FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Relocation Strategies for Free-Floating Car Sharing Operators

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Business Informatics

eingereicht von

Philipp Michael Manfred Dreßler
Matrikelnummer 00926545

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Inf. Dr.-Ing. Jürgen Dorn

Wien, 06.03.2018

(Unterschrift Verfasser) (Unterschrift Betreuung)

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Relocation Strategies for Free-Floating Car Sharing Operators

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Philipp Michael Manfred Dreßler
Registration Number 00926545

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Inf. Dr.-Ing. Jürgen Dorn

Vienna, 06.03.2018

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Philipp Michael Manfred Dreßler
Hahngasse 22/7, 1090 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

Acknowledgements

I would first like to thank my girlfriend Susanne for always encouraging me to keep on track and being incredibly supportive and loving during my studies. This accomplishment would not have been possible without you. Thank you for being there. Whenever.

No less important was the endless and loving support of my parents. You are inspiring role models to me and this work, as well as my graduation, is clearly dedicated to you. A further big thanks goes out to my whole, great family. You always had the right answers on all the really important questions I had.

Thanks a lot to Trevor for your corrections, who made this piece so much better.

Finally, a big shout-out to this awesome bunch of people that made my life between Vienna and Attnang so enjoyable over the last years. First and foremost to Leo, who kept pushing me during all the courses we took together.

Abstract

Free-floating car sharing offers a convenient alternative to public transportation in urban regions. However, due to the flexibility of this approach, operators must be able to regularly respond to imbalances in the car sharing system in terms of car location and availability. In order for free-floating systems to operate efficiently, vehicles need to be allocated geographically to meet user demand. Naturally, such systems get imbalanced over time, and the need for relocating vehicles emerges.

This thesis deals with the imbalance problem by presenting a method of data collection and analysis that offers deeper insight into user behavior in free-floating car sharing systems. The method introduced here will contribute to the improvement of operators' relocation strategies. It does so by answering the question of how vehicles in free-floating car sharing systems should be geographically relocated to best fulfill customers' needs depending on time of day, weather conditions, and points of interest in a city.

The proposed method represents a behavioral, generic approach to the discovery of knowledge about user behavior based on trip data from free-floating car sharing systems. It consists of the collection and normalization of trip data, as well as the subsequent analysis of trip locations, and the application of density-based clustering techniques to find correlations between points of interest and high-demand areas in the car sharing systems. By combining clustering results with geographical points of interest in a geographic information system tool, a wide range of hypotheses can be evaluated. The findings offer recommendations that can potentially improve relocation strategies.

The proposed method is easily applicable for various stakeholders interested in the topic. Using multiple car-sharing providers in Vienna, Austria as a case study, a concrete implementation is presented using open source software. In order to illustrate the validity and applicability of the proposed method, a number of hypotheses are proposed and evaluated. Implications drawn from the results provide three suggestions for the improvement of relocation strategies.

Kurzfassung

Free-floating car sharing stellt eine komfortable Alternative zum öffentlichen Verkehr in Städten dar. Die Flexibilität dieses Ansatzes bedingt die regelmäßige Optimierung der car sharing Systeme in Hinblick auf Position und Verfügbarkeit der Fahrzeuge, sodass diese dort zur Verfügung stehen, wo Kunden sie benötigen. Die Systeme tendieren von Natur aus zu einem unausgeglichenen Status, wodurch der Bedarf an Relokation entsteht.

Die Arbeit behandelt dieses Problem, indem eine Methode gezeigt wird, welche tiefere Einblicke in das Nutzerverhalten von free-floating car sharing Systemen gibt. Es soll zur Verbesserung der Relokationsstrategien der Anbieter beigetragen werden, indem beantwortet wird, wie Fahrzeuge in solchen Systemen geographisch repositioniert werden sollen, um den Kundenbedarf bestmöglich, in Abhängigkeit von zeitlichen, meteorologischen und geographischen Einflussfaktoren, zu decken.

Die vorgestellte Methode ist eine generische Vorgehensweise zur Wissensermittlung über das Nutzerverhalten in free-floating car sharing Systemen, welche auf empirischen Fahrtdaten basiert. Sie besteht aus dem Sammeln und Normalisieren von Fahrtdaten sowie der anschließenden Analyse dieser mit Hilfe von dichtebasierten Clustering-Verfahren, um den Zusammenhang zwischen Orten von Interesse und Standorten mit hoher car sharing Nachfrage zu ermitteln. Die Kombination von Clustering-Ergebnissen und geographischen Interessenspunkten in einem geographischen Informationssystem erlaubt die Evaluierung von vielfältigen Annahmen über die Systeme, welche zur Verbesserung der Relokationsstrategien beitragen können.

Der vorgestellte Ansatz ist leicht von Stakeholdern anwendbar. Die Arbeit zeigt auch eine konkrete Implementierung der Methode basierend auf Open Source Software. Es wurden einige beispielhafte Hypothesen durch diese Lösung evaluiert, um die Anwendbarkeit der Methode zu zeigen. Dieses Vorgehen resultierte in drei Vorschlägen zur Verbesserung von Relokationsstrategien.

Contents

CHAPTER 1

Introduction

1.1 Motivation

In the wide range of Smart City concepts, Smart Mobility is one of the most important ones for making life more enjoyable in big cities. Constant traffic growth, and thus the lack of parking spots, charging stations, and other perquisites for individual transportation, bring up the need for alternatives to individual car ownership. City governments, as well as operators of alternative travel options, are increasingly turning their attention toward shared mobility concepts as a viable solution to traffic congestion.

Moreover, there seems to be a clear cultural trend toward sharing and using over possessing in the recent years, especially for younger people living in urban regions that rely on individual transportation only rarely or infrequently. In general, many people living in urban regions are not dependent on a car, since public transportation is usually more readily available compared to rural areas. Nonetheless, most larger cities are still far from being classified as regions without the need for individual transportation. Therefore, car sharing has become an attractive alternative to car ownership for citizens living in urban areas.

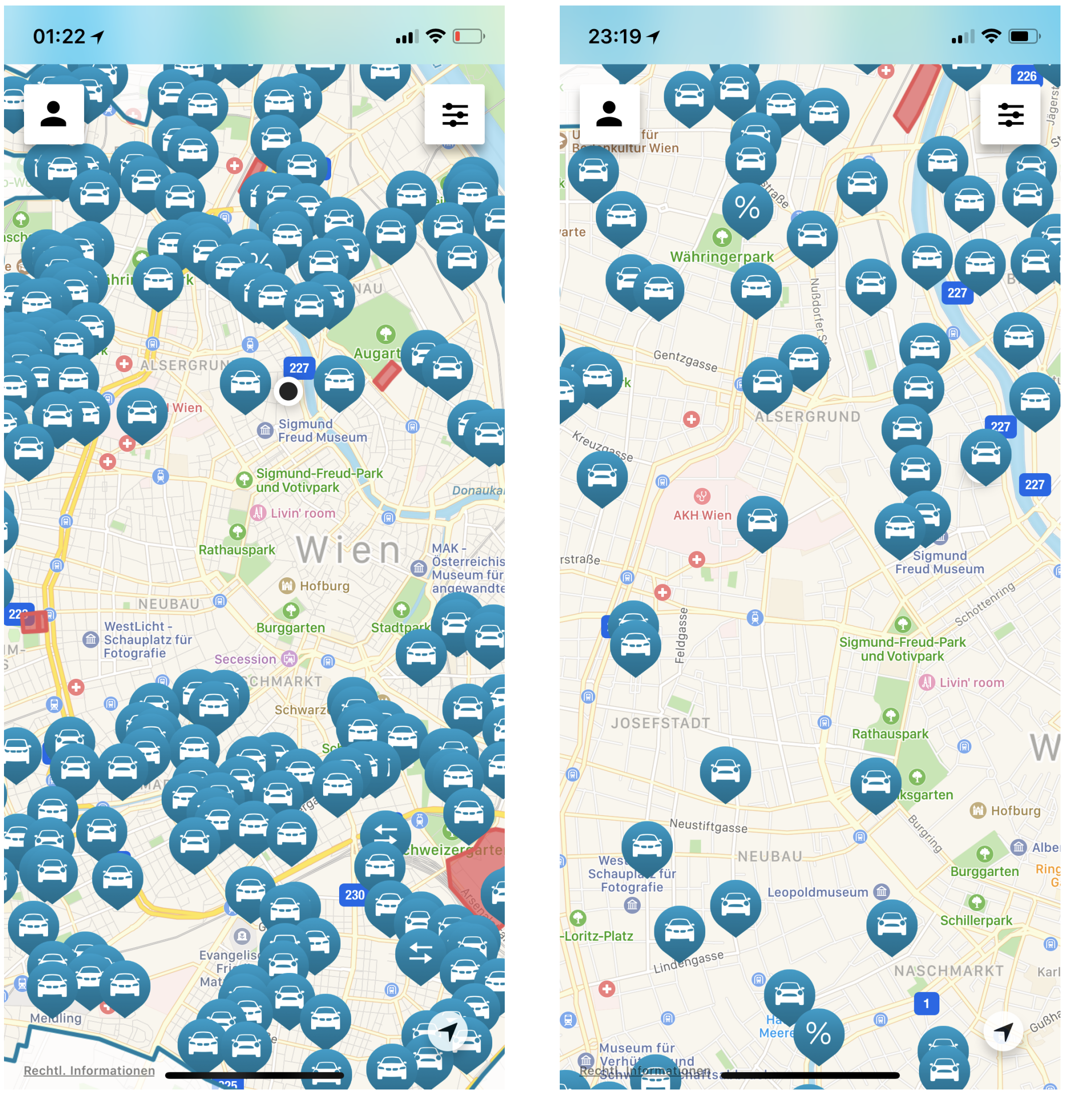
The wide distribution of information and communication technologies, especially mobile devices like smartphones with integrated Global Positioning System (GPS), has made more flexible approaches of Smart Mobility possible. Free-floating car sharing, for example, offers a convenient alternative to public transportation. However, due to the flexibility of this approach, operators must be able to regularly respond to imbalances in the car sharing system in terms of car location and availability. In order for free-floating systems to operate efficiently, vehicles need to be allocated geographically to meet user demand. Naturally, such systems get imbalanced over time, and the need for relocating vehicles emerges. Correcting this imbalance is one of the most pressing issues in free-floating car sharing systems.

The present thesis deals with this problem by presenting a method that offers deeper insight into user behavior in free-floating car sharing systems. The findings have implications for car sharing operators, city governments, and other stakeholders of this topic, who will be able optimize their operational planning strategies through a more scientific understanding of user behavior.

1.2 Problem definition

Commercial car sharing operators have been around for quite a while, offering conventional car rental by day or pay-by-hour station-based concepts where cars, once rented, have to be returned to the parking spot or station they were picked up from. Recently, companies offer so-called free-floating car sharing systems, where cars are rented by minute and can be dropped off at an arbitrary, legal location within a defined operating area. Usually, such operating areas are extended city centers in urban regions enriched with certain spots outside of these core areas, e.g. an airport. This approach is usually very user-friendly concerning flexibility; cars can be picked up spontaneously and dropped off at any available parking spot near the desired destination a user wants to travel to. On the other hand, potentially, cars in these systems won't be geographically distributed to perfectly fulfill customers' needs for very long. After a while, there will be a lack of cars in high demand regions, cars in low demand areas might not be used for quite some time, or vehicles might accumulate in certain spots in the city. These effects are usually highly dependent on the time of day, day of the week, or even the season. Furthermore, there may be a high dependence on the geographic and demographic characteristics of the particular city. A city's public transportation infrastructure may influence system balance as well. Weather conditions could also play a major role in the decision process of a citizen in terms of which sort of mobility option he or she chooses. Figure 1.1a shows an example of a car sharing system in a state of imbalance. The distribution depicted in the panel on the left could obviously be optimized using relocation strategies. It is clearly visible, that there are almost no vehicles available in the center of the city, where a larger number of potential customers would be willing to pick up a car.

To counteract potential profit losses and low car-usage rates arising from these problems, free-floating car sharing operators have to implement relocation strategies for their vehicles. These can be either user-based, where users relocate cars for certain incentives, or operator-based, where employees or contractors are delegated by the operators to do so. User-based relocation strategies could encourage a user to take a longer route to a car, or encourage a user to drop-off a vehicle in an area where cars are needed in the system. As incentives, the operator could offer special discounts or free driving time to the relocating user. Operator-based strategies usually focus on relocation agents delegated by the car sharing operators to relocate cars. This could be done by tow trucks or carpooling agents. Furthermore, the agents could use public transportation or folding bicycles to get from one car to the other. Often, the agents also take care of tasks like cleaning or refueling the cars while performing relocations. Good relocation strategies will distribute cars in the operating area so that they can be reached and rented by potential customers within a reasonable time span, and thus increase their availability and usage time. Figure 1.1b shows a state of a free-floating car sharing system, where vehicles are more efficiently distributed. It shows a similar map section as figure 1.1a, while the lack of vehicles in the affected region is less significant. It seems likely that the screenshot was taken shortly after a number of relocations were performed. The problem described above now leads



(a) Poor allocation (b) Better allocation

Figure 1.1: Two different states of the car sharing system in Vienna shown in the DriveNow [10] IOS App.

to the question that should be answered in this thesis: **How should vehicles in free-floating car sharing systems be geographically relocated to best fulfill customers' needs depending on time, weather, and points of interest in a city.**

Hypotheses

In order to answer the stated research question, a number of hypotheses are formulated and tested with empirical data. The hypotheses were developed based on a literature review 2 that addresses the key issues at the heart of the thesis: geography and infrastructure, time factors, and weather factors. A number of papers presented in the proceeding chapter deal with the influence of geographical or structural factors on car or bike sharing usage, e.g. the work of Braun, Hochschild and Koch [1]. The time factor plays a major role in much of the research in this area, while weather is also dealt with (e.g. Vogel and Mattfeld [34] and the work of Chang et al. [6]). Data availability is also an influential factor in terms of which hypotheses can be tested. Finally, the results will be discussed in light of car sharing customer behavior in the city of Vienna, Austria. The hypotheses that will be tested are the following:

Hypothesis 1: Daytime, rain, shopping facilities

Cars are very valuable when it comes to transporting goods like groceries or other items of daily use. It seems obvious that people might find it convenient to use cars for traveling home with their purchased items after a shopping trip. A lot of cities have certain streets and pedestrian areas, where shopping can be done very comfortably, as shops of different types are accumulated in compact areas. In case of rainfall or bad weather in general, such areas can be quite unpleasant to shop at because people have to walk outside when switching shops. Indoor shopping malls minimize weather influences, and it seems reasonable that people prefer shopping in malls when the weather is poor. Shopping malls are only one example of indoor shopping facilities. This logic could apply, for example, to indoor markets or temporal shopping events. After shopping at indoor facilities, it would be more comfortable to transport the shopped items home by car, especially if it is raining. This leads to the first hypothesis that shall be examined: **'During shop opening hours on rainy days, people tend to pick up car sharing vehicles around indoor shopping facilities.'**

Hypothesis 2: Daytime, amenities

Urban regions and cities usually offer a variety of travel options for people conducting their daily business during the daytime. While every person has their own daily routine, there are a number of amenities in cities that are visited regularly during the day for a higher number of people, and therefore have a high visiting frequency. Examples of those points of interest are universities, business parks, or amenities for health care. Sticking to latter example, the reasons for picking car sharing over alternatives could be the following: When people have to go to day clinics or hospitals, public transportation, as an alternative to car sharing usage, could be quite inconvenient. People might feel physically unhealthy, and therefore do not want to be around other passengers. They may desire to get to the clinic or hospital as quick as possible, or might not be in a state of getting there without the need of a friend or relative to drive them. They may also have a fixed-appointment to meet. The classic solution to this problem is to take a taxi. Still, it seems likely that car sharing would be a viable option in this scenario. Thus, it could be interesting to look into car sharing usage for these situations. The second hypothesis stated is the following: **During the daytime, customers tend to drop off car sharing vehicles near amenities for daily business.**

Hypothesis 3: Evenings, rain, leisure activities

Leisure and recreational activities might be popular, especially in the evenings when many people are finished with work for the day and have more free time. While the possibilities for such plans are fairly wide on evenings with dry and warmer weather, the possibilities narrow down on rainy days. For this purpose, it might be interesting to look into the influence of such conditions on car sharing usage. For example, when it comes to leisure activities enjoyed by people living in urban regions, going out to watch a movie in a cinema might be one of the most popular evening activities. While this assumption might not hold in the summer months, as the alternative activities in evenings are numerous, it could indeed hold on evenings of rainy days. Movie theaters are often located in the inner parts of cities, where naturally the free-floating car sharing operating areas are also located. With the next hypothesis, this assumption is examined: **On rainy evenings, customers tend to drop off car sharing vehicles around indoor leisure facilities.**

Hypothesis 4: High temperatures, public transportation

Usually, the public transportation infrastructure is very good in urban regions that also host car sharing systems. Still, there may be certain situations where citizens have a stronger need to use individual transportation, like using a car sharing vehicle. One could imagine that temperature has something to do with such decisions. For example, on very hot days, public transportation trains or buses could be inconvenient. This could push users to the decision to take their trip with a car sharing vehicle instead. The last hypothesis takes a closer look at the influence of temperature on car sharing usage: **On hot days, car sharing around subway stations increases compared to days with moderate temperature.**

1.3 Aim of the work

The aim of the thesis is to offer a number of suggestions for free-floating car sharing operators for how to relocate their cars more efficiently within a defined operating area. After collecting, transforming, and analyzing given data, the stated relocation strategy suggestions shall be explained in written form and illustrated by a geographical information system tool. The city of Vienna, Austria is used as a representative example for regions where free-floating car sharing is offered. In addition to these concrete suggestions, the general method introduced in this thesis is an important outcome of the work as well. Free-floating car sharing operators should be able to use the proposed method to get better insight into the patterns of car sharing usage within their operating areas. The method shows how empirical car sharing data can be evaluated with respect to various influencing factors, and can therefore be an important contribution to the choice of operator's relocation strategies during different conditions within their operating areas.

A technical outcome of the thesis is a prototype application written in Java that collects relevant data and transforms them into an evaluable form. Furthermore, a number of PostgreSQL statements on how to preprocess the data, exploratory analysis, as well as clustering on the given data will be provided.

Finally, the thesis gives general insight into free-floating car sharing user behavior with respect to certain influential factors like different weather conditions and time of day. The analysis performed on the points of interest in the city could also be of value for future work in this area.

1.4 Methodology

As a first step, a literature review was conducted to identify important factors that influence the positioning of cars in free-floating car sharing systems. According to the findings of this research, and considering the existing and available datasets, a number of hypotheses about free-floating car sharing usage were formulated. The chosen hypotheses were already described in chapter 1.2.

As the stated hypotheses are highly dependent on time, weather data, points of interest in a city, and of course, empirical data from car sharing operators, the relevant data needed to be collected before doing any evaluation of the hypotheses. To collect the empirical data from web pages of car sharing operators, a Java prototype was written. The tool collected relevant data over a time span of a few weeks and transformed this data to evaluable form. The data was then written into a relational database with an add-on for spatial and geographic objects for further investigation. The data concerning weather was provided by the Department of Building Physics and Building Ecology at the Vienna UT [9]. Lastly, spatial data concerning relevant points of interest in Vienna was taken from OpenStreetMap [21].

Before doing actual evaluations, the collected and available data had to be generally cleaned and normalized. This was performed by preprocessing the empirical data in the database in order to have a solid dataset ready to do the subsequent evaluations with. After this step, the data could be analyzed with respect to certain influence factors like time of day or weather situation. Furthermore, the weather data was analyzed and combined with the car sharing trip data. The relevant points and areas of interest were extracted from OpenStreetMap and loaded into the chosen tool for geographical information system visualization.

The data is held in a PostgreSQL [27] database. As PostgreSQL does not support geographical and spatial datasets out of the box, the PostGIS extension [25] was added to the database installation. The choice for the necessary tool for graphical representation of the data is QGIS [28], which integrates well with PostGIS and OpenStreetMap, and offers tooling support for visualizing spatial data in general.

The actual evaluation of the hypothesis was done by performing a cluster analysis on the empirical vehicle trip data, visualizing the generated clusters in QGIS, and manually interpreting their coherence to points of interest on the city map. The cluster analysis was done with respect to the data relevant for the hypothesis under examination, e.g. only data from time slots with rainy weather. The clustering was performed using functions provided by the PostGIS extension directly on the PostgreSQL database.

Finally, the evaluated hypotheses were used to formulate a set of relocation strategies applicable for car sharing systems in urban areas. The generic approach of the thesis is outlined in a way, that it is easily applicable to various hypotheses, car sharing operators could have. The hypotheses formulated in this thesis are only to be seen as a few examples for a wide field of use and are used to outline the presented method in a concrete application.

The research method of the thesis can be seen as a behavioral approach, as data will be collected, quantitatively analyzed, and experimented with. Based on the findings of these processes, certain theories will be deduced. However, the development of the Java prototype brings elements of design science into the research process.

1.5 Structure of the work

The problems of relocating vehicles efficiently in a free-floating car sharing system, as well as the potential solutions addressed in this thesis, were described in the introductory chapter 1. Chapter 2 sums up the current state of the art of the topic and its related work. The chapter was split up into subsections discussing different relocation approaches. As a next step, the thesis outlines the relevant data 3 that was used to perform the analysis necessary for the evaluation of the stated hypotheses. Chapter 4 describes the actual solution developed within the scope of the present master's thesis. It is divided into subsections dealing with the various steps necessary to accomplish the aim of the work, like developing the Java prototype, doing an exploratory analysis of the data and the actual evaluation of the stated hypotheses. The generic method applicable for a lot more than the stated hypotheses is also described in this chapter. The results chapter 5 outlines suggestions for car sharing operators resulting from the analyses, compares the results with related work, and discusses issues left open within the thesis. The closing chapter 6 sums up the thesis, states further steps for future work, and discusses which data could be useful for deeper insight into the topic. The data models A as well as the developed source code B are outlined in the appendix of the thesis.

CHAPTER 2

State of the Art

The literature on car sharing distinguishes between two categories of systems. The two categories are: The non-floating approaches, also named station-based or traditional car sharing, and free-floating approaches. Car sharing, as referred to in this thesis and related literature, is the process where a car is used by different people for a fee. Usually, such services are offered by commercial operators.

Non-floating car sharing refers to systems where cars have to be picked up and returned at certain, predefined rental spots. In practice, those spots could be either car parks or single parking lots where a vehicle has a fixed assignment. These systems can be further divided into one-way and two-way car sharing approaches. The former allows for the pickup of a vehicle from one station and giving it back at another one, while the latter requires a user to return a rented car at the same exact station that it was picked up from.

In free-floating car sharing systems, cars are usually allocated within a given operating area. Users, also referred to as customers or members, can pick up an arbitrary vehicle from the system, perform a trip, and return the car at any vacant parking lot within the operating area. Often, those operating areas are enlarged city centers or other population-dense areas. In these systems, the billing of a trip is usually done by the minute and users can find nearby cars with a smartphone app or website.

Furthermore, the literature discusses two types of relocation strategies. Relocation is regularly needed to ensure that vehicle supply adequately meets user demand. Relocation strategies can either be user-based or operator-based. User-based relocation approaches usually offer customers certain incentives to perform relocation trips, or at least adapt their trips to improve car distribution within the system. Operator-based relocation is based on employees performing dedicated relocation trips. Other approaches employ some combination of strategies from both user-based and operator-based models.

Literature concerning the similar concept of bike sharing also distinguishes between the categories explained above. While station-based approaches in bike sharing often use terminals to process the rental procedure, free-floating approaches to bike sharing otherwise use very similar concepts to those employed in car sharing systems.

2.1 Related work

When examining relocation of vehicles in free-floating car sharing systems, there are a few related fields of study that are relevant for current discussion. Those include fields like bike sharing, operational planning of taxi locations, or container shipping. In this section, an overview and the relation to the topic under examination shall be given.

The problem of repositioning certain items like vehicles, containers, or other things of interest, is a common problem in logistics. In any case, where systems can have a physical or geographical unbalanced state, the general need for relocation is given. Effective relocation is required for logistical systems to operate properly. The following sections outline how this problem is addressed in other transportation contexts.

Container shipping

In the example of container shipping, both Stahlbock and Voß [32] and Song and Dong [31] describe approaches to deal with imbalanced states in such systems. The former state that international container shipping is highly affected by imbalances, since Asia has large export flows of containers and comparatively small import flows. The authors propose methods to address this problem like foldable containers, as well as data mining in empty container management. They identify a remarkable research gap in the combination of data mining and container management. Song and Dong [31] compare methods of repositioning empty containers in container shipping by doing simulations using the different strategies.

Taxi deployment

The geographical positioning of taxis in a given business area is a field that several authors have dealt with. The correlation with the problem of relocation in a free-floating car sharing system is quite obvious. In a taxi system, cars can either wait at a station for their customers, or drive within an operating area or hope for ad-hoc business. In most of the cases, taxi companies and drivers use a combination of both. Taxi companies want to minimize the downtime of their cars, and therefore have to deal with similar problems as car sharing operators when it comes to maximizing their profit. Taxi companies also need to focus on geographical points in a city where cars are needed at certain points during the day or night.

Chang et al. [6] suggested a model for predicting taxi demand in certain areas of a city based on time, location and weather. They used data mining approaches to do so. This article is of special interest for this thesis as the authors collect, filter, and cluster spatio-temporal data and interpret the results. A few different clustering algorithms were taken into account in this work showing that the choice of the clustering method has an outsized influence on the outcome of such approaches.

