Random generation of realistic spatial data for use in classroom assessments

Patrick Gorry

14371466

Supervisor Name

Peter Mooney

Submitted in Partial Fulfilment of the Requirements of the Master of Science in Data Science and Analytics at Maynooth University

Department of Computer Science

Maynooth University

August 2022

Declaration and Approval

I declare that this work has not been previously submitted and approved for the award of a degree by this or any other University. To the best of my knowledge and belief, the research proposal contains no material previously published or written by another person except where due reference is made in the research proposal itself.

Student Name: Patrick Gorry

Student Number: 14371466

Student Signature: _____________________ Date: _____________________

The proposal of Patrick Gorry has been reviewed and approved by Peter Mooney.

Supervisor Signature: _____________________ Date: _____________________

Acknowledgement

I’d like to extend the utmost thanks to my supervisor Peter Mooney who has been incredibly helpful both during this project and during the Masters as a whole. I’d like to thank Jenny for all of her love and support throughout writing the year as well as my friends and family for supporting me and cheering me on.

Note to readers - additional support materials

This thesis "Random generation of realistic spatial data for use in classroom assessments" has a supporting GitHub repository which contains the source code of the RADIAN (**RA**n**D**om spat**I**al d**A**ta ge**N**erator) application. The repository also provides access to an extensive set of test scenario outputs which are too numerous to include in the thesis. Finally, a descriptive screencast video is provided to deliver a demonstration of the RADIAN software working.
The link to the repository is **https://github.com/paddeaux/msc_rng**.

Abstract

We describe the design, development, implementation and usage of a software-based tool called RADIAN (**RA**n**D**om spat**I**al d**A**ta ge**N**erator) to generate realistic spatial datasets for use within teaching and learning environments who require spatial datasets. RADIAN provides configurable functionality for users to generate spatial datasets containing geometric points with associated parameters/attributes which can then be easily imported into a PostgreSQL PostGIS database or visualised using a GIS or equivalent software. Much work has been carried out in areas such as statistics, machine learning and software testing on how to generate realistic datasets for testing hypotheses, machine learning model training and validation, algorithmic analysis, and rigorous software testing on real-world data. However, less work has been published on the generation of realistic spatial data. This thesis contributes to this body of work. We outline the theoretical approach we have used in RADIAN to generate random geometric points within a given spatial extent (polygon). We demonstrate the effectiveness of RADIAN with a suite of example scenarios of the spatial data generated. We believe RADIAN will be very useful to both teachers and spatial analysts requiring realistic randomly generated for spatial analysis purposes. The software code is available as open-source software via GitHub. The thesis concludes with some suggestions for further research and development work which are possible from the research and development of RADIAN.

Table of Contents

List of Figures

Chapter 1

Introduction

In this chapter we set the scene for the description of the main problem addressed by the thesis. We briefly describe the need for a software tool which can generate realistic spatial data efficiently and easily. Some currently available tools to perform this task are discussed briefly. In section 1.3 we introduce our solution to the overall problem with a brief discussion of the software implementation.

1.1 Problem Statement

The world of spatial data analysis and Geographic Information Systems (GIS) provide numerous interesting topics and powerful tools for many research and industrial applications within a broad range of scientific fields. When teaching the core concepts in GIS and spatial databases in the classroom having appropriate access to usable data for learning (in the case of students) and instruction (in the case of lecturers or demonstrators) is very important. While there is a great deal of spatial data available as Open Data some students can find it difficult to acquire suitable datasets to learn and practice with. For lecturers, teachers, and instructors it can also be difficult to find appropriate datasets for use as examples or as part of student assessments. Providing students with an interesting set of datasets during teaching and learning activities in spatial databases and GIS can be challenging but can improve student engagement (Burns and Chopra, 2016). It is possible to generate or engineer sample datasets from open or public datasets. However, this, often manual process, can be very time consuming particularly if a spatial dataset with thousands of geographic features is required. Being able to generate these datasets quickly is an attractive requirement particularly in fast-moving and dynamic classroom environments.

Data generation is an important tool for a variety of users from teachers who teach statistical methods or data science, to software designers and developers who want to demonstrate and test their software. Some studies have shown that students engage better with teachers and instructors who show originality and ingenuity in developing their own teaching materials (Anderson et al., 2021; Rao and Dave, 2019). For all of these users the use of synthetic data could be summarised to happen in a number of circumstances namely:

- when access to real data sets is difficult for various reasons. For example health or personal datasets, income datasets, and so on are usually private and not accessible for these types of purposes without specific licences and usage agreements, are understandably private,
- when real data sets are not available at the right level of granularity. It often happens that publicly available open data are often aggregated or screened,
- when real data sets are incomplete or unsuitable for the task at hand: a poor data sample may be missing key fields entirely.

Practical use-case situation: Let us consider the following use-case situation frequently encountered in classes and laboratory sessions requiring spatial data. A synthetic dataset is required to support the teacher or instructor in explaining a concept or application to the class. Real world data may or may not be available. While manually generating a synthetic dataset is not a complex problem it requires a number of steps and can take a long time (compared to the length of the class or teaching session) to create. The availability of a software tool which could quickly and accurately generate realistic synthetic datasets would allow teachers and instructors to quickly and easily use new and unseen datasets for teaching and learning. Indeed, when students require spatial data for a project application or indeed their own learning such a tool would allow for the generation of synthetic datasets they could easily incorporate into their own workflows without losing time on manual data creation.

A software tool is required which can generate synthetic or randomly-generated datasets for use as teaching and learning materials in spatial databases and spatial analysis courses. More specifically this tool must generate synthetic datasets which can scale geographically (small areas as well as large geographical areas), allow the specification of fields or attributes, operate in a semi-supervised or unsupervised fashion, and output the generated datasets to appropriate file formats such as GeoJSON or SQL files for further usage. Methodologically this tool should generate random or synthetic data which is a good substitute for real physically derived datasets.

The goals of the project loosely with the imputation of missing data. Many real-world datasets can often suffer from missing values, be that missing at random, missing not at random or missing completely at random. The ability to generate synthetic data to impute missing values can be be invaluable in many areas of data science. While imputation methods are beyond the scope of this work will be capable of being used to generate synthetic data useful for the testing and development of algorithms and data flows for data science applications.

1.2 Current Solutions

The random generation of data is applicable to a wide range of scientific fields (Golic, 2006). The ability to produce synthetic realistic data allows easier validation of models and algorithms where larger datasets can be difficult to acquire. In this section a number of related existing solutions will be outlined and discussed.

QGIS - Quantum GIS

QGIS is an open source geographic information system. It allows the plotting and mapping of spatial data points through a wide variety of file formats as well as generation of random points either within a given layer or polygon. The generation produces a set of random, approximately uniformly distributed, geographic points within a given polygon or layer. QGIS require two main parameters for such generation, the number of total points to be generated, and optionally a set minimum distance between any two points. The resulting layer can be plotted in numerous ways in QGIS as well as be exported in a variety of file formats. While this feature can be useful to quickly generate a set of points, the dataset lacks a realistic distribution, with points being spread out in the polygon in what appears to be a uniform distribution. These points are also assigned random point ID numbers, however as far as attribute data is concerned this is very limited.

Mockaroo

Online interfaces such as Mockaroo allow users to generate fake realistic datasets that can consist of a variety of columns, including ID numbers, names, locations, numbers, etc. The website allows 1000 row datasets to be generated free of charge with additional generation requiring a paid subscription. In terms of providing realistic generation Mockaroo is able to provide quality test data, in particular data rich with attributes. The tool is capable of generating Longitude and Latitude coordinates as well as Country names, Cities or area codes. The tool offers additional functionality in the form of Ruby scripting in order to alter the behaviour of columns, however it is unknown the extent of features in relation to adjusting the distribution of these spatial coordinates. With regards to user-friendliness, the UI and ease of adding new columns makes this tool quite useful for generating random geographic data.

Figure 1.2.1 shows an example of a spatial dataset generated for Ireland using Mockaroo. Unlike QGIS, which is open-source, Mockaroo is closed-source and as such we do not have access to the underlying source code and algorithms that control points generation. From repeated downloading of test data-sets, Mockaroo appears to use an internal database of pre-set locations of cities, keeping reference to the country, name, longitude and latitude of that city or town. Mockaroo then appears to randomly select from this list of cities and assign their location to the next generated row/point. In the case of Ireland, there appear to be approximately 176 unique cities/towns saved in the Mockaroo database, resulting in a large amount of clashes in the 1000 row dataset that is generated.

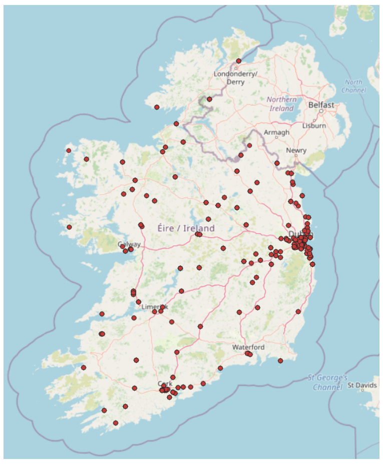


Figure 1.2.1: An example of a spatial dataset generated using Mockaroo for Ireland

GRD - from R

The generator of random data (GRD) is a software package designed for the R programming language by Calderini and Harding (2019) that allows for the creating of random datasets for use by undergraduate students, with a focus on those in the social sciences, for the practice and learning of statistical concepts. GRD allows a variety of distributions to be sampled from, by default using a normal distribution, and allows for any sample size and combination of qualitative and quantitative variables to be generated.

1.3 Our Solution - RADIAN

In this thesis we describe the design, development and implementation of a Python-based software solution called RADIAN (**RA**n**D**om spat**I**al d**A**ta ge**N**erator) for generating synthetic spatial datasets containing randomly-generated data specifically for use within a teaching and learning context in spatial databases and GIS. The target use-case teaching and learning environment is a postgraduate course on spatial databases (in particular for thesis supervisor Peter Mooney's CS621 "Spatial Databases" module delivered as Postgraduate level in Maynooth University). RADIAN will allow the specification of the characteristics of a spatial dataset including the geographic extent, number of geographic features, number of attributes per feature, and the specific data types for each feature. The software should then automatically, in a unsupervised fashion, generate a spatial dataset with these characteristics. RADIAN will utilize a Voronoi-based buffering system in order to generate points in a manner that will appear realistic to the end-user by localising generation of points within these Voronoi-polygons (Gold et al., 1996).

RADIAN should deliver or output the spatial dataset as a GeoJSON file (suitable for direct usage within a GIS system) and as a PostgreSQL dump file (suitable for import into a PostgreSQL PostGIS-enabled database). The GeoJSON file contains the representation of the dataset in GeoJSON which is an open standard format designed for representing simple geographical features, along with their non-spatial attributes. It is based on the JSON format. The PostgreSQL dump file is a text file containing SQL commands that, when loaded into the database server, will recreate the database in the same state. Both of these files can then be disseminated to students as learning materials or used by the lecturer or teacher in assessments. The data generated should allow lecturers and teachers to create data sets that look realistic but are also appropriately contextualized for the target student group and illustrative of deeper concepts (Kim et al., 2018). RADIAN will allow the specification of timestamps within ranges, randomly generated strings of characters, primitive data types and the selection of values from predefined lists of values (stored in CSV files). As RADIAN will be available as open source software via GitHub students may also use the tool for their own data generation purposes. RADIAN will greatly reduce the amount of time and effort required for teachers and instructors to generate realistic spatial datasets for teaching and learning purposes.

1.4 Structure of Thesis

The remainder of this thesis is organised as follows. In section 2 we describe background materials and related tools such as Mockaroo. Section 3 outlines the theoretical description for the problem including the outline of the methodology. Section 4 describes specific aspects of the implementation of the RADIAN software as described in section 3. Experimental analysis is outlined in section 5 for a selection of input scenarios and configurations of the RADIAN tool. The thesis closes with section 6 with some conclusions on the work and a discussion of some suggestions for future work and development on the problem of random spatial data generation.

Chapter 2

Background and Related work

In this chapter we report on a literature review to assess the state-of-the-art with regards to random data generation. We outline some of the already existing tools that perform a similar function to the proposed RADIAN software. We briefly discuss some of the positives and negatives of these existing tools.

2.1 Literature Review

To test any algorithm, software, or software-based system test data (Last et al., 2003), "dummy data"(Yee et al., 2018), or randomly generated data is usually used in such a way as to simulate the type of data or inputs that can be expected when these approaches are used in real-world situations. For example, web developers often test web-based applications with automated approaches which generate data to simulate real user inputs. These data are often generated to simulate edge cases or boundary conditions such as input of strings containing reserved programming code words or statements, very long strings, blank inputs, illegal numbers, combinations of punctuation characters and so on. Very often good software testing datasets are published openly on the Internet. For example, the Big List of Naughty Strings[1] is a large list of generated text strings. The strings compiled in the list are a collection of data that, when processed under certain circumstances, can cause problems in a web app. However, these types of data are not the focus of our RADIAN tool in this thesis. Rather, RADIAN is concerned with the generation of error-free datasets which could be considered a realistic and sufficient substitute, replacement or representative of some existing real-world dataset. For example, suppose one would like to have a spatial dataset which represents the geolocations of the confirmed cases of a specific disease. This dataset may not be publicly available for privacy reasons. However, a synthetic or randomly-generated dataset simulating such a dataset could be generated and then used in the analysis or activity requiring the data.

[1]`https://github.com/minimaxir/big-list-of-naughty-strings`

For this reason the literature search focused on the topic of synthetic or randomly-generated data for the purposes of use within an analysis or teaching and learning context. We decided against an extensive systematic literature review which took additional forms of test-data generation into consideration.

There is substantial literature available around the topic of generating synthetic data for a host of different applications and disciplines. Some authors, such as Krishnan and Jawahar (2016) who propose a framework to render large scale synthetic data for handwritten images, call the generation of synthetic data "an artform which tries to emulate natural processes as closely as possible". Very often privacy considerations drive the need for the generation of synthetic data. Drechsler and Reiter (2011) remarks that a large amount of redaction is required to protect the confidentiality of data subjects' identities and sensitive attributes many organisations decide to use synthetic data approaches instead. Papyshev and Yarime (2021) states that fully synthetic data can be understood as artificial data that are statistically or semantically similar to the original data but the new data have no identifiable information about their origin. However, there is no doubting that real behavioral data or datasets that describe human movements, actions or interactions are valuable and sensitive, in many domains, this type of data is unavailable, which makes the usage of synthetic data solutions attractive (Nikolenko, 2019).

As mentioned in the previous chapter (section 1.1) the main target application of RADIAN will be within a teaching and learning context. In the work of Kim et al. (2018) the authors argue that teaching data must satisfy three R's which are namely: "be **rich** enough to answer meaningful questions with"," be **real** enough to ensure that there is some context and "be **realistic** enough to convey to students that data like this could really exist in the real world". Eno and Thompson (2008a) shows the viability of using data mining models and inverse mappings to create realistic patterns into synthetic datasets. However, they argue that real-world datasets can sometimes have multiple patterns which might not be so easy to create inverse mappings for. The use of real-world data and scenarios can be included in learning experiences likely to enhance student learning with an emphasis on problem-based learning in real-life scenarios (Mebert et al., 2020). Creating more engaging assignments is a particular challenge to instructors, according to Burlinson et al. (2016), who suggest that improving the datasets and visualisations presented to students can improve retention levels of students in computing disciplines. In similar work to this Bart et al. (2017) argues that a major limitation of dataset preparation for teaching and learning is the expertise required to convert datasets into a format suitable for processing and usage. Although many tools and processes exists the authors argue that this "is still the work of a seasoned programmer" and not suitable for students undertaking introductory courses in computing-related topics. Mannino and Abouzied (2019) describe their tool called Synner that helps users generate real-looking synthetic data by visually and declaratively specifying the properties of the dataset such as each property's statistical distribution, its domain, and its relationship to other properties. However, we feel that one must already possess a strong knowledge or understanding of an existing datasets in order to use Synner effectively.

