Vienna University of Economics and Business

Master Thesis

Predicting Missing Values in the Context of an Open City Data Pipeline

Author:
Christoph Martin

Supervisor:
Univ.-Prof. Dr. Axel Polleres

January 2015

Abstract

The Open City Data Pipeline is a project initiated by Siemens AG in Vienna. Its goal is to collect data on cities from several publicly available sources and integrate them into a single data storage. In order to make this data easily accessible, extensible and machine-processable the data is stored in an RDF database. The data structure follows an extensible city data ontology which was created specifically for this data pipeline. A key challenge of the pipeline is missing information. Machine learning techniques can be applied to find latent structure in the data and infer missing information from actual observations. Principal Component Analysis is used in this work to reduce the number of dimensions. Three wide-spread regression methods are then applied to predict missing values. Standard prediction error measures are used to determine the accuracy of the algorithms. The results suggest that Principal Component Regression is a suitable method to fill in missing values and improve the usefulness of the Open City Data Pipeline to stakeholders such as governments, citizens and infrastructure providers.

To my parents

Acknowledgements

I would like to thank the people how helped my in writing this thesis:

First and foremost I would like to thank my supervisor Dr. Axel Polleres, who guided me through the entire process and continuously contributed valuable input. Also I would like to express my gratitude towards Siemens, the company which made this work possible. In particular, I would like to thank Herwig Schreiner, Stefan Bischof and Bettina Länger for their support and advice. Finally, I want to thank Patrik Schneider who offered valuable input for this thesis and who will continue to work on the Open City Data Pipeline after me.

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
CDP	Carbon Disclosure Project
CSV	Comma-Separated Values
GCIF	Global City Indicators Facility
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ISO	International Organization for Standardization
KNN	K-Nearest Neighbour
LOD	Linked Open Data
MAR	Missing At Random
MCAR	Missing Completely At Random
MNAR	Missing Not At Random
MSE	Mean Squared Error
OLS	Ordinary Least Squares
OWL	Web Ontology Language
PCA	Principal Component Analysis
PCR	Principal Component Regression
RDF	Resource Description Framework
RDFS	RDF Schema
RMSE	Root Mean Squared Error
RMSE%	Normalized Root Mean Squared Error
SKOS	Simple Knowledge Organization System
SPARQL	SPARQL Protocol and RDF Query Language
TDB	Triple Database
Turtle	Terse RDF Triple Language
UN	United Nations
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language

Chapter 1

Introduction

Cities are becoming an increasingly important anthroposphere for a growing number of people. Urbanization increased from under 30% in 1950 to 54% percent in 2014. This means that half of the entire population is now living in cities and this trend will continue in the future [56]. A variety of entities, for example governments, citizens and companies such as infrastructure providers, need to understand the status and performance of cities to act accordingly. Data in general and Open Data in particular can support decision making by providing greater insights into the operations of cities [45].

Numerous cities around the world have recognized this and it is one of the reasons for cities to publish Open Data (see for example [2], [3]). However, there are other reasons as well, e.g., transparency and legal requirements. While local Open Data initiatives often provide exhaustive and detailed data on a single city, comparing different cities is a tedious task. It requires the integration of data from different sources which is challenging due to several reasons, e.g. different data formats, measurement methodologies, time granularities, units and so on.

The Open City Data Pipeline is a project initiated by Siemens AG in Vienna. The pipeline is "[...] a system for gathering city performance indicators published as Open Data in order to ease the compilation of studies and reports" [43, p. 1]. Its goal is to collect data on cities from several different sources and integrate them into a single open data storage for others to use. The data covers heterogeneous areas like demography, economic aspects, culture, social and environmental aspects. To simplify usage and processing of the data by third parties, we are republishing the data as RDF Data (see Section 2.1.2). Adherence to the Linked Data principles set up by Tim Berners-Lee (see Section 2.1.1) should further facilitate the usage of the collected data.

Besides data integration difficulties there is another problem with the collected data: missing values. Missing data is a problem in a lot of domains and data on cities is no exception. Even large data collection efforts such as the Urban Audit data collections by Eurostat [40] cannot circumvent a certain ratio of missing values in their data sets. The issue is also exacerbated by combining and integrating data from different sources dealing with disjoint indicators for different cities.

The main goal of the pipeline is to gain a better understanding of cities in order to analyses shortcomings and support decision making. Imputing (i.e., predicting) the missing values based on the observed values (i.e. developing a prediction model) is therefore desirable for improving the scope and quality of the decision making. The usefulness of the model should be growing with its completeness. The two main objectives which need to be considered when predicting missing values are the amount of possible predictions (how many of the missing values can be predicted?) and the accuracy of the predictions. The best case scenario for the prediction effort would be an accurate prediction of all missing values in the collected data. The present thesis aims to provide methods and results on the prediction of missing values in this environment. It covers the data collection and integration architecture used to obtain the data, all steps which need to be taken before a prediction can be made (i.e. data export, preprocessing, cleansing, feature and observation selection), two approaches to prediction and the achieved results.

The remainder of the thesis is structured as follows:

In chapter 2 we review the technologies and concepts used in the course of this work. First, we give a quick introduction into Linked Open Data and its associated standards like RDF and SPARQL. Second, we briefly review data mining and introduce three common regression methods (k-nearest neighbour, linear regression, and decision trees) using some example data.

Chapter 3 answers the questions "Where does the data come from? How was it collected?"; it gives more insights into the context of the work by describing the Open City Data Pipeline in more detail. We describe the general architecture, the individual components and their relations and we present the data sources. Furthermore some information on the city data ontology is given.

Chapter 4 answers the question "How does the dataset look like?". It describes the integrated dataset which is subsequently used in Chapter 5. We delineate the process of extracting the data from the database and getting it into a suitable form for further analysis (i.e. preprocessing) and we give some additional information on the dataset characteristics.

Chapter 5 answers the question ”How can missing values be predicted?”. It describes different approaches to the problem of approximating missing values in the dataset. Principal Component Analysis (PCA) as a method of dimensionality reduction is introduced and integrated into a prediction methodology using Principal Component Regression (PCR). We present the results of the different approaches and we compare them.

Chapter 6 sums the work up, links it to ongoing related work and gives some hints to possible future work to further improve the Open City Data Pipeline in general and the prediction performance of missing values in particular.

Chapter 2

Preliminaries

In this chapter we introduce some general concepts which are used in the subsequent chapters of the work. We discuss the Linked Data principles, we explain what Linked Open Data is and why we are using it in the Open City Data Pipeline. We present RDF as a way of describing real word resources. The associated query language SPARQL is quickly introduced as well. We are also going to introduce some other city indicator frameworks from which our own city data ontology was partly derived. After a general introduction to Data Mining, we present three common regression methods to predict missing values: k-nearest neighbour, linear regression and decision trees. Finally, we describe Principal Component Analysis (PCA) as a tool for dimensionality reduction.

2.1 Linked Open Data

Data in general can support decision making by providing greater insights into the matter at hand. Data is called *Open Data* if it has three main properties [42]:

- **Availability and Access**
  The data has to be easily available in a conveniently accessible format.
- **Reuse and Redistribution**
  The usage terms have to allow for reuse and redistribution including integration with other data.
- **Universal Participation**
  The data has to be available to everyone regardless of person, organization and application.

The Open Definition project defines knowledge as open "[...] if anyone is free to access, use, modify, and share it — subject, at most, to measures that preserve provenance and openness" [51]. In the Open City Data Pipeline we focus on collecting "Open Data".

Linked Data denotes a set of best practices for publishing structured data on the Web. The main advantage of structured data over unstructured data is that it is readily machine readable and therefore automatically processable [5] .

Due to its limited restrictions on usage, Open Data is a natural candidate for publication as *Linked Open Data.* A continuously increasing number of interlinked datasets is published as Linked Open Data (LOD) on the web [47].

There are multiple reasons to use Linked Data in the Open City Data Pipeline. First, some of the data which is collected and integrated is already published in the form of Linked Data (e.g. data from DBpedia [4]) which enables easier integration because less data transformation effort is needed.

Second, we are republishing the collected and integrated data as RDF Data for easy access and reuse by third parties who are also interested in the domain of cities. The data can be processed by machines instead of humans. The Open City Data Pipeline is therefore a consumer as well as a publisher of Linked Open Data.

Third, the Linked Data approach offers the advantage of easy extensibility in the future. Possible extensions in addition to adding more data on city level, could include other spatial levels such as districts or countries as well. The semantic connection of these spatial levels (i.e. their hierarchical order) is represented in our ontology and could be exploited in complex queries, e.g. by adding geospatial query support [28].

2.1.1 Linked Data Principles

The following *Linked Data principles* were suggested by Tim Berners-Lee in 2006 with the goal of sharing structured data globally [7].

- Use URIs (Uniform Resource Identifier) as names for things
- Use HTTP URIs for easier lookup
- Use standards like RDF (Resource Description Framework) and SPARQL (SPARQL Protocol and RDF Query Language) to provide useful information to users
- Link to other URIs to foster discovery of additional information [7]

Along with these principles Tim Berners-Lee also introduced a 5 star rating system where a dataset receives 5 stars if the data is made available using open standards and if there are links to other people's data [7]. The exact criteria for the number of stars are listed below. The criteria are cumulative, i.e., a dataset has to fulfill all criteria to be considered a 5 star dataset.

- **1 Star**
  Data is available on the web with an open license regardless of the data format.
- **2 Stars**
  Data is available in a machine-readable format.
- **3 Stars**
  Data is available in a non-proprietary format (e.g. comma-separated value file instead of Excel file).
- **4 Stars**
  Data uses W3C open standards such as RDF and SPARQL to identify resources.
- **5 Stars**
  Data uses links to other datasets to provide context.

Data from the Open City Data Pipeline is made available using open standards and also links to other data sources. Furthermore it takes advantage of externally established vocabularies while also introducing an ontology which aims at integrating these vocabularies (see Section 3.3).

For example, in conformance with the Linked Data principles, HTTP URIs from dbpedia.org are used to identify cities in the pipeline, e.g.
`http://dbpedia.org/resource/Vienna` for Vienna. DBpedia in turn contains much more links from cities to other resources such as images, persons, organizations and so on. The usage of DBpedia URIs enables users to not only access the data made available directly in the pipeline (which is in general limited to numerical data), but also to obtain additional information and links by following the URI. Data collected from other sources than DBpedia are always mapped to these city URIs to further facilitate the discovery of more information on the cities.

2.1.2 Resource Description Framework (RDF)

The *Resource Description Framework* (RDF) provides a way to describe all kinds of resources such as real word objects, documents on the Web and also abstract concepts like a city. In 2004, RDF became a W3C recommendation [33]. RDF 1.1 became a W3C recommendation in 2014 [34]. Information in RDF is represented in the form of a triple. Each triple is comprised of a subject, a predicate and an object. Figure 2.1 shows a simple RDF statement in form of a triple defining a city.

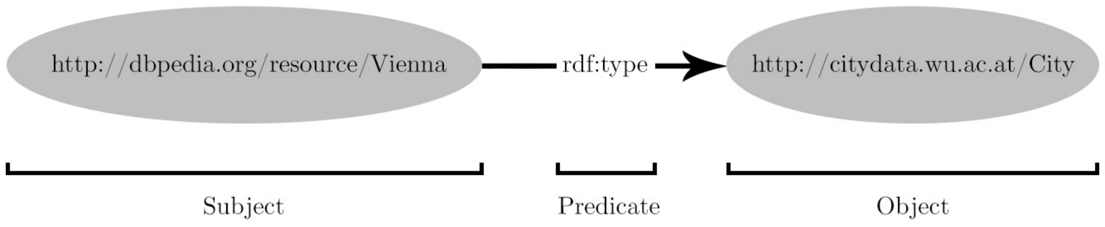

http://dbpedia.org/resource/Vienna
rdf:type
http://citydata.wu.ac.at/City
Subject
Predicate
Object

FIGURE 2.1: Example RDF Triple

URIs are used to identify entities at each position of a triple. In the example the URI `http://dbpedia.org/resource/Vienna` is in the subject position. This is a URI from DBpedia identifying Vienna, the capital of Austria, which is described by this RDF statement. The prefixed name `rdf:type` is in the predicate position. The part before the colon is the prefix, the part after the colon is the resource name. A prefix resolves to a specified IRI which needs to be explicitly defined. The prefixed name in the example resolves to the IRI `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`. This property is used to describe that something is an instance of a class. The class `http://citydata.wu.ac.at/City` follows in the object position. The triple says that Vienna is an instance of the class City. Typically more than one triple is necessary to sufficiently describe a resource. Multiple triples are combined in an RDF graph.

Several serialization formats for RDF exist (e.g. N-Triples, RDFa, RDF/XML). They are all suitable to express RDF statements but differ in their format. We use Turtle (Terse RDF Triple Language) [6] in the Open City Data Pipeline.

Turtle is an easy-to-read, human-friendly concrete syntax for RDF used for textual representations of RDF graphs. Subject, predicate and object are simply separated by a whitespace. Triples are terminated with a period symbol. Relative and absolute IRIs are enclosed in `<` and `>`.

Listing 2.1 describes the triple presented in Figure 2.1 as well as a German and an English label for the city using the Turtle serialization.

```
@prefix wu: <http://citydata.wu.ac.at/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<http://dbpedia.org/resource/Vienna> a wu:City ;
  rdfs:label  "Vienna"@en ;
  rdfs:label  "Wien"@de .
```

Listing 2.1: RDF Triples in Turtle serialization (example.rdf)

In this example the namespace prefix labels `wu:` and `rdfs:` are defined in the first two lines. Prefix labels are defined once at the beginning of an RDF document and can then be used to form prefixed names in triples which leads to a more compact data representation and also improves legibility by keeping the notation short. Prefixed names consist of a prefix label (e.g. `wu`), a separating colon and a local part (e.g. `City`). Together these three elements resolve to a IRI, in this case `wu:City` resolves to `http://citydata.wu.ac.at/City`.

Multiple predicate-object pairs with the same subject can be repeated in Turtle separated by a semicolon. In the example the subject for all three triples is `http://dbpedia.org/resource/Vienna`, therefore they are separated with a semicolon. This abbreviated form also - like prefixes - leads to a more compact data representation.

The second and third triple in Listing 2.1 have W3C's `rdfs:label` in the predicate position which is used to define human-readable names. Literals with languages tags as suffixes (`@en` for English and `@de` for German) are used in the object position of these triples. Literals are values of a certain type, e.g. strings, numbers or dates. In our example strings are used to assign human-readable names to the city of Vienna. Besides URIs and literals there are also blank nodes in RDF. In general, subjects or objects can be represented using blank nodes which are unnamed or anonymous nodes in an RDF graph without a URI which could identify them [34].

RDF Schema (RDFS)

The prefix `rdfs:` is used in Listing 2.1. *RDF Schema* (RDFS) is an extension to RDF adding semantic elements to the framework. RDFS enables the creation of ontologies which is not feasible with RDF alone. An "[...] ontology typically contains a hierarchy of concepts within a domain and describes each concept's crucial properties through an attribute-value mechanism." [15, p. 64]

RDF Schema was initially published in 1998 and is now a W3C recommendation [10]. It provides a set of classes and properties which are essential to model the relations and constraints between resources. Class names are starting with an uppercase letter by convention, whereas properties are starting with a lowercase letter.

Some of the most important RDFS classes are:

- **rdfs:Resource:** everything described in RDF is a resource.
- **rdfs:Class:** defines a resource as a class for other resources, e.g. `wu:City` in Listing 2.1.
- **rdfs:Literal:** literals (e.g. strings, numbers) are instances of this class.
- **rdf:Property:** the class of properties.

RDFS properties describe the relation between a subject resource and an object resource. They are instances of the class `rdf:Property`. Some of the most important RDFS properties which are also used in the Open City Data Pipeline are:

- **rdfs:domain:** a resource having a given property must be an instance of the class referenced.
- **rdfs:range:** defines that the values of a given property are instances of the class referenced. We use this in the Open City Data Pipeline to define the possible values of indicators.
- **rdfs:subPropertyOf:** used to define hierarchical relations between properties. We use this in the Open City Data Pipeline to map data source specific indicators to general indicators modelled in the city data ontology (see Section 3.3).
- **rdfs:subClassOf:** used to define hierarchical relations between classes.

Web Ontology Language (OWL)

Although RDFS extends the expressibility of RDF and enables the creation of ontologies including hierarchical relations it is still limited in its usefulness, especially in complex domains. It is not possible to express more sophisticated relations with RDFS, e.g. cardinality, disjointness or equality.

The *Web Ontology Language* (OWL) introduced in 2002 does support these missing features. It has become a W3C recommendation in 2004 [37]. The languages and standards in the Linked Data area are constantly evolving and developing, so in 2012 an extension to OWL, OWL 2, also became a W3C recommendation [59].

OWL is suitable to craft complex, sophisticated ontologies. It is not a single language but a family of formal ontology languages recommended by W3C. Due to the varying requirements of ontologies, three different languages were created: OWL Lite, OWL DL and OWL Full. These are smaller subsets enabling ontology creators to pick the complexity level which suits them best [37].

Some OWL properties used in the Open City Data Pipeline are the following:

- **owl:DatatypeProperty:** used to express relations between instances of classes and RDF literals or XML schema datatypes.
- **owl:ObjectProperty:** used to express relations between instances of two classes
- **owl:sameAs:** used to express the similarity between two instances, e.g. DBpedia uses this property to map redirected URIs to their respective correct URIs.

2.1.3 SPARQL Protocol and RDF Query Language

SPARQL (recursive acronym for SPARQL Protocol and RDF Query Language) is a graph-matching query language for RDF. It enables querying as well as manipulating RDF graph content. Filtering, ordering and powerful built-in functions make SPARQL a rich tool in efficiently handling RDF data [52].

SPARQL has four query forms for different purposes: `SELECT`, `CONSTRUCT`, `ASK` and `DESCRIBE`:

- `SELECT`
  Returns raw values - comparable to an SQL SELECT statement.
- `CONSTRUCT`
  Returns an RDF graph which is defined using triple templates
- `ASK`
  Returns a boolean value indicating whether the given pattern matches
- `DESCRIBE`
  Returns a description of the resources matching the given pattern

A simple SPARQL SELECT query is given in Listing 2.2. The query selects the labels of instances of the class `wu:City` from `example.rdf` (see Listing 2.1).

```
PREFIX wu: <http://citydata.wu.ac.at/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?label
FROM <example.rdf>
WHERE {
  ?city wu:City ;
          rdfs:label ?label .
}
```

Listing 2.2: A simple SPARQL query

The `SELECT` part of the query defines which variables of the result set should be returned. Variables are indicated by a `?` or `$` prefix. In this case only the variable `?label` is returned. The `WHERE` part of the query contains the graph pattern which is going to be matched against the data graph defined in the file `example.rdf`.

Multiple triple patterns can be specified in the `WHERE` clause. If they have the same subject they can be separated with a semicolon just like in the Turtle RDF serialization. Prefixes can be used in SPARQL as well. Prefixed names follow the syntax `prefix:name` and can then be used in the SPARQL query. In the example the prefixes `wu:` and `rdfs:` are used to shorten the query and to make it more legible. The pattern provided in the example is very simple. It contains only two triple patterns and the variables `?city` and `?label`. The bindings for the variable `?label` will be returned.

Our query looks for the labels for resources which are an instance of the class `wu:City`. The two results will be `Vienna` and `Wien`.

2.2 City Indicator Frameworks

An ontology was created specifically for the Open City Data Pipeline describing key concepts of the city domain such as spatial contexts (countries, cities, ...), indicator categories, indicators etc. This City Data Ontology (described in Section 3.3) is derived from other, already existing city indicator frameworks, which we briefly describe here.

2.2.1 Urban Audit

The Urban Audit data collections conducted by the national statistical institutes, the Directorate-General for Regional and Urban Policy (DG REGIO) and Eurostat are trying to assess the quality of life in European cities. The effort started in 1998 with a pilot phase. In September 2000 it was decided that the Urban Audit is useful and should be continued in the future. Currently, data collection takes place every three years with an annual data collection planned for a smaller number of indicators [40].