Bike sharing

Bike sharing is arguably the concept most related to car sharing systems. Traditionally, and just like in car sharing, bike sharing operators installed stations where bikes can be rented and returned. This approach is very similar to non-floating car sharing systems. Citybike Wien [8] is an example of an operator using a station-based bike sharing approach in Vienna. In recent years, free-floating bike sharing behaviors have gained more scientific attention, particularly as operators emerged in different cities. For example in Vienna, two operators, namely OFO [20] and oBike [19], started their service recently. Free-floating bike sharing, naturally, has a lot in common with the topic under examination. Relocation problems occur in such systems regardless of the used type of vehicle, be it bikes or cars. Both bikes and cars are used by individual consumers, and unlike with taxis, there is no dedicated driver for every vehicle in the system.

When it comes to bike sharing, Vogel and Mattfeld [34] proposed a way to use clustering methods to calculate demand forecasts in one-way, station-based bike sharing systems. They used two years of operational ride data from Vienna's bike sharing system *Citybike Wien* to perform linear regression and cluster analysis. After performing a preprocessing phase to aggregate and normalize the ride data, they analyzed it with respect to their temporal patterns. In the next step, the authors did a regression analysis on the influence of weather on the bike sharing usage in the system. Finally, a cluster analysis was done concerning the usage patterns of the different bike sharing stations. More concrete, the analysis targeted the determination of groups of stations with similar hourly pick-up and drop-off patterns. This resulted in a set of 5 clusters that were interpreted to be two commuter station clusters representing 1) people going to work in the morning and 2) back at night, 3) leisure use 4) tourist use, and finally 5) an average station cluster for stations that could not be interpreted to have a certain regular usage.

A recent paper from Caggiani, Ottomanelli, Camporeale and Binetti [2] deals with free-floating bike sharing. The field of study is fairly new and these bike sharing systems have similar characteristics to free-floating car sharing. The authors use different clustering techniques to divide the operating area into zones based on the spatio-temporal features of a city. Using these zones, relocation strategies are deducted using non-linear autoregressive neural networks. The clear aim of this work is a forecast model for free-floating bike sharing systems, which would also be adoptable for car sharing approaches.

2.2 Non-floating car sharing relocation

As station-independent, or free-floating, car sharing systems have rapidly developed in recent years due to their high dependence on information systems, scant research exists on this topic compared to station-based approaches. These can be categorized by traditional round trip systems, where a customer returns the car in the exact same station that they picked it up from, or multiple station shared vehicle systems. The latter consist of different stations where cars can be picked up and returned at. While relocation is not necessary in round trip systems, multiple station shared vehicle systems can quickly become imbalanced concerning the number of cars per station. Free-floating car sharing systems can often be abstracted to multiple station shared vehicle approaches since the high demand regions in their operating areas could be seen as stations in traditional rental systems. Due to this fact, a further look into the field of station-based car sharing relocation will be taken.

The number of cars in a station is an important factor for the availability of station-based car sharing approaches. The correct thresholds for when to move cars in or out of stations must be determined to fulfill business requirements. Cao et al. [3] show a method for threshold triggering in an electric car sharing system. A variation of this method could also be used in free-floating car sharing systems to decide when to move cars into, or out of certain high or low demand areas.

E. M. Cepolina and A. Farina [5] give a good overview of existing car sharing approaches with a particular focus on vehicles relocation techniques. They categorize relocation strategies in user-based and operator-based techniques. Operator-based relocation is usually performed by employees of the car sharing operator that move cars to regions or stations in which they are needed in. User-based strategies usually offer customers incentives for their contribution to relocating cars. Furthermore, the paper explains two techniques for relocating cars between stations or in an operating area. Towing, being a operator-based technique, moves cars chosen for relocation via a dedicated towing vehicle or other cars in the system. Ride sharing lets operator employees, customers, or a mix of both share trips between stations to perform relocations on other car sharing vehicles afterwards. Cepolina and Farina also discuss some concepts relevant when discussing free-floating car sharing. They characterize such a system with the properties of instant access, open-ended reservation, and one-way trips. The authors discuss the capillarity of these car sharing approaches. Capillarity is defined as the degree of diffusion of vehicles within the application area of the transport system.

Braun, Hochschild and Koch [1] developed a regression model describing the influence of different determinants on the usage intensity of car sharing in different regions in the city of Tübingen in Germany. They categorize the influence factors on car sharing usage into two classes. The first class consists of structural influence factors, such as population density, public transportation, distance from the city center, parking situation or pedestrian- and cycle-friendliness. The second class are social-demographic influence factors like age, education, income, personal and political attitude, or family status. Based on data from a local car sharing operator and information about the mentioned influence factors in the different town districts coming from the city government, a multiple regression analysis was performed. The authors found that, among others, the factors age group (30 to 44 years), votes for the Green Party, or population density have strong positive impact on the use of car sharing in a region. Factors with negative impact were as well found, e.g. the distance to a public transportation station or the percentage of foreigners in a certain region. The car sharing operator, from which the data was collected, only offers a station-based car sharing system. Nevertheless, the present regression model could also be derived in a free-floating car sharing system.

Morency et al. [18] showed typical types of user behavior by applying clustering methods to long-term historical data of a non-floating car sharing system in Montreal. They could distinguish between high- and low-frequency users, where the latter ones are the vast majority of the system under inspection. Moreover, they find five weekly patterns of usage frequency, where one is the dominant one. This dominant pattern is one where users use car sharing with a low frequency during weekdays and an increased one on weekends. Those weekly patterns change mainly during the holiday periods, so in summer as well as December and January.

2.3 Free-floating car sharing relocation

Compared to non-floating car sharing relocation approaches, there is less literature on free-floating car sharing relocation. Still there are a number of articles dealing with the topic. The following section will summarize them.

Herrmann, Schulte and Voß [13] conducted a survey offering interesting insights into car sharing users' acceptance of a system. They found that the majority of users would accept a maximum walking distance of 500 meters to an available vehicle, otherwise they would switch to another public transportation mode, or to a self-owned car. A brief majority (55 percent) of the survey participants would accept a maximum waiting time less or equal to 15 minutes before choosing another alternative of transportation. These findings can be taken into account to make assumptions about the size of relocation zones, in which the operating area of a free floating car sharing system should roughly be divided into. Furthermore, a vast majority of survey participants would accept longer walking routes to cars or determine a driving destination at the beginning of a trip for pricing reductions. Based on the findings of the survey, the authors derived a number of user-based relocation strategies. Pricing discounts play the major role in all of them. In concrete, the proposed strategies are the following: incentives to book more distant vehicles, incentives for more distant drop-off locations, paid relocation, and demand pooling. Those four strategies were evaluated in a discrete event simulation model.

Another work dealing with car sharing member behavior was published by Kopp, Gerike and Axhausen [14]. They compare the mobility patterns of members of free-floating car sharing members with those of people who do not use car sharing at all. To do so, they invited active customers of DriveNow [10] as well as non-members to take part in a survey in return for certain incentives like money or free trips in the car sharing system. After choosing a representative sample of participants, they developed a GPS-smartphone app that would track survey participants while they were en route for the time period of a week. To get even better insight, they asked participants to define the characteristics of their trip that was about to start. As a result, the authors could deal with data that is not only rare GPS-routes, but also enriched with the trip purpose (like education, work or business), the mode of transportation (car, bike, walk etc.) as well as socio-demographic factors about the survey participants. The results of this article show, that car sharers have a significantly higher education level and income compared to non car sharers. Furthermore, car sharers are more multi-modal in their choice of transportation, meaning they recorded more and shorter trips with diverse transportation modes. They also take more advantage of the available alternatives in public transportation and arrange their trips to be more flexible.

In another study, relevant statistical data collected from the car2go API [4] is provided by Kortum and Machemehl [15], who chose Austin, Texas as the playground for their case study. After collecting data about customers and vehicle usage provided by car2go, they transformed it to an evaluable form by cleaning it and doing an exploratory analysis. For their work, they had the opportunity to look into member data, providing insights into e.g. the location of their homes. They found that those member locations are not only accumulated downtown, but also around the university and just outside of the inner city. They did a breakdown of the usage data by day of week, hour of day or distance of time traveled. By analyzing the membership data, the authors were able to do a membership prediction analysis and thus offer information for car sharing operators about their future operating area. Finally, they proposed a way to do allocation modeling, i.e. relocation using C++ programming to cover the demand of car sharing vehicles during different time periods.

Weikl et al. [36] recently presented a simulation framework for proactive relocation strategies in free-floating car sharing systems. They use a relocation model developed by Weikl and Bogenberger [35] in a simulation based on historical vehicle positions in Munich. In this simulation, relocation as well as user-based vehicle movements are performed. The underlying relocation model is run in different time intervals and the scenarios are analyzed regarding to their satisfaction of the calculated demand. To do so, the service area of the free-floating car sharing system was divided on a macroscopic scale into zones and, on a finer scale, into microscopic hexagons used as sources and destinations of relocation movements. The authors show a significant improvement of the average percentage of satisfied demand when using relocations in free-floating car sharing systems. Another interesting task covered in this paper is the analysis of the historical vehicle positions and movement data. The data is broken down and illustrated by time of day, weekday as well as (difference of) inflow and outflow in certain regions in the business area of the system. This breakdown shows that in the morning, there is a tendency of user-based rides going towards inner city regions, as well as destinations out of the core operating area, like the airport. In the evening the trend is more or less the inverse of the morning one. On weekends, no clear trend is observable as sources and destinations of movements are more scattered over the whole business area. In general, the authors assume that by trend, vehicles conglomerate at the boundaries of the operating area. The underlying relocation model [35], when executed, is divided into a number of execution steps. For the present thesis, step I is of special interest as is outputs the optimal vehicle numbers per zone for a certain target period based on an analysis of the given historical movement data. In detail, zones with historical vehicle shortage and surplus, as well as historical demand indicators per zone will be supplied after the execution of this step. The optimal vehicle distribution for a given time period can the be calculated and an optimization model can calculate the optimal vehicle relocations.

While Weikl et al. developed a full simulation framework, Paschke, Balac and Ciari [24] focus on an agent-based simulation that extends the possibilities of the open source simulation framework MATSim by adding relocation activities to a system of agents performing daily routines. The simulation deals with a flee-floating car sharing system, but could also be applied to a station-based system. The simulation was performed based on the road network of Zürich. The operating area was, like in approaches described before, divided into smaller demand zones with a reasonable size. The size of this polygon was chosen so that customers, as with other studies, would take the effort to walk to the next car within a given zone. A fundamental difference from approaches described before is that the vehicle demand in the zones of the operating area is calculated based only on the last iteration of the simulation, rather than complete historical data from the car sharing system. This makes the demand calculation very dynamic and able to adapt to short-term changes of user behavior. In this work, a very simple relocation strategy was used. The zones where ordered by demand and vehicle relocations were performed from the zone with the currently highest surplus of cars towards the zone with the highest number of missing cars with respect to its demand.

Schulte and Voß [30] introduced a decision support approach by combining multiple modules. The demand forecast module calculates demand for certain areas in the operating area. It uses live data from car sharing provider's APIs in combination with regression models for long-term and neural networks for short-term demand calculation. The relocation planning module combines the live data with the results of the demand forecast module to select and weight optimal relocation strategies. Finally, an evaluation module performs a discrete event simulation to evaluate the relocation plans with respect to their cost, emissions, and impact. Another interesting aspect of this work is the usage of spatial clustering approaches to define relocation zones based on customer demand and requirements. This is a clear differentiation from the related work described before, as the authors use zones with fixed, predefined sizes for their relocation approaches.

2.4 Comparison and summary of existing approaches

In the last 5 years, free-floating car sharing systems has attracted significant scholarly attention. All the literature presented agrees on the need for improved relocation in car sharing approaches, and its significant impact on the usage of such systems. There seems to be a major focus on demand prediction and decision support in the existing literature. The most commonly used methods are regression analysis, as well as different forms of simulation.

In most of the papers under examination, historical data from existing car sharing systems or operators was used. When it comes to relocation approaches, the data is often used to define zones in the operating area. These zones are then used to perform the relocation simulations. For this purpose, clustering methods were used in some of the papers. In case of investigating the influence of points of interest or socio-demographic aspects on free-floating car sharing usage, regression models where the most common methods used.

To the best knowledge of the author, there are no papers that use geographical or spatial density-based clustering methods to examine the coherence of certain points of interest and car sharing usage in free-floating systems. Plenty of questions seem to be unanswered concerning the influence of public transportation and weather on usage patterns of customers when performing trips in different life situations. Another new method used in this thesis is the combination of trip data from different car sharing operators, which could also not be found in state of the art related work.

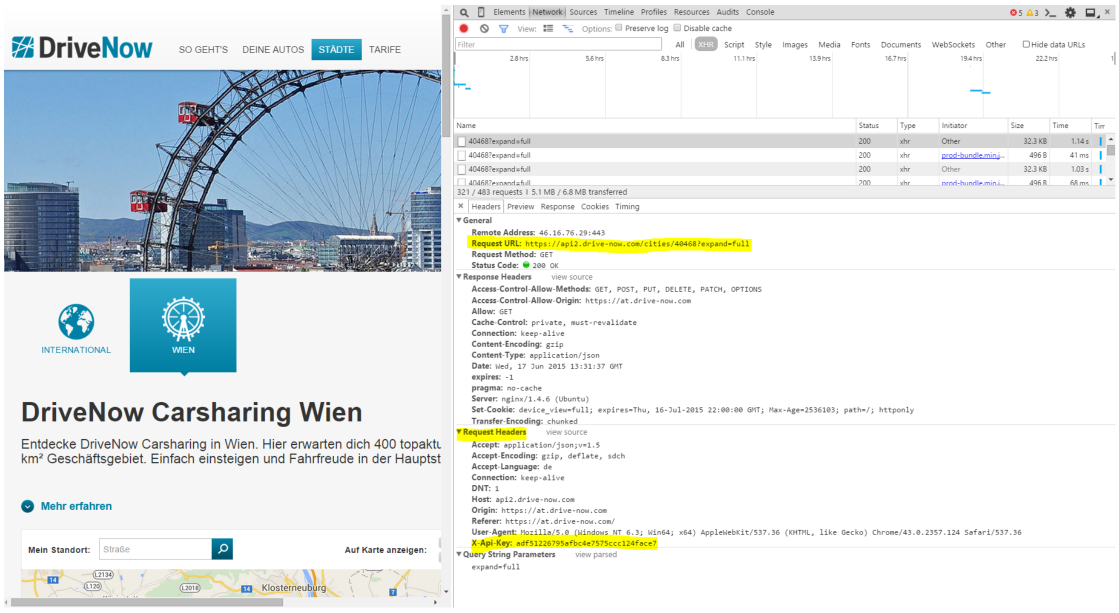
CHAPTER 3

Relevant Data

3.1 Empirical data from car sharing operators

The key source of data used for the current thesis is empirical data coming from free-floating car sharing operators, e.g. DriveNow [10]. They usually provide a map on their website that shows the geographical position and status (like fuel level or cleanliness) of the vacant cars in their fleet so customers can comfortably find them. In most cases, this dataset is transmitted to the client as a JSON document and can therefore be interpreted quite easily. In recent times, operators set their focus on their smartphone apps. The possibility of performing the booking process for a vehicle via their website was removed, while apps were steadily improved. Today, the whole rental process (locating, booking, and managing of vehicles) has to be performed by users with the app of the operator. Nonetheless, the back-end APIs for fetching lists of vacant cars is still available and can be used by 3rd party apps and websites that provide location information about vehicles, and sometimes also support booking of vehicles directly in their solutions. Examples for those apps are WienMobil [37] or Free2Move [12]. This back-end functionality is an opportunity to collect data from different operators usable for the tasks of this thesis. The following steps were performed to gain access to the discussed APIs.

When inspecting an operator website, meaning the site with the map for users to locate vacant cars, with the developer tools of an arbitrary browser, the API calls of the current website can be examined. Figure 3.1 shows an example screenshot of the DriveNow website [10] being inspected with the Google Chrome [7] developer tools. In Chrome's developer tools, the web service call of interest can be found in the network section. To make the inspection easier, it makes sense to filter the displayed network traffic by type to only show HTMLHttpRequests (in Chrome, the type filter is called XHR). After picking the right entry, the request URL, as well as other interesting information can be displayed under the 'Headers' tab of the developer tools. In some cases, the web service calls need certain request headers to return a result. For the example of DriveNow, an X-Api-Key header property has to be attached to the request. Figure 3.1 also highlights the necessary header property.



```
Name                 Status  Type  Initiator             Size
40468?expand=full    200     xhr   Other                 32.3 KB
40468?expand=full    200     xhr   prod-bundle.min.j...  496 B
40468?expand=full    200     xhr   Other                 32.3 KB
40468?expand=full    200     xhr   prod-bundle.min.j...  496 B
321 / 483 requests | 5.1 MB / 6.8 MB transferred

Headers  Preview  Response  Cookies  Timing
▼General
   Remote Address: 46.16.76.29:443
   Request URL: https://api2.drive-now.com/cities/40468?expand=full
   Request Method: GET
   Status Code: 200 OK
▼Response Headers     view source
   Access-Control-Allow-Methods: GET, POST, PUT, DELETE, PATCH, OPTIONS
   Access-Control-Allow-Origin: https://at.drive-now.com
   Allow: GET
   Cache-Control: private, must-revalidate
   Connection: keep-alive
   Content-Encoding: gzip
   Content-Type: application/json
   Date: Wed, 17 Jun 2015 13:31:37 GMT
   expires: -1
   pragma: no-cache
   Server: nginx/1.4.6 (Ubuntu)
   Set-Cookie: device_view=full; expires=Thu, 16-Jul-2015 22:00:00 GMT; Max-Age=2536103; path=/; httponly
   Transfer-Encoding: chunked
▼Request Headers     view source
   Accept: application/json;v=1.5
   Accept-Encoding: gzip, deflate, sdch
   Accept-Language: de
   Connection: keep-alive
   DNT: 1
   Host: api2.drive-now.com
   Origin: https://at.drive-now.com
   Referer: https://at.drive-now.com/
   User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.124 Safari/537.36
   X-Api-Key: adf51226795afbc4e7575ccc124face7
▼Query String Parameters     view parsed
   expand=full
```

Figure 3.1: Examination of the DriveNow web site [10] and web service calls.

Once the necessary request URLs and headers are extracted, they can be called via a HTTP get request with a browser, from Java code or with another supporting tool. The responses of the web services are usually a JSON document representing a list of currently vacant cars in the operator's system. Listing 3.1 shows the top level of the returned JSON document when requesting data from the DriveNow API. The data set contains various information objects, like a list of the different types of cars in the system (*carTypes*), information about the operating area (*businessAreaUrl, chargingStations, petrolStations, parkingSpaces, registrationStations*, etc.) or technical information about the system (e.g. *cityImageBaseUrl, showBusinessAreaByDefault*).

The actual data that is relevant for the current thesis is contained within the *cars* item of the JSON document. It represents the actual list of currently vacant vehicles, along with different vehicle properties. This part of the raw data will be described in more detail in the following section.

Every element in the *car* JSON item represents a property of a currently vacant car in the system. An example of such a car entry is shown in Listing 3.2. There are a few elements that can uniquely identify a car within the system. The *licensePlate* element contains the vehicle's license plate value, *name* contains a user-readable unique name and *id* is a technical identifier of the car. Further elements give information about general properties of a given car. The *group*, *make*, *series*, *variant*, *modelName* and *routingModelName* key-value pairs contain the concrete model of the car. *modelIdentifier* is a technical ID for that exact model. The *color* item obviously contains the color of the vehicle while the *equipment* array can indicate certain special equipment that comes with the car. The *fuelType* pair can take the values 'D' for Diesel, 'P' for Petrol and 'E' for Electric and indicates the type of the vehicle's motorization. The *transmission* object can take the values 'M' for manual or 'A' for automatic and show the vehicle's transmis-

```json
{
   "businessAreaUrl":" ... ",
   "callCenterPhoneNumber":"+43 \/ 800\/ 070702",
   "carTypes":{ },
   "cars":{ },
   "chargingStations":{ },
   "cityImageBaseUrl":" ... ",
   "countryLabel":"Austria",
   "emergencyStatus":{ },
   "fuelTypes":{ },
   "id":"40468",
   "isFlexiblePricingAvailable":false,
   "isoCountryCode":"AT",
   "latitude":48.208328,
   "longitude":16.372702,
   "mobileBusinessAreaUrl":" ... ",
   "mobileMenuItems":{ },
   "name":"Vienna",
   "parkingSpaces":{ },
   "petrolStations":{ },
   "prolongAvailable":true,
   "registrationStations":{ },
   "routingCityName":"vienna",
   "routingCountryName":"austria",
   "showChargingStationVisible":true,
   "showPetrolStationVisible":true,
   "showRegistrationStationVisible":true,
   "taxCondition":"incl. 20% VAT",
   "transmissionTypes":{ }
}
```

Listing 3.1: Raw vacant vehicle data from DriveNow [10] API, top level.

```
"cars":{
  "count":333,
  "items":[ {... },{
       "address":["Rothergasse 2","1220 Wien"],
       "carImageBaseUrl":" ... ",
       "carImageUrl":" ... ",
       "color":"mineral_grey_metallic",
       "equipment":[ ],
       "fuelLevel":0.21,
       "fuelLevelInPercent":21,
       "fuelType":"D",
       "group":"BMW",
       "id":"WBA1C710X0J953051",
       "innerCleanliness":"REGULAR",
       "isInParkingSpace":false,
       "latitude":48.234159,
       "licensePlate":"W-26990D",
       "longitude":16.45867,
       "make":"BMW",
       "modelIdentifier":"bmw_1er",
       "modelName":"BMW 1Series",
       "name":"Arnold",
       "parkingSpaceId":null,
       "rentalPrice":{
         "drivePrice":{"amount":"34",
            "currencyUnit":"ct\/min"},
         "offerDrivePrice":{ "amount":"34",
            "currencyUnit":"ct\/min"},
         "parkPrice":{ "amount":"19",
            "currencyUnit":"ct\/min"},
         "paidReservationPrice":{ "amount":"10",
            "currencyUnit":"ct\/min"},
         "isOfferDrivePriceActive":false
       },
       "routingModelName":"bmw-1er",
       "series":"1er",
       "transmission":"M",
       "variant":""
     },{ ... }]
}
```

Listing 3.2: Raw vacant vehicle data from DriveNow [10] API, car item level.

sion type. There are a few elements necessary for visualization reasons. *carImageBaseUrl* and *carImageUrl* are technical properties for the app and website containing the URLs to certain images of the concrete car to display when a user looks up its detail information.

There is also a JSON object containing pricing information about the given car. Its key is *rentalPrice* and it contains five specific objects. The *drivePrice* element indicates the price that will be charged to a customer when he or she actively drives the car while *parkPrice* will be charged if the vehicle is parked during a trip session without dropping the car off to make it available to other customers. There are certain situations where an operator might want to set incentives for customers to pick certain cars up and drive them to different locations (i.e. user-based relocation strategies). For these scenarios, the operator can set the *offerDrivePrice* property to a certain offer price and the *isOfferDrivePriceActive* property to true. Finally, it is possible to reserve cars as a user. Usually, the first 15 minutes of the reservation are free of charge while after this amount of time, a reservation fee will be charged per minute. This rate is set in the *paidReservationPrice* value.

While the former objects of the entries of the *cars* JSON array had relatively static character and described the general characteristics of a car sharing system vehicle, there is further information on the dynamic status of the car in the system. The following name/value pairs give information about the fuel status of the car. *fuelLevel* and *fuelLevelInPercent* show the current filling status of the vehicle tank in case of a fuel operated car. In case of an electric vehicle the properties show the loading status of the battery. The *innerCleanliness* key-value pair contains a status of the car about the cleanliness of its interior. This information is gathered by asking a user on the car's display before every trip.

Finally and most important for the current thesis, the next set of properties holds spacial information about the current location of the vacant car. The *address* item contains the current address of the vehicle and consists of multiple address line entries while *longitude* and *latitude* combine for its current GPS position. Often, operating areas of car sharing systems do no only consist of general zones where cars can be picked up and returned in. They also contain parking lots, e.g. a car park near an airport, where car sharing vehicles can be found. For these scenarios, the properties *isInParkingSpace* ('true' or 'false') and *parkingSpaceId* hold concrete information on where the car is parked.

The responses of other car sharing operator's web service APIs might have a somewhat different structure but contain very similar information about the cars in their systems. The most important content of the response is the list of vacant cars and their concrete geographical locations. A crucial property of the API's responses that is also a precondition for the approach of the current thesis is, that it only contains cars that are currently vacant and available for users. This aspect is important because trips have to be deducted from the available information. For more information see chapter 4.

3.2 OpenStreetMap

OpenStreetMap [21] is a collaborative community project offering worldwide geographical information on an open data basis. Users can contribute to the project by adding mapping information as well as properties of certain points or areas of interest. The latter is referred to as tagging an provides metadata about the map. OpenStreetMap's data is stored in a PostgreSQL database with PostGIS extension and is available via a number of technologies like web browsers (e.g. official web site [21]), apps (e.g. Maps.me [17]) or RESTful API [22].

The data structure offered by OpenStreetMap is divided into four elements being the basic components of their conceptual data model of the physical world. A *node* represents a point in space identified by (at least) an ID and a geographical position on earth. The latter is a GPS coordinate consisting of a longitude and an latitude. Examples for real-world objects abstracted by a node would be a tree, a public transportation station or a current position of a car in a free-floating car sharing system. *Ways* are ordered lists of nodes representing linear features such as boarders of certain areas like a park or a hospital. Ways are also used to represent open polylines like streets or rivers. *Relations* are the data structure to describe relations between different elements, e.g. that a given node is part of a given way. Finally, *tags* give elements meanings. They are key-value pairs attached to the elements that enrich them with metadata. There is no fixed dictionary of tag keys, but certain tags are interpreted by software like the map on the OpenStreetMap web site. Examples for tags could be the concrete type of a node. To show that a node is a restaurant, a collaborator could tag it with the key 'amenity' and value 'restaurant'.