A very interesting approach by Such et al. (2020) describes Generative Teaching Networks (GTNs), a general approach that employs deep neural networks that generate data and/or training environments that a learner (e.g. a freshly initialized neural network) trains on for a few SGD (Stochastic gradient descent) steps before being tested on a target task. GTNs may represent a first step toward the ambitious goal of algorithms that generate their own training data. Bellovin et al. (2019) invokes the "the magic of machine learning" for the generation of synthetic data which offers a generative, additive approach and the creation of "almost-but-not-quite replica data". They provide a series of recommendations around the use of synthetic data in data privacy applications.

2.2 Related Tools or Technologies

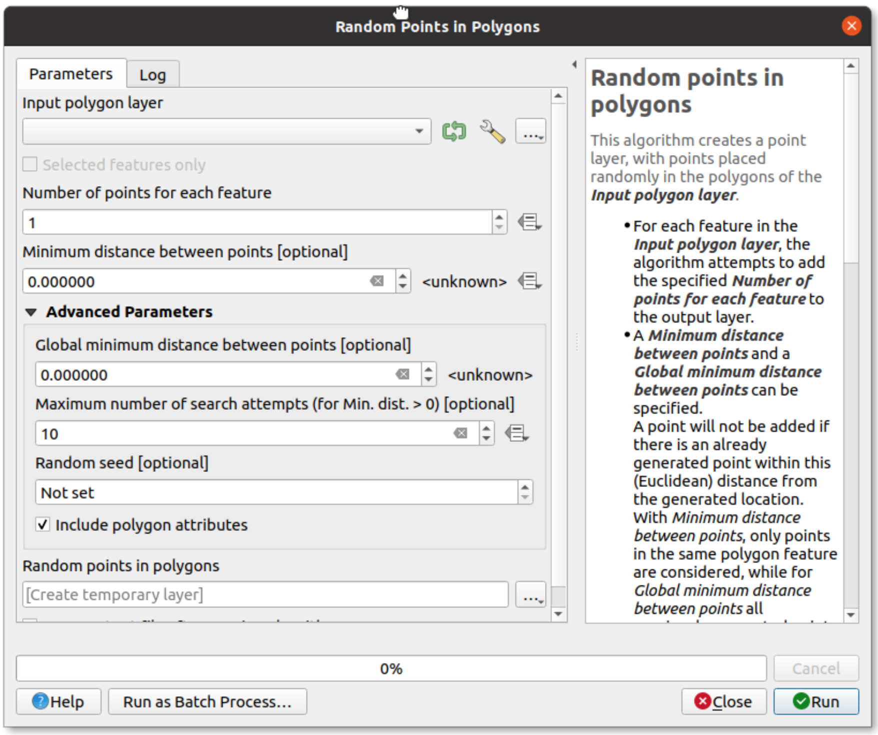
There are several tools or technologies available which can generate fake or random data. Within this section we specifically focus on those tools which can generate spatial data. In section 1.2 we briefly mentioned QGIS and Mockaroo. In the sections below we shall discuss both of these very popular and widely used tools.

QGIS - Quantum GIS

QGIS is a user friendly Open Source Geographic Information System (GIS) licensed under the GNU General Public License. QGIS is an official project of the Open Source Geospatial Foundation (OSGeo). It runs on Linux, Unix, Mac OSX, Windows and Android and supports numerous vector, raster, and database formats and functionalities. In Figure 2.2.1 on page 9 a screenshot of the QGIS dialog window for "Random Points in Polygons" [2] is shown. Here QGIS allows the specification of a polygon extent, the number of points to generate and the minimum distance between points. While an ID attribute is generated for each point there are no additional attributes generated by QGIS. The resultant dataset (initially presented as a virtual layer) is exported to an output file format of choice which includes GeoJSON, PostgreSQL dump file and ESRI Shapefile.

QGIS is an open-source tool built in Python, allowing us to look at the underlying code and algorithms at work in the generation process. For each iteration, a set of coordinates are generated corresponding to a point inside the polygon. These coordinates are generated by summing the minimum and maximum bounding box coordinates of the polygon in the X and Y axes. These results are then multiplied by a floating point value generated with Pythons `random.random()` method, which generates a floating point

[2] `https://github.com/qgis/QGIS/blob/2d1aa68f0d044f2aced7ebeca8d2fa6b754ac970/python/plugins/processing/algs/qgis/RandomPointsPolygons.py`


Random Points in Polygons

Parameters | Log

Input polygon layer

Selected features only

Number of points for each feature

1

Minimum distance between points [optional]

0.000000 <unknown>

Advanced Parameters

Global minimum distance between points [optional]

0.000000 <unknown>

Maximum number of search attempts (for Min. dist. > 0) [optional]

10

Random seed [optional]

Not set

✓ Include polygon attributes

Random points in polygons

[Create temporary layer]

Random points in polygons

This algorithm creates a point layer, with points placed randomly in the polygons of the ***Input polygon layer***.

- For each feature in the ***Input polygon layer***, the algorithm attempts to add the specified ***Number of points for each feature*** to the output layer.
- A ***Minimum distance between points*** and a ***Global minimum distance between points*** can be specified.
  A point will not be added if there is an already generated point within this (Euclidean) distance from the generated location.
  With *Minimum distance between points*, only points in the same polygon feature are considered, while for *Global minimum distance between points* all

0%

Cancel

Help | Run as Batch Process... | Close | Run

Figure 2.2.1: A screenshot of the QGIS dialog window for “Random Points in Polygons”

between 0 and 1. QGIS then checks that the point is contained within the polygon, as well as whether or not it meets the minimum distance requirements. If the point meets the criteria, then it will be appended to the feature list, and appropriate metadata including a random string variable and serial ID number are assigned to the point.

QGIS allows for a Minimum-distance parameter to be set, resulting in no two points having a distance between them lower than this value. In addition the number of search attempts can be specified. By default QGIS sets the maximum number of iterations to be $200 \times N$ where N is the total number of points to be generated. This results in the tool halting when computationally heavy point sets are requested, such as a large number of total points with a high minimum distance between points.

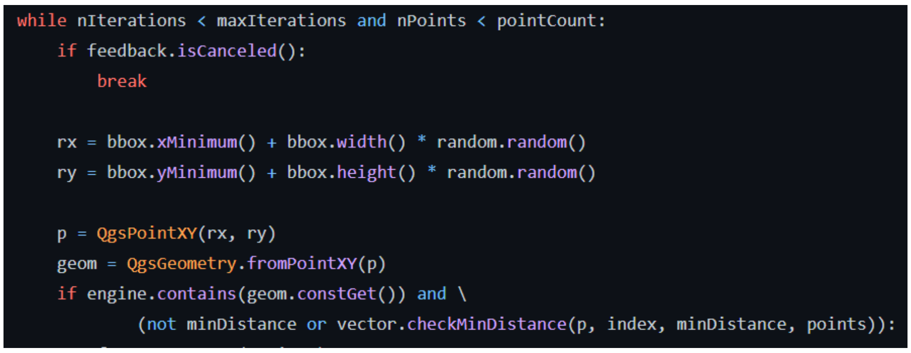
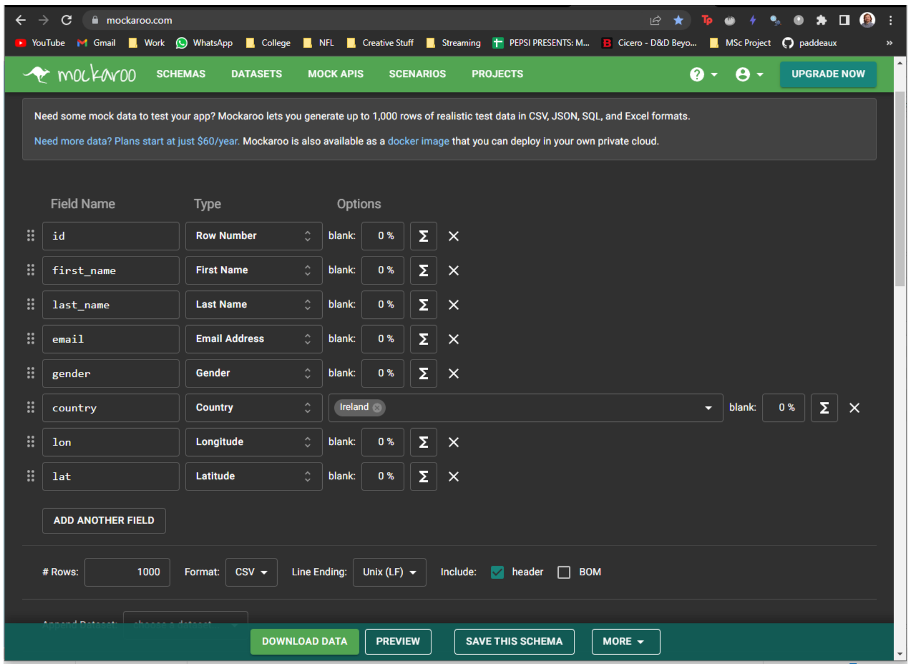

```
while nIterations < maxIterations and nPoints < pointCount:
    if feedback.isCanceled():
        break

    rx = bbox.xMinimum() + bbox.width() * random.random()
    ry = bbox.yMinimum() + bbox.height() * random.random()

    p = QgsPointXY(rx, ry)
    geom = QgsGeometry.fromPointXY(p)
    if engine.contains(geom.constGet()) and \
            (not minDistance or vector.checkMinDistance(p, index, minDistance, points)):
```

Figure 2.2.2: A screenshot of the QGIS source code - point coordinates are calculated

The points generated with QGIS do not possess any additional metadata, other than a simple integer ID value, with no options to provide extra columns or variables. To do so would require additional programming or development outside of the QGIS environment most likely by exporting the QGIS data as a format such as GeoJSON or SQLite.

Mockaroo

Mockaroo[3] on the other hand is a closed-source, premium service that allows users to create datasets of randomly generated variables, consisting of a wide variety of potential columns, such as names, email addresses, countries, and locations. Mockaroo provides a clean and friendly interface that allows users to select and name their desired variables, as well as add additional functionality through the built in Ruby scripting language. Mockaroo allows for generation of geographic data, allowing the option to include countries, cities, as well as longitude and latitude coordinates. In addition to this, filtering can be used to produce datasets centred in one location or country. Mockaroo is very easy to use as the entire interactions with the system are via a very intuitive user-interface. Figure 2.2.3 shows an example of the user-interface of Mockaroo for generating a set of geographic data points in Ireland.



Need some mock data to test your app? Mockaroo lets you generate up to 1,000 rows of realistic test data in CSV, JSON, SQL, and Excel formats.

Need more data? Plans start at just $60/year. Mockaroo is also available as a docker image that you can deploy in your own private cloud.

Field Name	Type	Options
id	Row Number	blank: 0 %
first_name	First Name	blank: 0 %
last_name	Last Name	blank: 0 %
email	Email Address	blank: 0 %
gender	Gender	blank: 0 %
country	Country	Ireland blank: 0 %
lon	Longitude	blank: 0 %
lat	Latitude	blank: 0 %

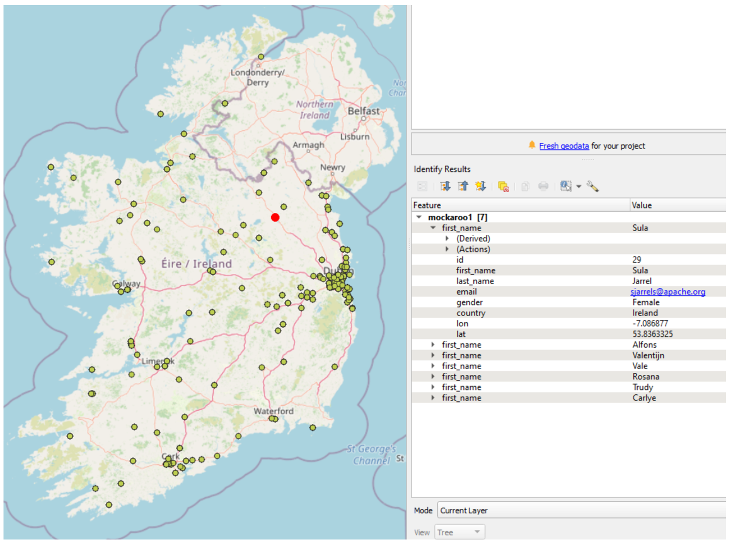
Rows: 1000 Format: CSV Line Ending: Unix (LF) Include: [✓ header] BOM

Figure 2.2.3: A screenshot of the Mockaroo UI

Mockaroo provides an easy-to-use interface with a wide variety of realistic behaving columns, available for download in a variety of formats including .CSV and .JSON. For geographic data however, Mockaroo operates under some limitations. Non-paying users are limited to datasets of just 1000 rows at a time, restricting the access and viability of the service for teaching. A major issue lies in the manner in which

[3]`https://www.mockaroo.com/`

it generates geographic data points, in particular when localized to a certain area. Using the example of Ireland, a dataset containing 1000 rows with the longitude and latitude coordinates of locations in Ireland was generated and downloaded as a .CSV file. This file was subsequently converted to a .GeoJSON format with Python and loaded into QGIS for viewing. Figure 2.2.4 shows that geographic datasets generated using Mockaroo result in multiple points being assigned the exact same location coordinates.



```
Feature                 Value
▾ mockaroo1 [7]
  ▾ first_name          Sula
      ▸ (Derived)
      ▸ (Actions)
        id              29
        first_name      Sula
        last_name       Jarrel
        email           sjarrels@apache.org
        gender          Female
        country         Ireland
        lon             -7.086877
        lat             53.8363325
  ▸ first_name          Alfons
  ▸ first_name          Valentijn
  ▸ first_name          Vale
  ▸ first_name          Rosana
  ▸ first_name          Trudy
  ▸ first_name          Carlye
```

Figure 2.2.4: A QGIS visualization of the Mockaroo-generated points showing 7 distinct points lying on the same location.

Analyzing the .csv file showed that out of the 1000 points in the dataset, there were just 176 unique locations present. This suggests that Mockaroo uses a built-in database of cities/towns in a given country, with pre-saved longitude and latitude coordinates, and simply assigns one randomly to a generated point when locations are specified. Thus the resulting dataset contains numerous duplicates, resulting in a rather poorly distributed set of points. Mockaroo provides a wide variety of common, realistic columns that can be added to generated datasets, which is invaluable for any generator of test data. However Mockaroo does not provide functionality for custom variables that would allow the generation of datasets to fit specific needs of the user at hand.

Chapter 3

Theoretical Formulation

In this chapter we outline a theoretical formulation for generation process implemented by the RADIAN software. Each distinct section of the generation process is outlined in an algorithmic context based in geometry.

3.1 Introduction to the theoretical formulation

Before outlining our implementation we outline the theoretical formulation for the generation of random points. While the real world application is spatial datasets for assessment, visualisation, and teaching the theoretical foundation is one based in geometry. We consider the problem of point generation as one of randomly generating a set of points within some two dimensional region. To deliver a realistic set of points we consider two phases to the generation of these points. Firstly we consider the **primary generation phase** which is focused on generating patterns of points as they radiate away from a specific centroid or focal point. This tries to approximate the real world effect of a small spatial area such as a city or town center with points becoming more dispersed as one moves away (radially) from this small area or region. **Secondary generation**, on the other hand, considers generating points within smaller polygon regions which attempts to approximate the real world process of administrative boundaries or districts. The patterns of point location within each smaller polygon region will be different. The final set of generated points is the union of the output from both the primary and secondary generation phases. It should be possible to use a mixture of both the primary and secondary generation phase. For example, it should be possible to choose than 60% of points are generated in the primary generation phase while the remaining 40% are generated in the secondary phase. By this rationale it should also be possible to only use one phase to generate all points.

