Spend it Locally: A Data Science system to analyze the socio-economic state of Westminster by estimating its daytime population distribution and more

Final Project Report

Anon author

Abstract

Daytime population refers to the number of people in an area during waking hours. Such information is highly valued by governmental and private entities due to its ability to indicate, amongst other things, an area's economic potential.

Previous work has focused on the estimation of daytime population at various spatiotemporal resolutions using census and social media mobility data. This report introduces an alternative method that uses places information as its sole data source, segments the population into six distinct demographic subpopulations and produces results at a very high spatial resolution. The models generated produce estimates for the daytime population demographic distributions and the associated people counts in the borough of Westminster, located in the city of London, United Kingdom.

As part of the Spend it Locally project managed by Westminster City Council. This report describes the construction of a data science system that facilitates the understanding of the region's socio-economic state, with the goal of advising urban planning strategies to entice consumerism and boost the local economy. The system processes and visualizes resident census data, the borough's places catalogue, multiple population estimation models and supply & demand metrics that indicate optimal locations for new businesses.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 15,000 words.

Anon

Acknowledgements

Anon

Contents

List of Figures

Listings

Nomenclature

Metadata: Data containing information about other data, as supposed to actual content relating to the entity the original data represents.

Normalization: A mathematical technique employed to adjust the scale of a collection of values.

Output Area (OA): An authority level used in the UK for census data collection.

place: Refers to an atomic entry in Google Maps Places API responses, usually representing a physical geographical landmark but generally representing any entity containing any of the Google Maps Places place type labels, which may or may not represent a physical geographical landmark. Examples of the latter case include road names, neighbourhoods and seasonal establishments. Also referred to as amenity, with less general terms like establishment or business also being common.

Places: Refers to the datasets derived from the Google Maps Places API. Commonly contains places information aggregated by borough or OA.

Short messaging Service (SMS): A protocol used for sending short messages over wireless phone networks.

Transport for London (TFL): A government agency that manages public transport in London.

User interface (UI): The system component that is designed to be the interactive portal used by the intended end user.

Westminster City Council (WCC): The managers of the Spend it Locally project. Also the local government of the borough of Westminster.

Chapter 1

Introduction

1.1 Project description

Westminster City Council (WCC) launched the initiative Shop Local to help kick start the local economy of the borough post the Covid-19 pandemic's economic stepbacks. This project, titled Spend it Locally, forms part of this initiative. Its goal is to investigate spending behaviours in Westminster and suggest a geo-targeted approach to encourage people to engage with small enterprises, which make up the majority of the local business population. A location-based information repository and visualization system would allow the council to better understand the socio-economic state of the borough and provide a strong evidence base to advise urban planning strategies aimed at closing the gap between retailers and consumers.

This project builds such a system, with the end goal (at the request of WCC) of proposing a passive strategy to entice local consumerism by placing new enterprises in areas of favourable socio-economic circumstances. These are identified by analysing the supply and demand dynamic between certain types of services and certain types of people throughout Westminster, which requires a location-based analysis of the borough's social and business population.

1.2 Project approach

The list of establishments in Westminster is mined from the Google Maps Places API; at the lack of a complete dataset containing the necessary information. The people population is divided into two subsets: Resident (nighttime) and Visitor (daytime). Spend it Locally includes extensive census data about the resident population, so the core task of the project becomes the

estimation of the visitor population. This is done using the aforementioned Places dataset and demographic distribution parameter files that segment the population into six groups based on people's primary reason for visiting Westminster and define the demographic makeup of each place. Supply and demand metrics are then derived using the business and demographically segmented social population.

The project builds a data science pipeline that processes resident and geographical Westminster data, produces various estimates of the borough's daytime population, derives metrics to indicate the optimal positioning of new establishments, and delivers all the results through a web application visualization system featuring interactive choropleth maps and auxiliary visualization devices.

The project presents a novel approach to daytime population estimation not encountered in the publically available literature. This instance of place-based population distribution estimation uses Google Maps Places API data as its sole data source combined with a segmentation of the population into demographics to produce models that accurately estimate, not only a total daytime population, but also all individual demographic subpopulations at very high spatial resolution. The models are versatile and can be adjusted by parameter tuning to illustrate real-life or hypothetical scenarios.

The rest of the report describes the background of the project and the design of the data science pipeline proposed. It reveals, in detail, the processing that generates key datasets and the results they convey. It delves into important implementation and software optimization features, and it explains important information takeaways in the context of Westminster.

Chapter 2

Background

2.1 Urban Planning in context

Urban planning refers to the process of making decisions regarding land use in urban areas for the general good of communities [23]. Population estimation, and in particular demographically segmented population distribution estimation is regarded as key information and is often employed in practical applications, with effort being put towards increasing its accuracy [2, 3, 16, 21, 26]. In this project, urban planning techniques are used to analyse the distribution of amenities & people in Westminster and identify areas of potential demand for certain types of businesses from certain demographics. This will passively encourage consumerism by making desirable places readily available to their main audience.

2.2 Westminster as a case study

The borough of Westminster is a highly diverse cultural and economic driver in London. Containing popular tourists attractions, the largest universities in the city, important financial districts, and the shopping epicentre of London; it is in the interest of the local government and the general public that informed decisions are made to preserve the borough's economic status and continue to improve its suitability to the people that make it run.

Many reports have investigated the socio-economic status of Westminster but none have done so while estimating and considering the borough's daytime population. The 2018 City Profile analyses the age distribution of residents, as well as their religious and political tendencies [19]. The Social Needs study [22] ranks seven key factors impacting the wellbeing of

residents, and an article from the Labour political party [13] paints a picture of the city after the pandemic, covering the return of tourists and the moving of office work from onsite to people's homes. Although these pieces are informative, they would benefit from an urban planning tool that allows them to put their ideas in geographical context, visualise them more effectively and produce much more useful predictive and prescriptive analysis results.

2.3 Population estimation using Places data

Population density estimation analysis has been done before with different source datasets. The paper [3] uses census information with statistical smoothing and [16, 21, 26] use anonymised mobility data or social media data from Twitter in the form of geotweets. In this project, the source for the daytime population distribution estimation is the Google Maps Places API. There are two reasons for this. Firstly, census or mobility data from social media platforms was unavailable and/or lacked the geographical resolution to be usable in the relatively small area of Westminster. Secondly, the Google Maps Places data is extensive and enriched enough to allow for such analysis to be performed. It includes not only registered businesses, but also services, religious centres, landmarks, green spaces, public transport stations and more. It also assigns class labels to entries that provide additional information such as the industry it belongs to and the type of business it is commonly recognised as.

Chapter 3

Related work

This chapter contains a summary and analysis of the literature for the project.

3.1 New business location optimization factors

Out of all the decisions required to be made when opening a business, its location is the most important. Bluntly put in [4] *"a bad location is sometimes impossible to repair"*. The reason is that many factors that characterize a business are influenced by its location. The articles [4, 15, 27] define four aspects applicable to high-street enterprises:

- **Demographic:** Build a target audience profile and ensure its proximity to the business location. Position a business in a community with a large proportion of the defined customer base.
- **Foot Traffic:** Depending on the business operational requirements, position it in an area of high foot traffic levels of its target audience.
- **Parking and Accessibility:** Ensure that parking and accessibility demands are met at all times during the working day.
- **Competition:** Establish the number of nearby competing establishments. Based on the business type and an operational strategy, decide whether a high or low competition environment is more suitable.

