In-Database Geospatial Analytics using Python

Avipsa Roy*
Arizona State University
Tempe, AZ, USA
avipsa.roy@asu.edu

Edouard Fouché
Karlsruhe Institute of Technology
Karlsruhe, Germany
edouard.fouche@kit.edu

Rafael Rodriguez Morales
Technische Universität Dresden
Dresden, Germany

Gregor Möhler
IBM Deutschland Research and Development GmbH
Böblingen, Germany

ABSTRACT

The amount of spatial data acquired from crowdsourced platforms, mobile devices, sensors and cartographic agencies has grown exponentially over the past few years. Nearly half of the spatial data available currently are stored and processed through large relational databases. Due to a lack of generic open source tools, researchers and analysts often have difficulty in extracting and analyzing large amounts of spatial data from traditional databases. In order to overcome this challenge, the most effective way is to perform the analysis directly in the database, which enables quick retrieval and visualization of spatial data stored in relational databases. Also, working in-database reduces the network overhead, as users do not need to replicate the complete data into their local system. While a number of spatial analysis libraries are readily available, they do not work in-database, and typically require additional platform-specific software. Our goal is to bridge this gap by developing a new method through an open source software to perform fast and seamless spatial analysis without having to store the data in-memory. We propose a framework implemented in Python, which embeds geospatial analytics into a spatial database (i.e. IBM DB2 ®). The framework internally translates the spatial functions written by the user into SQL queries, which follow the standards of Open Geospatial Consortium (OGC) and can operate on single as well as multiple geometries. We then demonstrate how to combine the results of spatial operations with visualization methods such as choropleth maps within Jupyter notebooks. Finally, we elaborate upon the benefits of our approach via a real-world use case, in which we analyze crime hotspots in New York City using the in-database spatial functions.

CCS CONCEPTS

• **Information systems → Data analytics**; **Geographic information systems**.

KEYWORDS

In-database analytics, geospatial analytics, crime analysis, spatial data, Python, Maps

ACM Reference Format:
Avipsa Roy, Edouard Fouché, Rafael Rodriguez Morales, and Gregor Möhler. 2019. In-Database Geospatial Analytics using Python. In *2nd ACM SIGSPATIAL International Workshop on Advances on Resilient and Intelligent Cities (ARIC'19), November 5–8, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3356395.3365546

1 INTRODUCTION

1.1 Motivation

Modern day technologies like mobile devices, social media networks, fitness apps [2] and rideshare platforms [8] have enabled seamless collection of georeferenced location data about human movement. Storage, analysis and management of spatial data requires identifying spatial objects as geometric data types. Databases storing spatial data provide an additional functionality to process geometric data types efficiently, with the help of spatial functions for computing spatial measurements such as distance, buffer etc. Managing dense spatial-temporal information, from location-based applications, in terms of storage and retrieval for analytics, is often a major challenge faced by organizations, due to the data complexity [19]. Efficient mechanisms for retrieving information from spatial databases [9] to generate meaningful maps has become a necessity for analysts, urban planners and geographers in general.

The natural choice for developers using geographic data has been to store the data in the form of geometric data types in databases with a spatial extender such as PostGIS [28] or IBM DB2® Spatial as per the Open Geospatial Consortium SQL Specification Guidelines [7]. Spatial analysts exploit this data to build maps, graphs, statistical models, and cartograms, making complex spatial relationships understandable through interpretable visualizations. Such representations of data can reveal historical shifts, as well as detect current and future patterns in time. Geospatial analytics can help organizations anticipate and prepare for possible changes due to changing spatial conditions or location-based events [27]. It forms the basis of many decision making efforts including risk management, urban planning, crime detection – to name a few – and helps decision makers better understand the geographical aspects that influence broader trends and may have future consequences.

With in-database approaches, parts of the analytic logic are executed within a data warehouse, which allows faster data processing. It also helps to transform data and move it back and forth between

*This is the corresponding author.
975 S Myrtle Ave, School of Geographical Sciences and Urban Planning, Arizona State University, Tempe, AZ 85281, USA.

ARIC'19, November 5–8, 2019, Chicago, IL, USA

the database and remote analytics applications. From one system to another, the availability of in-database analytic capabilities may vary. Typically, in-database analytic is enabled through a set of libraries and user-defined functions, such that they can:

- Access the data in the database, data warehouse or appliance in-situ, without needing to extract it to some interim format.
- Use the hardware, parallel processing capabilities and load balancing/processor management of the data infrastructure.
- Be accessed both from specialist analytic tools, e.g., for model creation or data quality tasks, and from operational systems.

