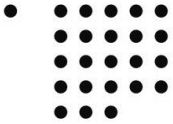


Scalable Geospatial Analytics with IBM DB2 and R

Master thesis
presented to Cologne University of Applied Sciences
in fulfilment of the requirements
for the degree of Master of Engineering in Automation & IT

Author:
Sumit Goyal

First examiner:
Prof. Dr. Heide Faeskorn-Woyke

Second examiner:
Dr. Michael Wurst

June 2015

COLOGNE UNIVERSITY OF APPLIED SCIENCES

Fakultät für Informatik und Ingenieurwissenschaften

Abstract

Scalable Geospatial Analytics with IBM DB2 and R

Today's highly computerized world generates a huge amount of data every moment. A large proportion of this data has an associated geospatial component. Geospatial analysis significantly enhances the classical data analysis and introduces unforeseen opportunities. But scalable methods are required to deal with enormous amounts of data. This study summarizes the theoretical foundations of geospatial analysis in a concise form. Concepts and principals relating to the geospatial data and analysis, and geospatial databases are furnished. A scalable solution is developed for geospatial analysis using statistical data analysis tool R and IBM DB2® Spatial Extender. The solution exploits the best features of both the tools. It utilizes easy to use, free, and open source R as the interface for geospatial operations and employs DB2® Spatial Extender with its performance enhancing features for in-database functionality. The developed interface aims to conceal the challenging features of geospatial data structure and analytics, and offers an intuitive and convenient platform. The acquired knowledge and the developed scalable solution is then exercised upon several practical problems and use-cases.

Acknowledgements

I would like to express my appreciation and thanks to my thesis adviser Prof. Dr. Heide Faeskorn - Woyke, for her sincere and valuable guidance and encouragement. I am grateful to my second adviser Dr. Michael Wurst, for sharing his expertise and dedicated involvement in every step throughout the process. Your advice on both the thesis and on my career have been priceless. I would especially like to thank Mr. Mathias Trumpp for his vital support in the process and for the diligent work review.

Words cannot express how grateful I am to my mother, father and brother for all of the sacrifices that you've made on my behalf. I will always be in a debt of gratitude for your unceasing support. I would also like to thank all of my friends and colleagues who supported me in writing the thesis, and encouraged me to strive towards my goal.

Contents

List of Figures

List of Tables

Abbreviations

DBMS	**D**atabase **M**anagement **S**ystem
DMS	**D**egrees, **M**inutes and **S**econds
GCS	**G**eographic **C**oordinate **S**ystem
GIS	**G**eographic **I**nformation **S**ystem
GML	**G**eography **M**arkup **L**anguage
GPS	**G**lobal **P**ositioning **S**ystem
LAT	**L**ocation **A**ware **T**echnology
MBR	**M**inimum **B**ounding **R**ectangle
OGC	**O**pen **G**eospatial **C**onsortium
RFID	**R**adio- **F**requency **I**dentification
SDBMS	**S**patial **D**atabase **M**anagement **S**ystem
SRS	**S**patial **R**eference **S**ystem
WKB	**W**ell **K**nown **B**inary
WKT	**W**ell **K**nown **T**ext
WWW	**W**orld **W**ide **W**eb

Chapter 1

Introduction and Motivation

1.1 Introduction

Answers to the questions: Where are we? Where is something happening? is a very basic need of the ever increasingly urbanized world. Information pertaining to location of objects or events is constantly present in our thinking. We contemplate these details in most of our daily activities and act accordingly. This information (Geospatial information) has enormous capabilities to revamp our decision making and hence paves the way for Geospatial Analytics. Geospatial Analytics is the technique of analyzing spatial data (data that have a geographical component) with the aim to extract previously unknown and potentially useful information.

The last few years have observed tremendous growth in the field of digital computing and computerized data acquisition. Most modern day transactions are computerized. Advanced data collection tools like barcode scanners for commercial products, digital cameras, scientific simulators and satellite remote sensing systems are available. In addition, a magnification in the usage of World Wide Web (WWW) as a global information system is observed. Consequently, we are capable of collecting vast amounts of data on a daily basis [1]. A large proportion of this data has an associated spatial component. This data provides us with an unprecedented opportunity to achieve deeper and faster insights, extract new, insightful information and formulate knowledge.

[1] Large Hadron Collider (LHC) alone produces 30 Petabytes (30 million Gigabytes) of data annually at European Organization For Nuclear Research (CERN) on the Franco-Swiss border [1].

There are various sources of spatial data like Earth observation satellites, census databases, weather and climate databases, etc. Location-aware technologies (LATs) like cell phones and in-vehicle navigation systems which have the capability to capture data on individual movement patterns, are abundant. Satellite images are prominent source of spatial data. Moreover, current tracking technologies like Global Positioning System (GPS) and Radio-Frequency Identification (RFID) are also leading to collection of large spatio-temporal datasets. Widespread use of social networks also generate massive amounts of data. For example, approximately 500 million[2] tweets are sent per day and approximately 20% of them divulge location information with geotagging or metadata [2–4]. These datasets are fundamental in many application contexts and hence recently a lot of interest has risen towards geospatial analysis [5, 6].

Discerning spatial data might be relatively more demanding than relational data. This is because of the huge size of spatial databases and the complexity of possible patterns that can be found [7]. Nevertheless, the need for accounting the spatial features of data has been long recognized in various fields like ecology, geology, epidemiology, defense, social sciences, medicine and public safety [8]. As 80% of the data is believed to have associated spatial component, it is inevitable to consider spatial characteristics in data mining and knowledge discovery [7].

The derived information has the capability of transforming business decisions, boosting efficiency and improving business strategy. This knowledge is advantageous not only in business environments but also in other fields like medicine, social sciences, government, etc. Geospatial data gives us a chance to understand complex geographic phenomena such as human-environment interaction, social-economic dynamics and global climate change [9]. However, the bulk of data is so overwhelming that it is beyond human capabilities to manually interpret it. Scalable techniques and automated tools are hence required to extract the essence of information stored and the discovery of patterns in raw data [10].

[2] Figure as per official blog from Twitter [2] as of May 2015.

1.2 Motivation

The task of knowledge discovery in spatial datasets is considerably challenging. Spatial datasets are usually bigger and more complex than common affair datasets. The user needs to take the spatial-referenced relationship among entities into account. Moreover, the structure of the spatial data is further complex than ordinary relational database tables. The data can take many forms as it also has to account for positional data along with the attribute data [7]. In addition, the general statistical visualization techniques are often not suitable for the results of spatial analysis tasks. Several Geographic Information Systems (GIS) are available which solve some of the problems but the user has to invest in a extra set of software products and training. Some open source solutions are also available but they are usually not scalable. All of these challenges might seem daunting for an individual with simple spatial analysis tasks in mind. The motivation behind this thesis is to meet the following three objectives:

- Summarize the theoretical foundations of geospatial analysis discipline in a compact and concise manner.
- Develop an easy to use R interface for scalable geospatial analysis using DB2® Spatial Extender at the back-end. The interface aims to conceal the challenging features of spatial data structure and analysis, and provide a scalable solution.
- Exercise the acquired knowledge and the scalable solution on practical problems and use cases.

1.3 Geospatial analysis in day-to-day life

One of the famous documented examples of geospatial analysis is about the study of cholera deaths in London by Dr. John Snow. The Broad Street Cholera Outbreak was a severe outbreak of cholera that occurred in Soho district of London, England in 1854. Dr. Snow, a physician, was unable to convince other scientists and doctors that cholera is a water-borne disease and water contaminated by sewage is the prime source of the disease. Later, he used some proto-GIS methods and charted the deaths

from the outbreak to prove his point [11]. The original map drawn by him showing the clusters of cholera cases during the cholera outbreak is presented in Figure 1.1.

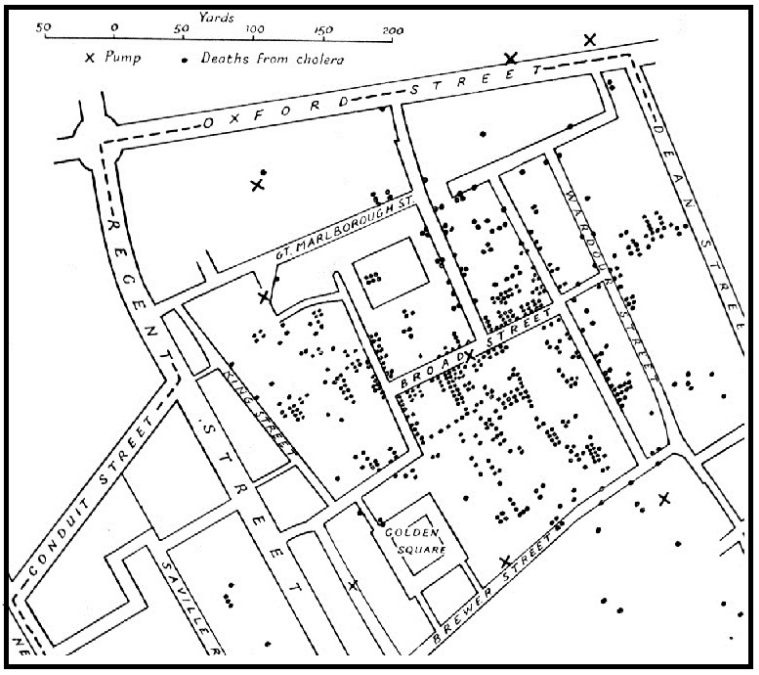


Figure 1.1: *Original map by Dr. John Snow showing the clusters of Cholera outbreak of 1854. Cholera cases are highlighted in black. A majority of Cholera deaths can be located in the vicinity of the Broad Street pump. This strengthens Dr. Snow's postulate that cholera is a water-borne disease [12].*

Dr. Snow drew Thiessen [3] polygons around the wells in the area and found out that a majority of deaths fell in the Thiessen polygon surrounding the Broad Street pump. An even larger proportion of the cholera deaths fell within the shortest travel distance around the Broad Street Pump. It was discovered later that the public well was infected from the fecal bacteria leakage from a close-by cesspit. Dr. Snow is today considered to be the pioneer of public heath research in a field known as epidemiology[14].

In the last decades geospatial analytics has developed and evolved. It has diverse applications and serve us in many ways. Crime mapping is a modern application of geospatial analytics. It involves the manipulation and processing of geospatially referenced crime data. Information regarding the crime patterns and the location of hotspots or high volume crime can be extracted through crime mapping. The results of the analysis can then be used to suppress and investigate criminal activities [15].

[3] Thiessen polygons are used to define an area of influence around a sample point. Any location inside the polygon is closer to that point than any of the other sample points. Thiessen polygons are named for the American meteorologist Alfred H. Thiessen (1872-1931)[13].

Geospatial analysis is successfully used in other fields as well, such as the analysis of urban structure and dynamics, smart cities, urban transportation systems etc.

1.4 Thesis structure

In Section 1.2, we presented the motivation and aim of the thesis. We also talked about the influence of geospatial analysis in our daily lives. Chapter 2, presents concepts and principles related to geospatial analysis domain. The chapter aims to provide the reader with a basic understanding of the spatial analysis discipline, including the issues and complexities in geospatial analysis. An introduction to spatial databases is presented in Section 2.5. A detailed description of the spatial database management system offered by IBM i.e. DB2® Spatial Extender is presented in Section 2.6. The statistical data analysis tool R is presented in Section 2.8. Section 2.9 explains the noted shape file format. In Chapter 3, we focus on developing the scalable solution for geospatial analysis. Section 3.1, introduces the reader to the ibmdbR package. The design goals of the spatial solution are defined in Section 3.2. Section 3.3, elaborates on the design of the spatial operations. The spatial operations build during the study are then explained in Section 3.4. The acquired knowledge and the functionality of the package is exercised over some real life use cases in Chapter 4. Chapter 5, presents the conclusion of the study.

Chapter 2

Geospatial Analysis Concepts and Principals

2.1 Geospatial data and information

Geospatial data, which is also known as spatial data, contains information about the location, shape and size of an object on planet earth, on other cosmic bodies or in virtual world environments of computer games. These objects have an associated spatial component and occur on the continuous surface [16]. Geospatial data is usually stored as coordinates and topology. The term spatial can also be used in the context as the concepts can be applied to data arrayed in any space, not only the geographic space. Hence, we use the terms 'Geospatial' and 'Spatial' interchangeably during this study. One important feature of spatial data is that in addition to the positional information and attributes, it also includes the spatial relationships among the entities. Embedding the data within some formal space generates implicit relationship among objects. Spatial data objects are often referenced as geographical features. Anything in the real world that has an identifiable location or can be imagined to have an identifiable location is a geographical feature [17]. Examples of graphical features:

- An object; for example, a lake, airport, island.
- A space; for example, area affected from a flood, service area of a food-chain, a state county.

- An incident at a location; for example, a taxi pickup or dropoff location.

Information derived from the data having a spatial component is referred to as spatial information. It is often facts and figures about the location, shape, size and interaction of geographic features. Spatial information alone or in combination with traditional relational data can provide valuable insights. Spatial information can be:

- Location of geographic features on the map (for example a taxi pickup location). This information is generally recorded in the form of latitudes and longitudes.
- Measurements related to geographic features (for example the distance covered in a taxi trip, or the area of a country).
- Interaction between geographic features (for example which taxi trips originated in a specific county or, which mountain ranges fall in a country).
- Relative location of geographic features (proximity of a location to flood zones).

2.2 Spatial analysis and its types

Spatial analysis is the process of examining the geographical features, their properties and relationships. It is conducted with the aim to extract potentially utilitarian and formerly unknown information. Spatial analysis utilizes statistical, overlay, modeling and other analytical techniques to reveal, predict or gain this knowledge in an efficient manner. It is an interactive and iterative process, involving several steps such as data selection, data reduction, data mining and the evaluation of data mining results. Classifying spatial analysis in a exclusive and exhaustive manner is difficult. This is because a large number of fields of research are involved and the data itself can take many forms.

Several fields of science have contributed to the spatial analysis domain and helped it to rise to its modern form. Early attempts at cartography and surveying are among the major contributors. In addition, botanical studies of global plant distributions, ecological studies of spatial population dynamics, epidemiology and spatial statistics have greatly contributed. Advancements in computational geometry and mathematics

provide fundamental tools for analysis. Scientific modeling supplies a useful framework for new approaches.

Large tables of spatial data obtained through census and surveys is used in urban and regional studies. Huge amount of detailed information is often simplified to extract main trends. Spatial autocorrelation is a common type of spatial analysis and measures the degree of dependency among observations in a geographic space. Moran's I, Geary's C, Getis's G and the standard deviational ellipse are classic spatial autocorrelation statistics. A more positive than expected spatial autocorrelation indicates the clustering of similar values across geographic space. Whereas, a negative spatial autocorrelation indicates that the neighboring values are more dissimilar than expected by chance.

Estimating the variables at unobserved locations in a geographic space based on the known value at observed locations is a commonly used type of spatial analysis. This is known as spatial interpolation. Kriging and inverse distance weighting are basic methods of spatial interpolation. Based on the origin propulsive variables, spatial interaction is used to estimate the flow of people, material or information between locations in geographic space. Computation method such as neural networks can also be used for the purpose. Spatial regression methods are used to provide information on spatial relationship among the involved variables.

2.3 Fundamental issues and common errors in spatial analysis

There are several fundamental issues in spatial analysis relating to the analytic operations to be used, definition of its objects of study, in the presentation of analytic results etc. Moreover, mathematics of space and ways of spatial data representation give rise to common errors. Many of these issues are active subjects of modern research.

- **Spatial dependency:** Spatial dependency refers to the co-variation of properties within geographic space. Spatial dependence is measured as the existence of statistical dependence in a collection of random variables. It is observed that

the characteristics at proximate locations are either positively or negatively correlated. This leads to spatial autocorrelation problem in statistics. Standard statistical techniques assume the observations to be independent but spatial dependency violates this assumption.

- **Scaling:** In spatial statistics there is no widely agreed upon scale independent method of analysis and spatial measurement scale remains to be a persistent issue. The results of statistical hypothesis tests are influenced by the choice of areal units [1].

- **Length:** Length of features in ecology measurements depend directly on the scale at which they are measured and experienced. So the measured length of geographical features like rivers, streets, et cetera, is highly context dependent [19].

- **Locational fallacy:** Locational fallacy refers to the errors generating from the particular spatial characterization chosen for the elements of study. This may implicitly limit the subject of study. For simplification purposes its common to reduce the existence of features to a single point, but the assumption might lead to wrong results [19].

2.4 Coordinate systems

The relative location of objects in a given area is defined using a framework called coordinate system. While dealing with spatial data, it is necessary to identify the coordinate system on which the data is based. It is a system which uses one or more numbers to uniquely determine the location of a geometric element on a manifold such as Euclidean space. When referring to geographic space, the coordinate system indicates the location of spatial features relative to earth's surface [20]. There are two main types of coordinate systems: geographic coordinate systems and projected coordinate systems.

[1] Stan Openshaw lamented that "the areal units (zonal objects) used in many geographical studies are arbitrary, modifiable, and subject to the whims and fancies of whoever is doing, or did, the aggregating." [18]

- **Geographic coordinate systems**

  A coordinate system which uses a three-dimensional spherical surface to identify locations on earth is called a geographic coordinate system (GCS). A GCS uses two angles, latitude and longitude to define a point on the globe [2]. The angles longitude and latitude are measured from the center of the earth to a point on the earth's surface. It includes an angular unit of measure, a prime meridian, and a datum [3] (based on a spheroid). Figure 2.1 shows a geographic coordinate system where a location is represented by the coordinates- longitude 60 degree East and latitude 55 degree North.