Urban Audit aims at providing an extensive look at the cities under investigation because its a policy tool to the European Commission and other authorities of the European Union (data is available for the EU nations, Norway, Switzerland and Turkey). "The projects' ultimate goal is to contribute towards the improvement of the quality of urban life" [17]. It is one of the most important data sources for the Open City Data Pipeline (see Section 3.2).

At the city level, the Urban Audit contains around 250 indicators. In the initial pilot phase there were over 500 but this number was significantly reduced. The collected indicators were divided into the following categories:

- Demography
- Social Aspects
- Economic Aspects
- Civic involvement
- Training and Education
- Environment
- Travel and Transport
- Information Society
- Culture and Recreation
- Perception Indicators

These categories are mostly the same as the indicator categories used in the Open City Data Pipeline (see Section 3.3). Urban Audit further divides each of the categories into subcategories, e.g. Demography is divided into Population, Nationality and Household Structure. The Perception Indicators are collected in opinion polls among a representative random sample of inhabitants of the cities.

The survey is conducted every three years (the last survey was in November 2012) whereas the other data is now available via the regular Eurostat website (http://ec.europa.eu/eurostat). The Urban Audit datasets all have identifiers starting with `urb_` and can be obtained from the regular online database of Eurostat.

All data is provided on a voluntary basis only which leads to varying data availability and missing values in the collected datasets. The present work aims at addressing this problem.

2.2.2 Global City Indicators Facility (GCIF)

The Global City Indicators Facility (GCIF) uses a standardized methodology to provide a set of city indicators with the aim of global comparability of city performance and knowledge sharing. GCIF is a program of the Global Cities Institute at the University of Toronto [19].

The indicators are organized around 20 ”themes” which are divided into two categories: city services and quality of life [20]. Currently the following themes are used:

City Services

- Education
- Finance
- Recreation
- Governance
- Energy
- Transportation
- Wastewater
- Fire and Emergency Response
- Health
- Safety
- Solid Waste
- Urban Planning
- Water

Quality of Life

- Civic Engagement
- Economy
- Shelter
- Culture
- Environment
- Social Equity
- Technology

Overall there are 115 indicators collected. The data is made available online via a relational database. However, only cities which become members and report their own data are gaining access to the database. Thus, the Open City Data Pipeline does currently not collect data from GCIF because it is not Open Data.

GCIF also made a standardization effort resulting in the ISO 37120 (International Organization for Standardization) standard for city services and quality of life in cities which was published in May 2014. The standard includes a set of indicators as well as a methodology to measure these indicators [13].

2.2.3 Carbon Disclosure Project (CDP)

The Carbon Disclosure Project (CDP) is an organization based in the UK aiming at using "[...] the power of measurement and information disclosure to improve the management of environmental risk" [12]. CDP runs several projects regarding environmental governance. The primary focus lies in the disclosure of emission related data. One of these projects is *CDP cities*, which was introduced in 2011. In this project, CDP has collected data on more than 200 cities worldwide. CDP cities offers a reporting platform for city governments using an online questionnaire. The questionnaire covers the following climate-related areas:

- Emissions
- Governance
- Climate risks and adaptation to them
- Opportunities
- Strategy

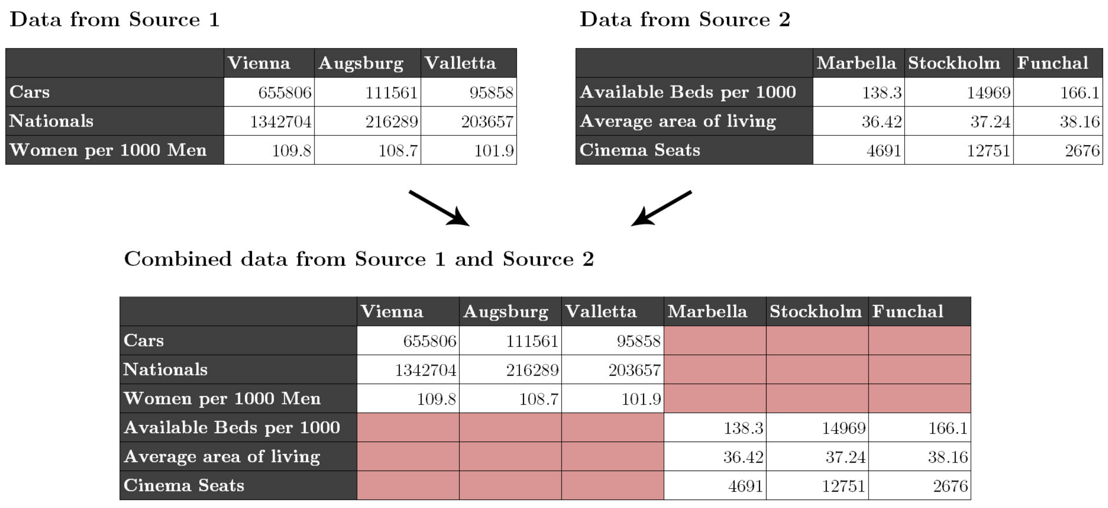
Whereas Urban Audit and GCIF aim to measure the general quality of life in cities, CDP focuses on environmental data (e.g. greenhouse gas emissions, energy sources). Urban Audit and GCIF are therefore more broad efforts to collect city data, e.g. on Demography, Social, Economic as well as Environmental aspects.

2.3 Data Mining

The objective of the Open City Data Pipeline is to support studies and reports by providing rich data on cities. The more data can be published the better. More people and organizations can benefit from the data in the Open City Data Pipeline if more data is provided and the data covers the city or indicator they are interested in. Unfortunately there are currently many missing values in our dataset.

Missing values are a prevalent problem across a wide variety of domains. Some of the possible reasons are faulty equipment, incorrect measurements, missing fields in data from forms or the data was simply not collected or available. Missing values are important for several reasons: Obviously there are less patterns and relations to be extracted from less data. The conclusions which can be drawn are therefore less strong and the decisions based on the data are less reliable.

For the Open City Data Pipeline the large missing ratio is partly due to incomplete data published by the data sources. Another reason is that combining data from different sources which publish data on disjoint cities and indicators (i.e. there are no overlapping rows or columns in the datasets) exacerbates the ratio of missing values.



Data from Source 1

	Vienna	Augsburg	Valletta
Cars	655806	111561	95858
Nationals	1342704	216289	203657
Women per 1000 Men	109.8	108.7	101.9

Data from Source 2

	Marbella	Stockholm	Funchal
Available Beds per 1000	138.3	14969	166.1
Average area of living	36.42	37.24	38.16
Cinema Seats	4691	12751	2676

Combined data from Source 1 and Source 2

	Vienna	Augsburg	Valletta	Marbella	Stockholm	Funchal
Cars	655806	111561	95858	🟥	🟥	🟥
Nationals	1342704	216289	203657	🟥	🟥	🟥
Women per 1000 Men	109.8	108.7	101.9	🟥	🟥	🟥
Available Beds per 1000	🟥	🟥	🟥	138.3	14969	166.1
Average area of living	🟥	🟥	🟥	36.42	37.24	38.16
Cinema Seats	🟥	🟥	🟥	4691	12751	2676

FIGURE 2.2: Growing number of missing values when integrating multiple datasets (mockup data)

Figure 2.2 illustrates this problem (simplified). In this example, there are no overlapping cities or indicators, therefore no data integration can be performed. Instead the data sets are concatenated and merged together. Each dataset has zero missing values considered separately. However, once you join them together half the combined dataset is empty which equals a missing ratio of 50%.

The obvious solution to the problem of missing values is getting additional data by collecting it. This is, in principle, the best option because data quality is (usually) high [1]. However, getting additional data is in our case not feasible because there are no resources to manually collect these vast amounts of data on thousands of cities. Getting data from individual city sites was also out of the scope of this work due to the high resource demand of such an effort. Only data covering multiple cities from large organizations and communities (e.g. Eurostat, Wikipedia) is currently integrated. An extensive search for these datasets was conducted and the sources found were integrated (see Section 3.2).

Since collecting additional data is not a feasible alternative in this context, some form of prediction (i.e. imputation of missing values) is required. The missing values need to be inferred from the existing data as accurately as possible which can be done with *Data Mining*.

"Data mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner" [21, p. 6]. It is an inter-disciplinary field which combines elements of statistics, database technology, machine learning and artificial intelligence. The unsuspected relationships discovered in the observed values are utilized to predict unobserved values.

2.3.1 Data Mining Models

Data mining aims at revealing unsuspected relationships in the data. There are empirical dependencies between indicators in the data. Data mining techniques are used to build an explicit model of these dependencies [48, Chapter 1.2.1].

Data Mining Models can be descriptive or predictive. Descriptive models summarize the data in a new and (hopefully) better way. Predictive models on the other hand allow inferring new information. An example of a descriptive data mining technique is *Clustering*. Clustering methods try to find homogeneous groupings of a set of observations by forming clusters of observations which are similar to each other [54].

Regression and *classification* models are examples for predictive data mining techniques aiming at predicting a value. If the value to be predicted is continuous or quantitative this is called a regression problem. Classification on the other hand involves predicting a non-numerical (or categorical) value [27]. The output of a regression model is therefore always a quantitative value whereas the output of a classification model is always a categorical value.

In the course of this work we are only concerned with predicting numeric values. Therefore, we are dealing with a regression problem. In general, in regression there are one or more variables used for the prediction (i.e. the input variables) and one variable to be predicted (i.e. the output variable). In the case of the Open City Data Pipeline the variables are indicators. Variables used for the prediction are called **predictors** here (other names are *independent variables*, *regressors*, *explanatory variables*), variables to be predicted are called **target** here (other names are *dependent variable*, *criterion variable*, *response variable*). The input of a regression model are therefore the predictors and the output of the model is a value for the target. A model describes how this transformation from inputs to output is achieved.

Observations for which both predictors and target variable are known are used for constructing a model. This model allows predicting the values of the target variable for observations for which only the predictors are known. The new information which is inferred is therefore the previously missing target value [21].

Regression models can be of different forms: they can consist of a search algorithm to find similar observations, they can consist of a tree which splits up the data into homogeneous subsets or they can consist of a simple linear formula describing which indicators need to be combined how to predict another indicator. Independent of the specifics of the model, it always describes how to combine the predictors to get to a value for the target variable.

2.3.2 Regression methods

In this section three regression methods are introduced: k-nearest neighbour, linear regression and decision trees. To illustrate the regression methods we are going to use the example data on German cities in Table 2.1. This data was extracted from the pipeline's database, its original data source is Eurostat.

City	**Price for taxi**	**Area of living**	**Unemployment rate**	**Household income**
Augsburg	8.30	38.9	9.6	18500
Berlin	10.00	37.9	16.7	17400
Bochum	7.30	38.4	10.1	19100
Bonn	7.90	43.3	5.6	21300
Bremen	9.50	41.9	11.8	17500
Cologne	8.24	40.2	9.2	19700
Dortmund	9.00	38.6	11.6	18500
Dresden	8.10	30.9	13.1	16900
Erfurt	9.10	35.2	15.7	17100
Essen	6.04	39.0	8.5	20200
Hamburg	10.50	40.2	9.9	19400
Nuremberg	13.00	40.7	11.2	19400
Frankfurt	12.00	38.5	7.6	?

Table 2.1: Example data

There are 13 observations of four indicators: price for taxi, average area of living, unemployment rate and median household income. The three indicators price for taxi, average area of living and unemployment rate are the predictors. Whereas for the sake of illustration we assume that the median household income is the target. The household income of Frankfurt is missing. It is now demonstrated how this missing value can be predicted using different regression methods.

K-Nearest Neighbour regression

K-nearest neighbour algorithms (KNN) are one of the most wide-spread data mining techniques applied in a variety of domains. "The algorithm is simple, easily understandable and reasonably scalable[...]" [60, p. 9]. KNN can be used in variants for clustering and classification as well as regression.

The first step of the algorithm is to figure out which observations are the closest (or nearest) to our target observation (in our case Frankfurt). To measure closeness we can use one of several so called *distance metrics*. There are multiple distance metrics available. One of the most commonly used is the Euclidean distance. The formula for the Euclidean distance of two observations x and y having N indicators is given in Equation 2.1.

$$D = \sqrt{\sum_{i=1}^{N}(x_i - y_i)^2} \tag{2.1}$$

A major drawback of calculating the distance directly from the raw data is that indicators with very high absolute values will far outweigh other predictors (in our example area of living would outweigh the other two predictors). Since all predictors should be of equal importance the data should be normalized. A value X is normalized to a value between 0 and 1 with the formula

$$X_N = \frac{X - Min}{Max - Min} \tag{2.2}$$

where Min is the minimal value of the indicator and Max is the maximum value of the indicator. The normalized data from Table 2.1 with the Euclidean distances to the target observation is shown in Table 2.2.

The column `k rank` shows the rank of the observations based on their distance to the target observation Frankfurt. The lower `k rank` is, the smaller is the distance of the observation to the target. Assume that we choose $k = 3$, i.e. we are looking at the three nearest neighbours. Based on the indicators in our example data, the three nearest neighbours of Frankfurt (i.e. the three observations with the lowest distance value to the target) in our example would be Hamburg, Nuremberg and Dortmund. The median household incomes in these cities are € 19400, € 19400 and € 18500 respectively (see Table 2.1).

The final step of the algorithm is to use the values of the found k nearest neighbours to calculate a prediction for the target. The easiest implementation is to calculate the mean

City	Price for Taxi	Area of living	Unemployment rate	Euclidean Distance	k rank
Augsburg	0.325	0.645	0.360	0.562	4
Berlin	0.569	0.565	1.000	0.870	10
Bochum	0.181	0.605	0.405	0.712	7
Bonn	0.267	1.000	0.000	0.728	8
Bremen	0.497	0.887	0.559	0.589	6
Cologne	0.316	0.750	0.324	0.576	5
Dortmund	0.425	0.621	0.541	0.562	3
Dresden	0.296	0.000	0.676	0.967	12
Erfurt	0.440	0.347	0.910	0.881	11
Essen	0.000	0.653	0.261	0.861	9
Hamburg	0.641	0.750	0.387	0.329	1
Nuremberg	1.000	0.790	0.505	0.397	2
Frankfurt	0.856	0.613	0.180		

TABLE 2.2: Example data normalized

of the k values which leads to an estimated median household income in Frankfurt of € 19100. There are also more fine-grained variants of the algorithm which are considering the distance of the neighbours by weighing the values with their respective distance to the target.

KNN algorithms can also be applied to classification problems. Here the selected k nearest neighbours decide with a "majority vote" on the predicted class of the target. For example, if the target in our example would be categorical and Hamburg and Nuremberg would belong to category A and Dortmund to category B, then Frankfurt would be assigned category A according to the majority vote (2 for A, 1 for B).

In Chapter 5 of the present work we use the function `knnImputation()` of the R package DMwR, which fills in missing values by applying KNN regression. "For each case with any NA value it will search for its k most similar cases and use the values of these cases to fill in the unknowns." [55] The string "NA" stands for a missing value in R. The function does not simply use the mean of the target values of the nearest neighbours. Instead, it obtains an average of their values weighted by the distance to the target case to fill in the missing values.

Linear Regression

Generally in *linear regression* analysis the objective is to find a linear relationship between a target (see Section 2.3.1) and one (simple linear regression) or more than one (multiple linear regression) predictors [53]. The following formulas show simple linear regression (2.3) and multiple linear regression (2.4) where Y is the target, $X_1 \dots X_p$ are predictors, β_0 is the Y-intercept, $\beta_1 \dots \beta_p$ are regression coefficients, and p is the number of predictors.

$$Y \approx \beta_0 + \beta_1 \times X \tag{2.3}$$

$$Y \approx \beta_0 + (\beta_1 \times X_1) + (\beta_2 \times X_2) + \ldots + (\beta_p \times X_p) \tag{2.4}$$

The linear relationship can be expressed as a so-called regression line going through the data points. The line will seldomly fit perfectly therefore the goal is to minimize the cumulated distance between the line and all data points which results in the best fitting model. There are a several ways of measuring this closeness. The most common approach is called ordinary least squares (OLS). The model is built such that the sum-of-squares of differences of predicted ($y\prime$) and observed (y) values is minimized:

$$minimize \sum (y - y\prime)^2 \tag{2.5}$$

Despite its simplicity linear regression often yields surprisingly accurate results: "Though it may seem somewhat dull compared to some of the more modern statistical learning approaches [...], linear regression is still a useful and widely used statistical learning method." [27, p. 59]

As one of the most basic techniques in statistics, linear regression is natively supported by the programming language R. It is part of the `stats` package which contains a wide range of functions for statistical calculations and also includes the function `lm()` for fitting linear models. It takes a data set and a specification of predictors and a target as input and produces a linear model as output.

Applied to our example data in Table 2.1 the `lm()` function produces the following linear relationship where X_1 is the value for the indicator "Price for taxi", X_2 is the value for the indicator "Area of living " and X_3 is the value for the indicator "Unemployment rate". The observations with values for the target indicator (Household income) are used to build the model, i.e. all observations except Frankfurt.

$$Y \approx 18445.34408 + (25.05874 \times X_1) + (97.85549 \times X_2) + (-334.94201 \times X_3) \tag{2.6}$$

The values X_1, X_2 and X_3 can now simply be replaced with their respective values of Frankfurt (12.00, 38.5, 7.6) and the equation can be solved to find the prediction for the Household income in Frankfurt:

$$18445.34408 + (25.05874 \times 12.00) + (97.85549 \times 38.5) + (-334.94201 \times 7.6) = €19967.93 \tag{2.7}$$

Decision Trees

Tree-based methods involve segmenting the data into multiple smaller regions. Each segmentation is based on splitting rules (i.e. a test on a predictor) which define how to process further. The predictor which provides the most information gain is tested at the root node of the tree. The result of the test defines which branch to follow to the next segmentation. Once the final segmentation is done one can take a metric of all the target values of observations which fall in the segment (e.g. mean, median) to fill in the missing target value.

Consider the example data in Table 2.1. *Decision trees* are built top-down starting from a root node and the process involves partitioning the data into subsets which contain homogenous observations (i.e. observations with similar values). Various metrics for homogeneity are available. We are going to use standard deviation in our example. The formula for sample standard deviation is

$$s = \sqrt{\frac{\sum(\chi - \mu)^2}{n}} \tag{2.8}$$

where χ is the observed value, μ is the sample mean and n is the number of observations. The standard deviation of the household income is 1295.18 (calculated with the first 12 observations, i.e. without Frankfurt).

The decision on which indicator is at the root node is based on the decrease in standard deviation after the dataset is split on this indicator. For example we are splitting the dataset on the indicator "Unemployment Rate". The algorithm is explained more easily if the predictors are categorical variables. (The algorithm works the same in principle for continuous variables.) Therefore, for simplicity, we are converting the numerical values to three categories: high, medium, low. The observations are therefore categorized as follows:

City	**Price for taxi**	**Area of living**	**Unemployment rate**	**Household income**
Augsburg	medium	medium	low	18500
Berlin	high	low	high	17400
Bochum	low	low	medium	19100
Bonn	low	high	low	21300
Bremen	high	high	high	17500
Cologne	medium	high	low	19700
Dortmund	medium	medium	medium	18500
Dresden	low	low	high	16900
Erfurt	medium	low	high	17100
Essen	low	medium	low	20200
Hamburg	high	medium	medium	19400
Nuremberg	high	high	medium	19400
Frankfurt	high	medium	low	?

Table 2.3: Example data with categorized indicators

The dataset is split into the following three tables based on the unemployment rate category.