Previous studies [15, 18, 26] have focused on in-database analytics for numeric data. To the best of our knowledge, spatial data, which are structurally complex and multivariate in nature, have not been accounted for so far. The open source crowdsourced mapping platform, OpenStreetMap (OSM) [5], with more than 4.3 million users, accelerates the generation of massive spatial information from community users and currently stores more than 6.3 billion geographic coordinates in its database. Similarly, social media data sources such as Twitter, generates an average of over 500 million geolocated tweets daily, and the volume of tweets is growing at a rate of nearly 30% each year. Making sense of such huge volumes of raw data is a critical challenge. Owing to its complexity, volume and multivariate nature, spatial data are often computation-intensive and require streamlined tools for better analysis. The integration of spatial data sets relies on spatial queries of different types, which we will address in this paper. These queries can be broken down into the following categories:

- **Contain/Within queries:** For a given geolocation, a user might want to find the containing polygon or multi-polygon. The cardinality may vary based on the number of use cases. For instance, while exploring Twitter data, we may want to determine the business unit in which a tweet author was located using OpenStreetMap, where both the number of points and the number of business unit objects are huge. As another example, a user might want to identify only tweet authors who were in airports, where the amount of accurate geometry shapes could be limited.
- **Buffer queries:** In reality,the accuracy of GPS data depends on several factors, such as the quality of the GPS receiver, the actual position of satellites, the surroundings. Thus, several meters of horizontal inaccuracy is very common. In such cases, computing the buffer of a geographic location as a polygon would make more sense, and using a containment query between polygon helps in getting around the problem.
- **Length/Area queries:** These queries help to calculate area, perimeter or length of polygon objects representing entities like counties, states, census block groups, lakes or streets. They are useful to analysts in order to infer the density of a given geographic entities.
- **Distance queries:** Often, researchers are interested in proximity calculations when they consider a geometry with respect to its context. For example, they might be interested in knowing how far is the nearest bus stop from a house? or which is the closest park in the neighborhood? The answer to these questions are mostly distance calculations - both straight line as well as geodesic.
- **Network Topology queries:** Some spatial databases (Eg: pgRouting in PostGIS) give users the ability to run pathfinding queries on tables of links. The topology support allows to create edges and faces from existing, non-network aware polygons and polylines. Creating topologies, computing shortest distance and calculating costs are some of the typical operations often used as queries in a spatial network.

Also, the large volume of the dataset in a single instance of a spatial table can be a hindrance, in term of loading the entire dataset in-memory and thereby reduce operational efficiency. Last but not the least, data manipulation and mining often is an interactive process [11], where short response times are preferred across multiple platforms and tools in an integrative framework. As a result, one often resorts to the extraction of small samples, or transfers the data to a cluster system for further processing [18]. However, samples may be unrepresentative of the real data distribution - when it comes to distance calculations or performing spatial joins. Distributed computing and use of high performance machine in turn gives way to high infrastructure expenses. Therefore, the need for the adoption of in-database capabilities have eventually grown.

1.2 Current Challenges

Depending upon the size of the study area (e.g., neighborhood, city, states, countries, etc.), spatial queries typically involve huge amounts of complex spatial objects (points, polygons and multi-polygons); they are are both highly data- and computation-intensive.

Spatial analytics problems usually involve combining spatial data with relational data from external sources to establish spatial relationships and hypothesize spatial patterns, e.g., determining the location of possible markets. One of the main challenges is that spatial analytics data typically is heterogeneous. Also, there are currently not many open-source and easy-to-use tools available for in-database analytics using spatial data. In-database geospatial analytics combines spatial data with other relational data from disparate sources to establish spatial relationships and hypothesize spatial patterns. It can help users with activities such as defining the areas in which you provide services, and determining locations of possible markets. The challenge here is that the data sources one is actually interested in are heterogeneous.

Traditionally, spatial analysis relies on proprietary GIS (Geographic Information System) tools [1], and there is a lack of tools dedicated to spatial analysis developed on standard open source platform, such as Python [6]. Also, existing GIS tools often are unable cope with the large volumes and complexity of the datasets involved in real-life spatial decision-making problems [10] which requires handling large datasets. To summarize, the complexity and heterogeneity of spatial analytics, as well as the lack of non-proprietary tools, motivates the development of open source spatial in-database frameworks.

1.3 Contributions & Paper Outline

We use a functional approach to solve those challenges using a Python based software package for performing fast and seamless spatial analysis without having to store the data in-memory. Our contribution is two-fold:

- We propose an extension of the *ibmdbpy* [18] framework called *ibmdbpy-spatial*. Our extended framework represents spatial data as geometries as a special attribute in a dataframe and enables spatial analysis via associated wrapper functions, which work by seamlessly pushing spatial queries as SQL operations into the database.
- We demonstrate the applicability and value of our framework via a case study, in which we analyze crime hotspots in New York City.

The remainder of the paper is organized as follows: Section 2 is the related work. Section 3 explains the principles of the ibmdbpy-spatial framework. Section 4 highlights the results of our case study. Section 6 is our conclusion.

2 RELATED WORK

A few interfaces for in-database analytics exists, e.g., the Blaze ecosystem [20]. Blaze provides an interface for multiple backends, (eg: SQL databases, NoSQL data stores, Spark, Hive, Impala, and raw data files), which simultaneously is a drawback, since this reduces the available functions to the common subset. Previous studies have looked at performing in-database learning from sparse tensors [26] but do not support spatial analysis. Simba [35] introduced a framework to perform geospatial analytics using a distributed database approach through Spark, with additional dependency on Java and Scala, i.e., users need prior knowledge of multiple languages to seamlessly integrate logical programming codes along with spatial queries in a single platform. PySAL [29] provides an analytics platform for geospatial data, but does not leverage database technology. The ibmdbpy framework [18] was proposed as a solution for in-database analytics for numeric data within IBM DB2 ® and paved the way for the spatial analysis framework that we present in the following sections of this paper.