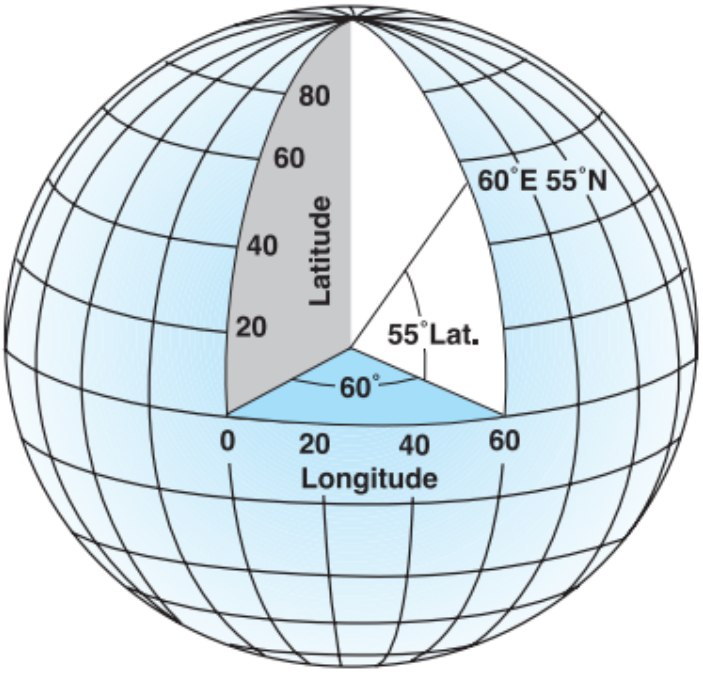


Figure 2.1: *A geographical coordinate system, indicating the location of a feature on earth's surface by the coordinates longitude 60 degree East and latitude 55 degree North. Longitude and latitude values are usually measured in decimal degrees or in degrees, minutes and seconds (DMS) [17].*

The east-west or horizontal lines are lines of equal latitude and are called parallels. The north-south or vertical lines are lines of equal longitude and are called meridians. These lines encompass the globe and form a grid network called a graticule. The line of zero latitude is called the equator and similarly the line of

[2] More precisely, since the earth is not a perfect sphere, these angles indicate the point on a ellipsoid which approximately fits the globe.

[3] A datum provides a frame of reference for measuring locations on the surface of earth.

zero longitude is called the prime meridian. Longitude and latitude values are usually measured in decimal degrees or in degrees, minutes and seconds (DMS). A spheroid or sphere defines the shape and size of a GCS whereas a datum defines the position of the spheroid relative to the center of earth [21].

- **Projected coordinate systems**

  A projected coordinate system is defined on a flat, two-dimensional surface. This coordinate system is always based on a geographic coordinates system but it uses linear units of measure for coordinates. It has the advantage that lengths, angles, and areas are constant across the two dimensions.

  The positions are identified by x, y coordinates on a grid, with the origin at the center of the grid. The x-coordinates specify the horizontal position and the y-coordinates specify the vertical position. The coordinates at the origin are x=0 and y=0. A map projection is the transformation of earth's three-dimensional surface to a flat map sheet. There are various kinds of map projections like conformal projections, equal area projections, equidistant projections etc., each designed for specific purposes.

2.5 Spatial databases

A spatial database management system (SDBMS) manages spatial data, defines spatial data types and stores spatial data. They are optimized to access and process the data that represents objects in a geometric space. Data in spatial databases are stored as coordinates, points, lines, polygons and topology. They provide special functions and indexes for querying and manipulating spatial data. Spatial databases use separate spatial indexes instead of normal indexes to speed up database operations. In addition to typical SQL queries, spatial databases support a wide variety of spatial operations. Open Geospatial Consortium (OGC) [4] specifies many such operations. For example:

[4] Open Geospatial Consortium is an international voluntary consensus standards organization which encourages development and implementation of open standards for geospatial content and services, GIS data processing and data sharing.

- **Constructor functions:** are used for creating new geometries by specifying the vertices of the shape. They are also used for converting geometries to and from various data exchange formats.
- **Comparison functions:** are used to compare geometries for intersection, boundaries and other information.
- **Observer functions:** are used to fetch specific information about a feature, such as the centroid of the geometry.
- **Spatial measurements:** return information like the area of the polygon, length of the line, distance between two geometries, etc.

Often a spatial database management system is a software module that works with an underlying database management system (DBMS). They inherit the traditional functionality provided by a DBMS. There is an assortment of spatial databases with varying range of capabilities available in the market, like DB2® Spatial Extender, Oracle Spatial, Microsoft SQL Server, PostgreSQL, CouchDB, MySQL, MongoDB, etc. DB2® Spatial Extender for Linux, UNIX and Windows v10.5.400.197 Fix Pack 4 (in the following abbreviated to DB2 Spatial Extender) is used in this study as the underlying spatial database management system. It provides the user with various performance-enhancing features, such as the efficient spatial indexes and parallel processing. A detailed description of the spatial capabilities of DB2 Spatial Extender is provided in Section 2.6.

2.6 DB2 Spatial Extender

DB2 Spatial Extender is an integrated feature of DB2 database software. It provides us with advanced spatial resources like spatial datatypes, reference systems, functions and stored procedures. The spatial datatypes are used to represent geometries such as points, lines and polygons. Spatial functions and features are available that interoperate with those datatypes. With this capability we can generate, analyze & exploit spatial information about geographic features. This allows us to add another element of intelligence to our database.

For recent DB2 versions (version > 9.7), the spatial extender comes with the standard DB2 installation image. In order to install it as a part of DB2, a custom install must be performed and the Spatial Extender must be added as an additional feature. It is required to create an environment that supports spatial operations. Installing Spatial Extender, embeds a GIS into the DB2 database. The types and functions implemented in DB2 Spatial Extender comply with the Open Geospatial Consortium[5] (OGC) and International Organization for Standardization (ISO) SQL/MM specifications [17].

2.7 Spatial resources supplied by DB2 Spatial Extender

We require resources to create and manage spatial columns and analyze spatial data. DB2 Spatial Extender provides the resources once the database is enabled for spatial operations. These resources are:

- Spatial data types to define different types of spatial data.
- Stored procedures for operations like import and export of data.
- Spatial grid indexes to enhance application query performance.
- Spatial functions that operate upon spatial data to generate spatial information or more spatial data.
- Definition of coordinate systems that determine the location of objects in a given area.
- Default spatial reference systems to efficiently represent geographical entities in the database.
- DB2 Spatial Extender's catalog that controls certain operations.
- The schemas: DB2GSE and ST_INFORMTN_SCHEMA. DB2GSE contains the objects like : stored procedures, spatial data types, the DB2 Spatial Extender catalog, and so on. Views in the catalog are also available in ST_INFORMTN_SCHEMA to conform with the SQL/MM standard.

[5]OGC is an international industry consortium and is leading the development of geospatial standards.

2.7.1 Geometries and data types supported by DB2 Spatial Extender

DB2 Spatial Extender supports a hierarchy of geometries which is defined by OGC document "OpenGIS Simple Features Specification for SQL" [22]. The spatial data types supported by DB2 Spatial Extender are implementations of these geometries. Figure 2.2 represents the geometries supported by DB2 Spatial Extender.

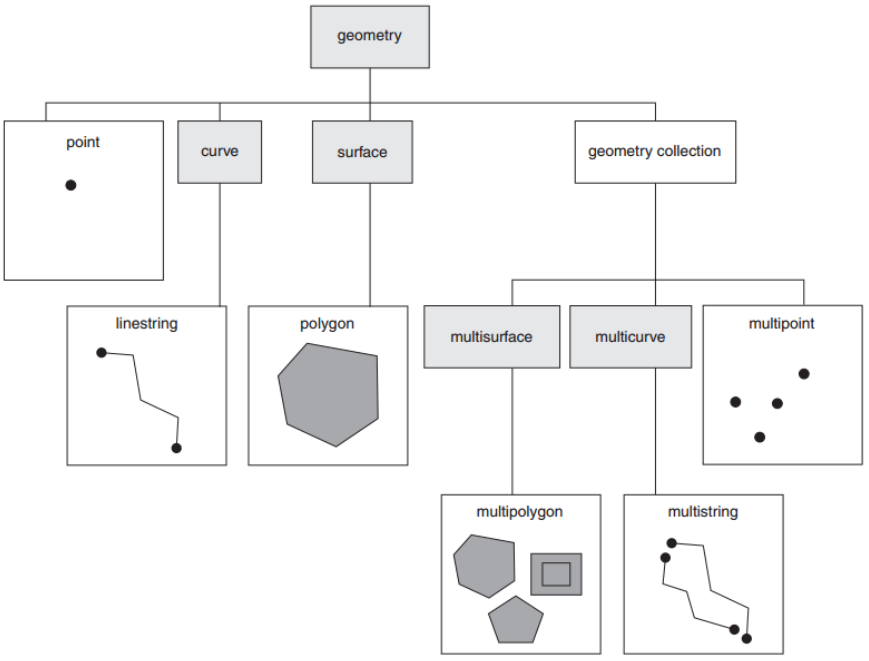


Figure 2.2: *Hierarchy of geometries supported by DB2 Spatial Extender. A superclass called geometry is the root of the hierarchy. The root type and other proper subtypes in the hierarchy are not instantiable [17].*

There are two major subtypes in the geometries, the base geometry subtypes, and the homogeneous collection subtypes. The base geometries include *Points* (A single point), *Linestrings* (A line between two or more points), *Polygons* (A polygon or surface within a polygon). The homogeneous collections include: *Multipoints* (A multiple point geometry collection), *Multilinestrings* (A multiple curve geometry collection with multiple linestrings), *Multipolygons*(A multiple surface geometry collection with multiple polygons). The homogeneous collections are collections of base geometries and have some extra properties over base geometries.

Unless the geometries are empty, they all include at least one X coordinate and one Y coordinate. The X coordinate denotes the location to the east or west of a reference point and the Y coordinate denotes the location to the north or south of the reference

point. In addition, the geometries that have an associated altitude or depth can have an optional Z coordinate. Each point forming the geometry can have this optional coordinate that represents an altitude or depth normal to the earth's surface. It is also possible to have an extra coordinate called the M coordinate (measure), that conveys information about a geographic feature.

The position occupied by the geometries in space is defined by their interiors, boundaries and exteriors. The space that is occupied by the geometry composes the interior. The boundary serves as the interface between the interior and the exterior. The exterior is all the space that is not occupied by the geometry. A geometry is referred to as a simple geometry if it obeys all the rules imposed on its subtype and non-simple otherwise. An empty geometry is the one which does not have any points. The interior, exterior, boundary and envelope are not defined for an empty geometry.

The minimum and maximum (X,Y) coordinates of a geometry form the Minimum Bounding Rectangle (MBR). Generally the MBR of geometries form a boundary rectangle except when the geometry is a point or linestring. The MBR of a point is the point itself and the MBR of a linestring is a linestring represented by the boundary of the source linestring.

Once a database is enabled for spatial operations, DB2 Spatial Extender provides the database with a hierarchy of data types based upon the supported geometries. The data types ST_Point, ST_LineString, ST_Polygon, ST_MultiPoint, ST_MultiLineString, ST_MultiPolygon and ST_GeomCollection from the hierarchy are instantiable. The hierarchy has data types for geographical features that can be considered as forming a single unit and also for the geometries that are made up of multiple components. Figure 2.3 represents the hierarchy of data types supplied by DB2 Spatial Extender.

- **Data types for single-unit features**

  The data types ST_Point, ST_LineString and ST_Polygon are used to store the coordinates of a geographic feature that can be perceived as forming a single unit.

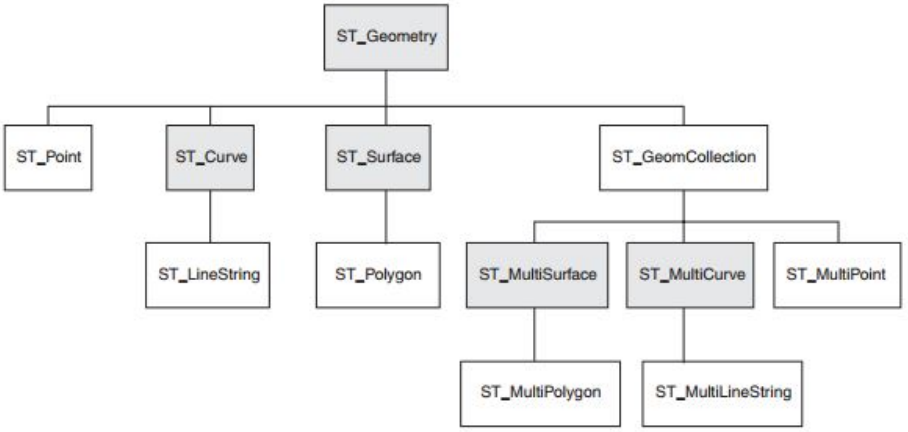


Figure 2.3: *Hierarchy of data types supported by DB2 Spatial Extender.Data types named in the shaded boxes are not instantiable [17].*

- **ST_Point:** is used to indicate a point in space which might refer to a discrete geographic feature. The size of the object may vary from very small like the location of a house to very big such as a country.
- **ST_LineString:** is used to store a set of coordinates which refer to a linear geographic feature. For example, the features like roads, rivers and pipelines.
- **ST_Polygon:** is used to indicate the extent of space covered by a multi-sided feature. It might be used to refer to the features such as a city, a country, a wildlife habitat, etc.

- **Data types for multi-unit features**

  The data types ST_MultiPoint, ST_MultiLineString and ST_MultiPolygon are used to store the coordinates of the geographic features that are made up of multiple units. Multi-unit does not mean a collection of individual entities rather, an aggregate of the parts that make up the whole.

  - **ST_MultiPoint:** is used to represent features made up of units whose locations are each referenced by an X and Y coordinate. For example, consider a table whose row represents the location of cars on a racing field.
  - **ST_MultiLineString:** is used to represent features that are made up of multiple linear units. ST_MultiLineString stores the coordinates for the location of these single linear units and the location of each feature as a

whole. For example, consider a table whose rows represent the network of streets.

– **ST_MultiPolygon:** is used to represent features made up of multiple multi-sided units. ST_MultiPolygon stores the coordinates for the location of the single multi-sided units and the location of each feature as a whole. For example, consider a table whose rows represent the boundary of counties in a state.

2.7.2 Populating spatial columns

Once the spatial columns are created, there are several ways to populate them with spatial data. Data can be imported from a previously existing source (data exchange files) or it can be derived from business data by using a geocoder. User can also create the data by using various spatial functions in conjunction with business data or other spatial data.

- **Importing and exporting spatial data**

  DB2 Spatial Extender can exchange spatial data between the database and external data sources. Data exchange files can be used to import spatial data to a DB2 database. Data can be exported in the form of data exchange files from which external sources can acquire it. There could be several reasons to import spatial data from other sources. We can acquire a lot of spatial information that is already available in the industry. We can supplement the data in our database with the data supplied by external sources, for example, a map vendor. It is possible to immigrate data between systems using the import and export process. These data exchange files can also be used to render the data in visual form by supplying the files to a geobrowser system. DB2 Spatial Extender supports shape files for data exchange. A detailed description of shape files is presented in Section 2.9. DB2 provides stored procedures namely `ST_IMPORT_SHAPE` and `ST_EXPORT_SHAPE` for data import and export respectively.

- **Geocoder**

In DB2 Spatial Extender, a geocoder is a scalar function that can be used to translate existing relational data into spatial data. The input i.e. relational data describes or names a location and the output is received in spatial terms. A geocoder can also be used to translate spatial data for conformity purposes. DB2 Spatial Extender can support user-supplied and vendor supplied geocoders. A geocoder can operate in two modes. In batch mode, all its input from a single table (or a subset of rows) are translated at once. In automatic mode, a geocoder translates data as soon as it is inserted or updated in a table [17].

2.7.3 Spatial grid indexes

DB2 Spatial Extender allows the user to create spatial grid indexes on the table to improve application query performance. This feature is specially beneficial when the queried table or tables contain many rows. the number of rows to be processed can be greatly reduced if appropriate indexes are created. Spatial Extender uses the minimum bounding rectangle (MBR) of a geometry to create spatial grid index.

Up to three index levels can be defined for a spatial column. An index entry consists of the grid cell identifier, the geometry MBR, and the internal identifier of the row that contains the geometry. DB2 optimizer considers the spatial grid indexes for the access plan if the spatial function are used in the WHERE clause. For the efficient use of the optimizer an appropriate grid size must be specified. The most common query window size must be taken into account while choosing the grid size. DB2 Spatial Extender also provides an Index Adviser utility. This utility examines the spatial column data and suggests appropriate grid sizes for typical query window sizes.

2.7.4 Spatial reference systems

A spatial reference system (SRS) is a group of parameters that is used to represent geographical entities. Among other parameters, a spatial reference system defines a specific coordinate system from which the coordinates are derived. Coordinates that indicate the maximum extent of the referenced space are also defined.

Traditionally coordinates in a coordinate system are decimal values that can be both negative or positive. To enhance performance, the negative coordinates are converted to positive values, and the decimal coordinates are converted into integer. These conversions are made by certain parameters like the *offset* and *scale factor*. A SRS specifies these parameters.

2.7.5 Spatial stored procedures and functions

DB2 Spatial Extender provides a variety of stored procedures. Some of those are, ST_CREATE_COORDSYS, ST_ENABLE_DB, ST_IMPORT_SHAPE, etc. These stored procedures are used to set up DB2 Spatial Extender and create projects that use spatial data.

DB2 Spatial Extender also provides a large number of spatial functions that can be used to program the applications. Before using the function, it is necessary to qualify the name of the function by the name of the schema to which the spatial function belongs i.e. DB2GSE. Several spatial functions can be invoked as methods as well. Depending upon there use, the functions can be organized into several categories. For example:

- **Constructor functions :** DB2 Spatial extender has functions that convert geometries to and from supported data exchange formats. A detailed description of the supported data formats is given in Section 2.7.6. The functions for creating geometries from these formats are known as constructor functions. For example, ST_PointFromWKB, ST_GeomFromWKB etc.
- **Comparison functions** : These functions are used to compare geometries for boundaries, intersections, and other information. For example, ST_Intersects, ST_Contains etc.

