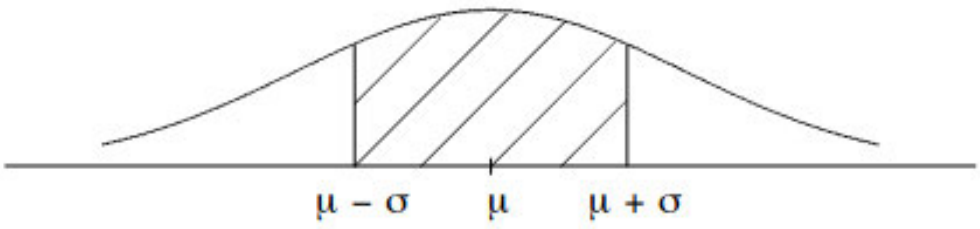

Figure 2.2: Normal curve: proportion of scores within 1 standard deviation of mean

2.2.2 Chi-Squared Test

The first Goodness-of-Fit test which is worth discussing is a **Chi-Squared Test** (χ^2). This test is applicable to check if a randomly selected continuous data fits the the specified continuous distribution. During the test, the data is grouped into k number of classes with the same probability (limitation is that each of such classes should contain at least 5 elements) [20]. The χ^2 test statistics can be calculated as follows:

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

where O_i is the observed frequency in ith class, E_i is a respective expected frequency, which is calculated by using the cumulative distribution function $F(x)$ of the x_i and x_{i-1} class limits:

$$E_i = F(x_i) - F(x_{i-1})$$

Now the null hypothesis, as well as the alternative one can be specified:

- H_0: The data follows the given continuous distribution;
- H_1: The data does not follow the given continuous distribution.

If the test statistic is larger than the critical value $\chi^2_{1-a,k-1}$, where $k-1$ denotes degrees of freedom - a number of independent observations minus a number of parameters, which are estimated, [26], and a - significance level - which refers to a rejection of the null hypothesis at a predetermined probability, [28], the H_0 null hypothesis is rejected at the significance level of a. If n parameters are estimated from the data then the degrees of freedom is calculated as $k - n - 1$ [20].

2.2.3 Kolmogorov-Smirnov Test

The second Goodness-of-Fit test is used in the work is Kolmogorov-Smirnov Test (K-S), which can also be applied to test a Goodness-of-Fit between a hypothesized CDF $F(x)$ and an empirical CDF $F_n(x)$ [20].

Assuming that $y_1 < y_2 < ... < y_n$ are the observed values of the order statistics of the random sample $x_1, x_2, ..., x_n$ of size n. When no two observations are equal, then the

empirical cumulative distribution function $F_n(x)$ is calculated as follows [20]:

$$F_n(x) = \begin{cases} 0, & x < y_1, \\ \frac{i}{n}, & y_i \leq x < y_{i+1}, \quad i = 1, 2, ..., n-1, \\ 1, & y_n \leq x. \end{cases}$$

To be more clear:

$$F_n(x) = \frac{1}{n} [number \quad of \quad observations \leq x]$$

As given in [25], D_n - the Kolmogorov-Smirnov test statistic is defined to be a maximum distance between the CDF $F(x)$ and the empirical CDF $F_n(x)$. It is calculated as follows:

$$D_n = \max\{D_n^+, D_n^-\},$$

where D_n^+ and D_n^- are given as:

$$D_n^+ = \max_{i=1,2,...,n} \left[\frac{i}{n} - F_n(x_i) \right],$$

$$D_n^- = \max_{i=1,2,...,n} \left[F_n(x_i) - \frac{i-1}{n} \right],$$

where x_i is a concrete random sample. According to [45], a close approximation of the distribution D_n fractiles is based on a constant d_α depending on n only. Therefore, the critical value of D_n can be obtained by:

$$t = \frac{d_\alpha}{\sqrt{n} + \frac{0.11}{\sqrt{n}} + 0.12}.$$

The null and alternative hypotheses are defines as follows:

- H_0: The data follows the given continuous distribution;
- H_1: The data does not follow the given continuous distribution.

The null hypothesis is rejected at a chosen significance level α, if D_n Kolmogorov-Smirnov test statistic is greater than the critical value t [20].

2.2.4 Lilliefors test

The Lilliefors test is a modified Kolmogorov-Smirnov test. It is best to use Kolmogorov-Smirnov test, when the parameters of the hypothesized distribution are fully known [43]. In case if the parameters are not known they have to be estimated based on the sample data. Assuming a sample with n observations, the Lilliefors statistics is calculated as follows:

$$D = \max_x |F^*(x) - S_n(x)|,$$

where $S_n(x)$ is the sample cumulative distribution function and $F^*(x)$ is the cumulative normal distribution. However, the statistics of Lilliefors and Kolmogorov-Smirnov tests are the same, the critical values are different, which leads to different results of normality of a distribution [43]. Same as in Kolmogorov-Smirnov test, in case if D is larger than the respective critical value, the null hypothesis is rejected.

2.2.5 Shapiro-Wilk test

The Shapiro-Wilk test is based on the idea of obtaining the test statistics by dividing the square of a linear combination of the sample order statistics by the symmetric estimate of variance [44]. The received ratio is scale and origin invariant, hence it can be used to test the normality hypothesis. In order to calculate the value of the statistic W, a complete random sample, $x_1, x_2, ..., x_n$, of size n is processed as follows:

1. The observations should be ordered to receive a sample $y_1 \leq y_2 \leq ... \leq y_n$
2. The usual symmetric unbiased estimate of $(n-1)\sigma^2$, where σ^2 is an unknown variance, is calculated

$$S^2 = \sum_{i=1}^{n}(y_i - \bar{y})^2 = \sum_{i=1}^{n}(x_i - \bar{x})^2.$$

3. If n is even i.e. $n = 2k$, then

$$b = \sum_{i=1}^{k} a_{n-i+1}(y_{n-i+1} - y_i),$$

where the a_{n-i+1} are provided in table 5 of [44]. In case if n is odd i.e. $n = 2k+1$, the values of $a_{k+1} = 0$

4. Calculate $W = \frac{b^2}{S^2}$
5. The points 1, 2, 5, 10, 50, 90, 95, 98, 99% of W distribution are provided in table 6 of [44]. Small values of W are indicating the non-normality of the distribution.

2.2.6 Anderson-Darling test

The Anderson-Darling test is the last of the Goodness-of-Fit tests considered and used in this work. The test algorithm itself tests the hypothesis that the sample was drawn from the population with a specified density function [21]. Assuming that $x_1 \leq x_2 \leq ... \leq x_n$ are the n ordered observations in the sample and $u_i = F(x_i)$. Then the Anderson-Darling statistic is calculated as follows:

$$A_n^2 = -n - \frac{1}{n}\sum_{j=1}^{n}(2j-1)[\ln u_j + \ln(1 - u_{n-j+1})],$$

if the Anderson-Darling statistic value is too high, then the hypothesis is rejected [21]. The null and alternative hypotheses are defines as follows:

- H_0: The data follows the given continuous distribution;
- H_1: The data does not follow the given continuous distribution.

The null hypothesis H_0 is rejected if the Anderson-Darling test statistic A_n^2 is larger than the critical value of the Anderson-Darling test at one of the significance levels $\alpha = 0.15, 0.10, 0.05$ and 0.01 [43].

2.3 K-Means clustering algorithm

The K-Means algorithm is an iterative algorithm that attempts to split a data set into predefined **K** non-overlapping subgroups, which are called clusters. Each data point of the set belongs to only one cluster. The algorithms makes the within-cluster data points as similar as possible, on the contrary it makes the clusters as different as possible. The sum of the squares of the distances between the data points and the cluster centroid

is calculated and supposed to be minimal for a data point to be assigned to a specific cluster. The less variation in clusters, the more homogeneous the data points within one cluster [29].

2.3.1 Algorithm description

The K-Means Clustering algorithm requires to have a precise number of clusters **k** because the center of the initial cluster may change during the iterations and this might lead to unstable clustering of the data points [47]. The algorithm is performed as follows:

1. As the first step, the amount of clusters **k** has to be chosen together with the maximum number of iterations;

2. The initialization process of **k** clusters needs to be performed and the centroid count feature should be calculated:

$$C_i = \frac{1}{M} \sum_{j=1}^{M} x_j$$

3. The Euclidean distance calculation is used to connect an observed data with a closest cluster:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

4. Data point allocation to each cluster is based on the comparison of distance between a data point with each cluster's centroid [47]:

$$a_{ij} = \begin{cases} 1, & d = mind(x_i, c_i) \\ 0, & otherwise. \end{cases}$$

5. Recalculate the cluster's midpoint position by using the objective function:

$$J = \sum_{i=1}^{n} \sum_{l=1}^{k} a_{il} D(x_i, c_l)^2,$$

where n is the amount of data points, k - number of clusters, a_{il} - denotes if data point x_i belongs to a cluster c_l. If the data point is a member of a cluster then $a_{il} = 1$, otherwise $a_{il} = 0$ [47].

6. If the midpoint of the cluster changes, then return to step 3, otherwise return the clustering result.

2.3.2 Elbow criterion

In order to select the suitable number of clusters the Elbow criterion can be used. The analysis can be achieved through plotting the relation between the number of clusters **k** and the Within-Cluster Sum of Squares (WSCC) or Sum of Square Distances (SSD), which is according to [41] is calculated as:

$$WSCC = \arg\min_{S} \sum_{i=1}^{k} \sum_{x_j \in S_i} ||x_j - \mu_i||^2,$$

where μ_i is a mean of data points in the cluster S_i.

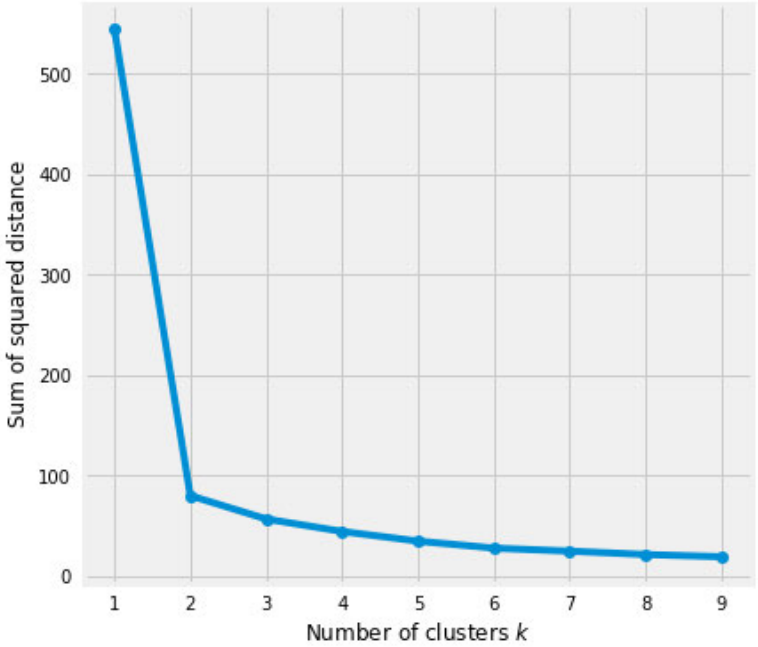

Figure 2.3: Relation between Sum of Square Distances and a number of clusters.

As it can be seen from figure 2.3 the value of 2 is a suitable number of clusters, since at the point of $k = 2$ graph starts to flatten out and forms an elbow [29].

2.4 Gradient descent algorithm

The gradient descent algorithm or the steepest descend algorithm is one of the simplest algorithms for minimizing a function [39]. The algorithm was first proposed by Cauchy in 1847 [27], where he described the idea of solving the following form nonlinear equation:

$$f(x_1, x_2, ..., x_n) = 0,$$

where f is a continuous, non-negative with real values function [39]. The idea behind it is that the function might initially decrease if the step towards the negative gradient is taken. Now the question arises, which step size is needed? Lets assume that we need to find a minimum of a function $f(x), \quad x \in R^n$. The gradient will be denoted as $g(x_k) = \nabla f(x_k)$ and the search direction as d_k. So for step calculation along the aforementioned search direction we need to have an equation like:

$$x_{k+1} = x_k + \alpha_k d_k, \quad k = 0, 1, ...,$$

where α_k is a step size calculated as:

$$\alpha_k = \arg\min_{\alpha} f(x_k + \alpha d_k).$$

The arg min depicts the argument of the function minimum [39]. In the gradient descent algorithm the search direction d_k is given as $d_k = -\nabla f(x_k)$. Now the gradient descend algorithm can be performed as shown in the Algorithm 1 [39]:

Algorithm 1 Gradient descent pseudo code

Initially given $x_0, d_0 = -g(x_0)$, and *tol* - convergence tolerance
for $k = 0$ to $maxIterations$ **do**
  Set $\alpha_k = \text{argmin } \phi(\alpha) = f(x_k) - \alpha g(x_k)$
  $x_{k+1} = x_k - \alpha_k g(x_k)$
  Compute $g_{k+1} = \nabla f(x_{k+1})$
  if $||g(x_{k+1})||^2 \leq tol$ **then**
    converged
  end if
end for

For our problem, the gradient ascent algorithm is needed, which requires to have an opposite search direction $d_k =$ as $d_k = \nabla f(x_k)$.

2.5 Web-Scraper

Web scraping process describes a procedure to query unstructured data from the web and in to transfer structured data [37]. This procedure can be divided into two steps:

- acquisition of the web resource;
- extraction of the desired information.

If a person wants to extract data from a web resource, they first need the URL. Will this be entered in the browser line or called up via a search engine, the browser starts one HTTP request to the corresponding web server and gets the resource back as an answer, to which the URL points. Often the response is an HTML document, but there are also other formats such as XML, JSON or multimedia formats for transmitting video or audio files. A scraper goes through the same process, but usually without it detour via a browser. In this case, the HTTP request is sent by the scraper that receives the response after the web server has processed the request [49].

In a thesis concluded by Nadine Dithmer [30], the web-scraping approach was described and developed. The we-scraper was written in python and could collect the data from the main real estate agencies websites like: Saga, Engelvoelkers, Immonet etc. The web-scraper collected the following apartment properties and saved them in a Coma Separated Value (CSV) file, which allows further analysis:

- Cold Rent in Euro;
- Area in square meters;
- Year the building was built;
- Street name;
- House number;
- District name;
- Web page where the data was scraped from;
- Timestamp of data scraping.

2.6 Sensitivity and specificity

In order to determine the accuracy of performed test, the sensitivity and specificity terms can be used. In this work, specificity and sensitivity are referred as true positive and true negative rates. These two terms depict the presence or absence of a test condition. For instance, in [46] the author presented an example based on test, which determines if a patient has a disease or not. The following terms can be described by the actual patient condition and test result [46]:

- True positive - the person has the disease and the test is positive;
- True negative - the person does not have the disease and the test is negative;
- False positive - the person does not have the disease and the test is positive;
- False negative - the person has the disease and the test is negative.