3 OVERVIEW OF IBMDBPY-SPATIAL

In-database analytics operations usually encapsulate complex SQL statements into functions of a data analysis framework like Pandas [25], using Python. These SQL statements are eventually translated to database queries at runtime, as so-called "SQL-pushdowns', whose results are retrieved as a data structure into local memory, typically in the form of a so-called "dataframe". A dataframe represents data in standard vector format, i.e., it is easy to manipulate and foster further exploratory analysis.

SQL-pushdowns reduce execution time for reading data and running complex queries on the data, compared to fetching the entire dataset into memory, which might lead to much network overhead. To demonstrate our approach, we use IBM DB2, which is available via a free entry plan on the IBM Bluemix® platform. Ibmdbpy-spatial wraps around the IBM DB2 database spatial extender, which supports multiple types of spatial queries through customizable spatial query engine, multi-level indexing, implicit parallel spatial query execution, and effective methods for amending query results through handling boundary objects.

3.1 In-database Analytics

The preliminary step of most data analysis applications is to first extract the data stored in a relational database. The process of data extraction is often a challenge for analysts and end users for several reasons: The complexity of data can be very high (e.g., spatial data), and the data representation is of varied types (e.g., point, polygon, numeric, string etc.).

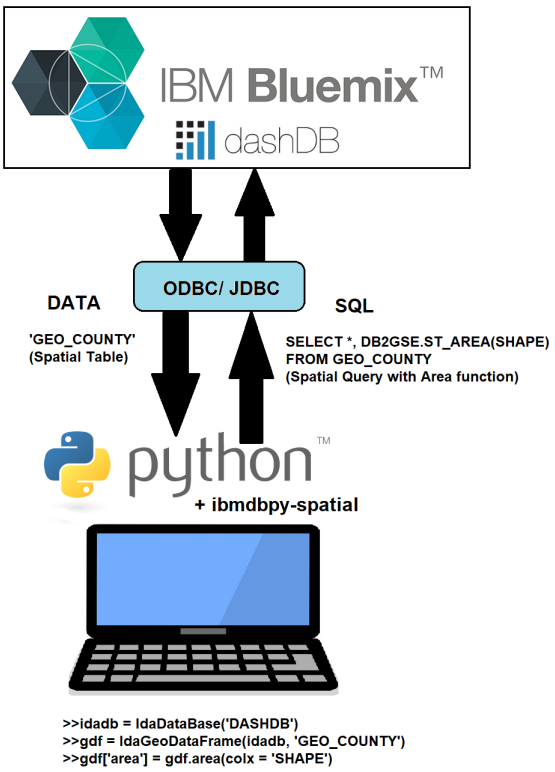

Figure 1: Framework for ibmdbpy-spatial

In-database analytics means that analytic capabilities are embedded directly in a relational/columnar database or a data warehouse software. These capabilities are specific to a particular database, data warehouse or a data appliance of a specific kind. In-database analytics operations usually translates complex SQL statements into a single function. For end users who are not efficient in query processing, this analytical approach seems like a reasonable alternative. The database queries are translated as SQL-pushdowns at runtime and the result is retrieved as a memory instance in the form of dataframes, which are easy to manipulate for further exploratory analysis and visualization.

Such an approach reduces overall data retrieval time thereby reducing the network overhead involved in running complex spatial queries on the entire dataset in-memory, often resulting in the working platform to crash midway.

3.2 In-database Analytics with Spatial Data

The idea of in-database geospatial analytics is to translate complex spatial queries into easy-to-use functions, represented in a standard programming language (E.g.: Python, R etc). In our study we develop a software package "ibmdbpy-spatial" using Python to implement this feature. We choose Python as our choice of programming language owing to its large user base and open source

nature and purpose of reproducibility. The package we develop internally uses database wrapper functions to translate spatial queries into well known Pandas [25] like syntax. Ibmdbpy-spatial, wraps standard OGC specific spatial queries and generates their Python equivalent. It uses a middleware API (pypyodbc/JayDeBeApi) to send the queries to an ODBC or JDBC-connected database for execution as shown in the workflow in Figure 1 [1]. The results are fetched and formatted into the well-known dataframe format in an open source framework in Python . Typically, spatial data used for building the ibmdbpy-spatial library can be categorized into 3 main spatial objects: points , lines and polygons. We show the topological framework that underlies spatial operations for ibmdbpy-spatial in Figure 2. The ibmdbpy-spatial framework allows remote spatial operations within databases by wrapping database-specific spatial operations as user-friendly Python functions with a simple syntax. Additionally, it benefits from the performance-enhancing features of in-database processing, such as columnar storage and parallel query processing.