There are several other categories of spatial functions provided by DB2 Spatial Extender. Detailed information about the offered stored procedures and functions is available in the *Spatial Extender User's Guide and Reference* [17].

2.7.6 Supported data formats

DB2 Spatial Extender supports various industry standard spatial data formats.

- **Well-known text (WKT) representation**

  Well-known text format allows to represent and exchange geometry data in American Standard Code for Information Interchange (ASCII) format. It is a text markup language for representing vector geometry objects on a map. WKT representation can also be used for transformations between spatial reference systems. The format was originally defined by OGC and is also referenced by the ISO "SQL/MM Part 3: Spatial" standard [23].

- **Well-known binary (WKB) representation**

  Well-known binary format is the binary equivalent of the WKT format. It is used to transfer and store the same information. This representation is also defined by the OpenGIS Consortium "Simple Features for SQL" specification and ISO "SQL/MM Part 3: Spatial" standard [23]. Byte stream for a point is the basic building block for WKB representation. It consists of two double values. Other geometries are built using the byte stream for geometries that are already defined.

- **Shape representation**

  Shapefile is a popular standard for representing geospatial vector data, developed and regulated by Environmental Systems Research Institute (ESRI). It is a widely used standard to represent spatial data. A detailed description of the format is presented in Section 2.9.

- **Geography Markup Language (GML) representation**

  Geography markup language is the XML grammar defined by the OpenGIS consortium to express geographical features. GML serves as a modeling language for geographic systems. It can also be used as an open interchange format for geographic transactions on the Internet. GML is capable of integrating all forms of spatial information [24]. There are several functions in DB2 Spatial Extender that generate geometries from representations in GML.

2.8 Statistical data analysis tool R

R is a free software environment and a well-developed programming language for statistical computing and graphics [25]. It is often referred to as a different implementation of the S language and environment which was developed by John Chambers and colleagues at Bell laboratories. Although there are some differences in the implementation of R and S, but most of the code written for S runs unaltered under R. It is designed as a fully planned and coherent system hence the term "R environment" is appropriate. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS [21].

R provides a broad range of statistical techniques like classification, linear and non-linear modeling, classical statistical tests, clustering, time-series analysis and much more. Since it is build for statistical purposes, it has a suite of operators for calculation on arrays and matrices. An assortment of intermediate tool for data analysis is also available. It has an effective data handling and storage facility. R is used extensively for graphical purposes as well. It enhances graphical facilities for data analysis and is capable of generating publication-quality plots with ease.

R supports procedural programming with functions and, for some functions, object-oriented programming with generic functions. C, C++ and Fortran code can be linked and called at run time for computationally-intensive tasks. It is also possible to manipulate R objects directly by using the code written in languages like C, C++, Python and Java. R is currently developed by the R Development Core Team. Initially it was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand. The source code for the R software environment is written primarily in C, Fortran and R. Native R has a command line interface, but there are several graphical front-ends available.

One of the most striking feature of the R environment is that it can be extended very easily with the help of packages. About eight packages are supplied with the R distribution. Other packages can be installed as per need from the CRAN family of internet sites covering a very wide range of modern statistics. Pools and studies of scholarly literature databases show that R's popularity has increased in recent years [26, 27].

2.8.1 sp Package

The `sp` package is a very powerful package which provides the classes and methods to deal with the spatial data in R [28]. It defines a uniform interface for holding spatial data and implements various data structures like points, lines, polygons and grids. One important feature of sp package is that instead of directly manipulating the class slots, it provides methods and functions to create the classes from elementary types. Each class is created using elementary types like matrices, dataframes or lists.

The spatial data classes implemented are points, grids, lines, rings and polygons. `sp` package provides classes both for only spatial information and spatial information along-with attribute information. Each spatial only class e.g. `SpatialPoints` has an extension of the form `SpatialPointsDataFrame` which can hold the attribute information as well.

2.9 Shapefile format

Shapefile is a popular standard for representing geospatial vector data. It is developed and regulated by Environmental Systems Research Institute (ESRI) [29]. Shapefiles can be easily copied and supported by a number of software packages. It allows to store nontopological geometry and attribute information for the spatial features in a dataset. Shapefile format was introduced with ArcView GIS version 2 in the early 1990s. It has a simple file structure which draws faster relative to other common formats. In-fact, shapefile is a collection of files with a common filename prefix. A shapefile consists necessarily of a main file (.shp), an index file (.shx) and a dBASE file (.dbf), and can optionally have other associated files (.sbn, .xml, .prj etc.) [30].

- **.shp**: The main file is variable-record-length file which contains the feature geometry itself. It consists of a single fixed length (100 bytes) header followed by one or more variable length records. Each record is a list of vertices describing a shape.
- **.shx**: The index file contains the offset of each record corresponding to the main file record. Similar to the .shp file it has a fixed 100-byte header, followed by any

number of 8-byte fixed-length records. These records store the record offset and record length. It allows seeking forwards and backwards in the shapefile quickly.

- **.dbf**: The dBASE file contains the attributes associated with the feature, with one record per feature. An alternative format called xBase can also be used.

In the above mentioned files, the records in each file correspond to each other in a sequence (i.e., the first record in the .shp file corresponds to the first record in the .shx and .dbf files, etc.).

ESRI shapefile has become a common standard for representing geospatial data and many leading organizations provide their geospatial datasets in the same format. There are some limitations to the shapefile format such as it can not store topological information and the size of both .shp and .dbf files can not exceed 2 GB ($2^{31} bytes$) [31]. A wide variety of software can be used to read and write geographical datasets in the shapefile format. Majority of the datasets used in this study were obtained in the shapefile format from various sources.

Chapter 3

An R Interface for Scalable Geospatial Analytics

3.1 ibmdbR

The `ibmdbR` package provides an efficient approach for statistical analysis using R. It automatically translates R language constructs into SQL statements. These statements are then executed against a scalable database in the background. This allows users to work with big data warehouses without the knowledge of SQL. However, the `ibmdbR` package does not provide any support for geospatial operations. One of the main objective of this thesis is to fill this gap. In this section, some features of the existing `ibmdbR` package are explained. In the later sections, we build up a scalable solution for geospatial analysis and processing following a design process.

R provides an effective platform for statistical analysis. But it is a prerequisite in most native R functions that the data to be processed must be present in the main memory. However, this might turn impractical or even impossible if you need to analyze a large amount of data. The package `ibmdbR` offers methods that push down the operations of R into the underlying database for execution and hence increasing the memory limits of R. In addition, the user profits from the performance-enhancing features of the underlying database management system, such as parallel processing, without having to interact with the database explicitly.

The user creates an object of the class ida.data.frame. It is a pointer to a table located in the database. The data frame does not store any data in local memory, instead it contains the metadata used to determine the table subset. The function set operates on these objects to perform various operations and keeps on updating the metadata in the data frame. Many operations can be performed without loading the content of this table or query into memory.

The `ibmdbR` package can be used in conjunction with IBM DB2© for Linux and Windows Version 10.5 (in the following abbreviated to DB2), as well as with IBM dashDB©. The appropriate client driver packages must be installed while using DB2 and a configured ODBC source is required. In the following it is assumed that an ODBC data source called "BLUDB" is already created.

3.1.1 Getting started

The user needs to download and install the package. A connection to the database is necessary before using any of the push-down functions of the `ibmdbR` package. This is done by executing the *idaConnect* function with appropriate parameters. *idaConnect* function is used to open an IDA database connection. The ODBC source, the user name and the password are supplied as the parameters as shown in the code below.

```
>install.packages('ibmdbR')
>library('ibmdbR')
>con <- idaConnect('BLUDB','USERNAME','PASSWORD')
```

Next, initialize the in-database functionality by executing the *idaInit* function. The *idaInit* function creates a singleton for the connection object such that it is not required to pass it as a parameter later on:

```
>idaInit(con)
```

Now we can use the rest of the functions in the package. `ibmdbR` package provides a set of functions to analyze the tables in the database and the data in those tables. For example, the function *idaShowTables* will return a `data.frame` that contains a list

of all tables in the current schema. The result to the *idaShowTables* function shown below consists of the table schema, table name, owner and type respectively.

```
> idaShowTables()
   Schema             Name   Owner   Type
1  POOL44             IRIS  POOL44     T
2  POOL44             CARS  POOL44     T
```

3.1.2 Working with ida.data.frame

In comparison to a normal data.frame in R which holds data in memory, an object of class ida.data.frame contains only a reference to a table or a query. An object of class ida.data.frame can be created by pointing to an existing table in the database. For example, for an existing table called 'IRIS' in the database, we can create an ida.data.frame object by executing the following statement, where `iris.ida` is the name of the new object.

```
>iris.ida <- ida.data.frame('IRIS')
```

The ibmdbR package overwrites many methods and functions defined for regular R data.frame objects. It uses SQL to push the execution of these methods down into the database. For example, the native *dim* method returns the dimension of a normal R data.frame, the overwritten method does the same for an ida.data.frame object.

```
> dim(iris.ida)
[1] 150 5
```

Another example is the *head* method. By default the method returns the first six rows of a data frame. It does the same for a ida.data.frame object by executing the appropriate SQL statements in the background. The statement below uses the head function with a ida.data.frame object. The output of various R statements have been structurally modified in the document to fit the page width, the output on R console might look different.

```
> head(iris.ida)
  SepalLength SepalWidth PetalLength PetalWidth Species
1         5.1        3.5         1.4        0.2  setosa
2         4.9        3.0         1.4        0.2  setosa
3         4.7        3.2         1.3        0.2  setosa
4         4.6        3.1         1.5        0.2  setosa
5         5.0        3.6         1.4        0.2  setosa
6         5.4        3.9         1.7        0.4  setosa
```

Similarly, there are several other methods (defined on the *data.frame* class) which the ibmdbR package overwrites, for example, *as.data.frame, sd, max, mean, min, length, print, names, colnames, summary, NROW, NCOL, var, cor* and *cov*.

More often than not it is a requirement in analysis process to work with a selection of the table instead of the full table. It is possible to do that with ida.data.frame objects in a similar way as with a regular *data.frame*. For example, the following statements, would select only the rows where iris 'Species' equals 'setosa' and only the columns 'PetalLength' and 'PetalWidth'. As can be seen in later statements , all methods and functions applied to an ida.data.frame object with selection will reflect it, which is why the *dim* method now returns 50 row and 2 columns instead of 150 rows and 5 columns.

```
> iris.ida2 <- iris.ida[iris.ida$Species=='setosa',c('PetalLength','PetalWidth')]
> dim(iris.ida2)
[1] 50 2
```

The package `ibmdbR` provides a variety of functions for pre-processing and analyzing data. The reference manual for the package contains more details and examples [32].

3.2 Design goals for geospatial functionality

Section 3.1 provides an overview of the existing `ibmdbR` package. The package provides an efficient approach for statistical analysis. As mentioned in the Section 1.2, we aim to further extend the package to include the operations that support geospatial analysis and processing at scale. In this section, we define the principal design goals for building up the geospatial capability in the package.

- **Scalability**

  Rendering a solution which is scalable beyond the present capabilities of R and the `sp` package is the prime goal. Geospatial datasets are usually very large in size. The present geospatial capabilities of R are not scalable as the data have to reside in main-memory. This limits the amount of that data that can be analyzed. Hence, scalable solutions are required for geospatial analysis and processing.

- **Adherence to the `sp` package**

  To maintain the adaptability of the package we decide to keep the form of the spatial operations as close as possible to those provided by the sp package and other supporting packages e.g. `rgeos` [33]. More details about the `sp` package are presented in Section 2.8.1.

- **Coalescing with `ibmdbr` package**

  Spatial operations are desired to be build up upon the existing classes and supporting functions provided by the `ibmdbR` package. Through this goal we can exploit the wide range of statistical functions provided by the package.

- **Intuitive operations**

  The goal is to design the operations in a manner that they are intuitive to the individuals who are not experts in the geospatial domain. The functions must be flexible and easy to use. Instead of writing long and complex queries, the user must be able to perform spatial operations in a convenient manner.

- **Lazy materialization**

  Functions are to be designed such that they support the concept of lazy materialization. This can be achieved by storing the spatial operations in the form of SQL queries and not materializing them until required.

3.3 Design

We seek to design the spatial operations such that they suffice the design goals defined in Section 3.2. For scalability, we use the DB2 Spatial Extender as the back-end. The spatial operations are pushed down to the database in the form of corresponding spatial

queries. Consequently, we are not limited to the amount of data that can fit into the main memory. We can analyze the data residing into the database without moving it into working memory. This also has the benefit that the data being analyzed is as current as possible.

To benefit from the concept of lazy materialization, we have designed the functions in a way that they create a view from the existing tables. The views are created based on the specified spatial operations. These views are not materialized until its necessary or in some cases might not be materialized at all. We can apply analysis functions over these views. This saves a considerable amount of time and computation. For example in *idaIntersects* function (details in Section 3.4.2), instead of creating a new table which contains the result, we create a view that represents the potential result. Each *idaIntersects* function call executes an SQL query of the form shown below:

```
CREATE VIEW viewName AS SELECT ( columns and intersect condition..)
```

We have built up the spatial capability upon the classes provided by the `ibmdbR` package. This has the benefit that we can use the variety of statistical analysis functions provided by the package on our spatial datasets if required. Changes have been made where deemed useful. For example the *head* function in the `ibmdbR` package returns the actual contents of the table subset. But this might turn impractical when using the function with spatial tables. Because the spatial columns in the table contain the long list of coordinates of the geometries. If we return the actual coordinates of the geometries, the R console is rapidly cluttered. To overcome this problem, we return the spatial data type of the column instead of the actual coordinates. This conveys the information about the spatial columns in a compact manner without cluttering the R console output. It was observed to be a better way of representing spatial information specially in the case of line and polygon geometries.

We can create an ida.data.frame object for a table with spatial columns by executing the following statement. Here, airports.ida is the name of the new object.

```
> airports.ida <- ida.data.frame('AIRPORTS')
```

Execute the following statements to check the class of the ida.data.frame object and for printing it.

```
> class(airports.ida)
[1] "ida.data.frame"

> airports.ida
SELECT "NAME","GEOSERVER","URL","SHAPE_AREA","SHAPE_LEN",
       "AIRPORT_ID","AIRPORT_BOUNDARY" FROM AIRPORTS
```

The user sees the ida.data.frame object as an SQL query (see output above). The actual structure of the ida.data.frame objects consists of four slots each containing a part of information about the subset of the database table. Operations progressively update the definition and structure of the ida.data.frame objects. The structure of the airports.ida object is as shown below.

```
> airports.ida # Typically not visible to the user
An object of class "ida.data.frame"
Slot "table":
[1] "AIRPORTS"
Slot "where":
[1] ""
Slot "cols":
[1] "NAME" "GEOSERVER"  "URL" "SHAPE_AREA" "SHAPE_LEN" "AIRPORT_ID" "AIRPORT_BOUNDARY"
Slot "colDefs":
list()
```

The slots 'table' and 'where' contain name of the database table to which the object points and where conditions of the select query respectively. The slots 'cols' and 'coldefs' contain information about the relevant column subset and new column definitions added to the object respectively. While fetching the data from the database table, these slots are assembled into an appropriate SQL query depending upon the performed operation.

While importing data, conformity with the data classes defined by the `sp` package (SpatialLinesDataFrame-class, SpatialPointsDataFrame-class etc.) is maintained. A class of functions (details in Section 3.4.3) is provided that facilitates the import of spatial data from the table in the database to R environment. The data is imported from the database and transformed such that it follows the design rules of the classes

in `sp` packages. For example, we can import the AIRPORTS dataset which contains the polygon boundaries of two airports in R environment. The R statements are as shown in the code block below. airports.ida is an object of the ida.data.frame class which points to the AIRPORTS table in the database. The class of the resulting dataset rAirports is SpatialPolygonsDataFrame which is defined in the `sp` package to hold spatial polygons with attributes.

```
> rAirports<-as.SpatialPolygonsDataFrame(airports.ida)
> class(rAirports)
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

While adding spatial attributes to the existing ida.data.frame objects such as the *area, centroid* etc., the attributes are not evaluated until its required (lazy evaluation). We expand the definition of the ida.data.frame object to include the spatial query. An extra column definition is added to the object as shown below:

```
Slot "colDefs":
$AREA
[1] "db2gse.ST_AREA('AIRPORT_BOUNDARY')"
```

The data classes defined in the `sp` package limit the number of spatial attributes to one per object. We extend the usability of the operations, as the ida.data.frame objects are designed to support more than one spatial column. This feature turns out to be very useful when we need to analyze the datasets that have more than one spatial columns. For example the Taxi dataset (Section 4.1.1) used in the study has two spatial columns called 'P_LOCATION' and 'D_LOCATION'. The columns represent the pickup and dropoff location of the taxi rides respectively.

3.4 Spatial operations

We have implemented several functions as an extension for the geospatial capability. Depending upon the use cases listed in Chapter 4, the functions explained below were

recognized to be of most importance. These function are used in wide range of geospatial analysis problems. The functions can be applied on the database tables containing spatial columns. In cases where the functions generate new geometries, the spatial reference system of the given geometries is used for creating the new ones.

3.4.1 idaBuffer

The *idaBuffer* function can be used to expand the given geometry and include the area within the specified width. The function takes a geometry, a distance, name of the spatial column and optionally the name of the new table as parameters. It returns a new ida.data.frame object with an extra column that represents the newly created geometry. Each point on the boundary of the resulting geometry is specified distance away from the given geometry. The spatial reference systems of the given geometry and the resulting geometry are same and the data type of the new column is ST_GEOMETRY. Each *idaBuffer* function call internally executes an SQL query of the following form and returns the view as a ida.data.frame object.