City	**Unemployment rate**	**Category Unemployment rate**	**Household income**
Berlin	16.7	high	17400
Bremen	11.8	high	17500
Dresden	13.1	high	16900
Erfurt	15.7	high	17100

Table 2.4: Subset with high Unemployment Rate

City	**Unemployment rate**	**Category Unemployment rate**	**Household income**
Bochum	10.1	medium	19100
Dortmund	11.6	medium	18500
Hamburg	9.9	medium	19400
Nuremberg	11.2	medium	19400

Table 2.5: Subset with medium Unemployment Rate

City	**Unemployment rate**	**Category Unemployment rate**	**Household income**
Augsburg	9.6	low	18500
Bonn	5.6	low	21300
Cologne	9.2	low	19700
Essen	8.5	low	20200

Table 2.6: Subset with low Unemployment Rate

The standard deviation of the Household income for the Table 2.4 is 238.48, for the Table 2.5 is 367.42 and for the Table 2.6 it is 1005.92. The mean standard deviation of the entire sample is therefore 537.28. The standard deviation reduction achieved with this split is 757.91, i.e. the difference between the standard deviation before the split (1295.18) and after the split (537.28).

This procedure is repeated for every predictor to find out which split provides the greatest standard deviation reduction. The predictor with the highest standard deviation reduction will be at the root node of the tree. For our example dataset this is the Unemployment Rate. The dataset is split according to the categories "high", "medium" and "low". The steps described above are now repeated on the three subsets to find the best possible split (providing the largest standard deviation reduction) for each subset.

Figure 2.3 shows the final decision tree. The unemployment rate is at the root node because splitting the data on it provides the greatest standard deviation reduction. Due to the limited number of observations in our example data the tree is very small. Real world decision trees (especially for larger dataset like ours, see Chapter 4) have far more nodes and segments.

Frankfurt has a low unemployment rate and (in the subset of cities with low unemployment rate) a high price for a taxi. This puts Frankfurt in the same subset as Augsburg (Household income € 18500) and Cologne (Household income € 19700). The average of these two values is € 19100 which is the prediction for the missing household income of Frankfurt.

Decision trees can be applied for classification and regression [27, chapter 8]. There are many different advanced variants of decision trees, e.g. CART (classification and regression trees), ID3 and C4.5 [44].

In the present work we use the so called random forest algorithm. Unlike in the example above, where we only built one tree, the idea is to build many decision trees from the same data set using random predictor selection to create trees with variation which

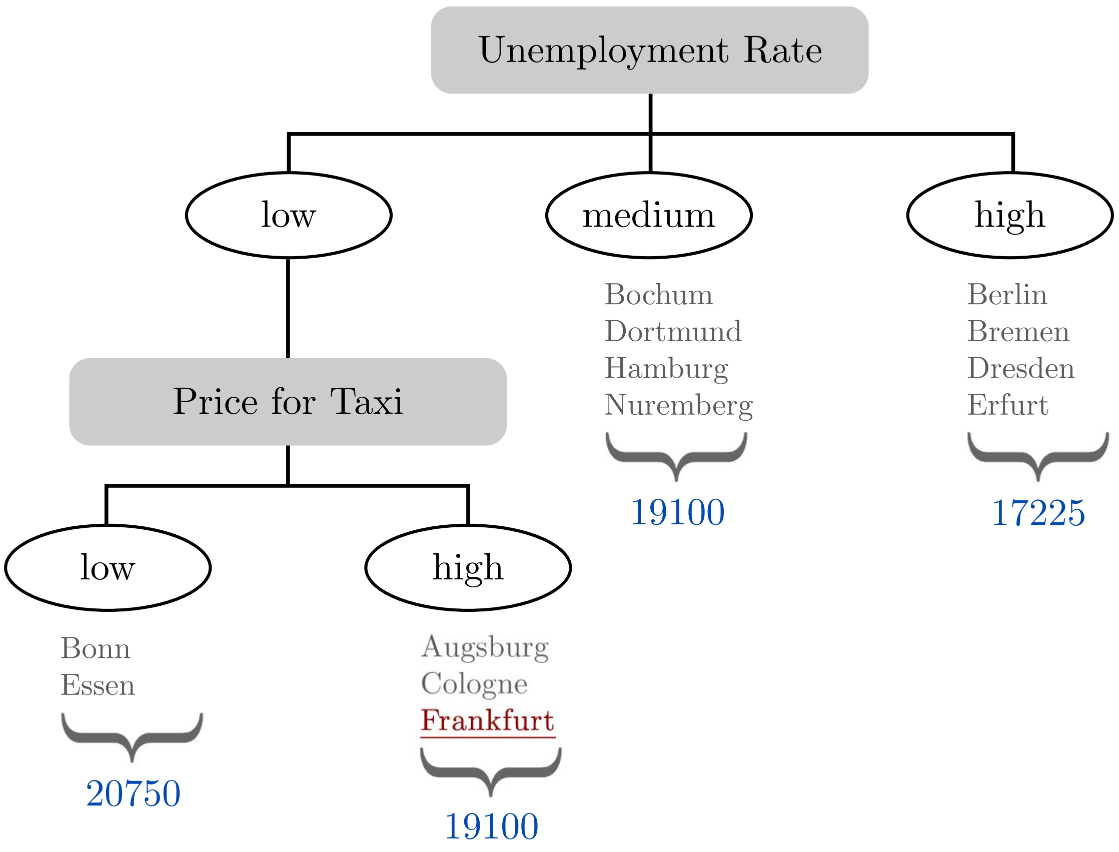


FIGURE 2.3: Decision Tree

together "vote" to arrive at a more accurate value. This way the predictive performance can be significantly increased.

We use the function `randomForest()` of the R package `randomForest` [36], which implements Breiman's random forest algorithm. The algorithm is described in detail in [9]. Trees are built by providing the function with a formula of the target and the predictor variables.

2.3.3 Prediction accuracy

We have seen three regression methods which build different models: KNN defines a similarity metric and searches the dataset for similar observations to come up with a prediction for a missing value. In linear regression we are creating a formula which describes how the predictor variables need to be combined to arrive at a prediction for the target variable. Finally tree-based methods aim at building a model in the form of a decision tree which builds subsets of the data based on splitting rules.

One of the most important properties of these models is the prediction accuracy. The error denotes the absolute difference between the predicted value and the actual value.

"Average error produced by a model is encapsulated in the mean squared error (MSE) or its square root, the root mean squared error (RMSE)." [58, p. 187] The formula for the root mean squared error is given in Equation 2.9 where y_t is the observed value of prediction t, $y\prime_t$ is the predicted value and n is the number of predictions.

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n}(y_t - y\prime_t)^2}{n}} \tag{2.9}$$

The main advantage of RMSE is that it is easily comprehensible because it has the same scale as the predicted variable, i.e. a RMSE of 500 for the indicator household income means that the average error of the model is € 500.

One disadvantage is that its not suitable for comparisons among indicators with different scales. Furthermore the RMSE does not contain any information on the range of possible values. It is only helpful in combination with information on the dataset. For this reason in this work the RMSE is normalized by dividing it by the range of the indicator y (i.e. the maximum value y_{max} minus the minimal value y_{min}) resulting in the model performance metric RMSE%. RMSE% is the metric used for evaluating all models built in this work. The formula is shown in Equation 2.10. An overview of other evaluation metrics is given in [21, chapter 7].

$$RMSE\% = (\frac{RMSE}{y_{max} - y_{min}}) \times 100 \tag{2.10}$$

Consider the data given in Table 2.7 as an illustration. It shows error metrics for the three predictions explained above assuming that the value for the average household income in Frankfurt, which was missing in Table 2.1, is actually € 19500.

Method	**Observed**	**Prediction**	**Error**	**MSE**	**RMSE**	**RMSE%**
knn (k=3)	19500	19100.00	400.00	160000.00	400.00	9.1
linear regression	19500	19967.93	-467.93	218958.48	467.93	10.6
decision tree	19500	19100.00	400.00	160000.00	400.00	9.1

Table 2.7: Error metrics for predictions on example data

RMSE has the same scale as the predicted indicator, i.e. Euro for household income. The error of the value predicted by the knn algorithm is € 400. This is helpful information in a setting where all indicators are expressed in Euro. However we are dealing with a large number of different indicators measured on different scales therefore the RMSE needs to be normalized to be helpful for comparisons among indicators with different scales (see column RMSE% in Table 2.7).

The lowest RMSE% is scored by both knn and decision tree with 9.1%. This observation implies that linear regression should not be used to predict missing values of the indicator median household income in this particular dataset. Throughout this work the three regression methods presented are always applied to all indicators and the best performing method is picked for that specific indicator. This leads to better prediction accuracy overall (see Section 5.1.3 and Section 5.2.3).

2.3.4 Overfitting

Focusing solely on error metrics like RMSE can lead to a serious problem in data mining called *overfitting*. While a model able to predict values with a very low error is better at describing the available dataset, at a certain point it is questionable whether this actually means that the model can generally be considered ”better”, i.e. whether it is describing the underlying relationships in the greater population from which the data stems or whether it is merely adjusting to the specific data at hand.

If ”[...] any of the same data that have been used for selecting a model [...] are then also used again for performance evaluation, then the evaluation will be optimistically biased.” [21, p. 139] In this case, the model is too close to the available data. It is not applicable for new, unseen data and therefore not capturing the unsuspected relationships we are interested in. It exclusively describes the available data but not the underlying structure and relations.

To avoid this phenomenon the data is split into two complementary subsets: a training set and a test set. The goal is to build a model which generalizes well to an independent dataset. This independent dataset is the test set.

The values for the target are known in both the training set and the test set. However, only the training set is used to build the model. This model is subsequently applied to predict the values in the test set. For KNN this means that the nearest neighbours are searched for in the training set. The target values of these nearest neighbours are then used to predict the target values in the test set (which are known - otherwise there could be no error metric calculated). Likewise, the formula in linear regression and the decision tree in tree-based regression are built with just the training set and subsequently applied to predict the target values in the test set.

This procedure is also referred to as cross-validation. Error metrics like RMSE are calculated for both the training and the test set. If a given method performs very well on the training set but has a large RMSE on the test set, the model is overfitting the data. The optimal solution would be a low RMSE for both the training and the test set [27, chapter 2.2].

2.3.5 Principal Component Analysis

PCA is a statistical technique for finding patterns in data of high dimension [49]. "The central idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of a large number of interrelated variables, while retaining as much as possible of the variation present in the data set" [30, p. 1]. The indicators in our dataset (i.e. dimensions) are in parts strongly correlated. Therefore, our high-dimensional dataset is highly suitable for PCA.

PCA reduces the number of dimensions without much loss of information. The new dimensions, called Principal Components (PCs) are uncorrelated (as opposed to the original indicators of our dataset, which are partly highly correlated). PCs are represented by Eigenvectors and Eigenvalues. The Eigenvector defines the direction of the PC in the high-dimensional space. The Eigenvalue is a number describing how much variance there is in the data in that direction. PCs are ordered decreasingly by their Eigenvalue. The Eigenvector with the highest Eigenvalue is called "the Principal Component" [49]. The first PC captures the largest part of the variance in all of the original indicators, i.e., it contributes most to explaining the original data from which the PCs were calculated.

PCA is a form of matrix decomposition. [48] gives an excellent general introduction into matrix decompositions. It suggests four interpretations of matrix decompositions:

- **Factor interpretation**
  There are hidden or underlying factors and the original data is a mix of these factors. The matrix decomposition uncovers these hidden underlying factors which have more direct significance.
- **Geometric interpretation**
  The rows of the original data can be viewed as coordinates in a multi-dimensional space. The matrix decomposition describes the same data with a new set of coordinates in a mult-dimensional space with less dimensions.
- **Component interpretation**
  There are different processes that contribute to the values in the original dataset. Each value represents a blend of multiple components. The matrix decompositions separates these components stemming from different processes.

- **Graph interpretation**
  The columns and rows can be viewed as nodes in a bipartite graph and the values in the original dataset represent the weight of the connection between them. A bipartite graphs is a graph in which the nodes are divided into two classes and every edge passes from one node class to the other. The matrix decomposition transforms this graph in a tripartite graph (i.e., three classes) with the row nodes, the column nodes and nodes acting as waystations. The book gives the following example to illustrate this: "[...] suppose we have a dataset whose rows are people, and whose columns are famous works of art. The people are asked to indicate, say on a scale of 1 to 10, how much they like any of the works of art with which they are familiar. In the graphical view, these entries become weights on edges linking people to works of art. After the decomposition, these direct links between people and works of art become indirect links passing through waystations. These waystations may correspond to different groupings of taste in art" [48, p. 34].

"Each of the interpretations is simply a way of looking at exactly the same decomposition." [48, p. 34] The factor interpretation seems to be the most natural interpretation for our dataset. Underlying factors of our indicators could be the size of the city, the economic strength, the cultural affinity etc. Of course there are too many indicators in our dataset to be able to reduce them to such broad factors.

Since the calculation requires a complete dataset without missing values, some form of imputation is needed before the PCA can be performed. One option is to fill the missing values with the column mean (i.e. the average value of the indicator) but there are more advanced methods, e.g. the regularized iterative PCA algorithm (also known as the expectation-maximization PCA, EM-PCA, see [46]). The implementation details are discussed in [32]. The EM-PCA algorithm is an advanced statistical method which considers the standard deviation, variance and mean from the observed values for an initial imputation and then performs PCA repeatedly on the imputed dataset until convergence (i.e. until there is no more significant improvement). "The iterative PCA method improves the prediction of the missing values compared to the mean imputation." [32, p. 87] Note that this imputation is aiming at keeping the properties of the indicators in the dataset such as standard deviation and variance intact. The goal is an imputation without distortion of the data to enable performing a PCA on the completed dataset.

Chapter 3

Open City Data Pipeline

In this chapter we describe the Open City Data Pipeline in more detail.

The pipeline is "[...] a system for gathering city performance indicators published as Open Data in order to ease the compilation of studies and reports" [43, p. 1]. Data from various sources (see Section 3.2) is collected (see Section 3.1.1), integrated (see Section 3.1.2) and republished as Open Data using an ontology specifically created for the Open City Data Pipeline (see Section 3.3). A Web Interface enables easy access to the data (see Section 3.1.4).

3.1 Architecture & Components

In this section, we outline the architecture of the Open City Data Pipeline. We describe the individual components and their relations.

The pipeline comprises the following components:

- Data collection (see Section 3.1.1)
- Data transformation & integration (see Section 3.1.2)
- Database (see Section 3.1.3)
- Web interface (see Section 3.1.4)

Figure 3.1 depicts the architecture of the Open City Data Pipeline. Data collection needs to be done first. It provides the raw data for the data transformation and integration component. Once the data is properly transformed and conformant with the city data ontology, it can be loaded into the database. The web interface sits on top of the architecture. There is a bidirectional communication scheme between the web interface and the database: SPARQL queries resulting from user interaction are sent to the database which performs the query and gets back the result to the interface. The web interface presents the query results to the user.

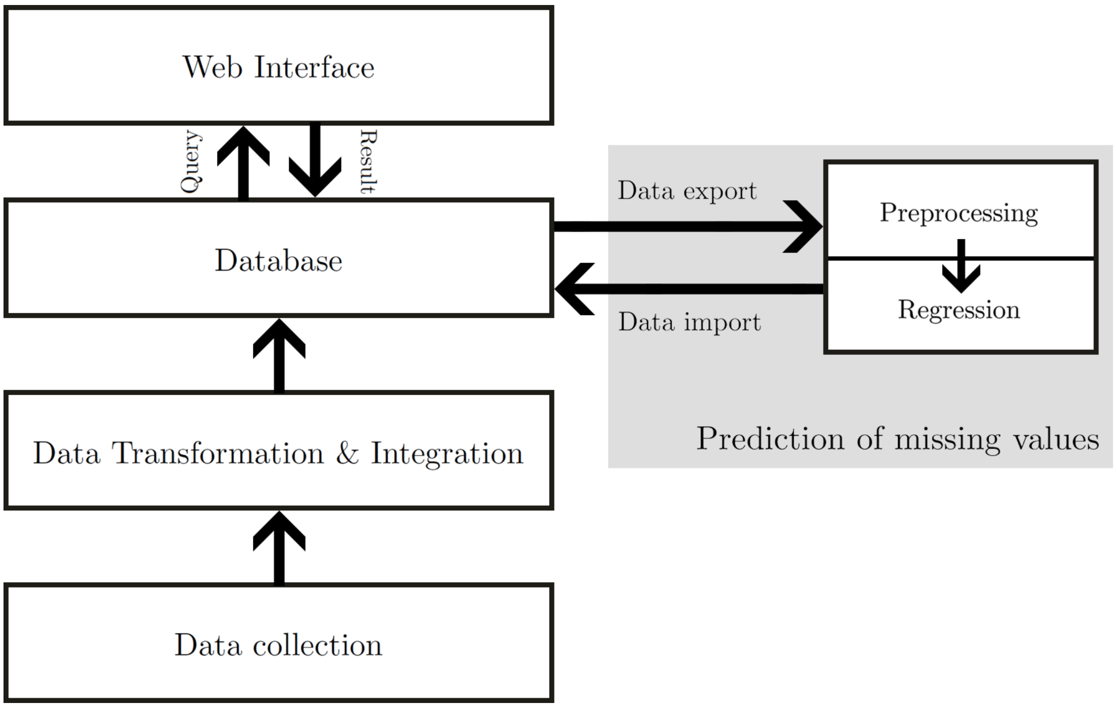


FIGURE 3.1: Architecture of Open City Data Pipeline

Prediction of missing values is done in R. The data needs to be exported from the database and preprocessed to convert it into a format suitable for R. After these preprocessing and cleansing steps (described in 4.2) the regression models are built and the missing values are predicted. The predicted values are then imported back to the database. This part of the architecture is currently implemented separately from the other components. The integration of this prediction component (i.e. automated data export, regression and data import) is the next step in the development of the Open City Data Pipeline.

3.1.1 Data Collection

Data collection is concerned with getting the data from the various sources onto the machine on which the Open City Data Pipeline is set up. In general this can be done via APIs (application programming interfaces) or by directly downloading data files provided by the source.

Getting data via APIs is advantageous because the internal representation of the data is no concern, i.e. an internal change in the way data is organized by the source should not affect your ability to get the data in the same format as before via the API. In practice however there is the disadvantage of poor performance compared to a direct download (e.g. when using the SPARQL end-point of http://dbpedia.org/), especially when collecting larger amounts of data.

Data collection in the Open City Data Pipeline is currently performed with various scripts. In general data can be crawled repeatedly over time to look for new data publications. However this process is not (yet) automated in the Open City Data Pipeline.

3.1.2 Data Transformation & Integration

Data collected from a foreign source is typically not in the format required by your application. The Open City Data Pipeline includes a triple store (i.e. a database for RDF data, see Section 3.1.3). The data in this database follows a specifically created, extensible city data ontology (see Section 3.3). All collected data is transformed to be conformant with this ontology.

This transformation and integration step is done with numerous bash and python scripts which take as an input the data collected in its raw format from the data source and which generate data conformant with the City Data Model ontology as output. This data is subsequently added to the database.

The ontology requires each city to be identified by a unique URI. The Open City Data Pipeline uses DBpedia HTTP URLs of the cities as identifiers. Data coming from other sources than DBpedia is therefore always mapped to such URLs. This ensures consistency and prevents the occurrence of multiple identifiers for the same city. Furthermore, it fosters the discovery of additional information on the cities by following the DBpedia URL.

3.1.3 Database (Apache Jena TDB)

The database used is an Apache Jena Triple Store (TDB). It stores all collected data as well as the city data ontology. The Web Interface (see Section 3.1.4) accesses this database to answer user queries. Jena is a "[...] free and open source Java framework for building Semantic Web and Linked Data applications" [50]. Apache Jena also includes an out-of-the-box SPARQL endpoint (called Fuseki) for serving RDF data over HTTP.

Apache Jena TDB is capable of storing data in the structure of a graph. "The heart of the Semantic Web recommendations is the RDF Graph as a universal data structure." [11, p. 1] The database for the Open City Data Pipeline is structured in multiple graphs: The default graph contains the city data ontology. Additionally, every data source (see Section 3.2) has its own named RDF graph storing the data coming from that particular source.