For example, let us assume there is a table containing trajectory of a storm and we want to find the length of the trajectory of the storm. In this case, the user is using a data analysis tool like Python to perform such operations, and would do the following:

```
# Connect to the database
>>> idadb = IdaDataBase('DASHDB')
# Load the data as a dataframe
>>> df = IdaGeoDataFrame(idadb,
        'SAMPLES.GEO_TORNADO',geometry = 'SHAPE')
# Get the length of trajectory via geospatial method
>>> df.length(unit = 'KILOMETER')
```

The above operations are translated into a SQL query as follows:

```
SELECT DB2GSE.ST_Length(SHAPE)
FROM SAMPLES.GEO_TORNADO;
```

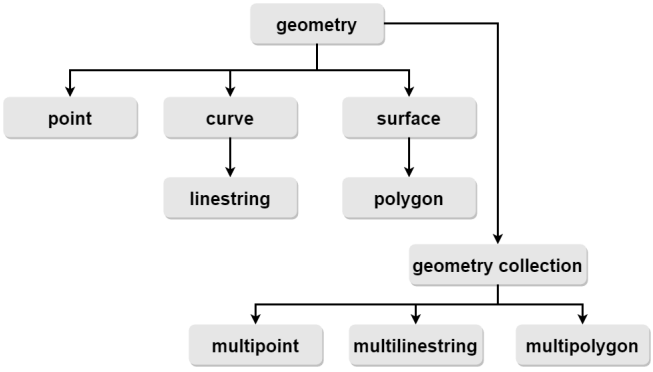

Figure 2: Topological framework of spatial data used in ibmdbpy-spatial

[1]Disclaimer: IBM, the IBM logo, ibm.com, DB2, dashDB and Bluemix are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml. Other company, product, or service names may be trademarks or service marks of others.

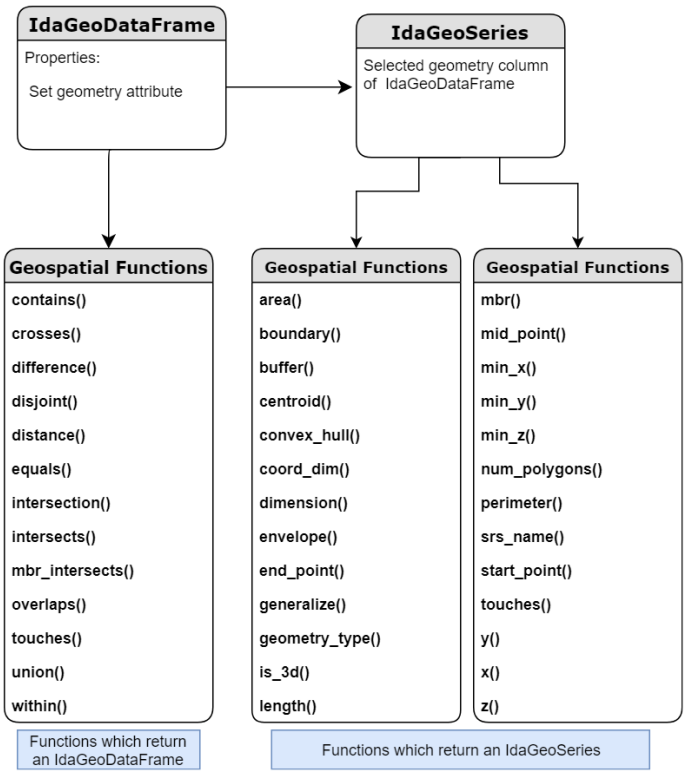

Figure 3: Spatial functions design workflow

The first step is to set up an ODBC/JDBC connection with the DB2 database. The spatial data is then identified by ibmdbpy-spatial as a special class called IdaGeoDataFrame that extends all the properties of a data frame with additional methods for geospatial data types like ST_Point, ST_LineString, ST_Polygon etc.

An IdaGeoDataFrame is a reference to a spatial table in a remote instance of the connected database. The IdaGeoDataFrame object references spatial data by means of the "geometry" attribute, which is a special column in the table with spatial data. When a spatial method is applied to an IdaGeoDataFrame (or a spatial attribute like area is called), these commands will always act on the "geometry" attribute. The geometries are represented as well-known-text (WKT) in Python.

Topological operations on geometric features is the most important functionality required to analyse customer data and derive meaningful relationships from the raw data. Certain spatial functions return information about ways in which geographic features relate to one another or compare with one another. Other spatial functions return information as to whether two definitions of coordinate systems or two spatial reference systems are the same. In all cases, the information returned is a result of a comparison between geometries, between definitions of coordinate systems, or between spatial reference systems. Some of the common topological operations, available as stored functions inside ibmdbpy-spatial are - AREA(), WITHIN(), DISTANCE(), BUFFER(), INTERSECT() to name a few. More functions shown in Figure 3 are available and can be found in the ibmdbpy-spatial documentation.

Each method is a Python wrapper which internally triggers a stored procedure from the spatial database which then translates into a dataframe in Python. The geospatial methods can operate on a single geometry or multiple geometries and each of the two behaviour is configured using a Handler method in Python which interprets the type of method being executed from the type of arguments passed.