```
CREATE VIEW tableName
 AS SELECT table_attributes,
 DB2GSE.ST_BUFFER(spatial_column,width) AS 'BUFFER_spatial_column'
 FROM spatial_table
```

The following statements give an example of *idaBuffer* function. Here, we expand the airport polygon boundary (AIRPORT_BOUNDARY) of the airports.ida object and include the area within the specified width. We select the spatial column using the *col* parameter and the name of the new table using the *tableName* parameter. We get a new ida.data.frame object as a result. Here we name it airports_area. From the output of the *head* function it can be seen that a new column is introduced. The new column is named with the concatenation of the word 'BUFFER' and the column name of the specified geometry. In this case, the column 'BUFFER_AIRPORT_BOUNDARY' represents the new geometry.

```
> # Expand the airport boundary by specified 'width'
> airports_area <- idaBuffer(airports.ida,
                             col=airports.ida$AIRPORT_BOUNDARY,
```

```
                              width=width,
                              tableName="AIRPORTS_AREA")

> head(airports_area)
NAME GEOS.. URL SHAPE_A.. SHAPE_LEN AIRPORT_ID AIRPORT_BOUNDARY BUFFER_AIRPORT_BOUNDARY
1 La.. 17.. http..   0        0        1      ST_MULTIPOLYGON     ST_GEOMETRY
2 Jo.. 97.. http..   0        0        2      ST_MULTIPOLYGON     ST_GEOMETRY
```

Internally the following SQL query was executed to create the new ida.data.frame object with the required column. Here we have specified the width in degree units. We are using the NAD83_SRS_1 spatial reference system for the AIRPORTS dataset, one degree is approximately equal to 108.542 kilometers (0.00369 degrees ≈ 400 meters) [17].

```
CREATE VIEW AIRPORTS_AREA AS
 SELECT NAME,GEOSERVER,URL,SHAPE_AREA,SHAPE_LEN,
        AIRPORT_ID,AIRPORT_BOUNDARY,
        DB2GSE.ST_BUFFER( AIRPORT_BOUNDARY,0.00369)
         AS BUFFER_AIRPORT_BOUNDARY
  FROM AIRPORTS
```

3.4.2 idaIntersects

It is a very common requirement in spatial analysis to check whether the two geometries intersect. The function *idaIntersects* is provided for this purpose. It can be easily applied to millions of rows with one or more spatial columns in the table. It takes two ida.data.frame class object as parameters. By default, it intersects the first spatial column of the first ida.data.frame object with the first spatial column of the second ida.data.frame object. In case the objects have more than one spatial columns, columns of interest can be specified using the *by.x* and *by.y* parameters. The function returns 1 if the geometries intersect and 0 (zero) is returned if they are disjoint. Internally, however, it executes, among other statements, a SQL query of the following form:

```
CREATE VIEW tableName
 AS SELECT (chosen_parameters)
 FROM TABLE_1 JOIN TABLE_2
 ON DB2GSE.ST_INTERSECTS(TABLE_1_SPATIAL_COLUMN, TABLE_2_SPATIAL_COLUMN)=1
```

The *returnDense* parameter of the function can be set to TRUE or FALSE depending upon the application. When set to TRUE the function returns only those records from the first ida.data.frame object that intersect with some geometry in the second ida.data.frame object. In the other case, all the records are returned.

The following statements determine whether the geometries in the TripAndFare (here a subset of 1000 records from the TripAndFare dataset (4.1.1)) and airports_area ida.data.frame objects intersect. Using the selection capability over the ida.data.frame objects, we have chosen three representative columns from the TripAndFare and airports_area ida.data.frame objects each. The *idaIntersects* functions takes the spatial column from the first object i.e 'P_LOCATION' and intersects it with first spatial column in the second object i.e. 'BUFFER_AIRPORT_BOUNDARY'. The dimension of the output is 1000 rows and 7 columns. From the output of the *head* function it can be seen that the record for the 'TRIP_ID' = 9399873 intersects with the polygon boundary of the 'AIRPORT_ID' = 2 (John F. Kennedy International Airport). This taxi ride is probably destined for the airport.

```
> airTaxi<-idaIntersects(TripAndFare[,c("TRIP_ID","P_LOCATION","D_LOCATION")],
+                        airports.ida[,c("AIRPORT_ID","NAME","AIRPORT_BOUNDARY")],
+                        tableName="AIRTAXI",
+                        returnDense = F)

> dim(airTaxi)
[1] 1000    7
> head(airTaxi,n=200)
TRIP_ID  P_LO.. D_LO.. AIRPORT_ID NAME  AIRPORT_B. INTERSECTS_TRIPANDFARE_1000_AIRPORTS
9399873  ST_POINT ST_POINT    2   John F. ST_MULTIP..                    1
3686347  ST_POINT ST_POINT    NA    <NA>  ST_MULTIP..                    0
11890564 ST_POINT ST_POINT    NA    <NA>  ST_MULTIP..                    0
4994452  ST_POINT ST_POINT    NA    <NA>  ST_MULTIP..                    0
```

```
162380   ST_POINT ST_POINT    NA    <NA>  ST_MULTIP..                    0
12881302 ST_POINT ST_POINT    NA    <NA>  ST_MULTIP..                    0
```

For the above geospatial operation the user just needs to use the *idaIntersects* function with required attributes, the query actually executed in DB2 Spatial Extender is shown below. Here it can be observed that the function is really intuitive and instead of writing a complex SQL command, the required operations can be performed through an easy to use function.

```
CREATE VIEW AIRTAXI
 AS SELECT l.TRIP_ID,l.P_LOCATION,
           l.D_LOCATION,temp.AIRPORT_ID,
           temp.NAME,temp.AIRPORT_BOUNDARY,
           NVL(temp.INTERSECTS_TRIPANDFARE_1000_AIRPORTS,0)
           AS INTERSECTS_TRIPANDFARE_1000_AIRPORTS
 FROM TRIPANDFARE_1000 l LEFT OUTER JOIN(SELECT x.TRIP_ID AS TRIP_ID,
                                                x.P_LOCATION AS P_LOCATION,
                                                x.D_LOCATION AS D_LOCATION,
                                                y.AIRPORT_ID AS AIRPORT_ID,
                                                y.NAME AS NAME,
                                                y.AIRPORT_BOUNDARY AS AIRPORT_BOUNDARY,
                                                DB2GSE.ST_INTERSECTS( x.P_LOCATION,
                                                y.AIRPORT_BOUNDARY )
                                                AS INTERSECTS_TRIPANDFARE_1000_AIRPORTS
                                           FROM TRIPANDFARE_1000 x JOIN AIRPORTS y
                                           ON DB2GSE.ST_INTERSECTS( x.P_LOCATION ,
                                            y.AIRPORT_BOUNDARY )=1) temp
 ON l.TRIP_ID = temp.TRIP_ID
```

3.4.3 Functions for data import to R

There could be several scenarios where the user wishes to import the spatial table or a subset of the table into R. One example of those use cases is when the user wants to plot the geospatial data using the plot functions of R. Therefore, functions are provided to enable the import of data from a table in DB2 to R environment. Imported data

conforms to the format specified by the `sp` package (Section 2.8.1). Plotting functions of R can then be used to plot geospatial data.

3.4.3.1 as.SpatialPointsDataFrame

This function can be used to import database tables that hold spatial attributes with spatial point locations. The data is imported in form of the SpatialPointsDataFrame class defined by the `sp` package. The positional information is stored as an object of the Spatialpoints class and the attribute information as an object of data.frame class of R. The statements below import a subset the TRIPANDFARE table from the database using the TripAndFare ida.data.frame object. The table TRIPANDFARE contains two spatial columns of the type ST_POINT. But the `sp` package in R supports only one spatial attribute per object. By default the *as.SpatialPointsDataFrame* chooses the first spatial column in the table. In case of more than one spatial columns, the required column can be explicitly specified in the import statement.

```
> # Import data from TRIPANDFARE table
> rTripAndFare<-as.SpatialPointsDataFrame(TripAndFare)
> class(rTripAndFare)
[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"
```

3.4.3.2 as.SpatialLinesDataFrame

This function can be used to import database tables that hold data consisitng of sets of lines, where each set of lines relates to an attribute row in the table. The data is imported in form of the SpatialLinesDataFrame class defined by the `sp` package.

3.4.3.3 as.SpatialPolygonsDataFrame

This function is for the database tables that hold spatial polygons with spatial attributes. The data is imported in form of the SpatialPolygonsDataFrame class defined by the `sp` package. The statements below import the USCOUNTIES dataset from the

database using the usCounties.ida object of the ida.data.frame class. USCOUNTIES tables contains information about the polygon boundaries of 3109 counties in USA along-with some attribute information.

```
> # ida.data.frame object for USCOUNTIES table
> usCounties.ida<-ida.data.frame("USCOUNTIES")


> # Import USCOUNTIES dataset
> rUsCounties<-as.SpatialPolygonsDataFrame(usCounties.ida)
> class(rUsCounties)
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

Once the data is available in the R environment, the plotting functions can be used to visualize the dataset. The statements below use the *plot* function in R to visualize the US counties polygon boundaries. The resulting figure is shown in Figure 3.1.

```
> plot(rUsCounties,
+      col="light yellow",
+      axes=TRUE,
+      bg="grey95",
+      cex.axis=1,
+      cex.lab=1,
+      xlab="Longitude",
+      ylab="Latitude")
> title(main="Counties in USA",cex.main=1.2)
```

Some functions like *spatial.col*, *spatial.type*, *ida.Proj4string*, etc. support other spatial functions and are not directly available to the user.

3.4.4 Functions for adding spatial information

There are several other functions developed to support spatial data processing and adding spatial information to existing geometries. For example, *area*, *distance* etc. Some of these functions are briefly explained below:

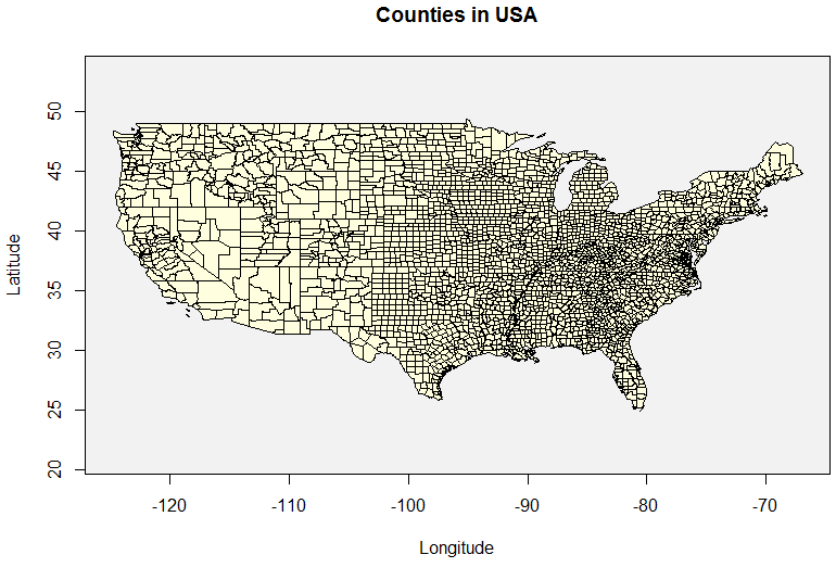


Figure 3.1: *This map represents the polygon boundaries of 3109 counties in United States of America. The x and y axis in the plot represent the longitude and latitude of the geographical features respectively. The spatial data to plot the polygons is imported from the database to R environment using the developed import functions.*

- **area**: This function can be used to include additional geospatial information in the form of the area of the geographical feature. The function takes a spatial column as the input parameter and returns the area covered by the geometry in the default measuring units. If the geometry is a polygon or multipolygon, then the area covered by the geometry is returned. The area of points, linestrings, multipoints, and multilinestrings is 0 (zero). If the geometry is null or is an empty geometry, null is returned.

  We expand the definition of the ida.data.frame object to include this spatial query. Following the lazy evaluation principle, the query is executed only when required by the user. An extra column definition is added to the 'colDefs' slot of the object:

```
Slot "colDefs":

$AREA

[1] "db2gse.ST_AREA('AIRPORT_BOUNDARY')"
```

  The statements below add an 'AREA' column to the airports.ida ida.data.frame object. The spatial column 'AIRPORT_BOUNDARY' is given as the parameter to the function. We can see the an extra attribute with the name AREA

which represents the area of the geometry in square degree units, is added at the end. Here, the area of the geometry is evaluated only when the *head* function is executed.

```
> # Area of the airports
> airports.ida$AREA<-area(airports.ida$AIRPORT_BOUNDARY)
> head(airports.ida)
  NAME GEO..  URL   SHA..  SHA..  AIRPORT_ID AIRPORT_BOUNDARY    AREA
1 La..  17.. http://  0      0          1   ST_MULTIPOLYGON 0.0001771799
2 Jo..  97.. http://  0      0          2   ST_MULTIPOLYGON 0.0009632901
```

The query which was executed in DB2 Spatial Extender at the background when using the *head* function in R is as given below:

```
SELECT NAME,GEOSERVER,URL,
       SHAPE_AREA,SHAPE_LEN,AIRPORT_ID,
       'ST_MULTIPOLYGON' AS AIRPORT_BOUNDARY,
       (db2gse.ST_AREA(AIRPORT_BOUNDARY)) AS AREA
FROM AIRPORTS FETCH FIRST 6 ROWS ONLY"
```

- **centroid**: The centroid function takes a spatial column as the input parameter and returns the geometric center, which is the center of the minimum bounding rectangle of the given geometry, as a point. The resulting point is represented in the spatial reference system of the given geometry. If the given geometry is null or is empty, then null is returned. In the statements below we add a column named 'CENTROID' to the usCounties.ida ida.data.frame object. Here, COUNTY_BOUNDARY geography is supplied as the spatial attribute to the *centroid* function. We can see that a column which represents the centroid ('CENTROID') of the corresponding county is added at the end.

```
> usCounties.ida$CENTROID <- centroid(usCounties.ida$COUNTY_BOUNDARY)
> head(usCounties.ida)
 COUNTY_ID    NAME    STATE_NAME  FIPS COUNTY_BOUNDARY              CENTROID
1   1  Lake of th..  Minnesota   27077 ST_MULTIPO.. POINT (-94.886732 48.869971)
2   2  Ferry         Washington  53019 ST_MULTIPO.. POINT (-118.480562 48.417890)
3   3  Stevens       Washington  53065 ST_MULTIPO.. POINT (-117.910098 48.394966)
```

```
4   4  Okanogan      Washington  53047 ST_MULTIPO.. POINT (-119.857786 48.468616)
5   5  Pend Oreille  Washington  53051 ST_MULTIPO.. POINT (-117.332550 48.522023)
6   6  Boundary      Idaho       16021 ST_MULTIPO.. POINT (-116.544371 48.748040)
```

- **distance**: The distance function takes two geometries as the input parameters and returns the shortest distance between any point in the first geometry to any point in the second geometry. The statements below calculate the distance between the pickup (P_LOCATION) and dropoff (D_LOCATION) locations of the taxi rides from the TripAndFare dataset.

```
> # Distance between taxi pickup and dropoff locations
> TripAndFare$DISTANCE<-distance(TripAndFare$P_LOCATION,TripAndFare$D_LOCATION)
> head(TripAndFare)
   TRIP_ID MEDALLION    ....  P_LOCATION D_LOCATION  DISTANCE
1 11911614 99F1C8..     ....  ST_POINT   ST_POINT  0.0000000
2 10394608 826481..     ....  ST_POINT   ST_POINT  5.3496043
3  2430331 DD7ADF..     ....  ST_POINT   ST_POINT  1.6699156
4   149805 8853EB..     ....  ST_POINT   ST_POINT  0.9851455
5  6170904 043766..     ....  ST_POINT   ST_POINT  0.6507103
6 11764402 6D5330..     ....  ST_POINT   ST_POINT  0.8935272
```

Chapter 4

Scalable Geospatial Analytics with R in Action

In this chapter, we utilize the knowledge acquired about the geospatial domain through Chapter 2 and the scalable solution developed in 3 on practical real-life scenarios. First we provide a description of all the datasets used during the course of the study. We then use the spatial operations on the defined use-cases. The use cases bear practical applications and find relevance in our day-to-day lives. All the data analysis and processing in this study is performed on a local machine with 8.00 GB RAM and a 64-bit Operating System. The machine has a Intel® Core™2 Duo CPU P8600 @2.40 GHz processor. R version 3.1.3 and DB2 version 10.5 were used for development.

4.1 Datasets

During the study we used several spatial datasets containing different kinds of geometries. For performing the spatial analysis on the datasets, we imported them into individual DB2 tables. ST_IMPORT_SHAPE and IMPORT stored procedures were used to import the shapefile and *.csv* format files respectively. Since, all the datasets we use refer to locations in U.S., NAD83_SRS_1 spatial reference system was chosen to represent geometries in the datasets. The chosen SRS is based on

the GCS_NORTH_AMERICAN_1983 coordinate system. This section gives a detailed description of the datasets as well as the sources from where the datasets are acquired.

4.1.1 Taxi dataset

The New York City's Taxi and Limousine commission [34] captures regular data about the taxi trips in the five boroughs of New York City (NYC). The commission used this data for a twitter campaign called #metricmonday. It involves some visualizations made weekly using the taxi data. Recently a civic hacker and data junkie called Mr. Christopher M. Whong [35] used the Freedom of Information Law (FOIL) to obtain this dataset from NYC Taxi and Limousine Commission. There are two folders of data, Faredata_2013 and Tripdata_2013. Each folder contains 12 files in *.csv* format, one for each month of the year 2013 and has about 173 million entries in total. The datasets combined amount to 45.1 GBs of data.