In this thesis, the OpenStreetMap data is used in combination with QGIS [28], an open source geographic information system tool. In QGIS, OpenStreetMap data can easily be displayed, imported, modified and analyzed.

3.3 Weather

Weather data is provided by the Department of Building Physics and Building Ecology at the Vienna UT [9]. They operate a weather station on top of the Vienna UT main building and provided and Excel file containing an evaluation of temperature and rainfall for every five minutes over a requested period. Table 3.1 shows the structure of the received data set.

date	**time**	**air temperature**	**relative humidity [%]**	**rainfall [mm]**
14.06.2015	17:45:00	21,2	76,3	0,1
14.06.2015	17:50:00	20,8	77,8	2,3

Table 3.1: Structure of weather data with entry examples.

The relevant data was received as a Microsoft Excel sheet containing data from June 1st, 2015 to August 10th, 2015. It lists 20020 lines of weather data for every five minutes for that period of time. The first column contains the date of the entry in format DD.MM.YYYY, the second one the time in format HH:MM:SS. The air temperature column holds values in degrees Celsius with one decimal place. The 4th entry of the lines is the relative humidity in percent and finally, column number 5 holds the rainfall entries in millimeters. One millimeter in this column equals one liter per square meter in the last 5 minutes.

CHAPTER 4

Suggested Solution

4.1 Generic method

The method proposed in this thesis is a generic approach to knowledge discovery based on trip data from free-floating car sharing systems. The hypotheses outlined in the introductory chapter represent examples of potential scenarios on which the method can be applied. There are many possible implementations of this approach, and it could lead to useful findings regardless of the underlying dataset. In this section, the generic method is described without reference to particular scenarios or hypotheses. A generic approach is the wide application of the conceptual and empirical procedures beyond a specific case study. Thus, the method outlined below may be generalized and implemented based on stakeholder needs. One goal of the thesis is to present a method that car sharing operators and other stakeholders can implement on individual datasets.

The method proposed in this thesis can be divided into two parts. The first is the data collection and preprocessing, the second represents the evaluation and interpretation of the dataset. While the first part is necessary for anyone who may not have direct access to trip data from car sharing operators, the operators themselves could potentially skip the data collection section and start directly with the evaluation.

The data for further evaluation of system balance in car sharing networks may come from multiple sources. While the extraction of vacant car snapshots outlined in detail later in this chapter necessitates a number of normalization steps, preprocessing, and various assumptions, the direct access to databases containing operator's historical trip data minimizes the amount of work to do before starting evaluations. In both cases, the exclusion of certain parts of the data might be necessary. Duplicates, as well as maintenance and relocation trips, would bias the result of the succeeding evaluations and should therefore be eliminated from the trip dataset.

Once a reliable dataset is found, it is recommended to do some exploratory analysis and descriptive evaluations. This brings the benefit of general, deeper insight into the trip behavior, and further more, leads to well-informed hypotheses and research assumptions. It is advisable to analyze the trip data with respect to different influence factors (in the example of this thesis weather and time of day), and to compare the structure of the resulting datasets. This task was done by the visualization of different database selections in this work. Another way to get good general insight into the data is the geographical visualization of specific spatial data points contained in the dataset. Using a geographic information system tool like QGIS, it is easy to present and filter a geo-spatial dataset and thus inspect its general structure. By first performing a cluster analysis on the dataset, the global hot spots of an operating area under examination can be identified.

The heart of the proposed method is the discovery of correlations between points of interest (represented by geographical locations) and location clusters based on filtered trip data. Points of interest relevant for a certain presumption one wants to investigate must be predefined and their geographical position has to be visualized using the chosen geographical information system tool. In this thesis, the geo points were extracted from the OpenStreetMap database. Once the point of interest locations are defined and setup, the empirical trip data should be clustered, and the produced hot spots should be visualized in combination with the points of interest. The proposed approach recommends the choice of a density-based clustering algorithm like DBSCAN. This algorithm produces clusters based on the distance between data points and their minimum number in a cluster, while excluding noise points from the result. This meets the requirements of the study's approach as a geographical accumulation of car sharing pickups and drop-offs shall be investigated. The choice of the input parameters for the algorithm has a high influence on the results. Their values shall be set based on experimenting, sorted k-dist graphs and, most important, domain knowledge.

To discover concrete knowledge of certain scenarios in car sharing systems, only a subset of the available data should be used in the clustering process. This subset should be produced by filtering the data by influence factors like weather or time of day, which are chosen according to the current hypothesis under examination. It is also important to define whether the pickup or dropoff times of the locations are of interest for a given hypothesis.

By a visual representation of the points of interest, in combination with the results from the clustering process, the interpretation of correlations between them can be recognized quite easily by observers. If clusters appear around a majority of the points of interest, the hypothesis under examination can be confirmed, or otherwise rejected. The following list sums the steps of the proposed method and shows the actions necessary when a given assumption should be evaluated:

1. Collect empirical trip data
2. Normalize trip data (exclude duplicates, exclude maintenance trips)
3. Define relevant points of interest
4. Define relevant trip location sub-dataset
5. Set up cluster analysis on trip location sub-dataset
6. Experiment with input parameters of DBSCAN until a meaningful number of clusters is found
7. Visualize point of interest and cluster data in a GIS tool

8. Confirm or reject hypothesis based on the distance between clusters and points of interest

By applying the outlined method, stakeholders can effectively evaluate assumptions, e.g. the influence of events in a city on the user behavior of free-floating car sharing systems. Furthermore, these findings can be used to improve relocation strategies of the system. While the proposed method can be applied to a variety of assumptions, subsequent chapters illustrate its application on a stated set of hypotheses. The dataset employed herein is taken from two car sharing operators' web sites.

4.2 Java prototype

The first step of the technical work done in this thesis was the development of a Java prototype B that collects data from different car sharing operators' web sites over a defined period of time and transforms the data into evaluable form. To do so, the requirement was split up into smaller problems:

1. Every 5 minutes, call operators' APIs and save the responses (the list of vacant vehicles and their status) into snapshot files.
2. To reduce disk space requirements, archive snapshot files in zip files.
3. Sort snapshot files in timely order and extract every contained car's information.
4. Depending on the current state of a car in the database and the current status in a snapshot file, deduce trip actions.
5. Add or modify the status of every car in a PostgreSQL database according to it's deduced trip actions.

Step 1: Generating vacant car snapshots - CarsharingDataCollector

The *CarsharingDataCollector* connects to the web service APIs of two car sharing operators and reads their data. To do so, a *HttpURLConnection* object is instantiated using the exact URL of the operators API. One of the two APIs used in this thesis needed additional request headers for the get request to be called. This property could be extracted by inspecting the operators homepage and looking at the headers that were attached to the API call by the web site. The get request to the second operator was successful without any further information than the URL. After reading the response (containing the list of currently vacant cars) to the end, the response text was written into a file on the local file system. As the contents of the file, so the web service responses, do not contain any information about the time, the call was performed at, the current time stamp was appended to the local file's name. Since the JSON structure of the responses (the contents of the file) vary between the different operators, the name of the operator is also used as a part of the filename. This makes it easier to work with the data in succeeding steps. The exact and commented code of this step can be found in the appendix B.

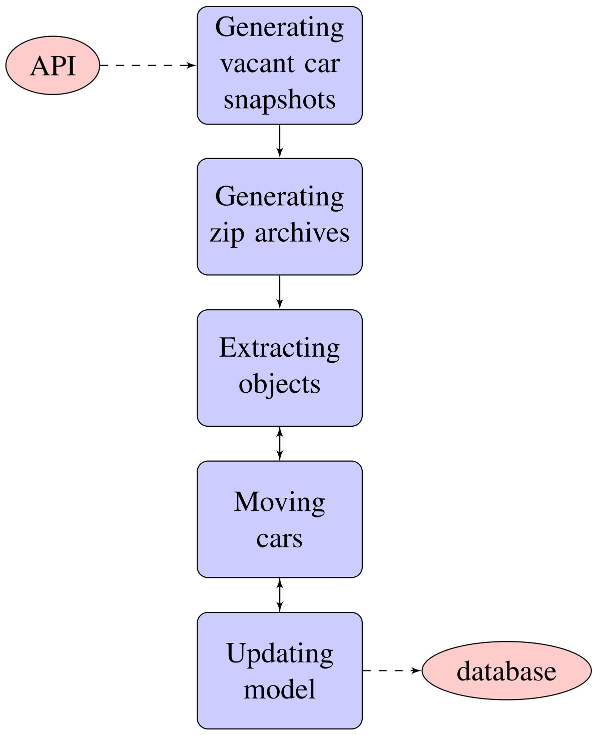


Figure 4.1: Flow chart of prototype actions.

The described actions were performed every five minutes for the period between June 18th, 2015 at 13:58 o'clock and July 22nd, 2015 at 14:12 o'clock. In the beginning of the time period, the interval between saving snapshots was even shorter, precisely 2 minutes. After analyzing the first results, it was assumed that car sharing trips that take less than 5 minutes are usually canceled reservations as the concerned cars did not change their location after the supposed dropoff. So the interval was finally set to 5 minutes, which did not significantly change the results of later steps performed with the concerned data.

Step 2: Generating zip archives

Step 1 resulted in a set of 19736 snapshot files that were zipped due to their high disk space usage. Following steps take the resulting zip files as preconditions for their actions.

Step 3: Extracting objects - CarsharingDataDecoder

The *CarsharingDataDecoder* represents the routine of transforming the collected snapshot files containing lists of vacant cars from car sharing operators to Java objects with data of interest. As an input, it takes the zip files that were generated in Step 2, which have to be copied to the *./Data* directory of the Java prototype for further execution. The routine sorts and opens every archive and, within them, every contained snapshot file in the order of the time stamp, the snapshot was taken at. As step one generated the files with the operator name as well as the snapshot time stamp, this information can be extracted from the file under investigation right away. As JSON structures vary between the different operators, the routine calls different classes to extract the information of interest from the file contents. The two classes are called *OperatorADecoder* and *OperatorBDecoder* and their code, along with the one from *CarsharingDataDecoder* itself, can be found in the appendix B. While the structure is still different between the operators JSON of the vacant car list, both of the decoder classes return a list of *Car* objects with the data structure outlined in Figure 4.2.

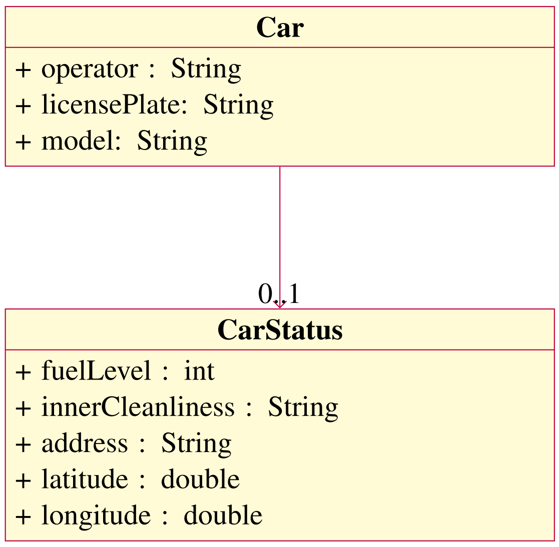


Figure 4.2: Data structure of decoded Java objects.

The *Car* and *CarStatus* classes encapsulate a subset of the information available in both car sharing operators' web service responses that is also relevant for the following work. *Car* holds relevant values of a vehicle in the system, that do not change between trips, one could declare them as static properties. *CarStatus* holds values that change between trips, so the ones being dynamic during the life cycle of a vehicle. The *Car* class holds three *String* attributes. The *operator* attribute is initialized the name of the car sharing operator, the represented vehicle is owned by. *licensePlate* holds the license plate value of the car and *model* is set to the exact model name of the current car. Furthermore, *Car* holds a reference to a *CarStatus* object representing the current dynamic properties of the car. These properties are abstracted by an *Integer fuelLevel* that contains the current fuel status of the vehicle on a range from 0 to 100 and the *innerCleanliness* value set with a variable *String* value describing the current cleanliness of the car. Next and most important for this thesis are three attributes abstracting the current position of the car. The *address String* is filled with the current address, the vehicle is parked at while *latitude* and *longitude* are two doubles abstracting the current GPS position of the car. The Java code for the classes can be found in the appendix B.

A list of combinations of *Car* and *CarStatus* can be deducted from every vacant car snapshot file processed in this step. One unique car can only appear once in a snapshot file, so the Car objects are also unique within the produced list of *Car-CarStatus* combinations with respect to their license plate, which will further be taken as the single unique identifier for *Car* objects. Neither *Car* nor *CarStatus* contain any timely information about the system as they are simply decoded entities extracted from different snapshot files. The timely aspect will be added in the next step.

Step 4: Moving cars - CarsharingDataDecoder

This next step is processed after every extraction of a list of *Car/CarStatus* objects from one single snapshot file and takes a list of such combinations as input. The goal of the step is to produce a history of car sharing trips for every single vehicle in the system by deducting those trips from the appearance and disappearance of cars in the vacant vehicle lists. The basic thought behind the performed actions is that *pickup* actions occur when cars vanish from vacant car lists and *dropoff* actions occur when they reappear. A pickup action happens at the start of a trip, i.e when a user starts a car sharing rental session. The time stamp of a pickup action is not always the exact moment, when a user starts a trip as the car already vanishes from the vacant car list when it is reserved by a user. A dropoff action is the end of a trip, so when a user returns the vehicle into the system to be available for other users. In the time between a dropoff and a pickup, a vehicle is parked and available for customers.

For every *Car/CarStatus* object of one single snapshot file (that were sorted in timely order before), one of the following scenarios will hold. Once the first fitting scenario is found for a car, the succeeding scenarios will not be checked or performed. For every action performed in the database, the time stamp of the current snapshot will be taken as the moment of action.

1. The current car is not yet known in the database. This represents the first appearance of the car in the system. The car will be created in the database and a dropoff action will be performed.

2. The current car is not parked, so currently on a trip in the database. As the car now appears in the vacant cars list a dropoff action will be performed.

3. The current car is parked in the database and it's location is different to the one in the vacant car list. In this case, the car was on a trip that was shorter than the interval time of taking snapshots. A pickup as well as a dropoff action will be performed on the database.

4. Finally, all the parked cars in the database have to be checked in case some of them went on trips. So, if a car is parked in the database and is not contained in the current list of vacant cars, a pickup action will be performed.

After performing these steps for every vacant car entry in every taken snapshot file in a timely order, a complete history of trips for every car will be available in the database.

Step 5: Updating model - CarsharingDbConnection

The different actions performed in the preceding step have to change the car history data in the database in the right manner. The different actions will be outlined here to get a better insight

```sql
insert into car (
  licenseplate,
  operator,
  model
)
values (?, ?, ?)
```

Listing 4.1: SQL for create new car action

in what technical actions have to be performed to cover the requirements described above. The underlying database model is explained in the Data Model Section 4.2. The exact code of the *CarsharingDbConnection* can be found in the appendix B.

Create: Create new car If a car appears in a vacant car list and is not contained in the car sharing database yet, this action creates it. The static properties of a car are thereby stored in the *car* table. Listing 4.1 shows the SQL for the action.

Create: Dropoff If a user drops off a given car, so ends his trip at a certain position, this action is to be called. It creates a new entry in the *carhistory* table with all the dynamic properties of the car at the moment of the dropoff. The *arrival* column is set to the time stamp of the current vacant car snapshot. The *departure* column is set to null and will be filled during a succeeding pickup action. In this action, the longitude and latitude values coming from the car sharing operators' APIs are transformed into PostgreSQL/PostGIS data types. This is done by the PostGIS functions *ST_SetSRID* and *ST_MakePoint*. The Spacial Reference ID (SRID) 4326 is the identifier for the coordinate system World Geodetic System 1984 (WGS 84), which is also used by OpenStreetMap or Google Maps. Listing 4.2 shows the SQL for the action.

Update: Pickup When a pickup action, so the start of a trip is recorded by the preceding business logic, the last *carhistory* entry in the database has to be updated. More concrete, only the departure column of the latest entry, which should be null in case of correct input data, will be set to the time stamp of the current vacant car snapshot. The rest of the columns of the entry stay the same because it is supposed that a car cannot be moved between a dropoff and a pickup action. This means that no properties of the car can be subject to change in this period of time. Listing 4.3 shows the SQL for the action.

Read: Check car existence The business logic has the requirement to check, if a car exists in the database in certain situations. This can technically be done by simply checking the existence of the car of interest's license plate in the *Car* table. Listing 4.4 shows the SQL for the action. If the count returns more than 0, the car exists. It does not otherwise.

Read: Check if car is parked Another requirement by the business logic is the check for the trip status of a given car. If a car is parked, it is not on an active trip. If it is not in parked status,

```sql
insert into carhistory (
  licenseplate,
  fuellevel,
  innercleanliness,
  address,
  arrival,
  departure,
  location
)
values (?,?,?,?,?,?,
  ST_SetSRID(
    ST_MakePoint(?,?),
      4326)
)
```

Listing 4.2: SQL for dropoff action

```sql
update carhistory
   set departure=?
   where licenseplate=?
      and arrival >= all (
         select arrival
            from carhistory
            where licenseplate=?
         )
```

Listing 4.3: SQL for pickup action

```sql
select count(*)
   from car
   where licenseplate=?
```

Listing 4.4: SQL for check car existence action

```sql
select count(*)
  from carhistory
  where licenseplate=?
  and departure is null
  and arrival >= all (
    select arrival
      from carhistory
      where licenseplate=?
    )
```
Listing 4.5: SQL for check if car is parked action

```sql
select address
  from carhistory
  where licenseplate=?
  and arrival >= all (
  select arrival
    from carhistory
    where licenseplate=?
  )
```
Listing 4.6: SQL for check if car has changed location action

a rental session (trip) is active for the car at the moment of the action call. Technically, the car is in parked status if it recently had a dropoff action called and no pickup action after that. The departure column of the *carhistory* entry would be *null* in that case. So the solution is to simply check for the *null* value in the latest *carhistory* entry of the concerned car. Listing 4.5 shows the SQL for the action. If the count returns more than 0, the car is parked. It is not otherwise.

Read: Check if car changed location To see if a car changed its location compared to a *Car* object coming from the list of vacant cars, the address of the latest *carhistory* entry in the database is compared to the one in the passed *Car* object. Listing 4.6 shows the SQL for the action.

Read: Determine all parked cars in the database A list of all parked cars is necessary for the business logic to check, if a pickup action is necessary due to the disappearance of a car from the vacant car list. For performance reasons, the query also filters the tuples by the car sharing operator of interest. Again, the indicator for a car to be parked is the null value in the departure column of the *carhistory* tuple. Listing 4.7 shows the SQL for the action.

```sql
select ch.licenseplate
  from carhistory ch, car c
  where ch.departure is null
  and c.licenseplate=ch.licenseplate
  and c.operator=? and
  ch.arrival >= all (
    select ch2.arrival
      from carhistory ch2
      where ch2.licenseplate=ch.licenseplate
  )
```

Listing 4.7: SQL for determine all parked cars in the database

Data Model

All the actions taken by the Java prototype presented before lead to a database filled with car sharing trip history. This database is described in the following section.

Figure 4.3 shows an entity relationship diagram of the developed and loaded database. On database level, the static and dynamic attributes that were already outlined in this section were implemented in a similar way. The *licensePlate* attribute is the primary key for the *Car* table while it also serves as a foreign key for *CarHistory*. *arrival* and *departure* are of data type *timestamp* and *location* has the PostGIS data type *geometry(Point, 4326)*. The second parameter of the data type is, just like described before, the Spacial Reference ID (SRID) 4326. It is the identifier for the coordinate system World Geodetic System 1984 (WGS 84). *fuelLevel* is a *smallint* and all other attributes are of type *varchar*.

The weather data was manually imported into the table *Weather*. The first two rows of the excel with the raw data were joined for a *timestamp* column *time*. The rest of the columns as well as the row structure were semantically kept the same way as described in section 3.3.

For performance reasons, a few indexes were generated on the columns *carhistory(licenseplate)*, *carhistory(arrival)*, *carhistory(departure)* and *weather(time)*. Furthermore, the materialized view *carsharing_data* was created after fully loading the database. It serves as a single point of entry to the database for the tools that will be worked with later on in this thesis (e.g. QGIS). The few is materialized due to the high volume of data that it is based on. Queries that use *carsharing_data* have perform pretty well and have access to all necessary data as the view combines the columns available in both the *Car* and the *CarHistory* table.

The complete create scripts for the data model as well as the materialized view are attached in the appendix A.

4.3 Data preprocessing and normalization

The actions performed by the Java prototype that were described in the former section led to a PostgreSQL database with the following quantity structure. After the initial load of the database:

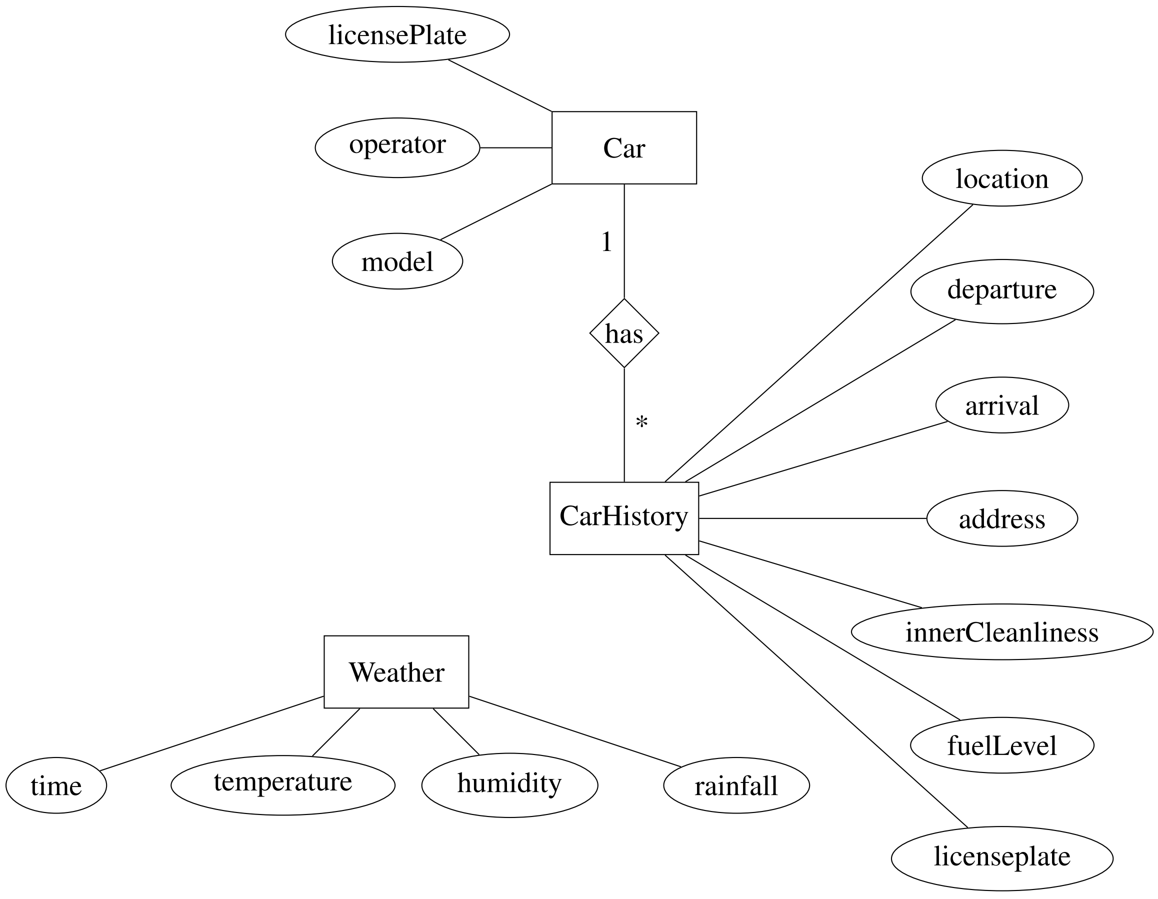


Figure 4.3: Entity relationship diagram of car sharing database.

- information from **19736 vacant car snapshots** coming from two car sharing operator APIs were processed.
- A number of **1514 unique cars** were added to the database (table *Car*),
- summing up for **206346 trips**. A trip is equal to a *CarHistory* tuple representing a combination of dropoff and pickup actions (while pickup column can be null at this point).
- The execution of the part of the Java prototype from *Extracting objects* to *Updating model* over all zipped snapshots files took **5 hours and 31 minutes** on the author's laptop.

As a next step, before doing any evaluation with the created data, a number of plausibility checks and data normalization were performed on the data set. The goal of this step is to preprocess the data in order to minimize noise in the data set, as well as be able to perform reliable analysis and evaluation in the following steps.

```
select id.*
  from ( select operator,
      lag(timestamp_file) over (
      partition by operator order by created
      ) as tf_prev,
      timestamp_file as tf,
      created
    from importeddata
    order by operator, created) as id
  where tf <= tf_prev
```

Listing 4.8: SQL for verification of correct data processing order.

```
select *
 from carhistory
 where departure <= arrival
```

Listing 4.9: SQL for verification of plausible dropoff and pickup times.

Plausibility checks

First, a number of SQL statements were executed in order to find data that is not plausible with respect to the nature of real world car sharing characteristics. One of them is to review if all the processed snapshot files were in the right order. This is a crucial condition as processing them in an unnatural timely order in the Java prototype would lead to wrong trip data in the database. For this reason, the SQL statement outlined in Listing 4.8 was executed and it was confirmed that it returns zero rows. The used table of this query is *importeddata*. The table has logging character as during the execution of the Java prototype, the operator and the snapshot time stamp of every processed snapshot file, as well as the time stamp of the tuple creation is inserted in it. The create statement of the *importeddata* table can be found in the appendix A. Basically, the statement sorts all logging entries by tuple creation, and checks via a window function if any time stamp of a tuple is older than its predecessor.