These sources identify the tallying of businesses & parking infrastructure, the estimation of population density, and the segmentation of population into demographic types as key factors for determining the optimal location for new businesses. The proposed solution derives and

visualizes such information. It also uses it to compute new enterprise optimal positioning metrics in Westminster.

3.2 Population density metrics

Population density is a broad term that refers to a measure of the number of people by unit area [24]. The definitions of *people* and *unit area* vary depending on what a particular metric is intended to measure. Some examples include *Arithmetic Density* as the total number of people by the total area of land, *Agricultural Density* as the total rural population by the total area of arable land, and *Urban Density*, which refers to the number of people living in an urban area by the total area of urban land. This report later introduces the definition of the density metric used in the project: *population by effective Output Area (OA) polygon area*, where Output Areas (in the UK) are defined as clusters of adjacent postcodes used for the collection of census data [8].

3.3 Daytime population definition

Daytime population is defined verbosely by [2] as the population distribution at times other than when people are expected to be at their residences, extending the implied time definition of business hours to also include evening hours. Consequently, it also defines nighttime population as the population distribution at times not considered to be daytime hours. The paper presents a general definition of people's movements as a function of geographic space and time:

> *daytime population = nighttime population + daytime incoming population - daytime outgoing population*

The paper also defines daytime population as a function of people's movements depending on their demographic type, in this case: workers, students, tourists and business persons:

> *daytime population = workers + school children + tourists + business travelers + residential nighttime residual population*

This definition not only assists in the understanding of its components but also makes it easier to collect the necessary information to perform the calculation as the summands are more specifically defined.

3.4 Daytime population distribution estimation methods

Traditionally, population estimation was based on people counts from census data. With time, the addition of more informative factors like demographic classifications, data source location and socio-economic characteristics have made the analysis of such data more fruitful [2]. The paper [3] describes methods to enhance census data by increasing its spatial and temporal dimension resolution and thus improving the accuracy and versatility of the results. Spatial decomposition of census data through areal-weighting is an interpolation approach that subdivides census polygon areas into a grid and assigns weights to grid cells depending on the proportion of the census polygon contained within them. This is an attempt to decompose census data aggregated by census areas into a more granular and uniform spatial dimension. Temporal decomposition is a technique used to decompose population data into temporal scales of any magnitude. This establishes the distinction between nighttime and daytime population as the geographical displacement from residential to business areas becomes apparent.

Recently, the widespread use of mobile phones and social media has given rise to people mobility datasets. These are derived from personal online interactions that have an attached geographical component. These datasets are also referred to as *social data* and have been used to estimate or improve population distribution estimates. An example is shown in Figure 3.1, where an estimate is built using geotweets [21].

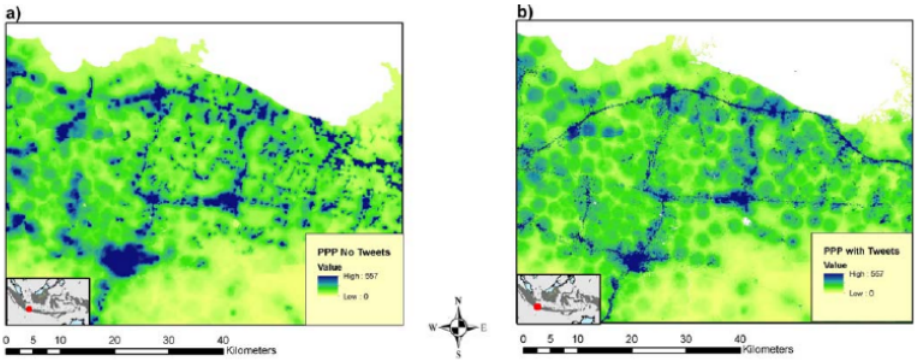


Figure 3.1: Population estimation in the island of Java, Indonesia. No geotweet data (left) vs Geotweet data (right) [21].

In [3] an approach named LandScan is described, that makes use of physical and social data to produce population distribution estimates based on characteristics of human behaviours and landscape changes over time. A component of this method is the *Interpolation and Occupancy Based temporal population estimation* approach, which specialises in analysing diurnal population change for metropolitan areas. It introduces the concept of facility occupancy curves

(Figure 3.2) to represent the number of people occupying facilities over time. This increases the accuracy of the temporal dimension that the population estimation is computed over when physical data (land use) is used as a covariate in the formulation. The transportation simulation-based approach follows the same principle, but focuses on tracking people movements along transportation networks.

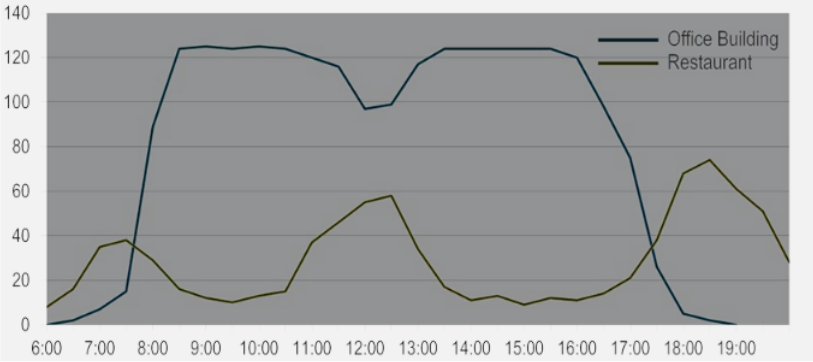
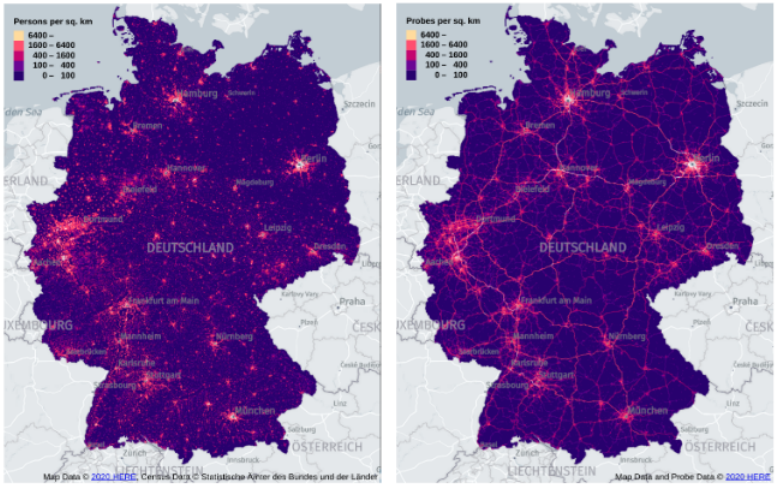


Figure 3.2: Conceptual representation of occupancy curves for two different activity locations or facilities [3].

As the quality of census and mobility data has improved, there have been attempts at extracting information using statistical and modern computational techniques. The authors of [16] use a Bayesian model to combine static population data from census and anonymised human mobility data (in the format of short trajectories) to produce a dynamic spatiotemporal population estimation model based on the time of day and properties of the mobility data. Some of their results are shown in Figure 3.3.



(a) Census **(b) Pseudo-Count**

Figure 3.3: The census population density and pseudo-count density in Germany [16].

A model-based approach is presented in [7] where a statistical model is calibrated to infer the number of people in an area using datasets of SMS activity, internet activity and Twitter geolocated tweets. The study's results (Figure 3.4) show a high correlation between the datasets used, and also the possibility to extrapolate the number of people at certain locations based on temporally relevant mobile phone activity data. Lastly, [26] explores the relationship between geo-referenced tweets and their respective spatiotemporal whereabouts. Semantic topic classification is used to detect human social activities in tweets, and patterns are then compared with census information to identify the significance and reliability of Twitter data. The results show that there is a positive correlation between the datasets regarding workplace-based activities.