Let us take an example to understand how the spatial functions work. We have two IdaGeoDataFrames for Customer and Counties of North America. We would like to find those customers with a high insurance value above $250000 and residing in the county of Austin. First, we begin by reading all the data for US counties and assign a geometry attribute to the IdaGeoDataFrame.

```
# Read a database table as an IdaGeoDataFrame
>>> idageodf1 = IdaGeoDataFrame(idadb,
                'SAMPLES.GEO_COUNTY',indexer='OBJECTID')
# Select the geometry attribute
>>> idageodf1.set_geometry('SHAPE')
```

Now, we select the counties which are in Austin, TX.

```
# Choose the counties in Austin
>>> idageodf1 = idageodf[idageodf['NAME']=='Austin']]
```

Following this step, we then choose all customers with an insurance value above $250,000 from the customers IdaGeoDataFrame.

```
# Select all customers data
>>> idageodf2 = IdaGeoDataFrame(idadb,
                 'SAMPLES.GEO_CUSTOMER',
                 indexer='OBJECTID')
# Set the geometry attribute
>>> idageodf2.set_geometry('SHAPE')
# Select customers with insurance value > $250,000
>>> idageodf2 = idageodf[idageodf
                 ['INSURANCE_VALUE']>250000]
```

Now that we have all the information about the customer an counties, we will try to find all those customer who reside in Austin and have an insurance value above $250,000. For the spatial query we need to use the WITHIN() function as shown in Algorithm 1 from ibmdbpy-spatial which will filter out all geometries corresponding to the customer locations that lie within polygons representing the counties in Austin, TX.

Internally, ibmdbpy uses objects with spatial methods as Geopandas objects, such as the well-known GeoDataFrame, but in fact, the data lies in a distant database. Invoking a method actually leads to the construction of a string that should be a valid spatial database query. Apart from the connectivity layer, ibmdbpy-spatial works independently from the underlying database system, since it generates standard OGC specific spatial queries.

The scripts can be used in an interactive framework with Jupyter notebooks [23], a web application as shown in Figure 5 for creating and sharing documents, containing live code, visualizations and explanatory text, which makes the spatial analysis interactive and independent of additional software installations. Once we have the result GeoDataFrame, we can just filter out a single customer using the matched index to find his/her location and insurance value associated with him/her.

Algorithm 1 Algorithm for within query in ibmdbpy-spatial

Require: Table *tab1* with polygon and *tab2* with points
Ensure: Query to find points within each polygon in *tab*

```
function WITHIN(tab)
    geom1 ← get_geometry(tab1)
    geom2 ← get_geometry(tab2)
    tabname1 ← get_name(tab1)
    tabname2 ← get_name(tab2)
    for p1,p2 ← geometries(geom1,geom2) do
        string ← string + p2.ST_WITHIN(p1)
    within ← string join with ","
    select ← "SELECT"
    from ← CONCATENATE(" FROM ",tab
    return CONCATENATE(select,within,from)
```

The results can be further used to visualize data and combine with other statistical operations available through the Pandas [25] data analysis library in Python. The Jupyter notebooks of our case study and spatial functions implementation can be found on GitHub[2].

```
#Select the customer locations within
#each county in Austin
>>> result = idageodf2.within(idageodf1)

#The indexes of both dataframes are shown
#RESULT = 1 indicates whether or not
#the customer resides in Austin
>>> result[result['RESULT']==1].head()
INDEXERIDA1     INDEXERIDA2        RESULT
    69879            2              1
    69934            2              1
    69965            2              1
    256660           2              1
    256682           2              1

>>> idageodf2[idageodf2['ID']==69879].head()
    ID      NAME               INSURANCE_VALUE
    69879   Angie Baumgardner  263388
```

4 CASE STUDY USING IBMDBPY-SPATIAL: MAPPING CRIME DENSITY IN NEW YORK CITY

4.1 Using in-database spatial function to calculate crime density

The New York city police department has gathered a huge amount of data over a period of 10 years and more and categorized the 7 major felonies committed in the city of New York. We can analyze

[2]https://github.com/ibmdbanalytics/ibmdbpy

this huge dataset [13] with efficient geospatial analytics tools using ibmdbpy-spatial to gain meaningful insights from the data.

The data contained 2.5 million records which were first loaded into the spatial database. The ibmdbpy-spatial package was then used to extract crime data for each borough. We used the within() function from the ibmdbpy-spatial package to calculate crime locations within each borough. The WITHIN() function on the entire database took less than 0.25s to run the query for the entire dataset, which is extremely low compared to the time taken to run in-memory spatial queries from standard geospatial libraries like Geopandas [14], which runs for nearly 2 s on a dual core machine with 64 GB RAM and 2.6Ghz processor. For the purpose of crime analysis in New York city we followed the algorithm shown in Algorithm 2.