3.1.4 Web Interface

The web interface enables easy access to the data stored in the database. It provides the user with a simple form by which the user can select one or more cities, one or more indicators and also the time period the user is interested in. The desired answer format can also be specified by choosing between HTML and XML. Figure 3.2 shows a screenshot of this simple web interface.

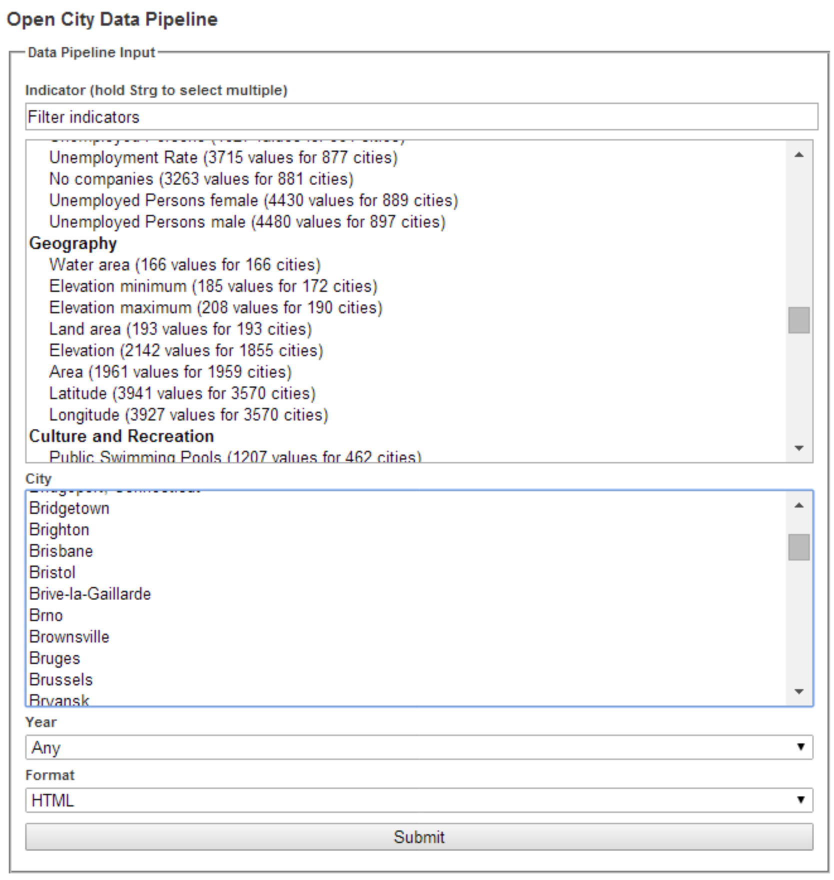
We are currently working on integrating the predicted values into the Web Interface as well. Once the form is filled out and submitted a SPARQL query is built from the user input and sent to the database. The result data is collected and presented to the user in the selected format. For example a user could request the median age and the total population of Vienna, Austria in 2008. Figure 3.3 shows the result of this query in HTML (on the left) and XML (on the right).

3.2 Data Sources

A common level for general comparisons among spatial units available in international statistics are countries. Therefore there are a lot of integrated datasets containing data on numerous countries. Data on cities on the other hand is more likely to be available individually, i.e. each city publishes its data on its own platform and therefore in its own structure, format and style (see for example [2], [3]).

Looking for data on (possibly hundreds or thousands of) individual city data sources and integrating them was not feasible within this work. Another approach is to search for large organizations (e.g. United Nations) or large communities (e.g. Wikipedia and the related effort to extract structured information from Wikipedia called DBpedia [4]) capable of collecting and integrating city data. A search for such datasets was conducted

WU WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS
SIEMENS
Open City Data Pipeline


Open City Data Pipeline

Data Pipeline Input

Indicator (hold Strg to select multiple)

Filter indicators

Unemployment Rate (3715 values for 877 cities)
No companies (3263 values for 881 cities)
Unemployed Persons female (4430 values for 889 cities)
Unemployed Persons male (4480 values for 897 cities)
Geography
Water area (166 values for 166 cities)
Elevation minimum (185 values for 172 cities)
Elevation maximum (208 values for 190 cities)
Land area (193 values for 193 cities)
Elevation (2142 values for 1855 cities)
Area (1961 values for 1959 cities)
Latitude (3941 values for 3570 cities)
Longitude (3927 values for 3570 cities)
Culture and Recreation
Public Swimming Pools (1207 values for 462 cities)

City

Bridgetown
Brighton
Brisbane
Bristol
Brive-la-Gaillarde
Brno
Brownsville
Bruges
Brussels
Bryansk

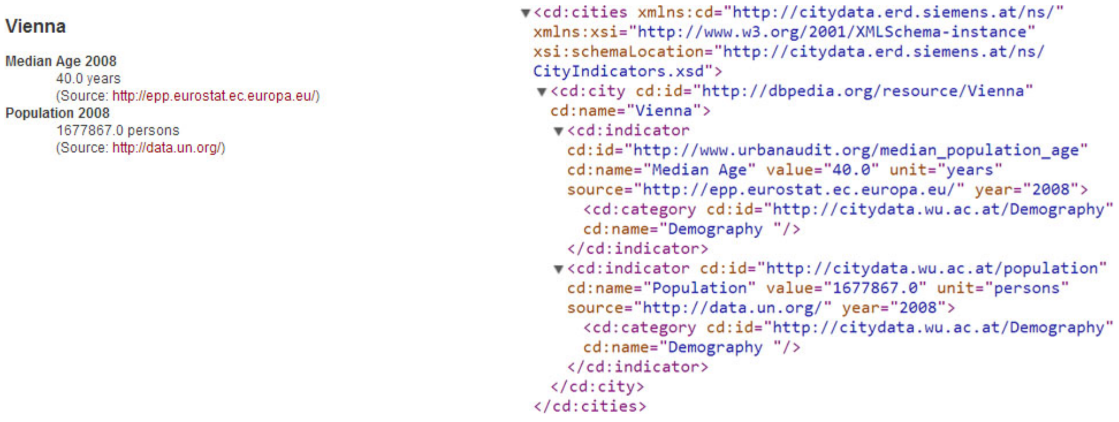
Year: Any

Format: HTML

FIGURE 3.2: Screenshot of web interface input form

and resulted in the following data sources, which were used as data sources for the Open City Data Pipeline:

- Eurostat (Urban Audit data, available at `http://epp.eurostat.ec.europa.eu/`)
- Wikipedia (DBpedia data, available at `http://dbpedia.org/`)
- United Nations (available at `http://data.un.org/`)
- Open Data Europe (available at `http://open-data.europa.eu/`)
- Worldbank (available at `http://worldbank.org/`)



Vienna

Median Age 2008
40.0 years
(Source: http://epp.eurostat.ec.europa.eu/)
Population 2008
1677867.0 persons
(Source: http://data.un.org/)

```
▼<cd:cities xmlns:cd="http://citydata.erd.siemens.at/ns/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://citydata.erd.siemens.at/ns/
  CityIndicators.xsd">
  ▼<cd:city cd:id="http://dbpedia.org/resource/Vienna"
    cd:name="Vienna">
    ▼<cd:indicator
      cd:id="http://www.urbanaudit.org/median_population_age"
      cd:name="Median Age" value="40.0" unit="years"
      source="http://epp.eurostat.ec.europa.eu/" year="2008">
        <cd:category cd:id="http://citydata.wu.ac.at/Demography"
        cd:name="Demography "/>
      </cd:indicator>
    ▼<cd:indicator cd:id="http://citydata.wu.ac.at/population"
      cd:name="Population" value="1677867.0" unit="persons"
      source="http://data.un.org/" year="2008">
        <cd:category cd:id="http://citydata.wu.ac.at/Demography"
        cd:name="Demography "/>
      </cd:indicator>
    </cd:city>
  </cd:cities>
```

Figure 3.3: Screenshot of query result in web interface

The Urban Audit data collections organized by Eurostat are trying to assess the quality of life in European cities. As this is an extensive effort in the context of city data in general, it is described in 2.2 in more detail.

The United Nations Statistics Division (UNSD) offers data on a wide range of topics (e.g. education, environment, health, technology, tourism etc.). The focus of the UN is on country level data but there are some data sets on cities available (for example demographic data and data on housing units). These were collected and integrated into the Open City Data Pipeline.

DBpedia, initially released in 2007, is an effort to extract structured information from Wikipedia and publish it as Linked Data [4]. On cities, DBpedia provides various basic indicators such as demographic or geospatial information (e.g. population, latitute/-longitude, elevation etc.). The Open City Data Pipeline extracts only numerical values from DBpedia. Other data types such as links to other resources (e.g. persons who are born in a city) and textual information (e.g. descriptions) are not collected.

Open Data Europe provides a point of access to a growing range of data from institutions of the European Union (EU) which is free to use for commercial and non-commercial purposes. Data from publishers such as Eurostat, the European Banking Authority or the European Environment Agency is available.

The Worldbank provides free and open access to economical data on countries around the globe. Unfortunately it does not provide data on a lower spatial level (e.g. for cities). However, data on countries is currently extracted and can be used for possible extensions of the pipeline in the future.

Due to the large number of cities on earth some formal limitation on which cities to include and which not to include had to be in introduced. The United Nations provide a list of capital cities and cities of 100.000 or more inhabitants which was used to define this limitation [57, Table 8].

3.3 City Data Ontology

The city data ontology is an RDF data model created specifically for the Open City Data Pipeline describing key concepts of the city domain. Some key concepts of our ontology are cities, indicator categories, indicators and city data contexts. The City Data Ontology is an RDF graph describing all of these concepts and their relationships using RDF, RDFS and OWL (see Section 2.1.2).

3.3.1 Cities

Cities are the items of interest for the Open City Data Pipeline. In conformance with the Linked Data Principles set up by Tim Berners-Lee, URIs of a well-established external data provider are used as identifiers for cities: DBpedia URIs [7].

Indicator data collected from all data sources is mapped to these URIs so people can discover more information on the cities by following the connection to DBpedia and considering additional information, which is available there, as well.

There are numerous possible definitions of the concept of a city. Eurostat for example defines three different spatial units for investigating cities (from smallest to largest):

- The Sub-City District
- The Core City
- The Larger Urban Zone [40, p. 9]

For the Open City Data Pipeline the concept of the Core City is of interest because it corresponds to the administrative definition and is generally adapted most frequently (i.e. it is most comparable to data from other sources although a perfect match over multiple data sources is unlikely because there are different city definitions and varying structures of local government).

3.3.2 Indicators

Indicators in the context of this work are defined as metrics on different aspects of cities. Every indicator belongs to a specific indicator category. The ontology defines the following eight indicator categories:

- Culture and Recreation
- Demography
- Economic Aspects
- Environment
- Geography
- Social Aspects
- Training and Education
- Travel and Transport

These indicator categories are derived from the categories defined by the Urban Audit data collection conducted by Eurostat [40]. All indicators were assigned to an indicator category manually. A list of indicators used for predicting missing values in Chapter 5 can be found in Appendix A. Note that this list does not contain all indicators defined in the ontology (due to removal of indicators during preprocessing and cleansing).

Listing 3.1 shows the definition of the indicator category demography in the ontology. It has the URI `http://citydata.wu.ac.at/Demography` and is simply defined as an Open City Data Pipeline category as well as a `skos:Concept`. In addition, it is given an English and German label. SKOS stands for Simple Knowledge Organization System. It is a data model providing common vocabulary for representation of taxonomies, classification schemes, thesauri etc. "A SKOS concept can be viewed as an idea or notion; a unit of thought." [38]

```
@prefix : <http://citydata.wu.ac.at/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .

:Demography
 rdfs:label "Demography "@en ;
 rdfs:label "Demographie "@de ;
 a :Category, skos:Concept .
```

Listing 3.1: Indicator category definition in City Data Model Ontology

The ontology contains descriptions (`rdfs:comment`) and labels (`rdfs:label`) for each indicator in English as well as in German. Listing 3.2 shows two indicator definitions for population and unemployment rate.

```
@prefix : <http://citydata.wu.ac.at/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dbpedia-owl: <http://dbpedia.org/ontology/> .

:population a owl:DatatypeProperty, rdfs:Property , :Indicator ;
  :unit "persons" ;
  rdfs:comment "Population total"@en ;
  rdfs:comment "Bevölkerung gesamt"@de ;
  rdfs:label "Population"@en ;
  rdfs:label "Bevölkerung"@de ;
  rdfs:domain :CityDataContext ;
  rdfs:range xsd:integer ;
  :category :Demography .

urbanaudit:population_on_the_1st_of_january_total
  rdfs:subPropertyOf :population .

dbpedia-owl:populationTotal
  rdfs:subPropertyOf :population .

:unemployment_rate a owl:DatatypeProperty, rdfs:Property , :Indicator ;
  :unit "%" ;
  rdfs:comment "Unemployment Rate"@en ;
  rdfs:comment "Arbeitslosenrate"@de ;
  rdfs:label "Unemployment Rate"@en ;
  rdfs:label "Arbeitslosenrate"@de ;
  rdfs:domain :CityDataContext ;
  rdfs:range xsd:decimal ;
  :category :EconomicAspects .

urbanaudit:unemployment_rate
  rdfs:subPropertyOf :unemployment_rate .

<http://dbpedia.org/property/unemployment>
  rdfs:subPropertyOf :unemployment_rate .
```

Listing 3.2: Indicator definitions in City Data Model Ontology

Like cities and indicator categories, indicators are assigned a URI as well , in this case `http://citydata.wu.ac.at/population` and `http://citydata.wu.ac.at/unemployment_rate`. Furthermore indicators also have associated units (e.g. persons, EUR). This is important because it enables integration of data with different scales measuring the same phenomenon (e.g. temperature in degrees Celsius and Fahrenheit), which is (partly) addressed in previous work [8]. The indicator population has persons as unit and the unemployment rate has a percentage as unit.

The range of possible values each indicator can have is defined using `rdfs:range` followed by an XML data type. The range of the unemployment rate is `xsd:decimal` whereas the population is always a whole number, therefore the data type is `xsd:integer`. These data types can be helpful for detecting outliers and other errors in the data and are therefore essential for ensuring data quality.

Duplicate data source specific indicators are mapped to a unifying city data ontology indicator. This is a crucial data integration principle of the Open City Data Pipeline. For example Listing 3.2 maps the indicators
`urbanaudit:population_on_the_1st_of_january_total` (Data source: Eurostat) and `dbpedia-owl:populationTotal` (Data source: DBpedia) to the general Open City Data Pipeline indicator `http://citydata.wu.ac.at/population`. The Open City Data Pipeline in its current implementation does not use any information on the data sources to determine the trustworthiness of a specific data point (i.e. there is no ranking saying that data coming from source A should always be trusted over data coming from source B). However such an extension can be done in the future because the provenance of every data point is stored in a City Data Context.

3.3.3 City Data Context

The central class of the ontology is the City Data Context. A City Data Context is a concept introduced for this particular domain. It can be viewed as a container holding valid indicator-value pairs from a specific data source for a specific city at a specific time (typically a year). It is therefore the link between a spatial context (i.e. the city), a time context, a data source and the indicators. City Data Contexts also store information on the provenance of data (i.e. the origin of a specific value).

Listing 3.3 shows (an excerpt of) a Turtle representation of a City Data Context for data from Eurostat on Vienna in 2011.

```
@prefix : <http://citydata.wu.ac.at/> .
@prefix urbanaudit: <http://www.urbanaudit.org/> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .

[ a :CityDataContext ;
  :dateValidity "2011-12-31T12:00:00Z"^^xsd:dateTime ;
  :periodValidity :annual ;
  :source <http://epp.eurostat.ec.europa.eu/> ;
  :spatialContext <http://dbpedia.org/resource/Vienna> ;
  urbanaudit:population_on_the_1st_of_january_total 1714142 ;
  urbanaudit:unemployment_rate 7.4 ;
] .
```

Listing 3.3: A City Data Context

City Data Contexts are represented with a blank node (denoted with `[ ]` in Turtle) because City Data Contexts do not need to be identifiable individually. They are merely a container for the actual data we are interested in.

All the triples in Listing 3.3 have the same blank node in the subject position, therefore they are separated using semicolons. The blank node is an instance of the class `http://citydata.wu.ac.at/CityDataContext`. The point of time for which the information is valid is defined with `:dateValidity` using a string of type `xsd:dateTime`. This datatype - which is itself a IRI - is preceded by `^^`. In our example the `:dateValidity` is December 31st 2011. We use the last day of the year if no more fine-grained time validity is provided by the data source.

The `:periodValidity` describes the expected duration of validity of the data. Typically this is annual but there are other possible values defined in the ontology like quarterly or monthly. The data source is stored as a URI to the publishers website. A triple with `:spatialContext` in the predicate position defines the city which the City Data Context describes (note that this is a DBpedia URI). Figure 3.4 depicts the (simplified) structure of a City Data Context.

In addition to the general information describing the properties of the specific City Data Context like the city, time, data source and expected duration of validity there are also the indicator value pairs. These contain the values of the indicators for this specific city at this specific time. In the example - which only contains an excerpt - there are two indicators with their corresponding values:
`urbanaudit:population_on_the_1st_of_january_total` and
`urbanaudit:unemployment_rate`. These data source specific indicators will be mapped

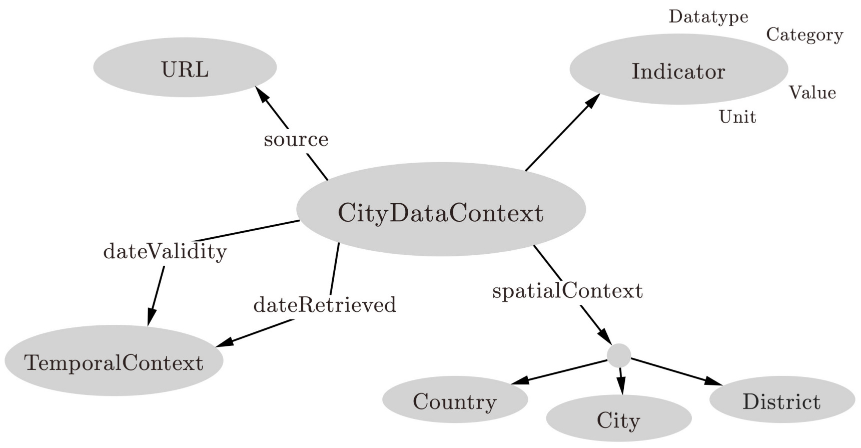


FIGURE 3.4: Structure of City Data Context (based on [41])

to the general Open City Data Pipeline indicators `http://citydata.wu.ac.at/population` and `http://citydata.wu.ac.at/unemployment_rate` as defined in the city data ontology (see Listing 3.2).

Chapter 4

The Dataset

In this chapter we describe the dataset which we use in Chapter 5 in more detail. The dataset used for predictions contains only a subset of all data stored in the TDB. We obtain this subset by querying the TDB and performing multiple preprocessing steps to cleanse the data before any kind of analysis is performed. These steps are necessary because otherwise there would be too many values missing for any reasonable prediction. We describe how the data export is done and what preprocessing steps are performed on the exported data. Finally we give some details on the dataset characteristics with particular focus on missing values.

4.1 Data export

Our Apache Jena TDB [50] can easily be queried with arbitrary SPARQL queries using the command line tool `tdbquery`, which is included in the Apache Jena framework.