The next data plausibility check is a technically very simple but semantically very important one. If there are tuples in the *CarHistory* table that have a departure time stamp that is before the arrival time, there must be a mistake in the processing of the snapshots. The query outlined in Listing 4.9 must not return any results in order for the data to be plausible. While this query only checks the validity of the data within tuples, it might be interesting to check the trip sequence over all the tuples of the trip data of single cars. Just like the previously processed sequence check, Listing 4.10 checks if there are any arrival entries in *CarHistory* that have a time stamp before the departure time stamp of the preceding *CarHistory* tuple of the same car. This would semantically mean that a car traveled back in time. The concerned query must not return any rows as well.

```sql
select ch.*
  from ( select licenseplate,
      address,
      lag(departure) over (
        partition by licenseplate order by created
      ) as dep,
      arrival,
      departure,
      created
    from carhistory
    order by licenseplate, created ) as ch
  where dep > arrival
```

Listing 4.10: SQL for verification of plausible dropoff and pickup times.

Data normalization

During the execution of the *Generating vacant car snapshots* step of the Java prototype, there were times when the continuous collection of snapshots was stopped due to e.g. maintenance on the author's laptop. While snapshots were usually taken every 5 minutes, causing a relatively precise trip history, a prototype downtime of more than 30 minutes could potentially bias the trip data. This would have the effect of a trip history for certain cars that has arrival and departure times that are more than 30 minutes before, or after the actual pickup or dropoff. As such errors could affect the evaluations, the arrival times after such downtimes, and the departure times before them, were set to null so they have no influence in further examinations. The Listings 4.11 and 4.12 show the SQL statements used for this task. Both of the queries use window functions to calculate the difference in the time stamps of the processed snapshot files and set time stamps around downtimes, that are greater than 30 minutes, to null.

Car sharing operators that the concerned data is collected from, offer the possibility of reserving a car before starting the actual rental session. This can prevent situations where users walk towards the next vacant car, and find that the car was taken by a different user in the meantime. Usually, operators offer a free reservation for a defined period of time for 15 or 30 minutes. On top of that, users can make reservations for a longer period of time if they want. This is usually offered for a special rate per minute. From a technical perspective, the car vanishes from the list of vacant vehicles shown on the homepage or app as soon as it is reserved because other users should not be able to make a reservation or start a rental session with the same car. Users that made a reservation for a car have no obligation to start a rental session afterwards. They can cancel a reservation or just leave it until the maximum time for a free reservation is exceeded. In both cases, the car will reappear in the list of vacant vehicles for other users but it's properties (location, fuel level, etc.) will still be the same as before the preceding reservation was made. By the way, the Java prototype was designed, such canceled reservations result in a *CarHistory* tuple with the same values of *fuelLevel*, *innerCleanliness*, *address* and *location* as the preceding tuple in the database. Still, reservation would be recognized as trips as the concerned car

```sql
update carhistory
  set arrival = null
  where arrival in (
    select timestamp_file
      from ( select operator,
          timestamp_file,
          lag(timestamp_file) over (
              partition by operator order by created
             ) as prev_timestamp_file,
          timestamp_file - lag(timestamp_file) over (
              partition by operator order by created
             ) as diff,
          created
      from importeddata
      order by operator, created) as id
      where extract(minute from diff) > 30
  )
```

Listing 4.11: SQL for deleting arrival times after downtimes in data collection.

```sql
update carhistory
  set departure = null
  where departure in (
    select prev_timestamp_file
      from ( select operator,
          timestamp_file,
          lag(timestamp_file) over (
              partition by operator order by created
            ) as prev_timestamp_file,
          timestamp_file - lag(timestamp_file) over (
              partition by operator order by created
             ) as diff,
          created
        from importeddata
        order by operator, created) as id
        where extract(minute from diff) > 30
  )
```

Listing 4.12: SQL for deleting departure times before downtimes in data collection.

vanished and reappeared in the list of free cars and the arrival and departure columns of the new tuple will be written for the reservation. As such tuples do not reflect trips with respect to the goal of this thesis as they don't have actual pickup and dropoff actions, they were deleted from the database. The problem gets even more complex when thinking about the possibility, that multiple canceled reservations can be sequentially occur in the trip history of a car. In such a sequence, may it contain only two canceled reservations or more, only the arrival time stamp of the first tuple and the departure of the last tuple in the sequence are of interest for the current thesis. Table 4.1 shows an abstracted example for such a sequence and highlights the times of interest.

licensePlate	fuelLevel	location	arrival	departure
W-12345A	90	Beispielgasse 2	1.6.2017 09:12	2.6.2017 08:33
W-12345A	82	Musterweg 7	**2.6.2017 10:12**	2.6.2017 11:14
W-12345A	82	Musterweg 7	2.6.2017 11:24	3.6.2017 12:10
W-12345A	82	Musterweg 7	3.6.2017 13:30	**3.6.2017 15:22**
W-12345A	75	Exempelstrasse 9	3.6.2017 16:00	3.6.2017 19:20

Table 4.1: Example for a sequence of canceled reservations in the *CarHistory* table.

The solution for the problem is to set the value of the departure time stamp of the first tuple in such a sequence to the departure value of the last tuple. Listings 4.13 and 4.14 show the SQL statements for solving the described problem, the former updates the departure time of the second last tuple while the latter deletes the last tuple of the sequence. Table 4.2 shows the result of the example from Table 4.1 after performing the SQL. Note that the departure time stamp of the last sequence tuple can be null (if it was generated by a snapshot that was the last before downtime of more than 30 minutes or the last in general). The strategy in this case does not change, the null value will still be taken as the new departure value of the first tuple in the sequence. Also note that the code shown in Listings 4.13 and 4.14 only handles the last pair of tuples in a sequence. To eliminate all duplicates, the code snippets have to be executed multiple times until all duplicates of all cars are deleted. In order to check if duplicates still exist, the *update* statement in the snippet can easily be adapted to be a select statement.

licensePlate	fuelLevel	location	arrival	departure
W-12345A	90	Beispielgasse 2	1.6.2017 09:12	2.6.2017 08:33
W-12345A	82	Musterweg 7	**2.6.2017 10:12**	**3.6.2017 15:22**
W-12345A	75	Exempelstrasse 9	3.6.2017 16:00	3.6.2017 19:20

Table 4.2: Example for a normalized sequence of canceled reservations in the *CarHistory* table.

Operator-based relocation trips

Car sharing operators in Vienna were asked for operational data but unfortunately they did not want to cooperate. This obviously brings up the problem that the evaluation data is incomplete

```sql
update carhistory
  set departure=sq.departure
  from (select *
    from (
      select licenseplate,
          lag(location) over (partition by licenseplate order
             by created) as prev_location,
          location,
          lead(location) over (partition by licenseplate
             order by created) as next_location,
          departure,
          lag(created) over (partition by licenseplate order
             by created) as prev_created,
          created
        from carhistory
        order by licenseplate, created
      ) as ch
    where prev_location = location
    and (
      (not next_location = location)
      or (next_location is null)
      ) -- last tupel in a sequence or very last tupel of car
    ) as sq
  where carhistory.licenseplate = sq.licenseplate
  and carhistory.created = sq.prev_created;
```

Listing 4.13: SQL for updating canceled reservation entries in *CarHistory* table.

due to the fact that it is not known which relocation strategies the operators use. As a result, assumptions have to be made about the characteristics of a trip in the car sharing system. Every trip could either be made by a customer, or could be a relocation trip initiated by the operator. The latter could, for example, be identified by having unusual routes, the improvement of the cleanliness of a car or the refilling of its gas. Different assumptions on the current strategies might of course lead to varying results in the evaluation. For this reason, it was examined which of the trips in the current dataset could be operator-based relocation trips, or in general, trips that were made in order to do car maintenance. Most of the free-floating car sharing operators offer incentives (like free driving time) to users, that fill up the fuel of cars that have a fuel level below a certain value (e.g. below 25%). Regardless of who exactly fills up the fuel for such cars, the reason for the resulting trip might be another one than those interesting for this thesis.

Due to the lack of complete data, the best possibility to find out which trips could be relocation trips is to see which tuples in the history of a car have a higher fuel level than the ones before them. This could be a good indicator for a maintenance trip that is not of primary interest

```sql
delete from carhistory del_ch
  where exists
   (select *
     from (
       select licenseplate,
           lag(location) over (partition by licenseplate order
              by created) as prev_location,
           location,
           lead(location) over (partition by licenseplate
              order by created) as next_location,
           lag(departure) over (partition by licenseplate
              order by created) as prev_departure,
           departure,
           lag(created) over (partition by licenseplate order
              by created) as prev_created,
           created
         from carhistory
         order by licenseplate, created
         ) as ch
     where ch.prev_location = ch.location
     -- last tupel in a sequence or very last tupel
     and (
       (not ch.next_location = ch.location)
       or (ch.next_location is null)
     )
     -- if it is the very last tupel, departure is set to null
     and COALESCE(ch.prev_departure,to_timestamp('01012000',
        'DDMMYYYY'))
       = COALESCE(ch.departure,to_timestamp('01012000',
          'DDMMYYYY'))
     and del_ch.licenseplate = ch.licenseplate
     and del_ch.created = ch.created );
```
Listing 4.14: SQL for deleting updated canceled reservation entries in *CarHistory* table.

```
select count(*) from (
  select licenseplate,
      lag(fuellevel) over (partition by licenseplate order by
         created) as prev_fuellevel,
      fuellevel
    from carhistory
    order by licenseplate, created
  ) as ch
  where fuellevel > prev_fuellevel+10
```
Listing 4.15: SQL for counting trips, during which gas was filled.

for the current thesis. Listing 4.15 shows a possibility to find out such trips via the *CarHistory* table. After looking further into the fetched data, it showed that sometimes the fuel level of a succeeding trip increases only by a small value. This seems to be a the result of a deviation in the measurement instruments of the cars rather than a refueling stop. That's why a minimum increase of 10% in the tank from one trip to the other was taken as the lower limit to declare a trip as maintenance.

After performing the outlined data normalization operations done previously, the *CarHistory* table contains a number of **170938 tuples**. The SQL in Listing 4.15 shows that only 3969 rows in the total data have the characteristic of a maintenance trip as described before. That's a share of **2,32 %**. This is quite a small value and it can't be found out for sure, whether those trips really are maintenance trips or just initiated by users, that take a refueling stop during a regular rental session. Further on, it will be assumed that this small value has no or only insignificant influence on the analysis and evaluation done in the thesis. The cooperation with car sharing operators to get a better dataset with clearly outlined relocation trips and exact time stamps would be interesting for future work.

4.4 Exploratory analysis

In this section, the prepared dataset will be analyzed with an exploratory approach to get better insight into it. It shall be shown what general influence different factors like day of week, time of day, or weather have on the collected empirical data. Furthermore, general insights into patterns and tendencies of the trip data shall be given and visualized.

Temporal influence

The first influence factor that will be examined and visualized is the day of week. Figure 4.4 illustrates the distribution of car sharing pickups over the different days of week. It shows that the number of pickups is relatively stable except for Friday, where pickup numbers are clearly higher and Sunday, where they are quite a bit lower compared to the other days. This pattern is similar when looking into the isolated data for each car sharing operator. The reasons for this pattern can only be assumed. The peak on Fridays could be explained by weekly commuters that leave town and use car sharing vehicles to get to train stations or to their private cars they leave on the outer parts of the city where they do not have to pay for parking them.

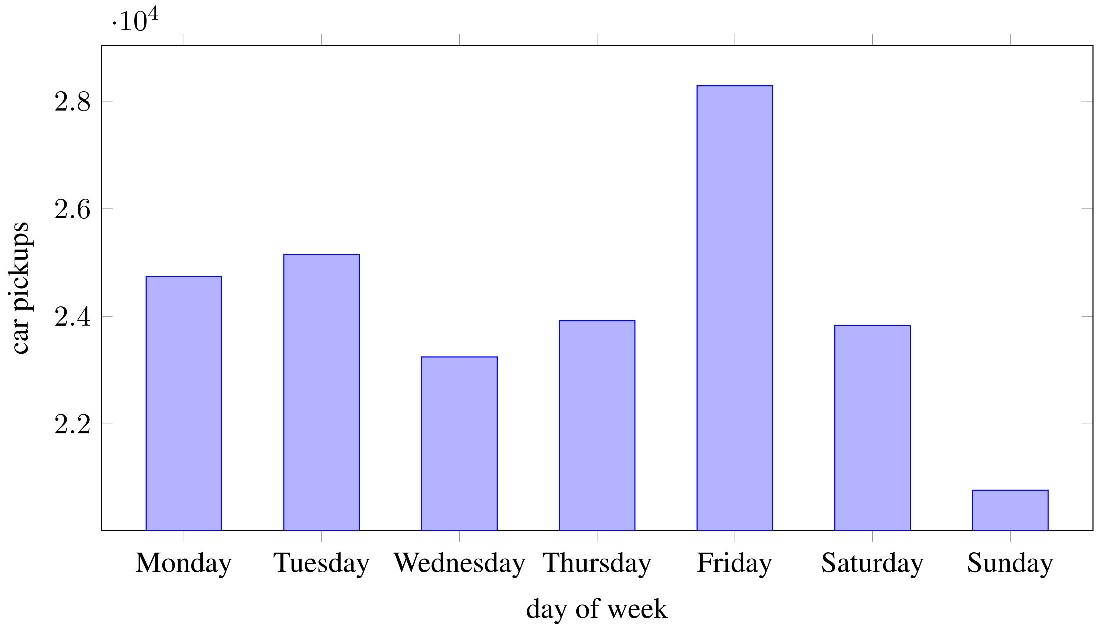


Figure 4.4: Car sharing departures by days of week.

The next figure 4.5 shows the distribution of car sharing pickups over the different fully recorded calendar weeks in the the period of investigation. It shows that calendar week 26 has a clearly higher pickup rate than the other outlined periods. This might have to do with the end of the summer term in calendar week 27. Students are target customers of car sharing operators and a share of them might leave town in July due to holiday time and off time at universities. In general, people might be on holiday in the summer months which could lead to the lower numbers in the calender weeks 27 to 29.

The distribution of pickups per hour of day 4.6 shows a clear peak of car sharing usage in the evening hours. The highest number of pickups is between 5 and 8 p.m. where, potentially, people travel home from work or to private or leisure activities. The combination of those two scenarios could lead to this high point in the data. From this point in time on, the pickup rate constantly decreases until 3 in the morning, where the general low is reached. From 4 a.m. on, the pickup rate increases again until reaching another peak, clearly visible around 8 a.m., where people probably drive to work or other spots of daily business in town. Both the 8 a.m. and the 6 p.m. peak represent the rush hour in the city.

While the described pattern seems pretty stable over weekdays, it is probably interesting to see, whether it changes on weekend days or not. Figure 4.7 shows the distribution of car sharing pickups per hour of day, filtered by day of week so only weekend days appear in the evaluation. As one could expect, the pattern is somehow similar. Still, the 8 o'clock peak does not show in this graph and the rush hour behavior is also not as concise. On the contrary, the graph is a

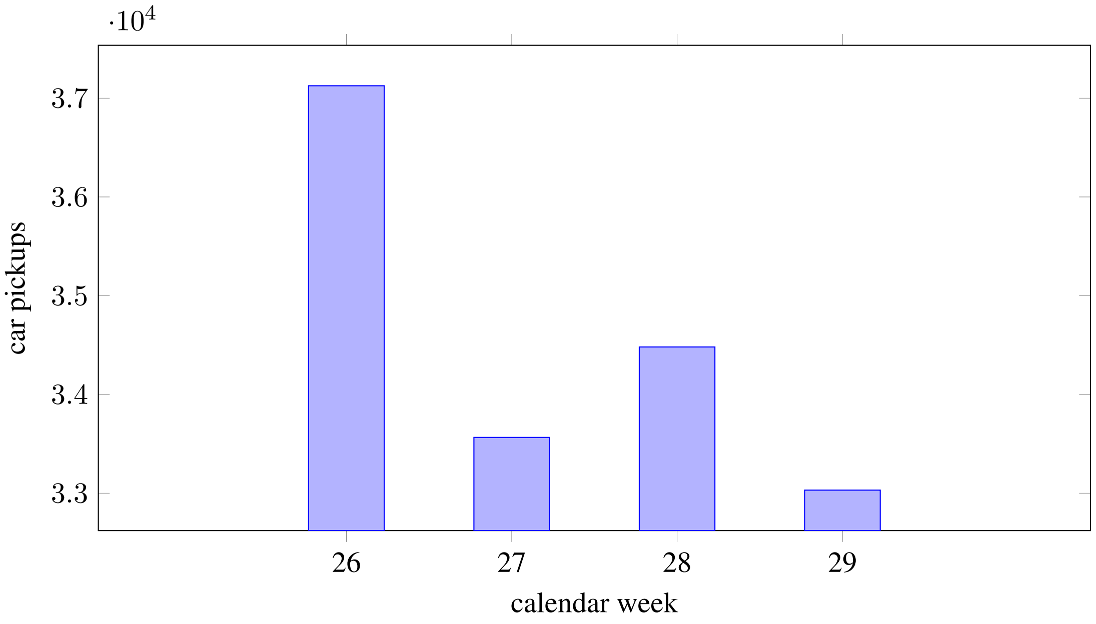


Figure 4.5: Car sharing departures by calendar weeks 2015.

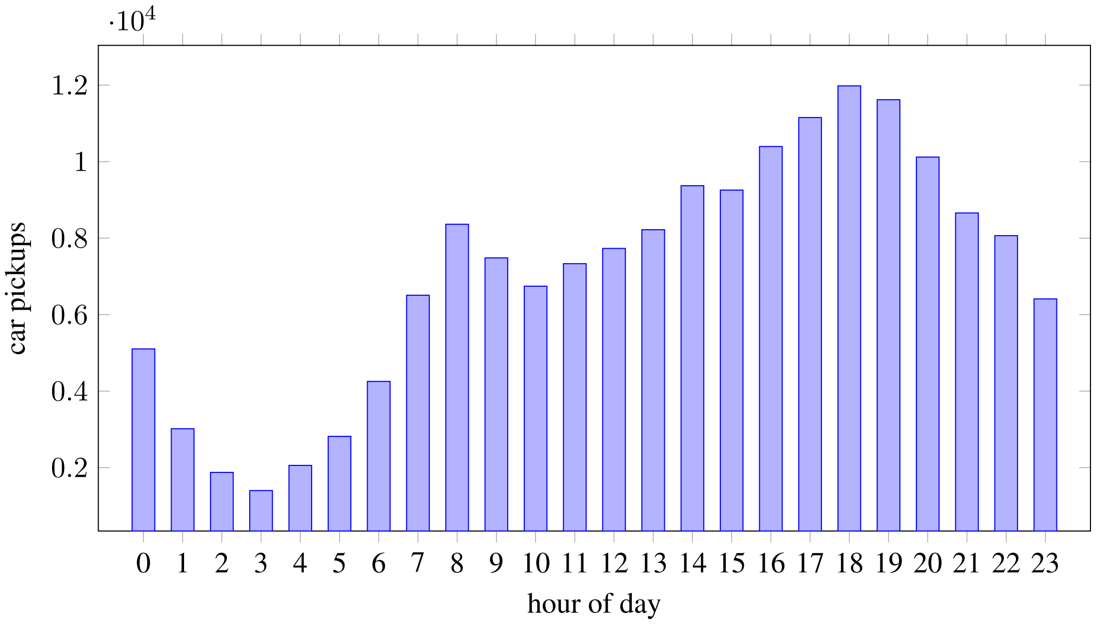


Figure 4.6: Car sharing departures per hour of day, total.

little more balanced throughout the day. This might be a result of the free time of people and the usage of cars for leisure activities that can be undertaken any time during the day. Of course, there is still a peek between 6 and 8 p.m. as evening activities often start by that time and people take cars to go to their location.

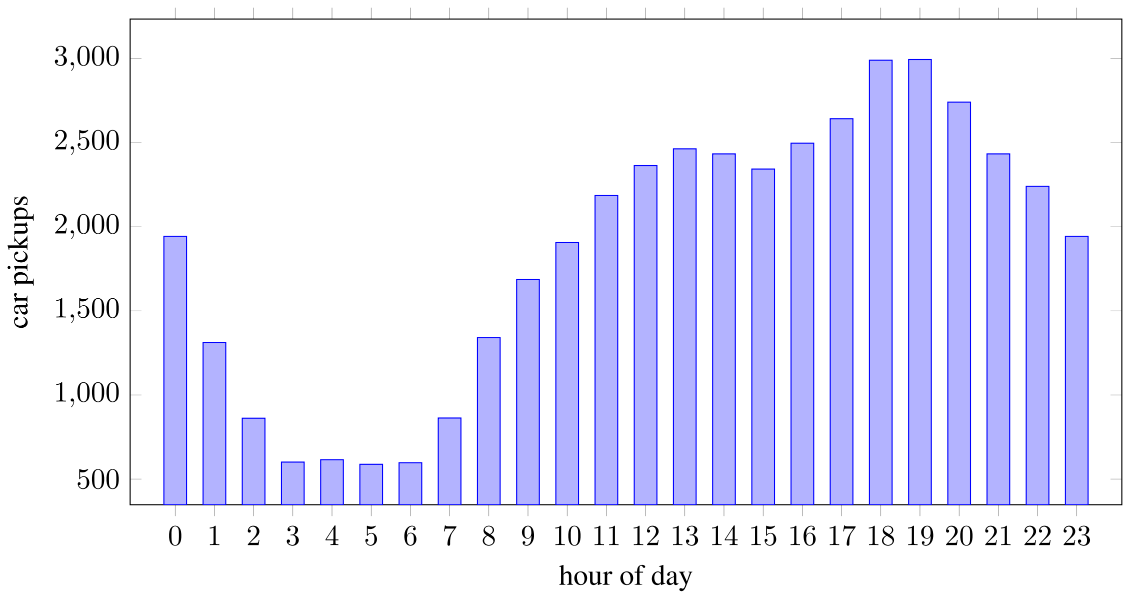


Figure 4.7: Car sharing departures per hour of day, weekends.

Weather data

The weather could play a major role in the usage of car sharing. To get general insight into the weather dataset, a few evaluations were performed. Due to the fact that the period in which weather data is available is longer than the one with empirical car sharing data, part of the weather data was filtered out of the data set. Only the data which comes from the period where both weather data and car sharing data is available should be taken for further evaluation. Figure 4.8 gives an overview of the maximum temperature per day in the dataset over the period of interest. While temperatures generally rise from mid of June to mid of July, the temperature peaks around 35 degrees Celsius. A few hot days are good to use for the evaluation of the hypothesis concerning public transportation. Figure 4.9 show the summed amount of rainfall per day over the same period of interest. It shows that there were four days with a larger amount of rainfall and a few more days with moderate rain. The rainy days are of special interest for the evaluation of the hypotheses, that take precipitation into account. The 8th of July shows as the day with the most rain and sums up to 13.7 liters per square meter over 24 hours. In general, the available data seems good enough to do the planned evaluations with as it offers a good variety of weather conditions over the period of investigation.

Visualization

To find patterns in the available geographical trip data, some sort of visualization for the prepared dataset stored in the PostgreSQL/PostGIS database is needed. As described before, QGIS [28]

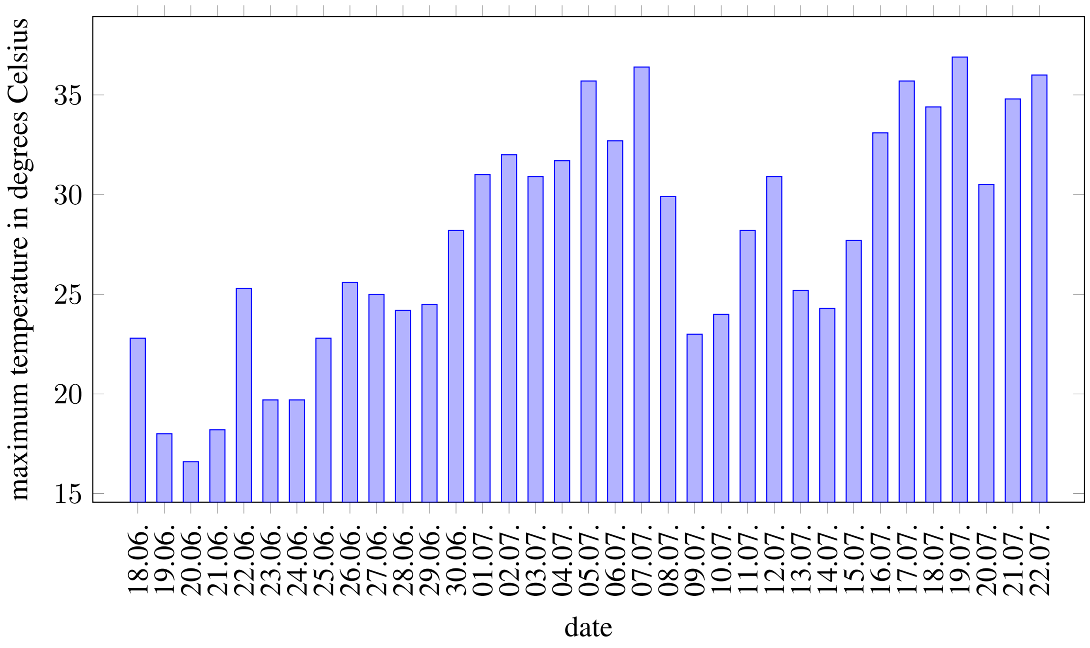


Figure 4.8: Maximum temperature per day.