Algorithm 2 Algorithm for computing area in ibmdbpy-spatial

Require: Table *tab* with spatial data
Ensure: Query for area of each polygon in *tab*

```
function AREA(tab)
    geom ← get_geometry(tab)
    tabname ← get_name(tab)
    for p1 ← geometries(geom) do
        string ← string + ST_AREA(p1)
    area ← string join with ","
    select ← "SELECT"
    from ← CONCATENATE(" FROM ",tab
    return CONCATENATE(select,area,from)
```

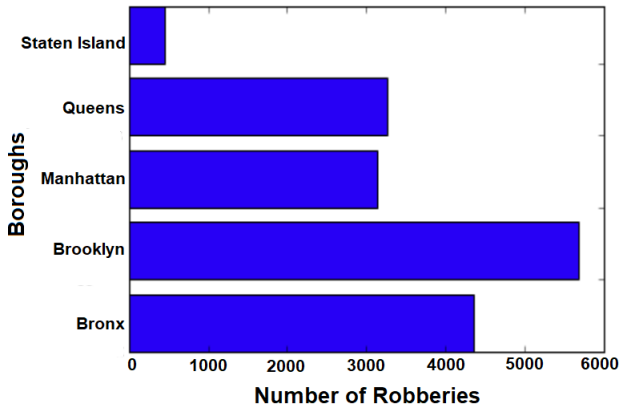

Figure 4: Overall distribution of robberies in New York city

We filtered the crimes by their type to extract the density of robberies occurring in each borough. Figure 4 show the number of crimes between 2010-2017 in each borough. We first read the spatial data as IdaGeoDataFrames and set their geometry attributes.

```
>>> nyc_gdf = IdaGeoDataFrame(idadb,'NYC_BOROUGHS',
                  indexer='OBJECTID')
>>> nyc_gdf.set_geometry('SHAPE')
>>> nyc_crime_gdf = IdaGeoDataFrame(idadb,
'NYC_CRIMES',indexer='OBJECTID')
>>> nyc_crime_gdf.set_geometry('SHAPE')
```

```
import getpass,jaydebeapi,jpype
uid = raw_input('Enter Username:')
pwd = getpass.getpass('Enter password:')
jdbc = jdbc_link + ':user=' + uid + ';password='
+ uid + ';password=' + pwd
idadb = IdaDataBase(dsn = jdbc)
print('Connection to dashDB successful!')
```

```
Enter Username:dash5548
Enter password:·····
Connection to dashDB successful!
```

```
# Read the data from dahsDB using ibmdbpy
import ibmdbpy
from ibmdbpy import IdaDataBase,IdaDataFrame,IdaGeoDataFrame,IdaGeoSeries
boros = IdaGeoDataFrame(idadb,'NYC_BOROUGHS',indexer = 'OBJECTID')
felonies = IdaGeoDataFrame(idadb,'NYC_CRIME_DATA',indexer = 'OBJECTID')
#Set the geometry attribute and calculate area of the boroughs
boros.set_geometry('GEO_DATA')
felonies.set_geometry('GEO_DATA')
boros['area_in_sq_km'] = boros.area(unit = 'KILOMETER')
```

```
boros.head()
```

	OBJECTID	BoroName	AREA_IN_SQ_KM
0	1	Staten Island	150.856763
1	2	Queens	282.911619
2	3	Brooklyn	179.997796
3	4	Manhattan	59.130826
4	5	Bronx	110.270598

Figure 5: Jupyter notebook executing ibmdbpy-spatial functions in Python

4.2 Results: Mapping crime density in New York city

Once the geometries were assigned we used a density calculation by counting the total number of robberies within each borough and dividing it by the area of the respective boroughs. We used the WITHIN() and AREA() functions to calculate the crime density per borough.

```
>>> manhattan = nyc_gdf[nyc_gdf["NAME"]=='Manhattan']
>>> area = manhattan.area()
>>> manhattan_crimes = nyc_crime_gdf.within(manhattan)
>>> counts = manhattan_crimes.query('RESULT==1')
>>> density = counts/area
```

We then visualized the results on a map using additional Python libraries matplotlib and folium using the outcomes from Figure 5 and using the following code.

We use a choropleth map to plot crime densities by each borough in New York city. We show the relative density or amount of crime occurring in different areas in Figure 6. The thematic shading by each borough from light to dark green indicates the crime density index shown at the top of the map as a continuous scale ranging from 0 to 0.5. Since the spatial aggregations of crimes were quite large it is not clear enough to distinguish much spatial dependency.