Listing 4.1 shows the query obtaining all available City Data Contexts (i.e. all indicator value pairs where the indicator has been explicitly modeled as an `:Indicator`) from the year 2004. The result set contains the city, the indicator and the value for the specified year for every indicator.

```
 1  SELECT DISTINCT ?city ?indicator ?value
 2  WHERE {
 3    GRAPH ?G {
 4      [] :spatialContext ?city ;
 5          ?ind ?value ;
 6          :dateValidity ?Date .
 7      ?city a :City .
 8    }
 9    ?ind rdfs:subPropertyOf ?indicator .
10    ?indicator a :Indicator .
11    FILTER (STR(year(?Date)) = "2004")
12  }
13  ORDER BY ?indicator ?city
```

Listing 4.1: SPARQL query obtaining all City Data Contexts of one year

Where multiple sources exist for the same indicator the individual indicators are mapped to a general indicator (*superindicator*). This is done with `rdfs:subPropertyOf` in Line 9 of Listing 4.1. For simplicity of querying all indicators are always subproperties of themselves. The additional triples to specify this reflexivity are added automatically by the import script. A query sent to the Open City Data Pipeline database should return all the available information regardless of the source (i.e. all the subproperties of the general indicator, see Section 3.3).

General information on the indicator structure and hierarchy (i.e. the city data ontology) is stored in the default graph of the database. Therefore, it is located directly in the WHERE clause of the query. The City Data Contexts on the other hand are stored in a separate graph named after the data source. Therefore the part of the query pattern which is matching City Data Contexts is within a `GRAPH ?G { }` construct because all the City Data Contexts are stored in named graphs.

We are using a SPARQL filter to limit our result set to the parts we are interested in. In the example, the year part (extracted with the function `year()`) of the variable `?Date` has to be 2004. Finally, the results are ordered by the indicator and subsequently by the city.

We are using the command line tool `tdbquery` to send queries to the Triple Database (TDB) (see Listing 4.2). The tool allows for exporting data in the common comma separated value format (CSV) which can be directly used by most data analysis tools such as R or SPSS.

```
tdbquery --loc=tdb-db1 --query query.rq --results CSV > 2004.csv
```

Listing 4.2: Exporting data with tdbquery

This command line statement uses three parameters:

- **loc**: location of the Apache Jena Triple Database.
- **query**: location of file containing the SPARQL query to execute.
- **results**: specifying the result format, in this case CSV.

The result of the query in 4.1 is a table with the columns city, indicator and value stored in a CSV file called `2004.csv`.

4.2 Preprocessing

We conduct several preprocessing steps in order to bring our data in a format suitable for further analysis. The required steps to achieve this are described in this Section.

Year	Source	City	Indicator	Value
2009	Eurostat	Hamburg	1 person households	485500
2009	Eurostat	Vienna	1 person households	396000
2009	Eurostat	Paris	1 person households	594434
2009	Eurostat	Hamburg	Population	1774224
2009	DBpedia	Vienna	Population	1686271
2009	Eurostat	Vienna	Population	1688271
2009	Eurostat	Vienna	Cars	663926
2009	Eurostat	Paris	Cars	532877

Table 4.1: Exported data (mockup data)

Table 4.1 shows some mockup data which we are going to use to illustrate the preprocessing steps.

4.2.1 Duplicates

We are using the programming language R in the following Chapter and R expects the data to be in tabular form with rows and columns. Tables are sets and can therefore not contain duplicates. We want our cities as rows, our indicators as columns and one (or no) value for each of the table cells. However, there are duplicate values in our data, i.e. more than one indicator value is available for a specific city-year combination. These duplicates are present because there are overlapping indicators from different sources. For example, population data is available from Eurostat as well as from DBbpedia.

In its current implementation, the Open City Data Pipeline does not use any information on the data sources to determine the trustworthiness of a specific data point. In such a case the value coming from the most trustworthy source would be taken while the other values would be deleted. Instead, duplicate values are replaced with their arithmetic mean, i.e. all available information is considered. The replacement with the arithmetic mean is a good solution if there are no outliers in the data. An outlier would seriously distort the value (i.e. the calculated mean). In order to avoid this, outlier detection can be applied, which is out of the scope of this work but subject to future work (see Section 6.2).

For example, in the mockup data in Table 4.1 there is one duplicate for Vienna for the indicator population. The two original values 1686271 (from DBpedia) and 1688271 (from Eurostat) are therefore replaced by their mean 1687271.

4.2.2 Transposition

The data needs to be transposed into a data table which has cities as rows and indicators as columns. The SPARQL query which performs the data export from TDB (see Section 4.1) results in a list format. A short `awk` script performs this transposition (see Listing 4.3).

```
awk -F\; '
NR>1 {
    # get indicators
    if(!($1 in indicators)) { indicator[++types] = $1 }; indicators[$1]++

    # get cities
    if(!($2 in cities)) { city[++num] = $2 }; cities[$2]++

    # create map with coordinates indicator/city containing the value
    map[$1,$2] = $3
}
END {
    # print first line (headers: city, indicator1, indicator2, ...)
    printf "%s;" ,"city";
    for(ind=1; ind<=types; ind++) {
        printf "%s%s", sep, indicator[ind];
        sep = ";"
    }
    print "";

    # for every city, print city name and corresponding values
    for(cit=1; cit<=num; cit++) {
        # print city name
        printf "%s", city[cit]
        for(val=1; val<=types; val++) {
            # print values for indicators
            printf "%s%s", sep, map[indicator[val], city[cit]];
        }
        print ""
    }
}'
```

LISTING 4.3: Transposing data from list format to table format

Transposing the mockup data in Table 4.1 would result in Table 4.2. The cities are now the rows and the indicators are the columns. Furthermore, the duplicate was already replaced by calculating the mean of all duplicate values.

4.2.3 Data cleansing

As a general heuristic, columns and rows with too few values are omitted from the analysis. The threshold chosen here is 10% of possible values. This cleanses the data

City Year	1 person households	Population	Cars
Hamburg 2009	485500	1774224	
Vienna 2009	396000	1687271	663926
Paris 2009	594434		532877

Table 4.2: Result of export query - transposed (mockup data)

from columns and rows carrying very little information which enables a better prediction performance.

The operation of trimming a table so that only the rows of interest remain is called *Selection* in relational algebra. All other rows are removed from the table. Rows are city/year-combinations in our dataset. Listing 4.4 shows an R script for deleting all rows with less values than specified as a threshold (10% in this case) from a list of dataframes called `datasets`. The list of dataframes corresponds to the list of the datasets from the different years, each containing the data from one year. A dataframe is a datatype in R corresponding to a table. All rows (i.e. city-year combinations) where less than 10% of all values are known are deleted from the dataframes. Keep in mind that the datasets still contain all columns at this time (regardless of the sparsity in them) and the 10% threshold is calculated from all these columns, not from the 146 columns which remain after the next preprocessing step.

```r
threshold <- 0.1
datasets <- lapply(datasets, function(df) {
  rows <- apply(df, 1, function(x) sum(!is.na(x)))
  df[rows > (ncol(df) * threshold), ]
})
```

Listing 4.4: Deleting rows (selection) with a value count below a threshold from a list of dataframes in R

The operation of trimming a table so that only the columns of interest remain is called *Projection*. All other columns are removed from the table. Columns are indicators in our dataset. Listing 4.5 does the same for columns as the code above for rows. All columns (i.e. indicators) where less than 10% of all values are known are deleted from the dataframes.

```
threshold <- 0.1
datasets <- lapply(datasets, function(df) {
  cols <- apply(df, 2, function(x) sum(!is.na(x)))
  return(df[, cols > (nrow(df) * threshold)])
})
```

Listing 4.5: Deleting columns (projection) with a value count below a threshold from a list of dataframes in R

The sparsely populated columns which are deleted with the code above are often indicators which are available at only one data source, e.g. data from DBpedia concerning weather phenomenon like temperatures, precipitation, snow days. These indicators could still be helpful for predicting some of the other, non-weather indicators. However, the present thesis focuses on an extensive approach which is able to predict many missing values with satisfactory accuracy. Further research on isolated indicators and their correlations is subject to future work.

4.3 Dataset characteristics

Table 4.3 shows the distribution of available and missing values across the years2004-2011. For the analysis conducted in Chapter 5 we are considering all the data from these years. The rationale behind this is that every city at every year is a valid configuration of all indicators and therefore contains valuable information.

For the remainder of this thesis we use an integrated dataset with data from the years 2004 to 2011. The export query shown in Listing 4.1 is therefore repeated for every year between 2004 and 2011. The resulting datasets are then cleansed as described above and afterwards merged together to create a single, large dataset.

year	**cities**	**indicators**	**table cells**	**filled values**	**missing values**	**percent missing**
2004	629	146	91834	45317	46517	50.65
2005	387	146	56502	26642	29860	52.85
2006	406	146	59276	28251	31025	52.34
2007	395	146	57670	26952	30718	53.27
2008	579	146	84534	47898	36636	43.34
2009	559	146	81614	43871	37743	46.25
2010	861	146	125706	70084	55622	44.25
2011	864	146	126144	82307	43837	34.75
all	1064	146	683280	371322	311958	45.66

Table 4.3: Distribution of values across years

The Open City Data Pipeline's database contains data ranging from the years 1970 to 2014. However, the amount of data varies from year to year and most of the data concerns the years after 2000. Furthermore the indicators covered are not the same for all of these 45 years. The years 2004 to 2011 were chosen because the datasets from these years contain a lot of data (relatively) from the same indicators.

The final dataset (containing data of 8 years) has 4680 rows (translating to city-year combinations) and 146 columns (translating to indicators). This results in 683280 table cells and possible values. 371322 of these table cells are filled with values, 311958 are empty.

4.3.1 Cities

There is a total of 1064 distinct cities (4680 city-year combinations equal to rows) in the dataset. There is an average of 79.34 data points per city-year combination (out of 146 columns, equals 54.34%). Figure 4.1 shows a histogram of the distribution of values per row. It shows how the values (and therefore also the missing values) are distributed over the cities. For instance, 224 city-year combinations have more than 140 values (out of 146 possible values).

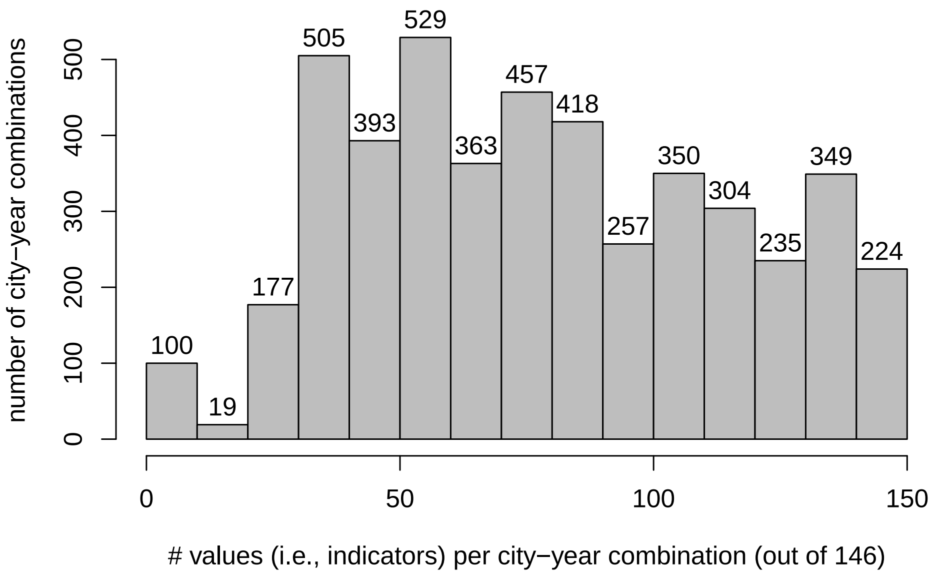


FIGURE 4.1: Histogram of distribution of values per row

Three quarters of the rows (3486 or 74.49%) have more than 50 values and 1462 (or 31.24%) have more than 100 values out of 146 possible.

4.3.2 Indicators

There is a total of 146 indicators in the dataset (see Appendix A). There is an average of 2543.30 data points per indicator (out of 4680 rows). Figure 4.2 shows - in the form of a histogram - that there are no indicators with less than 500 values because this is an already cleansed dataset after preprocessing was performed (see Section 4.2).

For instance, 18 indicators have between 1001 and 1500 values (out of 4680 possible values). Most indicators have between 1001 and 3500 data points. Population indicators are at the high end of the scale since they are the most prevalent and most easily obtainable data points.

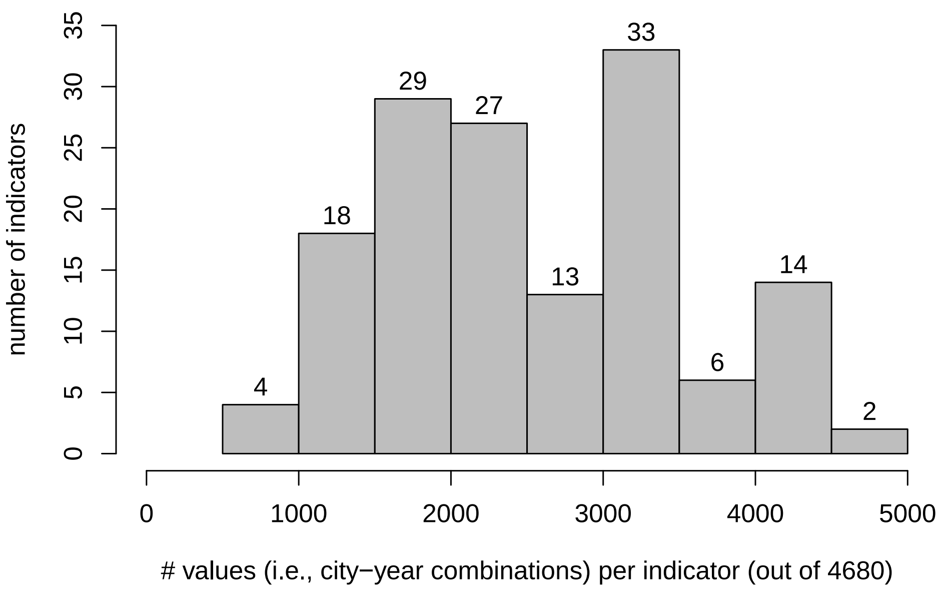


FIGURE 4.2: Histogram of distribution of values per indicator

Table 4.4 shows the distribution of values across indicator categories. There is a total of 7 indicator categories. Geographical information was not used in the dataset, although there could be some potential for future work, e.g. by calculating the physical vicinity of cities using latitude, longitude and altitude and using this distance in the prediction model (see Section 6.2).

category	**indicators**	**indicators %**	**values**	**values per indicator**	**values %**
Culture and Recreation	9	6.16	18365	2041	4.95
Demography	61	41.78	194380	3187	52.35
Economic Aspects	22	15.07	48070	2185	12.95
Environment	10	6.85	18750	1875	5.05
Social Aspects	26	17.81	54499	2096	14.68
Training and Education	8	5.48	16819	2102	4.53
Travel and Transport	10	6.85	20439	2044	5.50
Total	146	100.00	371322	2543	100.00

TABLE 4.4: Distribution of values across indicator categories

If there were an even distribution of indicators across indicator categories each category would contain 14.29 indicators. However, the distribution in the dataset is not even. There are more demographic indicators than indicators of other categories and in addition these demographic indicators contain more values per indicator. This is due to the fact that this information is easier to obtain and in many cases already available to the authorities without additional effort because detailed demographic information is regularly collected in censuses. 52% of all values contain demographic information, followed by social (15%) and economic (13%) indicators.

4.3.3 Missing Values

The dataset has 683280 table cells and 371322 actual values. Therefore in the entire dataset 45.66% of values are missing even after cleansing which makes working with this dataset challenging.

Table 4.3 indicates the missing ratio per year. Between 53% (2007) and 35% (2011) of the values are missing.

Early on in our work we tested the assumption that the ratio of missing values (called *missingness*) in our dataset is dependent on the size of the city. The underlying idea being that bigger cities are more likely to provide more data than smaller ones. A strong correlation between the missingness and one feature in the data would constitute data Missing not at random (MNAR). "When data are MNAR there is presumably some model that lies behind missingness." [24, p. 4] Many data mining techniques are based on the assumption that the missingness in the data is not MNAR. However, the correlation coefficient between the number of missing values per city and the population of the city was found to be around 9% for the dataset (Pearson product-moment correlation coefficient). This observation means that the initial assumption was wrong because missingness cannot be explained with the size of the city.

Chapter 5

Predicting Missing Values

Two approaches to finding predictions for missing values are presented in this chapter. For both of these approaches we will describe the necessary steps which need to be taken to predict the missing values. Finally we will present the results achieved and we will compare them.

5.1 Approach 1 - Building complete subsets

In the first approach we try to build models which directly use available indicators as predictors to predict a target indicator. In order to do that we are using the correlation matrix of the data to find indicators which are suitable as predictors for a specific target indicator.

Subsequently, we build a complete subset from our data, i.e. we first perform a projection on our data table (keeping only the predictors and the target as columns) and afterwards we perform a selection on our data table (keeping only rows without missing values).

We split our complete subset into training and test set and train our model on the training set. The prediction accuracy is evaluated by applying the trained models to our test set and investigating the average error using our prediction accuracy metric RMSE% (see Section 2.3.3).

5.1.1 Overview

In this section we list the necessary steps to build the regression models and make the predictions. These steps are subsequently described in more detail. Steps 2-7 need to be performed for every column (indicator) of the dataset that needs to be predicted:

1. Calculate the correlation matrix of the dataset.
2. Search for indicators with either a strong positive or a strong negative correlation with the target indicator.
3. Create a subset with the target indicator plus the indicators with the highest absolute correlation coefficients (the predictors).
4. Delete all rows with missing values. The subset is now a complete dataset without missing values.
5. Split the dataset into a training set and a test set.
6. Train the model using the training set.
7. Apply the model to the test set and compare the predicted values with the observed values.

5.1.2 Prediction

The steps presented below are repeated for every column in the dataset. Listing 5.1 shows the `for` loop in R to iterate over all columns in the dataset (referenced by the variable `data`). The variable `target` contains the column name of the current iteration.

```
for(target in colnames(data)) {
  # CODE
}
```

Listing 5.1: Looping over all columns in R

Correlation coefficients

The first step of the approach is to calculate the correlation matrix. In order to be able to apply the regression methods presented in Section 2.3.2 a complete subset needs to be built. Due to the large ratio of missing values (see Section 4.3.3) not all of the indicators can be used to build the predictive models because there is not a single observation which has no missing values for any indicator.

A random selection of indicators would be possible but there is a metric which helps determining whether a specific indicator is suitable for predicting another indicator: the correlation coefficient. "A correlation coefficient reflects the strength or degree of linear association between variables or the extent to which the two variables behave alike or vary together." [53, p. 37]

Equation 5.1 gives the formula for the Pearson's product-moment sample coefficient between two indicators X and Y where x are values of the indicator X, y are values of the indicator Y, k is the number of observations and $\bar{x}$ and $\bar{y}$ are the sample means of X and Y.

$$r_{xy} = \frac{\sum\limits_{i=1}^{k}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum\limits_{i=1}^{k}(x_i - \bar{x})^2 \sum\limits_{i=1}^{k}(y_i - \bar{y})^2}} \tag{5.1}$$

Finding and selecting those indicators with a high absolute correlation coefficient (i.e. a highly positive or a highly negative coefficient) with the target indicator gives a set of indicators which can be used as predictors in a regression model. A helpful structure in this endeavour is the correlation matrix.

Correlation matrix

A correlation matrix of indicators $X_1, \ldots, X_n$ is a symmetric $n \times n$ matrix (where n is the number of indicators) whose i, j entry is $r(X_i, X_j)$. The entries of the matrix are the correlation coefficients (see Equation 5.1) indicating how strong the correlation is between the two corresponding indicators. The matrix shows, in a condensed form, how each indicator correlates with all other indicators.

Listing 5.2 shows how this correlation matrix is calculated in R.

```
data <- read.csv("/path/to/data.csv")
corrmatrix <- cor(scale(data), use="pairwise.complete.obs")
```

Listing 5.2: Calculating the correlation matrix in R

R allows calculating the correlation matrix with Pearson's product-moment coefficient using the built-in `cor()` function despite the dataset having missing values. With `pairwise.complete.obs` the correlation between each pair of indicators is computed using all complete pairs of observations on those indicators. The data is normalized using the `scale()` function, which centers the data by subtracting the column mean and scales the data by dividing the (centered) columns by their standard deviations.