To calculate the true positive rate (TP_{rate}) and true negative rate (TN_{rate}) the following two formulas can be used:

$$TP_{rate} = \frac{\text{TP}}{\text{N of P-class}},$$
$$TN_{rate} = \frac{\text{TN}}{\text{N of N-class}},$$

where, TP - number of true positives, N of P-class - number of positive cases correctly classified as belonging to the positive class, TN - number of true negatives, N of N-class - number of negative cases correctly classified as belonging to the negative class [23].

3 Requirements

The main requirement of this project is to create a backend app, which is able to validate the apartment data like address, cold rent and apartment area based on the data provided by the web-scraper [30] by feeding the properties through the respective validators and obtaining a boolean result.

In order to construct the backend app the following requirements have to be fulfilled:

1. The app has to be developed in Python programming language;
2. Use the web-scraper [30] to obtain Hamburg apartment data;
3. Separate validator classes have to be created to validate the respective apartment property;
4. Each validator has to have a method which returns a boolean value depicting the validity of a particular apartment property;
5. Find a lightweight, open source and python implementable API for apartment address validation;
6. Research the approaches for validation of apartment cold rent and area and determine how well they perform and if they are suitable for this kind of a problem. The following three approaches are suggested as a starting point:
    - standard deviation approach
    - multivariate normal distribution approach
    - K-Means clustering approach
7. The algorithms should not involve creation of neural networks

The apartment data has to be divided into two parts: training and test data, which leads to an additional requirement of data partitioning. According to Pareto principle [31], the data partitioning of 80/20% is suggested. In case of small amount of apartments (either per Hamburg district or in general) the partitioning of 70/30% is required.

4 Concept

This chapter presents the approach to validation and analysis of the web-scraped apartment data. First the address validation approach is discussed where the reasons of suitable API are given as well as the overall idea how to check that the address is valid. Following the address validation, three approaches for apartment cold rent and area are discussed. These approaches had to be easily maintainable and should not involve neural networks, since they will lead to over complication of the implementation. For this purpose, the following three approaches are discussed: standard deviation approach based on each apartment property, multivariate normal distribution approach and K-Means clustering approach.

4.1 Address validation approach

One of the aims of the work is to validate the user input, which comes from the web-page user interface to be real in order to avoid inconsistent and wrong data, which might be inserted into the database for the future processing. For address validation one should refer to a database, which contains building addresses. The projects aims to use open source tools, which restricts the amount of potential analogous geodata systems.

4.1.1 Geo-database and API choice

One of the most popular and free to use geodata systems is OpenStreetMap, which is used in a project as a main database for address validation. Other geodata systems like OsmAnd or Organic Maps are based on the OpenStreetMap itself, which does not fit as an appropriate alternative. The usage of popular Google Maps or Apple Maps leads to proprietary restrictions. Hence, OpenStreetMap is considered to be the suitable option for the project's purpose.

To validate the house addresses one need to access OpenStreetMap database, via an API, and fetch the object (a node), which depicts the building with the same address as in the user input. There are various APIs, which allow the developer to get/post the data from/to OSM database by using the provided endpoints. Each of them has its own restrictions based on the functionality or usage policy. Since OpenStreetMap is the actual database it provides its own API, [1].

The current version of OSM Editing API (v.6) was deployed in April 2009 and it is based on the ideas of RESTful API, which means that the user requests to the OSM web-service is made through HTTP GET, POST, PUT and DELETE messages. Each response has a return value in XML format, which might be cumbersome to use in the backend application, so a separate parser is needed to convert the returned XML into a JSON object, [1]. The OSM Editing API is mostly designed for the map data edit rather than read-only purpose, which is the basis for the application. The API's documentation suggests to use Xapi or Overpass API, which are more focused on fetching data from the database, [10]. According to the Xapi documentation this API was not recommended, since the original native installations were not available anymore, [18], hence, Xapi was not considered in this work.

The other suggested API was Overpass API, which is exclusively focused on read-only implementations. This API uses Overpass Query Language (Overpass QL) to extract the data from the database. The source code needed to perform the query is divided into statements, which have to end up with a ";" sign. The query execution is performed by a process in a step-by-stem manner. Overpass QL has various statement types, [12]:

- Settings - optional variables, which are set in the first statement of the query;
- Block statements - groups of statements;
- Standalone queries - complete statements on their own, which can create a set (results of statements), manipulate the contents of an existing set or send the end results of a query to an output location.

The structure and the usage of Overpass API, with its complex Query Language, was considered to be overwhelming for the OSM database address extraction. The decision was made to leave this API aside due to the fact that over complicating the structure of the application software will lead to potential errors and unused functionality. The criteria for a best matching API is that it needs to have a proper endpoint and a reasonable amount of options for specifying the query. Additionally, since the **requests** library

is going to be used to fetch the addresses, the API has to be easily integrated into the **python** application.

The most suitable API for the purpose of extracting addresses from OpenStreetMap database was considered Nominatim API, which is a free geocoding service, which directly uses OSM geographic data, [38]. The API implementation is user friendly and allows to specify the address name, which comes from the user input, then the API searches for matches in the database. In case if no match was found, an empty array is returned. It is worth mentioning that there are various data query return formats, which could be specified by a developer i.e. XML, JSON, JSONv2, GeoJSON and GeocodeJSON, [13]. For project's purpose and readability reasons, the JSON format was used.

4.1.2 Validation method

After determining the tools needed for address validation, the actual methodology was developed. First, two instances are needed: validator and analyzer. The validator is responsible for accessing OpenStreetMap database via Nominatim API and returning a boolean decision if the address is valid or not. The validator refers to a list of predefined classification types provided by OpenStreetMap, to determine if the response from Nominatim API is an address, which lies in a residential area. If so, the address is considered to be valid. The addresses, which are not part of the residential area are disregarded, since they belong to shops, cafes, businesses etc. The analyzer acts as a preparation instance, which prepares the address data and puts each address through the validator to calculate the true positive and false negative rates. As it was mentioned before in additional requirements the data has to be split into 70/30% for training and testing purposes. In this approach the data was scraped from the real estate agencies websites, which is considered to be real and valid. There is no invalid or fake data, so the data partitioning does not take place in this approach and only true positive and false negative rates are considered.

4.2 Standard deviation approach

After the address validation, the apartment properties like cold rent and area need to be checked. The idea of checking these properties lies in the domain of how well this data fits the distribution of apartments over the cold rent price range and apartment areas.

The original data (real apartment properties) is used to build and plot the distributions of:

- apartment percentage over the price range;
- apartment percentage over the area range;

The distributions were further checked for normality by applying Goodness-of-Fit tests.

For further analysis one analyzer and two validators need to be created. Similarly to the address validation, the analyzer prepares the data and puts each apartment property through the respective validator to obtain a boolean result. The validators in this approach are similar but validate each apartment property separately. Each validator calculates the standard deviation and the median based on the data from the 70% of the scraped data. Then each apartment property either cold rent or area, can be checked if it lies in between the median plus/minus some amount (threshold) of standard deviation of a particular apartment property. The aim is to find the best suited threshold value, which increases the area of property validity, while still giving the best rates of true-positives and true-negatives.

In addition to the real apartments scraped from the real estate agency web-pages, the fake apartment properties can be generated by using uniformly distributed data points containing cold rent and apartment area. This data (real and fake apartments) are then labelled and can be used to train the validator to find a best matching threshold value. In order to find the aforementioned threshold value, the gradient ascend algorithm, described in section 2.4, is used. The gradient ascend algorithm is used to achieve the highest possible threshold value when the score result is not changing or fluctuating around some point.

After obtaining the threshold value, the test data comprising of 30% of all labelled data is checked by the validator and the true-positive and true-negative rates are calculated.

4.3 Multivariate normal distribution approach

Another way to validate the apartment properties obtained after the web-scraping is to use multivariate normal distribution approach, which contrary to the cold rent and area distribution approach, discussed in section 4.2, considers both apartment properties at once. Apartment cold rent and area are taken into account together as a data point

with two parameters. Hence, in this approach one analyzer and only one validator are needed. Similarly to previously discussed approach, the validator returns a boolean result denoting if a particular apartment is valid or not.

In multivariate normal distribution approach, the aim is to find a best matching threshold value, which is used to increase the size of a confidence region (ellipse), which depicts a set of valid apartments. Similarly to the previously discussed algorithm in section 4.2, the fake apartments are generated by introducing uniformly distributed data points comprising of apartment cold rent and area. The real and fake apartments are labelled and then 70% of them are used for validator training, where 30% of the total data is used for testing.

During the training process the validator calculates the initial parameters needed for building the confidence ellipse like: covariance matrix based on cold rent and area distributions, eigenvectors and eigenvalues. The threshold value is used to adjust the width and height of the ellipse. Whilst training, the shape of the ellipse is modified according to the threshold value, which is calculated by using the gradient ascent algorithm discussed in section 2.4.

After the best suited threshold value is calculated, the ellipse shape is updated and the testing based on the 30% of the total data can be performed. The testing process allows to determine if a particular data point (cold rent and area values) are lying inside the confidence ellipse. If a data point is inside, the apartment is considered to be valid. Hence, the true-positive and true-negative rates are calculated.

4.4 K-Means clustering approach

K-Means clustering approach differs from the above described approaches. First of all this approach is based on unsupervised learning methodology, which means that the data for analysis is not labelled. K-Means clustering algorithm was chosen as a third approach, since it allows to partition the unlabeled data into non-overlapping subgroups called clusters as discussed in section 2.3. The apartment data, which is obtained from the web-scraper, is considered to be real and can form one cluster of real apartments. On the other hand the data does not consist of fake or unreal apartments, which could form a second cluster. The aim of the approach is to check if K-Means clustering is a suitable algorithm for apartment data validation after fake apartments are introduced.

In order to use K-Means algorithm, python package **sklearn** can be used, which provides the functionality to create an instance of K-Means, perform data clustering and predict if some data fits one of the clusters.

The fake apartments can be created by populating the uniformly distributed data points, each of which consist of cold rent and area. The fake data points should be mixed with the original real apartments so that the K-Means clustering algorithm determines to which of two clusters a particular data point belongs. Since, the data is unlabelled, the algorithm does not now for sure, where is fake and where is real apartment. As it was discussed before the whole data is split as 70/30% for training and testing. First the algorithm should be trained based on the 70% of the data, where it performs the clustering based on the provided data. Then the algorithm predicts the cluster affiliation of each of the 30% test data points. To check how well the algorithm performs the true-positive and true-negative rates are calculated.

5 Implementation

5.1 Preliminary data preparation

The web-scraper [30] returns a Comma Separated Value (CSV) file comprised of apartment addresses and properties scraped from most popular housing agency websites in Hamburg. Each line of the CSV file has the same structure as follows: cold rent in Euro, area in square meters, the year when the building was built, street name, house number, district in Hamburg, postal code, website name and time stamp when the information was found. Figure 5.1 shows the general outlook of the received housing data in CSV format.

```
452.78,68,,Max-Eichholz-Ring,4,Lohbrügge,21031,saga,2022-07-01 19:20:42.526669
360.88,45,,Luruper Chaussee,75,Bahrenfeld,22761,saga,2022-07-01 19:20:42.528734
399.15,53,,Weimarer Str.,23,Wilhelmsburg,21107,saga,2022-07-01 19:20:42.542200
720,37.49,1965,Noldering,33,,22309,immowelt,2022-07-01 19:20:42.524828
868,28,2020,,,,,immowelt,2022-07-01 19:20:42.540413
1061,70,1988,Grindelallee,126,Rotherbaum,20146,immowelt,2022-07-01 19:20:42.617386
630,56,1967,Arno-Holz-Weg,2,Wilstorf,21077,immowelt,2022-07-01 19:20:42.624639
1275,85,1978,,,,22529,immowelt,2022-07-01 19:20:42.629216
992,32,2020,,,Barmbek-Nord,22305,immowelt,2022-07-01 19:20:42.691274
```

Figure 5.1: The CSV file outlook

As it can be seen from the second last line in figure 5.1 there were entries, which did not include full address but only a postal code. For the purpose of address validation such entries were filtered out from the file and were not considered in this work. Additionally, to have a proper estimation of the algorithm correctness the duplicate addresses were removed by using the linux command provided in code listing 5.1, since they pointed to the same building address but had an apartment number in them.

Listing 5.1: Linux command to merge all .csv files and ensure entry uniqueness

```
cat *.csv | cut -d ',' -f 9 --complement | sort | uniq > merge.csv
```

Hence, only one of such duplicate addresses was left for validation. After the file was completely prepared, the data from the file was read by using **numpy** python library and a corresponding **loadtxt()** method, which specifies the path to the CSV file, delimiter, data type to read and columns of the given CSV file as depicted in code listing 5.2.

Listing 5.2: CSV file read method

```
def read_csv(file_path):
    """Reads the CSV file"""
    csv_data_array = np.loadtxt(file_path, delimiter=",",
        dtype=str, usecols = (0, 1, 2, 3, 4, 5, 6))
    return csv_data_array
```

After the file was read, the entries were stored in the **numpy** array. In order to have the data coming from the web-scraper consistent, each entry of the aforementioned array was packed into an object by defining a python dataclass **Apartment**. Thus, the following structure of the class was developed as shown in code listing 5.3:

Listing 5.3: Apartment dataclass

```
@dataclass(frozen=True, eq=True)
class Apartment:
    cold_rent: str
    area: str
    street_name: str
    house_number: str
    postal_code: str
```

Now the data is prepared and can be used for further analysis.

5.2 Apartment addresses validation

5.2.1 Preparation of address entries for analysis

Since the data preparation was already done and all the CSV file entries were properly packed into the array of **Apartment** objects the next, specific for address validation approach, filtering is still needed. The array of Apartment objects was filtered in such a way that the entries which have not empty street name, house number and postal code

were left and the others were simply ignored. After having a careful look through the CSV file it was found that the house number which contains a letter i.e. **14 b** has a space between the house number and a letter. This means that the result was not pointing to the residential area but to the street where this building was located. Therefore, all the house numbers of the Apartment array were adjusted to not having a space between the house number and a letter. The aforementioned filtering processes were comprised into two methods: **filter_apartments()** and **fix_house_number()** as shown in the code listing 5.4.