3.2 Overall problem formulation

Given a polygon extent P generate N (with $N > 0$) point objects all of which are contained within the boundary of P. The polygon P is assumed to be a valid geometric polygon. No points are generated if P is invalid. For every $n \in N$ there will be t attributes or properties. For each point n the attribute set will be $\{n_0, n_1 \ldots n_{t-1}\}$ where $t \geq 1$. While ordering in the set $\{n_0, n_1 \ldots n_{t-1}\}$ is not required for convenience we use n_0 as the primary key or unique identifier for each point. We set a ratio r to be in the range $0 \leq r \leq 1$ such that a value for r indicates the number of points generated in the primary and secondary generation phase below. If $r * N$ points are specified to be generated in **the primary phase** then $(1 - r) * N$ points are generated in **the secondary phase**. The number of points generated in the primary generation phase is referred to as N_p while the number of points generated in the secondary generation phase is referred to as N_s. By this formulation a value of $r = 0$ will exclude the primary generation phase and all points will be generated in the secondary generation phase with $N = N_s$ and $N_p = 0$. In the same way where $r = 1$ all points are generated by the primary generation phase with $N = N_p$ and $N_s = 0$. These extreme values of r can be chosen depending on the dataset requirements.

3.3 Primary Generation Phase

Authors such as Shi et al. (2012); Dietzel et al. (2005); Nong et al. (2018) demonstrate that based on urban growth phase theory, spatial urban evolution can be a general temporal oscillation between phases of diffusion and coalescence. Diffusion is defined as the dispersion of patches, while coalescence is the fusion of patches into one patch. In the primary generation phase we use this as an inspiration of our approach where we start off with patches (Voronoi regions) and then progress through the phase by dissolving these regions which loosely simulates the process of coalescence. The polygon P is divided into 256 Voronoi regions $\{v_1, v_2 \ldots v_{256}\}$ with approximately the same spatial area. The division of the polygon P into 256 polygons was decided upon after some experimental testing and appears to work appropriately for most input polygons P. There are four main steps within the primary generation phase and these are outlined as follows:

1. Centroid selection
2. Distance classification based on the selected centroid
3. Boundary coalescence or dissolving of the Voronoi regions $\{v_1, v_2 \ldots v_{256}\}$
4. Point generation within coalesced boundaries

Centroid selection

In the primary generation phase there are two options for distance calculation based on the selection of the input polygon P's centroid. These are outlined as follows:

1. **True Centroid:** The actual centroid of the input polygon P is calculated as C_P and is used in all distance calculations in the primary generation phase

2. **Moving or random centroid:** Here the centroid C_{P_m} of the input polygon P is assumed to be a randomly choosen point in the region around the true centroid C_P

Distance classification based on the selected centroid

In the next step we use the true centroid C_P to classify the distance from each Voronoi region $\{v_1, v_2 \ldots v_{256}\}$ to the corresponding choice of centroid. Five distance classes are constructed using the true centroid C_P. This is illustrated in Figure 3.3.1 and Figure 3.3.2. The radius of this circle is calculated as a circle which contains or intersects all of the Voronoi regions $\{v_1, v_2 \ldots v_{256}\}$.

1. **True Centroid:** In this case the circular region or distance class for each Voronoi region $\{v_1, v_2 \ldots v_{256}\}$ is calculated by the distance from the centroid of each region $\{v_1, v_2 \ldots v_{256}\}$ to the true centroid C_P of C. The colouring of regions in Figure 3.3.1 illustrates this concept visually.

2. **Moving or random centroid:** In this case the circular region or distance class for each Voronoi region $\{v_1, v_2 \ldots v_{256}\}$ is calculated by the distance from the centroid of each region $\{v_1, v_2 \ldots v_{256}\}$ to a randomly chosen point C_{P_m} representing the centroid of C. The point C_{P_m} is always contained within the first circular region. The colouring of regions in Figure 3.3.2 illustrates this concept visually.

Boundary coalescence

Each Voronoi region $\{v_1, v_2 \ldots v_{256}\}$ is assigned a distance class based upon which of the 5 circular distance regions it lies within. Coalescence or dissolving of the Voronoi region polygons entails combining polygons from $\{v_1, v_2 \ldots v_{256}\}$ based upon a unique attribute value and removing their interior geometry. This unique attribute value is their distance classification. Dissolving is mainly used for generalization and always creates simplified boundaries from their more complex parent boundaries. The result of this process is shown in the right hand side of both and Figure 3.3.1 Figure 3.3.2 where 5 large polygons are created from the boundaries of the Voronoi regions within each distance class.

Point generation within coalesced boundaries

At this point the process of generating random points begins within the primary generation phase. The 5 polygons from the coalescence stage are used and $r * N$ points must be generated. As there are 5 polygons we generate $\frac{r*N}{5}$ points for each polygon.

For the inner most polygon we generate $\frac{r*N}{5}$ points. Then for the two inner most polygons we generate $\frac{r*N}{5}$ points. This continues for the three inner most and so on until all $N_p = r * N$ points are generated.

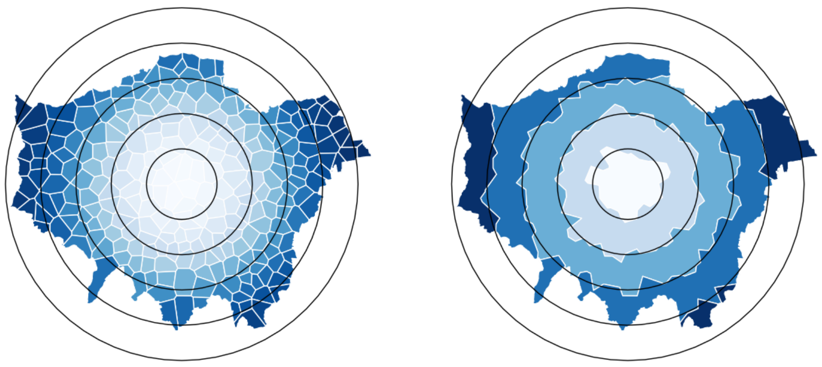


Figure 3.3.1: Classifying distance based on circular regions around the centroid for the input polygon and the Voronoi regions

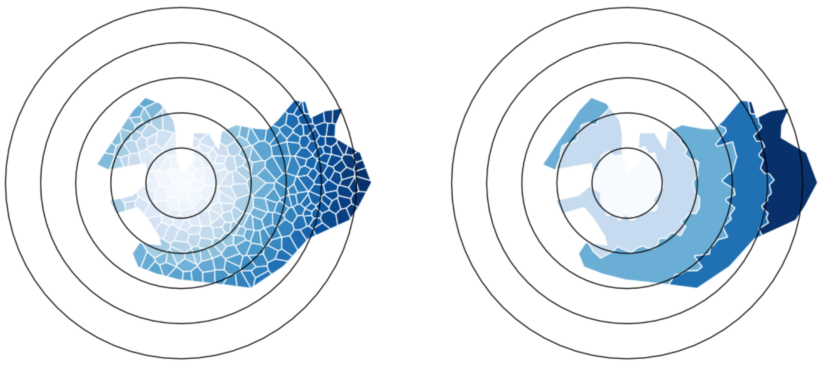


Figure 3.3.2: Classifying distance based on circular regions around a moving centroid for the input polygon and the Voronoi regions

3.4 Secondary Generation Phase

This phase of point generation is separate to the previous phase and shall only happen if $(1 - r) * N > 0$. Here we use Voronoi polygons again but these are different to those generated in the primary generation phase. Here V voronoi polygons are generated within P. Points are generated within each of the Voronoi polygons in V as follows:

- **Type 0:** This means that there is no secondary generation phase implemented at all. Then $N_s = 0$ and $N_p = N$.
- **Type 1:** The Voronoi polygons in V all have approximately the same spatial area. Each $v_i \in V$ is allocated the same number of points which is $\frac{N_s}{V}$.
- **Type 2:** The Voronoi polygons in V have variable spatial area and the number of points allocated is weighted by the area - larger polygons $v_i \in V$ will have more points than smaller area polygons.
- **Type 3:** The Voronoi polygons in V have variable spatial area but $v_i \in V$ is allocated the same number of points which is $\frac{N_s}{V}$

For Type 1, 2 and 3 the points are generated in the Voronoi polygons as specified above.

3.5 Attribute generation

At this point N points have been generated where $N = N_p + N_s$. The final step is to generate the required attribute information (property name and value). As stated above n_0 is the primary key or unique identifier for each point. Within the remaining properties $\{n_1, n_2 \ldots n_{t-1}\}$ there will be one integer value, one randomly generated string and a timestamp expressed within a range of datetimes T_1 and T_2 where T_1 is chronologically before T_2. The remaining properties, if specified, are drawn from external lists where property n_x has its value drawn from a list x. The list x may have duplicate values which act as a weighting for values within the list. Values within the list x have a higher probability of being assigned to n_x when they have higher overall weighting.

Chapter 4

Software Implementation

In this chapter we outline in detail the software implementation of the RADIAN project. We describe the use of the RADIAN software and we outline the software specifications and requirements. The input parameters are described and the implementation of each stage of the random generation process is outlined, from the initial buffer generation, to the primary and secondary points generation, as well as the additional metadata generation.

4.1 How will RADIAN be used in practice?

Why is this piece of software needed? In section 1.1 we gave a short description of the practical implementation of the RADIAN software. Here we can describe this practical usage in more detail. It is important to consider the practical need for this tool within the environment of the classroom or laboratory teaching session. In a subject such as Spatial Databases sample datasets are required for students to use. The teacher must try to find the most suitable datasets for teaching examples, assignments, sample assessments and mandatory assessments. When we say suitable we are considering a dataset with the following characteristics:

- **Features:** The dataset contains an appropriate number of geographical features with those features having an appropriate number of attributes
- **Scale:** The dataset is represented in a suitable geographical scale - for example representing the geographical area of a city, state or province, country or another arbitrary spatial area.
- **Sensible:** When a particular spatial analysis technique, algorithm or query is executed on the dataset the returned results or outputs are sensible within the context. An example of a situation we want to avoid is where the points in the dataset are all clustered closely together within a small

spatial area.

- **Realistic:** The dataset while obviously randomly generated can easily be accepted or imagined by both the students and the teacher as being representative of a similar real world dataset. An example of such a representative dataset is illustrated in Figure 4.1.1 where the real world context is given as the locations of positive tests for a virus or illness in a study population.

- **Interesting:** This is certainly a very subjective characteristic but is mainly based on the experience and knowledge of the teacher or instructor. Students usually engage more with the data if there is some relatable context to the data - for example some thematic content that is attractive to the student cohort such as location of cafes or restaurants in a city.

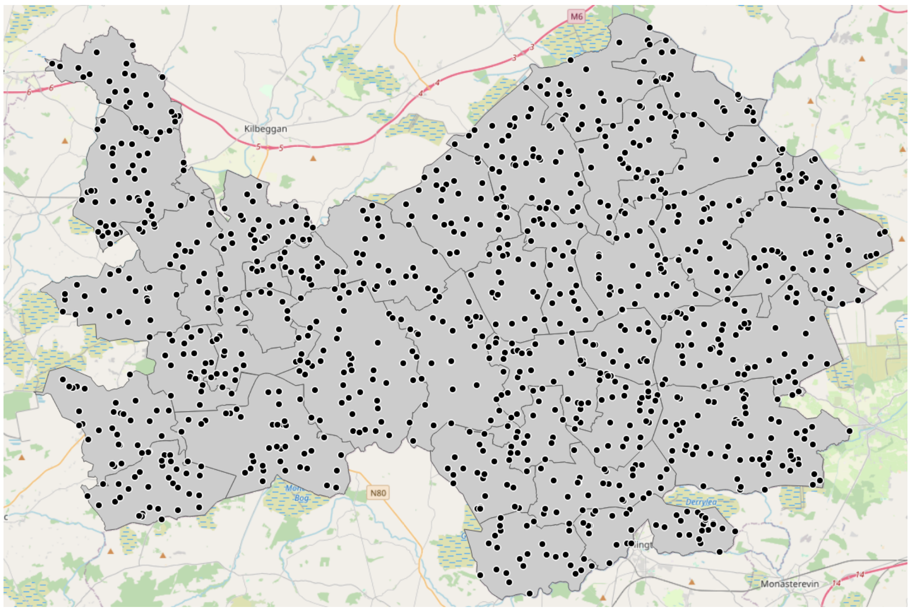


Figure 4.1.1: An example of a randomly generated point dataset for part of County Offaly, Ireland. This dataset satisfies all of the conditions outlined in section 4.1

4.2 Software Specification

The primary focus of this project is to produce a tool which will generate a set of random geographic points for use by students in assessment. More specifically the software tool should provide the following functionality:

- RADIAN should attempt to generate points in a manner such that they appear as if they were representations of real-world data, rather than randomly generated.

- RADIAN should allow for input of a GeoJSON polygon for the boundary polygon or extent within which points will be generated. This polygon can be any well-formed polygon.

- RADIAN is intended to be used for the generation of spatial data for many different regions and areas of the world. This tool must be completely independent of any existing datasets in the real world. The tool should operate without any training or linkages to existing datasets.
- RADIAN should provide functionality for the exporting of generated data in both GeoJSON and PostgreSQL formats.
- RADIAN should allow for the creation of attributes or properties for each of the generated points. By default these should include randomly generated integers, floats, strings, and timestamps. Where appropriate ranges (min, max) values should be specified.
- RADIAN should specify all coordinates in WSG84 or EPSG:4326 (Latitude Longitude)
- In addition to the default properties from the previous point, the generated points should optionally allow the addition of extra meta-data to allow the data to appear more realistic, through the importing of .CSV files containing lists of potential values along with weights to inform the distribution of these values.
- RADIAN should allow for granular control of the spatial generation, allowing for multiple methods of data generation, to be specified by the user. A default, silent run, mode should also be available

4.3 Software Implementation Requirements

At the outset of this project it was clearly communicated that the proposed software must be implemented in Python. It should also be easy to install and use on other Python-ready systems. To make the software as reusable (and indeed capable of future development) a specific set of libraries should be used. Therefore, the implementation requirements for the software are summarised as follows:

Python: The project required that RADIAN be written in the Python programming language. Python is an incredibly versatile language with a wide variety of packages well suited to manipulation of geographic data as well as random generation. The tool was written using Python 3.10 along with up-to-date versions of the following packages.

GeoPandas: GeoPandas [1] is a GIS-focused extension of the popular ”Pandas” Python package. Pandas allows for the creation of data structures known as ”DataFrames” which allow for easy storing, arranging and manipulation of data through a variety of input and output formats. GeoPandas, in conjunction with the Shapely package, extends Pandas’ functionality to allow the performing of spatial operations on

[1] `https://geopandas.org/en/stable/`

Geometric datatypes, such as Points, Linestrings, and Polygons. The native support for importing and exporting GeoJSON file formats, a common datatype for the storing of geospatial datasets, provided further justification for the use of this package.

Shapely The Shapely[2] package allows for the implementation and manipulation of spatial objects in Python. It is heavily utilized by the GeoPandas package as any geometry object stored in a GeoDataFrame is in the form of a Shapely object. Shapely Points and Polygons are used extensively throughout the generation process.

Geovoronoi: The Geovoronoi[3] package allows for the creation of Voronoi diagrams, which are a core element of the random point generation algorithm. A Voronoi diagram is a collection of polygons, each with a "seed", that consist of regions with points that are closer to that regions' seed than to any other seed. The main functionality of the package is the takes a list of points/coordinates as input and calculates the corresponding Voronoi regions. This is used along with Shapely in order to produce a set of Voronoi polygons contained inside the original GeoJSON inputted source polygon. This in turn allows for a variety of different generation patterns.