The datasets contain several interesting attributes about the taxi trips in NYC. The most interesting ones being the pickup and dropoff location for each taxi trip. Amount of the tip paid in each taxi ride is also another interesting attribute. The amount of tip paid for a service is an important part of a workers daily wages in several professions and sometime there are social rules about the amount of tip to be paid for a service. The datasets also have a temporal aspect as they contain the information about the date and time of pickup and dropoff for each taxi ride. The datasets are described below.

- **TripData_2013**

  TripData_2013 dataset contains the data about the taxi trips of the yellow medallion taxis in NYC. It contains information on the attributes like medallion number of the taxi, license number of the driver, pickup and dropoff date and time of the trip, pickup and dropoff locations of the passengers etc. The first two attributes i.e. medallion and hack_license have been anonymized by Taxi and Limousine Commission (TLC) to protect the individual identities. Table 4.1 gives the description of attributes available in the dataset. The description in the table is

based upon the taxi rides information for the month of January 2013, considering the repetitive behavior of the taxi rides over the year, the summary for other months is expected to be similar.

Table 4.1: *`TripData_2013` dataset attributes description and summary. The table provides an overview of the structure of the dataset. Majority of variables are either `character` or `numeric` type. It can be seen from the range column that several attributes have values in unrealistic ranges. Hence, the dataset needs to be cleaned before being used for analysis.*

Attribute	Type	Range	Levels	Comment
medallion	Character	—	13426	taxicab unique identification number
hack_license	Character	—	32224	the driver's license number
vendor_id	Character	CMT or VTS	2	identifier for taxi vendor
rate_code	Integer	0-210	14	fare code for different type of rides
store_and_fwd_flag	Character	N or Y	2	purpose unidentified
pickup_datetime	Date Time	year 2013	—	date and time of passenger pickup
dropoff_datetime	Date Time	year 2013	—	date and time of passenger dropoff
passenger_count	Integer	0-255	10	number of passengers in the taxi
trip_time_in_secs	Integer	0-10800	—	time taken for the trip
trip_distance	Integer	0-100 miles	—	total distance covered on the trip
pickup_longitude	Numeric	-2771-112.40 degrees	—	longitude of passenger pickup location
pickup_latitude	Numeric	-3548-3310 degrees	—	latitude of passenger pickup location
dropoff_longitude	Numeric	-2351-2229	—	longitude of passenger dropoff location
dropoff_latitude	Numeric	-3548-3477	—	latitude of passenger dropoff location

- **FareData_2013**

  The dataset FareData_2013 contains information related to the fare for the taxi ride. It contains information like the medallion number of the taxi, license number of the driver, mode of payment for the fare, amount of tip paid, total amount paid etc. The medallion number and drivers license number have been anonymized in this dataset as well. Table 4.2 gives the description of attributes available in the dataset.

Table 4.2: *`FareData_2013` dataset attributes description and summary. The table provides an overview of the structure of the dataset. Majority of variables are either `Character` or `Numeric` in nature. It can be seen from the range column that several attributes have values in unrealistic ranges. Hence, the dataset needs to be cleaned before being used for analysis.*

Attribute	Type	Range	Levels	Comment
medallion	Character	—	13426	taxicab unique identification number
hack_license	Character	—	32224	the driver's license number
vendor_id	Character	CMT or VTS	2	identifier for taxi vendor
pickup_datetime	Date and Time	year 2013	—	date and time of the trip
payment_type	Character	—	5	mode of payment
fare_amount	Numeric	2.5-500$	—	amount of fare for the trip
surcharge	Numeric	0-12.5$	—	surcharge on fare amount
mta_tax	Numeric	0-0.5$	—	amount of tax
tip_amount	Numeric	0-200$	—	tip to the taxi driver
tolls_amount	Numeric	0-20$	—	amount of tolls
total_amount	Numeric	2.5-650$	—	total amount paid to the driver

The PICKUP_LATITUDE, PICKUP_LONGITUDE, DROPOFF_LATITUDE, and DROPOFF_LONGITUDE columns in the TripData_2013 dataset represent the pickup and dropoff locations of the taxis in the form of coordinates. After loading the datasets to the database, we transform these columns to ST_POINT geometries. Each record in the datasets TripData_2013 and FareData_2013 represent information about a single taxi ride. We merged the two datasets into one, named 'TRIPANDFARE' and removed the duplicate columns. The available datasets are huge in size and the data is completely real. Several attributes of the dataset might have been manually entered by the taxi cab driver. It is a possibility that the dataset has a large amount of corrupt values. This noise can be present in the form of outliers, unrealistic values or missing values. For this reason the datasets need cleaning. We perform the cleansing for the month of January 2013 which has 14,776,615 records and automate the process for the rest of the months.

- **Data cleaning**

  Since the datasets TripData_2013 and FareData_2013 are complementary to each other and each record in the TripData_2013 dataset corresponds to a record in FareData_2013 dataset, we clean the datasets in conjunction. From the data exploration process it is observed that the attributes: `medallion, hack_license`, `vendor_id`, `rate_code` and `store_and_forward_flag` have fairly distributed values and do not need further processing. Similarly the attributes , `pickup_datetime` and `dropoff_datetime` do not have corrupt records.

  Several taxi rides record an unrealistic number of passengers (`passenger_count`) for the ride. The website for NYC Taxi & Limousine Commission [36], informs that maximum number of passengers allowed in a yellow taxi cab is four in a four passenger taxicab and five for a five passenger taxi cab. But in our dataset the number of passengers range from 0 to 255 which indicates some error in data entry. We need to exclude the false records. The entries with less than 1 passenger and more than 5 passengers are deleted. In the process we removed 520,235 records from 14,776,615 records.

  We also remove the entries where the duration of trip (`trip_time_in_secs`) is less than 120 seconds (2 minutes) or more than 10,800 seconds (3 hours). This

condition removes additional 271,065 rows. There are some anomalies in the recorded distance of the trip (`trip_distance`). The entries with a recorded trip distance of less than half of a mile (0.5 miles) or more than 50 miles are removed. This condition eliminates additional 448,436 records.

The location attributes in the dataset i.e. `pickup_longitude, pickup_latitue, dropoff_longitude` and `dropoff_latitude` are most likely automatically generated at the start and the end of the trip. But the attributes unexpectedly have a lot of corrupt values. Several records are out of the valid longitude (-180°to 180°) and latitude range (-90°to 90°). We can simply remove these records by comparing the longitude and latitude values to the envelope of NYC. An envelope is the rectangular bounding box of a geometry. Figure 4.1 shows the envelope of New York City.

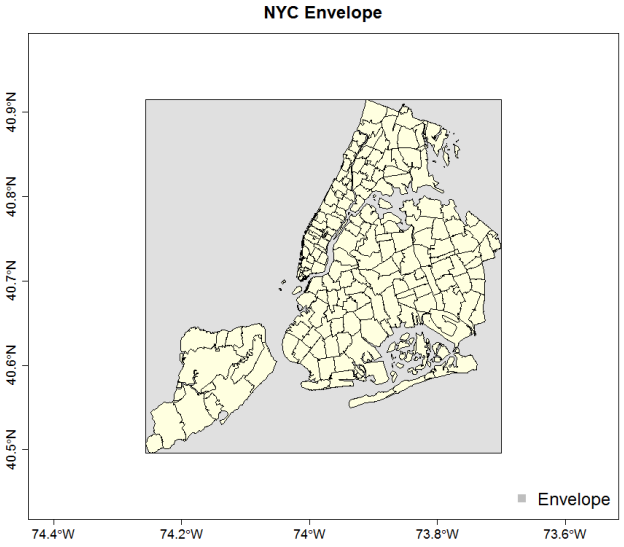


Figure 4.1: *This map represents the rectangular bounding box for New York City. It is also called as the envelope of the geometry. We use this envelope to exclude the false values of longitudes and latitudes.*

The above condition eliminates another 244,676 records in which the pickup or the dropoff location is falsely entered. Moreover, a number of taxi trips are recorded to originate or to end in the water body surrounding the NYC as indicated in the Figure 4.2. This behavior seems impractical, hence we need to exclude these entries as well. But comparing each taxi pickup and dropoff location to each ZIP code boundary is a very computation intensive task. Instead

we compare the locations with the convex hulls[1] of the two big land masses of NYC. The two convex hulls are shown in Figure 4.3. We eliminate the records in which either the pickup location or the dropoff location does not fall in either of the convex hulls. In the process we eliminate further 36290 records.

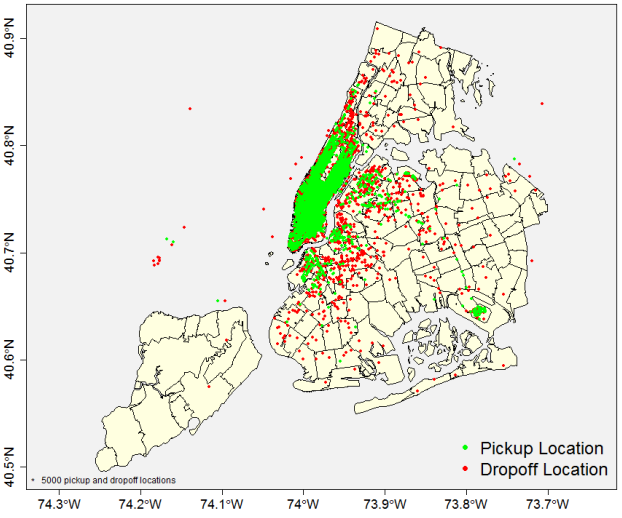


Figure 4.2: *The taxi trip pickup or dropoff locations which fall outside the land boundary need to be eliminated. These trips are falsely recorded to originate or end on the water covered area.*

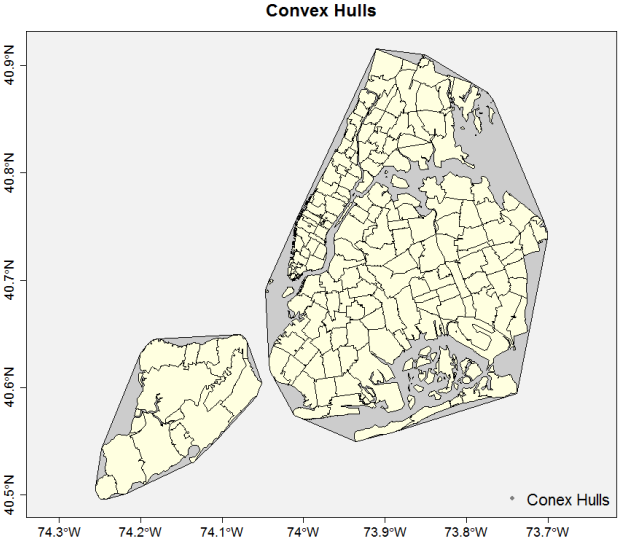


Figure 4.3: *The figure shows the convex hulls of two big land masses in NYC. We use these convex hulls to eliminate the taxi ride records which were registered to start or end on water covered areas.*

[1] A convex hull is the smallest convex polygon that contains all subgeometries.

There are a number of outliers in attributes of the FareData_2013 dataset. Some records have a very high value of the fare amount. These records might be true in real life but we remove because these outliers can disturb the averages of the various amounts. We remove the rows where:

- `fare_amount` is less than 0$ or greater than 300 $.
- `tip_amount` is less than 0$ or greater than 100 $.
- `total_amount` less than 0$ or is greater than 400 $.

The above three conditions remove another 33 records altogether. There are five levels of the attribute `PAYMENT_TYPE`, but the distribution is highly skewed for two levels namely 'CRD ' and 'CSH'. Rest three levels have less than 0.22% records altogether and hence are statistically insignificant. We keep only the two most common payment modes and can safely discard the other three. In the process we remove 29,786 records.

One important discovery in the data analysis process is regrading the attribute `TIP_AMOUNT`. Figure 4.4 shows the distribution of payment mode. It can be seen that the mode of payment has a distribution of approximately 55% and 45% between payment by card (CRD) and cash payment (CSH). A tip amount is indicated in approximately 55% of the taxi rides. But almost all the notified tips are for the taxi rides in which the payment was made by card. This fact seems very unrealistic that the passengers pay tip only when they pay by card. A possible explanation to this behavior could be that in the cases where payment is made by card the paid tip is automatically registered but the taxi cab driver does not acknowledge the tips paid in cash, probably to save on the tax amount. In the scenarios where TIP_AMOUNT is being analyzed, it is a better choice to consider only the trips where the payment mode was by card, so that a more realistic distribution of the amounts is observed.

4.1.2 ZIP code boundaries

New York city ZIP Code Tabulation areas (ZCTA5) is a polygon theme representing zip code areas in New York City. The dataset is provided by U.S. Department of

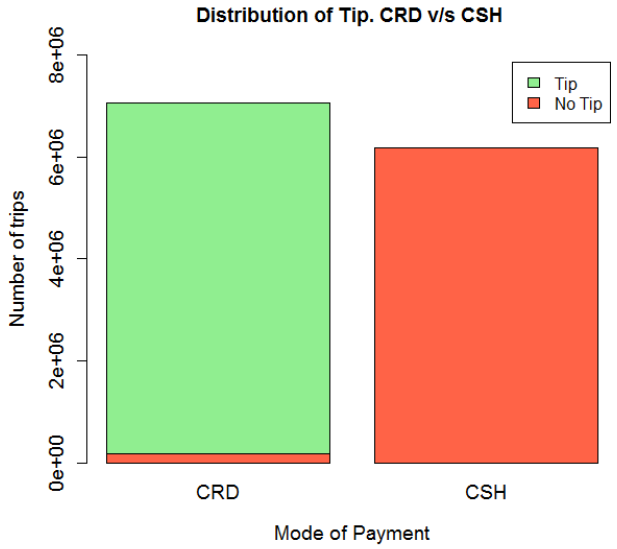


Figure 4.4: *The barplot represents the distribution of records in which the tip was paid. It can be clearly observed that no tip is acknowledged in most records where cash payment was made. At the same time almost all records in which the payment was made through card register a tip.*

Commerce, U.S. Census Bureau, and Geography Division. It contains the shape files for the boundaries of NYC zip codes and some additional attributes. This data allows us to analyze the statistics refined to the ZIP code area level. The file contains the details about 200 different ZIP code areas in NYC. Table 4.3 provides a summary of the dataset. The ZIP Code boundaries are plotted in Figure 4.5.

Table 4.3: *`ZIP_Code_Boundaries` dataset attributes description and summary. The dataset contains the boundary and other attribute information about 200 ZIP Codes in NYC. The table provides an overview of the structure of the dataset. All the variables are `Character` type in nature.*

Attribute	Type	Range	Levels	Comment
ZCTA5CE00	Character	—	200	geographic mail delivery area identification code
CLASSFP00	Character	B5	1	Federal Information Processing Standards (FIPS) 55 class code
MTFCC00	Character	G6350	1	MAF/TIGER feature class code
FUNSTAT00	Character	S	1	Census 2000 functional status

New york City is the most densely populated major city in the United states and it consists of five boroughs, each of which is a county of New York State. The five boroughs are as follows: Brooklyn, Queens, Manhattan, the Bronx, and Staten Island. The map in Figure 4.6 represents the five different boroughs of New York City.

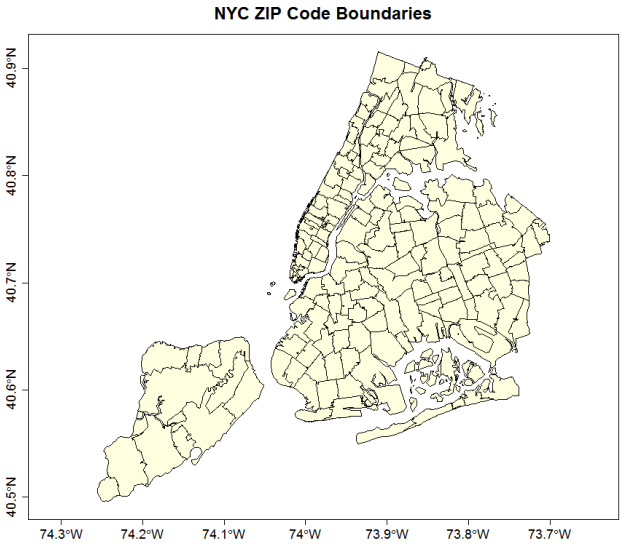


Figure 4.5: *This map represents 200 ZIP code areas in NYC. ZIP codes are a system of postal codes used by the United States Postal Service (USPS) since 1963. They help in a quick and efficient mail delivery.*

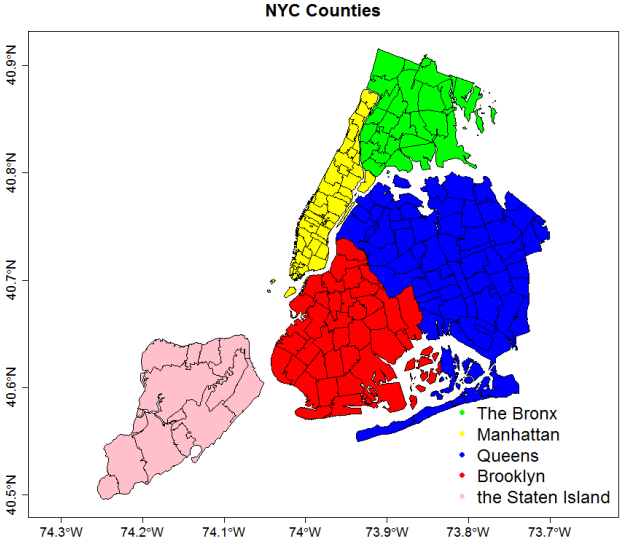


Figure 4.6: *This map represents the ZIP code areas in NYC. NYC is divided in five counties namely: Manhattan, Bronx, Queens, Brooklyn and Staten Islands. The map indicates each county in a different color.*

4.1.3 Airport Polygon

Airport polygon dataset contains information about two main airports in NYC i.e La Guardia Airport and John F. Kennedy International Airport. The dataset is public and is provided by Department of Information Technology & Telecommunications (DoITT). It is available in shapefile format for download from the NYC Open Data website [37]. The dataset contains the polygon boundaries of the airports in addition with other attributes like Name, Website, Geoserver, etc. A description and summary of the attributes is presented in Table 4.4. The plot in Figure 4.7 shows the location of airports in NYC.