Listing 5.4: Address specific filtering

```
def filter_apartments(apartments):
    apartments_with_address = [apartment for apartment in
        apartments if apartment.street_name != '' and \
                      apartment.house_number != '' and \
                      apartment.postal_code != '']

    return apartments_with_address

def fix_house_number(house_number):
    """ Removes a space between the house number and a letter """
    return "".join(house_number.split(" "))
```

5.2.2 Nominatim API configuration and setup

In order to use Nominatim API the endpoint and return data format have to be specified first. As it can be seen from the code listing 5.5, the endpoint /**search** was configured to access the OSM database through Nominatim API and the return data format was chosen to be JSON.

Listing 5.5: Nominatim API endpoint

```
endpoint = "https://nominatim.openstreetmap.org/search?<params>"
format = "json"
```

As it was discussed before, request library was used to perform the database query. The **requests** library **get** method requires the endpoint and optional parameters to be set,

[14]. In addition to the endpoint and format the following parameters were set to specify the query:

- format - output format of the database query
- country - Germany;
- city - Hamburg;
- postalcode - was extracted from the user input;
- street - in Nominatim API street is a street name and a house number all together disregarding the order.

The code listing 5.6 depicts the request made by **requests** through Nominatim API to OSM database. The snippet of the code shows that the address information entered by a user was accessed via the properties and assigned to the query parameters. The **requests get()** method return value is casted to JSON via calling **json()** method on the API response data.

Listing 5.6: Requests call

```
valid_status = 200

params = dict()
params['format'] = format
params['country'] = 'Germany'
params['city'] = 'Hamburg'
params['postalcode'] = apartment.postal_code
params['street'] = "{} {}".format(apartment.street_name,
    apartment.house_number)

response = requests.get(endpoint, params = params)

if response.status_code == valid_status:
   data = response.json()
```

5.2.3 Address validation

The array of Apartments having a full address was fed through the Nominatim API in order to find all matches in OSM database.

The received array of potential matches was filtered to get the most probable node which was lying in the residential area. According to the official OSM documentation [6] there are various tags that can be used to identify what the node exactly is. Since the aim is to validate the addresses, which belong to residential areas, the following node types and respective tags were used to filter the end result coming from Nominatim API response:

- type **building** has tags:
  - apartments;
  - bungalow;
  - cabin;
  - detached;
  - farm;
  - house;
  - houseboat;
  - residential;
  - semidetached_house;
  - terrace;
  - yes - almost always denotes the building where a living area can be rented;
- type **place** has tags:
  - house;
- type **highway** has tags:
  - residential - denotes the street, which lies in a residential area.

During the process of validation, shown in code listing 5.7, the **sleep()** method from **time** library was used in order to prevent the API overload [8] with the requests. Since, Nominatim API allows to have 1 request in 1 second [8] it was decided to have a delay of 1.5 seconds in order to observe the responses visually in the console.

Listing 5.7: Address validation algorithm

```
n_of_valid = 0 # keeps the number of positively validated
    addresses
for (i, apartment) in enumerate(apartments):
    address_validator = AddressValidator()
    if address_validator.validate(apartment):
        n_of_valid += 1

    time.sleep(1.5) # pause for each request
    print("Iteration: {}".format(i))
    print("Number of Valid Addresses: {}\n".format(n_of_valid))

true_positives = (n_of_valid / len(apartments)) * 100
false_negatives = 100.0 - true_positives
```

During the validation process, the ratios of true-positives and false-negatives were calculated.

5.3 Apartment cold rent and area validation

5.3.1 Preparation of apartment property entries for analysis

To perform the numerical analysis on the data which is stored in the Apartment array, this data has to be filtered to contain cold rent and apartment area as numeric values (all the data from the CSV file was read as string). Previously, the apartment addresses could be used as string type. In this case, the cold rent and apartment area needed to be converted into float. For this purpose the method **is_number()** was developed, the code listing 5.8 is provided below.

Listing 5.8: The example of checking for float like string entries

```
def is_number(n):
    try:
        float(n)
    except ValueError:
        return False
    return True
```

The method **is_number()** was used to check if an entry can be cast to float, if so, then the method returned **True** and the entry could be considered further. If the **ValueError** occurs, then the error was expected and the method returned **False**. The aforementioned method was used as filter to filter out every apartment in the Apartment array, that could not be used for the analysis. The data which passed the filtering was collected and put into a numpy array of Apartment objects. The filtering method **filter_apartments()** was developed as shown in code listing 5.9.

Listing 5.9: Filtering apartments for numeric parameters

```
def filter_apartments(apartments):
    return [apartment for apartment in apartments if
            ut.is_number(apartment.cold_rent) and
            ut.is_number(apartment.area)]
```

The real apartments, stored in the filtered array of Apartments described before, were divided into two parts: 70% of the apartments were used for the training purpose and the rest 30% were used as a test data to validate the approach. At this point there were only apartments, which were considered to be real, since their properties were obtained through the web-scraper. In order to perform a proper training of a validation model the presence of real and fake apartments is needed. Fake apartments were created by populating the uniformly distributed data points with area range from 5 to 250 m^2 and with the cold rent range from 100 to 3000 EUR. Analogous to the real apartments, fake apartments were split into two parts: training (70% of the total) and testing (30% of the total) data arrays. The data points were then packed as arrays of Apartment objects. Since, a supervised learning approach was intended to be used, the data used for training and testing should have been labelled. As it can be seen from code listing 5.10, for labelling the apartments a dictionary was used, which contained an Apartment object and a flag: True - real apartment or False - fake apartment. After labelling the

apartments, the arrays of real and fake training apartments as well as real and fake testing apartments were concatenated.

Listing 5.10: Apartment labelling

```
def label_apartments(real_apts, fake_apts):
    labeled_real_apartments = [(apt, True) for apt in real_apts]
    labeled_fake_apartments = [(apt, False) for apt in fake_apts]
    return labeled_real_apartments + labeled_fake_apartments
```

5.3.2 Implementation of standard deviation approach

In order to train the model cold rent validator and area validator were created. The idea behind the validators is that the labelled training apartments were passed to the respective validator with a predefined starting threshold value - the number of standard deviations needed. Then the validator builds up an array cold rent/area float values of real apartments. Based on the obtained array, the validator calculates the standard deviation and the median values. This data was needed for implementing gradient ascend algorithm.

As it can be seen from code listing 5.11, the threshold step size and number of training iterations should be set. After that, for each iteration the threshold size is altered based on the step size and score function derivative value.

Listing 5.11: Cold rent validator training

```
    threshold_step_size = 0.2
    training_iterations = 1000
    for _, iteration in enumerate(range(training_iterations)):
        self.threshold += threshold_step_size *
            calculate_score_derivative()
```

The score function is calculated through evaluating given labelled apartments (real and fake) by checking if an apartment lies in between the specified distance from the median. As it is shown in figure 5.2, the apartment (denoted as Apt.) property like cold rent or area is checked to be greater than the difference of median and threshold amount of standard deviations as well as to be less than the sum of median and threshold amount

of standard deviations. If the property lies in between the aforementioned values, then it is considered to be valid.

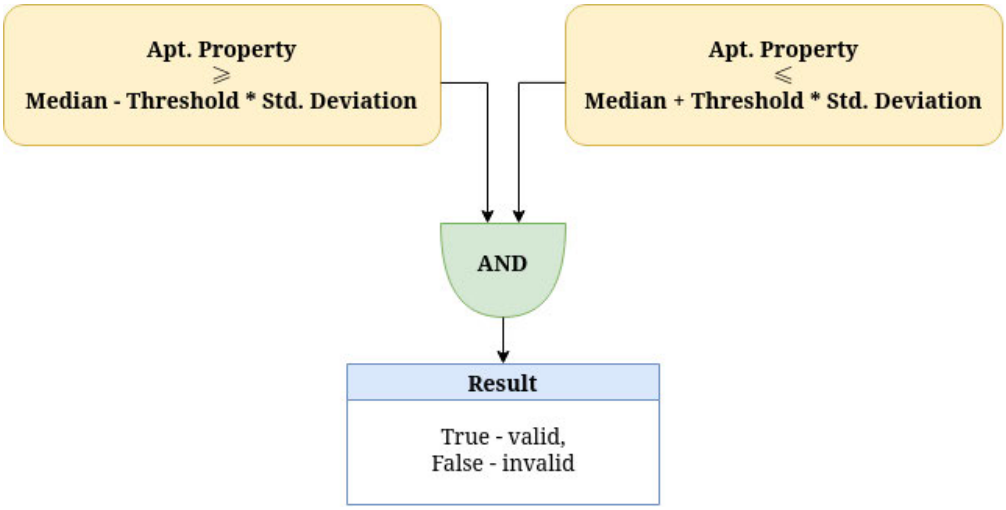

Figure 5.2: Apartment property validation process

During the evaluation of labelled training apartments, each apartment is validated based on the aforementioned algorithm. Then the amount of apartments validated as true positives - number of real apartments, which were correctly validated and true negatives - number of fake apartments, which were correctly validated, was calculated. These two values were used to calculate the amount of times the algorithm validated a particular apartment correctly. Each value was divided by the total amount of respective apartments to get the the percentage ratio. After, the results were summed up and divided by 2 to be normalized between 0 and 1. The score function was calculated through the following formula:

$$\text{score} = \frac{\frac{\text{number of true positives}}{\text{total amount of real apartments}} + \frac{\text{number of true negatives}}{\text{total amount of fake apartments}}}{2},$$

As it was mentioned previously, during the training process showed in code listing 5.11, the numerical approximation of score function derivative was calculated. For the calculation the differential step size was experimentally chosen in a way that the numerical approximation of the score derivative results in a non-zero value. The differential step size was used to increase the threshold value which was then used for calculating the score value. The score value was calculated for two threshold values:

- threshold value in current training iteration;
- increased by differential step size threshold value of the current iteration.

The numerical approximation of score derivative is then calculated by the following formula:

$$\text{num. approx. of score derivative} = \frac{\text{score by increased threshold} - \text{score by threshold}}{\text{differential step}},$$

The threshold step size, number of training iterations and the differential step are three hyper-parameters used in the supervised learning process. These values can be determined experimentally, by running the algorithm and observing when the score is not changing anymore or fluctuating around some point. The used hyper-parameters can be found for cold rent validator in appendix A.7, for area validator in appendix A.8. After the training process was finished, the test real and fake apartments were validated and the ratios of true-positives and true-negatives were calculated.

5.3.3 Implementation of multivariate normal distribution approach

In the second approach the focus was on the idea how to validate the apartment based on the multivariate normal distribution. For this purpose the same starting point as in subsection 5.3.2 was used, where for training 70% of real apartments together with 70% of fake uniformly distributed apartments as well as for testing 30% of real apartments together with 30% of fake uniformly distributed apartments were collected. As previously the data was labelled and passed to the validator for training.

The confidence ellipse - an ellipse over the multivariate distribution of data points, which, in this approach, can be built to denote the data points of valid apartments (inside the ellipse) and invalid apartments (outside of the ellipse). In order to build the staring confidence ellipse, the arrays containing values of cold rent and area of real apartments were obtained. Then the covariance matrix based on the aforementioned distributions was calculated by using **cov()** method of **numpy** library. The eigenvectors and eigenvalues were calculated from the covariance matrix by using **numpy.linalg.eig()** method. These parameters were needed to build the starting confidence ellipse.

Listing 5.12: Confidence ellipse creation

```
def update_ellipse(self):
    return Ellipse(xy=(np.mean(self.real_apartments_cold_rents),
              np.mean(self.real_apartments_areas)),
              width=self.eigvals[0]*self.threshold*2,
              height=self.eigvals[1]*self.threshold*2,
              angle=np.rad2deg(np.arccos(self.eigvects[0, 0])),
              label='threshold: {:.3f}'.format(self.threshold),
              edgecolor='firebrick',facecolor='none')
```

The code listing 5.12 shows the **update_ellipse()** method, which was used for the first confidence ellipse creation as well as for the ellipse shape update based on the threshold value, which was changing during the training process. As it can be seen from code listing 5.12, the method builds up a confidence ellipse by setting its height, width and angle based on the calculations explained before. It was assumed to have a threshold starting value to be 2, which will be adjusted during the training process. The training is performed in a similar way as in subsection 5.3.2 with the adjustment, that the confidence ellipse has to be updated every time the threshold value changes.

The figure 5.3 shows the basic process of apartment properties validation. First the cold rent and area of the passed apartment are packed into a tuple, which represents a data point in the multivariate distribution based on cold rent and area distributions. As a second step, the built confidence ellipse checks if the data point lies within its boundaries and decides if the apartment can be considered valid.

The code listing 5.13 shows the training process of the confidence ellipse validator. The number of training iterations is significantly smaller than in the cold rent or area validators, which is related to a quite time-consuming process of confidence ellipse creation and calculation of the same threshold value for ellipse's height and width.

Listing 5.13: Confidence ellipse validator training

```
    threshold_step_size = 0.2
    training_iterations = 20
    for _, iteration in enumerate(range(training_iterations)):
        self.threshold += threshold_step_size *
            calculate_score_derivative()
        self.ellipse = self.update_ellipse()
```

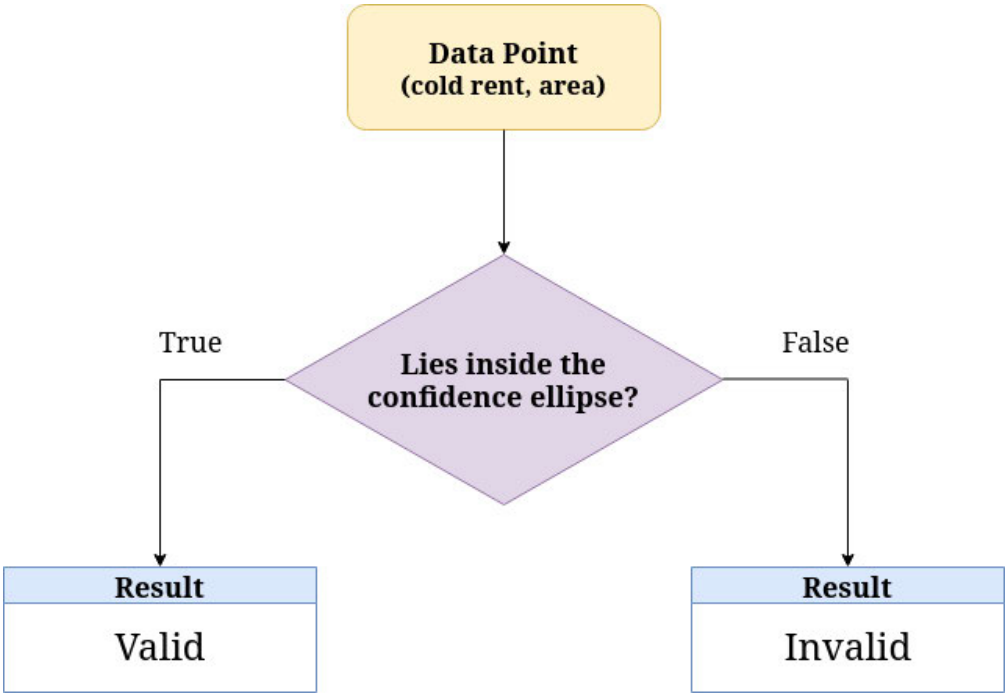

Figure 5.3: Apartment property validation process based on the confidence ellipse usage

As it was discussed before there are three hyper-parameters, which can adjust the training process: number of training iterations, threshold step size and differential step. The used hyper-parameters are shown in appendix A.9.