Building subset

The regression methods we are using (see Section 2.3.2) need a complete dataset (i.e., without missing values) as input. Listing 5.3 shows the R code for building a complete subset which can be used for the prediction. We are interested only in the correlation coefficients of the target column. These are stored in the vector `correlations`. To get the indicators which are of interest we sort the vector `correlations` by the absolute correlation coefficient in decreasing order. We use the absolute correlation coefficient because both indicators with highly positive correlations and indicators with highly negative correlations are useful as predictors for the target.

```r
correlations <- as.numeric(corrmatrix[target ,])
correlations <- sort(abs(correlations), decreasing = TRUE)
# 1 target and 4 predictors = 5
highest <- head(correlations , 5)

columns <- c(names(highest))
subset <- data[,columns]
subset <- subset[complete.cases(subset),]
```

Listing 5.3: Building the subset for the prediction in R

We use the function `head()` to select the 5 correlation coefficients with the highest values. The number 5 corresponds to the number of predictors (in this case 4) plus 1 target. These are stored in the variable `highest`. Figure 5.1 illustrates this procedure on a small correlation matrix with $n = 10$, with indicator X_4 being the target indicator. In this example the indicators X_1, X_6, X_7 and X_9 have the highest absolute correlation with the target indicator X_4, therefore these are picked as predictors and need to be in the subset which is built subsequently.

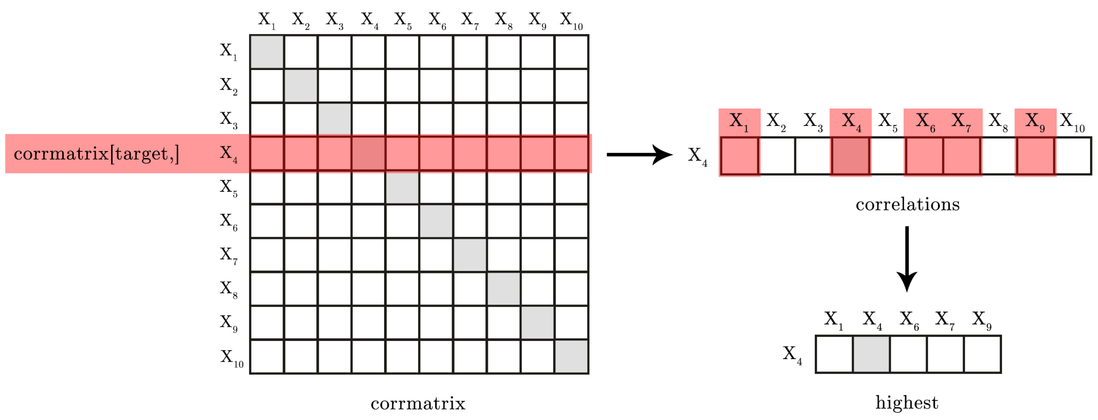

cormatrix[target,]
X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8 X_9 X_{10}
cormatrix
X_4
correlations
X_1 X_4 X_6 X_7 X_9
highest

FIGURE 5.1: Finding indicators for building the subset of predictors and target (see Listing 5.3)

A subset with just these 5 indicators (i.e. 4 predictors and 1 target indicator) is built and referenced by the variable `subset` (see line 6 in 5.3). Only the observations with a value for the predictors as well as the target indicator can be used for building the model. The other observations are omitted with the function `complete.cases()`. This step significantly reduces the number of rows in the subset due to the high ratio of missing values in the data.

Split data into training set and test set

Now that the subset with the target and the predictors is built, the data needs to be split into a training set and a test set (see Section 2.3.4). Listing 5.4 shows the R code to achieve that. The ratio is 50:50 in this example, i.e. the training set and test set have an equal number of observations (or a difference of maximum 1 if the total number of observations is uneven). We use the function `sample()` to get a random sample of half the observations. Finally we build the training set by only keeping those sample observations and we build the test set by deleting these sample observations and keeping the rest.

```
sample_size <- floor(0.5 * nrow(subset))
sample <- sample(seq_len(nrow(subset)), size = sample_size)
trainingset <- subset[sample, ]
testset <- subset[-sample, ]
```

LISTING 5.4: Splitting data into a training set and a test set in R

We are now applying the three regression methods presented in 2.3.2. Half of the data (i.e. the training set) is used to build the model. We use the model to predict the target indicator in the other half (i.e. the test set). We are calculating the standardized root mean squared error (RMSE%, see Section 2.3.3) to evaluate the performance of the regression models on each of the 146 indicators in the dataset.

K-Nearest Neighbour regression

Listing 5.5 shows the code for applying the KNN regression algorithm (see Section 2.3.2) to impute the target column in the test set using the data in the training set, i.e. the nearest neighbours are searched for only in the training set.

```
1 library(DMwR)
2 testset[,target] <- NA
3 prediction <- knnImputation(data=testset, k=9, scale=F, meth="weighAvg",
     distData=trainingset)
```

Listing 5.5: Applying KNN in R (first approach)

In this case we are using the 9 nearest neighbours. The decision to select $k = 9$ is based on experiments. Figure 5.2 shows the RMSE% for $k = 2$ to $k = 19$ and different values for the number of predictors.

We load the necessary package DMwR in line 1 [55]. We delete the values in the target column of the test set in line 2 because they are going to be imputed by the `knnImputation()` function. We use the method `weighAvg`, which means the weighted average of the neighbours' values will be inserted for the missing values (see Section 2.3.2).

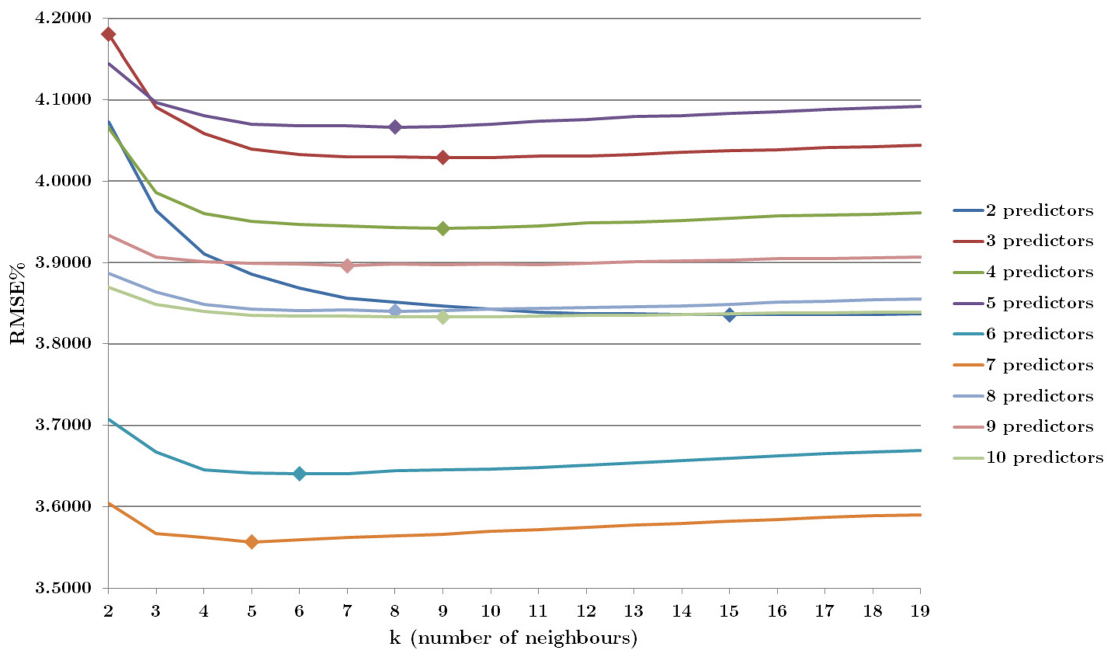


Figure 5.2: Approach 1: finding the best value for k

The lowest RMSE% for each number of predictors is marked with a diamond. With the exception of 2 predictors where $k = 15$ yields the lowest RMSE% the optimal value for k is between 5 and 9. We chose $k = 9$ because it yields an RMSE% close to the minimum for all configurations.

Linear Regression

Listing 5.6 shows the code for applying linear regression (see Section 2.3.2) on the training set to build the model. The built model is stored in the variable `model`. We are applying the `predict()` function on the test set to predict the target column using the predictor columns as specified in the model.

```
model <- lm(formula, data=trainingset)
prediction <- predict(model, newdata=testset)
```

Listing 5.6: Applying linear regression in R

Decision Trees

The syntax for applying the random forest algorithm in R (see Section 2.3.2) as shown in Listing 5.7 is very similar to linear regression. It requires the package `randomForest` to be loaded. Again the model is built using the training set and afterwards applied to predict the target column in the test set.

```
library(randomForest)
model <- randomForest(formula, data=trainingset)
prediction <- predict(model, newdata=testset)
```

Listing 5.7: Applying random forest algorithm in R

5.1.3 Results

There are two main properties to evaluate our approaches for predicting missing values. First, it is important to build models which are able to predict many (preferably all) missing values in our dataset. The number of possible predictions is a key metric for determining the usefulness of the approach.

Second, the prediction accuracy of the models is essential. The Open City Data Pipeline can only be useful for its purpose of serving as a data source for the compilation of studies and reports if it provides high-quality, accurate data and predictions.

Number of possible predictions

The number of predictors used to build the regression models is the key property influencing the number of possible predictions. The more predictors are used the fewer predictions can be made, because the model can only predict the missing values for those city-year combinations where the values for all the predictors are known.

We are building the regression models using a complete subset with the predictors and the target as columns. This subset contains city-year combinations where the values for the predictors and the value for the target are known, i.e., the model inputs (the predictors) as well as the model output (the target) need to be known. For the predictions, only the model inputs (i.e., the predictors) need to be known. The number of possible predictions is therefore the number of city-year combinations for which *all* the predictors are known but the target is unknown. There are (on average) very few city-year combinations having values for all the predictors and no value for the target. Raising the number of predictors will lower the number of city-year combinations having values for all the predictors, therefore: The more predictors are used the fewer predictions can be made.

Predictors	**Possible predictions (mean)**	**Possible predictions (total)**
2	188.10	27463
3	129.68	18934
4	102.56	14974
5	86.65	12651
6	72.34	10562
7	58.75	8578
8	44.88	6553
9	40.86	5965
10	37.95	5541

Table 5.1: Evaluation of possible predictions with Approach 1

Table 5.1 shows how many predictions are possible with the models using different numbers of predictors. This number corresponds to the city-year combinations which have values for all predictors but are missing a value for the target indicator.

Even with very simple models using just two predictors one can only predict 188 values on average per indicator (i.e. circa 27500 values in total). This corresponds to only 8.8% of all missing values in our dataset. Note that, for a subset with 10 predictors, even though the mean number of possible predictions per indicator is circa 38, there are 75 indicators (out of 146) for which not a single predictions can be made. This phenomenon can be explained with the structure of the dataset: for example, there are numerous population category and population proportion indicators (see Appendix A). These population indicators are not only highly correlated (i.e. they are selected as predictors for each other), but they also occur together. For most cities, either all (or at least many) or none (or at least few) of these indicators are available, which limits the number of possible predictions.

In total there are 311958 missing values in our dataset. Even with models using just two predictors we are only able to predict 8.8% (27463) of all missing values. Section 5.2 presents an approach which aims at overcoming this shortcoming.

Prediction accuracy

The three regression methods (K-Nearest Neighbour, Linear Regression and Random Forest Decision Trees, see Section 2.3.2) were applied to predict the values of all 146 columns in the test set. Please note that the prediction accuracy is dependent on the random observation selection used for splitting the data into a training set and a test set and will therefore vary for every time the code is run. However results can be replicated by setting a seed in R with the built-in function `set.seed()` as shown in Listing 5.8. Once a seed is set, random sampling is always going to choose the same sample, i.e. the training set and the test set will always be the same.

```r
set.seed(100)
```

Listing 5.8: Setting a seed in R for replicability

The performance of the regression models were evaluated for 2 to 10 predictors. Figure 5.3 shows the results. All three regression methods have a mean RMSE% between 3.500% and 4.500%. The k-nearest neighbour algorithm appears to work best on average for this dataset. When using 7 predictors this algorithm yields an average RMSE% of 3.562 (see Table 5.2).

To get a better feeling of what these values for RMSE% actually mean, here are two examples: an RMSE% of 3.5 means that predictions for the unemployment rate are off by 1.4% on average, for the average household income the error corresponds to € 2235.

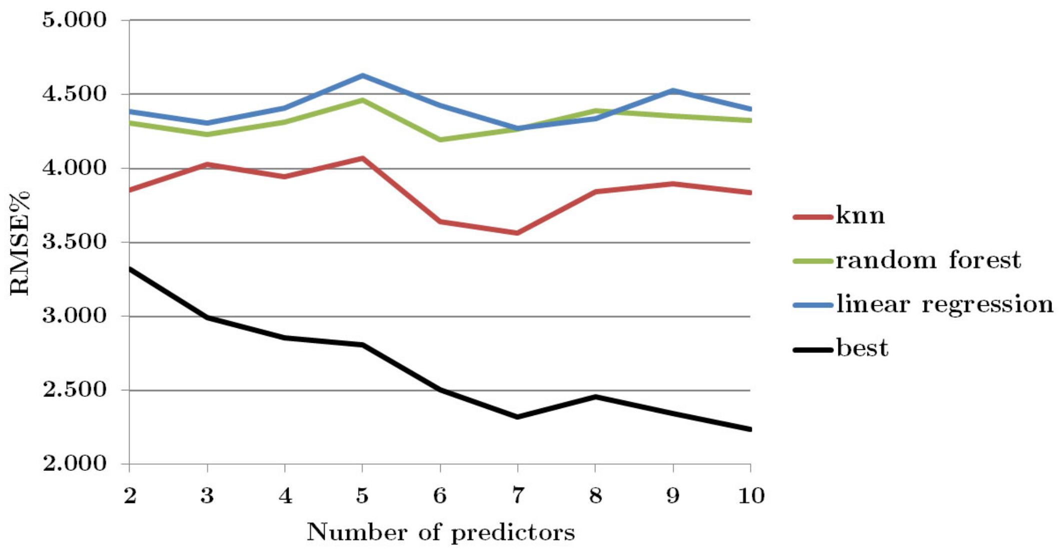


FIGURE 5.3: Approach 1: RMSE% of test set for different number of predictors

There is no significant improvement of the RMSE% achieved by any of the three regression methods by increasing the number of predictors. However, as we are essentially building 438 different models (146 × 3, i.e. every indicator with every regression method) it is natural that some target indicators are better predicted by one regression method and some by another.

Overall the best result can be achieved by picking the best performing regression method for every indicator. This way the missing values of every indicator are predicted by the method performing best on this specific indicator. The black line in Figure 5.3 shows that by doing that the overall result can be improved consistently over any single regression method used. With this methodology the average RMSE% can be reduced significantly. For 7 predictors the average RMSE% is 2.322 which is an improvement of 35% over the best performing single regression method KNN.

Table 5.2 shows the numbers depicted in Figure 5.3.

5.2 Approach 2 - Principal Component Regression

In the second approach, instead of directly using indicators as predictors we first perform a Principal Component Analysis (PCA) to reduce the number of dimensions of the dataset and use the remaining compressed dimensions (so called Principal Components)

Predictors	**knn**	**random forest**	**linear regression**	**best**
2	3.847	4.308	4.386	3.317
3	4.029	4.228	4.307	2.994
4	3.945	4.315	4.410	2.858
5	4.068	4.464	4.630	2.807
6	3.641	4.195	4.423	2.505
7	3.562	4.265	4.268	2.322
8	3.842	4.388	4.334	2.455
9	3.896	4.353	4.528	2.342
10	3.834	4.327	4.401	2.240

Table 5.2: Approach 1: RMSE% of test set for different number of predictors

as predictors instead. This procedure is called Principal Component Regression (PCR) [30].

We make use of the correlation matrix again to elicit the Principal Components (PCs) which are suitable predictors for a specific indicator. Once the predictors are found we build a subset with just the predictors (i.e. PCs) and the target (i.e. an indicator) as columns and use this subset to train our models. As in Approach 1 we are evaluating the accuracy of the predictions with the error metric RMSE%.

5.2.1 Overview

In this section we first briefly list the necessary steps for this approach. Subsequently we describe how these steps are done in R in more detail. Steps 2-10 need to be performed for every indicator of the dataset that needs to be predicted (see loop in 5.1.2):

1. Impute the missing values using the regularized iterative PCA algorithm [46]. This completed dataset (i.e., without missing values) is an intermediate step in order to be able to perform the Principal Component Analysis (PCA).
2. Remove the target indicator from the dataset. This step is important because otherwise the PCs would include the target information which needs to be avoided in the context of PCR.
3. Perform PCA on the completed dataset. This results in a set of Principal Components as columns instead of the indicators.
4. Append the target indicator to the Principal Components (PCs) as additional column. The columns of the dataset are now the PCs and the target indicator.
5. Calculate the correlation matrix of this dataset.

6. Create a subset with the target indicator plus the PCs with the highest absolute correlation coefficients.
7. Select only the rows which have a value for the target indicator. Only these rows are relevant for building the model.
8. Split the dataset into a training set and a test set.
9. Train the model (KNN, linear regression and random forest) using the training set with the PCs as predictors.
10. Apply the model to the test set and compare the predicted values with the observed values.

5.2.2 Prediction

The central element of this approach is the Principal Component Analysis (PCA, see Section 2.3.5). "PCA is a useful statistical technique that has found application in fields such as face recognition and image compression, and is a common technique for finding patterns in data of high dimension." [49, p. 1]

The calculation of PCA is performed with the R package FactoMineR [26]. Since the calculation requires a complete dataset without missing values some form of imputation is needed before the PCA can be performed. A "[...] shortcoming of standard approaches to PCA is that it is not obvious how to deal properly with missing data." [46, p. 2]

The authors of the FactoMineR package have also created another package for handling missing values in PCA called missMDA. The missMDA package contains an imputation function implementing the regularized iterative PCA algorithm which "[C]an be used as a preliminary step before performing a PCA on an [sic] completed dataset." [25]

Listing 5.9 shows the code for importing the missMDA package, reading in data and imputing missing values with the regularized iterative PCA algorithm using the first ten Principal Components and three random initializations. After running the code the variable `imputed` refers to the imputed dataset which is now complete (i.e. has no missing values) and therefore suitable as input for the PCA.

```r
library(missMDA)
data <- read.csv("/path/to/data.csv")
imputed <- imputePCA(data, ncp=10, nb.init=3, scale=TRUE)
```

Listing 5.9: Imputing data with the regularized iterative PCA algorithm in R

Now that the missing values are imputed we can perform the actual PCA on our dataset. Listing 5.10 shows the code for loading the FactoMineR package and performing the PCA. The target column is deleted from the dataset before performing the PCA (see line 3). Otherwise the information which needs to be predicted would be included in the resulting Principal Components which are subsequently used as predictors in the model. This can lead to overfitting (see Section 2.3.4). Before the target column is deleted the information is stored in a variable `target` because we need the information later to calculate the error metric. In order to get meaningful results the data is scaled to have a comparable range of values.

```
1 library(FactoMineR)
2 target <- imputed[target]
3 imputed[target] <- NULL
4 result <- PCA(imputed, scale.unit=TRUE, ncp=85)
```

Listing 5.10: Performing PCA in R

The first 85 PCs are kept in the result in Listing 5.10. Typically the first few PCs capture the majority of the variance in the data [30]. The variance of a PC corresponds to its significance in explaining the values in the original dataset. We want to keep those PCs which have at least some significance in explaining the original values. All PCs explaining more than 0.0075% of the variance in the data were kept. This resulted in the first 85 PCs. In our dataset the first 10 PCs capture 96%, the first 85 PCs capture 99.86% of the entire variance in the data. Figure 5.4 shows the percentage of variance explained by the first 50 PCs.