Sklearn: Scikit-Learn, or Sklearn[4], is a machine learning library for Python. The primary use for it in this project is its implementation of the K-Means clustering algorithm. K-Means clustering is used on a set of uniformly generated points in order to divide a spatial region into an arbitrary set of sub-regions for the purposes of random point generation.

GeoJSON: The GeoJSON[5] package allows for easy importing and exporting of GeoJSON file formats as well as objects such as Features and FeatureCollections which are used to produce GeoDataFrames alongside geopandas.

Random: The random[6] package in Python allows functionality for pseudo-random number generation from various distributions, including the generation of random ranges as well as individual numbers. The uniform distribution is the primary means of generating random coordinates for geographic points within the bounds of a polygon.

Matplotlib: Producing plots and visualizations of both the points and the various stages in the buffering process is invaluable for debugging the tool and ensuring that it is performing as intended. Matplotlib[7] provides functionality for producing plots of the final output on the original polygon and/or a basemap provided by OpenStreetMap via the Contextily package.

[2]`https://pypi.org/project/Shapely/`
[3]`https://pypi.org/project/geovoronoi/`
[4]`https://scikit-learn.org/stable/`
[5]`https://pypi.org/project/geojson/`
[6]`https://docs.python.org/3/library/random.html`
[7]`https://matplotlib.org/`

Github: A Github[8] repository was created for the project to allow for easy version control and reproducibility of the software. It will also allow for both instructors and students to download and use software.

4.4 RADIAN Input Parameters

RADIAN is controlled by a user edited JSON file consisting of a dictionary of parameters, allowing for easy editing of the runtime conditions of the generation. This was decided during the early stages of the development. The JSON configuration file removes the need for a specific user-interface to allow for the selection of parameter values. The parameters and their input details are outlined as follows:

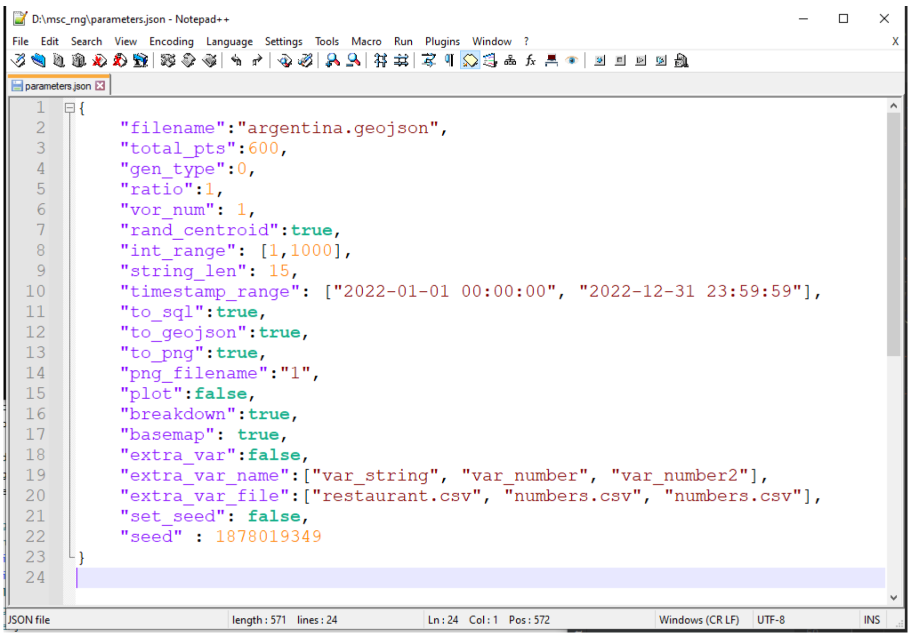
- `filename` : Provides the name/directory of the .geojson source polygon.
- `total_pts`: The total number of points to be generated within the source polygon.
- `gen_type`: Controls the type of secondary generation to be used. Expects a number between 0 and 3:
  - **0** results in no secondary generation taking place.
  - **1** results in secondary generation with voronoi regions of approximately equal area, each with an equal proportion of the total secondary points generated within.
  - **2** results in secondary generation within voronoi regions of variable area, each with proportion of points based on their geographic area.
  - **3** results in secondary generation within voronoi regions of variable area, each with an equal proportion of points.
- `ratio`: A value between 0 and 1 that indicates the ratio of primary to secondary points to be generated. 1 would result in 100% of the total points being assigned to primary generation, 0.5 would result in a 50/50 split between primary and secondary generation.
- `vor_num`: The number of regions to be generated in the secondary point generation. This is constrained to be a max of 32 for variable-area generation and 256 for equal-area generation.
- `rand_centroid`: A boolean indicating if a moving random centroid (true) or the original source polygon centroid (false) is to be used in points generation.
- `int_range` : A list with length two, indicating the range of the random integers to be generated and assigned to each point, e.g. `[1,100]`

[8]`https://github.com/paddeaux/`

- `string_len` : Integer value that specifies the length of the random string to be generated and assigned to each point.
- `timestamp_range` : A list with length two, indicating the range within the random timestamp for each point is to be generated, e.g. `["2022-01-01 00:00:00", "2022-12-31 23:59:59"]`
- `to_sql`: A boolean that indicates if the final dataset will be exported to an SQL dump file upon execution.
- `to_geojson`: A boolean that indicates if the final dataset will be exported to a GeoJSON file upon execution.
- `to_png`: A boolean that indicates if the final dataset will be outputted.
- `plot`: A boolean that indicates if the final dataset will be plotted using *matplotlib* along with the source polygon upon execution.
- `breakdown`: A boolean that indicates if the above plot will include a breakdown of the final generation by displaying the primary, secondary, and full points set on individual plots.
- `extra_var`: A boolean that indicates if extra variable columns will be added using inputted CSV files.
- `extra_var_name`: A list of strings indicating the variable name(s) to be used for the extra variables added with the additional CSV file(s).
- `extra_var_file`: A list of strings indicating the file names of the CSV files corresponding to the given additional variables to be added to the dataset.
- `set_seed`: A boolean that indicates if the user will set a generation seed, allowing for reproducibility.
- `seed`: Integer value that serves as the seed for random generation.

4.5 Primary Generation Implementation

The total number of points is divided between primary and secondary generation, dictated by the `ratio` parameter. The desired generation pattern is one that reflects a real distribution of points in a geographic area which can be more specifically an urban region or city. The most common pattern observed in geographic data of cities is that of point locations, eg. schools, shops, etc., being concentrated more greatly towards the central hub of the region (Lan et al., 2020; Calafiore et al., 2021), as shown in Figure 4.5.1 which shows the real world location of fast food restaurants in Greater London. To replicate this type of


D:\msc_rng\parameters.json - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
parameters.json
JSON file
length : 571 lines : 24
Ln : 24 Col : 1 Pos : 572
Windows (CR LF)
UTF-8
INS

```
{
    "filename":"argentina.geojson",
    "total_pts":600,
    "gen_type":0,
    "ratio":1,
    "vor_num": 1,
    "rand_centroid":true,
    "int_range": [1,1000],
    "string_len": 15,
    "timestamp_range": ["2022-01-01 00:00:00", "2022-12-31 23:59:59"],
    "to_sql":true,
    "to_geojson":true,
    "to_png":true,
    "png_filename":"1",
    "plot":false,
    "breakdown":true,
    "basemap": true,
    "extra_var":false,
    "extra_var_name":["var_string", "var_number", "var_number2"],
    "extra_var_file":["restaurant.csv", "numbers.csv", "numbers.csv"],
    "set_seed": false,
    "seed" : 1878019349
}
```

Figure 4.4.1: An example set of parameters stored in a JSON dictionary

pattern in RADIAN we organise the region into five circular regions around the source polygon centroid regardless of whether this is the original centroid or the randomly generated moving centroid (controlled by `rand_centroid`). The first circular region includes the 'hub' or 'center' whereas the fifth or outer circular region encloses the entire region or polygon. In the case of Figure 4.5.1 this will include the entire polygon for Greater London.

In order to achieve the desired pattern of points distribution, that is where points are more concentrated towards the centroid with density decreasing the further away from said centroid, a series of five Voronoi-based buffer regions are created. Python's Shapely package provides a .buffer method that produces circular buffers which could be used for this purpose, however the regular and predictable shaping of these buffers lead to these circular patterns being visible for higher numbers of points being generated. Voronoi regions were instead generated to create irregularly shaped regions that would be influenced by the overall shape of the original polygon, while still maintaining a strong element of randomness.

The algorithm for buffer creation is as follows:

- A set of 256 approximate equal-area Voronoi regions is generated inside the given source polygon
- For each of the above Voronoi regions, the distance between their centroid, and the source polygon centroid d is calculated and stored in a geodataframe.
- The maximum distance is then used to determine the classification of each voronoi polygon:
  - Five distinct buffer regions are required, so the variable `dist_break` $= (d_{max} - d_{min})/5$

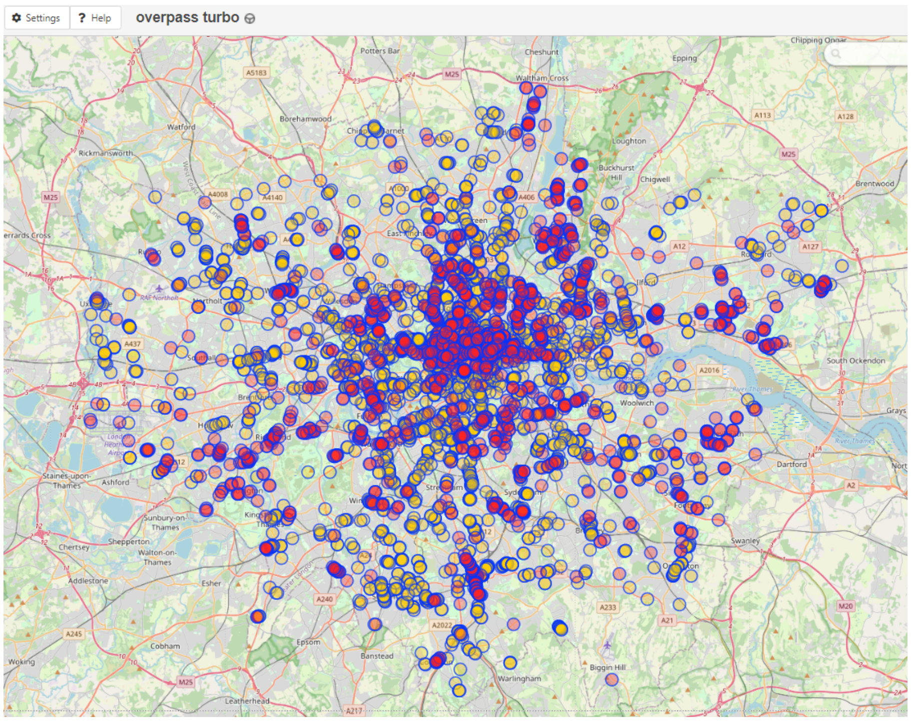


Figure 4.5.1: A real set of fast food restaurant locations in Greater London taken from OpenStreeMap using the OverPass Turbo API

is calculated.

  – `dist_break` $\times\ i$ where $1 <= i <= 5$ will give the radius of each successive buffer region.
  – The Python pandas `.cut` method is used to assign each polygon a class numbered 1 to 5, based on the value of d for each voronoi polygon
  – Voronoi polygons with d in the range $[0, \texttt{dist\_break} \times 1]$ would be assigned to class 1, voronoi polygons with d in the range $[\texttt{dist\_break} \times 1, \texttt{dist\_break} \times 2]$ will be assigned to class 2, etc.

- The classified Voronoi polygons are then unioned into the five distinct buffer regions using the GeoPandas `.dissolve` method

Figure 4.5.2 shows an example of a set of 256 generated Voronoi regions within a source polygon, along with the resulting Voronoi-based buffer regions that are produced as a result of their unions.

In the case of the greater London area, it's urban centre lies approximately around it's the true geographic centre of the region, while on the other hand areas such as Boston or Tokyo have their urban centres located closer to the coast. To reflect this behaviour the `rand_cent` parameter will result in a *moving centroid* being generated in a rectangular region around the true centroid. This allows further randomness as the distribution of points is entirely shifted accordingly depending on the generated moving centroid.

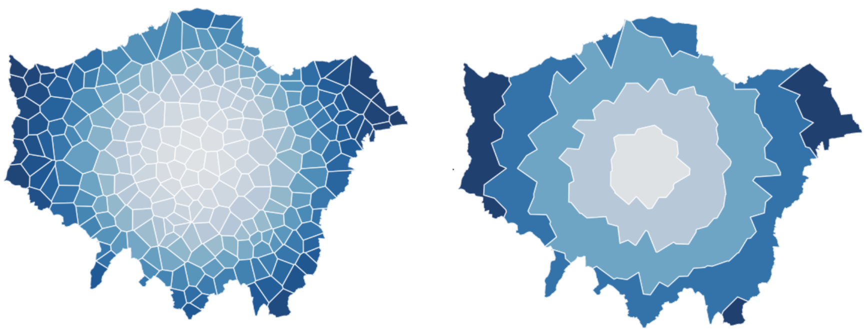


Figure 4.5.2: Voronoi-based buffer regions (right hand side) for Greater London (left hand side 256 regions)

Figure 4.5.3 shows an example similar to Figure 4.5.2, this time using a randomly generated moving centroid to serve as the centre point of generation.

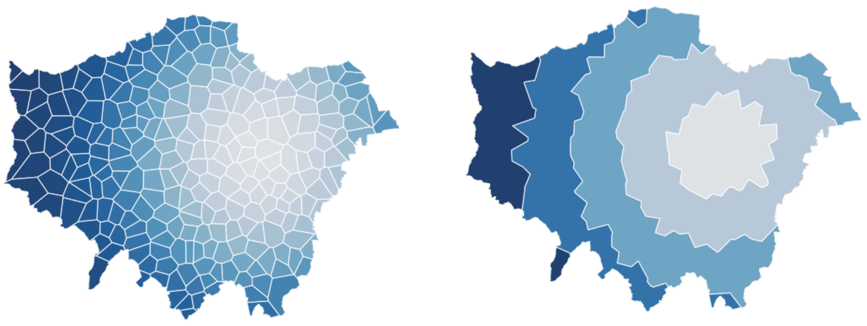


Figure 4.5.3: Voronoi-based buffer regions around a moving centroid

Points generation is performed using circular buffers of increasing size. In a given polygon, five circular buffers of increasing size are generated centred around the given centroid. These buffer regions overlap one another, meaning that first region is enclosed by the second, the first and second are enclosed by the third and so on. The overlapping nature of these buffers results in a higher probability of points generating in the earlier/smaller regions being much higher than that of the later/larger regions. This generation results in points concentrating more strongly towards the centre of the region with density decreasing the further out from the centroid.

Point coordinates are generated using the `random` Python package by choosing Longitude and Latitude coordinates from a uniform distribution between the ranges of the valid bounds of the given polygon region. Once a point has been generated, the tool uses the Shapely `.within` method to check firstly if the point lies within the source polygon, and then if the point lies within the current circular buffer. If both conditions are true then the point will be appended to a list data structure. Once all points have been generated the list is returned as a GeoPandas `GeoDataFrame`.

4.6 Secondary Generation Implementation

While the primary stage performs point generation using the full extent of the source polygon, the secondary stage allows for more granular and localized generation of points. This allows for further variation in the overall set of generated points as well as reflecting realistic patterns of points distribution in real-world settings. For example the secondary generation could be used to simulate districts in a city or perhaps towns in larger areas. It can also be used to simulate the development of clusters of points away from the central areas of the input polygon extent. The `vor_num` parameter controls the number of Voronoi regions to be created in the secondary generation stage. In in order to generate a set of points with which the Voronoi regions can be generated, the K-Means clustering algorithm is implemented using the Python `Sklearn` package. This allows for the creation of a K-Means object that is assigned the number of required clusters, i.e. the number of Voronoi regions required, and a set of 500 randomly generated points in a uniform distribution. By performing K-Means clustering on these uniform points, a set of points representing the centroids of these clusters are generated, which in turn are used to then create the appropriate number of secondary Voronoi regions.