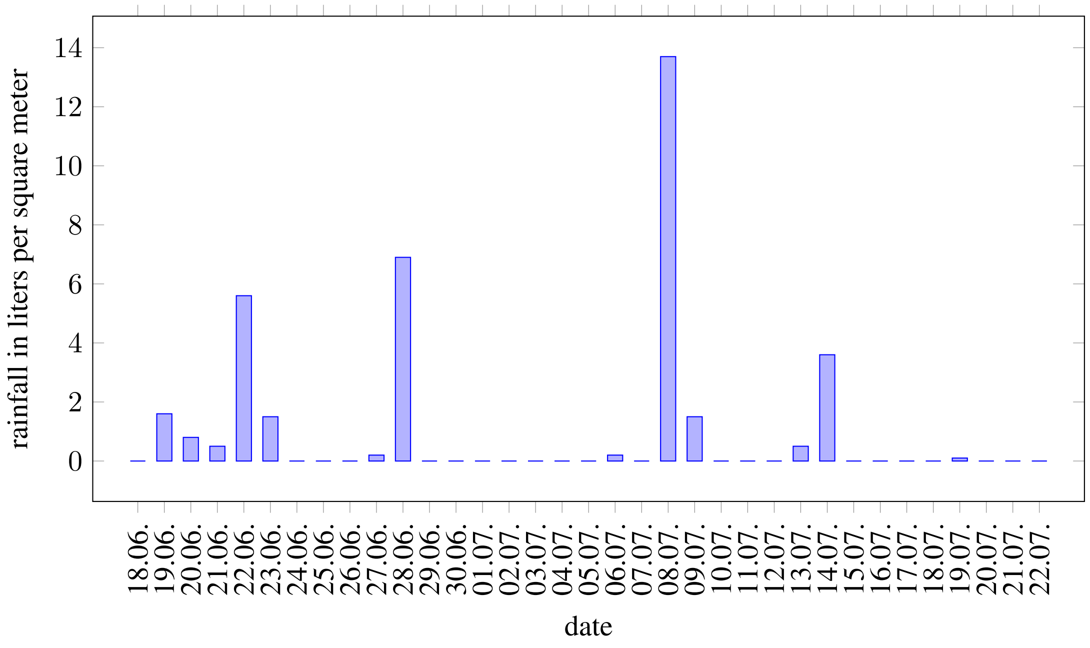


Figure 4.9: Rainfall per day.

offers a lot of functionality for such use cases and was therefore chosen as the visualization tool. The data presented in QGIS is organized in layers. To get a connection between the trip data and geographical locations in the city of Vienna, a base layer was added as the lowest one, presenting a map of Vienna, where further on, data points can be placed. There is a variety of possibilities for the source of this layer. OpenStreetMap as well as Google Maps are integrable via the OpenLayers plugin [29], which offers the data sources via web access. The base maps can therefore be scrolled, zoomed and inspected just like one would open them on their original URLs in a browser window.

QGIS furthermore integrates OpenStreetMaps in a way that its data can be downloaded and worked with locally on the user's host. Using this method, a map section of Vienna and it's surroundings was downloaded and certain elements of interest were extracted from the dataset. When exporting geographical elements from a downloaded OpenStreetMap topology, QGIS offers three types of elements: *Nodes*, *polylines* and *polygons*. Similar to the terms described in Section 3.2, a node represents a point in space, a polyline is a linear feature and is called *way* in OpenStreetMap notation. Finally, a polygon would be a closed polyline (or way) and represent areas on the map. Those three features can be extracted with their tags attached. Using the export of polylines and adding them as a layer to the QGIS project results in a base map in only one color that does not distract as much as a colored Google Maps or full OpenStreetMap base but still gives a good overlook of the city and it's locations. Further evaluations can be projected on top of this base map in color and will draw the observers attention on the important features on the map.

As a next step for visualization purposes, the spatial data stored in the database should be displayed in QGIS. For this purpose, QGIS offers the possibility to connect to PostgreSQL databases. It even offers a specialized *PostGIS Layer*, which fetches data from the database and recognizes geometry (spatial) columns in tables. Choosing a table with such a column leads to the visualization of the contained data on the map. The data can further be filtered by the table's other attributes or generally by adding an SQL filter statement.

Figure 4.10 shows the created base map of Vienna used for further evaluation. Figure 4.11 illustrates all available trip data, so all the 170938 location points from the spatial database on top of the base map. Both the Figures are exports from the QGIS software.

The locations depicted in Figure 4.11 reveal some insight into what the operating areas of the car sharing operators look like. The data covers the majority of the inner city with parts of the big districts north east of the Danube. Moreover, there are obviously two dropoff possibilities that are further away from the core operating area. A parking lot at the airport offers customers the option to go back and forth between the airport and the city center. This explains the cumulated locations in the south east of the figure. Another external parking possibility is offered around the big shopping mall south of Vienna, the *Shopping City Süd*. The cumulated points are clearly visible at its location as well. The characteristics of those external dropoff possibilities are more the ones of a non-floating car sharing system as the area for pickup and dropoff is limited to a quite small area. There are a few more points scattered around the core operating area. These are most likely errors in the data or maintenance appearances. The true reason for these particular observations cannot be distinguished without the cooperation of the car sharing operators.

Figure 4.10: Base map overview of Vienna (QGIS export).

Clustering algorithm

As the aim of this work focuses on the usage of clustering methods to explore the correlation between points of interest in a city and car sharing usage, an obvious and important question is which clustering algorithm to perform those evaluations with. Since an evaluation of different clustering algorithms exceeds the scope of this masters thesis, further evaluations will be done using one single algorithm. The clustering algorithm that was chosen is DBSCAN [11]. DBSCAN was initially developed for spatial databases and offers a density based method to find clusters in data sets. A necessary parameter is *Epsilon*, which will be determined by experimental analysis on the different datasets under investigation. A further parameter of the algorithm is *minPoints*, which indicates the minimum number of points, a cluster must have. However, the authors of DBSCAN [11] propose to set *minPoints* to the value 4 as their experiments showed that higher values do not change the results significantly for 2-dimensional data. In comparison to many other algorithms, DBSCAN can handle noise data points that do not belong to any cluster, which seems very reasonable for the current use case of clustering geographical points of car sharing trips. As geographical clusters with high density are of special interest for this thesis, DBSCAN seems like a good choice. Nevertheless, future work could deal with different clustering algorithms on the present data as other algorithms might lead to different of better results.

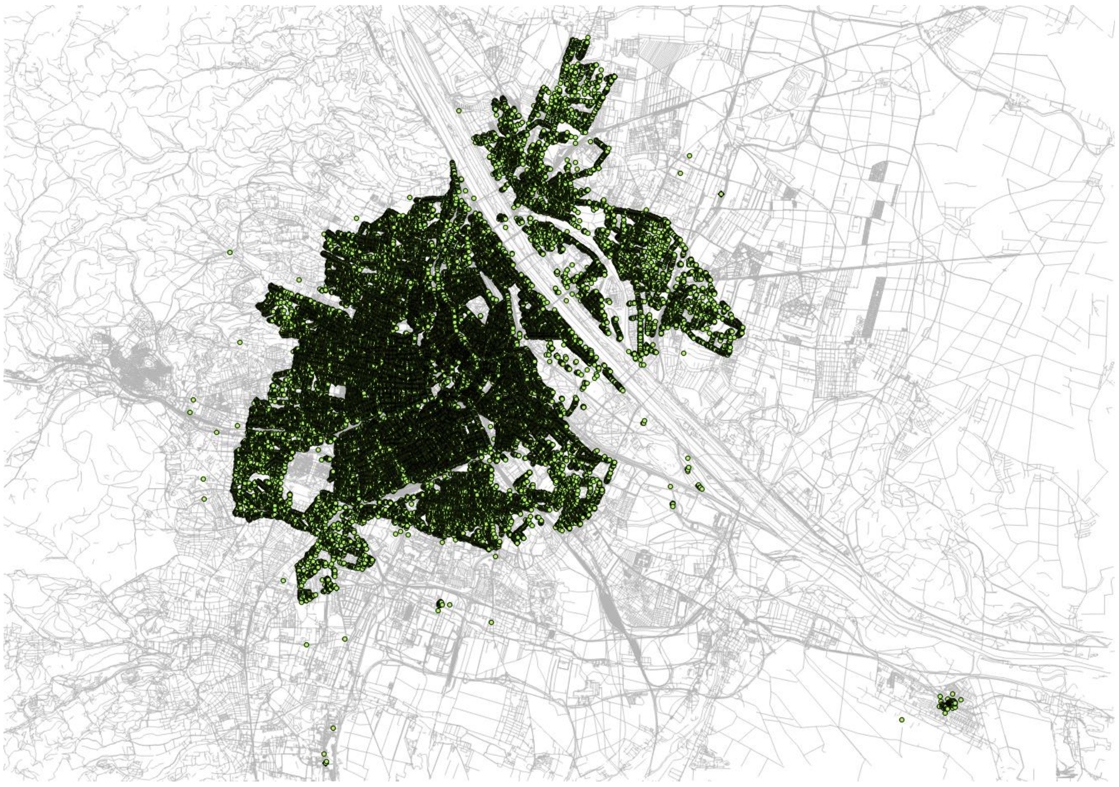


Figure 4.11: Complete dataset of trip data (QGIS export).

The set up PostgreSQL database has, like described before, a PostGIS extension installed. PostGIS offers a great number of useful functionality when it comes to evaluations on geographical data. One of them is the support for DBSCAN clustering on the spatial data stored in a database. The PostGIS function *ST_ClusterDBSCAN* [26] represents an implementation of a two-dimensional DBSCAN algorithm based on geo-spatial data and returns an integer value for each geometry in the passed dataset. This integer value represents the constructed cluster, the concerning point (or geometry) is belonging to. For noise points, so geometries, that do not belong to any cluster, a null value will be returned. The function has three parameters. The *geometry* parameter is the set of geometries, the algorithm should cluster, *eps* is the epsilon value of DBSCAN and *minpoints* defines the minimum amount of clustered points, a single cluster should contain. The function can easily be included into PostgreSQL statements and assigns the concerned car sharing locations to clusters.

Before going deeper into the available and prepared data, it might be interesting to get a general insight using cluster analysis. As the dataset is based on trips and weather from the city of Vienna, as a first step general hot spots or tendencies for car sharing usage were explored.

To do so, it was first tried to generate a layer in QGIS that contains the clustered points. The first try was a layer based on the complete dataset that computed the clusters in the scope of QGIS. This ended up in very poor performance of QGIS as after every scrolling, zooming or similar action, the whole underlying SQL statement was executed. The clustering statement on the whole dataset of 170938 points took between one and two minutes (depending on the input parameters). As this approach did not lead to a very satisfying result, the next try was the creation of a materialized view based on the clustering statement. The performance was, as expected, quite good with this method as the result of the query was stored permanently in the database after just one execution and no complex calculations had to be done in QGIS after that.

Experimenting with the input parameters of *ST_ClusterDBSCAN* finally led to the clustering result outlined in Figure 4.12. The visible points are only those contained in calculated clusters. The points of each cluster were given the same color using QGIS's style functionality. The style is *Categorized* and the configured column for the categorization is *cluster_id*. The input parameters for this result were *0.001 degrees for epsilon* and *250 minimum points* per cluster. The underlying materialized view was created with the PostgreSQL statement shown in Listing 4.16.

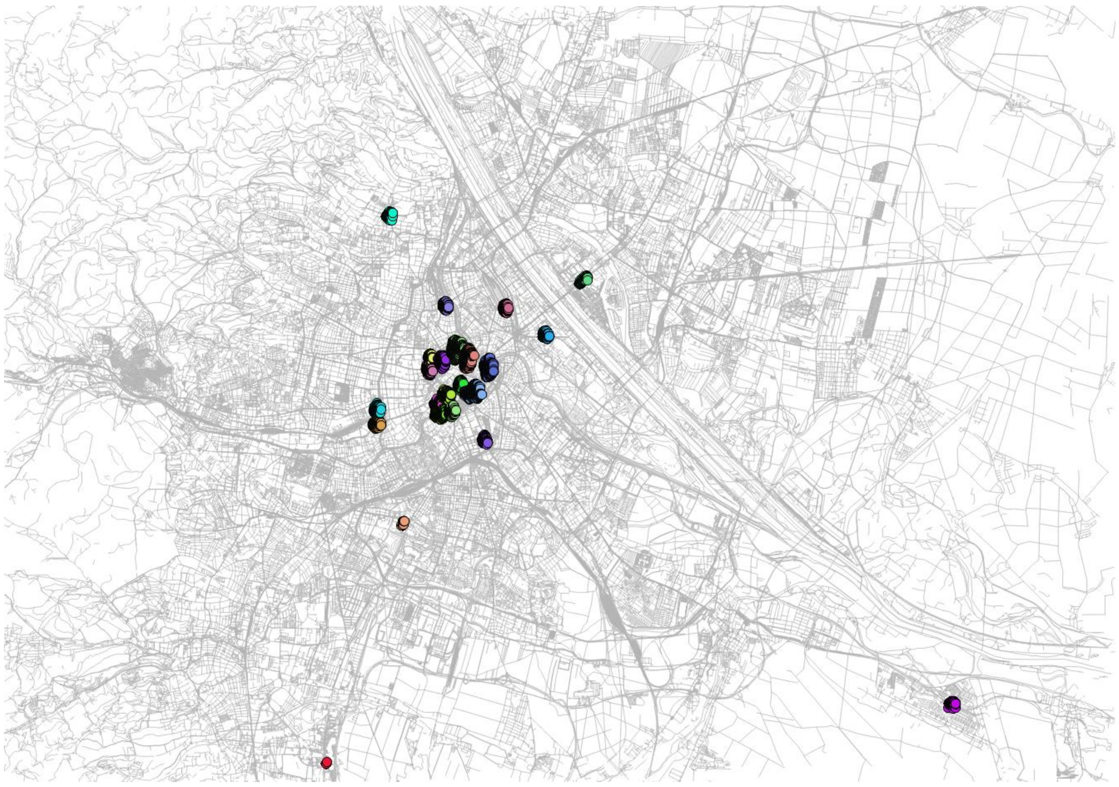


Figure 4.12: Cluster analysis on complete dataset (epsilon 0.001 degrees, minimum 250 points.)

Figure 4.12 clearly outlines the general car sharing hot spots in the city. As expected, clusters accumulate in the inner city center where a lot of office buildings, tourist attractions and other points of interest are located. There are a number of clusters located around universities. The Technical University of Vienna in the south of the city center as well as the University of Vienna in the west have multiple clusters located around their location. Another cluster appears in the second district near the Vienna University of Economics and Business. Another pattern seems to be observable around railway stations. The Westbahnhof in the west, the Hauptbahnhof in the

```sql
create materialized view evaluation as
  select row_number() over () as id, sq.*
    from (
      select cd.location,
          ST_ClusterDBSCAN(cd.location, eps := 0.001,
             minPoints := 250)
          over () as cluster_id
        from carsharing_data cd
    ) sq
  where cluster_id is not null
```

Listing 4.16: Materialized view for cluster analysis on complete dataset.

south as well as the Franz-Josefs-Bahnhof in the north all have clusters near their geographical positions. Another set of clusters appears around business parks. A cluster is located near the Vienna International Center north of the Danube just like south of the Meidling train station, where another business park with multiple shopping facilities and a big cinema are located. The northernmost cluster is right in the middle of a residential area. It is located right on the outer border of the operator's operating area and could be a result of commuters that leave their private cars outside of town due to the short-term parking zone in the city center. Finally and most obvious, there are 2 clusters outside of the core operating areas in the south. One of them represents the parking lot at the airport while the other is located near the operating area extension at the *Shopping City Süd*.

4.5 Point of interest analysis

Section 4.4 already outlined a few of the possibilities QGIS has concerning the integration of OpenStreetMap data. This section will take things a bit further and describe how specific facilities of interest and their locations can be extracted from the OpenStreetMap database. While the extraction of *nodes*, *polylines* and *polygons* was described before, a closer look at *tags* will be made at this point. Each of the three extractable features can have tags attached, which give the feature meaning. Tags, as outlined in the chapter Relevant Data 3, can be seen as metadata attached to the features. In general, tags are simply a list of key-value pairs, so they can contain arbitrary content. Still, there is a number of tags that are interpreted by client applications and therefore have to follow some sort of standard. The Map Features page in the OpenStreetMap Wiki [23] describes a good number of tags, that the community agreed on and therefore the list acts as an informal standard for users. As most features can be described using only a few of those tags, it is easy to extract certain features with common attributes by filtering by their tag values. Using this tagging method, the points of interest for this thesis are used from OpenStreetMap.

To use them in further evaluations, the *nodes* and the *polylines* of the area of interest in and around Vienna were extracted from OpenStreetMap and written into a SpatialLite database

```
"amenity" in ("university", "college")
```

Listing 4.17: QGIS filter statement for OpenStreetMap data, Universities.

contained in the QGIS project. QGIS offers the functionality for this step by navigating to the top menu and choosing Vector -> OpenStreetMap (holds for QGIS version 2.18.2). When exporting the data to SpatialLite, the user can choose the type of feature to export as well as which *tags* should be exported with them. For the purpose of this thesis, *nodes* and *polygons* are of special interest as they can both represent points of interest in a city. For this reason, both of them were exported and used as layers on top of the base map in QGIS. The tags that were exported with them were the ones with the following keys:

- *name*: The general name of a feature/location.
- *leisure*: Tag used for identifying facilities for leisure activities or sports.
- *amenity*: Tags different facilities that can be used or visited by citizens (e.g. bars, schools, universities, restaurants, etc.).
- *public_transport*: Identifies locations with public transport characteristics.
- *shop*: Tag used for facilities that sell goods.

Those tags are the ones containing the necessary information to identify the points of interest for the hypotheses to evaluate. After the export job is done, the layer for nodes and one for polygons is automatically added to the project. As a next step, the data should be filtered in order to only display facilities and points of interest on the map. For example, if only the universities should be visible on the map and used as points of interest, the filter shown in Listing 4.17 could be added to both the points and the polygons layer in QGIS. This would result in the map shown in Figure 4.13. The Figure shows only the city center of Vienna and it's universities, so does not outline the complete operating area.

4.6 Evaluation of hypotheses

Using a combination of all the methods explained before, the stated hypotheses can be evaluated. By laying multiple QGIS layers over the base map, a correlation between points of interest and clusters generated from relevant empirical data can be deduced. The hypotheses can be confirmed or rejected using the available data. Where possible, counter samples were used to check the validity and compare evaluation results. One general assumption will be made in the evaluation of every hypothesis. As the parking lots, that are not directly attached to the operating area, have non-floating car sharing characteristics, they appear as clusters in the majority of the evaluations. The reason for this behavior is, that cars can only be rented and returned into parking lots without any choice for the location of pickup and dropoff. So all data points in those areas have (almost) the exact same location. For this reason, the clusters formed over those areas will be ignored when evaluating the hypotheses.

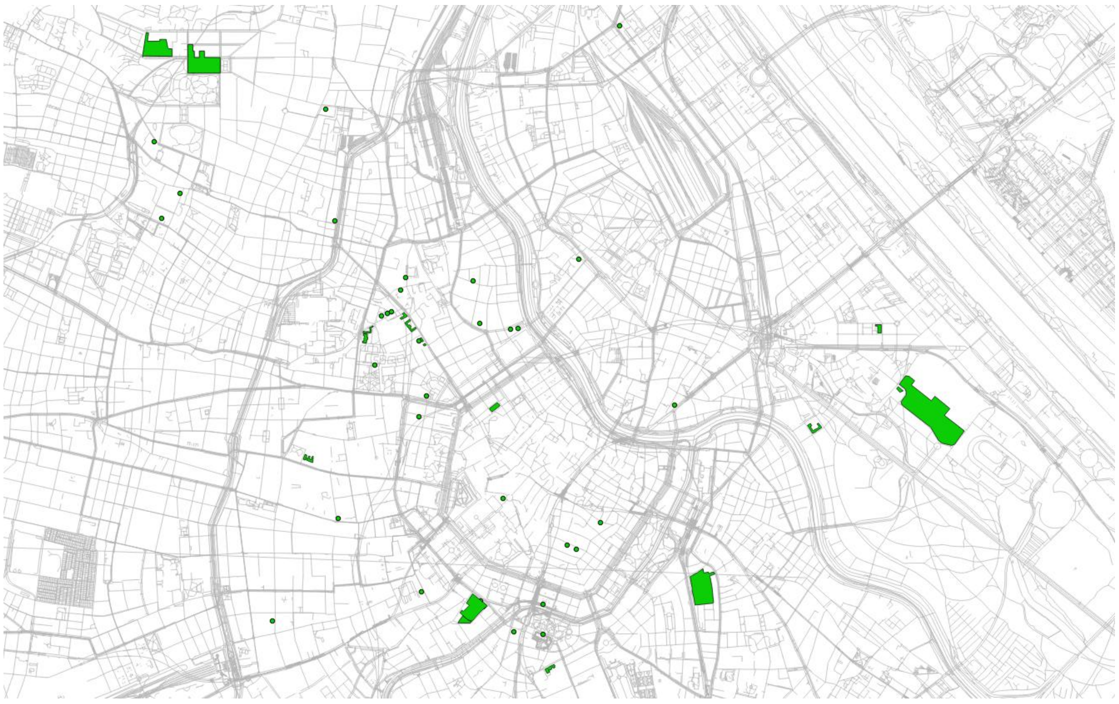


Figure 4.13: OpenStreetMap points of interest, Universities (QGIS export)

Choosing input parameters for DBSCAN

The clustering algorithm DBSCAN has two input parameters *epsilon* and *minPoints*. Obviously, the choice of those parameters has a big influence on the outcome of the clustering process. There is a good number of literature discussing the topic of parameter approximation, but most of the papers come to the conclusion that the best way to choose them is by integrating domain knowledge into the decision process. There is a method of approximation, the original DBSCAN paper [11] suggests, that gives good insight into the structural properties of a given data set. For better parameter choice in the context of this thesis, the suggested method was applied to a subset of the data available. The method is described in good detail in the paper. It deals with the interpretation of the *sorted k-dist graph* and in general consists of the following steps:

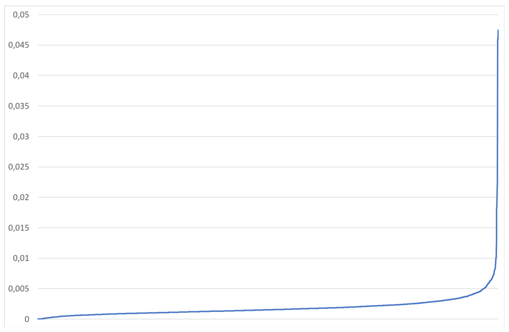
- Choose the distance function between two data points. PostGIS offers *float ST_Distance (geometry g1, geometry g2)* for this matter.
- For every point in the domain, calculate the distance to it's k-th nearest neighbor.
- Sort the points by the calculated distance.
- Print a graph based on the sorted distance data. This graph is called the *sorted k-dist graph*.
- Find the first (or any) 'knee' in the data and use the *k-distance* of it as *epsilon* and *k* as the *minPoints* parameter for DBSCAN on the dataset.

```sql
select neighbordistance from (
   select source.id as sourceId,
         destination.id as destinationId,
         ST_distance(source.location, destination.location) as
            distance,
         lead(ST_distance(source.location,
            destination.location), 4) over (partition by
            source.id order by ST_distance(source.location,
            destination.location)) as neighborDistance
      from (select id, location
            from carsharing_data) as source,
         (select id, location
            from carsharing_data) as destination
      ) as neighbors
where sourceId = destinationId
order by neighbordistance
```

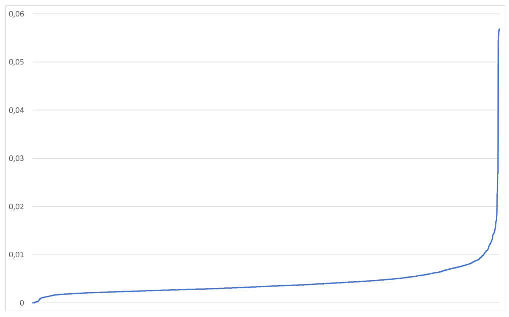
Listing 4.18: SQL statement for computation of *sorted 4-dist graph*.

The described method was applied to a subset of the available location data points. The subset was the same as used in the evaluation of Hypothesis 1 and is described in more detail later on. Listing 4.18 shows the SQL statement for the calculation of the data for the *sorted 4-dist graph*. As the statement is computationally quite heavy, it was only applied on the relatively small subset of data points containing 3365 tuples (the full set of data consists of 170938 tuples) but it can be assumed, that the structure of the data is similar in the full data set. Figure 4.14 shows the sorted 4th- and 15th-dist graph for the estimation of *epsilon*. In both cases, for *k=4* as well as for *k=15*, the 'knee' of the graph is at a similar location of the distance axis. Also, it shows that the general curve is quite similar between the two *k* values. When the suggestion of the authors of DBSCAN would be followed, a good *epsilon* value choice would be the one right at the knee of the curve. In this case, all points in the data with distance values lower than the chosen *epsilon*, so the ones to the left of the knee, would be core points and therefore part of clusters. Points with higher values would mainly be classified as noise. The knees for both cases are very far to the right of the curve, which can be interpreted in a way that there are not a lot of noise points and a majority of points being classified as core points when choosing the suggested epsilon value. Furthermore, cluster algorithm results react very sensible on changes of epsilon towards lower values. In general, one could say that the data is quite well distributed over the operating area because of the curve being pretty flat on the majority of it's behavior.