5.3.4 Per district analysis implementation of apartment cold rent and area

The analysis, in both approaches discussed in subsections 4.2 and 4.3, was based on the data coming from all apartments collected by the web-scraper from real estate agencies in Hamburg. Since, each district in Hamburg differs mostly in rent prices, it was decided to perform the aforementioned two approaches on each district and observe, which approach fits best for the analysis.

For per district analysis, the array of Apartment objects was additionally filtered to have the apartments, which contain a postal code. A separate JSON file was created, with Hamburg district postal codes obtained from [3]. In the analysis the following districts were considered:

- Mitte;
- Nord;
- Eimsbüttel;
- Altona;
- Bergedorf;
- Harburg;
- Wandsbek.

By using **load()** method from python **json** library the postal codes were loaded and could be used for assigning an apartment to a particular Hamburg district. As it can be seen from code listing 5.14, the apartments were filtered and collected in a list if the particular apartment postal code was belonging to a district with a given name as a function argument.

Listing 5.14: Apartment filtering which belong to a specific district

```
def filter_for_district(apartments, zip_per_district, district_name):
    """Extracts the entries with a specified district"""
    return [apartment for apartment in apartments if
        apartment.postal_code in zip_per_district[district_name]]
```

After the apartments were filtered with respect to the district they were fed into two **analyze()** methods from the approaches developed before.

5.3.5 Implementation of K-Means clustering approach

The K-Means clustering algorithm is a unsupervised machine learning algorithm, so the implementation of it is slightly different from what was discussed before. The array of Apartment objects was still used and the objects were filtered to have numeric values of cold rent and area. Similarly to previous implementations, the real apartment data was split into two parts of 70% for training process and 30% for testing. The fake apartments were also built by using the uniformly distributed data as before. The major difference in data preparation was that the data points were not labelled so the algorithm decides to which cluster each data point belongs. First of all the real training apartment data points were plotted in order to observe the original shape, which is shown in figure 5.4.

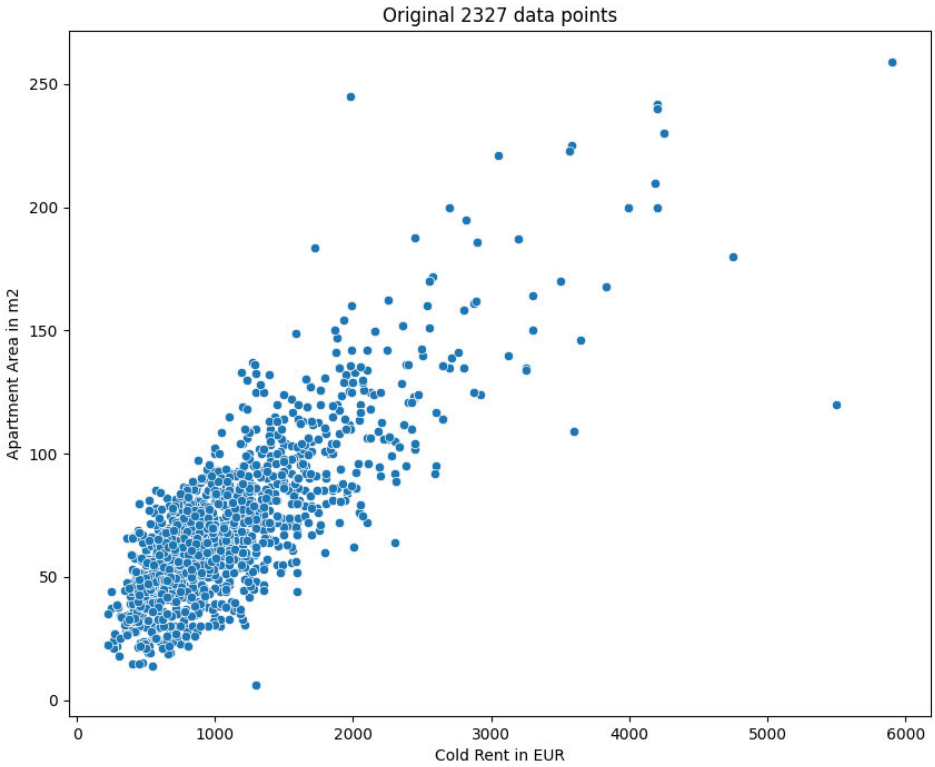

Figure 5.4: Original real apartment data points

In order to perform K-Means clustering the **KMeans** class from python package **sklearn** was used. Next the uniformly distributed training data points together with the original training data were packed as **pandas** DataFrame with cold rent as one column and the apartment area as the other as shown in code listing 5.15.

Listing 5.15: Data packing for K-Means algorithm

```
# prepare train data
rent_train_data = [float(cro.cold_rent) for cro in
                   train_apartments] # cold rent
area_train_data = [float(cro.area) for cro in
                   train_apartments] # apartment area
df_real_train = pd.DataFrame(zip(rent_train_data,
    area_train_data), columns = ['ColdRent', 'Area'])

# prepare fake train data
fake_rent_train_data = [float(cro.cold_rent) for cro in
                   uni_train_apartments] # cold rent
fake_area_train_data = [float(cro.area) for cro in
                   uni_train_apartments] # apartment area

# save all train data in a dataframe
df_all_train = pd.DataFrame(zip(rent_train_data +
    fake_rent_train_data, area_train_data + fake_area_train_data),
    columns = ['ColdRent', 'Area'])
```

Created DataFrame was analyzed through the K-Means algorithm, as shown in code listing 5.16, where the number of clusters equals to 2: valid and invalid apartments and **init** parameter is set to **random**, which means it chooses n clusters observations at random from data for the initial centroids [15]. By applying **fit()** method on k-means instance the k-means clustering is computed.

Listing 5.16: Implementation of K-Means algorithm

```
number_of_clusters = 2
kmeans = KMeans(n_clusters = number_of_clusters, init = 'random')
kmeans.fit(df_all_train)
```

As it can be seen from the figure 5.5, the plot shows all training data, including real and fake apartments, grouped into two clusters.

After performing the K-means clustering the test data points were validated to determine the performance of this approach. The test data was build in the same way as training data but at this point the arrays were not concatenated inside the DataFrame. This was done in order to simplify the calculation of the true-positives and true-negatives.

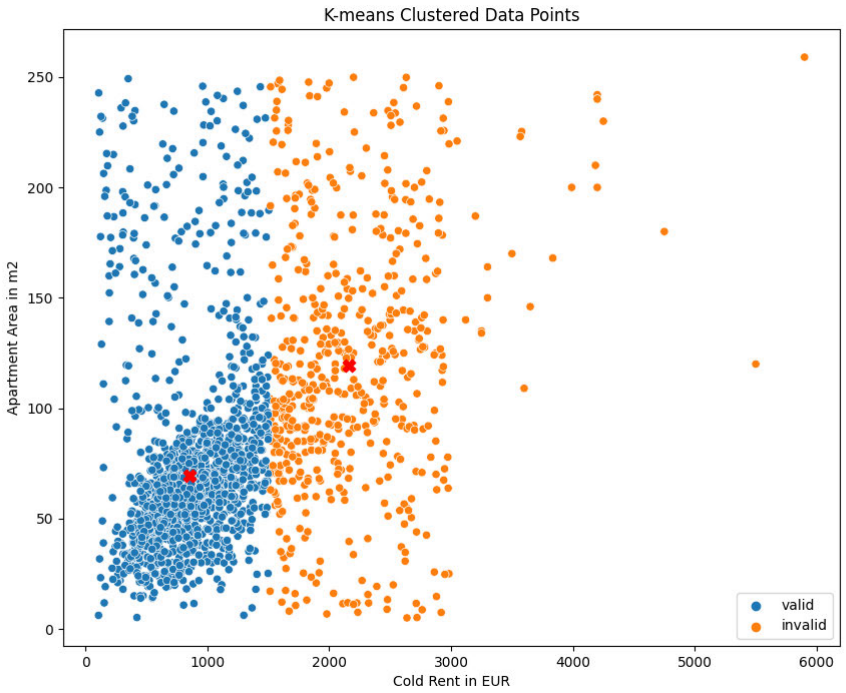

Figure 5.5: Clustered real and fake apartment data points

To calculate the aforementioned values, first the affiliation of test data points was determined. The **predict()** method was used on the trained instance of K-Means. The method received a DataFrame consisted of cold rent and area of unknown apartments and returned an array containing the cluster order number. Code listing 5.17 shows the implementation of performance calculation.

Listing 5.17: K-Means performance calculation

```
cluster_affiliation_real_test_data = kmeans.predict(test_real_df)
cluster_affiliation_fake_test_data = kmeans.predict(test_fake_df)

true_positives = sum(1 for real_apt_aff in
cluster_affiliation_real_test_data if real_apt_aff == 0)
true_negatives = sum(1 for fake_apt_aff in
cluster_affiliation_fake_test_data if fake_apt_aff == 1)
p_true_positives = (true_positives /
    len(cluster_affiliation_real_test_data)) * 100
```

```
    p_true_negatives = (true_negatives /
        len(cluster_affiliation_fake_test_data)) * 100
```

For testing the total of 1143 real and fake apartments were used.

6 Results

6.1 Apartment address validation results

As it can be seen from table 6.1 nearly 91% of all entries were positively validated. The result shows quite high probability of false-negatives, which can be related to imprecise apartment address, which could have a typo in the street name or it lacks a number in a postal code. This imprecision adds a small constraint on the usage of the API, since it requires a full and real address in order to perform a database query and return a meaningful result.

Number of Address Entries	2049
Valid Addresses	1863
P(TP)	90.92%
P(FN)	9.08%

Table 6.1: Address validation result

6.2 Standard deviation approach results

The hyper-parameters, determined during the experiment are shown in table 6.2

Hyper-Parameter	Cold Rent Validator Values	Area Validator Values
N of training iterations	1000	100
Threshold step size	0.2	0.1
Differential step	0.052	0.001

Table 6.2: Hyper-Parameters of cold rent and area validators

After the training process based on 2327 real apartments and 2327 fake apartments the threshold values for the cold rent validator and area validator were obtained and showed in table 6.3.

Cold rent validator threshold	1.82
Area validator threshold	2.0

Table 6.3: Threshold values obtained after training

To visualize the distribution of real cold rent prices and real apartment areas, **matplotlib** library was used and the distributions were plotted with respect to the percentage of the apartments belonging to a particular value range. As it can be seen from figure 6.1 and figure 6.2 the distributions look close to the shape of a normal distribution.

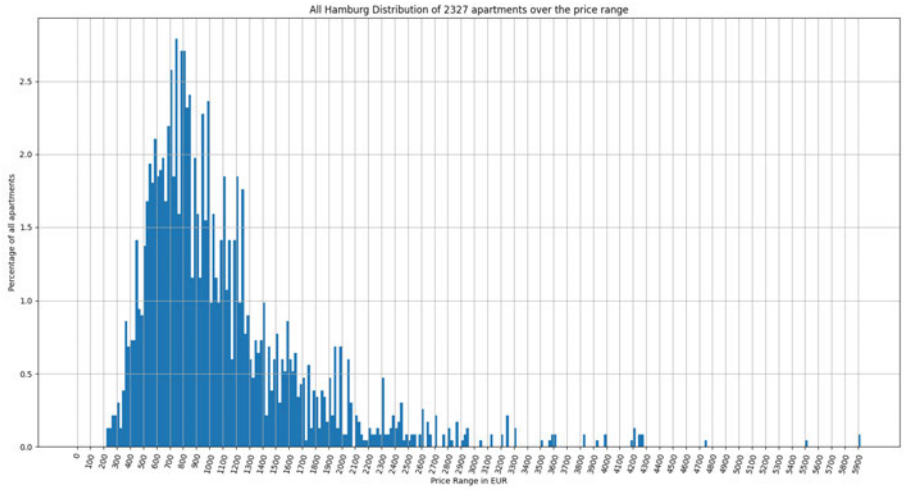

Figure 6.1: Distribution of apartment cold rent prices in Hamburg

In order to move on with the analysis of the test data (as known as 30% data) the aforementioned distributions were checked for being normal-like distributions. For this purpose the following Goodness-of-Fit tests were performed on the obtained cold rent and area distributions:

1. Chi-Squared test;
2. Shapiro-Wilk test;
3. Lillefors test;
4. Anderson-Darling test;
5. Kolmogorov-Smirnov test.

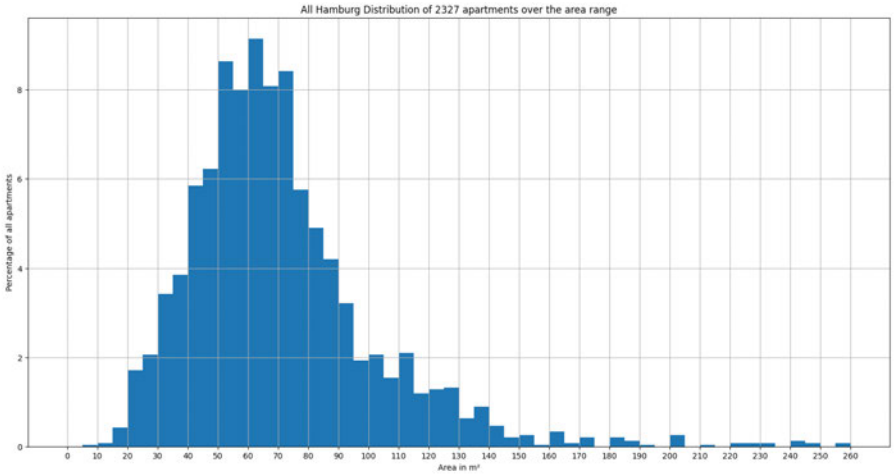

Figure 6.2: Distribution of apartment area in Hamburg

After performing Goodness-of-Fit tests it was determined that both cold rent and area distributions are normal-like according to Kolmogorov-Smirnov test.

Finally, to measure the performance of the approach, the test data, containing 30% of real apartments and 30% fake apartments, was validated and the ratios of true-positives and true-negatives were calculated. For testing the total of 1143 real and fake apartments were used. The results are presented in table 6.4.