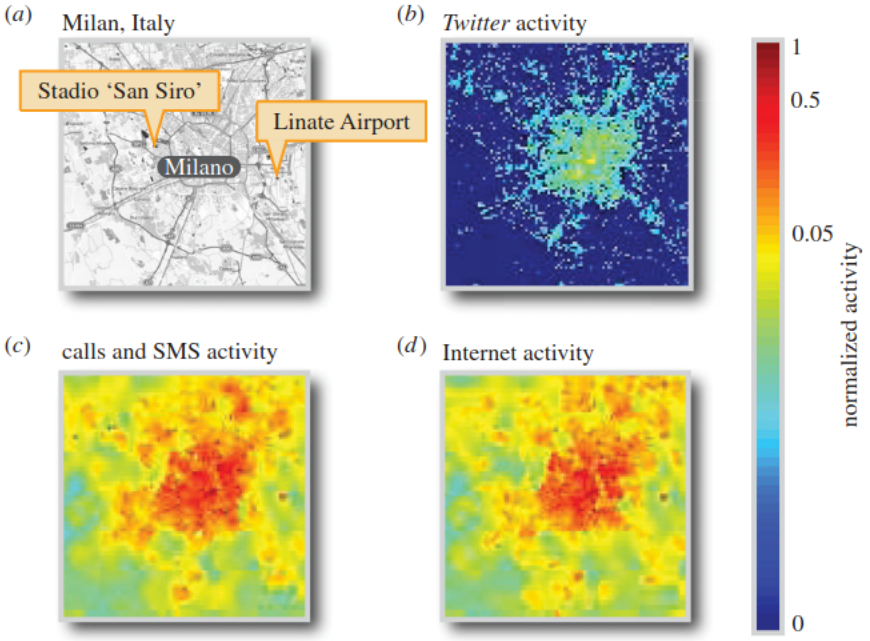


Figure 3.4: Crowd size estimation using SMS, internet and Twitter activity [7].

In summary, population estimation is very useful and sought after data. Census information has historically been used to analyse resident populations, but recently, the importance of daytime population estimates has received more attention. This has resulted in the disaggregation of traditional census data into finer spatial and temporal scales and the addition of social and physical data including: mobility information, land use, transport network positioning and occupancy curves for places. This project focuses on daytime population distribution estimation using enriched land use data inspired by [3] over a segmented population based on the definition of daytime population presented in [2].

Chapter 4

Objective specification, system design and methodology

This chapter outlines the requirements of the project, the system design of the proposed software deliverable and the taken development approach. While here only a general overview of the final structure of the project is described, Chapter 5 details the statistical background behind all data processing decisions, alternative solutions to data science problems, and the reasoning behind the chosen data visualization devices. Design choices are more effectively explained in the context of their implementations.

4.1 Objectives

Having reviewed the background and previous work of the project, the core task can now be reiterated as:

> The estimation of the daytime population distribution in Westminster by OA, where the population is the sum of six well-defined subpopulation distributions, one for each of the demographics: worker, student, tourist, shopper, leisurer and chorer. The estimates are produced as functions of entries in the Places dataset per OA and a model file containing demographic distributions for all place types. All component sub-distributions, the daytime distribution and the total population distribution (daytime & nighttime) should be visualized via an interactive choropleth map of Westminster.

The rest of the tasks include the visualization of the resident's data provided by WCC, the computation of optimal locations for new businesses, and the building of an application to compile, visualize and explore the results of the data analysis. All project goals are listed in this subsection and divided into functional and non-functional requirements.

4.1.1 Functional requirements

The tables below include a high-level listing of the functional requirements of the project. Each is expressed in the context of a software implementation but all tasks contain data science components. Chapters 5, 6 and 7 describe in detail how and to what extent each requirement was sufficed.

Data science

Code	Requirement
F-DS 1	Process and visualize all provided resident datasets
F-DS 2	Collect, process and visualize Places data from Google Maps Places API
F-DS 3	Derive, process and visualize demographic distributions for each demographic
F-DS 4	Derive, process and visualize all relevant population distribution totals
F-DS 5	Derive, process and visualize supply/demand indicator data
F-DS 6	All data visualization instances have a geographical component

User interface

Code	Requirement
F-UI 1	The system is web-based
F-UI 2	Contains visualizations for all dataset-column combinations available
F-UI 3	Geographical visualizations are interactive
F-UI 4	Non-geographical visualizations are interactive
F-UI 5	Geographical visualizations support continuous and categorical legends
F-UI 6	When applicable, presents data in a borough and OA context
F-UI 7	When applicable, visualizations can make use of dynamic scales
F-UI 8	When applicable, dynamic scales can be disabled

4.1.2 Non-functional requirements

Non-functional requirements establish conditions on the level of performance, usability and versatility of the system. The table below sets the project's benchmarks.

Code	Requirement
NF 1	The data processing pipeline is executed fully in under 6 hours
NF 2	The user interface application runs smoothly on a browser on an average machine
NF 3	An attempt has been made to delegate processing away from the web-application
NF 4	An attempt has been made to make the user interface intuitive to use
NF 5	The system can process and visualize new datasets relatively easily
NF 6	The system is suitable for automation and deployment

4.2 Technologies

The data science project is written in Python. This language was chosen for its suitability to data science domains. Two particularly useful libraries during development were Pandas [20] for processing tabular data and GeoPandas [9] for dealing with geographical data. The online tool Mapshaper [17] was used in pre-processing to change the coordinate systems of UK authority geographical datasets. Python was also convenient when mining the Google Maps Places API for the Places dataset as a Google Services Python API [12] is free to use on GitHub.

The data visualization tool chosen is Google Maps. Choropleth maps are built using the Google Maps Javascript API [10] and other non-geographical visualizations use Google Charts [11]. The user interface is a web application written in Javascript. It is designed to be tool-agnostic, consisting of a simple HTML index file and a few JS files to organize the non-UI code. All data visualization is done using Google services but most of the critical functionality is written using the D3 library [6].

4.3 System design

The system is comprised of two projects and four datastores, as shown in Figure 4.1. The Data Mining Project is a Python data processing pipeline that sequentially operates on and moves raw data from the raw data datastore to the focused data datastore, then to the processed data datastore, and finally to the visualizable data datastore. The other project is the User Interface and Data Visualization System. It is a Javascript application that visualizes all the datasets

in the visualizable datastore through a web application. In the current implementation, all datastores are simply designated local directories on the host machine.

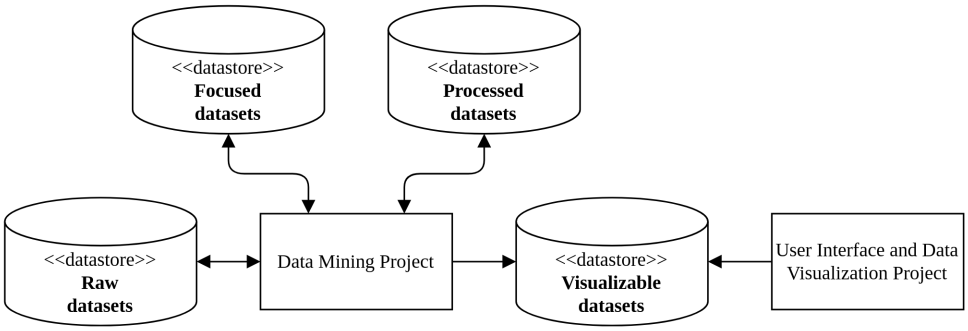


Figure 4.1: System structure diagram.