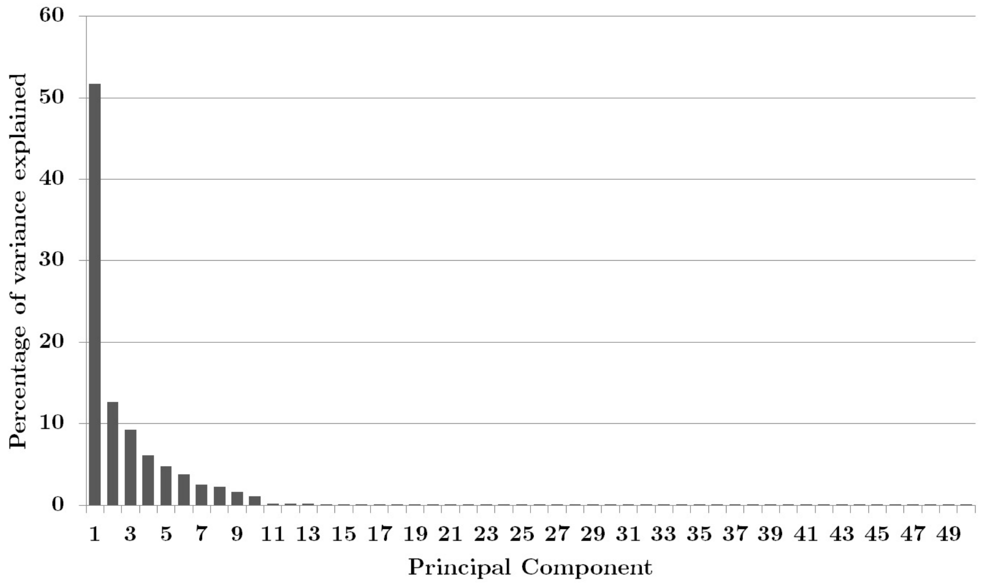


FIGURE 5.4: Variance explained by first 50 Principal Components

The first Principal Component explains more than half of the entire variance in our dataset. This translates to a high importance of this PC when it comes to predicting the individual indicators. However it does not mean that the other PCs with lower variance should be neglected in the regression. When choosing the predictors for building a regression model the relations between the target and all the PCs should be examined because it is always possible that a PC with a small variance is very important for explaining a specific indicator in the dataset [29].

It is difficult to interpret the exact meaning of a Principal Component in a high-dimensional space like ours (it is sometimes possible for lower-dimensional data, see [30, Chapter 4]). However we do have highly correlated indicators in our dataset which are generally suitable for dimensionality reduction, e.g. Deaths female and Deaths male, 1-person households and Proportion 1-person households, Unemployed Persons and Unemployment Rate etc.

The new columns of the dataset are the 85 principal components (PCs). These are a condensed representation of the original indicators which are going to be used as predictors now. This is the key difference between this approach and the approach presented in 5.1. Figure 5.5 illustrates the dimensionality reduction achieved by the PCA. The columns of the dataset are now the PCs and the target instead of the indicators and the target. The Principal Components are the new predictors.

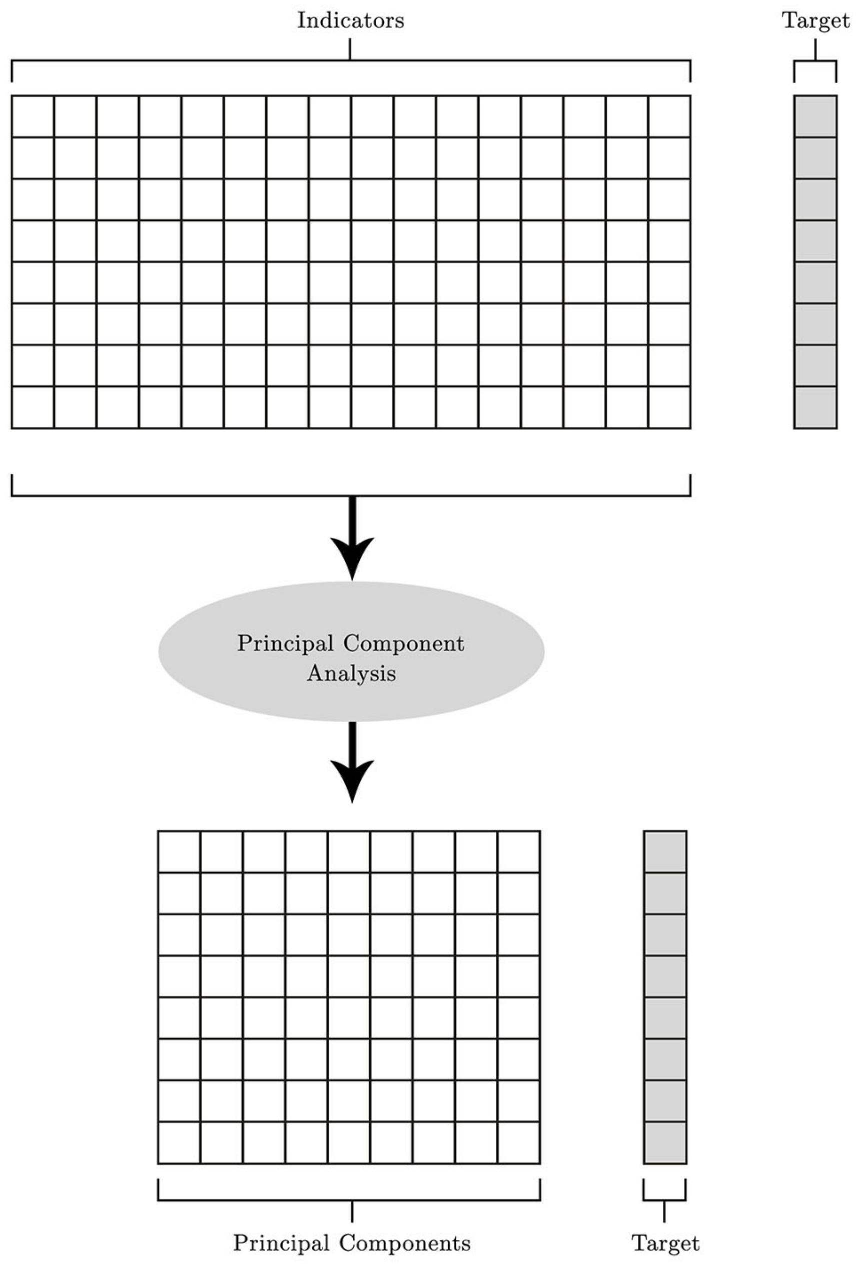


Figure 5.5: Dimensionality reduction with PCA

Building subset

Listing 5.11 shows the R code for building a subset which can be used for the prediction. The target column which was deleted from the dataset before performing the PCA is now appended again to the PCs with the R function `cbind()`. To find out which PCs are correlated with the target column and therefore suitable as predictors, the correlation matrix is calculated (see Section 5.1.2).

We are interested only in the correlation coefficients of the target column. These are stored in the vector `correlations`. To get the PCs of interest (highly positive correlation or highly negative correlations are of interest) we sort the vector `correlations` by the absolute correlation coefficient in decreasing order.

We use the function `head()` to (in this example) select the 7 correlation coefficients with the highest values. (The value 7 is just chosen in this example, in fact we experimented with different numbers of predictors, see Section 5.2.3.) These 7 values translate to the target (which has the highest possible correlation coefficient of 1 with itself) and 6 predictors. A subset with just these 7 columns is built and stored in the variable `subset`. Only the observations with a value for the target are used for building the model. The other observations are omitted with the function `complete.cases()`.

```r
data <- cbind(result , target)

corrmatrix <- cor(data)
correlations <- as.numeric(corrmatrix[target ,])
correlations <- sort(abs(correlations), decreasing = TRUE)
highest <- head(correlations , 7)

columns <- c(names(highest))
subset <- data[,columns]
subset <- subset[complete.cases(subset),]
```

Listing 5.11: Building the subset for the prediction in R

Split data into training set and test set

Now that the subset with the target and predictors is built the data needs to be split into a training set and a test set (see Section 2.3.4). The corresponding R code is the same as for the first approach and shown in Listing 5.4.

Once the training set and test are built the three regression methods presented in 2.3.2 can be applied.

K-Nearest Neighbour regression

Listing 5.1.2 shows the R code for applying the KNN regression algorithm in R (see Section 2.3.2) to impute the target column in the test set using the data in the training set, i.e. the nearest neighbours are searched for only in the training set. For the second approach, we chose a value of $k = 6$. Extensive experiments with the dataset have shown that a value of 6 for k yields the lowest RMSE%.

Linear Regression

We use R's built-in function `lm()` to apply linear regression on the training set to build the model. Listing 5.6 shows the necessary R code for building the model and using it for predicting the values of the target indicator in the test set (see Section 5.1.2).

Decision Trees

Finally our last regression method is the random forest algorithm. Listing 5.7 shows the R code to apply the algorithm to grow the tree and use it for predictions on the test set.

5.2.3 Results

Figure 5.6 shows the RMSE% for different numbers of predictors. On average over all indicators the K-Nearest Neighbour regression method works best in our dataset. For 56 predictors the average RMSE% of KNN is 3.455. There is no more improvement achieved by adding more predictors. The Random forest algorithm works also very well for lower number of predictors but starts yielding worse results for 12 predictors or more.

Remember from 5.4 that only the first PCs provide significant contributions to explaining the total variance in the data. For a larger number of predictors the algorithm needs to adjust the weight of the PCs with lower prediction usefulness. KNN seems to be very well suited for this situation. Linear regression works best with around 60 predictors (RMSE% 3.497) but the results are also getting worse for higher numbers of predictors.

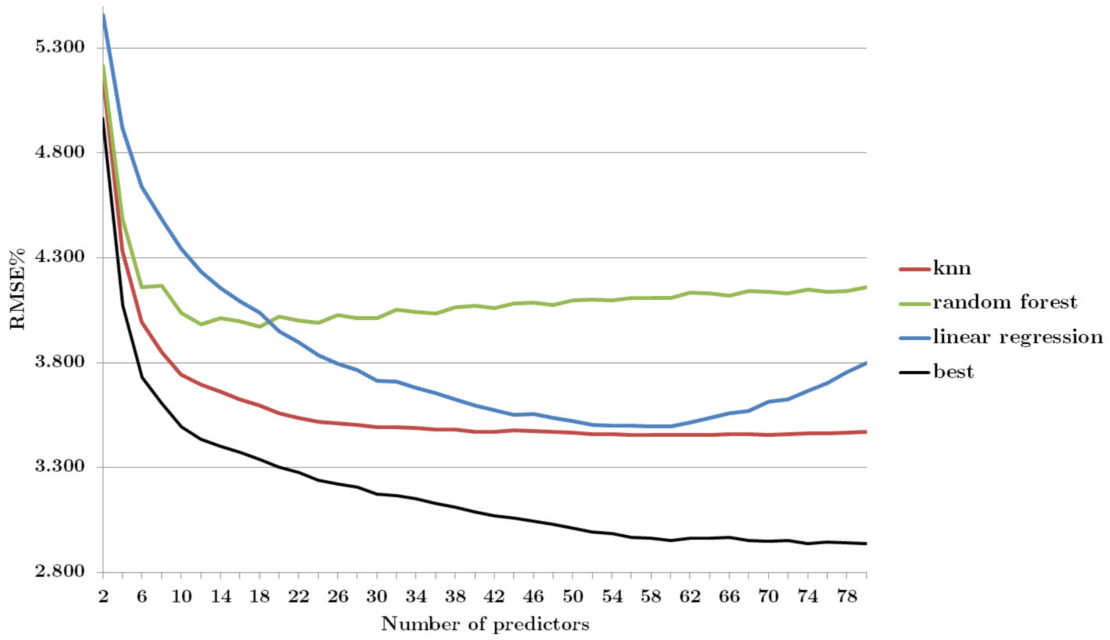


FIGURE 5.6: Approach 2: RMSE% of test set for different number of predictors

As stated above the overall results can be improved by selecting the best performing regression method for every indicator (see Section 5.1.3). The black line in Figure 5.6 shows the average RMSE% if for every indicator the regression method with the lowest error is selected. With this selection the results can be improved consistently over any single regression method. Table 5.3 shows the RMSE% values between 50 and 80 predictors.

The data shows that when selecting the best performing regression method per indicator the results can be further improved by adding more and more predictors. An RMSE% of 2.953 is achieved with 60 predictors, with 80 predictors the number goes down to 2.937 (-0.016 or -0.0008 per added predictor). The performance improvement gets smaller and smaller for every added predictor. This result is consistent with the decreasing contribution of every PC to explaining the variance in the data (see Figure 5.4).

A closer look at the results for the model with 80 predictors shows that the average RMSE% scored with KNN is 3.471, with random forest 4.161 and with linear regression 3.799. As mentioned above choosing the best method per indicator yields a lower RMSE% of 2.937. On average, KNN is performing best over all indicators. However linear regression is the best method for 88 indicators (60% of all indicators), KNN for 51 indicators (35%) and random forest (although far worse on average) is the best method for 7 indicators (5%). This result implies that linear regression works very well for a

Predictors	**knn**	**random forest**	**linear regression**	**best**
50	3.468	4.096	3.521	3.010
52	3.459	4.100	3.505	2.991
54	3.461	4.098	3.500	2.985
56	3.455	4.109	3.500	2.967
58	3.455	4.109	3.496	2.963
60	3.456	4.109	3.497	2.953
62	3.455	4.136	3.515	2.965
64	3.455	4.132	3.536	2.962
66	3.460	4.120	3.560	2.969
68	3.460	4.142	3.570	2.953
70	3.456	4.136	3.613	2.950
72	3.460	4.129	3.627	2.953
74	3.463	4.149	3.666	2.939
76	3.463	4.138	3.703	2.945
78	3.467	4.141	3.753	2.940
80	3.471	4.161	3.799	2.937

Table 5.3: Approach 2: RMSE% of test set for different number of predictors

lot of indicators but can also be very inaccurate for other indicators. Therefore it is the best method for most indicators but on average the RMSE% scored with KNN is lower.

Of the 10 indicators with the lowest RMSE% 7 are population categories (e.g. population female, population aged 15-19 male etc.). Interestingly, all of these 7 population category indicators are best predicted with linear regression. The other 3 indicators (median household income, proportion lone pensioner households and tourist bed places) are all best predicted with KNN. The RMSE% of all of these 10 indicators is below 0.400. Table 5.4 shows these 10 indicators and their RMSE (same scale as indicator) and the normalized RMSE (RMSE%). The abbreviation *linreg* stands for linear regression and *rforest* stands for random forest.

Table 5.5 shows the 10 indicators with the highest RMSE% using the best regression method per indicator. These are the indicators where the differences between the predictions of the best performing regression method and the actual observed values are the greatest (based on the range of values of the indicator). The RMSE% for these indicators is between 6.738 and 8.268.

On average there are 1679.2 values available for each of the 10 indicators with the highest RMSE% (i.e. the training set used to build the models has on average 839.6 values because half of the available values are used for the training set and the other half for the evaluation with the test set). For the indicators with the lowest RMSE% there are on average 3026 values available. This observation indicates that the prediction results are actually getting better the more data is available to build the regression

Indicator	Best regression Method	RMSE	RMSE%
Median household income	KNN	2434.69	0.100
Population 20 24 female	linreg	1705.74	0.135
Population 20 24 male	linreg	2329.62	0.174
Proportion lone pensioner households	KNN	27.78	0.195
Population female	linreg	22431.86	0.216
Population 35 44 female	linreg	2113.30	0.302
Population 15 19 male	linreg	1817.94	0.322
Population	linreg	81158.19	0.353
Population 25 34 female	linreg	3428.61	0.371
Tourist Bed Places	KNN	50422.36	0.390

Table 5.4: Approach 2: 10 indicators with lowest RMSE%

Indicator	Best regression Method	RMSE	RMSE%
Proportion living in owned dwellings	KNN	6.95	8.268
Commute duration	KNN	3.28	8.206
Days with high NO2 concentrations	KNN	5.14	8.151
Price public transport	KNN	9.80	8.080
Proportion living in houses	KNN	6.87	8.048
Proportion living in apartments	KNN	6.60	7.561
Average area of living	KNN	2.95	7.536
Infant mortality rate per 1000 live births	rforest	2.07	7.522
Avg household size	KNN	0.11	7.010
Average NO2 concentration	KNN	5.29	6.738

Table 5.5: Approach 2: 10 indicators with highest RMSE%

models. There is also a positive outlook from this: the more data on existing indicators is collected in the future in the Open City Data Pipeline, the better the predictions of the missing values are going to get.

Comparison to Approach 1

As already mentioned there are two main properties to evaluate our approaches for predicting missing values: The number of possible predictions and the prediction accuracy.

We have seen in 5.1.3 that Approach 1 is very limited with regard to the number of possible predictions. Depending on the number of predictors, between 5000 and 27000 values can be predicted in total. In Approach 2 we are imputing our dataset and we are building condensed new columns called Principal Components. This leads to a completed dataset which can be used to predict all the missing values (i.e. 311958 in total). Based on the first essential property of evaluation, the number of possible predictions, Approach 2 is highly preferable to Approach 1.

Prediction accuracy is the second important property for the evaluation. Using 7 predictors in Approach 1 the RMSE% is 2.322 on average. Approach 2 yields an average RMSE% of 2.937 for 80 predictors. Judged on the prediction accuracy alone Approach 1 is more valuable. However, considering both evaluation properties (i.e. number of possible predictions and prediction accuracy) Approach 2 should be preferred due to its ability to predict all the missing values.

Chapter 6

Conclusions, Future Work and Related Work

This chapter comprises a few concluding words summarizing this work and also suggestions for future work on improvements of the Open City Data Pipeline.

6.1 Conclusion

We presented two different approaches to predicting missing values in the context of an Open City Data Pipeline. The architecture from which the data stems has been introduced and the individual architecture components have been described. A subset from all the collected data has been built and used as input for building regression models.

In the first approach to predicting missing values we use the indicators directly as predictors in our regression models. This approach delivers an average prediction accuracy (measured with RMSE%) between 2.0 and 3.5. An RMSE% of 3.5 means that the difference between actual values and predictions is on average 3.5% of the range of values (the maximum value minus minimal value of an indicator), e.g. for our dataset an RMSE% of 3.5 means that predictions for the unemployment rate are off by 1.4% on average, for the average household income the error corresponds to € 2235. Considering the complex dataset and the high ratio of missing values this error metric is quite satisfying. However we have seen that this approach is very limited in its ability to predict many missing values. Even when building very simple models with few predictors we are only able to predict 8.8% of all missing values, because the model requires existing indicators as input and these indicators do have a lot of missing values.

The second approach uses Principal Components (PCs) as predictors instead of using indicators directly from our dataset. Using the functions of the R packages FactoMineR and missMDA ([26], [25]), we perform a Principal Component Analysis on our dataset resulting in new, condensed columns (the PCs). These columns are then used to build our regression models (Principal Component Regression, PCR).

When using models with 80 predictors and picking the best performing regression method we achieved a RMSE% of 2.937 with the second approach. The prediction accuracy is therefore slightly worse compared to the first approach where we used the indicators directly as predictors. However the results are still useful, since this approach does not only deliver satisfying prediction accuracy, but is additionally also able to predict all missing values in our dataset. Therefore the results suggest that the second approach should be preferred over the first approach for predicting missing values in the Open City Data Pipeline.

By providing the predicted values along with the transparent publication of confidence (by publishing the expected error metric), the Open City Data Pipeline becomes more useful for third party users of city data because there is more data available which eases the compilation of studies and reports. These studies and reports are essential in supporting all kinds of city stakeholders such as governments, citizens or infrastructure providers in their decision making process.

6.2 Future Work and Related Work

The Open City Data Pipeline could be improved in numerous ways. Figure 3.1 shows the architecture of the pipeline. We are currently working on the import from the regression back to the database (i.e. adding predicted values to the database). It is important to also report the confidence in the predicted value (i.e. the expected error margin) to provide transparency to third parties which can then decide for themselves whether they want to trust the specific predicted value or not.