The intended method in this thesis requires the human readability and interpretability of the generated clusters. The *sorted k-dist graph* analysis shows, that for high values of *epsilon*, the majority of all points in the domain would be core points, which might not be a desired outcome of the clustering approach. For the clusters to be interpretable in a good way, only a limited number is preferable. The continuous calculation of *k-dist graphs* for different datasets



(a) k=4



(b) k=15

Figure 4.14: Sorted k-dist graphs of subset of empirical car sharing data.

is computationally quite heavy. For those reasons, domain knowledge and experiments with different *epsilon* and *minPoints* values led to the results shown in the following sections about the evaluation of the stated hypotheses. The experiments included methods like fixing one parameter to a certain value and varying the other one or choosing *minPoints* dependent on the underlying (sub-)dataset. In any case, parameters were chosen based on domain knowledge and varied until a satisfying cluster count and distribution was reached. To make useful assumptions about car sharing behavior, the clusters should emphasize certain areas or points of interest. That's why a very large or very small number of clusters is not desirable.

Experiments based on all the four formulated hypotheses showed that an *epsilon* value of 0.001, which equals approximately 111 meters, is a good value to use with the given data set. The *minPoints* value can roughly be deducted from the size of the sub-sample used for clustering. For the outlined experiments, a *minPoints* value between 0,1 % and 0,2 % of the sub-sample tuple size led to reasonable results. After all, these rules were followed in every experiment to be able to compare results between different clustering approaches.

Hypothesis 1: Daytime, rain, shopping facilities

'During shop opening hours on rainy days, people tend to pick up car sharing vehicles around indoor shopping facilities.'

When thinking about indoor shopping facilities, as stated in the introductory chapter, malls might be a good example to base the evaluations on. The aim of this section should be to proof or decline, that there are more car sharing pickups around such facilities on rainy days and during shop opening hours compared to dry days. Before doing any technical evaluations, one should think about, what rainy (or dry) day and shop opening hours exactly are with respect to the available data.

The *weather* table in the database gives information about precipitation in every five minutes in the observation period. Summing the values up, it is easy to get rain amount per hour for a certain period. Rainy hours shall from here on be classified as those, that have **more than one liter per hour** precipitation. The hours below this threshold value shall be interpreted as dry hours. Typical shop opening hours in Vienna are from **9 a.m. to 8 p.m.**, which also are the

```
select row_number() over () as id, sq.* from (
  select cd.location,
    ST_ClusterDBSCAN(cd.location, eps := 0.001, minPoints :=
       6) over () as cluster_id
   from carsharing_data cd, (
      select time::date as theDate,
       EXTRACT(hour from time) as theHour,
       sum(rainfall) as rain
      from weather
      group by thedate, thehour
    ) as rainyHours
   where cd.pickup::date = rainyHours.thedate
   and EXTRACT(hour from cd.pickup) = rainyHours.theHour
   and EXTRACT(dow from cd.pickup) = 0
   and rainyHours.rain >= 1
   and rainyHours.theHour between 9 and 19
  ) sq
where cluster_id is not null
```

Listing 4.19: Clustering statement for pickups on rainy hours with shops open.

times taken into account for the evaluation of the current hypothesis. The sub-dataset, on which the clustering process for this hypothesis should be based on, must therefore be filtered with respect to those values. The interesting time stamp for this hypothesis is the pickup column. After applying all outlined filters on the data, 3365 tuples remain as the sub-dataset used for the clustering process. Listing 4.19 shows the SQL statement developed concerning this matter.

To be able to find a coherence between the clusters generated by the algorithm, a set of points of interest has to be defined. For the hypothesis under examination, malls should be taken as a representative subset for indoor shopping facilities. It shall be investigated, if clusters of car sharing pickups occur around those facilities. To visualize only the points of interest concerning malls based on the OpenStreetMap data set, the filter **"shop" in ("mall")** was added to the full dataset of points and the one of polygons. The malls and therefore points of interest for this hypothesis are displayed on the base map after this step. The chosen color for those facilities is green. Table 4.3 shows a summary of the parameters used for the evaluation.

Figure 4.15 shows the result of the evaluation of the hypothesis. In this and the following visualizations, points of interest are colored in green and clusters have a color gradient from yellow to red, depending on the size of the cluster with respect to the count of contained points. The darker the points of a cluster, the bigger the cluster. When interpreting the results, there indeed are a few points of interest with clusters in their immediate vicinity. Nevertheless, those points of interest are mainly located in the center of the operating area, i.e. the city center. There are a few well-known malls a bit outside of the inner city, like the 'Donauzentrum' in the north-east, the 'Kaufpark Alt-Erlaa' in the south or the 'Millenium City' in the north. None of those

parameter	**value**
epsilon	0.001
minPoints	6
POI filter	“shop“ in (“mall“)
weather filter	rain >= 1 liter / m^2
time filter	pickups workdays, 09:00 - 19:59

Table 4.3: Parameters for evaluation of hypothesis 1.

important Points of interest have clusters near them. This can clearly be interpreted as evidence for the rejection of the hypothesis under examination.

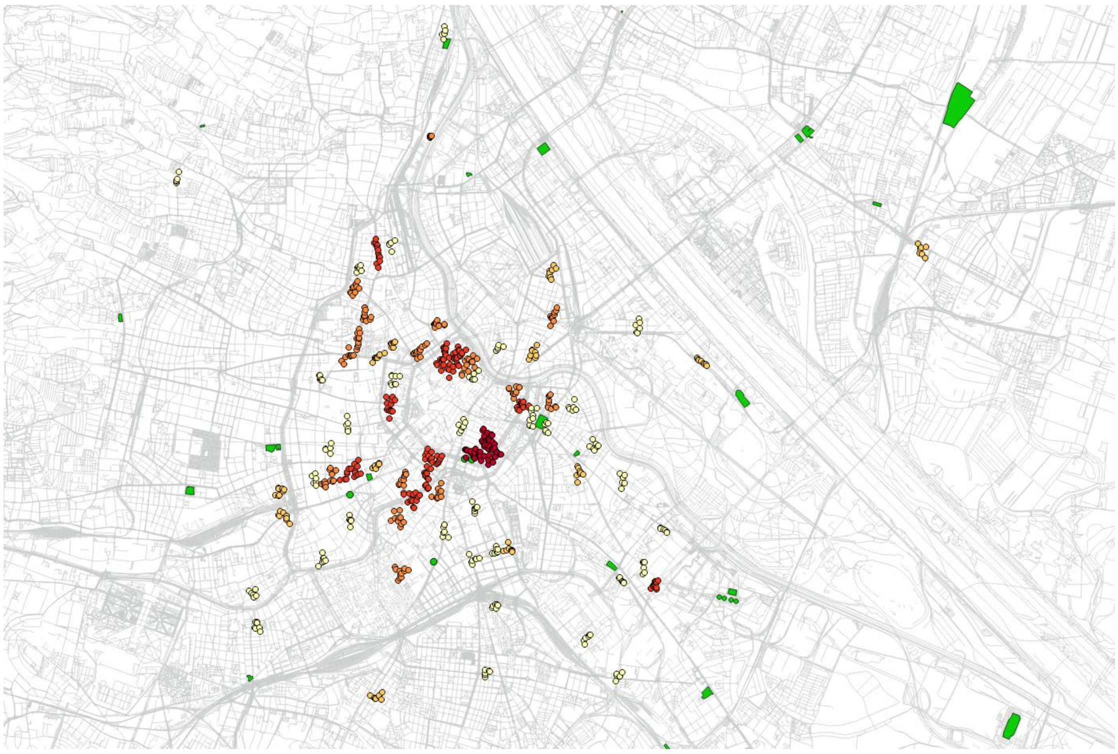


Figure 4.15: Evaluation of Hypothesis 1: Shopping malls and pickups during rainy hours on workdays.

This interpretation is substantiated when taking a look at the cluster analysis of a counter sample for the hypothesis. Figure 4.16 shows a cluster analysis done with different filters on the weather data. The evaluation is based on the same data set with slightly changed parameters. More concrete, the rain filter was changed to **< 1 liter / m^2** so only pickups in dry weather are contained in the base data. Furthermore, the **minPoints** value was increased to **100** due to the significantly higher number of location entries. In this evaluation, it is clearly visible, that the majority of points of interest have pickup clusters near them. This means that not only is there no increase in car sharing usage around indoor shopping facilities during shop opening hours when it rains, the usage also seems to decrease in those conditions. Based on the data available for this thesis, and the chosen parameters, this hypothesis must be rejected.

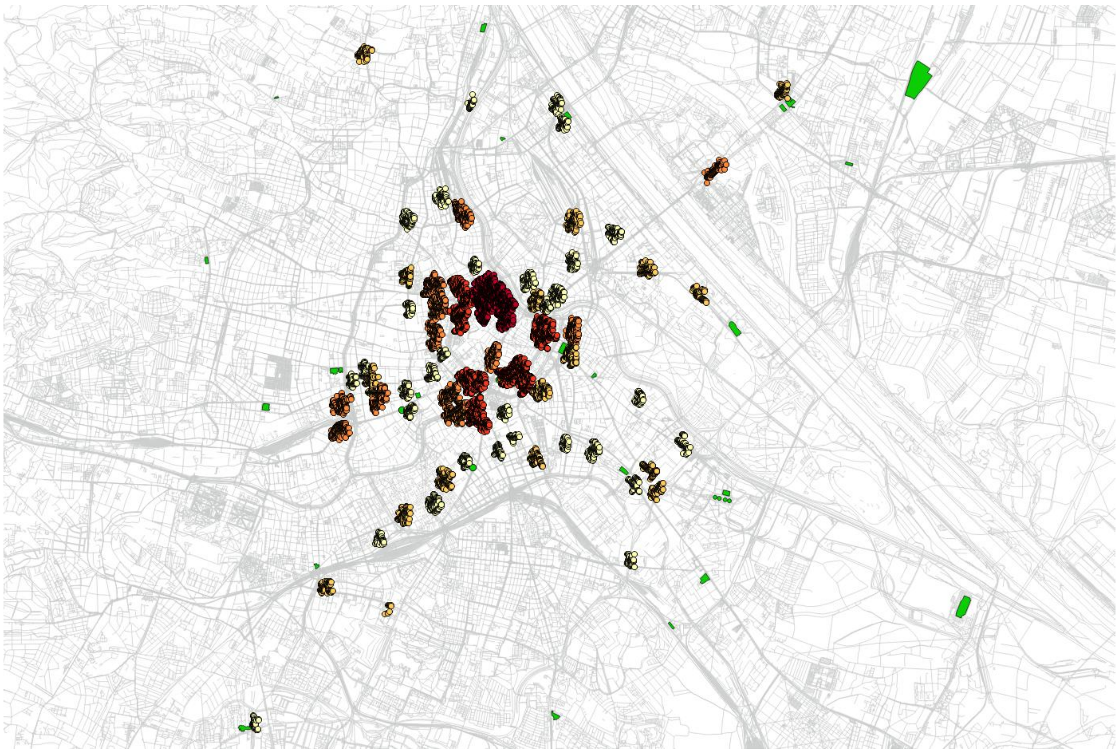


Figure 4.16: Counter sample evaluation of Hypothesis 1: Shopping malls and pickups during dry hours on workdays.

Hypothesis 2: Daytime, amenities

During the daytime, customers tend to drop off car sharing vehicles near amenities for daily business.

There are many amenities for daily business, therefore one certain type was chosen to be the representative set of points of interest for this hypothesis. Hospitals appear in a meaningful number within the operating area, so they are a good choice to do evaluations with. To show hospitals from the OpenStreetMap database on the base map, the filter **"amenity" in ("hospital")** was set on both the points and the polygons layer of the point of interest data.

Again, a few requirements of the hypothesis have to be interpreted and defined as filters on the dataset. The cluster algorithm should only be based on data points that occurred during the daytime. As a first thought, one could interpret daytime as the period from sunrise to sunset, from the early morning hours to late evening or the time between the morning and the evening rush hour. Thinking a bit further, the rush hour periods could have a big impact on the data set because, as shown earlier, a large share of trips occur in the morning and in the evening.

```sql
select row_number() over () as id, sq.* from (
  select cd.location,
    ST_ClusterDBSCAN(cd.location, eps := 0.001, minPoints :=
       45) over () as cluster_id
  from carsharing_data cd
   where EXTRACT(hour from cd.dropoff) between 10 and 15
   and EXTRACT(dow from cd.dropoff) not in (0,6)
   ) sq
where cluster_id is not null;
```

Listing 4.20: Clustering statement for dropoffs in the daytime on workdays.

Therefore, it could make sense to exclude the rush hour periods from the evaluation dataset as they would probably bias the result of the evaluation because clusters would mainly appear around business trip dropoff locations. For this reason, the chosen time filter parameter is from **10 a.m. to 4 p.m.**. In contrary to the first hypothesis, the time stamp of interest for this evaluation is the dropoff one as it shall be examined, if people drop vehicles off around the highlighted points of interest. Another filter set on the trip data is the limitation of the points based on weekdays. Weekend trips to hospitals would potentially be e.g. visits or emergencies and shall not be part of the evaluation data as they cannot be defined as daily business. For this reason, only data from Monday through Friday shall be taken into account.

Listing 4.20 shows the developed SQL statement that was used for the evaluation based on the given requirements of the hypothesis. It contains all the filters described earlier and is based on a dataset with a tuple size of 34080. Table 4.4 gives an overview of the used parameters, that were already discussed in this section.

parameter	**value**
epsilon	0.001
minPoints	45
POI filter	“amenity“ in (“hospital“)
weather filter	-
time filter	Monday to Friday, 10:00 - 15:59

Table 4.4: Parameters for evaluation of hypothesis 2.

When having a look at the results of the evaluation visualized in Figure 4.17, it is noticeable that many of the highlighted points of interest are far off the next cluster. The clusters obviously accumulate in the city center, where a few of them are also near hospitals. Nonetheless, it is not distinguishable whether those clusters are formed because of the highlighted points of interest or because of other reasons. For example, there are a few clusters around the ’Allgemeines Krankenhaus’, the ’Krankenhaus der barmherzigen Brüder’ or the ’Sophienspital’, that probably have relation to the points of interest. On the other hand, the clusters around the ’Sophienspital’ in the west of the inner city could also have to do with the near ’Westbahnhof’, a major train station right next to the hospital that offers train connections and various shopping possibilities. An area, where a correlation would be clearly visible, is the outer part of the operating area. A few major Viennese hospitals are located in the north ('SMZ Nord'), the east ('SMZ Ost') or the west ('Wilheminenspital') of the city. All of those points of interest are clearly visible on Figure 4.17 and are within the operating area of the operators under examination. None of them have any clusters in their surroundings. Many of the highlighted hospitals are located on a larger area. For this reason, the *epsilon* of the clustering method was increased for a counter sample test as the clusters around those points of interest might have a bigger geographical size. This change in parameters also did not show any accumulation of points around the outer, major hospitals.

In general, the evaluation result seems to show a tendency towards shopping trips rather than daily business. Many of the clusters appear around the major shopping centers described in the last hypothesis. Based on the given result, no clear relocation strategy can be suggested with respect to hospitals as representative points of interest for amenities of daily business. So this hypothesis has to be rejected based on the available data and current method.

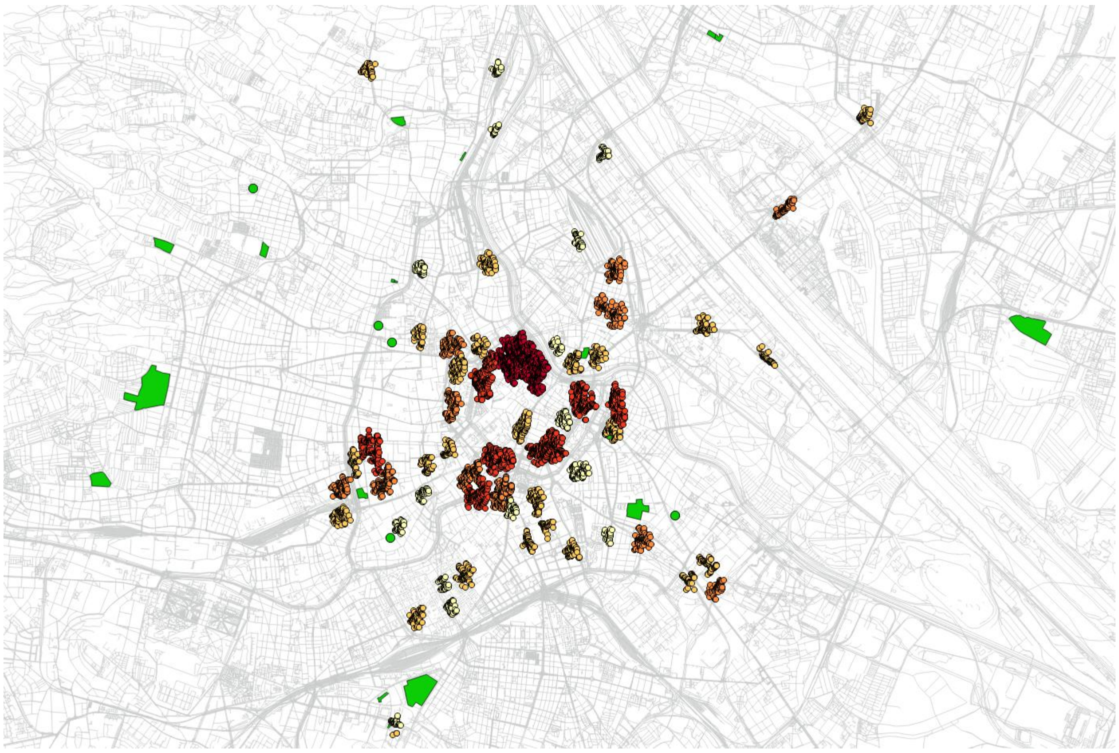


Figure 4.17: Evaluation of Hypothesis 2: Hospitals and dropoffs on workdays.

Hypothesis 3: Evenings, rain, leisure activities

On rainy evenings, customers tend to drop off car sharing vehicles around indoor leisure facilities.

```sql
select row_number() over () as id, sq.* from (
  select cd.location,
    ST_ClusterDBSCAN(location, eps := 0.001, minPoints := 5)
       over () as cluster_id
  from carsharing_data cd, (
    select time::date as theDate,
      EXTRACT(hour from time) as theHour,
      sum(rainfall) as rain
    from weather
    group by thedate, thehour
  ) as rainyHours
  where cd.dropoff::date = rainyHours.thedate
  and EXTRACT(hour from cd.dropoff) = rainyHours.theHour
  and rainyHours.rain >= 0.5
  and rainyHours.theHour between 18 and 23
) sq
where cluster_id is not null;
```

Listing 4.21: Clustering statement for dropoffs during rainy evenings.

As already outlined in the introductory hypothesis definition, one of the most popular indoor leisure activity might be going out to watch a movie in a cinema. As there are many cinemas within the operating area under examination, they might also be representative points of interest for the evaluation of the thesis. To highlight cinemas stored in OpenStreetMap's database, the filter **"amenity" in ("cinema")** was added to point and polygon layers. Another good representative choice would be theaters ("amenity" in ("theatre")), but taking both those points of interest types would produce too many highlighted locations on the base map. For this evaluation, cinemas shall be the amenities to go with.

Obviously, the relevant time stamps from the *carsharing_data* table are the ones from the *dropoff* column. It shall be examined, whether customers take trips towards the leisure facilities. Evenings shall be defined as the period of time **between 6 p.m. and midnight** for this case as this is potentially the time where people do the most leisure activities. As there is not a very big number of rainy evenings in the available weather dataset, the rain threshold value was decreased compared to Hypothesis 1 in order to get a larger sample size. 0.5 liters / m^2 meter per hour shall be the threshold value for this hypothesis. This amount of precipitation should be enough for people living in urban regions to decide for an indoor leisure activity over an outdoor one.

Listing 4.21 shows a possibility for a clustering statement for the described situation. The underlying select statement returns a subset of locations consisting of 2872 tuples. Also for this hypothesis, the chosen parameters were summed up in Table 4.5.

The results of the evaluation of the hypothesis is shown in Figure 4.18. It seems pretty obvious that there indeed are a lot of points of interest that have trip clusters near them. The accumulation of clusters in the city center is by far not as concise as in the evaluation results in

parameter	value
epsilon	0.001
minPoints	5
POI filter	“amenity“ in (“cinema“)
weather filter	rain >= 0.5 liters / m^2
time filter	daily, 18:00 - 23:59

Table 4.5: Parameters for evaluation of hypothesis 3.

Hypothesis 2. Clusters are well-spread over the operating area, and a lot of the major cinemas in the city have some of them attached. The cinemas in the city center, e.g. the ’Lugner Kino’, the ’Apollo’ or the ’Haydn Cinema’ in Vienna’s seventh district are clearly surrounded by dropoff clusters. Even more notable, also the cinemas outside of the inner city are mostly attached to generated clusters. This can be interpreted as clear evidence between the hypothesis conditions and indoor shopping facilities with cinemas as representatives. Further experiments with counter samples were also done. If the filter values are changed in a way that all dropoff points in the same period of time but with rain less than 0.5 liters / m^2 (’dry evenings’) are taken into account for the clustering algorithm, the clusters appear more distributed over the whole operating area and the proximities of cinema points of interest are not quite as filled with clusters as in the original sample. This could be interpreted in a way that car sharing users prefer other activities on dry evenings or just do not use car sharing as much on these days.

Based on the reasons outlined above, this hypothesis shall be confirmed by the method under examination, and based on the available data. There seems to be a clear correlation between rainy evenings and car sharing dropoffs around cinema locations.

Hypothesis 4: High temperatures, public transportation

On hot days, car sharing around subway stations increases compared to days with moderate temperature.

Vienna has a very well-developed public transportation system with a dense network of stops for trains, subway, trams and buses. Its five subway lines combine for 109 stops spread over the whole city. As the subway lines and their stops are potentially the most frequented stops in the system, they shall be taken as the representative points of interest. When filtering the point layer of QGIS’s OpenStreetMap data by **“public_transport“ in (“station“)**, all the subway stations appear as highlighted points of interest on the base map. A further layer for polygon data is not necessary to show all the concerned locations for this evaluation.

When it comes to filtering trip data, the first question that arises is how to define hot days. The weather data breakdown outlined earlier in this thesis shows, that there are a few days with maximum temperatures **over 30 degrees Celsius**, which might be a good threshold value for the hypothesis under examination. The weather data should therefore be investigated by the maximum temperatures per hour and those hours with a higher than 30 degrees max should be taken into account for clustering. The SQL in Listing 4.22 shows a further filter concerning time. Only data points, that occurred in subway system operating hours shall be taken into

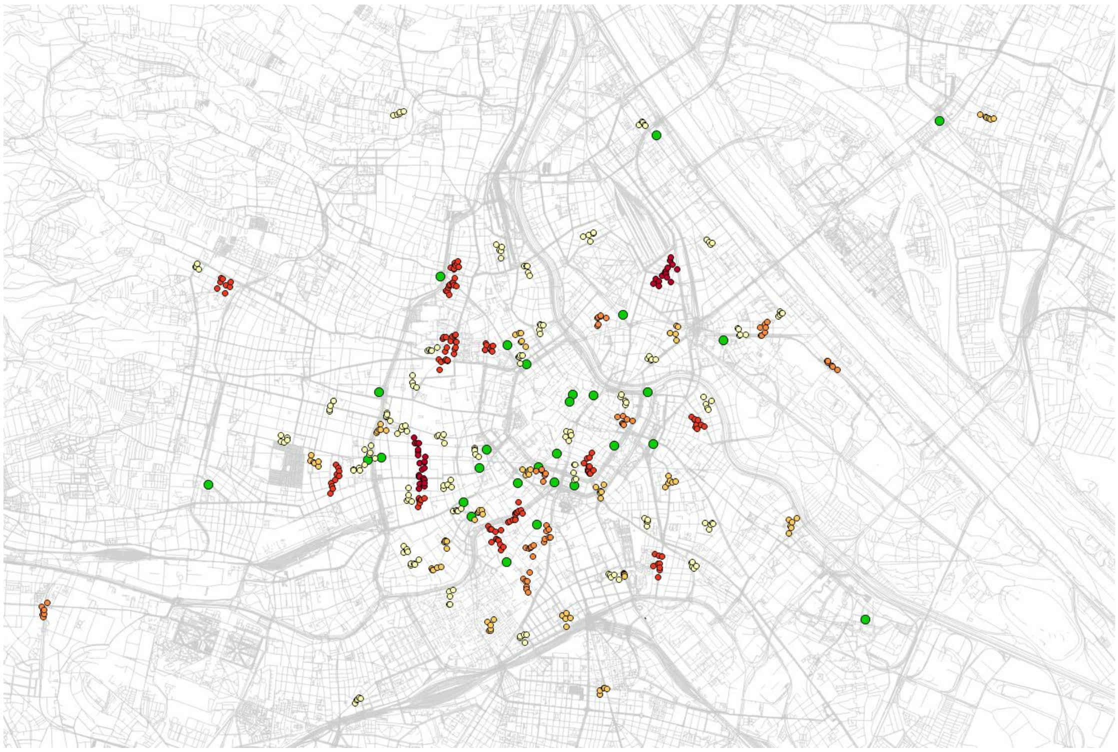


Figure 4.18: Evaluation of Hypothesis 3: Cinemas and dropoffs on rainy evenings.

account. While this might seem unnecessary because of the low likelihood of temperatures above 30 degrees during night times in Vienna, it is important for counter sample tests done with data coming from time periods with temperatures below the threshold value. The used time stamps for this case are the pickup ones as it should be found out, if customers pick up car sharing vehicles rather than taking the subway on such hot temperature days. The outlined clustering statement is based on a sub dataset of 32208 tuples. The parameters for the evaluation are outlined in Table 4.6.

parameter	**value**
epsilon	0.001
minPoints	35
POI filter	“public_transport“ in (“station“)
weather filter	temperature >= 30 degrees Celsius
time filter	daily, 05:00 - 23:59

Table 4.6: Parameters for evaluation of hypothesis 4.