Standard Deviation Approach Performance	
P(TP)	90.67%
P(TN)	67.42%

Table 6.4: Performance of standard deviation approach

As it can be seen from table 6.4, the approach tends to be quite imprecise for detecting fake apartments having only 67.42% of correctly identified fake apartments. On the contrary, the algorithm showed a good tendency of real apartment validation with 90.67% of correctly identified real apartments obtained from the real estate agencies websites.

6.3 Multivariate normal distribution results

The hyper-parameter values were determined experimentally and shown in table 6.5.

Hyper-Parameter	Confidence Ellipse Validator Values
N of training iterations	20
Threshold step size	0.2
Differential step	0.5

Table 6.5: Hyper-Parameters of confidence ellipse validator

After the training process based on 2327 real apartments and 2327 fake apartments the threshold value for the confidence ellipse validator was calculated and shown in table 6.6.

Confidence ellipse validator threshold	1.86

Table 6.6: Threshold value obtained after training

To visualize the multivariate normal distribution based on the real cold rent and area apartment values, **matplotlib** library was used. As it can be seen from the figure 6.3, confidence ellipse covers the major part of the data points leaving some small amount of them outside of its boundaries. The outliers mostly lie in the region with cold rent higher than nearly 2200 EUR and apartment area larger than approximately 130 m^2.

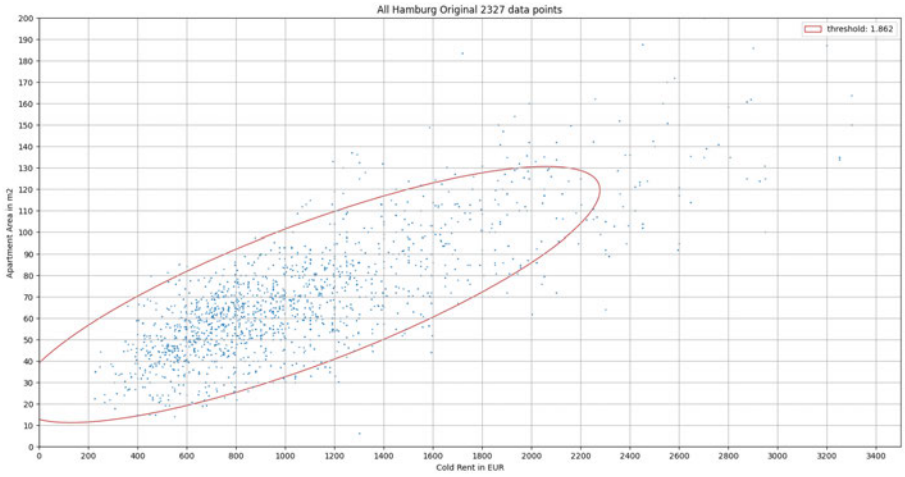

Figure 6.3: Multivariate distribution of apartments cold rent and area in Hamburg

In order to measure the performance of the approach, the test data, containing 30% of real apartments and 30% of fake apartments, was validated and ratios of true-positives and true-negatives were calculated. For testing the total of 1143 real and fake apartments were used. The results are presented in table 6.7.

Multivariate Distribution Approach Performance	
P(TP)	90.07%
P(TN)	87.69%

Table 6.7: Performance of multivariate distribution approach

As it can be observed from table 6.7, the multivariate distribution approach performs significantly better than the standard deviation approach discussed earlier, which can be seen from the rate of true-negatives. Current approach has a similar rate of true-positives, but has nearly 20% higher rate in identifying the fake apartments.

6.4 Per district analysis results

For example reasons Hamburg-Mitte district analysis is presented. The corresponding plots for other districts can be found in appendix.

To visualize the distribution of real cold rent prices and real apartment areas the distributions were plotted against the percentage of apartments belonging to a particular value range. As it can be seen from figures 6.4 and 6.5, the distributions have normal-like shape, which was proven by Kolmogorov-Smirnov Goodness-of-Fit test.

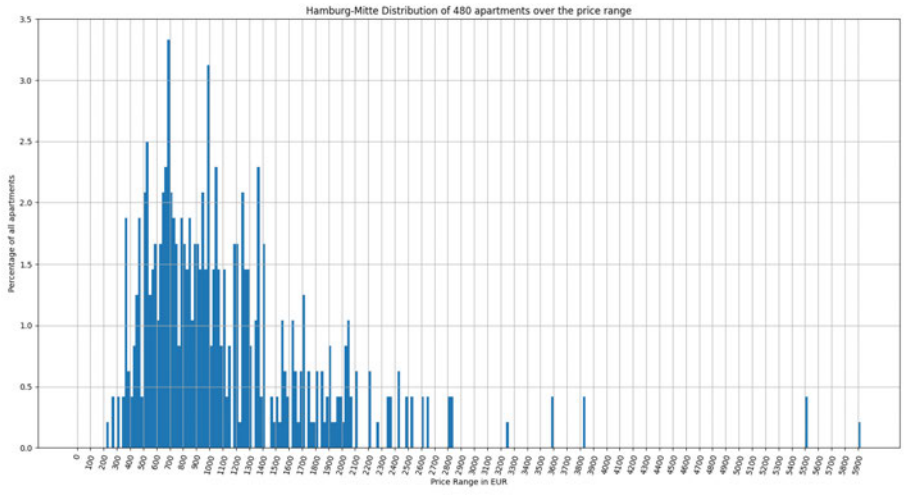

Figure 6.4: Distribution of apartment cold rent prices in Hamburg-Mitte

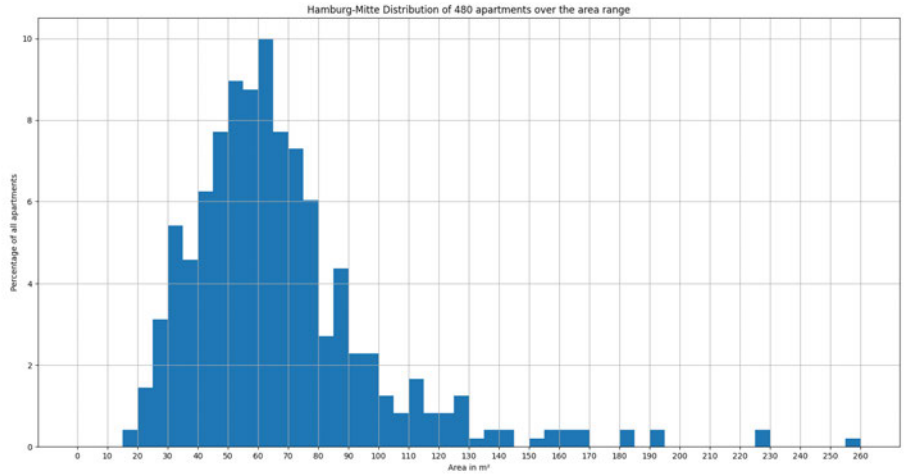

Figure 6.5: Distribution of apartment area in Hamburg-Mitte

Similarly to the standard deviation approach the multivariate normal distribution approach was performed and the distribution of cold rent prices with respect to the apartment area was plotted, which is shown in figure 6.6

To measure the performance of the standard deviation approach and multivariate normal distribution approach based on the data from Hamburg-Mitte district, the test data, containing 30% of real apartments and 30% of fake apartments, was validated and the ratios of true-positives and true-negatives were calculated. as it can be seen from table 6.8, the standard deviation approach still performs not very well for identifying the fake apartments, which is shown by low percentage of correctly identified apartments. On the contrary, the multivariate normal distribution approach performs slightly worse in validation real apartments but quite well in identifying the fake ones.

Parameter	Cold Rent Validator	Area Validator	Confidence Ellipse Validator
Threshold value	1.99	2.10	1.89
P(TP)	93.60%		87.50%
P(TN)	63.46%		80.00%

Table 6.8: Performance measurement results based on Hamburg-Mitte apartment data

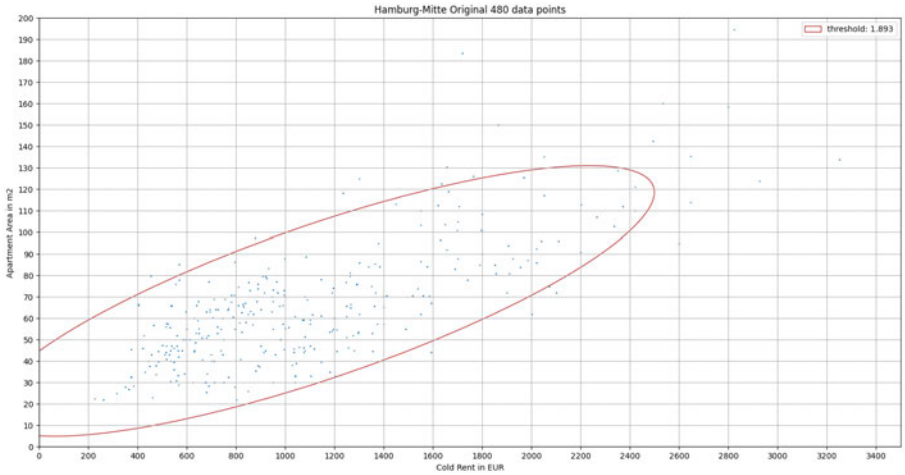

Figure 6.6: Multivariate distribution of apartments cold rent and area in Hamburg-Mitte

6.5 K-Means clustering results

The results are presented in table 6.9.

K-Means Clustering Approach Performance	
P(TP)	79.86%
P(TN)	48.68%

Table 6.9: Performance of K-Means clustering approach

As it can be observed from table 6.9, K-Means clustering approach does not perform good neither in identifying real apartments nor in identifying fake apartments. Less than 50% of fake apartments were correctly validated, which leads to a conclusion that the approach is not suitable for apartment property validation.

7 Alternative methods for apartment property validation

In chapter 4 three different methods for apartment property validation were discussed. All of them were based on the requirement that the method should not involve the building of neural networks to avoid the over complication of the overall implementation and ease of maintenance. However, while researching the most suitable methods for the stated problem in chapter 1, the methods, which could be considered as an alternative approaches, were also investigated. In this chapter the Support Vector Machines and Decision Trees are discussed. The other alternative methods for apartment property validation were not investigated due to the project time limitation.

7.1 Support Vector Machines

One of the methods involves Support Vector Machines (SVM) - a supervised learning methodology, in which the model is learning by assigning the labels to the investigated objects, [42]. The SVMs are widely used in life sciences and biological applications. The idea behind the SVM is that it is given, for instance, two labelled clusters of data points and an unknown data point i.e cold rent and area, it should decide whether that unknown data point belongs to one of the labelled clusters. As an example, the author in [42], presents a SVM methodology based on the hyperplane separation of human genes *ZYX* and *MARCKSL1* shown in figure 7.1.

As it can be seen from figure 7.1, the blue dot denotes the unknown data point which is to be predicted to belong to one of the clusters divided by the hyperplane.

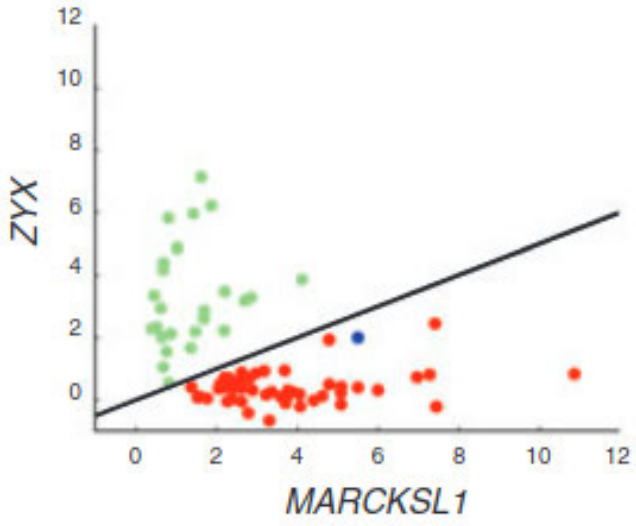

Figure 7.1: SVM hyperplane separation of human genes ZYX and MARCKSL1

This method would be suitable of the project purpose if the data received from the web-scraper was containing the fake apartments as well, so that the model could distinguish between the clusters. Additionally, the real apartment data together with the uniformly distributed data points comprising fake apartments does not fit the method. A better separation is needed to divide the apartments from all 4 sides (top, bottom, left and right), since each of them could have either a higher/lower cold rent or area.

7.2 Decision Trees

The other option for apartment data validation are the decision trees. Decision trees are used for a decision making in a form of a binary tree (the simplest example with two branches), [35]. The initial point from which the two branches are splitting apart is called an internal node, which acts like a condition based on which the left or the right branch is used. The end of the branch which is not splitting anymore is called a leaf, which depicts the decision made, [35]. The example of a decision tree based on the survival rate of the Titanic passengers is shown in figure 7.2, [4].

While splitting the tree one should know how deep the decision tree has to be. This question is generally based on the training data that is used in the experiment. If the data set is huge, the tree is becoming larger, which leads to performance issues. It is

suggested to set the maximum depth, the length of the path from the internal node to the furthest leaf, of the tree, [35].

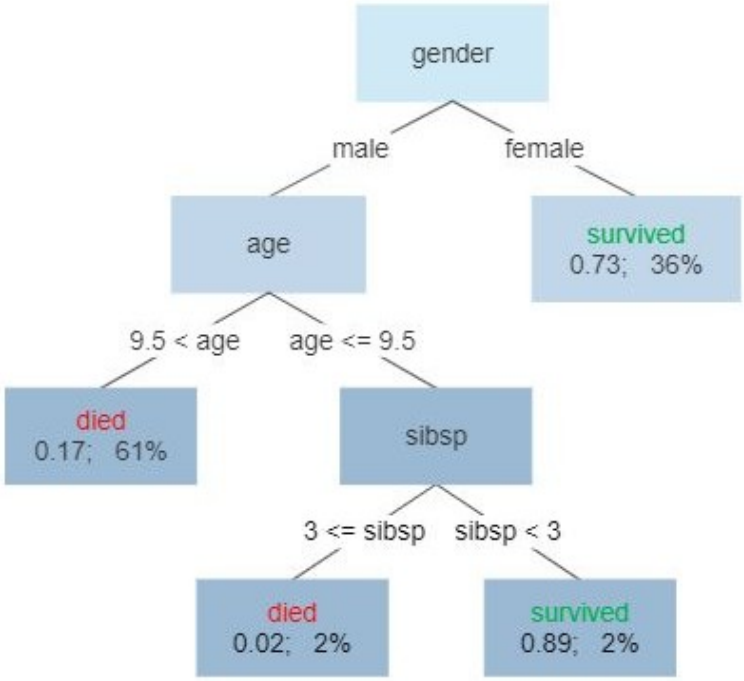

Figure 7.2: Survival rate of Titanic passengers presented in a form of a decision tree

Whilst being quite a simple methodology, the decision trees have disadvantages, [35]:

- deep decision trees tend to have performance issues;
- they are largely dependent on the data set to which applied;
- the developer can be more biased towards one of the decisions.