Table 4.4: *Airport Polygon dataset attributes description and summary. The dataset contains only records, each corresponding to one airport in NYC. In the dataset, the location and the shape of the airport is represented through a polygon.*

Attribute	Type	Range	Levels	Comment
NAME	Character	—	2	name of the airport
GEOSERVER	Numeric	17879877-97393969		geoserver of the airport
URL	Character	—	2	URL of the airport
SHAPE_AREA	Numeric	0	1	variable unused
SHAPE_LEN	Numeric	0	1	variable unused

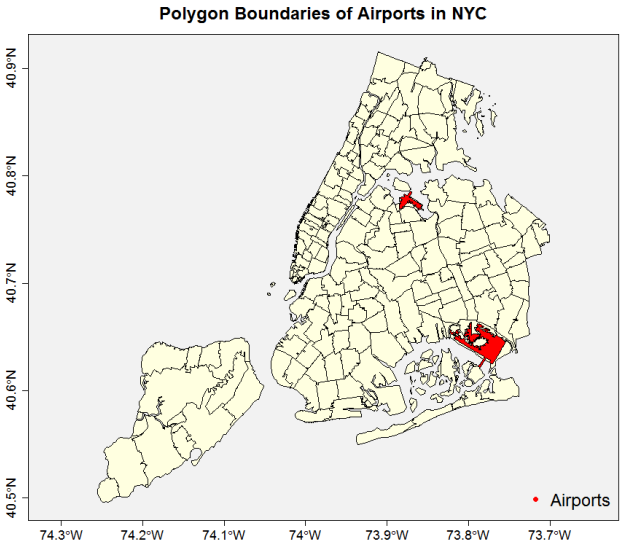


Figure 4.7: *This map represents the two airports in NYC i.e. La Guardia Airport and John F. Kennedy international Airport. The airport boundaries are shown in red.*

4.1.4 NYC Theaters

Theaters dataset is a listing of the theaters in NYC and has been publicly provided by Broadway Theater. It is available for download in shapefile format from the NYC Open Data website [38]. The dataset contains the information about the location of the 117 theaters in NYC. It has the attributes like the name, URL and address of the theater associated with the positional information. A description and summary of the attributes is presented in table 4.5. The map in figure 4.8 indicates the location of the theaters in NYC. It can be seen from the plot that most theaters are located in Manhattan.

Table 4.5: *`Theaters` dataset attribute description and summary. The dataset contains the information about the location of the 117 theaters in NYC. The table provides an overview of the structure of the dataset. All the variables except the ZIP code are `Character` type in nature.*

Attribute	Type	Range	Levels	Comment
NAME	Character	—	117	name of the theater
TEL	Character	—	87	telephone contact of the theater
URL	Character	—	117	link to the theater website
ADDRESS1	Character	—	114	part of the theater address
ADDRESS2	Character	—	6	part of the theater address
CITY	Character	—	3	city where the theater is situated
ZIP	Integer	—	22	corresponding ZIP code

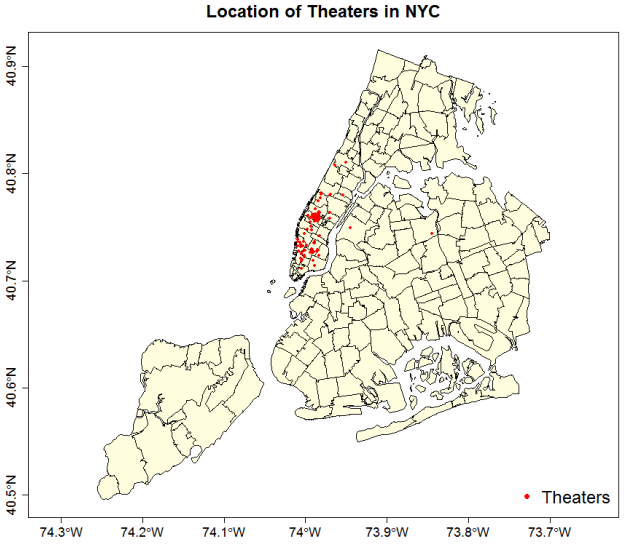


Figure 4.8: *This map represents the location of 117 theaters in NYC. It is evident from the plot that most of them are located in Manhattan.*

4.1.5 NYC Subway entrances

The Subway dataset is provided by Metropolitan Transportation Authority (MTA). It contains the location of 1645 subway entrances in NYC. It also contain the details about the name of the station and the lines for which the subway can be taken from that station.

Table 4.6: *`Subways` dataset attributes description and summary. The dataset contains information about 1645 subway entrances in NYC. The table provides an overview of the structure of the dataset. All the variables are `Character` type in nature.*

Attribute	Type	Range	Levels	Comment
NAME	Character	—	1645	name of the subway station
URL	Character	http://www.mta.info/nyct/service/	1	link to MTA website
LINE	Character	—	93	lines for which you can take the subway

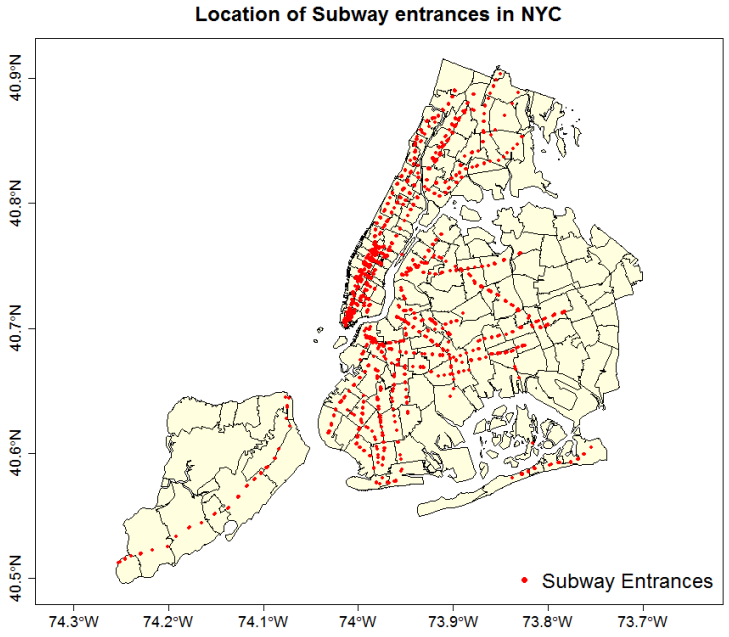


Figure 4.9: *This map indicates the location of subway stations in NYC.*

4.2 Predict tip amount

This use case aims to predict the amount of tip that a taxi driver is expected to receive at the end of the trip. We wish to showcase the influence of geospatial information over the prediction and the scalabilty of the used spatial operations. The `TIP_AMOUNT` column of the TaxiAndFare table is categorized into two classes. The first class "$> 5D$"

represents the taxi trips in which the driver got a tip of more than 5$. The second class *"*$< 5D$*"* represents the taxi trips in which the driver got a tip of less than 5$. Depending upon the normal row attributes and added geospatial information we try to predict the tip amount (*"*$> 5D$*"* or *"*$< 5D$*"*).

4.2.1 Data preparation

Data preparation is a fundamental stage of data analytics process and is often the longest and most difficult part. It is generally defined as the process of cleaning and transforming the selected data and is likely to be performed multiple times and not in any prescribed order. The data is formatted, for example, by joining or by aggregation so that it is suitable for data analysis. Data preparation involves a number of tasks including attribute selection, data transformation and determining the area to examine. In general, we need to transform our data from which we build our models. The knowledge about the data gained through exploratory data analysis is very helpful in taking the decisions while cleaning and preparing the data [39]. Parts of R code are provided in the document as and when deemed required, the complete code for the process can be viewed in Appendix A.

- **TaxiAndFare dataset**

  We take all the records from the TaxiAndFare dataset for the month of January 2013, which has $\sim$ 15 million rows.

  - **Condition**

    We consider only those entries in which the payment was made using a card for the reason mentioned in Section 4.1.1. Using the selection capability over ida.data.frame objects, this can be easily done as shown in the statement below:

    ```
    # Consider only the trips where payment mode is card.
    TripAndFare<-TripAndFare[TripAndFare$PAYMENT_TYPE=="CRD",]
    ```

  - **Categorize Tip Amount**

`TIP_AMOUNT` column of the TaxiAndFare table is categorized into two classes. The labels, either *"$> 5D$"* or *"$< 5D$"* depending upon the amount of tip received are added to a new column called 'TIP_RANGE'.

```
TripAndFare$TIP_RANGE<-ifelse(TripAndFare$TIP_AMOUNT<5,"<5D",">5D")
```

– **Remove columns**

1. `TRIP_ID`: Identifier for the TaxiAndFare table. This column does not contribute useful information so it can be removed from further processing.
2. `MEDALLION`: This attribute is the unique identifier for the taxicab. This column can be removed.
3. `HACK_LICENSE,VENDOR_ID`: Identifier for the driver's license number and the taxi vendor respectively.
4. `RATE_CODE`: This attribute has a value of 1 for 98% cases, hence can be safely removed from further analysis.
5. `STORE_AND_FWD_FLAG`: This attribute has an unidentifiable purpose.
6. `TRIP_TIME_IN_SECS`: The value of this attribute is usually not known before the end of the trip, hence this column is not useful for this predictive problem.
7. `PICKUP_LONGITUDE, PICKUP_LATITUDE` : The information related to the pickup location is already available from the `P_LOCATION` column. Hence, these two columns can be deleted.
8. `DROPOFF_LONGITUDE, DROPOFF_LATITUDE`: The information about the dropoff location is already available from the `D_LOCATION` column.
9. `FARE_AMOUNT, TOTAL_AMOUNT`: Since the `TIP_AMOUNT` is usually a percentage amount of the fare amount. If the predicted variable is in direct relation with an input variable, we might get unrealistic results for a prediction problem. Hence, these two columns should not be kept for this predictive use case.
10. `TIP_AMOUNT`: We already added a column called 'TIP_RANGE ' which contains information about the tip for the trip.
11. `TRIP_DISTANCE`: The amount of the tip paid is directly related to the distance traveled in the taxi. Hence we will remove this attribute.

12. `PICKUP_DATETIME`: Since the tip is paid at dropoff from the taxi. We consider the dropoff date and time and ignore this attribute.
13. `MAT_TAX` and `TOLLS_AMOUNT`: These attributes have no apparent influence in this use-case.

– **Add columns**

These attributes are derived from the `DROPOFF_DATETIME` column. In the present form the `DROPOFF_DATETIME` column is not useful, as it indicates a complete time stamp. We divide the time stamp into several potentially useful parts.

1. `WEEKDAY`: The day of the trip might have an impact on the amount of tip payed by the customer. In a separate analysis it was observed that the customers pay more tip on weekdays in comparison to weekends. To derive this information we use the *weekdays* function provided by the ibmdbR package. This function returns the name of the day for a given time stamp.
2. `DATE`: People might pay more tip at the start of the month or vice-versa, depending upon the when they get their salaries. This information is derived using this *substr* function.
3. `HOUR`: It is likely that the taxi trips at the odd hours in a day might receive extra tip. This information is also derived using this *substr* function.

Since we have extracted the required information from the `DROPOFF_DATETIME` column, we can safely remove it from the further analysis. We add two more columns namely `PICKUP` and `DROPOFF` as the place holders for the derived spatial information. The purpose of these two columns is explained in the Section 4.2.2.

- **THEATERS dataset**

We assume that the taxi trips having dropoff location within 100 meters radius of a theater location are destined for the corresponding theater. To identify these taxi rides we need to include the area with the specified width around each theater location. We accomplish this with the help of the *idaBuffer* function as shown

below. In the process we also remove the undesired columns from the object and keep only two columns namely 'THEATER_ID 'and 'THEATER_LOCATION ' (by explicitly specifying the subset in the *idaBuffer* function).

```
theaters_area<-idaBuffer(theaters[,c("THEATER_ID","THEATER_LOCATION")],
                  col=theaters$THEATER_LOCATION,
                  width=width,
                  tableName="THEATERS_AREA")
```

We provide the *theaters* ida.data.frame object as input to the *idaBuffer* function. If there are more than one spatial columns in the data.frame we need to specify the column around which we wish to draw our geometry (*col* parameter). With the help of the *width* parameter we specify the distance to be used for the buffer around the geometry. Last parameter is the name of the resulting view. If the *tableName* parameter is not specified, a randomly generated table name is assigned to the resulting view.

The above statement results into a new ida.data.frame object (theaters_area) with an additional column named 'BUFFER_THEATER_LOCATION', as shown in the statement below. This column contains information about the polygon of the specified width created around the theater location.

```
> head(theaters_area)
  THEATER_ID THEATER_LOCATION BUFFER_THEATER_LOCATION
1          1         ST_POINT             ST_GEOMETRY
2          2         ST_POINT             ST_GEOMETRY
3          3         ST_POINT             ST_GEOMETRY
4          4         ST_POINT             ST_GEOMETRY
5          5         ST_POINT             ST_GEOMETRY
6          6         ST_POINT             ST_GEOMETRY
```

- **AIRPORTS dataset**

  In the airports dataset we have the location of the two airports in the form of polygon geometry. But it might be possible that the taxis drop the passengers in the vicinity of the airports and not directly inside the airport building, depending upon the design of the airport. To account for those cases, we create an area

with a 400 meters distance around each point on the airport geometries. We accomplish this with the help of the *idaBuffer* function as shown below:

```
airports_area<-idaBuffer(airports.ida[,c('AIRPORT_ID','NAME','AIRPORT_BOUNDARY')],
                         col=airports.ida$AIRPORT_BOUNDARY,
                         width=width,
                         tableName="AIRPORTS_AREA")
```

We provide the *airports.ida* ida.data.frame object as input to the *idaBuffer* function. We keep only three columns namely 'AIPORT_ID', 'NAME' and 'AIRPORT_BOUNDARY' from the AIRPORTS dataset. The above statement results into a new ida.data.frame object with an additional column. The column is named 'BUFFER_AIRPORT_BOUNDARY' by default. The output of the above statement can be seen below.

```
> head(airports_area)
 AIRPORT_ID          NAME        AIRPORT_BOUNDARY       BUFFER_AIRPORT_BOUNDARY
1      1     La Guardia Air...     ST_MULTIPOLYGON           ST_GEOMETRY
2      2     John F. Kenned...     ST_MULTIPOLYGON           ST_GEOMETRY
```

- **MAINAREAS dataset**

  This is a manually created dataset. The dataset consists of four records, each representing an area around a location in NYC. These are the locations from where people hail most number of taxis. The contents of the MAINAREAS dataset are as shown below:

```
> head(main_areas)
      NAME             MAIN_LOCATION_ID    LOCATION   BUFFER_LOCATION
1 Penn Station                 1             ST_POINT     ST_GEOMETRY
2 Grand Central                2             ST_POINT     ST_GEOMETRY
3 Columbus Circle              3             ST_POINT     ST_GEOMETRY
4 Lexington Avanue             4             ST_POINT     ST_GEOMETRY
```

4.2.2 Geospatial information

In this section we add geospatial information to our dataset using the developed geospatial operations.

- Airports: Intersect pickup and dropoff location of each taxi trip with the area defined for the airports. If the *idaIntersects* function returns 1, mark the corresponding column (`PICKUP` or `DROPOFF`) for that row as "AIRPORT". In this process we identify the taxi trips which are potentially destined or originated from the area around airports. The statements below show the process for dropoff locations, the same process is followed for pickup locations as well:

```
> GEOINFO_AIRPORT<-idaIntersects(TripAndFare,
+                     airports_area[,c("BUFFER_AIRPORT_BOUNDARY")],
+                     by.x=TripAndFare$D_LOCATION,
+                     by.y=airports_area$BUFFER_AIRPORT_BOUNDARY,
+                     tableName="GEOINFO_AIRPORT",
+                     returnDense=F)

> GEOINFO_AIRPORT$DROPOFF<-ifelse(GEOINFO_AIRPORT$INTERSECTS_TRIPANDFARE
_JAN_AIRPORTAREA==1,"Airport","NA")
> head(GEOINFO_AIRPORT,n=20)
 TRIP_ID  PAS.. PAY.. SUR.. P_LOC.. D_LOC.. WEEKDA DATE HOUR TIP_RANGE DROPOFF
1  103..   1    CRD    0.0  ST_PT  ST_PT    Wed     30   12       <5D    <NA>
. .    .     .      .     .      .       .      .    .         .     .
10 140..   2    CRD    0.0  ST_PT  ST_PT    Sun     27   18       <5D    <NA>
11 971..   2    CRD    0.0  ST_PT  ST_PT    Wed     23   17       >5D   Airport
. .    .     .      .     .      .       .      .    .         .     .
20 103..   1    CRD    0.0  ST_PT  ST_PT    Wed     30   10       <5D    <NA>
```

- Theaters: Same process for Theaters as for the Airports dataset is used. If the *idaIntersects* function returns 1, mark the corresponding column (`PICKUP` or `DROPOFF`) for that row as "THEATER".

- Main Areas: Same process for Main Areas as for the Airports dataset is used. If the *idaIntersects* function returns 1, mark the corresponding column (`PICKUP` or `DROPOFF`) for that row as "MAINAREA".

We are done with our data preparation process and we have successfully incorporated the available geospatial information into our dataset with $\sim$ 15 million rows. Here we

saw that the spatial operations can be successfully applied over large spatial datasets of various kinds. This indicates the scalability of the solution for geospatial analysis and processing.