4.3.1 Data Mining project

The data mining project creates a processing pipeline consisting of four main executable scripts, which in turn execute other scripts that perform operations on individual datasets. This flow of control is shown in Figure 4.2. The raw datastore contains datasets that have been either provided by WCC or collected from online sources. Information about their origins is listed in Appendix B. The one exception to this pattern is the Places dataset, which is also considered raw data and is the result of the Google Maps Places API data mining operation done by the *run_scrape_places.py* script. It produces 783 JSON files (one for each OA) and it is excluded from the automated pipeline due to its running time of approximately 40 minutes and its estimated cost of $ 300.

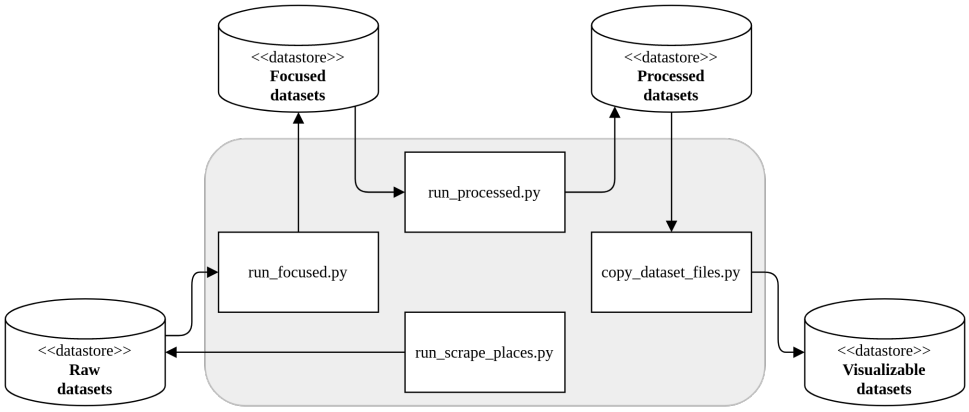


Figure 4.2: Data Mining project structure diagram.

The *run_processed.py* script cleans the raw datasets (both of content and metadata), filters them by Westminster and stores them in the focused datastore. The *run_processed.py* script

then applies processing operations to the focused datasets and stores them in the processed datastore. From here, the *copy_dataset_files.py* script copies those datasets that are chosen to be visualized to the visualizable datastore, ready for the user interface to access them.

The proposed data processing pipeline design has several advantages. It organizes datasets logically depending on the level of processing that has been applied to them. It implicitly implements a checkpoint system at every datastore, where intermediate results are saved before being used in further processing. This not only creates datasets that can be used in a variety of other tasks but also avoids having to re-process intermediate results in the case of a system error. Because of the acyclic nature of the dependencies between datasets, when raw data is updated, only datasets depending on it will be re-processed, thus minimizing overall running times and all associated costs.

4.3.2 User Interface and Data Visualization project

The website design consists of a single page application split into two columns, featuring a Google Maps interactive map of Westminster on the right, and a toolbox section on the left. This structure is shown in Figure 4.3. The left section contains most of the interactive devices of the application, including dataset and column selectors, map control buttons, OA selection indicators and auxiliary (side) visualizations.

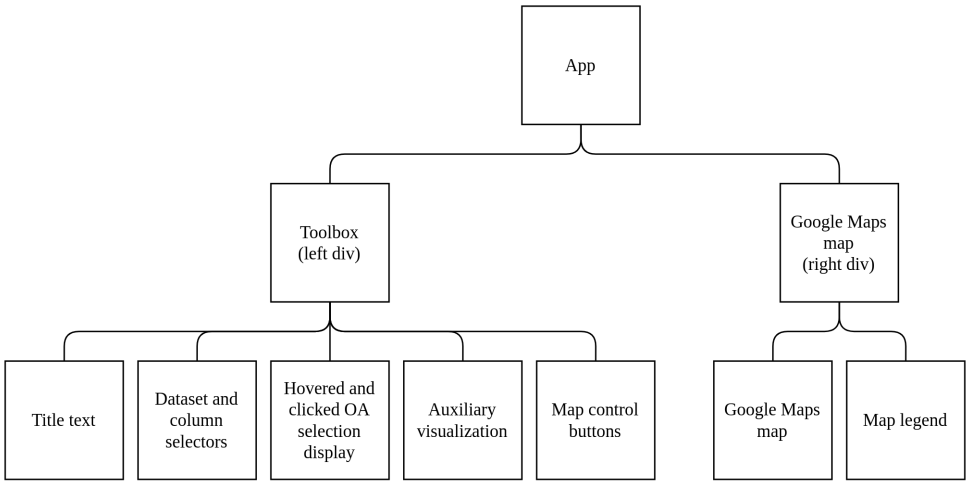


Figure 4.3: User interface project HTML component hierarchy diagram.

The flow of control between the Javascript files is shown in Figure 4.4. *index.js* initializes the application's components and is responsible for event handling. The *state.js* file holds the current state of the application via a collection of global variables. The files *generateScales.js* and *load_and_process_data.js* generate a dataset-column-scale and an OA-dataset-column map-

ping respectively. These data structures are used to index information and significantly reduce lookup times, thus allowing the web application to handle over 400 columns loaded as static resources and access all queryable information in constant time.

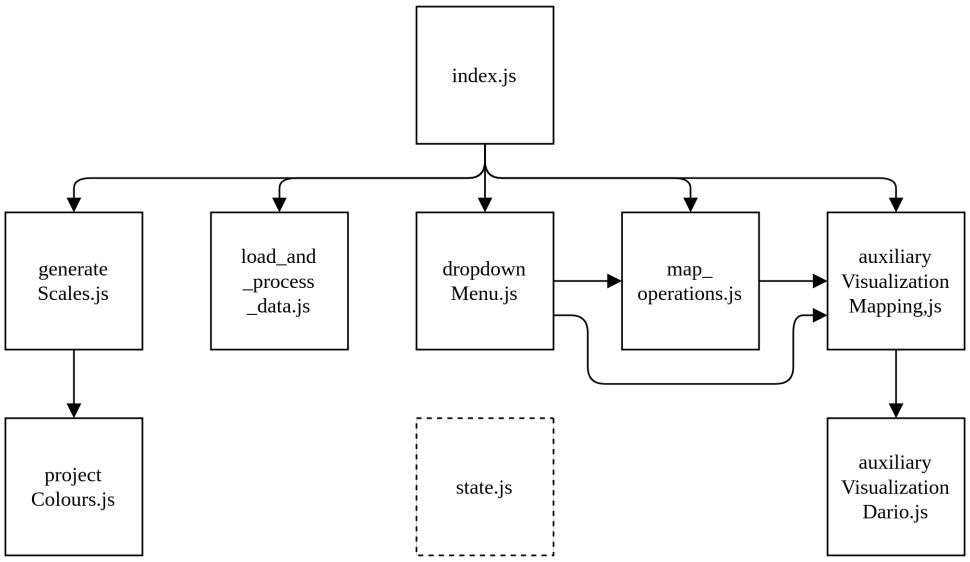


Figure 4.4: User interface project Javascript files usability diagram.