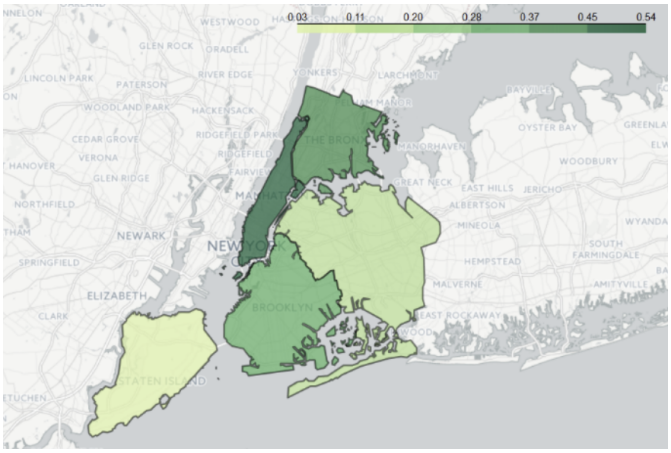

Figure 6: Map of crime density in New York city

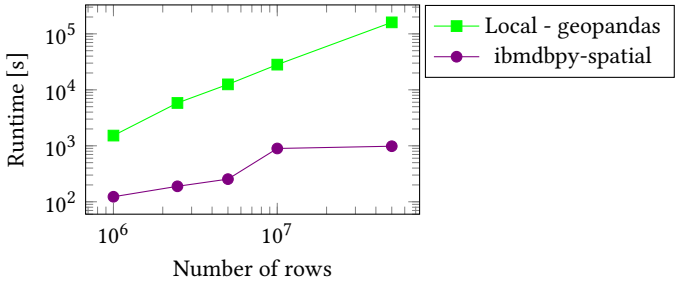

Figure 7: Crime density estimation with equal resources

But clearly the northern most boroughs are much more crime prone compared to the southern part of the city.

We also compared the performance of spatial queries by increasing the volume of data as shown in Figurre 7, randomly sampled from the original dataset. We assessed the scalability using a Jupyter notebook running on a 64-bits operating Windows 10 operating system. The machine contains 16 GB RAM and a Intel Xeon processor at 2.60 GHz. We use Python version 3.7. By the time of the experiments, the version 0.1.0 of *ibmdbpy* is installed. We connect to a distant database using JDBC. The distant database is an IBM dashDB enterprise instance, hosted on IBM Bluemix® cloud platform.

4.3 Discussion

The importance of crime data analysis has played an important role in public safety and planning smart cities [22]. Areas of concentrated crime activity are often referred to as hot spots. Hotspots have been defined by previous researchers in the literature in terms of hot spot addresses [16, 32], as well as hot spot blocks [33, 34], some also examine clusters of blocks [12]. Crime analysts often look at hotspots to identify concentrations of individual events that might indicate a series of related crimes and try to link these to underlying social conditions. The results shown in Figure 6 clearly indicate the spatial distribution of robberies in New York city. We found that the highest crime density is in Brooklyn and the lowest crime density is in Staten Island. Although the size of the boroughs have a direct impact on the density of crimes, we can also see in Figure 4 that the overall frequency of crimes in these boroughs match up. The entire analysis was performed in Jupyter notebook without any prior database installation. We found that it is efficient to process large datasets with more than 2.5 million rows using ibmdbpy-spatial and the results can be visualized on the fly using simple Python scripts. This relieves the user from additional hurdles of multiple software installations for spatial analysis or the need for additional GIS software tools that are commonly used for visualizing maps. We also found that the challenges in using in-memory tools where the network overload might slow down data analysis in case of large spatial operations, ibmdbpy-spatial was able to overcome those. Our study introduces a new method for the exploratory analysis of spatio-temporal data in an efficient and fast manner with the help of the Python package ibmdbpy [17]. The primary and most interesting outcome of our study was applying in-database analytics approach to geolocated crime data stored in a traditional enterprise data warehouse, IBM DB2, in this case.

5 CONCLUSION

Crime analytics [24] is a major part of a building a smart cities. With growing availability of IoT (Internet-of-Things) devices [21], it is possible to gather real-time data about crimes in and around a specific neighborhood, which help in developing a smart city approach. The data processing complexity, which is usually a hindrance in such analyses, can be easily handled by the in-database geospatial analytics approach in ibmdbpy-spatial [30]. The results can be further combined with other forms of data sources from social media platforms to extract more information about crime locations, improve vulnerable areas and thereby build a safer community [31].

Users can also visualise the results in a more meaningful fashion with additional Python libraries like matplotlib[4] and folium[3]. We develop a scalable andeasy-to-use framework for in -database analytics using spatial data in this study. Policymakers and local authorities can use this framework to identify areas of high or low crime rates and further investigate socio-economic and demographic characteristics of those neighborhoods which might contribute to a visible high crime density location. "ibmdbpy-spatial", is a first step towards promoting a more structured investigation on the relationship between crime rates and overall quality of life via crowdsourced data and thereby help create safer communities.

6 FUTURE WORK

In the next phase of the work, we would introduce additional capabilities within ibmdbpy-spatial to identify k-nearest neighbors and k-means clustering to understand the spatial effects of crime distribution in the study area. The detailed analysis could also help explore if there are any social underpinnings as to why the crime densities vary over space.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Edzer Pebesma at the Institute for Geoinformatics, University of Münster for his constructive feedback and Dr. Trisalyn Nelson for providing financial support to present the work.