4.2.3 Modeling

With the help of the *as.data.frame* method we fetch our resultant dataset into the R environment. The relevant statements are given in the code block below. We can see at the incorporated geospatial information using the *head* funtion. the columns 'DROPOFF' and 'PICKUP' hold the calculated geospatial information. This dataset now is like any other R data.frame object and we can use models of our choice like, random forests, support vector machine, boosted decision trees, neural networks, etc. on the dataset. The Rattle GUI is used in this study for the modeling purpose [39]. It is built in the statistical language R. It allows us to rapidly work through the data processing, modeling and evaluation phases of a data mining project.

```
> R_Geo_Taxi <- as.data.frame(GEO_TAXI)
> head(R_Geo_Taxi)
  TRIP_ID PASSENGER_COUNT PAY.. SURCHARGE  WEEKDAY DATE HOUR TIP_RANGE DROPOFF PICKUP
1 11005745               1   CRD       0.5  Tuesday   29   20       <5D    <NA> MAINAREA
2 10512045               4   CRD       0.5 Thursday   31   00       <5D THEATER MAINAREA
3 10512045               4   CRD       0.5 Thursday   31   00       <5D THEATER MAINAREA
4 10512045               4   CRD       0.5 Thursday   31   00       <5D THEATER MAINAREA
5 10512045               4   CRD       0.5 Thursday   31   00       <5D THEATER MAINAREA
6 14115377               1   CRD       0.0   Sunday   27   10       <5D    <NA> MAINAREA
```

In this use case we chose to model using a conditional tree. We have divided our dataset into a proportion of 70/15/15 i.e. 70% of the records are used as the training dataset and 15% each for validation and testing. TIP _RANGE is set as the target variable and rest others as input variables. We accept the default tuning parameters in the Rattle interface as our aim is to not obtain a perfect model but to see the influence of geospatial information on the model. We build the conditional tree using 100,000 rows from the resulting dataset and evaluate the generated tree on the test set. The following results were obtained:

```
Fehlermatrix für das Modell Entscheidungsstruktur bei R_Geo_taxi_1L [Test] (Zähungen):
     Vorausgesagt
Ist     <5D   >5D
  <5D 13534   188
  >5D   354   924

Error matrix for the Entscheidungsstruktur model on R_Geo_taxi_1L [Test] (proportions):
      Predicted
Actual  <5D  >5D Error
   <5D 0.90 0.01  0.01
   >5D 0.02 0.06  0.28

Globaler Fehler: 0.03613333, Averaged class error: 0.09727719
```

The model has a very low error rate and performs very well. It has a very low number of false positives and false negatives. To test the influence of geospatial information we remove the columns representing geospatial information i.e. 'PICKUP' and 'DROPOFF' and train the model again with the same parameters. The obtained results are as shown below.

```
Error matrix for the Decision Tree model on R_Geo_taxi_1L [test] (counts):
     Predicted
Actual <5D   >5D
  <5D 13709    13
  >5D  1157   121

Error matrix for the Decision Tree model on R_Geo_taxi_1L [test] (proportions):
      Predicted
Actual  <5D  >5D Error
   <5D 0.91 0.00  0.00
   >5D 0.08 0.01  0.91

Overall error: 0.078, Averaged class error: 0.08742176
```

Here we can see that the model with geospatial information is considerably better than the model built without the geospatial information. This indicates the positive influence of including geospatial information in our analysis process.

4.3 Predict tip percentage

Another interesting usecase is to predict the percentage of the tip that a taxi driver is expected to receive at the end of the ride. Amount of tip is usually a percentage of the basic amount to be paid by the customer. Therefore, we add a new column ('TIP_PERCENTAGE') to the dataset which represents the percentage of tip for that record.

4.3.1 Data preparation

The data preparation for the use case is very much similar to the previous one. Since, the use case is related to TIP_AMOUNT, we consider only the records where the payment was made by card. In this use case, instead of categorizing the absolute amount of tip, we categorize the percentage of tip. The tip amount is a percentage of the sum of three amounts i.e. FARE_AMOUNT, SURCHARGE and MAT_TAX. The statements shown below calculate the tip percentage. It is then categorized it into two classes with labels '< 20%' and '> 20%'. They represent whether tip was less than 20% or more, respectively.

```
# Calculate Tip Percentage
> TripAndFare$TIP_PERC<-(TripAndFare$TIP_AMOUNT*100)/(TripAndFare$FARE_AMOUNT
                                                      +TripAndFare$SURCHARGE
                                                      +TripAndFare$MAT_TAX)
> # Categorize Tip Amount
> TripAndFare$TIP_PERCENTAGE<-ifelse(TripAndFare$TIP_PERC<20,"<20%",">20%")
```

We added the geospatial information to the dataset in the same way as the previous usecase. The sample of the final dataset used for modeling is as shown below:

```
> head(Geo_Predict_Tip)
PASSENGER_COUNT PAYMENT_TYPE SURCHARGE TIP_PERCENTAGE WEEKDAY DATE HOUR DROPOFF  PICKUP
1               1          CRD       0.5           <20%  Sunday   13   04    <NA>    <NA>
2               2          CRD       0.5           <20%  Sunday   13   04    <NA>    <NA>
3               2          CRD       0.5           <20%  Sunday   13   04    <NA>    <NA>
```

```
4            1          CRD       0.5          <20%  Sunday   13   04    <NA> THEATER
5            5          CRD       0.5          >20%  Sunday   13   04    <NA>    <NA>
6            3          CRD       0.0          >20%  Sunday   13   11    <NA>    <NA>
```

4.3.2 Modeling

For this use case we chose to build a random forest model using 200,000 records of the final dataset. We selected 200 trees in the forest and number of variables to be considered at each split to be 3. The results of the modeling process are as shown below. The model performs quite well. It can be noted here that the columns bearing the geospatial information are of significant importance in the model.

```
   Type of random forest: classification
   Number of trees: 200
   No. of variables tried at each split: 3
   OOB estimate of  error rate: 18.44%
Confusion matrix:
      <20%  >20% class.error
<20% 91333  7672  0.07749104
>20% 18139 22856  0.44246859

Variable Importance
===================
                 <20%   >20% MeanDecreaseAccuracy MeanDecreaseGini
PICKUP          60.13 141.91               145.55           808.57
PASSENGER_COUNT 18.88 126.07                97.60          1372.50
HOUR            31.85  57.20                81.34          5269.09
DROPOFF         25.40  97.23                77.07           505.81
DATE            25.81  31.32                59.93          5200.11
SURCHARGE       32.64  27.17                47.18           444.88
WEEKDAY         14.67  23.63                18.63          1538.79
```

4.4 Predict taxi pickups

This use case allows us to predict the number of taxi pickups in user defined regions (in the following referred to as zones) at a given day and hour in future. Geospatial analysis allows us to formulate such use cases which would not be possible otherwise. We use the existing pickup time information available through the 'PICKUP_DATEIME' attribute. We add the geospatial information by identifying the zone from which the taxi trip started. We collect this information by intersecting the polygon boundaries of the zones with the pickup locations of the taxis.

We can use regression to predict the number of taxi pickups from a particular zone at a day and hour in future. The outcome might be useful for the taxi dispatchers and possibly the taxi drivers so that they can efficiently position the taxis in the zones where they are required the most. It might save on the waiting time and fuel costs. The target variable for the model is the number of taxi pickups per zone aggregated by the day and time.

4.4.1 Data preparation

- **TaxiAndFare dataset**

  We take 10 Million records from the TaxiAndFare dataset for the month of January 2013.

  - **Remove columns**

    For this use case we don't require attributes other than the following three. Hence we remove all other attributes from the dataset.

    1. `TRIP_ID`: Identifier for the TaxiAndFare table. This attribute represents each individual taxi trips and is later used to aggregate the number of trips.
    2. `PICKUP_DATEIME`: Indicates the date and time of the taxi pickup.
    3. `P_LOCATION`: Represents the location of taxi pickup.

  - **Add columns**

    These attributes are derived from the `PICKUP_DATETIME` column. In the present form the `PICKUP_DATETIME` column is not useful as it indicates a

complete time stamp. We divide the time stamp into several potentially useful parts.

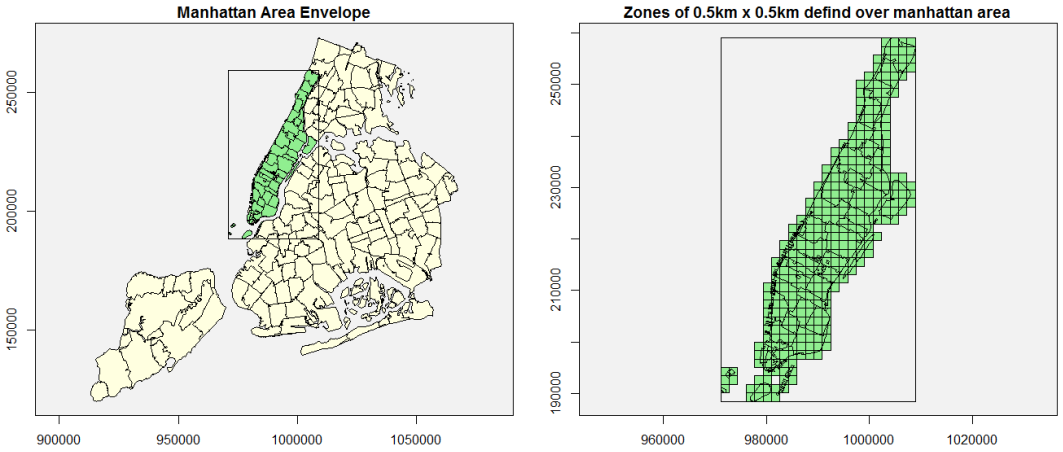
1. `WEEKDAY`: This attribute is derived using the *weekdays* function of the ibmdbR package. This function returns the name of the day for a given time stamp.
2. `DATE`: This attribute is derived using the *substr* function. It returns the date from for a given time stamp. We do not require the complete date with month and year as we are considering only the rides for the month of January.
3. `HOUR`: This information is derived using this *substr* function. By selecting just the hour from the complete pickup time, we aggregate all the taxi rides in that hour together.

Since we have extracted the required information from the `PICKUP_DATETIME` column we can remove it from the further analysis. The processed form of the TaxiAndFare dataset is as shown below:

```
> head(TripAndFare)
  TRIP_ID P_LOCATION  WEEKDAY DATE HOUR
1 7514559   ST_POINT Saturday   26   03
2 8210002   ST_POINT Thursday   24   17
3 7394753   ST_POINT   Friday   18   01
4 8507736   ST_POINT   Monday   21   14
5 5570776   ST_POINT   Sunday   20   09
6 5923051   ST_POINT  Tuesday   22   12
```

- **ZONES dataset**

  ZONES dataset is manually created by dividing the area of the Manhattan borough into zones. We have considered only Manhattan because more than 95% of all the taxi trips fall into that region of NYC and hence it is viable choice for this use case. The selected area is then divided into arbitrary zones of approximately 0.5 km x 0.5 km width. Here a random zone size has been selected for the purpose of the use case. The user might vary the zone size or specify customized zones of interest depending upon the application. The selected area



(a) *Envelope of Manhattan* ***(b)*** *Zones created in the Manhattan area*

Figure 4.10: *In this use case we consider only the taxis which originated from Manhattan. (a) shows the envelope of Manhattan area in NYC. (b) represents the randomly created zones in Manhattan.*

of NYC (Manhattan) and the created zones are shown in Figure 4.10. The area of Manhattan is divided into 328 zones of equal size. The dimension and first six records of the zones dataset are shown below:

```
> zones<-ida.data.frame("ZONES")
> dim(zones)
[1] 328   2
> head(zones)
  ZONE_ID   ZONE_BOUNDARY
1       1 ST_MULTIPOLYGON
2       2 ST_MULTIPOLYGON
3       3 ST_MULTIPOLYGON
4       4 ST_MULTIPOLYGON
5       5 ST_MULTIPOLYGON
6       6 ST_MULTIPOLYGON
```

4.4.2 Geospatial information

We add geospatial information to our dataset by comparing the taxi pickup locations with the zone boundaries. A new ida.data.frame object called 'taxi_zones' is created which contains with other attributes the id of the zone (ZONE_ID) from which the

taxi ride started. The statements below show the process. The inessential columns are removed from the dataset definition.

```
> taxi_zones <- idaIntersects(TripAndFare,
                          zones[,c("ZONE_ID","ZONE_BOUNDARY")],
                          by.x=TripAndFare$P_LOCATION,
                          tableName="TAXI_ZONES",
                          returnDense=T)

> taxi_zones <- taxi_zones[,!names(taxi_zones) %in% c("ZONE_BOUNDARY"
,"INTERSECTS_TRIPANDFARE_1M_ZONES","P_LOCATION")]
> head(taxi_zones)
   TRIP_ID  WEEKDAY DATE HOUR ZONE_ID
1 13822757   Friday   25   12       3
2  8314669   Monday   21   13       5
3   476589  Tuesday   15   13       5
4 14713538  Tuesday   01   13       5
5 10724540 Thursday   31   20       5
6  4046670  Tuesday   15   18       5
```

From the output of the *head* function, we can see that we have successfully added the information regarding the taxi zones using the spatial operations. We then fetch the resultant dataset into the R environment for modeling purpose. The dataset is then grouped with the elements DATE, ZONE_ID, WEEKDAY and HOUR. A sample of the dataset is shown below. The 'Pickup_Count' column indicates the the total number of taxi pickups in a zone on a specific day and hour.

```
> head(PICKUP_ZONE_AGGREGATE)
 Date Weekday Hour Zone Pickup_Count
10  1 Tuesday  00    1     2
18  1 Tuesday  01    1     5
29  1 Tuesday  02    1     3
37  1 Tuesday  03    1     9
43  1 Tuesday  04    1     4
54  1 Tuesday  05    1     1
```

The user can now use the modeling techniques of choice to predict the number of taxi pickups. Further information like the weather data, frequency of subways in the zones, etc. can be added to improve the prediction model. Our aim here is to showcase the scalabilty of the solution and the possibilities of different kinds of analysis that geospatial analysis offers.

Chapter 5

Conclusion

In Chapter 1, we stated the growing importance of geospatial analysis and how it opens up unforeseen opportunities. Geospatial analysis can significantly benefit the businesses, governments, environmental institutions, individuals and other organizations. We setup the objectives for the study as, to familiarize the reader with the geospatial discipline, develope a scalable solution for geospatial analysis, and implementation of the solution on practical problems. In Chapter 2, we imparted understanding about the main principals and concepts of the geospatial domain in a succinct form without overwhelming the reader with information. Spatial database systems and spatial resources were described. The concepts mentioned are basic building blocks for advanced manoeuvres in geospatial domain.

The prime goal of the thesis was to develop a scalable solution for geospatial analysis and processing. Several design goals were defined for an effective and efficient solution. We successfully developed a scalable solution which can be constructively used with large spatial datasets. Developed spatial functions maintain adherence with the `sp` package for the purpose of adaptability. If required, the spatial data can be imported from the database tables into R environment. The imported data conform to spatial data classes defined in the `sp` package. Moreover, we developed the spatial functions by coalescing with `ibmdbR` package and utilizing the defined classes. This allows us to use the wide range of statistical analysis functions provided by the `ibmdbR` package. The operations are intuitive in nature and can also be conveniently used by the non experts in the geospatial domain. Developed spatial operations use lazy loading to load

only those parts of data that are actually required. This supports a faster approach of data analysis and processing.

In Chapter 4, we successfully employed the solution on several use cases. The use cases chosen for the study have applied applications. We were able to use the spatial operations on a wide range of datasets of different types and sizes. During the course, we observed notable enhancement in analysis capabilities and possibilities by utilizing geospatial information. We developed the spatial operations which were perceived to be most commonly used for geospatial analysis. For specific use cases that require certain spatial operations, they can be adequately added using the provided framework.

Bibliography

[1] Wlcg.web.cern.ch. Welcome to the worldwide lhc computing grid — wlcg., 2015. URL http://wlcg.web.cern.ch/. Accessed: 2015-02-01.

[2] Blog.twitter.com. Measuring Tweets. Twitter Blog, 2010. URL https://blog.twitter.com/2010/measuring-tweets. Accessed: 2015-05-10.

[3] Internetstats.com. Twitter Usage Statistics - Internet Live Stats, 2011. URL http://www.internetlivestats.com/twitter-statistics/. Accessed: 2015-05-10.

[4] Pressroom.usc.edu. Twitter And Privacy: Nearly One-In-Five Tweets Divulge User Location Through Geotagging Or Metadata, 2015. URL https://pressroom.usc.edu/twitter-and-privacy-nearly-one-in-five-tweets.

[5] Fosca et al. Giannotti. Trajectory pattern analysis for urban traffic. *Proceedings of the Second International Workshop on Computational Transportation Science*, ACM:43–47, 2009.

[6] N.; Sathiyabama S.; Geetha, R.; Sumathi. A survey of spatial, temporal and spatio-temporal data mining. *Journal of Computer Applications*, 1, Oct-Dec 2008.

[7] Deren Li and Shuliang Wang. Concepts. Principals and Applications of Spatial Data Mining And Knowledge Discovery . pages 27–29, August 2005.

[8] Brian Klinkenberg. The true cost of spatial data in canada. *The canadian Geographer*, pages 37–49, 2003.

[9] Jeremy Mennis and Diansheng Guo. Spatial data mining and geographic knowledge discovery-an introduction. *Computers, Environment and Urban Systems*, (33(6)):403–408, 2009.

[10] Jiawei Han and Micheline Kimber. Data mining: Concepts and techniques. Second Edition, 2006.

[11] Ph.ucla.edu. John snow and the broad street pump: On the trail of an epidemic, 2012. URL http://www.ph.ucla.edu/epi/snow/snowcricketarticle.html. Accessed: 2015-03-10.

[12] Udel.edu. R tops data mining software poll, 2015. URL https://www.udel.edu/johnmack/frec682/cholera/snow_map.png. Accessed: 2015-03-11.