4.3.3 Integration and optimization strategy

By design, the system is made to adapt to new datasets without affecting performance. This is achieved in three main ways: Firstly, a metadata file is created in the data mining project that holds information about each column of all datasets. This includes the column name, type, range and other identifying information. All datasets (previously seen or not) found in a specified location will be included. With access to this file, the website does not have to process all 400+ columns during loading, hence increasing performance. Secondly, the metadata file is used in the creation of scales for all columns. Legend creation usually involves obtaining the type and range of a column, which is done by iterating through all the values. In this case, all the required information is included in the loaded metadata file and accessing it is constant time. Lastly, a dataset-column-OA-auxiliary visualization mapping is used to assign side visualizations to all combinations of datasets and columns. Although default visualizations for all cases exist, adding bespoke ones for new datasets simply require adding an entry to the mapping.

4.4 Methodology

In this project the development of the data processing pipeline, the user interface and the visualization system was done simultaneously. All three happened to be part of the data science pipeline created and their designs were heavily influenced by results of previous iterations. A usual development cycle would be as follows:

1. Create or update datasets in the data processing project.
2. Visualize the results in the user interface through the visualization system.
3. Check that datasets are processed adequately.
4. Check that datasets are visualized adequately.
5. Check that the interactivity suffices the information use case.
6. Make changes to any of the three aforementioned components.
7. Repeat from 1.

The next two chapters explain the *whys* and *hows* to the *whats* presented in this chapter, including implementation details and approach evaluations.

Chapter 5

Data Mining Design and Implementation

This chapter goes over the creation of all datasets in the project, explaining in detail the reasoning behind the statistical techniques used, their implementation and the resulting data visualizations. Figure 5.1 shows the dependency graph for the focused datasets, indicating how they are generated from the original raw datasets.

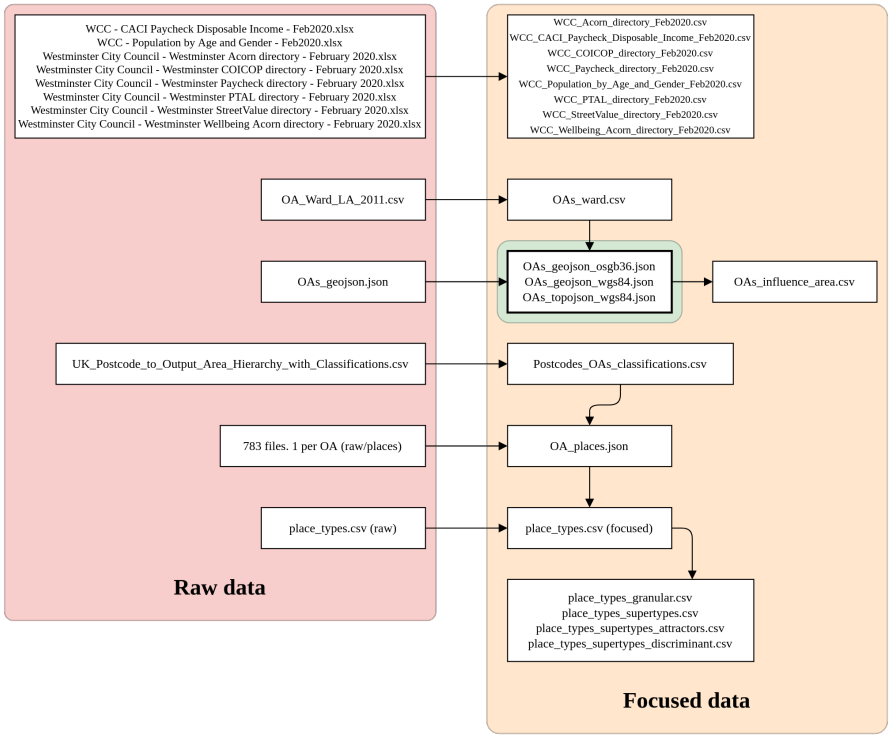


Figure 5.1: Data Mining project dataset dependency graph (raw to focused).

Figure 5.2 shows the corresponding information for the processed datasets. The orange regions are equivalent in both diagrams. Prefixes in the datasets' names help identify their content:

- [**OA**]: Non-resident information about OAs.
- [**Residents**]: Resident information about OAs.
- [**Places**]: Any derivation of the number of places by OA.
- [**POC_Demographic_distribution**]: Explainable demographic distribution datasets.
- [**Demographic_distribution**]: Final demographic distribution models.
- [**Population**]: Nighttime, daytime and total population distributions.
- [**Supply_Demand**]: Supply and demand dynamic metrics.

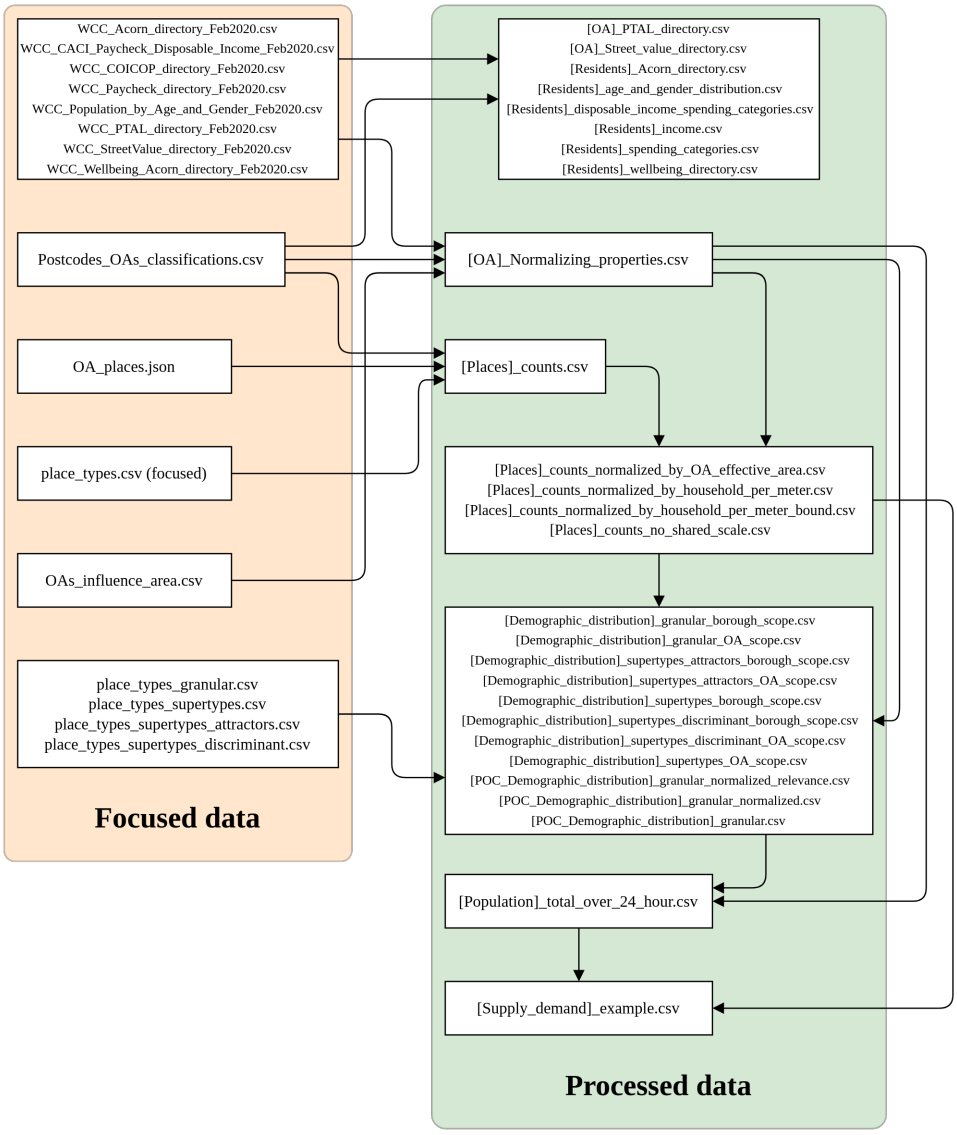


Figure 5.2: Data Mining project dataset dependency graph (focused to processed).