For the validation of the apartment properties, the decision trees could be used if the data obtained from the web-scraper was consistent, meaning that in the future web-scraping, the cold rents of the apartments could change quite drastically due to the potential economy inflation in the country. For the purpose of the project the decision trees were not chosen, since they could involve the unfairness of the condition making and would need the solid maintenance if the apartment cold rent prices rise up.

8 Conclusion and future improvements

8.1 Outcome

In this work the problem of data veracity was investigated based on the validation of apartment data collected from the real estate agencies websites. Various approaches for specific apartment property validation were developed like: address validation based on the OpenStreetMap database and Nominatim API usage, validation of cold rent price and apartment area through the standard deviation, multivariate normal distribution and K-Means clustering approaches.

As it was discovered, the address validation is heavily dependable on the data quality, which is checked by the Nominatim API. This includes proper street names, house numbers with letters, that should not be separable and incomplete postal codes. If the user makes a typo in one of the address parameters, the request is highly likely to be rejected, which obviously results in data loss for the analysis. Overall, the Nominatim API was considered to be a reliable and good choice for address validation and search in OpenStreetMap database. A minor disadvantage of Nominatim API is the number of requests per second, which are allowed to be performed according to the API usage policy. One request per second might be slow for huge amounts of data. Still, the addresses can be buffered and fed through the API in a safe way without usage policy violation.

It was also determined that the multivariate normal distribution approach performs, in most cases, better than the standard deviation approach, whilst, the K-Means clustering was considered to be a complete outsider for apartment property validation. In addition to all Hamburg apartment data, the per Hamburg district analysis was also performed resulting in a confirmation that the multivariate normal distribution approach is a better suited methodology for apartment property validation. The downside of the per district analysis is the fact that there is a lack of real apartment data, which means that the analysis needed to be based on at most of 200 apartments in some Hamburg districts. In

case of the small amount data it is suggested to use all apartments in the aforementioned approaches.

8.2 Potential improvements

The validation approaches could be used as an API for a Front-End application for crowdsourcing of apartment data. The data entered by a user, which could be a potential attacker, can be buffered and validated in a separate flow before entering the database. During the analysis there were no fake apartments, which could probably point to an industrial area or warehouses. Having this information non-biased is quite complicated and can be achieved by a crowdsourcing approach, where the volunteers will enter apartment addresses, which lie in non-residential areas. Having this data, it will be possible to calculate the ratios of true positives and true negatives, which will then provide a good overview of the approach performance. In addition, the amount of apartment data can be increased by a regular web scraping during a year, which will allow to base the cold rent and area analysis on the larger amount of data, which will lead to more precise results.

Bibliography

[1] : *API v0.6 - OpenStreetMap Wiki*. – URL https://wiki.openstreetmap.org/wiki/API_v0.6#When_NOT_to_use_the_API. – Zugriffsdatum: 2022-06-11

[2] : *Art 15 GG - Einzelnorm*. – URL http://www.gesetze-im-internet.de/gg/art_15.html. – Zugriffsdatum: 2022-08-25

[3] : *Bundesland Hamburg*. – URL https://www.suche-postleitzahl.org/hamburg.20e. – Zugriffsdatum: 2022-08-26

[4] *Decision tree learning*. – URL https://en.wikipedia.org/w/index.php?title=Decision_tree_learning&oldid=1116935696. – Zugriffsdatum: 2022-10-21. – Page Version ID: 1116935696

[5] : *Digital Help for Haiti - The New York Times*. – URL https://web.archive.org/web/20171011154945/https://gadgetwise.blogs.nytimes.com/2010/01/27/digital-help-for-haiti/. – Zugriffsdatum: 2022-06-06

[6] : *Map features - OpenStreetMap Wiki*. – URL https://wiki.openstreetmap.org/wiki/Map_features. – Zugriffsdatum: 2022-06-17

[7] : *Node - OpenStreetMap Wiki*. – URL https://wiki.openstreetmap.org/wiki/Node. – Zugriffsdatum: 2022-06-06

[8] : *Nominatim Usage Policy (aka Geocoding Policy)*. – URL https://operations.osmfoundation.org/policies/nominatim/. – Zugriffsdatum: 2022-06-17

[9] : *OpenStreetMap*. – URL https://www.openstreetmap.org/. – Zugriffsdatum: 2022-06-06

[10] : *OSM API Usage policy.* – URL https://operations.osmfoundation.org/policies/api/. – Zugriffsdatum: 2022-06-11

[11] : *OSM XML - OpenStreetMap Wiki.* – URL https://wiki.openstreetmap.org/wiki/OSM_XML. – Zugriffsdatum: 2022-06-06

[12] : *Overpass API/Overpass QL - OpenStreetMap Wiki.* – URL https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL. – Zugriffsdatum: 2022-06-11

[13] : *Place Output Formats - Nominatim Documentation.* – URL https://nominatim.org/release-docs/develop/api/Output/. – Zugriffsdatum: 2022-06-12

[14] : *Quickstart — Requests 2.28.1 documentation.* – URL https://requests.readthedocs.io/en/latest/user/quickstart/#make-a-request. – Zugriffsdatum: 2022-08-14

[15] : *sklearn.cluster.KMeans.* – URL https://scikit-learn/stable/modules/generated/sklearn.cluster.KMeans.html. – Zugriffsdatum: 2022-08-23

[16] *Wikipedia:About.* – URL https://en.wikipedia.org/w/index.php?title=Wikipedia:About&oldid=1091417331. – Zugriffsdatum: 2022-06-06. – Page Version ID: 1091417331

[17] : *Wohnraum für alle – statt Profite für wenige.* – URL http://hamburg-enteignet.de/. – Zugriffsdatum: 2022-06-06

[18] : *Xapi - OpenStreetMap Wiki.* – URL https://wiki.openstreetmap.org/wiki/Xapi#Limitations. – Zugriffsdatum: 2022-06-11

[19] ADMIN: *Mietenspiegel 2021 zeigt Scheitern der rot-grünen Wohnungspolitik.* – URL https://hamburg-enteignet.de/2021/12/13/pm-13-12-21-mietenspiegel-2021-zeigt-scheitern-der-rot-gruenen-wohnungspolitik/. – Zugriffsdatum: 2022-08-25

[20] AHSANULLAH, Mohammad ; KIBRIA, BM ; SHAKIL, Mohammad: Normal distribution. In: *Normal and Student st Distributions and Their Applications*. Springer, 2014, S. 7–50

[21] ANDERSON, Theodore W. ; DARLING, Donald A.: A test of goodness of fit. In: *Journal of the American statistical association* 49 (1954), Nr. 268, S. 765–769

[22] Arsanjani, Jamal J. ; Barron, Christopher ; Bakillah, Mohammed ; Helbich, Marco: Assessing the quality of OpenStreetMap contributors together with their contributions. In: *Proceedings of the AGILE*, 2013, S. 14–17

[23] Batista, Gustavo E. ; Prati, Ronaldo C. ; Monard, Maria C.: A study of the behavior of several methods for balancing machine learning training data. In: *ACM SIGKDD explorations newsletter* 6 (2004), Nr. 1, S. 20–29

[24] Bennett, Jonathan: *OpenStreetMap*. Packt Publishing Ltd. – 21 S. – Google-Books-ID: SZfqRcPXApoC. – ISBN 9781847197511

[25] Blischke, Wallace R. ; Murthy, DN P.: *Reliability: modeling, prediction, and optimization*. John Wiley & Sons, 2011

[26] Box, George E. ; Hunter, William H. ; Hunter, Stuart u. a.: *Statistics for experimenters*. Bd. 664. John Wiley and sons New York, 1978

[27] Cauchy, Augustin u. a.: Méthode générale pour la résolution des systemes d'équations simultanées. In: *Comp. Rend. Sci. Paris* 25 (1847), Nr. 1847, S. 536–538

[28] Chow, Siu L.: *Statistical significance: Rationale, validity and utility*. Bd. 1. Sage, 1996

[29] Dabbura, Imad: *K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks*. – URL https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a. – Zugriffsdatum: 2022-07-17

[30] Dithmer, Nadine: *Web Scraping als Instrument zur Erhebung von Mietpreisen am Beispiel des Hamburger Immobilienmarktes*. 2022

[31] Dunford, Rosie ; Su, Quanrong ; Tamang, Ekraj: The pareto principle. (2014)

[32] Fan, Hongchao ; Zipf, Alexander ; Fu, Qing ; Neis, Pascal: Quality assessment for building footprints data on OpenStreetMap. In: *International Journal of Geographical Information Science* 28 (2014), Nr. 4, S. 700–719

[33] Foody, Giles ; See, Linda ; Fritz, Steffen ; Mooney, Peter ; Olteanu-Raimond, Ana-Maria ; Costa Fonte, Cidalia ; Antoniou, Vyron: *A Review of OpenStreetMap Data*. Ubiquity Press. – 37–59 S

[34] Gordon, Sue: The normal distribution. In: *Sydney: University of Sydney* (2006)

[35] Gupta, Prashant: *Decision Trees in Machine Learning.* – URL https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052. – Zugriffsdatum: 2022-10-21

[36] Guptill, S. C. ; Morrison, J. L.: *Elements of Spatial Data Quality.* Elsevier. – Google-Books-ID: MUjgBAAAQBAJ. – ISBN 9781483287942

[37] Hillen, Judith: Web scraping for food price research. In: *British Food Journal* (2019)

[38] Karmacharya, Amrit: Positional Accuracy of Online Geocoding Services: Case Study of Bhaktapur District. In: *Journal on Geoinformatics, Nepal* 17 (2018), Nr. 1, S. 16–21

[39] Meza, Juan C.: Steepest descent. In: *Wiley Interdisciplinary Reviews: Computational Statistics* 2 (2010), Nr. 6, S. 719–722

[40] Minghini, Marco ; Frassinelli, Francesco: OpenStreetMap history for intrinsic quality assessment: Is OSM up-to-date? | Open Geospatial Data, Software and Standards | Full Text. 4, Nr. 1. – URL https://opengeospatialdata.springeropen.com/articles/10.1186/s40965-019-0067-x#Sec17. – Zugriffsdatum: 2022-06-05

[41] Mohamad, Ismail B. ; Usman, Dauda: Standardization and its effects on K-means clustering algorithm. In: *Research Journal of Applied Sciences, Engineering and Technology* 6 (2013), Nr. 17, S. 3299–3303

[42] Noble, William S.: What is a support vector machine? In: *Nature biotechnology* 24 (2006), Nr. 12, S. 1565–1567

[43] Razali, Nornadiah M. ; Wah, Yap B. u. a.: Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. In: *Journal of statistical modeling and analytics* 2 (2011), Nr. 1, S. 21–33

[44] Shapiro, Samuel S. ; Wilk, Martin B.: An analysis of variance test for normality (complete samples). In: *Biometrika* 52 (1965), Nr. 3/4, S. 591–611

[45] Stephens, Michael A.: Use of the Kolmogorov–Smirnov, Cramer–Von Mises and related statistics without extensive tables. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 32 (1970), Nr. 1, S. 115–122

[46] Swift, Amelia ; Heale, Roberta ; Twycross, Alison: What are sensitivity and specificity? In: *Evidence-Based Nursing* 23 (2020), Nr. 1, S. 2–4

[47] Syakur, MA ; Khotimah, BK ; Rochman, EMS ; Satoto, Budi D.: Integration k-means clustering method and elbow method for identification of the best customer profile cluster. In: *IOP conference series: materials science and engineering* Bd. 336 IOP Publishing (Veranst.), 2018, S. 012017

[48] Yun, Beong I.: An ad hoc approximation to the Gauss error function and a correction method. In: *Applied Mathematical Sciences* 8 (2014), Nr. 86, S. 4261–4273

[49] Zhao, Bo: Web scraping. In: *Encyclopedia of big data* (2017), S. 1–3

[50] Zielstra, Dennis ; Zipf, Alexander: Quantitative studies on the data quality of OpenStreetMap in Germany. In: *Proceedings of GIScience* Bd. 2010, 2010

A Appendix

Parameter	Cold Rent Validator	Area Validator	Confidence Ellipse Validator
Threshold value	1.91	2.00	1.96
P(TP)	90.15%		86.33%
P(TN)	55.00%		83.33%

Table A.1: Measurement results based on Hamburg-Altona apartment data

Parameter	Cold Rent Validator	Area Validator	Confidence Ellipse Validator
Threshold value	1.95	2.00	1.81
P(TP)	96.15%		86.00%
P(TN)	81.25%		86.67%

Table A.2: Measurement results based on Hamburg-Bergedorf apartment data

Parameter	Cold Rent Validator	Area Validator	Confidence Ellipse Validator
Threshold value	2.00	2.00	1.96
P(TP)	91.27%		85.13%
P(TN)	80.00%		81.82%

Table A.3: Measurement results based on Hamburg-Eimsbuettel apartment data

Parameter	Cold Rent Validator	Area Validator	Confidence Ellipse Validator
Threshold value	2.02	2.00	2.03
P(TP)	92.11%		93.67%
P(TN)	78.26%		91.67%

Table A.4: Measurement results based on Hamburg-Harburg apartment data

Parameter	Cold Rent Validator	Area Validator	Confidence Ellipse Validator
Threshold value	1.97	2.00	1.84
P(TP)	92.08%		88.71%
P(TN)	68.05%		83.78%

Table A.5: Measurement results based on Hamburg-Nord apartment data

Parameter	Cold Rent Validator	Area Validator	Confidence Ellipse Validator
Threshold value	1.97	2.00	1.93
P(TP)	93.50%		87.24%
P(TN)	71.67%		84.75%