The results of the evaluation can be seen in Figure 4.19. The clusters generated from the method show that there are a lot of pickup accumulations around subway stations. Similar to evaluations shown earlier, this can be seen the best in the outer parts of the operating area,

```
select row_number() over () as id, sq.* from (
  select cd.location,
    ST_ClusterDBSCAN(cd.location, eps := 0.001, minPoints :=
       35) over () as cluster_id
  from carsharing_data cd, (
    select time::date as theDate,
      EXTRACT(hour from time) as theHour,
      max(temperature) as maxTemp
    from weather
    group by thedate, thehour
    having max(temperature) >= 30
  ) as hotHours
  where cd.pickup::date = hotHours.thedate
  and EXTRACT(hour from cd.pickup) = hotHours.theHour
  and hotHours.theHour between 5 and 23
) sq
where cluster_id is not null;
```

Listing 4.22: Clustering statement for pickups during hours with high temperatures.

for example in the areas east of the Danube, where there are many recreation areas for summer months, like public pools, are located. It seems that even though people could easily take subway lines, as they are right next to a lot of generated clusters, they still pick up car sharing vehicles due to convenience. An example for a counter sample experiment result is shown in Figure 4.20, which was made with the same *epsilon* value but a higher *minPoints* parameter of 140 due to the larger sample size. The sample is based on data from the time period, but coming from hours with maximum temperatures less than 30 degrees Celsius. It shows, that the clusters generated by this sample have different locations and structures. Their closeness to the highlighted points of interest is not as pronounced and some of the subway stations in the outer parts of the city do not have any clusters in their proximity. The area east of the Danube described above is a good example for this behavior.

Based on the available data and the method presented in this thesis, the current hypothesis can be considered confirmed due to the reasons outlined above.

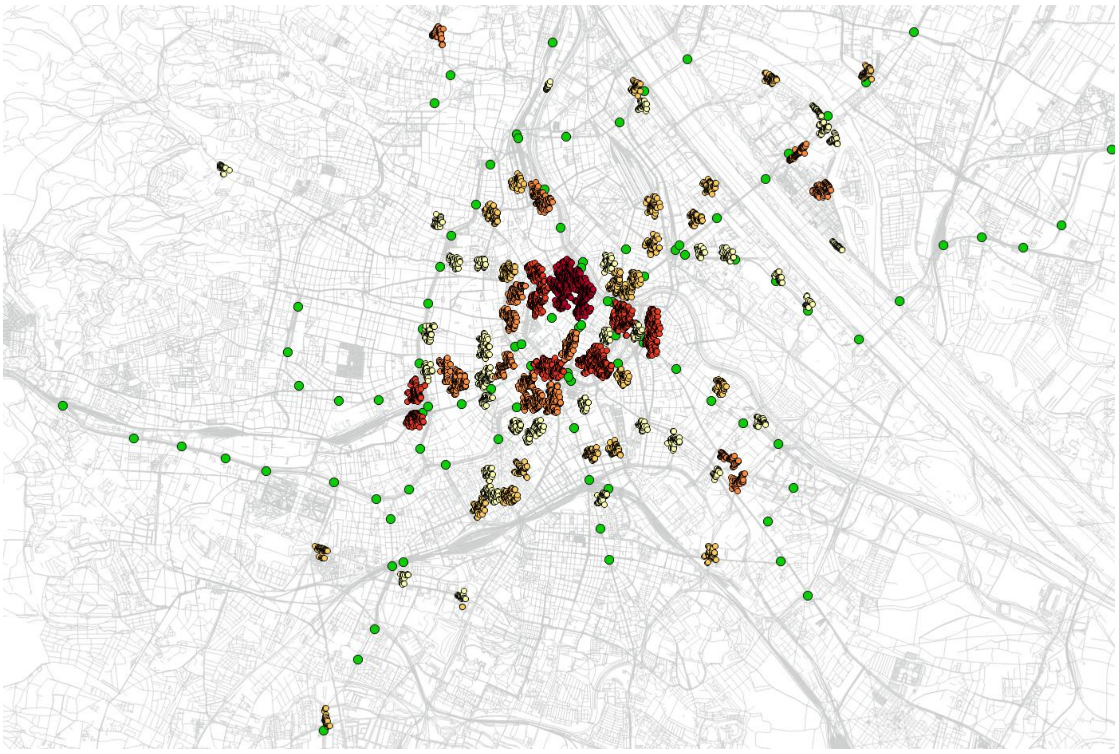


Figure 4.19: Evaluation of Hypothesis 4: Subway stops and pickups during hours with high temperature.

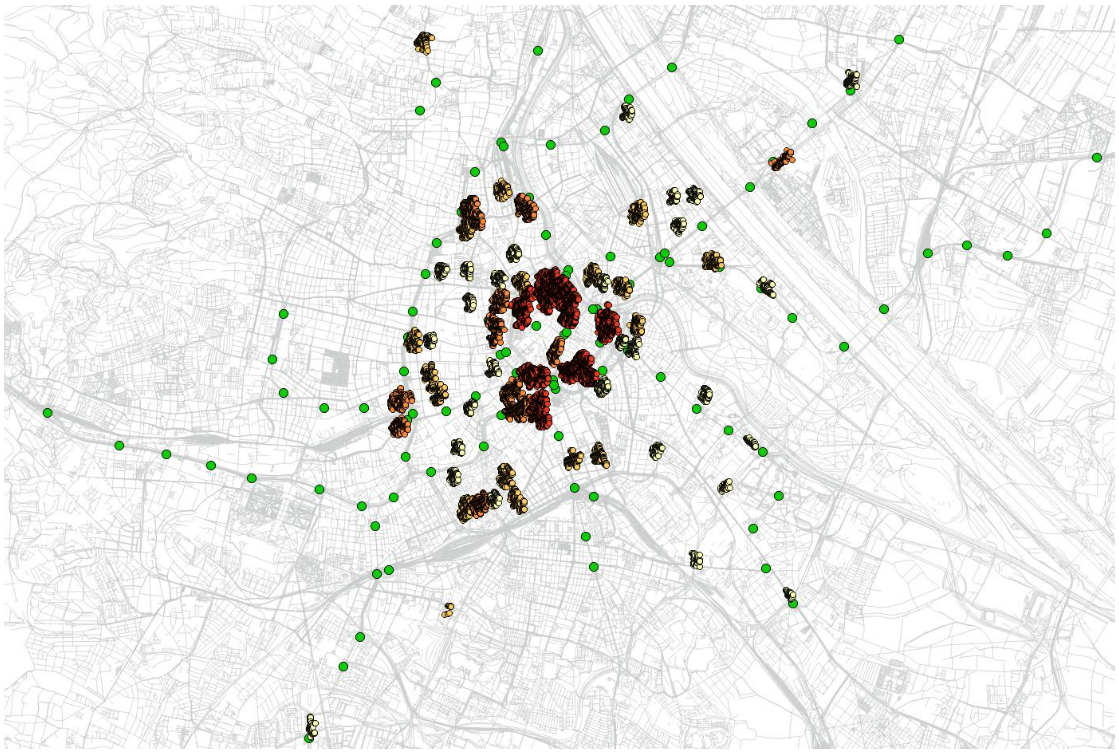


Figure 4.20: Counter sample evaluation of Hypothesis 4: Subway stops and pickups during hours with temperatures below 30 degrees Celsius.

CHAPTER 5

Results

5.1 Method

The presented method is a generic approach to knowledge discovery of usage patterns in free-floating car sharing systems. The method is easily applicable for car sharing operators, as well as other potential stakeholders. The method can be used to evaluate various assumptions about system structures and usage patterns in the operators' operating areas, and therefore makes a valuable contribution to the improvement of their relocation strategies. Independent of the results derived from the hypotheses in this thesis, the approach can be used with any dataset containing trip data. A wide range of hypotheses related to free-floating car sharing usage could be evaluated using the outlined method. The application potential of this approach extends far beyond the current study. The current study validated the proposed method on a specific example, and the method was presented in a detailed, technical way based on open source software. However, the concrete implementation tested herein can of course be imitated by operators. In addition, the conceptual and methodological innovations discussed in this thesis could also be implemented with a different set of tools, e.g. tools already used in the daily business of the operators' affected departments.

While the presented method focuses mainly on improvement of operator's own relocation strategies, it could also be employed to conduct competitive analysis. Operators could easily implement the approach to gain insight into the behaviors of competitors' systems. Through competitive analysis, they could adapt their own operating areas or focus their relocation destinations towards high-demand areas of their competition.

5.2 Operator suggestions

The hypotheses that were evaluated in the preceding chapter give an insight into the characteristics of free-floating car sharing user behavior in Vienna. Two of the four stated hypotheses could be confirmed using the proposed method, the other two had to be rejected. From the evaluations, a number of suggestions for car sharing operators can be deduced.

Suggestion 1: User incentives or operator-based relocations on rainy evenings around indoor leisure facilities: The evaluation of hypothesis 3 shows that car sharing users tend to drop off cars near indoor leisure facilities in case of rain in the evening hours. The effect of such a behavior is the accumulation of cars near those facilities, while the vehicles could be missing at other, high-demand locations. To avoid this negative effect, user incentives could be offered in order to motivate the user to end their current trip near a region with higher demand compared to the already filled area around the target facility. To make this possible, user incentives need to be possible in an existing system. If this is not the case, the operator could use relocation agents to minimize the clustering behavior around the affected locations. To react on such situations as quick as possible, the employees responsible for relocation could be strategically placed within the operating area in case of bad weather. In case of a weather forecast with rain for an evening period, the car sharing system could be brought in a state which counteracts this accumulated vehicle appearance.

Suggestion 2: Coverage of high demand of vehicles around public transportation stations during hot summer days: Hypothesis 4 was confirmed due to the accumulation of car sharing pickups around subway stations in hours with high maximum temperatures. This identifies subway stations as high demand regions during such hot days. People tend to pick up cars around them, which brings revenue potential when covering the higher demand of car sharing usage in those areas. Therefore, existing relocation strategies could be adapted in a way, that they react on high temperatures and arrange relocations accordingly.

Suggestion 3: Coverage of high demand of vehicles around recreational areas during hot summer days: A side effect of the evaluation of hypothesis 4 was the occurrence of rather large clusters around outdoor recreational areas like public pools under the described conditions (hot days). The most noticeable example is the area around Vienna's 'Alte Donau', where a number of clusters appeared in the evaluation of the data. That brings another obvious revenue potential that can be exploited quite easily by prompting the right relocation strategies.

Hypotheses 1 and 2 had to be rejected based on the available data and the proposed method and therefore cannot be the base for any operator suggestions. Apart from the actual evaluation of hypothesis, an exploratory analysis of the underlying data was made revealing some information that could be quite useful for operators. The break down of car sharing departures per day shows that the smallest number of trips in the systems are started in the early morning hours. This could be the best time to do operator-based relocation trips, as the reorganization of the system does not interfere with customer trips, and fewer cars are needed in the system.

The few parking lots outside of the core operating area (airport, shopping centers) appeared as clusters in the majority of evaluation approaches due to their nature of relatively large numbers of vacant cars in a quite small geographical area. They have the characteristics of a station in a non-floating car sharing system and can therefore be excluded from the present approach of density-based cluster analysis. For those areas, an approach presented by Cao et al. [3] could be a useful one applicable for the organization of vacant car amounts and the relocation of them.

5.3 Critical reflection

The presented method requires a human being to do an important part of the evaluation, namely the interpretation of the correlation between clusters and points of interest. For this reason, an easy scaling to multiple cities or operating areas requires a equally increased amount of human resources. It would be preferable if the whole evaluation process could be automated and therefore integrated in existing decision support systems. The results of the evaluations presented in this thesis only hold for the available set of underlying data. If the proposed method would be executed based on a complete data set coming directly from the operator's databases, the process would probably lead to different results.

Over the course of the evaluations, continuously a high number of clusters appeared in the city center. This behavior might mask clusters, that are of actual interest for a hypothesis under examination. A solution for this problem could be segmentation of the operating area into smaller sections and running the proposed methods on the different sections. A method for the segmentation of the operating areas was proposed as parts of the papers by Caggiani, Ottomanelli, Camporeale and Binetti [2] or by Weikl et al. [36].

In this thesis, only one clustering algorithm, DBSCAN, was used to find accumulation of car sharing vehicles. Other algorithms might lead to different, maybe even better, results. The examination of results from different clustering algorithms would have exceeded the scope of the thesis, but would be interesting for future work.

Compared to related work, the contribution of this thesis is the analysis of spatial car sharing data with density-based clustering techniques. This approach was nowhere to be found in the related work to the best of the authors knowledge. Furthermore, the combination of data from multiple car sharing operators seems new in this research field. Past studies dealt with the regression analysis of empirical car sharing data, and also clustering methods were used to divide operating areas into different demand zones using partitioning clustering techniques. A combination of different approaches (e.g. regression analysis with density-based clustering analysis) would be a valuable contribution for future work in this area.

In summary, the proposed method contributes to the field by giving deeper insight into free-floating car sharing behavior by finding correlations between certain points of interest and density-based location clusters of vacant cars that can be used to improve car sharing operator's relocation strategies.

CHAPTER 6

Summary and Future Work

This thesis gave insight into free-floating car sharing behavior in order to improve the relocation strategies of car sharing operators. First, a literature review of the current state of the art of the research field was made and hypotheses related to car sharing behavior were formulated that should be evaluated by a proposed method. The data, that was relevant for further evaluations, and its sources were outlined before the generic approach on how to perform knowledge discovery on the data was introduced. A Java prototype, which was developed in order to collect and transform the data to evaluable form, was presented. The Java program writes the transformed data into a spatial database, which was presented in the following section with it's most important elements. After collecting the dataset necessary for further tasks, a section of the thesis described data preprocessing and normalization tasks before an exploratory analysis on the data ready for final evaluation gave a general, coarse insight into Viennese free-floating car sharing behavior based on temporal and meteorological influence factors. After this, the used visualization techniques and clustering algorithm were presented in further detail and general problems with input parameters were discussed. After presenting the chosen method to extract and interpret data from OpenStreetMap as a source for points of interest, the clustering of filtered data and their correlation to points of interest was presented in order to evaluate the formulated hypotheses. This was done by choosing representatives for points of interest in the map data, defining a relevant subset of trip location entries, performing the cluster algorithm with certain parameters and interpreting a visual result for every hypothesis. The hypotheses were further on being confirmed or rejected in order to formulate suggestions on how to improve operator's relocation strategies as a result of the thesis. A critical reflection and a discussion about limitations completes the presented work.

The presented method is applicable for anybody interested in free-floating car sharing structures, while it could be very valuable for operators offering such systems. The method starts with the collection and normalization of the base dataset. These steps might not be necessary for operators, that have direct access to their historical data, on which they could base the proposed evaluation method on.

6.1 Limitations

The clustering approach presented relies on high input data quality. Solid, reliable results can be produced when complete and correct input data are used. Car sharing operators with active systems in Vienna were asked to cooperate with the study, but did not want to offer data or other internal knowledge. Therefore, the data had to be deduced from other sources described earlier in the thesis. The datasets and code snippets were further anonymized in order to comply with the wishes of the operators. As a result, the author had to work with incomplete data. The evaluation could only be made over a limited period of time, as it had to be collected in real time. This period of time was in the summer months, so no seasonal variations could be investigated and the results only represent car sharing usage behavior during this time of the year. The underlying data was also incomplete concerning user trips. Operator-based relocation trips could not be distinguished clearly from regular user trips. Therefor, the relocations had to be kept as part of the base data for clustering. Furthermore, a dropoff action with a subsequent pickup action right after it can theoretically not be noticed by the developed algorithm as it only takes snapshots every 5 minutes. Such a trip would be interpreted as one single long trip and the dropoff and pickup location, which would be a significant part of the data set, would be ignored. One could also discuss the data quality of the used point of interest dataset. OpenStreetMap is a community project with great success and the data seems to be of high quality in most cases. Still, some discrepancies appeared over the course of the evaluations. In this manual evaluation method, those errors do not have a lot of influence, in a fully automatic method, on the other hand, results could get biased. The whole method could be adapted to deliver better results, if certain additional data would be available. For example, the user's reasons for the trip would be a great addition to the dataset. Kopp, Gerike and Axhausen [14] presented a method to collect such data.

6.2 Future work

Future work based on this thesis could contain the evaluation of data based on different density-based clustering algorithms to optimize produced results. An automatic evaluation method that can be executed without the need for manual evaluation would be a great addition to existing decision support systems. Obviously, the method could be applied on data coming from other cities and car sharing operators and to gain even more insight into the car sharing systems, the sources and destinations of the trips under examination could be taken into account besides the pure location data of vacant vehicles. Another aspect that would be interesting to investigate is the absence of clusters, so areas that could further on be defined as low-demand regions. With such knowledge, an operating area could be adjusted to fulfill customers needs better. Last but not least, the combination of different data sources would be an interesting addition to the presented method. For example, the combination of car sharing and bike sharing data could be a next step. Also, adding data about demographic city characteristics, like from the Viennese city government [16], Statistik Austria [33] or the Open Data Portal [38] would contribute to more insight into the topic. Finally, the addition of public transportation data, like malfunctions or train delays, would cover a lot of ground not yet investigated by current research work.

APPENDIX A

Data Models

A.1 Car sharing database

```
create table car (
  licensePlate varchar(20) PRIMARY KEY,
  operator varchar(20) not null,
  model varchar(100) not null,
   created timestamp not null default CURRENT_TIMESTAMP
);

create table carhistory (
  licenseplate varchar(20) references car(licensePlate),
  fuelLevel smallint not null,
  innerCleanliness varchar(20) not null,
  address varchar(255) not null,
  arrival timestamp,
  departure timestamp,
  location geometry(Point,4326) not null,
   created timestamp not null default CURRENT_TIMESTAMP
);

create table weather (
  time timestamp not null,
  temperature float not null,
  humidity float not null,
  rainfall float not null
);

```

```
create table importeddata (
  operator varchar(100) not null,
  timestamp_file timestamp not null,
   created timestamp not null default CURRENT_TIMESTAMP
);

CREATE INDEX ON carhistory(licenseplate);
CREATE INDEX ON carhistory(arrival);
CREATE INDEX ON carhistory(departure);
CREATE INDEX ON weather(time);

create materialized view carsharing_data as
  select c.licenseplate as licenseplate,
   ch.address as address,
   ch.arrival as dropoff,
   ch.departure as pickup,
   ch.fuellevel as fuellevel,
   ch.innercleanliness as innercleanliness,
   c.operator as operator,
   c.model as model,
   ch.location as location,
   row_number() over (order by c.licenseplate, ch.arrival) as
      id
   from car c, carhistory ch
  where c.licenseplate = ch.licenseplate;
```

APPENDIX B

Source Code

B.1 Java prototype

Collecting data from operators APIs

CarsharingDataCollector.java

```java
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Hashtable;

/**
 * Represents the main class of the prototype collecting snapshots
    of free cars from car sharing operators APIs.
 * @author Philipp Dressler
 */
public class CarsharingDataCollector {

  public static void main(String[] args) {

    //replace with actual API URLs
    final String operatorAUrl = "https://<url_to_operatorA_API>";
    final String operatorBUrl =
       "https://<url_to_another_operator_API>";

    //some APIs don't need headers...
    Hashtable<String, String> operatorAProperties = new
       Hashtable<String, String>();
    //operatorAProperties.put("", "");

    //... and other do.
    Hashtable<String, String> operatorBProperties = new
        Hashtable<String, String>();
    operatorBProperties.put("X-Api-Key", "<some_guid>");

    //initialize connections
    OperatorConnection operatorAConnection = new
        OperatorConnection(operatorAUrl, operatorAProperties);
    OperatorConnection operatorBConnection = new
        OperatorConnection(operatorBUrl, operatorBProperties);

    //loop indefinitely
    while (true) {

      String operatorAData = "";
      String operatorBData = "";

      //get snapshots from both operators...
      try {
        operatorAData = operatorAConnection.getCarData();
      } catch (IOException e) {
        System.out.println("ERROR - operatorAConnection: " +
            e.toString());
      }

      try {
        operatorBData = operatorBConnection.getCarData();
      } catch (IOException e) {
        System.out.println("ERROR - operatorBConnection: " +
            e.toString());
      }

      //... and write them into files with unique filename.
      PrintWriter out;
      try {
        String fileName = new
            SimpleDateFormat("'operatorA_'yyyyMMddHHmmss'.txt'")
           .format(new Date());
        out = new PrintWriter(fileName);
        out.println(operatorAData);
        out.close();
      } catch (FileNotFoundException e) {
        e.printStackTrace();
      }

      try {
        String fileName = new
            SimpleDateFormat("'operatorB_'yyyyMMddHHmmss'.txt'")
            .format(new Date());
        out = new PrintWriter(fileName);
        out.println(operatorBData);
        out.close();
      } catch (FileNotFoundException e) {
        e.printStackTrace();
      }

      //finally, sleep a little and start over again
      try {
        Thread.sleep(300000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
}
```

OperatorConnection.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Hashtable;

/**
 * Represents the connection to a car sharing operators API that
    returns the status of the car sharing system.
 * @author Philipp Dressler
 */
public class OperatorConnection {

  //the URL to the API
  private String url;

  //the header properties needed to connect to the API
  private Hashtable<String, String> headerProperties;

  /**
   * Initializes a new OperatorConnection object with the given
      properties.
   * @param url The URL to the API.
   * @param headerProperties The header properties needed to connect
      to the API.
   */
  public OperatorConnection(String url, Hashtable<String, String> headerProperties) {
    this.url = url;
    this.headerProperties = headerProperties;
  }

  /**
   * Establishes a connection to the Object's API URL using the given request header properties to get a snapshot of the operators vacant car data.
   * @return A String containing the raw json car data returned from the API of the operator.
   * @throws IOException In case of connection error.
   */
  public String getCarData() throws IOException {

    //open connection to URL
    URL obj = new URL(url);
    HttpURLConnection con = (HttpURLConnection) obj.openConnection();
    System.out.println("Establishing connection to " + url);

    con.setRequestMethod("GET");

    //add necessary request headers
    for (String key : headerProperties.keySet()) {
      System.out.println("Adding request header property " + key + ": " + headerProperties.get(key));
      con.setRequestProperty(key, headerProperties.get(key));
    }

    //print response code
    int responseCode = con.getResponseCode();
    System.out.println("Response Code : " + responseCode);

    //read and return car data
    BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
    String inputLine;
    StringBuffer response = new StringBuffer();

    while ((inputLine = in.readLine()) != null) {
      response.append(inputLine);
    }
    in.close();
    con.disconnect();

    return response.toString();
  }
}
```

Transforming data into evaluable form in a PostgreSQL/PostGIS database

CarsharingDataDecoder.java

```java
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Date;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.zip.ZipEntry;
import java.util.zip.ZipException;
import java.util.zip.ZipFile;

import org.json.JSONException;


public class CarsharingDataDecoder {

  /**
   * The main method used to decode snapshot files taken from
      operator APIs and load them into the spatial database.
   * The method runs through all the files in all the archives in
      temporal order and performs the necessary operations on the DB.
   * The data used here should be the one created by the
      CarsharingDataCollector class.
   * PRECONDITION: Snapshot files compressed into (multiple) .zip
      formatted archives located in ./Data folder
   * @param args
   * @throws ZipException
   * @throws IOException
   * @throws SQLException
   * @throws ParseException
   */
  public static void main(String[] args) throws ZipException,
      IOException, SQLException, ParseException {

    //initialize DB connection
    CarsharingDbConnection dbConnection = new
        CarsharingDbConnection(
        "jdbc:postgresql://localhost/carsharing_db", "postgres",
      "<your_password>");

  //archive files with car sharing data have to be located under
      ./Data
  File directory = new File("Data");
  File[] zipFiles = directory.listFiles();
  //files have to be sorted in order to have them in temporal order
  Arrays.sort(zipFiles);

  //iterate over all zip files in Data directory
  for(File zipDataFile : zipFiles) {

    System.out.println("Processing zip-archive
        "+zipDataFile.getName());

    ZipFile zipFile = new ZipFile(zipDataFile);
    Enumeration<? extends ZipEntry> entryFiles =
        zipFile.entries();

    //iterate over all zip archive entries
    while(entryFiles.hasMoreElements()){

      ZipEntry entryFile = entryFiles.nextElement();
      System.out.println("Processing file "+entryFile.getName());

      //retreive contents of entry file
      String data =
          streamToString(zipFile.getInputStream(entryFile));

      ArrayList<Car> fileCars;
      Timestamp snapshotTimestamp;
      SimpleDateFormat dateFormat = new
          SimpleDateFormat("yyyyMMddHHmmss");
      String operator;

      try {
        //operators have different json file structures - use
            different decoders
        if (entryFile.getName().startsWith("operatorA_")){

          //retrieve all cars from the current file as Car objects
          OperatorADecoder operatorADecoder = new
              OperatorADecoder(data);
          fileCars = operatorADecoder.getCars();

          //create Timestamp object for DB
          //file name must be operatorA_YYYYMMDDHHMMSS.txt
           Date parsedDate =
              dateFormat.parse(entryFile.getName().substring(9,
                 23));
             snapshotTimestamp = new
                 Timestamp(parsedDate.getTime());
            operator = "operatorA";

          } else if (entryFile.getName().startsWith("operatorB_")){

            //retrieve all cars from the current file as Car objects
            OperatorBDecoder operatorBDecoder = new
                OperatorBDecoder(data);
            fileCars = operatorBDecoder.getCars();

            //create Timestamp object for DB
            //file name must be operatorB_YYYYMMDDHHMMSS.txt
             Date parsedDate =
                 dateFormat.parse(entryFile.getName().substring(7,
                 21));
             snapshotTimestamp = new
                 Timestamp(parsedDate.getTime());
             operator = "operatorB";