The method of secondary generation is controlled by the `gen_type` parameter:

- `gen_type` $= 0$: No secondary generation takes place, final points set will only consist of those generated by primary generation.
- `gen_type` $= 1$: Secondary generation is carried out with approximately equal-area Voronoi regions. This is achieved by using uniformly spaced K-Means cluster centroids in the generation process.
- `gen_type` $= 2$: Secondary generation is carried out with variable-area Voronoi regions. This is achieved by using centrally focused K-Means cluster centroids in the generation process, resulting in smaller secondary Voronoi centroids towards the centre of the source polygon and larger regions out towards the boundaries of the source polygon. In this case points are assigned to each region based on the size of the Voronoi polygon, with larger polygons receiving more points than smaller regions.
- `gen_type` $= 3$: Secondary generation is carried out with variable-area Voronoi regions, as in the above generation method. Here points are equally distributed among all of the secondary Voronoi regions.

Figure 4.6.2 demonstrates the difference in the pattern of secondary Voronois between the equal and variable-area Voronoi methods. If the `gen_type` parameter has been set to zero, then no secondary

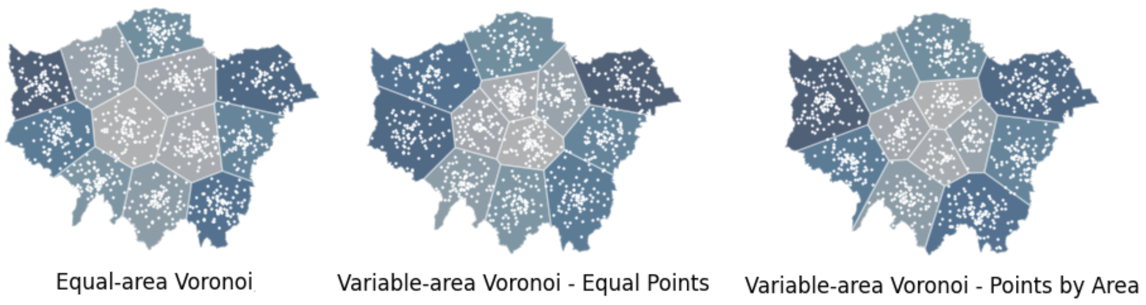

Equal-area Voronoi
Variable-area Voronoi - Equal Points
Variable-area Voronoi - Points by Area

Figure 4.6.1: Secondary Generation of 1000 points in 12 Voronoi regions with each secondary generation type

generation will occur and the `GeoDataFrame` generated in the primary stage will be used in the following metadata generation stage (section **??**). Figure 4.6.1 shows a side-by-side comparison between the three secondary generation methods.

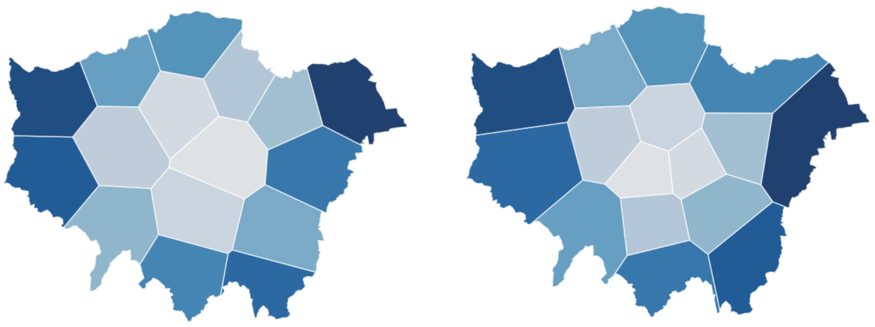


Figure 4.6.2: Example of equal (left) and variable (right) area voronoi regions

The `ratio` parameter specifies the proportion of the total points to be generated that will be assigned to the primary generation and to the secondary generation. A value of 0.6 would result in 60% of the points being assigned to the primary generation and the remaining 40% would be assigned to the secondary generation. An demonstration of this is shown in Figure 4.6.3. The `set_seed` and `seed` parameters allow the user to specify a particular Integer seed value that will influence the random generation. This allows generated datasets to be reproduced on separate machines once identical parameters and seed values are set between both instances.

4.7 Additional point attribute generation implementation

After initial points generation each point is assigned three randomly generated variables, including a random string of ASCII characters, a random integer value, and a random timestamp. These are described as follows:

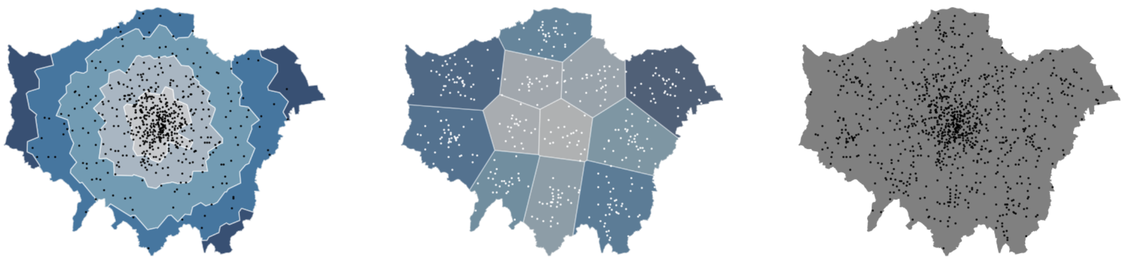
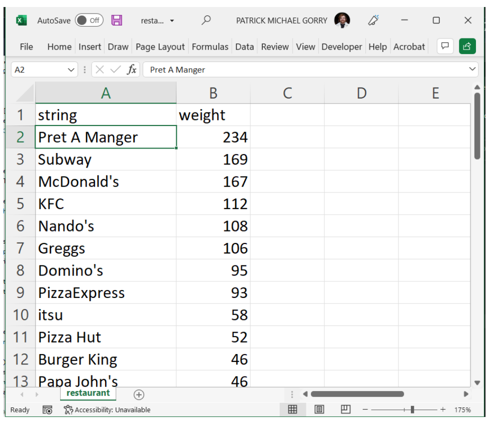


Figure 4.6.3: Random generation of 1000 points with a 60/40 split of primary to secondary generated points

- `rand_string`: A random string of alphanumeric characters of length `string_len` as specified in the program parameters.
- `rand_int`: A random integer value within the range specified in the program parameters, `int_range`.
- `rand_ts` : A random ISO format timestamp value within the range specified in the program parameters, `timestamp_range`.

In addition to the above values, points can be assigned custom number and string variables based on the input from a CSV file specified by the user. The `extra_var` parameters allow the user to specify the additional variable names as well as the specific CSV files with which to generate the data. Each CSV file must consist of two columns, the first with a list of potential values (e.g. a list strings containing common restaurant names) followed by a column of corresponding weights for each name. These weights allow the user to specify a distribution by which these values should appear in the dataset. Higher weights indicate that values will appear more often in the final dataset compared to values with lower weights. The weightings can all be set to the same value to ensure that every value has an equal probability of appearing in the output data.



	A	B	C	D	E
1	string	weight			
2	Pret A Manger	234			
3	Subway	169			
4	McDonald's	167			
5	KFC	112			
6	Nando's	108			
7	Greggs	106			
8	Domino's	95			
9	PizzaExpress	93			
10	itsu	58			
11	Pizza Hut	52			
12	Burger King	46			
13	Papa John's	46			

Figure 4.7.1: Example of additional variable CSV file

Chapter 5

Experimental Analysis

In this chapter the parameters and subsequent outputs for a number of differing scenarios will be outlined and discussed in order to demonstrate how the developed RADIAN software meets the requirements of this project. It also gives a flavour of the many possible different types of outputs that can be generated with the RADIAN software.

5.1 Overview of the scenarios

We call a specific run of RADIAN with a particular set of parameters a *scenario*. In the sections below we have chosen both geometrically and spatially interesting scenarios. Some of the scenarios are based upon the need to test RADIAN for different shapes while the other scenarios are taken from experience with the teaching of CS621 Spatial Databases in Maynooth University. Over the years of teaching this module certain types of datasets have emerged as being popular among the students. Section 5.1.1 describes the random spatial data generated for various geometric shapes whilst the remaining sections describes the process for some well known spatial regions.

The following sections consider more familiar geographic boundaries for the remaining scenarios. In section 5.1.2 we consider the geographic boundary of Greater London which is used in several examples in CS621 and often used as the source for data for Continuous Assessment lab exams. We then move to section 5.1.3 where the country boundary for Argentina is used. This scenario is motivated by our interest in how RADIAN reacts to this very elongated shape. The scenario in section 5.1.4 depicts the country of Iceland with its mostly convex shape except for the extreme north east. The scenario presented in section 5.1.5 considers the largest geographical area in our examples namely the United States of America. We consider generating a fictitious dataset representing the approximate locations of fast-food restaurants within the United States. A similar dataset is used depicting coffee shops in one

CS621 homework assessment. Finally, we look at point generation for the same polygon shape with RADIAN, QGIS and Mockaroo. The experimental analysis chapter closes with section 5.2 where we provide a discussion of the observations and results from the scenarios.

A complete set of scenarios for `gen_type` $\{0, 1, 2, 3\}$ is provided in the GitHub repository[1] for this work. Furthermore, each scenario contains both moving centroid and original centroid generation. For both of these cases primary generation only, secondary generation only and standard (a mixture of both) are also shown. Due to the constraints of space it is not feasible to show all possible configurations or parameters here in the written thesis.

5.1.1 Scenario 0 - Geometric Shapes

In order to observe how RADIAN performs in general, a number of scenarios were performed with simple geometric objects given in `.GeoJSON` format. As well as demonstrating how point generation performs, these scenarios provide clear illustrations of how the overall shape of the input polygon influences the generation, in particular how the shapes of the Voronoi-based buffers are informed by the original source polygon. Figure 5.1.1 displays an example of the resulting points generation within a collection of ordinary geometric shapes, including a circle, square, triangle and a star. The main parameter values set for each shape were as follows:

- **Circle**: 3000 total points at `gen_type` = 1, at a 0.5 ratio with 12 secondary Voronoi regions, using the original centroid for primary generation. The **Circle** provides a valuable scenario because it is the geometric shape used by the primary generation phase of the RADIAN tool. This scenario allows us to see the distribution of points within this geometric object.
- **Square**: 2000 total points at `gen_type` = 2, at a 0.7 ratio with 6 secondary Voronoi regions, using the moving centroid for primary generation. The **Square** object is not very common in real-world geographic shapes for administrative boundaries. However, we used this scenario to investigate how RADIAN would distribute points within this shape.
- **Triangle**: 5000 total points at `gen_type` = 2, at a 0.8 ratio with 10 secondary Voronoi regions, using the moving centroid for primary generation. The **Triangle** was used to investigate how RADIAN would generate points given the circular regions of the primary generation phase would circumscribe the triangle.
- **Star**: 2000 total points at `gen_type` = 3, at a 0.5 ratio with 5 secondary Voronoi regions, using the original centroid for primary generation. The **Star** shape was used to investigate how RADIAN would deal with both convex and concave regions in the same input shape.

[1]`https://github.com/paddeaux/msc_rng`

- **The shape of S**: 2000 total points at `gen_type` = 2, at a 0.8 ratio with 8 secondary Voronoi regions, using the moving centroid for primary generation. The **Shape of S** was used to investigate how RADIAN would deal with a very elongated shape. The shape of S is reasonably common in geographic shapes as this could be considered as a respresentation of real-world features as rivers, river flood plains, and so on.

We were very pleased with the resultant point generation for all of the chosen geometric shapes. The shape of S is probably the one shape which is most difficult because of the very narrow shape and points are clustered heavily at the border of the shape. This could be representative of point locations near the bank of a river.

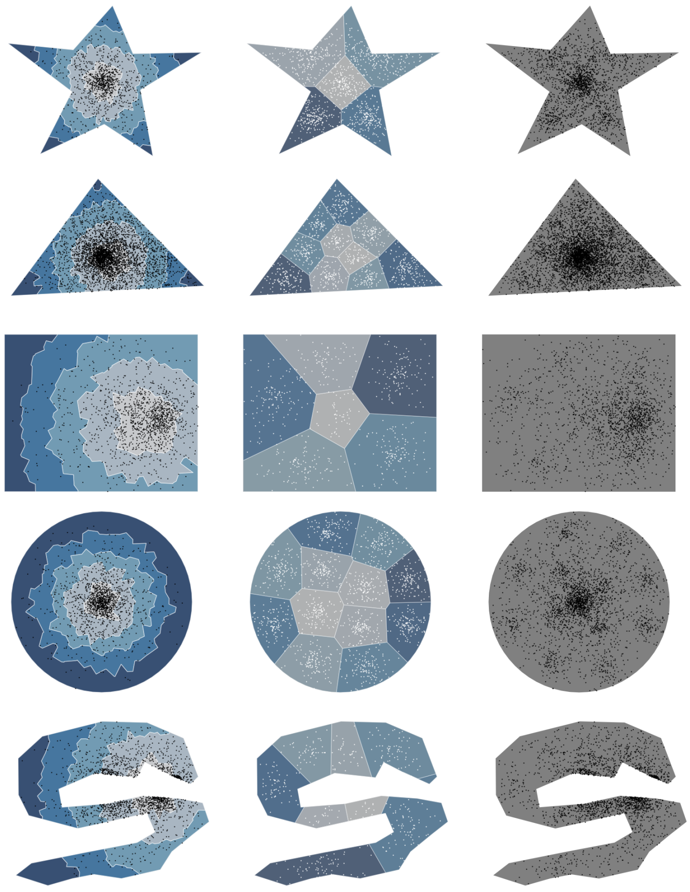


Figure 5.1.1: Random generation within a selection of normal geometric shapes

5.1.2 Scenario 1 - Greater London Area

This scenario demonstrates the first type of generation which only utilizes the primary generation stage. This is achieved by setting `gen_type` = 0 and `ratio` = 1. The polygon in this scenario represents the Greater London area. This was chosen for being recognisable visually and the fact that there already exists a great deal of open data sources for the area. This scenario also demonstrates the moving centroid functionality with the breakdown shown in Figure 5.1.2 showing how the moving centroid, and overall shape of the original polygon, influence the form and location of the Voronoi-based buffers created during the primary generation phase.

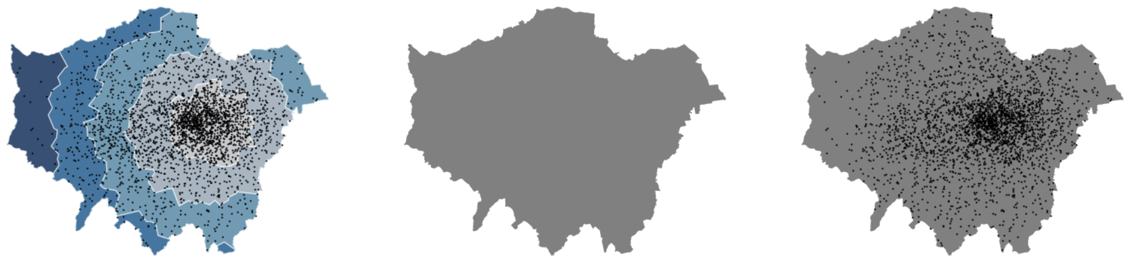


Figure 5.1.2: Scenario 1: Primary generation with a polygon of the Greater London Area

5.1.3 Scenario 2 - Argentina

This scenario demonstrates the equal-area, equal-proportion stage of secondary generation. A simplified polygon representing Argentina was selected for this scenario. The elongated shape of this polygon is unique and demonstrates the ability of RADIAN to deal with unusually-shaped inputted polygons. In this scenario 3000 points are generated in a 50/50 ratio with 8 secondary Voronoi regions and a moving centroid for primary generation. The shape of this polygon is unique in it’s elongated shape displaying how the tool deals with polygons of different shapes. The resulting breakdown is shown in Figure 5.1.3.