[13] Support.esri.com. Thiessen polygons - gis dictionary, 2015. URL http://support.esri.com/en/knowledgebase/GISDictionary/term/Thiessen%20polygons. Accessed: 2015-03-11.

[14] Udel.edu. GIS analysis of snow's london cholera map, 2015. URL http://www.udel.edu/johnmack/frec682/cholera/cholera2.html.

[15] Alex H. and Kate B. MAPPING AND ANALYSING CRIME DATA . *London, Taylor and Francis*, pages 9–22, 2001.

[16] Haining Robert. Spatial data analysis: Theory and practice. *Cambridge: Cambridge University Press*, pages 37–49, 2003.

[17] Spatial extender user's guide and reference.

[18] Stan Openshaw. *Ecological fallacies and the analysis of areal census data*, volume 16.1. 1984.

[19] E-education.psu.edu. Understanding spatial fallacies — the learner's guide to geospatial analysis, 2015. URL https://www.e-education.psu.edu/sgam/node/214. Accessed: 2015-05-24.

[20] Mathworld.wolfram.com. Coordinate System – from Wolfram MathWorld, 2015. URL http://mathworld.wolfram.com/CoordinateSystem.html. Accessed: 2015-03-16.

[21] R-project.org. R: What is R?, 2015. URL http://www.r-project.org/about.html. Accessed: 2015-03-11.

[22] Open Geospatial Consortium Inc. OpenGIS Simple Features Specification For SQL. version (1.1), May 1999. URL http://www.opengeospatial.org/standards.

[23] Open Geospatial Consortium Inc. OpenGIS Implementation Standard for Geographic information- Simple feature access - Part 1: Common architecture . version (1.2.1):51–62, May 2011. URL http://www.opengeospatial.org/standards/sfa.

[24] Open Geospatial Consortium Inc. OpenGIS Geography Markup Language (GML) Encoding Standard. version (3.2.1):51–62, August 2007. URL http://www.opengeospatial.org/standards/gml.

[25] www.r project.org/. The R Project for Statistical Computing, 2015. URL http://www.r-project.org/. Accessed: 2015-03-11.

[26] Rexeranalytics.com. 2011 data miner survey summary, 2011. URL http://www.rexeranalytics.com/Data-Miner-Survey-Results-2011.html. Accessed: 2015-03-11.

[27] David Smith. R tops data mining software poll, 2012. URL http://java.sys-con.com/node/2288420l. Accessed: 2015-03-11.

[28] Edzer J. Pebesma and Roger S. Bivand. Classes and methods for spatial data in R. *R News*, 5(2):9–13, November 2005. URL http://CRAN.R-project.org/doc/Rnews/.

[29] Esri.com. Esri - GIS Mapping Software, Solutions, Services, Map Apps, and Data, 2015. URL http://www.esri.com/. Accessed: 2015-03-15.

[30] Esri shapefile technical description. July 1998. URL http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.

[31] Esri. Arcgis desktop 9.3 help - geoprocessing considerations for shapefile output. April 2009. URL http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=Geoprocessing%20considerations%20for%20shapefile%20output.

[32] IBM Corporation. *ibmdbR: IBM in-Database Analytics for R*, 2015. URL http://CRAN.R-project.org/package=ibmdbR. R package version 1.36.7.

[33] Roger Bivand and Colin Rundel. *rgeos: Interface to Geometry Engine - Open Source (GEOS)*, 2014. URL http://CRAN.R-project.org/package=rgeos. R package version 0.3-8.

[34] Nyc.gov. NYC Taxi & Limousine Commission, 2015. URL http://www.nyc.gov/html/tlc/html/home/home.shtml. Accessed: 2015-04-15.

[35] Chris Whong. Chriswhong.com, 2015. URL http://chriswhong.com/. Accessed: 2015-03-23.

[36] Nyc.gov. NYC Taxi & Limousine Commission - Passenger Information, 2015. URL http://www.nyc.gov/html/tlc/html/passenger/passenger.shtmll. Accessed: 2015-04-15.

[37] Department of Information Technology & Telecommunications (DoITT). Airport Polygon, 2014. URL https://data.cityofnewyork.us/d/xfhz-rhsk?category=City-Government&view_name=Airport-Polygon. Accessed: 2015-05-24.

[38] NYC Open Data. Theater, 2015. URL https://data.cityofnewyork.us/Recreation/Theaters/kdu2-865w. Accessed: 2015-03-23.

[39] G. J. Williams. *Data Mining with Rattle and R. The art of Excavating Data for Kmowledge Discovery. Use R!* Springer, 2011.

Appendix A

R-code

Usecase- Predict Tip Amount

```
1
2  ##################################################################
3  # Connect to the database
4  # DSN, Userid and Password as parameters
5  ##################################################################
6
7  con <- idaConnect("spatial_dsn",'USERID','PASSWORD')
8
9  ##################################################################
10 # Initialize the in-database functionality
11 ##################################################################
12
13 idaInit(con)
14
15 ##################################################################
16 # List of all tables in the current schema
17 ##################################################################
18
19 idaShowTables()
20
21 ##################################################################
22 # TRIPANDAFRE DATASET
23 ##################################################################
```

```
24
25 # TRIPANDFARE_JAN- Records for the month of January the TRIPANDFARE table
26 TripAndFare<-ida.data.frame("TRIPANDFARE_JAN")
27
28 # Consider only the trips payed by card.
29 TripAndFare<-TripAndFare[TripAndFare$PAYMENT_TYPE=="CRD",]
30
31
32 ## Remove the attributes which are usually unknown before the end of the trip
33 # 1. TRIP_TIME_IN_SECS
34 # 2. TOTAL_AMOUNT
35 # 3. FARE_AMOUNT
36
37 TripAndFare<-TripAndFare[, !(names(TripAndFare) %in% c('TRIP_TIME_IN_SECS',
38                                                        'TOTAL_AMOUNT',
39                                                        'FARE_AMOUNT'))]
40
41 ## Remove the useless columns like (acc. to exploration in rattle):
42 # 1. MEDALLION- identifier
43 # 2. HACK_LICENSE- Identifier
44 # 3. VENDOR_ID- Does not make sense to keep it for now
45 # 4. RATE_CODE
46 # 5. STORE_AND_FWD_FLAG
47 # 6. TOLLS_AMOUNT
48 # 7. MAT_TAX
49
50 TripAndFare<-TripAndFare[, !(names(TripAndFare) %in% c("MEDALLION",
51                                                        "HACK_LICENSE",
52                                                        "VENDOR_ID",
53                                                        "RATE_CODE"))]
54 TripAndFare<-TripAndFare[, !(names(TripAndFare) %in% c("STORE_AND_FWD_FLAG",
55                                                        "TOLLS_AMOUNT",
56                                                        "MAT_TAX"))]
57
58 ## Remove the absolute longitudes and latitudes
59 ## As we already have the pickup and dropoff point locations
60 # 1. PICKUP_LONGITUDE
```

```
# 2. PICKUP_LATITUDE
# 3. DROPOFF_LONGITUDE
# 4. DROPOFF_LATITUDE

TripAndFare<-TripAndFare[, !(names(TripAndFare) %in% c("PICKUP_LONGITUDE",
                                        "PICKUP_LATITUDE"))]
TripAndFare<-TripAndFare[, !(names(TripAndFare) %in% c("DROPOFF_LONGITUDE",
                                        "DROPOFF_LATITUDE"))]


# Remove the trip distance column
TripAndFare<-TripAndFare[, !(colnames(TripAndFare) %in% c("TRIP_DISTANCE"))]

##################################################
# Time factors (Derived Information)
##################################################

# Add a column for day of the week in the dataset. Since tip amount might depend upon the day.
TripAndFare$WEEKDAY<-weekdays(TripAndFare$PICKUP_DATETIME)

# Add a column for the date of trip in the dataset.
TripAndFare$DATE<-substr(TripAndFare$PICKUP_DATETIME,9,10)

# Add a column for the Hour of trip in the dataset.
TripAndFare$HOUR<-substr(TripAndFare$PICKUP_DATETIME,12,13)

# Drop pickup_datetime and dropoff_datetime column as the usefull info has been extracted.
TripAndFare<-TripAndFare[, !(colnames(TripAndFare) %in%
                        c("PICKUP_DATETIME",
                          "DROPOFF_DATETIME"))]


##################################################
# Categorize Tip Amount
##################################################

TripAndFare$TIP_RANGE<-ifelse(TripAndFare$TIP_AMOUNT<5,"<5D",">5D")
```

```
98
99  # remove the numerical TIP_AMOUNT column
100 TripAndFare<-TripAndFare[, !(colnames(TripAndFare) %in% c("TIP_AMOUNT"))]
101
102
103 ###################################################
104 # Geospatial info
105 ###################################################
106
107
108 #++++++++++++#
109 #+ AIRPORTS +#
110 #++++++++++++#
111
112 ###################################################
113 # DATA FRAME FOR AIRPORTAREA
114 ###################################################
115
116 airports_area<-ida.data.frame("AIRPORTAREA")
117
118 ###################################################
119 # INTERSECT AIRPORT AREA AND TAXIs (DROPOFF)
120 ###################################################
121
122 idaDeleteViewOrTable("GEOINFO_AIRPORT_D")
123 t1<-Sys.time()
124 GEOINFO_AIRPORT_D<-idaIntersects(TripAndFare,
125                  airports_area[,c("BUFFER_AIRPORT_BOUNDARY")],
126                  by.x=TripAndFare$D_LOCATION,
127                  by.y=airports_area$BUFFER_AIRPORT_BOUNDARY,
128                  tableName="GEOINFO_AIRPORT_D",
129                  returnDense=F)
130 Sys.time()-t1
131
132 GEOINFO_AIRPORT_D$DROPOFF<-ifelse(
133                    GEOINFO_AIRPORT_D$INTERSECTS_TRIPANDFARE_JAN_AIRPORTAREA==1,
134                        "Airport","NA")
```

```
135 # Remove inessential columns
136 GEOINFO_AIRPORT_D<-GEOINFO_AIRPORT_D[,!names(GEOINFO_AIRPORT_D) %in%
137                                 c("BUFFER_AIRPORT_BOUNDARY",
138                                   "INTERSECTS_TRIPANDFARE_JAN_AIRPORTAREA")]
139
140
141 ##################################################################
142 # INTERSECT AIRPORT AREA AND TAXIs (PICKUP)
143 ##################################################################
144 idaDeleteViewOrTable("GEOINFO_AIRPORT_P")
145 t1<-Sys.time()
146 GEOINFO_AIRPORT_P<-idaIntersects(GEOINFO_AIRPORT_D,
147                             airports_area[,c("BUFFER_AIRPORT_BOUNDARY")],
148                             by.x=GEOINFO_AIRPORT_D$P_LOCATION,
149                             by.y=airports_area$BUFFER_AIRPORT_BOUNDARY,
150                             tableName="GEOINFO_AIRPORT_P",
151                             returnDense=F)
152 Sys.time()-t1
153
154 GEOINFO_AIRPORT_P$PICKUP<-ifelse(
155                     GEOINFO_AIRPORT_P$INTERSECTS_GEOINFO_AIRPORT_D_AIRPORTAREA==1,
156                             "Airport","NA")
157 # Remove inessential columns
158 GEOINFO_AIRPORT_P<-GEOINFO_AIRPORT_P[,!names(GEOINFO_AIRPORT_P) %in%
159                                 c("BUFFER_AIRPORT_BOUNDARY",
160                                   "INTERSECTS_GEOINFO_AIRPORT_D_AIRPORTAREA")]
161
162
163 #++++++++++++
164 #+ THEATERS
165 #++++++++++++
166
167 ##################################################################
168 # DATA FRAME FOR THEATERAREA
169 ##################################################################
170
171 theaters_area<-ida.data.frame("THEATERAREA")
```

```
172
173 ##################################################################
174 # INTERSECT THEATER AREA AND TAXIs (DROPOFF)
175 ##################################################################
176
177 idaDeleteViewOrTable("GEOINFO_THEATER_D")
178
179 GEOINFO_THEATER_D<-idaIntersects(GEOINFO_AIRPORT_P,
180                                  theaters_area[,c("BUFFER_THEATER_LOCATION")],
181                                  by.y=theaters_area$BUFFER_THEATER_LOCATION,
182                                  by.x=GEOINFO_AIRPORT_P$D_LOCATION,
183                                  tableName="GEOINFO_THEATER_D",
184                                  returnDense=F)
185 GEOINFO_THEATER_D$DROPOFF<-ifelse(
186                     GEOINFO_THEATER_D$INTERSECTS_GEOINFO_AIRPORT_P_THEATERAREA==1
187                              "THEATER",
188                              GEOINFO_THEATER_D$DROPOFF)
189 # Remove inessential columns
190 GEOINFO_THEATER_D<-GEOINFO_THEATER_D[,!names(GEOINFO_THEATER_D) %in%
191                                    c("BUFFER_THEATER_LOCATION",
192                                      "INTERSECTS_GEOINFO_AIRPORT_P_THEATERAREA")]
193
194
195 ##################################################################
196 # INTERSECT THEATER AREA AND TAXIs (PICKUP)
197 ##################################################################
198
199 idaDeleteViewOrTable("GEOINFO_THEATER_P")
200
201 GEOINFO_THEATER_P<-idaIntersects(GEOINFO_THEATER_D,
202                                  theaters_area[,c("BUFFER_THEATER_LOCATION","THEATER_ID")],
203                                  by.y=theaters_area$BUFFER_THEATER_LOCATION,
204                                  by.x=GEOINFO_THEATER_D$P_LOCATION,
205                                  tableName="GEOINFO_THEATER_P",
206                                  returnDense=F)
207 GEOINFO_THEATER_P$PICKUP<-ifelse(
208                     GEOINFO_THEATER_P$INTERSECTS_GEOINFO_THEATER_D_THEATERAREA==1
```

```
209                          "THEATER",
210                          GEOINFO_THEATER_P$PICKUP)
211 # Remove inessential columns
212 GEOINFO_THEATER_P<-GEOINFO_THEATER_P[,!names(GEOINFO_THEATER_P) %in%
213                              c("BUFFER_THEATER_LOCATION","THEATER_ID",
214                                "INTERSECTS_GEOINFO_THEATER_D_THEATERAREA")]
215
216
217 #+++++++++++++++++
218 #+ MAIN LOCATIONS
219 #+++++++++++++++++
220
221 #1. Penn Station
222 #2. Grand Central
223 #3. Columbus Circle
224 #4. Lexington Avenue
225
226
227 ###################################################
228 # DATA FRAME FOR MAINAREAS
229 ###################################################
230
231 main_area<-ida.data.frame("MAINAREAS")
232
233 ###################################################
234 # INTERSECT THEATER AREA AND TAXIs (DROPOFF)
235 ###################################################
236
237 idaDeleteViewOrTable("GEOINFO_MAIN_D")
238
239 GEOINFO_MAIN_D<-idaIntersects(GEOINFO_THEATER_P,
240                          main_area[,c("BUFFER_LOCATION")],
241                          by.x=GEOINFO_THEATER_P$D_LOCATION,
242                          tableName="GEOINFO_MAIN_D",
243                          returnDense=F)
244 GEOINFO_MAIN_D$DROPOFF<-ifelse(
245                GEOINFO_MAIN_D$INTERSECTS_GEOINFO_THEATER_P_MAINAREAS==1,
```

```
246                         "MAINAREA",
247                         GEOINFO_MAIN_D$DROPOFF)
248 # Remove inessential columns
249 GEOINFO_MAIN_D<-GEOINFO_MAIN_D[,!names(GEOINFO_MAIN_D) %in%
250                           c("BUFFER_LOCATION",
251                             "INTERSECTS_GEOINFO_THEATER_P_MAINAREAS")]
252
253
254 ##################################################################
255 # INTERSECT THEATER AREA AND TAXIs (PICKUP)
256 ##################################################################
257
258 idaDeleteViewOrTable("GEOINFO_MAIN_P")
259
260 GEOINFO_MAIN_P<-idaIntersects(GEOINFO_MAIN_D,
261                         main_area[,c("BUFFER_LOCATION")],
262                         by.x=GEOINFO_MAIN_D$P_LOCATION,
263                         tableName="GEOINFO_MAIN_P",
264                         returnDense=F)
265 GEOINFO_MAIN_P$PICKUP<-ifelse(
266                 GEOINFO_MAIN_P$INTERSECTS_GEOINFO_MAIN_D_MAINAREAS==1,
267                         "MAINAREA",
268                         GEOINFO_MAIN_P$PICKUP)
269 # Remove inessential columns
270 GEOINFO_MAIN_P<-GEOINFO_MAIN_P[,!names(GEOINFO_MAIN_P) %in%
271                           c("BUFFER_LOCATION",
272                            "INTERSECTS_GEOINFO_MAIN_D_MAINAREAS")]
273 GEOINFO_MAIN_P<-GEOINFO_MAIN_P[,!names(GEOINFO_MAIN_P) %in%
274                           c("P_LOCATION",
275                             "D_LOCATION",
276                             "TRIP_ID",
277                             "PAYMENT_TYPE")]
278
279
280 ##################################################################
281 # Fetch the dataset with added spatial information to R
282 ##################################################################
```

```
283

284 R_Geo_Taxi <- as.data.frame(GEOINFO_MAIN_P)
```

Declaration of Authorship

I, Sumit Goyal, hereby declare that this thesis titled, 'Scalable Geospatial Analytics with IBM DB2 and R' is independently written by me. This work was done wholly or mainly while in candidature for a master degree at the Cologne University of Applied Sciences. I have indicated all passages whose exact wording or analogy have been derived from published or unpublished work of others. All sources and other means used for this thesis have been indicated by me. Neither the exact content nor major segments of the thesis have yet been submitted to an examination authority.

_____

Stuttgart, 05 June 2015