          } else {
            System.out.println("Unknown Filetype: " +
                entryFile.getName());
            continue;
          }
        } catch (JSONException e) {
          System.err.println(e.toString());
          continue;
        }

        //at this point, we have an ArrayList of Car objects
            representing the parsed file
        System.out.println("File contains "+fileCars.size()+"
            entries");

        Hashtable<String, Car> htFileCars = new Hashtable<String,
            Car>();

        //for every contained in the file...
        for(Car car : fileCars) {
          if (!dbConnection.existsCar(car.getLicensePlate())) {
            //if current car does not exist in the database: new
                car in System, create and dropoff
            dbConnection.createCar(car);
            dbConnection.carDropoff(car, snapshotTimestamp);
            System.out.println("Car appeared: " + car);
          }
          else if(!dbConnection.isCarParked(car)) {
```

```java
            //if car is not parked, so on a trip in the database and now appears in the free car snapshot: car dropoff
            dbConnection.carDropoff(car, snapshotTimestamp);
            System.out.println("Car dropoff: " + car);
          }
          //car is surely parked from here on
          else if(dbConnection.changedLocation(car) ) {
            //if location of current car is different to its database location: car pickup and dropoff
            dbConnection.carPickup(car.getLicensePlate(), snapshotTimestamp);
            dbConnection.carDropoff(car, snapshotTimestamp);
            System.out.println("Car pickup and dropoff: " + car);
          }

          //finally, transform fileCars to Hashtable for pickup check
          htFileCars.put(car.getLicensePlate(), car);
        }

        //for every parked car of the current operator in the database...
        for (String licensePlate : dbConnection.getParkedCars(operator)) {
          if(!htFileCars.containsKey(licensePlate)) {
            //if car is parked in the database and vanished from the free cars snapshot: car pickup
            dbConnection.carPickup(licensePlate, snapshotTimestamp);
            System.out.println("Car pickup: " + licensePlate);
          }
        }

        //finally, log import history in database
        dbConnection.createImportHistory(operator, snapshotTimestamp);
      }

      zipFile.close();
      System.out.println("Archive "+zipFile.getName()+" sucessfully decoded and imported into database.");
    }
  }

  /**
   * Returns the String coming from a given InputStream in UTF-8 encoding
   * @param stream The input stream to extract data from.
   * @return Stream contents in String format.
```

```
     * @throws IOException
     */
    private static String streamToString(InputStream stream) throws
        IOException {
      StringBuilder inputStringBuilder = new StringBuilder();
        BufferedReader bufferedReader = new BufferedReader(new
            InputStreamReader(stream, "UTF-8"));
        String line = bufferedReader.readLine();
        while(line != null){
            inputStringBuilder.append(line);
            inputStringBuilder.append('\n');
            line = bufferedReader.readLine();
        }
        return inputStringBuilder.toString();
    }

}
```

Car.java

```
/**
 * The class encapsulating the properties of a single car in a car
    sharing system.
 * @author Philipp Dressler
 */
public class Car {

   private String operator;
   private String licensePlate;
   private String model;
   private CarStatus status;

   /**
    * Retrieves the value of the operator of the current car object.
    * @return The operator name.
    */
   public String getOperator() {
      return operator;
   }

   /**
    * Sets the operator value of the current car object.
    * @param operator Name of the operator.
    */
   public void setOperator(String operator) {
      this.operator = operator;
   }
```

```

  /**
   * Retrieves the value of the license plate of the current car
       object.
   * @return The license plate value.
   */
  public String getLicensePlate() {
    return licensePlate;
  }

  /**
   * Sets the license plate value of the current car object.
   * @param licensePlate Value of the license plate.
   */
  public void setLicensePlate(String licensePlate) {
    this.licensePlate = licensePlate;
  }

  /**
   * Retrieves the value of the model of the current car object.
   * @return The model name.
   */
  public String getModel() {
    return model;
  }

  /**
   * Sets the model value of the current car object.
   * @param model Name of the model.
   */
  public void setModel(String model) {
    this.model = model;
  }

  /**
   * Retrieves the current status of the car object.
   * @return The status of the car.
   */
  public CarStatus getStatus() {
    return status;
  }

  /**
   * Sets the status of the car object.
   * @param status Status to set car to.
   */
  public void setStatus(CarStatus status) {
    this.status = status;
  }
```

```java

  @Override
  public String toString() {
    return operator + " " + licensePlate + " " + model + " @ " +
        status;
  }


}
```

CarStatus.java

```java
/**
 * The class representing a status, a car can have in a car sharing
     system.
 * @author Philipp Dressler
 */
public class CarStatus {

  private int fuelLevel;
  private String innerCleanliness;
  private String address;
  private double latitude;
  private double longitude;

  /**
   * Retrieves the fuel level of the car in the current status.
   * @return The fuel level of the car.
   */
  public int getFuelLevel() {
    return fuelLevel;
  }

  /**
   * Sets the fuel level of the car status object.
   * @param fuelLevel The fuel level to set.
   */
  public void setFuelLevel(int fuelLevel) {
    this.fuelLevel = fuelLevel;
  }

  /**
   * Retrieves the inner cleanliness of the car in the current
       status.
   * @return The inner cleanliness value of the car.
   */
  public String getInnerCleanliness() {
    return innerCleanliness;
  }

  /**
   * Sets the inner cleanliness of the car status object.
   * @param innerCleanliness The inner cleanliness to set.
   */
  public void setInnerCleanliness(String innerCleanliness) {
    this.innerCleanliness = innerCleanliness;
  }

  /**
   * Retrieves the address of the car in the current status.
   * @return The address of the car in the current status.
   */
  public String getAddress() {
    return address;
  }

  /**
   * Sets the address of the car status object.
   * @param address The address to set.
   */
  public void setAddress(String address) {
    this.address = address;
  }

  /**
   * Retrieves the latitude of the position of the car in the
      current status.
   * @return The latitude value of the car.
   */
  public double getLatitude() {
    return latitude;
  }

  /**
   * Sets the latitude of the car in the current status.
   * @param latitude The latitude value to set.
   */
  public void setLatitude(double latitude) {
    this.latitude = latitude;
  }

  /**
   * Retrieves the longitude of the position of the car in the
      current status.
   * @return The longitude value of the car.
   */
  public double getLongitude() {
    return longitude;
  }

  /**
   * Sets the longitude of the car in the current status.
   * @param longitude The longitude value to set.
   */
  public void setLongitude(double longitude) {
    this.longitude = longitude;
  }

  @Override
  public String toString() {
    return address + " (" +latitude + "," + longitude + ")";
  }

}
```

OperatorADecoder.java

```
import java.io.IOException;
import java.util.ArrayList;

import org.json.JSONArray;
import org.json.JSONObject;

/**
 * The class that encapsulates methods to decode raw API data from
    operator A to Java objects.
 * @author Philipp Dressler
 */
public class OperatorADecoder {

  private JSONObject data;

  /**
   * Initializes the decoder object with a given data String.
   * @param data The raw car data to decode.
   * @throws IOException
   */
  public OperatorADecoder(String data) throws IOException {
    this.data = new JSONObject(data);
  }

  /**
```

```java
   * Decodes the raw car data to Car and CarStatus objects
     accordingly to the json file structur of the API of operator A.
   * @return The list of cars decoded from the car data String.
   */
  public ArrayList<Car> getCars() {

    ArrayList<Car> cars = new ArrayList<Car>();

    JSONArray jsonCars =
        data.getJSONObject("cars").getJSONArray("items");
    for (int i = 0; i < jsonCars.length(); i++) {

      Car car = new Car();
      CarStatus status = new CarStatus();

      car.setOperator("OperatorA");
      car.setLicensePlate(jsonCars.getJSONObject(i)
          .getString("licensePlate"));
      car.setModel(jsonCars.getJSONObject(i).getString("modelName"));
      status.setFuelLevel(jsonCars.getJSONObject(i)
          .getInt("fuelLevelInPercent"));
      status.setInnerCleanliness(jsonCars.getJSONObject(i)
          .getString("innerCleanliness"));
      JSONArray jsonAddress =
          jsonCars.getJSONObject(i).getJSONArray("address");
      status.setAddress(jsonAddress.get(0) + ", " +
          jsonAddress.get(1));
      status.setLatitude(jsonCars.getJSONObject(i)
          .getDouble("latitude"));
      status.setLongitude(jsonCars.getJSONObject(i)
          .getDouble("longitude"));
      car.setStatus(status);

      cars.add(car);
    }

    return cars;
  }
}
```

OperatorBDecoder.java

```java
import java.io.IOException;
import java.util.ArrayList;

import org.json.JSONArray;
import org.json.JSONObject;
```

```java

/**
 * The class that encapsulates methods to decode raw API data from
   operator A to Java objects.
 * @author Philipp Dressler
 */
public class OperatorBDecoder {

  private JSONObject data;

  /**
   * Initializes the decoder object with a given data String.
   * @param data The raw car data to decode.
   * @throws IOException
   */
  public OperatorBDecoder(String data) throws IOException {
    this.data = new JSONObject(data);
  }

  /**
   * Decodes the raw car data to Car and CarStatus objects
     accordingly to the json file structur of the API of operator
     B..
   * @return The list of cars decoded from the car data String.
   */
  public ArrayList<Car> getCars() {

    ArrayList<Car> cars = new ArrayList<Car>();

    JSONArray jsonCars = data.getJSONArray("placemarks");
    for (int i = 0; i < jsonCars.length(); i++) {

      Car car = new Car();
      CarStatus status = new CarStatus();

      car.setOperator("Operator B");
      car.setLicensePlate(jsonCars.getJSONObject(i)
          .getString("name"));
      car.setModel("Smart");
      status.setFuelLevel(jsonCars.getJSONObject(i).getInt("fuel"));
      status.setInnerCleanliness(jsonCars.getJSONObject(i)
          .getString("interior"));
      status.setAddress(jsonCars.getJSONObject(i)
          .getString("address"));
      JSONArray jsonCoordinates =
        jsonCars.getJSONObject(i).getJSONArray("coordinates");
      status.setLatitude(Double.parseDouble(jsonCoordinates
          .get(1).toString()));
      status.setLongitude(Double.parseDouble(jsonCoordinates
            .get(0).toString()));
        car.setStatus(status);

        cars.add(car);
      }

      return cars;
    }
}
```

CarsharingDbConnection.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.sql.Types;
import java.util.ArrayList;

/**
 * Represents the class encapsulating all methods to load the car
    sharing trip database with given car data.
 * @author Philipp Dressler
 */
public class CarsharingDbConnection {

  private Connection con;
  private ResultSet res;

  private PreparedStatement stmtCreateCar;
  private PreparedStatement stmtCreateCarHistory;
  private PreparedStatement stmtUpdateCarHistory;
  private PreparedStatement stmtCreateImportedData;

  private PreparedStatement stmtSelectExistsCar;
  private PreparedStatement stmtSelectIsCarParked;
  private PreparedStatement stmtSelectChangedLocation;
  private PreparedStatement stmtSelectgetParkedCars;

  /**
   * Initializes the database connection.
   * @param url The connection string for the database connection.
   * @param usr The user name to connect with.
   * @param pwd The password to use for the connection.
   * @throws SQLException
   */
  public CarsharingDbConnection(String url, String usr, String pwd)
     throws SQLException {

    System.out.println("Establishing connection to "+url+"...");

    con = DriverManager.getConnection(url, usr, pwd);

    //prepare parameterized SQL statements
    stmtCreateCar = con.prepareStatement("insert into car
       (licenseplate, operator, model) values (?, ?, ?)");
    stmtCreateCarHistory = con.prepareStatement("insert into
       carhistory (licenseplate, fuellevel, innercleanliness,
       address, arrival, departure, location) values
       (?,?,?,?,?,?,ST_SetSRID(ST_MakePoint(?,?),4326))");
    stmtUpdateCarHistory = con.prepareStatement("update carhistory
       set departure=? where licenseplate=? and arrival >= all
       (select arrival from carhistory where licenseplate=?)");
    stmtCreateImportedData = con.prepareStatement("insert into
       importeddata (operator, timestamp_file) values (?, ?)");

    stmtSelectExistsCar = con.prepareStatement("select count(*) from
       car where licenseplate=?");
    stmtSelectIsCarParked = con.prepareStatement("select count(*)
       from carhistory where licenseplate=? and departure is null
       and arrival >= all (select arrival from carhistory where
       licenseplate=?)");
    stmtSelectChangedLocation = con.prepareStatement("select address
       from carhistory where licenseplate=? and arrival >= all
       (select arrival from carhistory where licenseplate=?)");
    stmtSelectgetParkedCars = con.prepareStatement("select
       ch.licenseplate from carhistory ch, car c where ch.departure
       is null and c.licenseplate=ch.licenseplate and c.operator=?
       and ch.arrival >= all (select ch2.arrival from carhistory
       ch2 where ch2.licenseplate=ch.licenseplate)");

    System.out.println("Connected!");
  }

  /**
   * Closes the database connection and all attached resources.
   */
  public void closeConnection() {
    try {
      if (res != null) {
        res.close();
      }
        if (stmtCreateCar != null) {
          stmtCreateCar.close();
          }
          if (stmtCreateCarHistory != null) {
            stmtCreateCarHistory.close();
          }
          if (stmtUpdateCarHistory != null) {
            stmtUpdateCarHistory.close();
          }
          if (stmtCreateImportedData != null) {
            stmtCreateImportedData.close();
          }
          if (stmtSelectExistsCar != null) {
            stmtSelectExistsCar.close();
          }
          if (stmtSelectIsCarParked != null) {
            stmtSelectIsCarParked.close();
          }
          if (stmtSelectChangedLocation != null) {
            stmtSelectChangedLocation.close();
          }
          if (stmtSelectgetParkedCars != null) {
            stmtSelectgetParkedCars.close();
          }
          if (con != null) {
             con.close();
          }
      } catch (SQLException ex) {
          ex.printStackTrace();
      }
  }

  /**
   * Inserts a new car into the database.
   * @param car The car object to add to database.
   * @throws SQLException
   */
  public void createCar(Car car) throws SQLException {
    stmtCreateCar.setString(1, car.getLicensePlate());
    stmtCreateCar.setString(2, car.getOperator());
    stmtCreateCar.setString(3, car.getModel());

    int affected = stmtCreateCar.executeUpdate();
    if(affected != 1) {
      throw new SQLException("Problem on inserting new car. Rows
          affected: "+affected);
    }
  }

  /**
```

```java
 * Adds a new car status to the database. This represents the dropoff action in the car sharing system.
 * During the inserting operation, also the longitude and latitude properties of the car status are transformed into PostGIS points with SRID 4326.
 * @param car Car object to perform dropoff action for.
 * @param arrival Timestamp of dropoff operation.
 * @throws SQLException
 */
public void carDropoff(Car car, Timestamp arrival) throws SQLException {
  stmtCreateCarHistory.setString(1, car.getLicensePlate());
  stmtCreateCarHistory.setInt(2, car.getStatus().getFuelLevel());
  stmtCreateCarHistory.setString(3, car.getStatus().getInnerCleanliness());
  stmtCreateCarHistory.setString(4, car.getStatus().getAddress());
  stmtCreateCarHistory.setTimestamp(5, arrival);
  stmtCreateCarHistory.setNull(6, Types.TIMESTAMP);
  stmtCreateCarHistory.setDouble(7, car.getStatus().getLongitude());
  stmtCreateCarHistory.setDouble(8, car.getStatus().getLatitude());

  int affected = stmtCreateCarHistory.executeUpdate();
  if(affected != 1) {
    throw new SQLException("Problem on inserting carhistory. Rows affected: "+affected);
  }
}

/**
 * Sets the departure time stamp of the latest status entry of a car. This represents a pickup action in the car sharing system.
 * @param licensePlate The license plate of the car.
 * @param departure The departure time stamp to set on the database entry.
 * @throws SQLException
 */
public void carPickup(String licensePlate, Timestamp departure) throws SQLException{
  stmtUpdateCarHistory.setTimestamp(1, departure);
  stmtUpdateCarHistory.setString(2, licensePlate);
  stmtUpdateCarHistory.setString(3, licensePlate);

  int affected = stmtUpdateCarHistory.executeUpdate();
  if(affected != 1) {
    throw new SQLException("Problem on updating carhistory. Rows affected: "+affected);
  }
}
```

```

  /**
   * Create an import history entry for logging purposes.
   * @param operator The operator to log the import for.
   * @param fileTimestamp The time stamp of the imported data set.
   * @throws SQLException
   */
  public void createImportHistory(String operator, Timestamp
      fileTimestamp) throws SQLException {
    stmtCreateImportedData.setString(1, operator);
    stmtCreateImportedData.setTimestamp(2, fileTimestamp);

    int affected = stmtCreateImportedData.executeUpdate();
    if(affected != 1) {
      throw new SQLException("Problem on creating import history.
          Rows affected: "+affected);
    }
  }

  /**
   * Determines the existence of a car with a given license plate in
       the database.
   * @param licensePlate The license plate to check the existence
       for.
   * @return True if car exists in database, false otherwise.
   * @throws SQLException
   */
  public boolean existsCar(String licensePlate) throws SQLException {
    stmtSelectExistsCar.setString(1, licensePlate);

    res = stmtSelectExistsCar.executeQuery();
    while(res.next()){
      if(res.getInt(1) > 0) {
        return true;
      }
    }
    return false;
  }

  /**
   * Determines the trip status of a car in the database.
   * @param car The car to check trip status for.
   * @return True if the car is currently parked, false otherwise.
   * @throws SQLException
   */
  public boolean isCarParked(Car car) throws SQLException {
    stmtSelectIsCarParked.setString(1, car.getLicensePlate());
    stmtSelectIsCarParked.setString(2, car.getLicensePlate());

    res = stmtSelectIsCarParked.executeQuery();
    while(res.next()){
      if(res.getInt(1) > 0) {
        return true;
      }
    }
    return false;
  }

  /**
   * Compares the location of a given car to the one of the same car
      in the database.
   * @param car The car object to do the location check for.
   * @return True if the location is different, false otherwise.
   * @throws SQLException
   */
  public boolean changedLocation(Car car) throws SQLException {
    stmtSelectChangedLocation.setString(1, car.getLicensePlate());
    stmtSelectChangedLocation.setString(2, car.getLicensePlate());

    res = stmtSelectChangedLocation.executeQuery();
    while(res.next()){
      if(!res.getString(1).equals(car.getStatus().getAddress())) {
        return true;
      }
    }
    return false;
  }

  /**
   * Retrieves a list of all the currently parked cars of a given
      operator in the database.
   * @param operator The operator do retrieve the list for.
   * @return The list of all the currently parked cars.
   * @throws SQLException
   */
  public ArrayList<String> getParkedCars(String operator) throws
     SQLException {
    stmtSelectgetParkedCars.setString(1, operator);

    res = stmtSelectgetParkedCars.executeQuery();

    ArrayList<String> licensePlates = new ArrayList<String>();
    while(res.next()){
      licensePlates.add(res.getString(1));
    }
    return licensePlates;
  }
}
```

Bibliography

[1] Andreas Braun, Volker Hochschild, and Andreas Koch. Intraregionale Unterschiede in der Carsharing-Nachfrage - eine GIS-basierte empirische Analyse. IAW Discussion Papers 99, Institut für Angewandte Wirtschaftsforschung (IAW), 2013.

[2] Leonardo Caggiani, Michele Ottomanelli, Rosalia Camporeale, and Mario Binetti. Spatio-temporal clustering and forecasting method for free-floating bike sharing systems. In Jerzy Świątek and Jakub M. Tomczak, editors, *Advances in Systems Science: Proceedings of the International Conference on Systems Science 2016 (ICSS 2016)*, pages 244–254, Cham, 2017. Springer International Publishing.

[3] Guangyu Cao, Lei Wang, Yong Jin, Jie Yu, Wanjing Ma, Qi Liu, Aiping He, and Tao Fu. Determination of the vehicle relocation triggering threshold in electric car-sharing system. In Yingmin Jia, Junping Du, Weicun Zhang, and Hongbo Li, editors, *Proceedings of 2016 Chinese Intelligent Systems Conference: Volume I*, pages 11–22, Singapore, 2016. Springer Singapore.

[4] car2go API. https://github.com/car2go/openapi. Accessed: 2017-12-21.

[5] Elvezia Cepolina and Alessandro Farina. Urban Car Sharing: An Overview Of Relocation Strategies. In J.W.S. Longhurst, editor, *WIT Transactions on The Built Environment*, volume 128, pages 419–431, 2012.

[6] Han-wen Chang, Yu-chin Tai, and Jane Yung-jen Hsu. Context-aware taxi demand hotspots prediction. *International Journal of Business Intelligence and Data Mining*, 5(1):3–18, December 2010.

[7] Chrome, Google LLC. https://www.google.at/chrome/. Accessed: 2018-01-30.

[8] Citybike Wien. https://www.citybikewien.at/de/. Accessed: 2017-12-19.

[9] Department of Building Physics and Building Ecology, Vienna UT. http://www.bpi.tuwien.ac.at/. Accessed: 2018-01-19.

[10] DriveNow Austria GmbH. https://www.drive-now.com/. Accessed: 2018-01-22.

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.

[12] Free2Move, GHM Mobile Development GmbH. https://www.free2move.com. Accessed: 2018-01-29.

[13] Sascha Herrmann, Frederik Schulte, and Stefan Voß. Increasing Acceptance of Free-Floating Car Sharing Systems Using Smart Relocation Strategies: A Survey Based Study of car2go Hamburg. In Rosa G. González-Ramírez, Frederik Schulte, Stefan Voß, and Jose A. Ceroni Díaz, editors, *Computational Logistics: 5th International Conference, ICCL 2014, Valparaiso, Chile, September 24-26, 2014. Proceedings*, pages 151–162. Springer International Publishing, Cham, 2014.

[14] Johanna Kopp, Regine Gerike, and Kay W. Axhausen. Do sharing people behave differently? An empirical evaluation of the distinctive mobility patterns of free-floating car-sharing members. *Transportation*, 42(3):449–469, May 2015.

[15] Katherine Kortum and Randy Machemehl. Free-floating carsharing systems: innovations in membership prediction, mode share, and vehicle allocation optimization methodologies. *Center for Transportation Research, University of Texas at Austin*, 2012.

[16] Magistrat der Stadt Wien. https://open.wien.gv.at/site/open-data/. Accessed: 2018-02-27.

[17] MAPS.ME. https://maps.me/download/. Accessed: 2018-02-01.

[18] Catherine Morency, Martin Trepanier, and Bruno Agard. Typology of carsharing members. In *Transportation Research Board 90th Annual Meeting, Washington, D.C.*, January 2011.

[19] oBike. https://www.o.bike/at/. Accessed: 2017-12-19.

[20] OFO AUT GmbH. http://www.ofo.com. Accessed: 2017-12-19.

[21] OpenStreetMap. https://www.openstreetmap.org/. Accessed: 2018-01-19.

[22] OpenStreetMap API. https://wiki.openstreetmap.org/wiki/api_v0.6. Accessed: 2018-02-01.

[23] OpenStreetMap Feature List. https://wiki.openstreetmap.org/wiki/map_features. Accessed: 2018-02-22.

[24] Stefan Paschke, Milos Balać, and Francesco Ciari. Implementation of vehicle relocation for carsharing services in the multi-agent transport simulation matsim. In *Arbeitsberichte Verkehrs- und Raumplanung*, volume 1188. IVT, ETH Zürich, Zürich, August 2016.

[25] PostGIS. http://postgis.net/. Accessed: 2018-01-19.

[26] PostGIS Documentation. http://postgis.net/docs/st_clusterdbscan.html. Accessed: 2018-02-20.

[27] PostgreSQL. http://www.postgresql.org/. Accessed: 2018-01-19.

[28] QGIS. http://www.qgis.org/de/site/index.html. Accessed: 2018-01-19.

[29] QGIS OpenLayers Plugin. https://plugins.qgis.org/plugins/openlayers_plugin/. Accessed: 2018-02-20.

[30] Frederik Schulte and Stefan Voß. Decision support for environmental-friendly vehicle relocations in free- floating car sharing systems: The case of car2go. *Procedia CIRP*, 30:275–280, December 2015.

[31] Dong-Ping Song and Jing-Xin Dong. Effectiveness of an empty container repositioning policy with flexible destination ports. In *Transport Policy*, volume 18, pages 92–101. January 2011.

[32] Robert Stahlbock and Stefan Voss. Improving empty container logistics – can it avoid a collapse in container transportation? In L. Kroon, T. Li, and R. Zuidwijk, editors, *Liber Amicorum In Memoriam Jo Van Nunen*, pages 217–224. Rotterdam School of Management, Erasmus University, January 2010.

[33] Statistik Austria. http://www.statistik.at/web_de/statistiken/index.html. Accessed: 2018-02-27.

[34] Patrick Vogel and Dirk C. Mattfeld. Strategic and operational planning of bike-sharing systems by data mining – a case study. In *Computational Logistics*, pages 127–141. Springer, 2011.

[35] Simone Weikl and Klaus Bogenberger. An Integrated Relocation Model for Free-Floating Carsharing Systems: Field Trial Results. *Transportation Research Record: Journal of the Transportation Research Board*, 2536:19–27, 2015.

[36] Simone Weikl, Klaus Bogenberger, and Nikolas Geroliminis. A Simulation Framework for Proactive Relocation Strategies in Free-Floating Carsharing Systems. *Submitted for presentation at 95th Annual Meeting of the Transportation Research Board (TRB), Washington, D.C.*, January 2016.

[37] WienMobil, Wiener Linien GmbH & Co KG. https://www.wienerlinien.at/eportal3/ep/channelview.do/pagetypeid/66526/channelid/-3600060. Accessed: 2018-01-29.

[38] Wikimedia Österreich. http://data.opendataportal.at/dataset. Accessed: 2018-02-27.