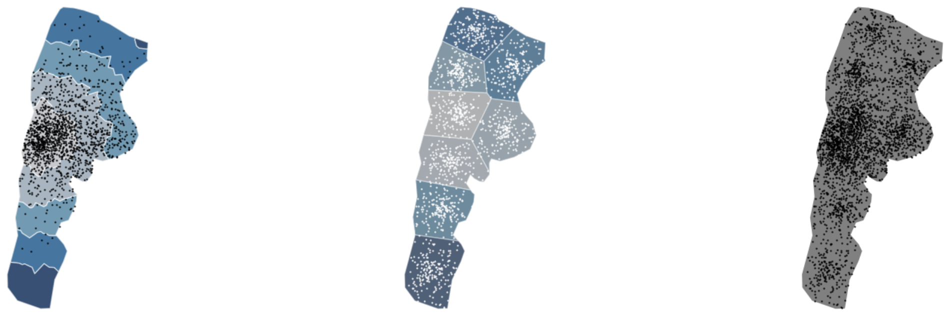


Figure 5.1.3: Scenario 2: Random generation within simplified polygon of Argentina

5.1.4 Scenario 3 - Iceland

This scenario demonstrates the variable-area, area-based proportion stage of secondary generation. A simplified polygon representing Iceland was used for this scenario. The shape of this polygon is generally common, with the added irregularity of the shape towards the north west, demonstrating how the tool manages generation with irregularly shaped regions or more specifically mixtures of convex and concave regions. Again, as before, 3000 points are generated among 8 secondary Voronoi-regions. The `ratio` parameter is set to 0 in order to limit generation to the secondary level only. The resulting breakdown is shown in Figure 5.1.4

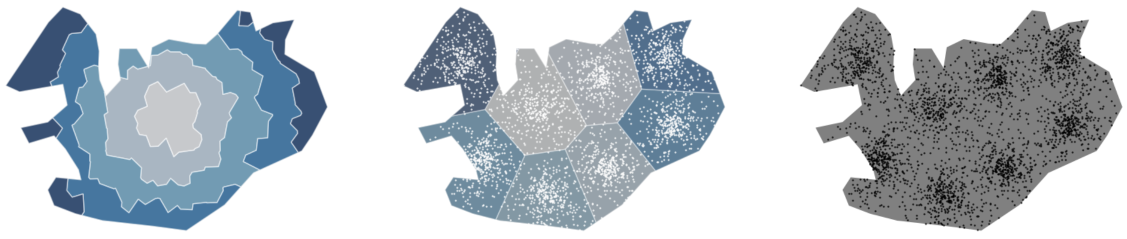
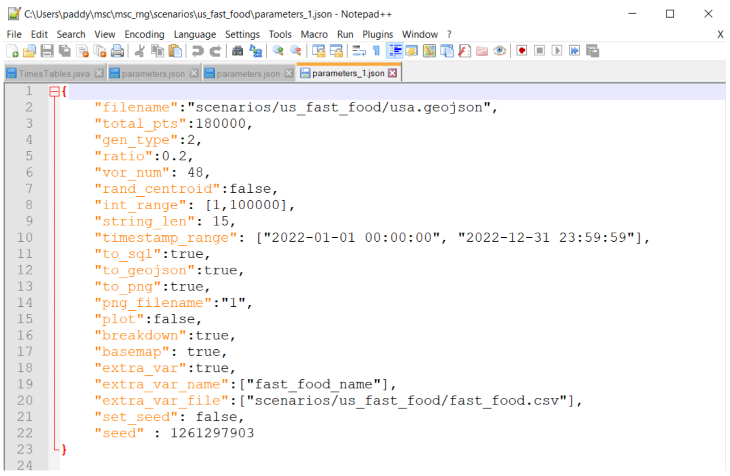


Figure 5.1.4: Scenario 3: Random generation within simplified polygon of Iceland

5.1.5 Scenario 4 - United States Fast Food Restaurants

This scenario serves as what could be considered the limit case of RADIAN and involves a polygon with a significantly larger area than those previously used. The simplified polygon used to represent the continental United States has an area of 7,966,133km^2. This area was chosen for multiple reasons. It has a much larger area than that of the polygons used in the previous scenarios so it is useful to analyze any differences in the behaviour of the software. It is also a region which is instantly recognizable both at a country level and a administrative/state level. This scenario was also used to demonstrate ways in which the additional attributes can be assigned to the datasets to produce more realistic looking data which in turn is then more interesting to the students using the data. In 2020 there were approximately 186,290 Lock (2022) quick-service restaurants in the US. With this figure in mind a dataset of 180,000 points was generated within the US polygon in order to produce a synthetic dataset of fast food restaurant locations. A list of 10,000 fast-food locations in the US was converted to a .CSV file, along with their proportional weights, to allow the points generated by the software to each have a fast-food franchise name assigned to them, with the names appearing in a similar distribution as they would in the real world. Figure 5.1.5 shows the parameters specified for generation. This scenario utilizes 48 secondary Voronoi regions in order to simulate a synthetic version of the state delinations of the 48 states in the continental US. Figure 5.1.6 shows the resulting dataset, generated with a ration of 1:4 ratio of primary to secondary points to simulate points distributions state by state. The resulting dataset demonstrates there is a crossover here between the ability for the current version of the RADIAN tool to generate very large datasets against areas such as the USA where the shape is so easily recognisable to users. The intensive of the circular clustered regions in Figure 5.1.6 is not realistic. To achieve more realistic looking results one would need to consider using less data points or considering more points generated by the primary generation phase.


C:\Users\paddy\msc\msc_rng\scenarios\us_fast_food\parameters_1.json - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
TimesTables.java
parameters.json
parameters.json
parameters_1.json

```
{
        "filename":"scenarios/us_fast_food/usa.geojson",
        "total_pts":180000,
        "gen_type":2,
        "ratio":0.2,
        "vor_num": 48,
        "rand_centroid":false,
        "int_range": [1,100000],
        "string_len": 15,
        "timestamp_range": ["2022-01-01 00:00:00", "2022-12-31 23:59:59"],
        "to_sql":true,
        "to_geojson":true,
        "to_png":true,
        "png_filename":"1",
        "plot":false,
        "breakdown":true,
        "basemap": true,
        "extra_var":true,
        "extra_var_name":["fast_food_name"],
        "extra_var_file":["scenarios/us_fast_food/fast_food.csv"],
        "set_seed": false,
        "seed" : 1261297903
}
```

Figure 5.1.5: Scenario 4: Random generation within a selection of normal geometric shapes

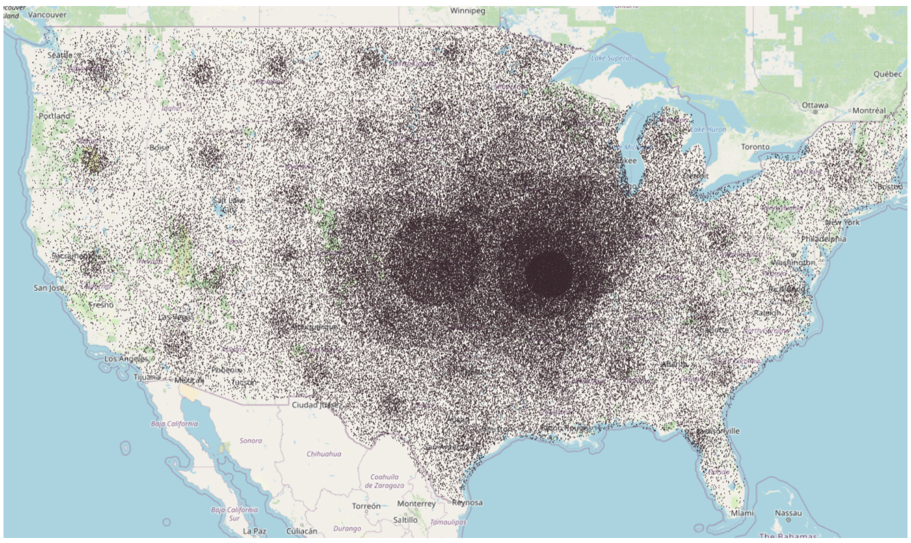


Figure 5.1.6: Scenario 4: QGIS visualisation of the 180,000 point dataset of synthetic fast-food restaurants

5.1.6 Scenario 5 - Tool Comparison

The final scenario serves as a comparison between resulting points generation of RADIAN, QGIS, and Mockaroo. It is important to observe the effectiveness of RADIAN in comparison to these pre-existing tools to measure its effectiveness. The visualizations (produced in QGIS) shown in Figure 5.1.7 demonstrates the distinct differences in the pattern of generation between each piece of software.

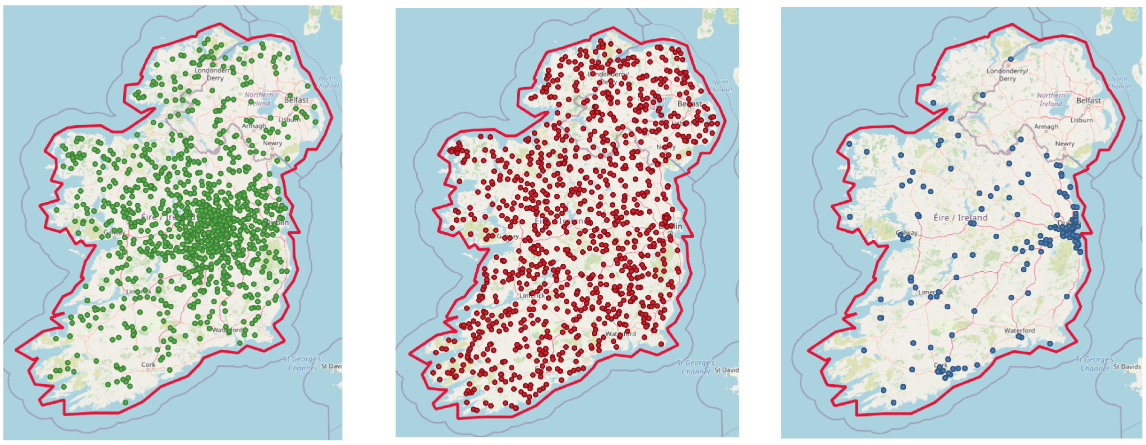


Figure 5.1.7: Scenario 5: 1000 points generated with the project (green), Mockaroo (blue), and QGIS (red)

5.2 Discussion of the scenarios

In this section we briefly discuss the scenarios described above to delivered a general overview of the success of the RADIAN tool.

5.2.1 Scenarios 0 - 4

The above scenarios outline the effectiveness of RADIAN in the creation of realistic random spatial data with a variety of generation patterns and options for additional metadata. RADIAN is able to produce realistic datasets with minimal effort on the part of the user. We believe, for datasets consisting of $<= 10,000$ distinct points, RADIAN does not produce any clear indications of the data being artificial. However, this will require future user testing and surveys to obtain actual qualitative feedback on the generated datasets.

The USA scenario (Section 5.1.5 serves as demonstration of the limits of RADIAN at high values of N, where N is the total number of points to be generated. Figure 5.1.6 displays the generated dataset as viewed in QGIS, in which clearly circular and artificially dense regions of points appear. In the previous scenarios, polygon regions of significantly smaller geographic areas were used. The use of voronoi-based buffers for the primary generation helps in disrupting particularly artificial-looking patterns from emerging in the data. The secondary generation uses circular-based buffers for its generation and has a significant role to play in the problems we see in the USA example.

In the majority of practical scenarios, and in the previous scenarios shown, the secondary regions will be more sparsely populated due to the division of secondary points among the secondary voronoi regions, as well as the overall division between primary and secondary points. In these smaller regions, with lower values for N, the underlying use of circular buffers does not reveal itself in the same manner as shown in the USA scenario. This is due to the overall lower values of N as well as the lower density of points at play as described above. This provides a clear example of future work for the RADIAN software, in allowing for realistic generation at much larger spatial scales. However, as mentioned previously, many of the target problems and examples for RADIAN application in CS621 will be for areas more similar to Greater London than the United States.

5.2.2 Scenario 5 - Tool comparision

We were very interested to create a scenario comparing RADIAN with the two other similar tools. Section 5.1.6 describes this comparison. The points generated using the ”Random points in polygon” function in QGIS have a distinctly uniform distribution with very little evidence of clustering or realistic looking patterns appearing. This is contrasted by the RADIAN generation which utilizes its primary and secondary generation (using 32 secondary voronoi regions) and results in a clustering of points towards a given moving centroid, as well as some small amounts of clustering occurring in the secondary regions. A common facet of the RADIAN and QGIS generation is that the generation is based only around the shape and location of an inputted polygon. Mockaroo on the other hand bases itself on a chosen country/city that is pre-set in its own database. This results in point distribution being much more predictable as the possible range of locations points is more limited than the two polygon-based tools. As stated previously, there appear to be 176 unique possible locations in the Mockaroo database for cities/towns in Ireland, and as such the 1000 generated points are distributed amongst them, resulting in several clashes at each point. While the pattern shown in the visualization displays a apparent realistic distribution, i.e. points are concentrated at the capital in Dublin, with some clusters appearing around the country, the limiting factor of the possible point locations reduces the usability of the tool to generate geographic data for classroom or other situations.

Figure 5.2.1 shows the underlying generation for the RADIAN-produced dataset, displaying the underlying clustering that occurs at the primary and secondary levels. RADIAN clearly displays a strong level of variance while also maintaining a sense of realism in its generated points. In terms of producing realistic looking and customizable datasets for use by students and lectures in assessments, it is clear that the RADIAN software is the most effective tool for this purpose. Future work could carry out user studies to support this claim.

Another distinction between RADIAN and the other tools is the customization of the point metadata. Mockaroo generates realistic metadata rather well however the user is limited to the a list of pre-set variable types, while QGIS provides no means of generating additional metadata for the generated points.