REFERENCES

[1] 2019. ArcGIS Desktop: Release 10. http://www.arcgis.com/
[2] 2019. Creating better cities by using big data. https://metro.strava.com/
[3] 2019. Folium-Library. https://python-visualization.github.io/folium/
[4] 2019. Matplotlib Library - Python. https://matplotlib.org/
[5] 2019. OpenStreetMap. https://www.openstreetmap.org/
[6] 2019. Python 3.6. https://www.python.org/
[7] 2019. Simple Feature Access - Part 2: SQL Option. https://www.opengeospatial.org/standards/sfs
[8] 2019. Uber Movement: Lets find smarter ways forward, together. https://movement.uber.com/
[9] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel Saltz. 2013. Hadoop gis: a high performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment* 6, 11 (2013), 1009–1020.
[10] Gennady Andrienko, Natalia Andrienko, Piotr Jankowski, Daniel Keim, M-J Kraak, Alan MacEachren, and Stefan Wrobel. 2007. Geovisual analytics for spatial decision support: Setting the research agenda. *International journal of geographical information science* 21, 8 (2007), 839–857.
[11] Luc Anselin. 1999. Interactive techniques and exploratory spatial data analysis. *Geographical Information Systems: principles, techniques, management and applications* 1, 1 (1999), 251–264.
[12] Richard L Block and Carolyn Rebecca Block. 1995. Space, place and crime: Hot spot areas and hot places of liquor-related crime. *Crime and place* 4, 2 (1995), 145–184.
[13] Open Calgary. [n. d.]. nypd 7 major felony incidents. https://data.cityofnewyork.us/Social-Services/nypd-7-major-felony-incidents/k384-xu3q
[14] GeoPandas Developers. 2019. GeoPandas 0.4. 0.
[15] Joseph Vinish D'silva, Florestan De Moor, and Bettina Kemme. 2018. AIDA: abstraction for advanced in-database analytics. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1400–1413.
[16] John Eck and David L Weisburd. 2015. Crime places in crime theory. *Crime and place: Crime prevention studies* 4 (2015).
[17] Edouard Fouche. 2017. ibmdbpy. https://pypi.org/project/ibmdbpy/
[18] Edouard Fouché, Alexander Eckert, and Klemens Böhm. 2018. In-database analytics with ibmdbpy. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*. ACM, 31.
[19] Michael Goodchild, Robert Haining, and Stephen Wise. 1992. Integrating GIS and spatial data analysis: problems and possibilities. *International journal of geographical information systems* 6, 5 (1992), 407–423.
[20] Continuum Analytics Inc. 2018. The Blaze Ecosystem. http://blaze.pydata.org
[21] Roozbeh Jalali, Khalil El-Khatib, and Carolyn McGregor. 2015. Smart city architecture for community level services through the internet of things. In *2015 18th International Conference on Intelligence in Next Generation Networks*. IEEE, 108–113.
[22] Zaheer Khan, Ashiq Anjum, Kamran Soomro, and Muhammad Atif Tahir. 2015. Towards cloud based big data analytics for smart future cities. *Journal of Cloud Computing* 4, 1 (2015), 2.
[23] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. Jupyter Notebooks-a publishing format for reproducible computational workflows.. In *ELPUB*. 87–90.
[24] Olivera Kotevska, A Gilad Kusne, Daniel V Samarov, Ahmed Lbath, and Abdella Battou. 2017. Dynamic network model for smart city data-loss resilience case study: City-to-city network for crime analytics. *IEEE Access* 5 (2017), 20524–20535.
[25] Wes McKinney. 2015. pandas: a Python data analysis library. *see http://pandas. pydata. org* (2015).
[26] Hung Q Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. 2017. In-database factorized learning. CEUR Workshop Proceedings.
[27] Hasan Poonawala, Vinay Kolar, Sebastien Blandin, Laura Wynter, and Sambit Sahu. 2016. Singapore in motion: Insights on public transport service level through farecard and mobile data analytics. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and data mining*. ACM, 589–598.
[28] Paul Ramsey et al. 2005. Postgis manual. *Refractions Research Inc* 17 (2005).
[29] Sergio J Rey, Luc Anselin, Xun Li, Robert Pahle, Jason Laura, Wenwen Li, and Julia Koschinsky. 2015. Open geospatial analytics with PySAL. *ISPRS International Journal of Geo-Information* 4, 2 (2015), 815–836.
[30] Avipsa Roy and Rafael Rodriguez. [n. d.]. ibmdbpy-spatial. https://pythonhosted.org/ibmdbpy/geospatial.html
[31] Sumit Shah, Fenye Bao, Chang-Tien Lu, and Ing-Ray Chen. 2011. Crowdsafe: crowd sourcing of crime incidents and safe routing on mobile devices. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 521–524.
[32] Lawrence W Sherman, Patrick R Gartin, and Michael E Buerger. 1989. Hot spots of predatory crime: Routine activities and the criminology of place. *Criminology* 27, 1 (1989), 27–56.
[33] Ralph B Taylor, Stephen D Gottfredson, and Sidney Brower. 1984. Block crime and fear: Defensible space, local social ties, and territorial functioning. *Journal of Research in crime and delinquency* 21, 4 (1984), 303–331.
[34] David Weisburd and Lorraine Green. 1995. Policing drug hot spots: The Jersey City drug market analysis experiment. *Justice Quarterly* 12, 4 (1995), 711–735.
[35] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. 2016. Simba: Efficient in-memory spatial analytics. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 1071–1085.