Table A.6: Measurement results based on Hamburg-Wandsbek apartment data

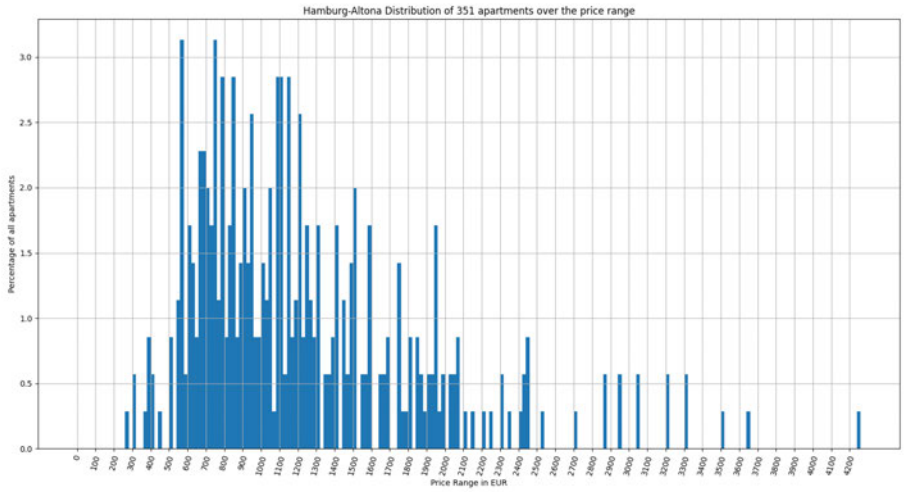

Figure A.1: Distribution of apartment cold rent prices in Hamburg-Altona

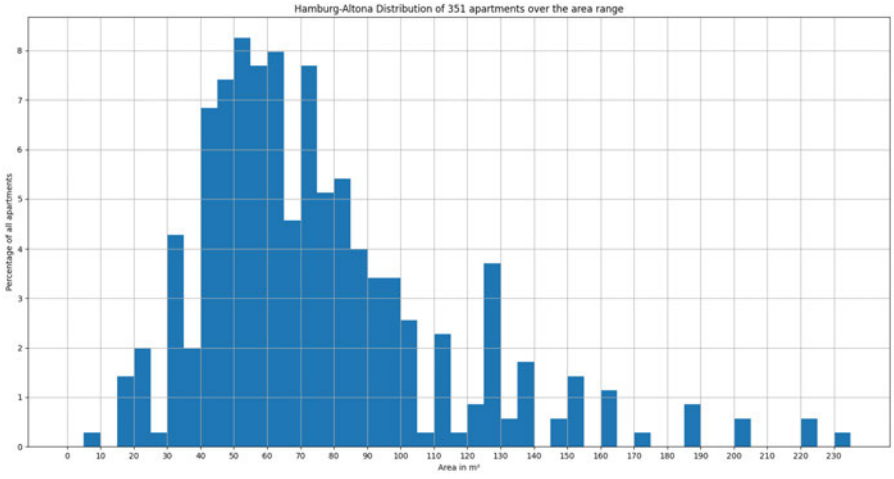

Figure A.2: Distribution of apartment area in Hamburg-Altona

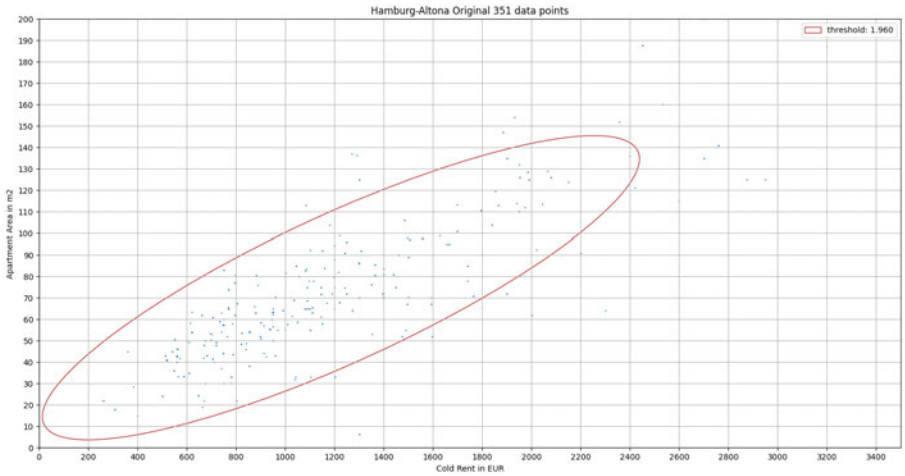

Figure A.3: Multivariate distribution of apartments cold rent and area in Hamburg-Altona

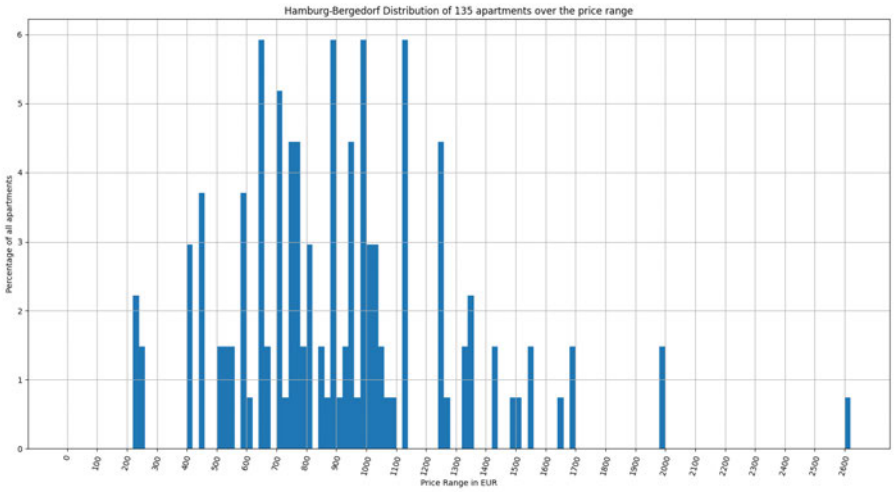

Figure A.4: Distribution of apartment cold rent prices in Hamburg-Bergedorf

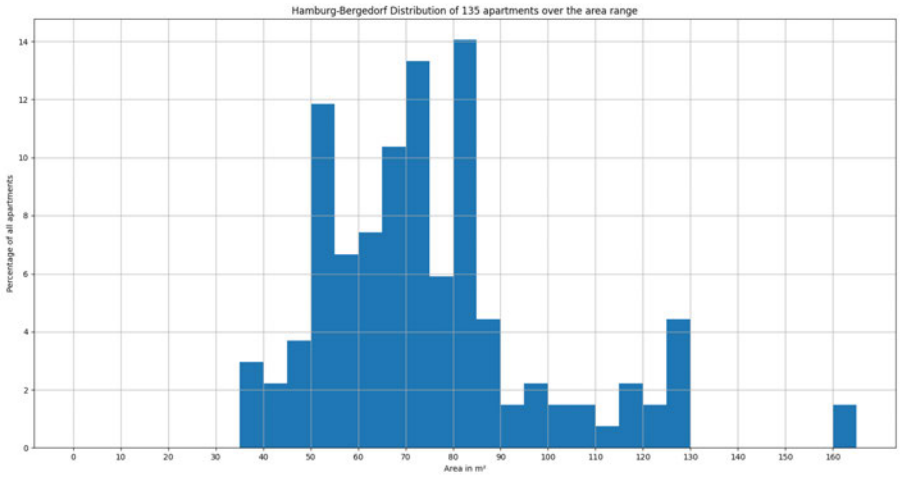

Figure A.5: Distribution of apartment area in Hamburg-Bergedorf

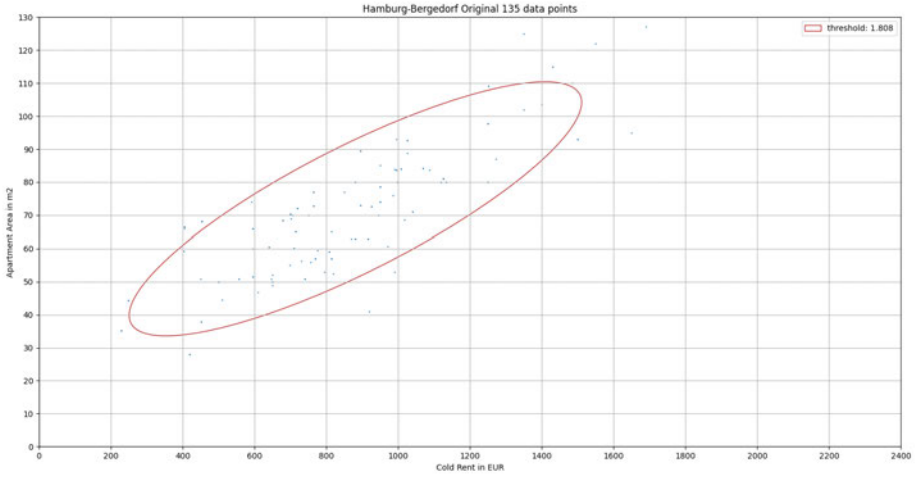

Figure A.6: Multivariate distribution of apartments cold rent and area in Hamburg-Bergedorf

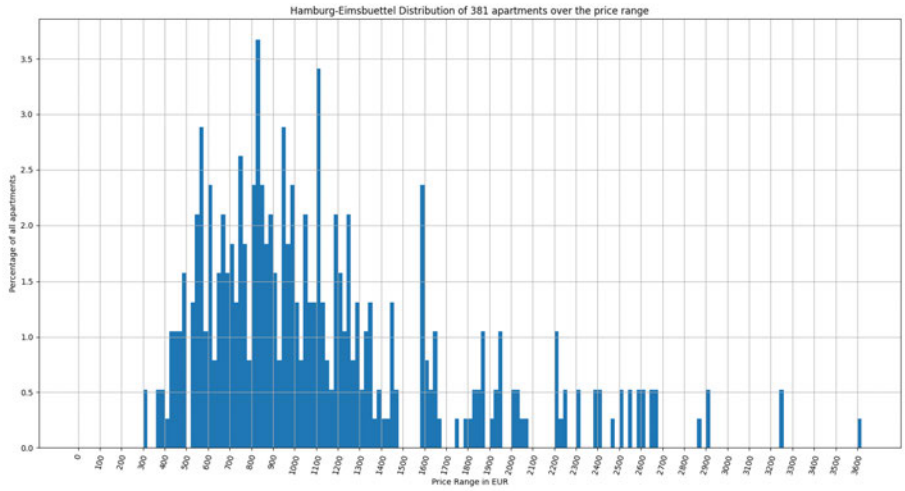

Figure A.7: Distribution of apartment cold rent prices in Hamburg-Eimsbuettel

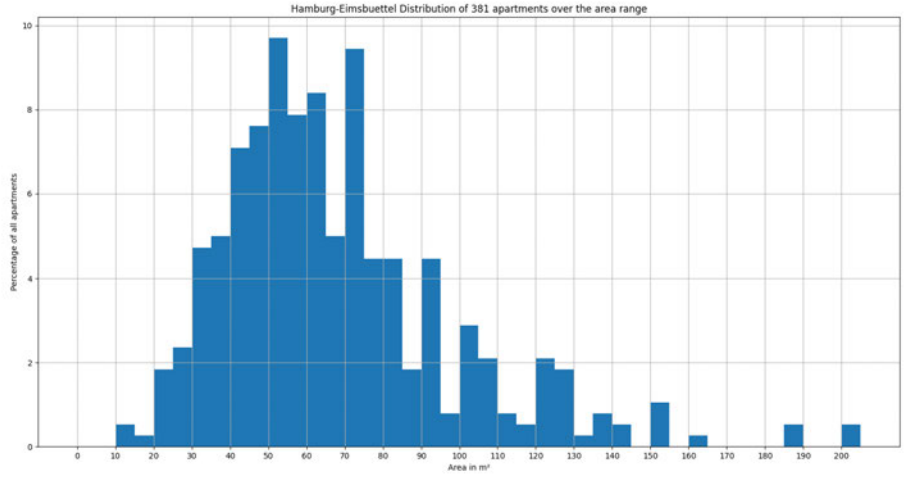

Figure A.8: Distribution of apartment area in Hamburg-Eimsbuettel

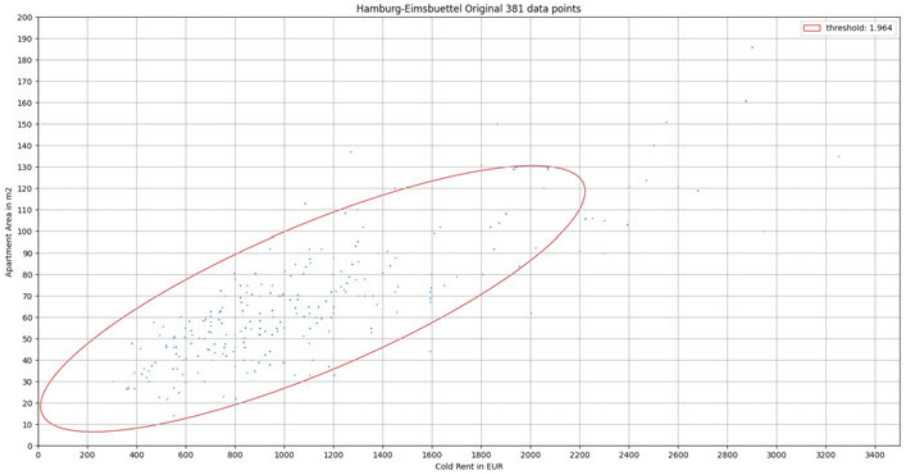

Figure A.9: Multivariate distribution of apartments cold rent and area in Hamburg-Eimsbuettel