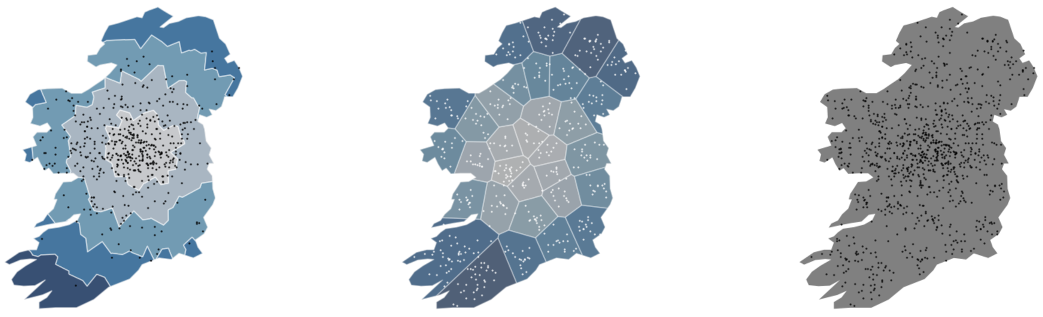
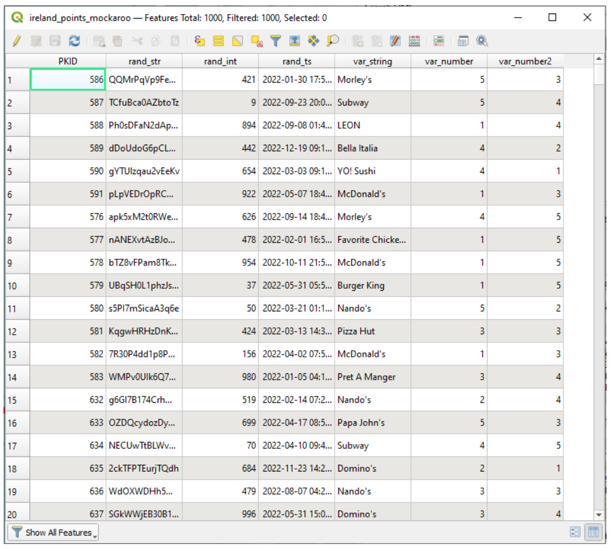


Figure 5.2.1: Scenario 5: A breakdown of the generation process behind the data shown in Fig 5.1.7

Figure 5.2.2 shows an example of the additional data that can be generated by RADIAN, with random integers, strings, and timestamps being generated by default. In addition to this the user can also specify custom columns using user-inputted .CSV files, in this example restaurant names and additional random number values.



ireland_points_mockaroo — Features Total: 1000, Filtered: 1000, Selected: 0

	PKID	rand_str	rand_int	rand_ts	var_string	var_number	var_number2
1	586	QQMrPqVp9Fe...	421	2022-01-30 17:5...	Morley's	5	3
2	587	TCfuBca0AZbtoTz	9	2022-09-23 20:0...	Subway	5	4
3	588	Ph0sDFaN2dAp...	894	2022-09-08 01:4...	LEON	1	4
4	589	dDoUdoG6pCL...	442	2022-12-19 09:1...	Bella Italia	4	2
5	590	gYTUlzqau2vEeKv	654	2022-03-03 09:1...	YO! Sushi	4	1
6	591	pLpVEDrOpRC...	922	2022-05-07 18:4...	McDonald's	1	3
7	576	apk5xM2t0RWe...	626	2022-09-14 18:4...	Morley's	4	5
8	577	nANEXvtAzBJo...	478	2022-02-01 16:5...	Favorite Chicke...	1	5
9	578	bTZ8vFPam8Tk...	954	2022-10-11 21:5...	McDonald's	1	5
10	579	UBqSH0L1phzJs...	37	2022-05-31 05:5...	Burger King	1	5
11	580	s5Pl7mSicaA3q6e	50	2022-03-21 01:1...	Nando's	5	2
12	581	KqgwHRHzDnK...	424	2022-03-13 14:3...	Pizza Hut	3	3
13	582	7R30P4dd1p8P...	156	2022-04-02 07:5...	McDonald's	1	3
14	583	WMPv0UIk6Q7...	980	2022-01-05 04:1...	Pret A Manger	3	4
15	632	g6GI7B174Crh...	519	2022-02-14 07:2...	Nando's	2	4
16	633	OZDQcydozDy...	699	2022-04-17 08:5...	Papa John's	5	3
17	634	NECUwTtBLWv...	70	2022-04-10 09:4...	Subway	4	5
18	635	2ckTFPTEurjTQdh	684	2022-11-23 14:2...	Domino's	2	1
19	636	WdOXWDHh5...	479	2022-08-07 04:2...	Nando's	3	3
20	637	SGkWWjEB30B1...	996	2022-05-31 15:0...	Domino's	3	4

Figure 5.2.2: Scenario 5: A sample of the additional attributes generated by RADIAN

During the running of the above scenarios two variables were created to track the progress of the tool in the generation process. `global_accepted_points` and `global_rejected_points` keep track of the number of successfully generated points and rejected points respectively. A point is considered rejected if, during the point generations process, its coordinates lie outside of both or either one of the main source polygon (or secondary voronoi polygon) or current circular buffer regions, depending on what stage of the generation process is taking place. The ratio between the two variables is known as the `rejection_ratio` and can be considered a measure of the efficiency of the generation process. A generation of 1000 points would typically result in between 5000-7000 rejected points, resulting in a rejection ratio of approximately 0.15 ± 0.01. This value was consistent regardless of the generation parameters, presence or lack of secondary generation, and size of the original source polygon. This means that on average for every generated point that is acceptable, six will be rejected. The runtime-performance of RADIAN is still quite fast, but this rejection ratio shows that there is still room for a great deal of improvement in the generation process. This will be an issue for future work particularly if the RADIAN tool is consider for deployment as a web-based application. Different strategies around point selection will yield different rejection rates. But there will be trade-offs around each point selection strategy. For example, a machine learning based approach may require real-world data for training data before it can generate synthetic points.

Chapter 6

Conclusion and Future Work

In this final chapter we review the work presented in this thesis, consider the impact of the work, and look for opportunities to improve and extend the research into the future.

6.1 Conclusions

The software tool RADIAN produced in this project has successfully fulfilled the specifications outlined in this thesis. In summary, RADIAN is an accessible and efficient software tool that achieves its primary aim - to generate a set of random geographic points for use by students (and lecturers) in assessment in courses requiring spatial data. The tool is able to produce randomly generated, but realistic, geographic datasets, with a variety of options for producing particular spatial patterns in the data. RADIAN allows for multi-phase generation, at either considering the whole polygon or at a more localized scale through the primary and secondary generation stages respectively as described in sections 3.3 (Primary) and 3.4 (Secondary) respectively. In addition to producing geographic points in a realistic manner, RADIAN also provides methods to include additional attribute data, through the use of user-specified .CSV files, allowing for increased customization of the produced datasets. The tool allows for generated datasets to be exported in GeoJSON and SQL dump formats, produce .png plots showing either the final dataset on a basemap image, or a breakdown of the full generation process, between primary and secondary generation. The addition of a seed allows for greater reproducability in the generated datasets, allowing students to replicate exact datasets using the same parameter and seed values. The use of a .JSON parameter file to control configuration of the generation process allows for simplified customization of the resulting datasets. We believe that RADIAN delivers on the methodological approach outlined in Chapter 3 and generates random or synthetic data which can be considered a good workable substitute for real physically derived datasets. The example scenarios described in Chapter 5 provide a good overall selection of the configuration options and outputs from the tool. More examples are provided in the supporting GitHub repository[1] In Table 6.1 we provide a review of how the requirements for the project have been successfully delivered. At the outset RADIAN was conceived as a software tool which supports teaching and learning by generating usable, realistic and interesting spatial data. The thesis has not concentrated on the overall run-time efficiency of the software in no small due to the fact that the software was required as a desktop-based tool and generation times are in the order of seconds for most of the scenarios we have considered. The ultimate success of RADIAN at this stage of its development will be its deployment within the classroom for CS621 in the next delivery of the module.

6.2 Future Work

There are several very interesting extensions to this work which will provide ample opportunities for extensions to the software but also theoretical considerations. In this thesis we have developed an approach for the generation of random points based on our own methodology (see section 3) which uses the combination of a Voronoi polygon approach and distance relationships with polygon centroids to generate patterns of spatial points. One very positive aspect of the work is that there are several exciting directions for future work on RADIAN. The RADIAN tool as presented here in this thesis is an excellent starting point for some more indepth future research beyond the current scope of this thesis.

Generation Methods: There are many potential models which could be used to generate points exhibiting a realistic geographical distribution. Future work could consider the generation of random point datasets along road, rail or water networks corresponding to the input polygon. Using existing real-world datasets in this way is not new. For example, in Chapuis et al. (2018) the authors generate spatially explicit synthetic populations from global (census and GIS) data. Kaur et al. (2020) use openly available heart disease and diabetes datasets as well as the MIMIC-III diagnoses database[2] to generate synthetic health data for machine learning applications. We believe that alternative approaches to spatial data generation could take inspiration from the work on urban design and city architecture which has emerged over the last 100 years and continues to influence urban design and simulation (Wang et al., 2021; Alonso et al., 2018).

Point rejections: RADIAN currently generates output in GeoJSON files and PostgreSQL PostGIS dump files. With some additional work output could be generated in other popular spatial data formats including ESRI Shapefiles, Geopackage (GPKG), Well Known Text (WKT) or KML for use in Google's mapping products. The rejection ratio for the generation process is generally around 0.15, resulting in 6 points being rejected for every accepted point. Future work would include further improvement of the

[1]`https://github.com/paddeaux/msc_rng`
[2]`https://physionet.org/content/mimiciii/1.4/`

Specification or requirement	**Comments**
RADIAN should attempt to generate points in a manner such that they appear as if they were representations of real-world data, rather than randomly generated.	**ACHIEVED**. Using the methodological approach outlined in Chapter 3 with both primary (section 3.3 and secondary 3.4 generation the scenarios outlined in Chapter 5 display how this has been achieved
RADIAN should allow for input of a GeoJSON polygon for the boundary polygon or extent within which points will be generated. This polygon can be any well-formed polygon.	**ACHIEVED**. Chapter 5 describes a number of different types of polygon boundaries. The software code will not allow an illegal or ill-formed GeoJSON file to be processed and will generate an appropriate error message for the user.
RADIAN is intended to be used for the generation of spatial data for many different regions and areas of the world. This tool must be completely independent of any existing datasets in the real world. The tool should operate without any training or linkages to existing datasets.	**ACHIEVED**. The only geographic data input to the tool is the GeoJSON representation of the polygon region. No other linkages are required to geographical datasets. At this stage of the tool's development no training data is required but this is being considered for future work (see section 6.2)
RADIAN should provide functionality for the exporting of generated data in both GeoJSON and PostgreSQL formats.	**ACHIEVED**. Every completed run of the RADIAN software produces both a GeoJSON and PostgreSQL dump file. Both of these files can be used immediately.
RADIAN should allow for the creation of attributes or properties for each of the generated points. By default these should include randomly generated integers, floats, strings, and timestamps. Where appropriate ranges (min, max) values should be specified.	**ACHIEVED**. Three attributes or properties can be specified over a specific range. An integer value can be generated within a specified range. A timestamp value can be generated within a specific timestamp range. Finally, a string of a specific length (containing randomly generated characters) is also included.
RADIAN should specify all coordinates in WSG84 or EPSG:4326 (Latitude Longitude)	**ACHIEVED**. This is an important requirement for the outputs of the RADIAN software. WSG84 or EPSGS:4326 is probably the most popular and well known geographic projection and will be most familiar to students in spatial data oriented courses. RADIAN uses meter-based projections for calculations of buffers, centroid distances and so on. However, the output geometry is transformed to WSG84 or EPSG:4326
The generated points should optionally allow the addition of extra metadata to allow the data to appear more realistic, through the importing of .CSV files containing lists of potential values along with weights to inform the distribution of these values.	**ACHIEVED**. The addition of extra metadata and attributes drawn from real-world lists contributes to making the data appear more realistic. The use of additional metadata is an optional configuration parameter. An example of attributes generated in this way is shown in Figure 5.2.2 in Section 5.2
RADIAN should allow for granular control of the spatial generation, allowing for multiple methods of data generation, to be specified by the user. A default, silent run, mode should also be available	**ACHIEVED**. As outlined in Chapter 4 the RADIAN software is highly parameterised and controlled by a JSON file which allows for the easy configuration of input parameters. With the specification of an input GeoJSON file, a silent run (without any further configuration) is performed by default.

Table 6.1: A review of the main requirements of the project and their successful delivery

generation algorithm, with a focus on reducing the number of attempts necessary to create an acceptable point. This would also improve the overall runtime of the tool. This could be achieved by dynamically adjusting the acceptable bounds within which coordinates can be generated during the various multi-stage buffer-based generation phases. This could help reduce the number of points generated outside of the current buffer/polygon that would inevitably be rejected later on in the generation process. These types of considerations are important if RADIAN was developed and deployed as a web-based tool in the future. Then the overall runtimes would be a more criticial factor particularly as input polygon scale and N grow larger.

Artificial approaches: The scope of this software is primarily for classroom assessments, where relatively small datasets (approx. $< 10,000$ points being generated are sufficient. As demonstrated in the USA Fast Food scenario (section 5.1.5), the realistic behaviour of the tool begins to fail for large values of N, where N is the total number of points to be generated and where the input polygon is a well known or recognisable spatial area. Future work in this regard would be a reworking of the generation process, to maintain the realistic pattern of generation, without the underlying artificial patterns used in generation being brought to the surface. Authors such as Bart et al. (2017) argue that artificially generated datasets have learning values for students. While they do not allow students to discover trends and facts about reality one could use statistical techniques to ensure some relative accuracy of the data. The authors suggest further research must be conducted to see if this is technically feasible and still perceived as authentic by students.

Using additional datasets: In this thesis we have discussion the generation of random point geometries within a given specified polygon. While points are the easiest geometric data type to work with the real-world is composed of points, lines and polygons. More complex generation strategies are required when we begin to consider the random generation of lines and polygons. With lines or polygons one must consider the resolution of the objects - that is how many points are used to define either geometric shape. One must also ensure that generation strategies do not generate illegal geometric objects which break the rules around the mathematical definitions of these objects. For example, polygons must not have self-intersections, should generally be concave, should always be closed, and so on. It may also be necessary that the polygons are adjacent to other polygons, avoid intersections, and so on. With this we see that any random generation of polygons will require the specification of a set of constraints around what will be required of the final randomly generated dataset. A variation on this problem is the random generation of points in relation to their spatial relationships with another dataset. For example, given a dataset representing roads or streets the challenge might be to generate points on or close to the lines. As in the previous discussion this scenario could introduce its own constraints around the placement of these points.

Machine learning and AI: One possible avenue for future extension of this work is to consider approaches where the characteristics or patterns of existing datasets are learned or extracted and then subsequently used to generate synthetic data. In the work of Eno and Thompson (2008b) the authors demonstrate that data mining techniques (in particular, decision trees) can discover patterns that can then be used to build synthetic data sets. These synthetic data sets can be of any size and will faithfully exhibit the same (decision tree) patterns. While generating generating synthetic data on which, for example, machine learning algorithms can be trained and validated, one must be careful not to compromise the privacy of the original dataset (Yoon et al., 2019) especially when using a real-world dataset as a model. This is becoming more prevalent as personal privacy awareness grows around publicly-available datasets. Work already exists in this regard with authors such as Drechsler and Reiter (2011) suggesting that from empirical studies machine learning approaches such as regression trees can result in the production of synthetic datasets that provide reliable estimates and low disclosure risks for census and statistical type datasets. Such et al. (2020) suggest Generative Teaching Networks (GTNs) which may represent a first step toward the ambitious goal of algorithms that generate their own training data.

Bibliography

Alonso, L., Zhang, Y. R., Grignard, A., Noyman, A., Sakai, Y., ElKatsha, M., Doorley, R., and Larson, K. (2018). Cityscope: a data-driven interactive simulation tool for urban design. use case volpe. In *International conference on complex systems*, pages 253–261. Springer.

Anderson, R. C., Bousselot, T., Katz-Buoincontro, J., and Todd, J. (2021). Generating buoyancy in a sea of uncertainty: Teachers creativity and well-being during the covid-19 pandemic. *Frontiers in Psychology*, 11:614774.

Bart, A. C., Whitcomb, R., Kafura, D., Shaffer, C. A., and Tilevich, E. (2017). Computing with corgis: Diverse, real-world datasets for introductory computing. *ACM Inroads*, 8(2):66–72.

Bellovin, S. M., Dutta, P. K., and Reitinger, N. (2019). Privacy and synthetic datasets. *Stan. Tech. L. Rev.*, 22:1.

Burlinson, D., Mehedint, M., Grafer, C., Subramanian, K., Payton, J., Goolkasian, P., Youngblood, M., and Kosara, R. (2016). Bridges: A system to enable creation of engaging data structures assignments with real-world data and visualizations. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, page 18–23, New York, NY, USA. Association for Computing Machinery.

Burns, C. and Chopra, S. (2016). A meta-analysis of the effect of industry engagement on student learning in undergraduate programs. *The Journal of Technology, Management, and Applied Engineering*, 33(1).