However, every existing component of the architecture has potential for improvement. For example, there could be more data sources accessed and integrated which would lead to a richer dataset. Currently only data published by large organizations and communities is integrated. However, different types of rather unstructured data could also be evaluated and integrated. For this we need new technologies and approaches, e.g. to automatically find related data tables in a large set of tables [14].

Another area where additional work can be done is outlier detection. Due to the heterogeneous data sources it is obvious that some of the imported values in the pipeline could be incorrect. In a simple preliminary investigation we defined a corridor of accepted values by calculating the following thresholds where *sd* is the standard deviation of the indicator:

$$Median \pm 3 \times sd \tag{6.1}$$

Using this simple thresholds too many values would have been identified as outliers, hence is not suitable for our dataset. However, there are other, more sophisticated outlier detection methodologies which could be applied in future work to achieve better results [22].

Data quality could be improved in the future by applying outlier detection algorithms. A recent work on outlier detection in Linked Data is [18] which also uses data from DBpedia to test the approach. Prediction in the Open City Data Pipeline at the moment is working on exported CSV data from the triple store. There are also methods for working directly on RDF data using Tensor Factorization [39].

In future work on the prediction component one could also look at more advanced Data Mining algorithms such as Support Vector Machines or Neural Networks [60]. Some preliminary work with Singular Value Decomposition (SVD) as method for predicting the missing values did not yield satisfactory prediction results, therefore we moved on to Principal Component Analysis (PCA) in combination with three regression methods (KNN, linear regression, random forests). Generally, with this approach, there are a lot of configuration options, which have to be further investigated to improve the results, e.g., the threshold for the amount of missing data in the preprocessing phase, the number of principal components retained in the dataset, the number of predictors etc.

The concept of distance and vicinity is essential in many regression methods. Currently we are only looking at vicinity based on social, economic, demographic indicators etc. However there is also the possibility to use the spatial vicinity in the real world for predictions, testing the assumption that cities which are physically close are also similar in socio-economic areas. The necessary geospatial information (coordinates and elevation) is already collected and available.

Bibliography

[1] A. C. Acock. Working With Missing Values. *Journal of Marriage and Family*, 67(4):1012–1028, 2005.

[2] Administration of City of Vienna. Open Data Initiative of the city of Vienna. `https://open.wien.at/site/open-data/`, November 2014.

[3] Administration of New York City. Open Data Initiative of New York City. `https://data.cityofnewyork.us/`, November 2014.

[4] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A Nucleus for a Web of Open Data. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, editors, *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer Berlin Heidelberg, 2007.

[5] S. Auer, J. Lehmann, and A.-C. Ngonga Ngomo. Introduction to Linked Data and Its Lifecycle on the Web. In A. Polleres, C. d'Amato, M. Arenas, S. Handschuh, P. Kroner, S. Ossowski, and P. Patel-Schneider, editors, *Reasoning Web. Semantic Technologies for the Web of Data*, volume 6848 of *Lecture Notes in Computer Science*, pages 1–75. Springer Berlin Heidelberg, 2011.

[6] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, and G. Carothers. RDF 1.1 Turtle: Terse RDF Triple Language. Technical report, W3C, February 2014.

[7] T. Berners-Lee. Linked Data Design Issues. `http://www.w3.org/DesignIssues/LinkedData.html`, July 2006.

[8] S. Bischof and A. Polleres. RDFS with Attribute Equations via SPARQL Rewriting. In P. Cimiano, O. Corcho, V. Presutti, L. Hollink, and S. Rudolph, editors, *The Semantic Web: Semantics and Big Data*, volume 7882 of *Lecture Notes in Computer Science*, pages 335–350. Springer Berlin Heidelberg, 2013.

[9] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[10] D. Brickley and R. Guha. RDF Schema 1.1. Technical report, W3C, 2014.

[11] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: Implementing the Semantic Web Recommendations. In *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers and Posters*, WWW Alt. '04, pages 74–83, New York, NY, USA, 2004. ACM.

[12] CDP Worldwide. Carbon Disclosure Project - About Us. `https://www.cdp.net/en-US/Pages/About-Us.aspx`, January 2015.

[13] Committee ISO/TC 268 Sustainable development in communities. ISO 37120:2014 Indicators for city services and quality of life. ISO 37120.

[14] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding Related Tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 817–828, New York, NY, USA, 2012. ACM.

[15] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The Semantic Web: the roles of XML and RDF. *Internet Computing, IEEE*, 4(5):63–73, September 2000.

[16] L. Dijkstra and H. Poelman. Cities in Europe: The New OECD-EC Definition. *Regional Focus*, 1:2012, 2012.

[17] Eurostat, the Statistical Office of the European Union. Urban Audit - Reference Metadata. `http://ec.europa.eu/eurostat/cache/metadata/en/urb_esms.htm`, May 2014.

[18] D. Fleischhacker, H. Paulheim, V. Bryl, J. Völker, and C. Bizer. Detecting Errors in Numerical Linked Data Using Cross-Checked Outlier Detection. In P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. Knoblock, D. Vrandečić, P. Groth, N. Noy, K. Janowicz, and C. Goble, editors, *The Semantic Web – ISWC 2014*, volume 8796 of *Lecture Notes in Computer Science*, pages 357–372. Springer International Publishing, 2014.

[19] M. S. Fox. A foundation ontology for global city indicators - global cities institute working paper no. 03, August 2013.

[20] Global Cities Institute, University of Toronto. GCIF Indicators. `http://www.cityindicators.org/`, 2011.

[21] D. J. Hand, P. Smyth, and H. Mannila. *Principles of Data Mining.* MIT Press, Cambridge, MA, USA, 2001.

[22] V. J. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

[23] D. Hoornweg, M. Freire, F. Nunez, N. Palugyai, R. Huppman, D. Blaha, T. Robinson, C. Graham, V. Smith, G. Booth, et al. Global City Indicators Program Report Part of a Program to Assist Cities in Developing an Integrated Approach for Measuring City Performance, 2008.

[24] D. C. Howell. The analysis of missing data. *Handbook of Social Science Methodology*, 2008.

[25] F. Husson and J. Josse. R package reference manual: missMDA. `http://cran.r-project.org/web/packages/missMDA/missMDA.pdf`, December 2013. Version 1.7.2.

[26] F. Husson, J. Josse, S. Le, and J. Mazet. R package reference manual: FactoMineR. `http://cran.r-project.org/web/packages/FactoMineR/FactoMineR.pdf`, August 2014. Version 1.27.

[27] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer New York, 2014.

[28] K. Janowicz, S. Scheider, and B. Adams. A Geo-semantics Flyby. In S. Rudolph, G. Gottlob, I. Horrocks, and F. van Harmelen, editors, *Reasoning Web. Semantic Technologies for Intelligent Data Access*, volume 8067 of *Lecture Notes in Computer Science*, pages 230–250. Springer Berlin Heidelberg, 2013.

[29] I. T. Jolliffe. A note on the use of principal components in regression. *Applied Statistics*, pages 300–303, 1982.

[30] I. T. Jolliffe. *Principal Component Analysis, Second Edition.* Springer-Verlag New York, Inc., 2002.

[31] J. Josse, M. Chavent, B. Liquet, and F. Husson. Handling Missing Values with Regularized Iterative Multiple Correspondence Analysis. *Journal of Classification*, 29(1):91–116, 2012.

[32] J. Josse and F. Husson. Handling missing values in exploratory multivariate data analysis methods. *Journal de la Société Française de Statistique*, 153(2):79–99, 2012.

[33] G. Klyne and J. J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. W3C Recommendation, 2004. *World Wide Web Consortium, http://w3c. org/TR/rdf-concepts*, 2004.

[34] G. Klyne, J. J. Carroll, and B. McBride. RDF 1.1 Concepts and Abstract Syntax. Technical report, W3C, February 2014.

[35] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, August 2009.

[36] A. Liaw and M. Wiener. R package reference manual: randomForest. `http://cran.r-project.org/web/packages/randomForest/randomForest.pdf`, July 2014. Version 4.6-10.

[37] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language - Overview. Technical report, W3C, February 2014.

[38] A. Miles and S. Bechhofer. Skos simple knowledge organization system reference. *W3C recommendation*, 18:W3C, 2009.

[39] M. Nickel. Tensor factorization for relational learning. `http://nbn-resolving.de/urn:nbn:de:bvb:19-160568`, August 2013.

[40] Office for Official Publications of the European Communities. Urban Audit. Methodological Handbook. 2004.

[41] A. Polleres. Lecture Notes: City Data Pipeline. `https://ai.wu.ac.at/~polleres/presentations/`, March 2014.

[42] A. Polleres. Lecture Notes: Linked (Open) Data for IoT. `https://ai.wu.ac.at/~polleres/presentations/`, April 2014.

[43] A. Polleres, S. Bischof, and H. Schreiner. City Data Pipeline - A report about experiences from using Open Data to gather indicators of city performance. In *Proceedings of European Data Forum 2014*, Athens, Greece, March 2014.

[44] J. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986.

[45] T. Rosenberg. Armed With Data, Fighting More Than Crime. `http://opinionator.blogs.nytimes.com/2012/05/02/armed-with-data-fighting-more-than-crime/`, May 2012.

[46] S. Roweis. EM Algorithms for PCA and SPCA. In *in Advances in Neural Information Processing Systems*, pages 626–632. MIT Press, 1998.

[47] M. Schmachtenberg, C. Bizer, A. Jentzsch, and R. Cyganiak. Linking Open Data cloud diagram 2014. `http://lod-cloud.net/`, August 2014.

[48] D. Skillicorn. *Understanding Complex Datasets: Data Mining with Matrix Decompositions*. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series. Taylor & Francis, 2007.

[49] L. I. Smith. A tutorial on Principal Components Analysis. `http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf`, 2002.

[50] The Apache Software Foundation. Apache Jena Documentation. `http://jena.apache.org/`, October 2014.

[51] The Open Knowledge Foundation. Open Definition. `http://opendefinition.org/od/`, November 2014.

[52] The W3C SPARQL Working Group. SPARQL 1.1 Overview. W3C Recommendation. Technical report, W3C, March 2013.

[53] C. A. Tompkins. Using and interpreting linear regression and correlation analyses: Some cautions and considerations. *Clinical Aphasiology*, 21:35–46, 1992.

[54] L. Torgo. *Data Mining with R: Learning with Case Studies*. Chapman & Hall/CRC, 1st edition, 2010.

[55] L. Torgo. R package reference manual: DMwR. `http://cran.r-project.org/web/packages/DMwR/DMwR.pdf`, August 2013. Version 0.4.1.

[56] United Nations, Department of Economic and Social Affairs, Population Division. World Urbanization Prospects: The 2014 Revision, Highlights. 2014.

[57] United Nations Statistics Division. Demographic Yearbook 2012. `http://unstats.un.org/unsd/demographic/products/dyb/dyb2012.htm`, 2012.

[58] C. J. Willmott. On the validation of models. *Physical geography*, 2(2):184–194, 1981.

[59] World Wide Web Consortium and others. OWL 2 web ontology language document overview. 2012.

[60] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. McLachlan, A. Ng, B. Liu, P. Yu, Z.-H. Zhou, M. Steinbach, D. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.

Appendix A

List of indicators

This table contains a list of all the indicators in the dataset described in Chapter 4 including the unit and the category of each indicator.

Indicator	Unit	Category
Number of available beds per 1000 residents	no	Culture and Recreation
Number of bed-places in tourist accomodation establishments	no	Culture and Recreation
Number of cinema seats	no	Culture and Recreation
Number of cinema seats per 1000 residents	no	Culture and Recreation
Number of museum visitors per year	persons	Culture and Recreation
Number of public libraries	no	Culture and Recreation
Number of theatres	no	Culture and Recreation
Number of tourist overnight stays per year per resident	no	Culture and Recreation
Total nights spent in tourist accomodation establishments	no	Culture and Recreation
1-person households total	no	Demography
Average size of households (number of persons)	persons	Demography

Indicator	Unit	Category
EU Foreigners as a proportion of the total population	%	Demography
EU Foreigners total	persons	Demography
Households with children aged 0-17 total	no	Demography
Households with lone-pensioners above retirement age	no	Demography
Lone parent private households with children aged 0-17 total	no	Demography
Median Population Age	years	Demography
Nationals as a proportion of the total population	%	Demography
Nationals total	persons	Demography
Non-EU Foreigners as a proportion of population	%	Demography
Non-EU Foreigners total	persons	Demography
Number of Women per 100 Men	no	Demography
Number of Women per 100 Men aged 75 years and over	no	Demography
Old Age Dependency Ratio (Population aged 65+ to population aged 20-64)	%	Demography
Population aged 0-4 female	persons	Demography
Population aged 0-4 male	persons	Demography
Population aged 0-4 total	persons	Demography
Population aged 15-19 female	persons	Demography
Population aged 15-19 male	persons	Demography
Population aged 15-19 total	persons	Demography
Population aged 20-24 female	persons	Demography
Population aged 20-24 male	persons	Demography
Population aged 20-24 total	persons	Demography
Population aged 25-34 female	persons	Demography
Population aged 25-34 male	persons	Demography
Population aged 25-34 total	persons	Demography
Population aged 35-44 female	persons	Demography
Population aged 35-44 male	persons	Demography

Indicator	**Unit**	**Category**
Population aged 35-44 total	persons	Demography
Population aged 45-54 female	persons	Demography
Population aged 45-54 male	persons	Demography
Population aged 45-54 total	persons	Demography
Population aged 55-64 female	persons	Demography
Population aged 55-64 male	persons	Demography
Population aged 55-64 total	persons	Demography
Population aged 65-74 female	persons	Demography
Population aged 65-74 male	persons	Demography
Population aged 65-74 total	persons	Demography
Population aged 75 and over female	persons	Demography
Population aged 75 and over male	persons	Demography
Population aged 75 and over total	persons	Demography
Population change over 1 year	%	Demography
Population female total	persons	Demography
Population living in private households (excluding institutional households)	persons	Demography
Population male total	persons	Demography
Population total	persons	Demography
Private households (excluding institutional households)	no	Demography
Proportion of 1-person households	%	Demography
Proportion of households that are lone-parent households	%	Demography
Proportion of households that are lone-pensioner households	%	Demography
Proportion of households with children aged 0-17	%	Demography
Proportion of total population aged 0-4	%	Demography
Proportion of total population aged 15-19	%	Demography
Proportion of total population aged 20-24	%	Demography
Proportion of total population aged 25-34	%	Demography
Proportion of total population aged 35-44	%	Demography

Indicator	**Unit**	**Category**
Proportion of total population aged 45-54	%	Demography
Proportion of total population aged 55-64	%	Demography
Proportion of total population aged 65-74	%	Demography
Proportion of total population aged 75 and over	%	Demography
Activity Rate (Proportion of economically active people)	%	Economic Aspects
Activity Rate male (Proportion of economically active female people)	%	Economic Aspects
Activity Rate male (Proportion of economically active male people)	%	Economic Aspects
Average disposable annual household income	EUR	Economic Aspects
Economically Active Population female	persons	Economic Aspects
Economically Active Population male	persons	Economic Aspects
Economically Active Population total	persons	Economic Aspects
Employment divided by Jobs	%	Economic Aspects
Employment Jobs in agriculture and fishery (NACE Rev. 2: A)	no	Economic Aspects
Employment Jobs in construction (NACE Rev. 2: F)	no	Economic Aspects
Employment Jobs in mining manufacturing energy (NACE Rev. 2: B-E)	no	Economic Aspects
Median disposable annual household income	EUR	Economic Aspects
Number of companies	no	Economic Aspects
Number of unemployed persons female	persons	Economic Aspects
Number of unemployed persons male	persons	Economic Aspects
Number of unemployed persons total	persons	Economic Aspects
Proportion of employment in agriculture and fishery	%	Economic Aspects
Proportion of employment in construction (NACE Rev. 2: F)	%	Economic Aspects
Proportion of employment in industries (NACE Rev.1.1 C-E)	%	Economic Aspects

Indicator	**Unit**	**Category**
Unemployment Rate	%	Economic Aspects
Unemployment rate female	%	Economic Aspects
Unemployment rate male	%	Economic Aspects
Accumulated ozone concentration	microgram per m3	Environment
Annual average concentration of Nitrogen dioxide (NO2) in microgram per cubic meter	microgram per m3	Environment
Annual average concentration of Particulate Matter (PM10) in microgram per cubic meter	microgram per m3	Environment
Number of days with Nitrogen dioxide (NO2) concentrations exceeding 200 micrograms per cubic meter	days	Environment
Number of days with Ozone (O3) concentrations exceeding 120 micrograms per cubic meter	days	Environment
Number of days with Particulate Matter (PM10) concentrations exceeding 50 micrograms per cubic meter	days	Environment
Percentage of the urban waste water load (in population equivalents) treated according to the applicable standard	%	Environment
Price of a cubic meter (m3) of domestic water	EUR	Environment
Share of population connected to sewerage treatment system	%	Environment
Total use of water in cubic meter (m3)	cubic meter	Environment
Average annual rent for housing per square meter (m2)	EUR	Social Aspects
Average area of living accomodation (in square meters per person)	m2	Social Aspects

Indicator	**Unit**	**Category**
Crude birth rate per 1000 inhabitants	persons/1000 inhabitants	Social Aspects
Crude death rate per 1000 inhabitants	persons/1000 inhabitants	Social Aspects
Households in private rented housing	no	Social Aspects
Households in social housing	no	Social Aspects
Households owning their own dwelling	no	Social Aspects
Infant mortality rate (per 1000 live births)	persons/1000 live births	Social Aspects
Infant mortality rate (per year)	%	Social Aspects
Number of apartments	no	Social Aspects
Number of conventional dwellings	no	Social Aspects
Number of deaths per year under 65 due to diseases of the circularoty or respiratory systems	no	Social Aspects
Number of households living in apartments	no	Social Aspects
Number of households living in houses	no	Social Aspects
Number of houses	no	Social Aspects
Number of live births per year	no	Social Aspects
Number of murders and violent deaths	no	Social Aspects
Proportion of households living in apartments	%	Social Aspects
Proportion of households living in houses	%	Social Aspects
Proportion of households living in owned dwellings	%	Social Aspects
Total deaths per year	no	Social Aspects
Total deaths per year female	no	Social Aspects
Total deaths per year male	no	Social Aspects
Total deaths under 65 per year	no	Social Aspects
Total deaths under 65 per year female	no	Social Aspects
Total deaths under 65 per year male	no	Social Aspects
Number of children aged 0-4 in day care (public and private) per 1000 children aged 0-4	persons/1000 persons	Training and Education

Indicator	**Unit**	**Category**
Number of children aged 0-4 in day care or school	persons	Training and Education
Persons aged 25-64 with ISCED level 0 1 or 2 as the highest level of education	persons	Training and Education
Persons aged 25-64 with ISCED level 3 or 4 as the highest level of education	persons	Training and Education
Proportion of working age population qualified at ISCED level 5 or 6	%	Training and Education
Students in higher education (ISCED level 5-6) female	persons	Training and Education
Students in higher education (ISCED level 5-6) male	persons	Training and Education
Students in higher education (ISCED level 5-6) total	persons	Training and Education
Average length of journey to work by private car in km	km	Travel and Transport
Cost of a combined monthly ticket (all modes of public transport for 5-10 km in the central zone)	EUR	Travel and Transport
Cost of a taxi ride of 5km to the center at day time	EUR	Travel and Transport
Length of bicycle network (dedicated cycle paths and lanes)	km	Travel and Transport
Number of deaths in road accidents	no	Travel and Transport
Number of private cars registered	no	Travel and Transport
Number of registered cars per 1000 population	no	Travel and Transport
People commuting into the city	persons	Travel and Transport
People commuting out of the city	persons	Travel and Transport
People killed in road accidents per 10000 population	persons/10000 persons	Travel and Transport