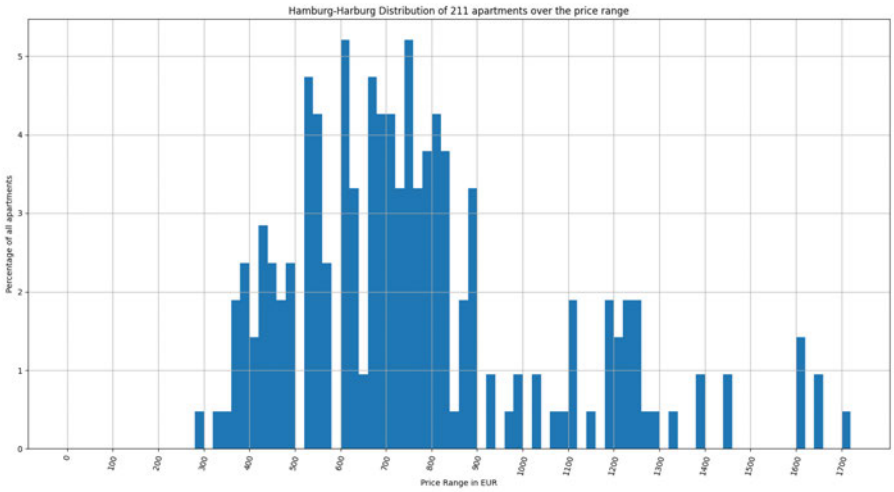

Figure A.10: Distribution of apartment cold rent prices in Hamburg-Harburg

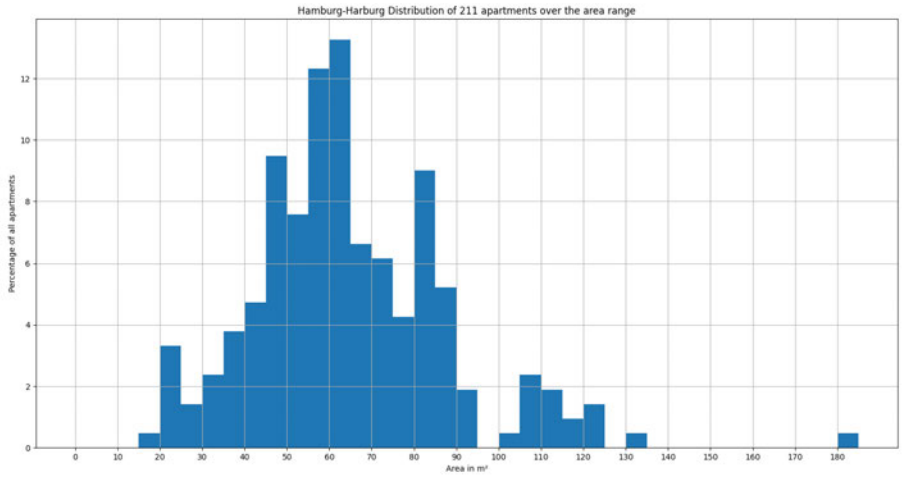

Figure A.11: Distribution of apartment area in Hamburg-Harburg

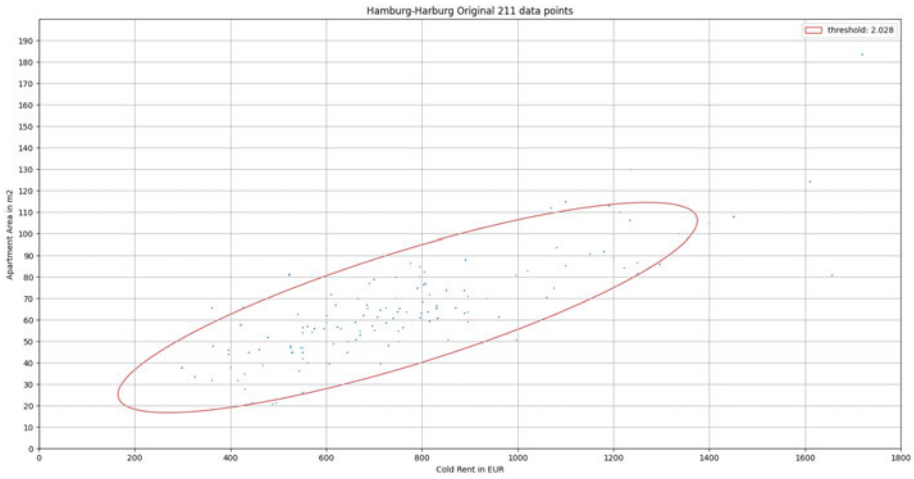

Figure A.12: Multivariate distribution of apartments cold rent and area in Hamburg-Harburg

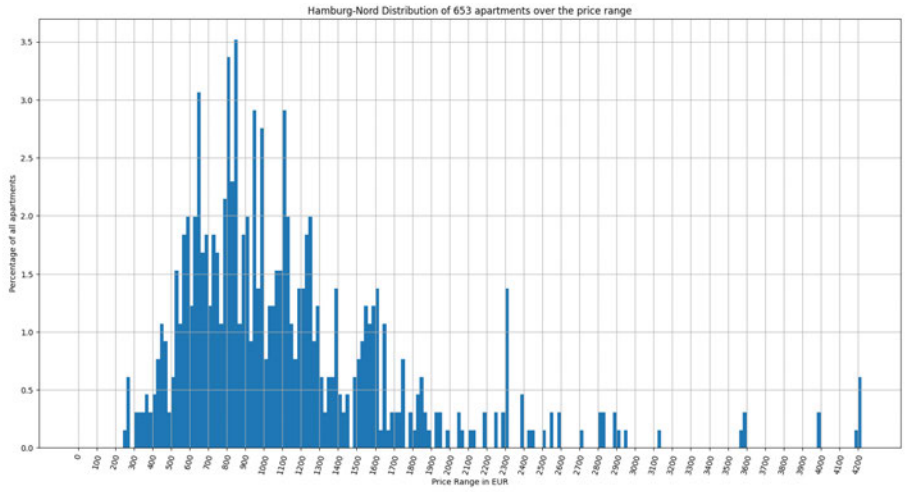

Figure A.13: Distribution of apartment cold rent prices in Hamburg-Nord

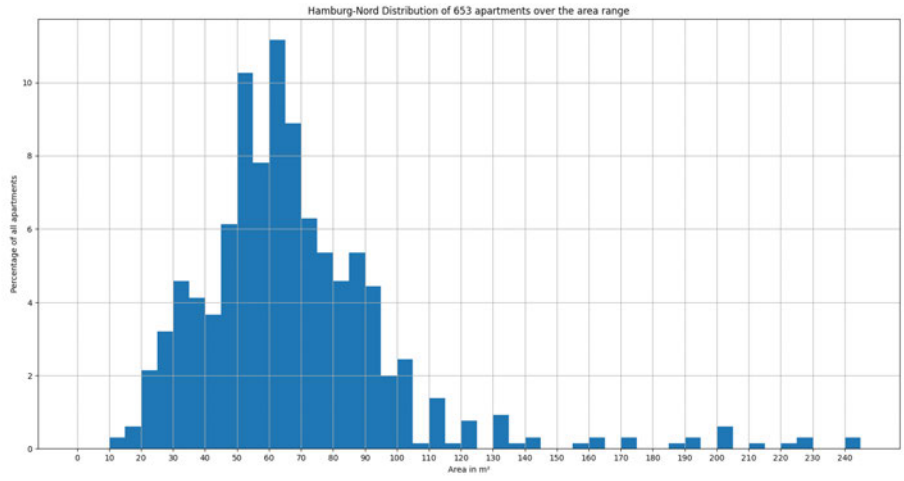

Figure A.14: Distribution of apartment area in Hamburg-Nord

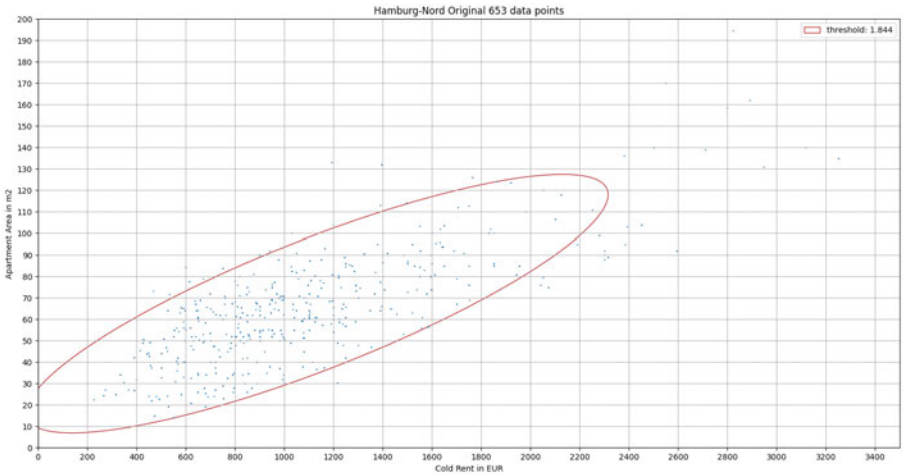

Figure A.15: Multivariate distribution of apartments cold rent and area in Hamburg-Nord

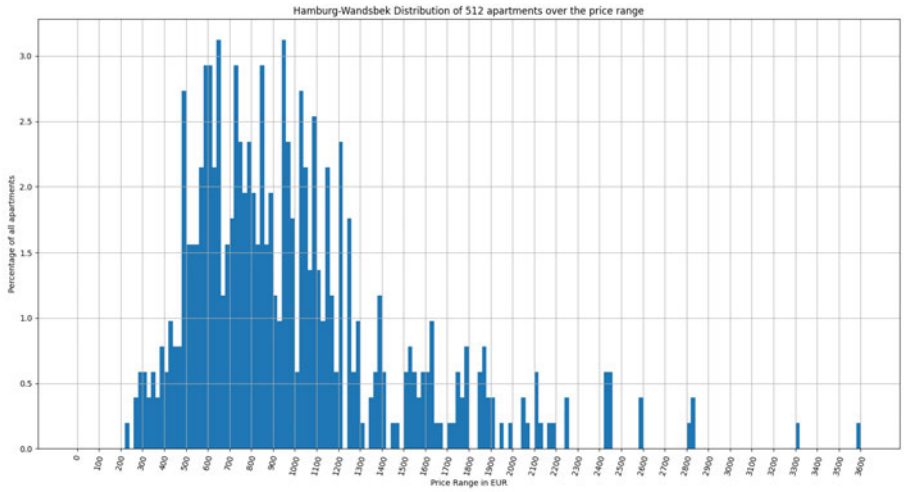

Figure A.16: Distribution of apartment cold rent prices in Hamburg-Wandsbek

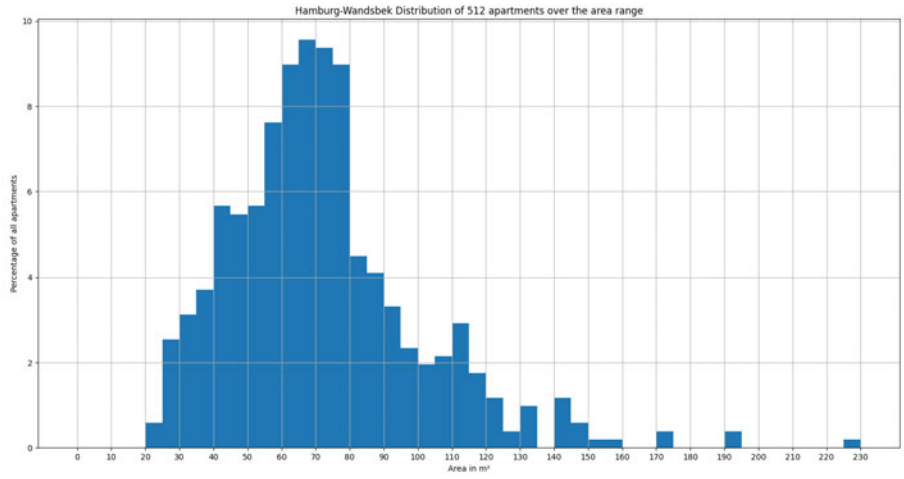

Figure A.17: Distribution of apartment area in Hamburg-Wandsbek

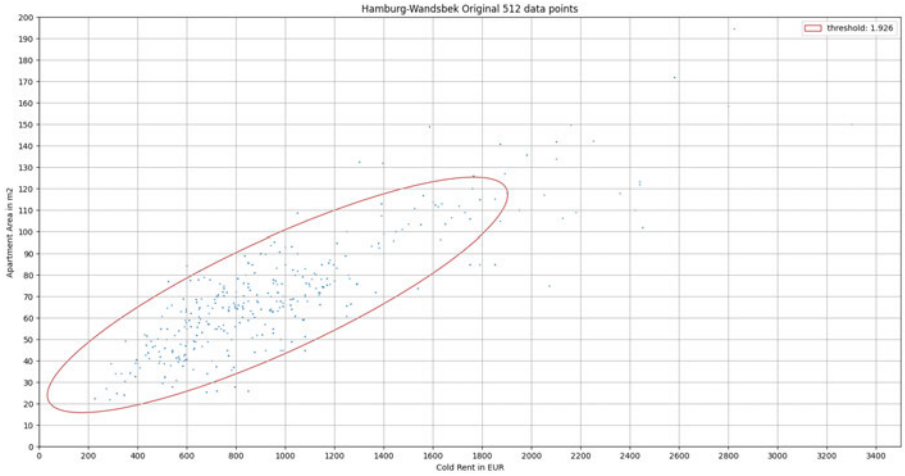

Figure A.18: Multivariate distribution of apartments cold rent and area in Hamburg-Wandsbek

N	Num. of training iterations	Threshold step size	Differential step
1	500	0.1	0.1
2	500	0.1	0.15
3	500	0.1	0.05
4	500	0.2	0.1
5	500	0.2	0.15
6	500	0.2	0.05
7	1000	0.1	0.1
8	1000	0.1	0.15
9	1000	0.1	0.05
10	1000	0.2	0.1
12	1000	0.2	0.15
13	1000	0.2	0.05
14	1000	0.2	0.051
15	1000	0.2	0.052
16	1000	0.2	0.053
17	1000	0.2	0.054
18	1000	0.2	0.055
19	1000	0.3	0.1
20	1000	0.3	0.15
21	1000	0.3	0.05
22	1000	0.4	0.1
23	1000	0.4	0.15
24	1000	0.4	0.05
25	1000	0.5	0.1
26	1000	0.5	0.15
27	1000	0.5	0.05

Table A.7: List of used hyper-parameters in a standard deviation approach for cold rent validator

N	Num. of training iterations	Threshold step size	Differential step
1	100	0.1	0.001
2	100	0.1	0.002
3	100	0.1	0.003
4	100	0.1	0.004
5	100	0.1	0.005
6	100	0.1	0.01
7	100	0.1	0.02
8	100	0.1	0.03
9	100	0.1	0.04
10	100	0.1	0.05
11	100	0.1	0.1
12	100	0.1	0.15
13	100	0.1	0.2
14	100	0.2	0.001
15	100	0.2	0.005
16	100	0.2	0.01
17	100	0.2	0.05
18	100	0.2	0.1
19	100	0.2	0.15
20	100	0.2	0.2

Table A.8: List of used hyper-parameters in a standard deviation approach for area validator

N	Num. of training iterations	Threshold step size	Differential step
1	10	0.1	0.05
2	10	0.1	0.1
3	10	0.1	0.15
4	10	0.1	0.2
5	10	0.1	0.25
6	10	0.1	0.3
7	10	0.1	0.35
8	10	0.1	0.4
9	10	0.1	0.45
10	10	0.1	0.5
11	20	0.2	0.1
12	20	0.2	0.15
13	20	0.2	0.2
14	20	0.2	0.25
15	20	0.2	0.3
16	20	0.2	0.35
17	20	0.2	0.4
18	20	0.2	0.45
19	20	0.2	0.5
20	20	0.3	0.1
21	20	0.3	0.15
22	20	0.3	0.2
23	20	0.3	0.3
24	20	0.3	0.35
25	20	0.3	0.4
26	20	0.3	0.45
27	20	0.3	0.5

Table A.9: List of used hyper-parameters in a multivariate normal distribution approach

Declaration

I declare that this Bachelor Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used.

City Date Signature