Calafiore, A., Palmer, G., Comber, S., Arribas-Bel, D., and Singleton, A. (2021). A geographic data science framework for the functional and contextual analysis of human dynamics within global cities. *Computers, Environment and Urban Systems*, 85:101539.

Calderini, M. and Harding, B. (2019). Grd for r: An intuitive tool for generating random data in r. *The Quantitative Methods for Psychology*, 15:1–11.

Chapuis, K., Taillandier, P., Renaud, M., and Drogoul, A. (2018). Gen*: a generic toolkit to generate spatially explicit synthetic populations. *International Journal of Geographical Information Science*, 32(6):1194–1210.

Dietzel, C., Oguz, H., Hemphill, J. J., Clarke, K. C., and Gazulis, N. (2005). Diffusion and coalescence of the houston metropolitan area: evidence supporting a new urban theory. *Environment and Planning B: Planning and Design*, 32(2):231–246.

Drechsler, J. and Reiter, J. P. (2011). An empirical evaluation of easily implemented, non-parametric methods for generating synthetic datasets. *Computational Statistics Data Analysis*, 55(12):3232–3243.

Eno, J. and Thompson, C. W. (2008a). Generating synthetic data to match data mining patterns. *IEEE Internet Computing*, 12(3):78–82.

Eno, J. and Thompson, C. W. (2008b). Generating synthetic data to match data mining patterns. *IEEE Internet Computing*, 12(3):78–82.

Gold, C. M., Remmele, P. R., and Roos, T. (1996). Voronoi methods in gis. *Advanced School on the Algorithmic Foundations of Geographic Information Systems*, pages 21–35.

Golic, J. D. (2006). New methods for digital generation and postprocessing of random data. *IEEE transactions on computers*, 55(10):1217–1229.

Kaur, D., Sobiesk, M., Patil, S., Liu, J., Bhagat, P., Gupta, A., and Markuzon, N. (2020). Application of Bayesian networks to generate synthetic health data. *Journal of the American Medical Informatics Association*, 28(4):801–811.

Kim, A. Y., Ismay, C., and Chunn, J. (2018). The fivethirtyeight r package:'tame data'principles for introductory statistics and data science courses. *Technology Innovations in Statistics Education*, 11(1).

Krishnan, P. and Jawahar, C. V. (2016). Generating synthetic data for text recognition.

Lan, T., Kandt, J., and Longley, P. (2020). Geographic scales of residential segregation in english cities. *Urban Geography*, 41(1):103–123.

Last, M., Friedman, M., and Kandel, A. (2003). The data mining approach to automated software testing. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, page 388–396, New York, NY, USA. Association for Computing Machinery.

Lock, S. (2022). Number of quick service restaurants in the united states from 2011 to 2020 with a forecast for 2021.

Mannino, M. and Abouzied, A. (2019). Is this real? generating synthetic data that looks real. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, UIST '19, page 549–561, New York, NY, USA. Association for Computing Machinery.

Mebert, L., Barnes, R., Dalley, J., Gawarecki, L., Ghazi-Nezami, F., Shafer, G., Slater, J., and Yezbick, E. (2020). Fostering student engagement through a real-world, collaborative project across disciplines and institutions. *Higher Education Pedagogies*, 5(1):30–51.

Nikolenko, S. I. (2019). Synthetic data for deep learning.

Nong, D. H., Lepczyk, C. A., Miura, T., and Fox, J. M. (2018). Quantifying urban growth patterns in hanoi using landscape expansion modes and time series spatial metrics. *PloS one*, 13(5):e0196940.

Papyshev, G. and Yarime, M. (2021). Exploring city digital twins as policy tools: A task-based approach to generating synthetic data on urban mobility. *Data amp; Policy*, 3:e16.

Rao, A. R. and Dave, R. (2019). Developing hands-on laboratory exercises for teaching stem students the internet-of-things, cloud computing and blockchain applications. In *2019 IEEE Integrated STEM Education Conference (ISEC)*, pages 191–198. IEEE.

Shi, Y., Sun, X., Zhu, X., Li, Y., and Mei, L. (2012). Characterizing growth types and analyzing growth density distribution in response to urban growth patterns in peri-urban areas of lianyungang city. *Landscape and Urban Planning*, 105(4):425–433.

Such, F. P., Rawal, A., Lehman, J., Stanley, K., and Clune, J. (2020). Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. In *International Conference on Machine Learning*, pages 9206–9216. PMLR.

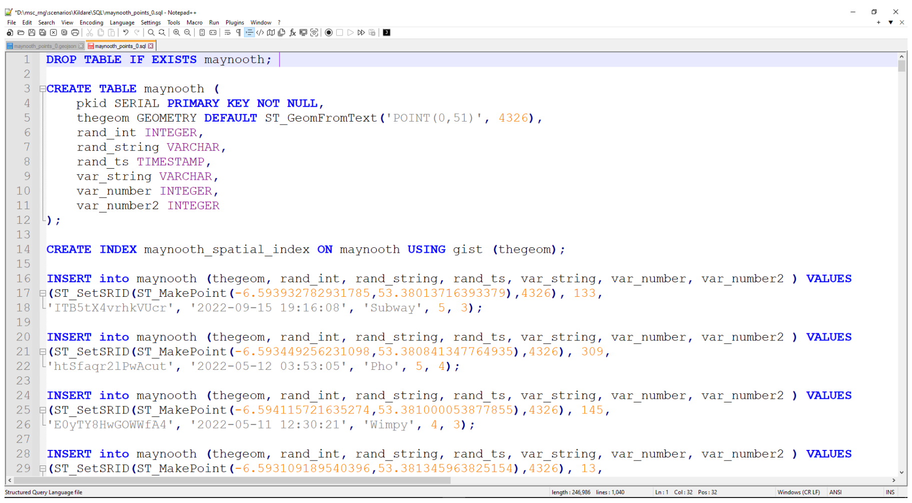
Wang, J., Cao, S.-J., and Yu, C. W. (2021). Development trend and challenges of sustainable urban design in the digital age. *Indoor and Built Environment*, 30(1):3–6.

Yee, O. S., Sagadevan, S., and Malim, N. (2018). Credit card fraud detection using machine learning as data mining technique. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 10(1-4):23–27.

Yoon, J., Jordon, J., and van der Schaar, M. (2019). PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations*.

Appendix A

Appendix 1

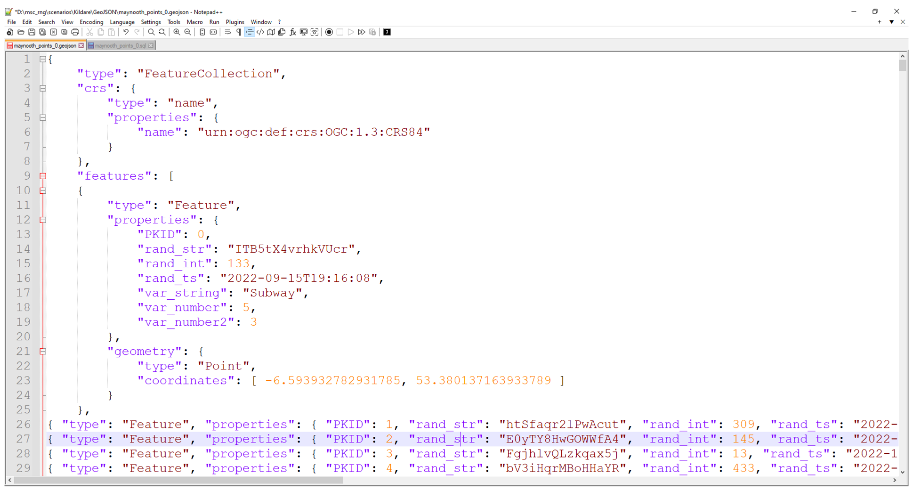


*D:\msc_rng\scenarios\Kildare\SQL\maynooth_points_0.sql - Notepad++

```sql
DROP TABLE IF EXISTS maynooth;

CREATE TABLE maynooth (
    pkid SERIAL PRIMARY KEY NOT NULL,
    thegeom GEOMETRY DEFAULT ST_GeomFromText('POINT(0,51)', 4326),
    rand_int INTEGER,
    rand_string VARCHAR,
    rand_ts TIMESTAMP,
    var_string VARCHAR,
    var_number INTEGER,
    var_number2 INTEGER
);

CREATE INDEX maynooth_spatial_index ON maynooth USING gist (thegeom);

INSERT into maynooth (thegeom, rand_int, rand_string, rand_ts, var_string, var_number, var_number2 ) VALUES
(ST_SetSRID(ST_MakePoint(-6.593932782931785,53.38013716393379),4326), 133,
'ITB5tX4vrhkVUcr', '2022-09-15 19:16:08', 'Subway', 5, 3);

INSERT into maynooth (thegeom, rand_int, rand_string, rand_ts, var_string, var_number, var_number2 ) VALUES
(ST_SetSRID(ST_MakePoint(-6.593449256231098,53.380841347764935),4326), 309,
'htSfaqr2lPwAcut', '2022-05-12 03:53:05', 'Pho', 5, 4);

INSERT into maynooth (thegeom, rand_int, rand_string, rand_ts, var_string, var_number, var_number2 ) VALUES
(ST_SetSRID(ST_MakePoint(-6.594115721635274,53.381000053877855),4326), 145,
'E0yTY8HwGOWWfA4', '2022-05-11 12:30:21', 'Wimpy', 4, 3);

INSERT into maynooth (thegeom, rand_int, rand_string, rand_ts, var_string, var_number, var_number2 ) VALUES
(ST_SetSRID(ST_MakePoint(-6.593109189540396,53.381345963825154),4326), 13,
```

Structured Query Language file   length : 246,986   lines : 1,040   Ln : 1   Col : 32   Pos : 32   Windows (CR LF)   ANSI   INS

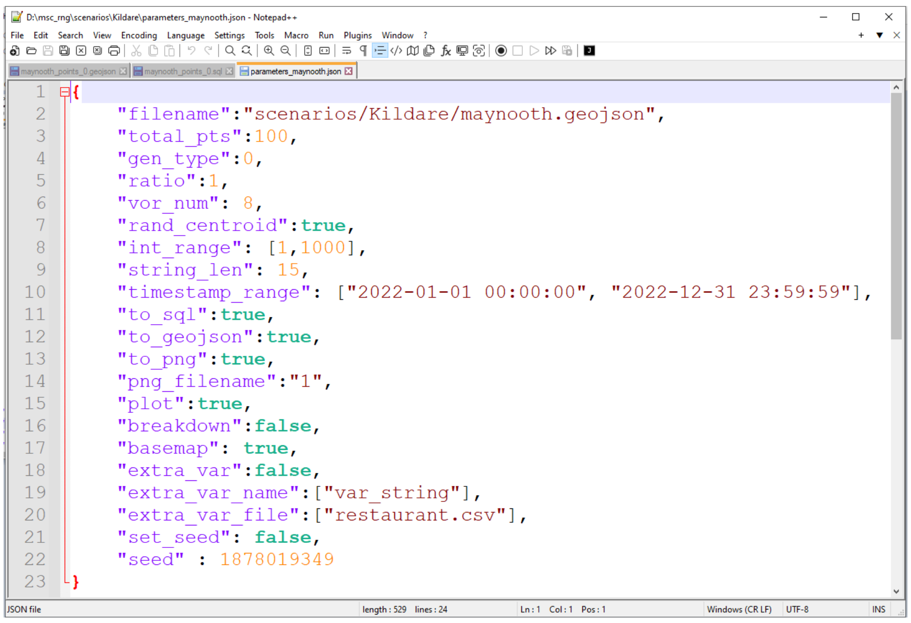
Figure A.0.1: Sample SQL file for 100 points generated within a polygon representing Maynooth



*D:\msc_rng\scenarios\Kildare\GeoJSON\maynooth_points_0.geojson - Notepad++

```json
{
    "type": "FeatureCollection",
    "crs": {
        "type": "name",
        "properties": {
            "name": "urn:ogc:def:crs:OGC:1.3:CRS84"
        }
    },
    "features": [
    {
        "type": "Feature",
        "properties": {
            "PKID": 0,
            "rand_str": "ITB5tX4vrhkVUcr",
            "rand_int": 133,
            "rand_ts": "2022-09-15T19:16:08",
            "var_string": "Subway",
            "var_number": 5,
            "var_number2": 3
        },
        "geometry": {
            "type": "Point",
            "coordinates": [ -6.593932782931785, 53.380137163933789 ]
        }
    },
{ "type": "Feature", "properties": { "PKID": 1, "rand_str": "htSfaqr2lPwAcut", "rand_int": 309, "rand_ts": "2022-
{ "type": "Feature", "properties": { "PKID": 2, "rand_str": "E0yTY8HwGOWWfA4", "rand_int": 145, "rand_ts": "2022-
{ "type": "Feature", "properties": { "PKID": 3, "rand_str": "FgjhlvQLzkqax5j", "rand_int": 13, "rand_ts": "2022-1
{ "type": "Feature", "properties": { "PKID": 4, "rand_str": "bV3iHqrMBoHHaYR", "rand_int": 433, "rand_ts": "2022-
```

Figure A.0.2: Sample GeoJSON file for 100 points generated within a polygon representing Maynooth


D:\msc_rng\scenarios\Kildare\parameters_maynooth.json - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
maynooth_points_0.geojson
maynooth_points_0.sql
parameters_maynooth.json
JSON file
length : 529 lines : 24
Ln : 1 Col : 1 Pos : 1
Windows (CR LF)
UTF-8
INS

```json
{
    "filename":"scenarios/Kildare/maynooth.geojson",
    "total_pts":100,
    "gen_type":0,
    "ratio":1,
    "vor_num": 8,
    "rand_centroid":true,
    "int_range": [1,1000],
    "string_len": 15,
    "timestamp_range": ["2022-01-01 00:00:00", "2022-12-31 23:59:59"],
    "to_sql":true,
    "to_geojson":true,
    "to_png":true,
    "png_filename":"1",
    "plot":true,
    "breakdown":false,
    "basemap": true,
    "extra_var":false,
    "extra_var_name":["var_string"],
    "extra_var_file":["restaurant.csv"],
    "set_seed": false,
    "seed" : 1878019349
}
```

Figure A.0.3: Sample parameter file for 100 points generated within a polygon representing Maynooth

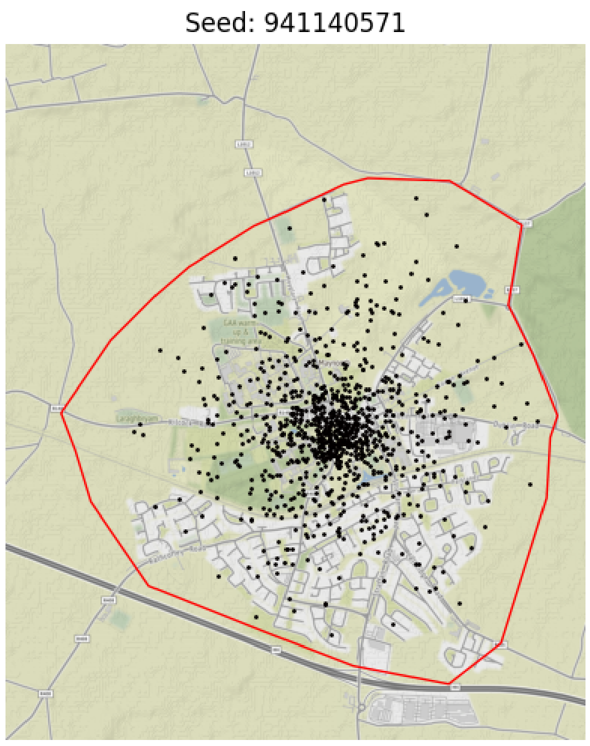


Figure A.0.4: Resulting PNG output of generated points on a basemap