5.1 Authority geographical data

This section produces the authority boundary datasets that make possible the rendering of the Westminster map subdivided by OAs shown in Figure 5.3. The processing pipeline is as follows:

1. Prepare a list of all OAs in Westminster by filtering the *OA_Ward_LA_2011.csv* dataset by the borough name.
2. Filter the polygons in the raw geographical boundaries file *OAs_geojson.json* by Westminster OAs.
3. Change the coordinate system of the new GEOJSON file from Ordnance Survey Notation or British National Grid (osgb36), which is in meters, to World Geodetic System (wgs84) which is in degrees. This is done using Mapshapper. Save a version in each coordinate system.
4. Convert the wgs84 file from GEOJSON to the TOPOJSON format. This is a much more storage efficient method for storing geographical data with topological components.
5. Visualize the TOPOJSON file in the user interface.

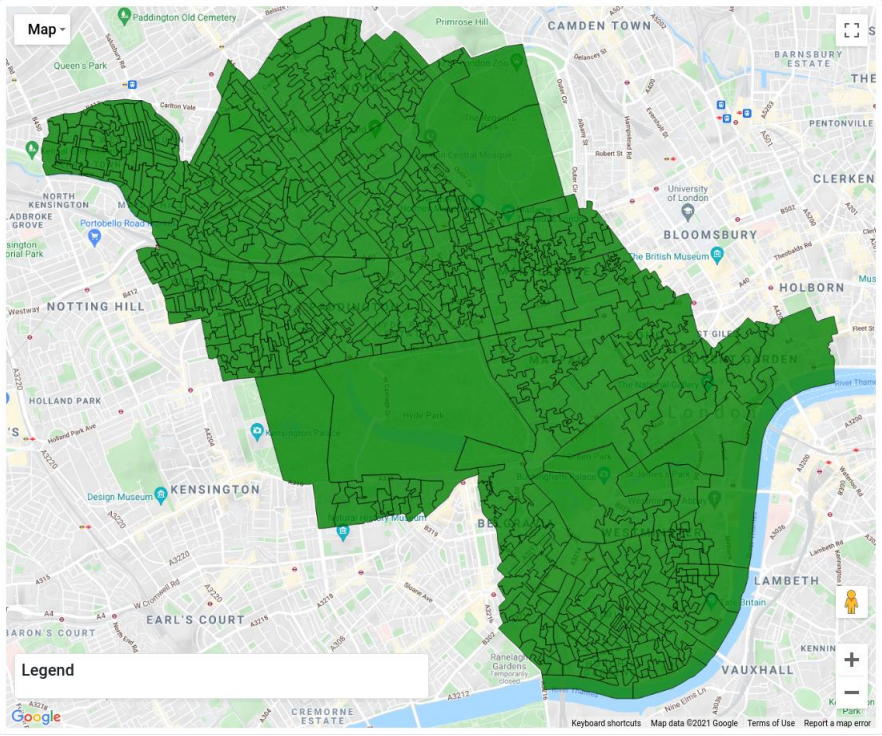


Figure 5.3: Westminster Output Areas (OA) visualized on Google Maps.

The osgb36 and wgs84 versions of the file are then used for constructing the *OAs_influence_area.csv* dataset. It contains numeric metadata about the polygons associated with the OAs in both

meters and degrees. This includes polygon area, polygon area centroid, polygon bounding coordinates, maximum distance from the area centroid to any of the bounding coordinates, etc. These columns are used in a variety of operations later on in the pipeline.

5.2 CACI tabular datasets

This section relates to the tabular datasets from the data provider company CACI [5]. They paint a picture of residents in Westminster by covering a range of social, demographic and economic factors. These datasets are originally derived from UK census data. The first step in their processing is to convert them from XLSX to CSV files. This was done manually using LibreOffice Calc's export tool with default settings. Most datasets are additionally joined with the *Postcodes_OAs_classifications.csv* file which assigns an OA to each postcode record. This is shown in Listing 5.1.

```
    # Load and rename the postcode-OA mapping dataset.
    postcode_oa_df = pd.read_csv(DATA_DIR + "focused_data/authorities/" + "
     Postcodes_OAs_classifications.csv")
    postcode_oa_df = postcode_oa_df[["pcd7","oa11cd"]].rename(columns={"pcd7": "Postcode", "
     oa11cd": "OA"})
    ...
    # Join dataframes on postcode column.
    joined_df = postcode_oa_df.merge(ptal_directory_df, on = "Postcode")
    ...
```

Listing 5.1: Adding the postcode-OA mapping to a dataset.

5.2.1 PTAL (public transport accessibility) directory

This dataset (originally from TFL) assigns a public transport accessibility index (value) and class label to each postcode in Westminster. Its processing involves grouping the postcodes into OAs and applying a mean operation to the continuous column and a mode operation to the categorical column, as shown in Listing 5.2.

```
    # Dataframe groupby operation to return a frequency count of a series group.
    def get_frequencies(x):
        return x.value_counts().to_dict()

    # Return the mode of a frequency count.
    def convert_frequency_count_to_mode(x):
        if len(x.keys()) <= 0:
            return null
        else:
            return max(x, key = x.get)
    ...
    # Group by OA.
    AGG_DICT = {
        "Public Transport Accessibility Index": np.nanmean,
        "Public Transport Accessibility Level": get_frequencies
```

```
    }
    ...
    grouped_df = joined_df.groupby("OA").agg(AGG_DICT).reset_index()
    ...
    for k in FREQUENCY_COLS:
        grouped_df[k] = grouped_df[k].apply(lambda x: convert_frequency_count_to_mode(x))
    ...
```

Listing 5.2: OA PTAL directory.csv aggregation operation per column.

The visualized results are shown in Figure 5.4. Both choropleth maps show the same pattern, and other non-mean operations applied to the categorical column produce a very similar outcome. The results indicate that metropolitan areas are more accessible than the surrounding ones. OAs with higher PTAL index values follow the distribution of underground and bus stations in the borough, which are more common in central London.

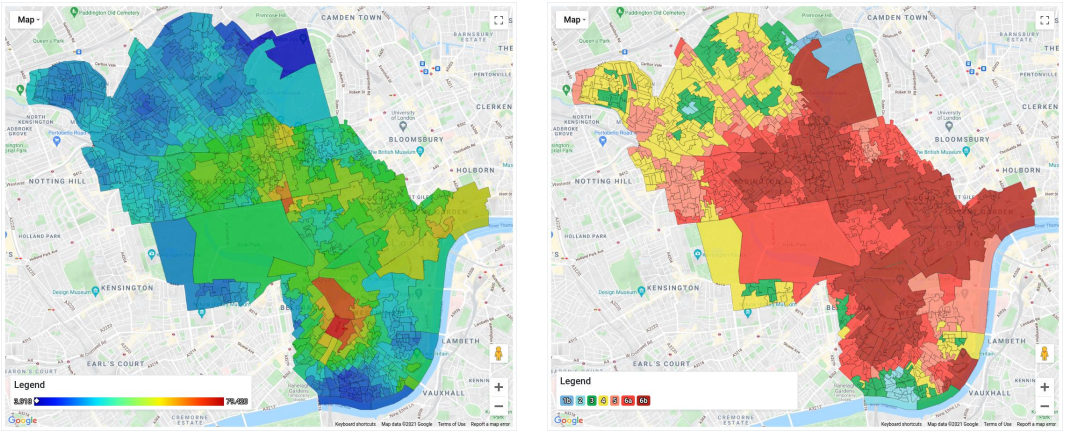


Figure 5.4: PTAL value (left) and class (right) by OA.

5.2.2 Street value directory

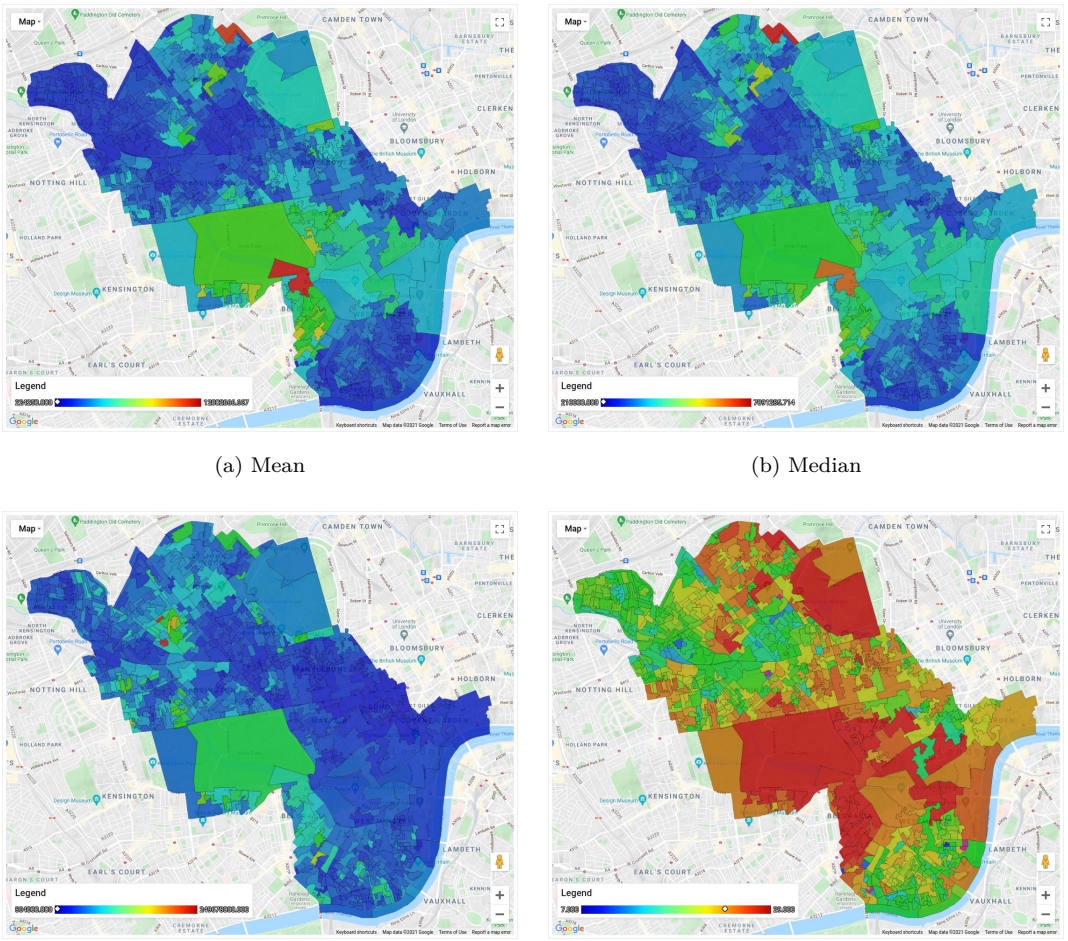
The street value dataset calculates the value of postcodes in Westminster given the worth of the houses (and consequently streets) in the area. The processing is similar to the PTAL dataset but this time the mean, median and sum operations are applied to the street value column. The aggregation assignment function is shown in Listing 5.3.

```
    # Assign aggregation operation based on column type.
    def generate_aggregation_map_street(column_types, string_cols, integer_cols,
     categorical_cols):
        dict = {}
        for i in range(len(column_types.index)):
            c = column_types.index[i]
            if c in categorical_cols:
                dict[column_types.index[i]] = pd.Series.mode
            elif c in integer_cols:
                dict[column_types.index[i]] = np.sum
            elif c in string_cols:
```

```
            pass
        else:
            dict[column_types.index[i]] = np.nanmean
    return dict
...
```

Listing 5.3: Street Value directory aggregation operation per column.

The outcome of the processing is shown in Figure 5.5. In this case, the mean and median operations are good indicators for the value bands assigned to the OAs (inversely related) but the sum case does not perform as well. The results show a clear distinction between residential and non-residential areas, with street values of residential areas being dramatically lower. Interestingly, some green spaces in the borough, like Hyde Park, tend to be higher valued than entertainment hubs such as Soho.



(a) Mean

(b) Median

(c) Sum

(d) Band indicator

Figure 5.5: Street Value directory dataset columns.

5.2.3 Residents Acorn directory

The Acorn [1] dataset is a consumer segmentation tool that segments the UK population by postcode using demographic, social population and consumer information. There are 59 Type labels available and these are arranged into Groups and later Categories. It is a great tool for determining what kind of people live in an area.

Processing is similar to the PTAL dataset but without the presence of numerical columns. Frequency counts are generated for the group and category columns, the mode of each is taken, and the dataset is cleaned before being saved. The latter operation is shown in Listing 5.4.

```
# Rename columns.
grouped_df = grouped_df.rename(columns={"Acorn Category": "Acorn_category", "Acorn Group": "
 Acorn_group"})
# Save dataframe to CSV.
common.save_dataframe_to_csv(DATA_DIR + "processed_data/acorn/", grouped_df, "[Residents]
 _Acorn_directory.csv")
...
```

Listing 5.4: Acorn dataset renaming columns and saving.

The visualized columns are shown in Figure 5.6. The maps showcase how undefined values are dealt with by the default *undefined* class bounded the colour black. The results are very similar, suggesting most OA categories are dominated by a single group. The classes *Not Private Households* and *Rising Prosperity* dominate most of the borough, with a tendency towards more central areas. *Urban Adversity* and *Affluent Achievers* are common in more residential areas.

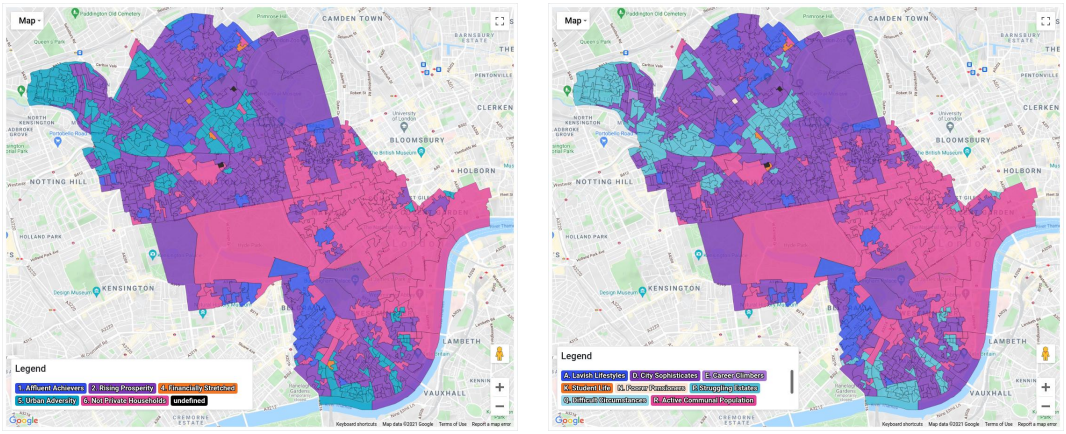


Figure 5.6: Acorn directory category (left) and group (right) by OA.

5.2.4 Residents wellbeing directory

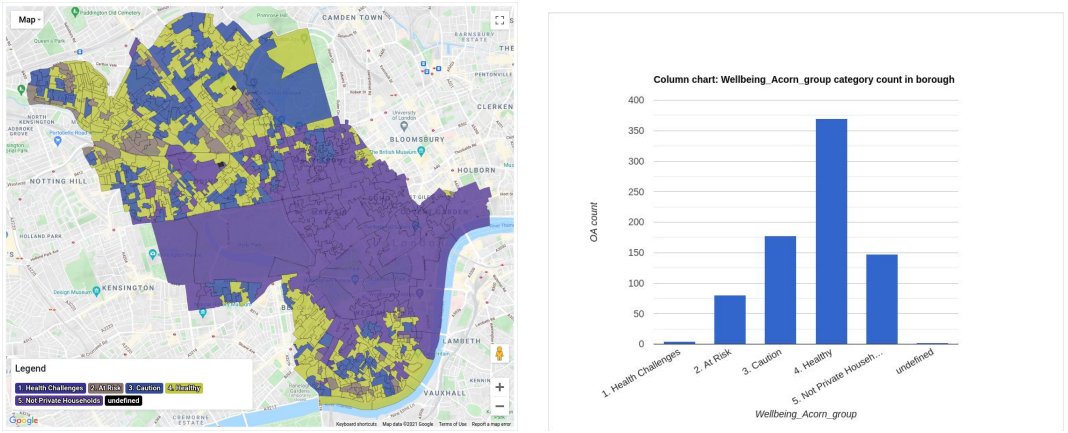
The Acorn Wellbeing directory is similar to the Acorn directory but considering only health and wellbeing issues. Its processing is virtually the same, but due to a lack of classification

labels, the class index to class labels mapping shown in Listing 5.5 is used. The class labels (along with their colours) are sourced from the Acorn Wellbeing directory User Guide.

```
# Map dataset column values to corresponding verbose labels.
ACORN_WELLBEING_GROUP_MAP = {
    1.0: "1. Health Challenges",
    2.0: "2. At Risk",
    3.0: "3. Caution",
    4.0: "4. Healthy",
    5.0: "5. Not Private Households"
}
...
```

Listing 5.5: Acorn Wellbeing dataset group class index to group class label mapping.

The visualized results in Figure 5.7 show a similar trend to the Acorn spread. The *Not Private Households* class dominates most of the borough centre, indicating resident population in this area is very low. The auxiliary visualization group count histogram shows that residential areas are predominantly *Healthy*.



(a) Group by OA

(b) Group count in borough

Figure 5.7: Wellbeing directory group breakdown.

5.2.5 Residents spending categories

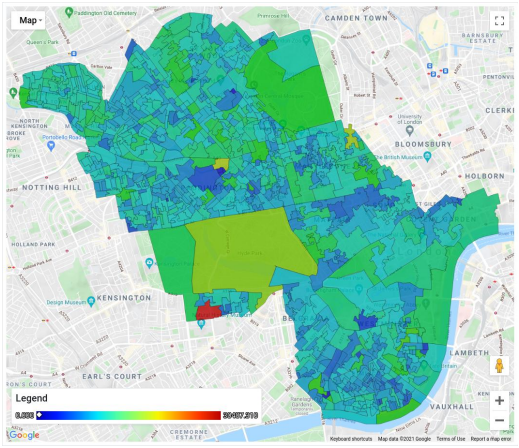
This dataset segments and quantifies resident spending categories. Being the only dataset already grouped by OA, its processing included cleansing column names and generating the *average* and *shared_scale* columns, as shown in Listing 5.6.

```
# Create average columns.
for col in COLS:
    coicop_directory_df["[average_per_person_in_OA] - " + col] = coicop_directory_df[col] /
 coicop_directory_df["Total Population 2019"]
# Create shared scale columns.
for col in COLS:
    coicop_directory_df["[shared_scale] - [average_per_person_in_OA] - " + col] =
 coicop_directory_df["[average_per_person_in_OA] - " + col]
```

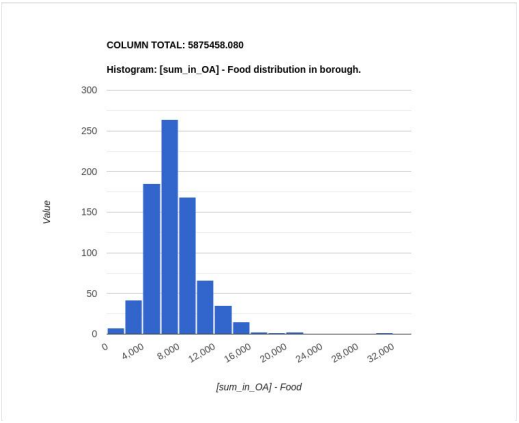
```
7    ...
```

Listing 5.6: Resident spending categories new columns.

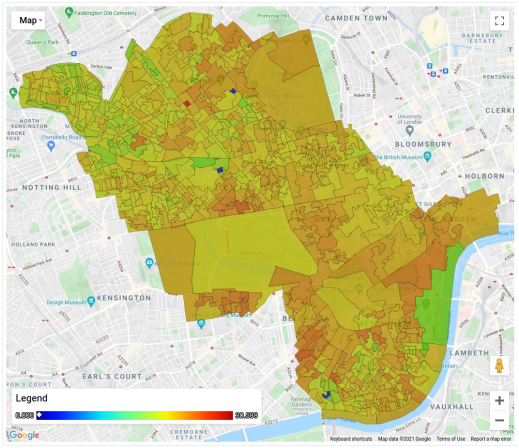
The *average* columns turn the original *category spending total sum by OA* into *average spending values per person (per week)* by dividing each record by the total population of the OA. This results in a more indicative and useful choropleth map visualization, shown in Figure 5.8. The Food by person in OA average histogram shows a narrow distribution around the range 20 to 28, fitting in nicely with the UK's national average of food spending per person per week of £ 26.50 [28].



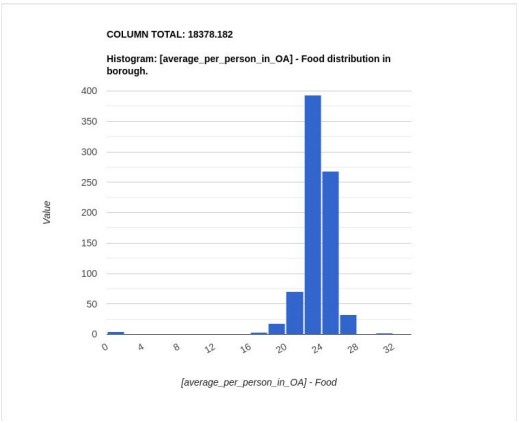
(a) Food by OA sum



(b) Food by OA sum distribution



(c) Food by person in OA average



(d) Food by person in OA average distribution

Figure 5.8: Residents spending categories food OA total vs food per person.

The *shared_scale* columns are an added feature aimed at visualizing multiple choropleth maps on the same scale/range/scope. Although the side visualization histograms already serve this purpose, maps can offer a better comparative view. These columns have their scale adjusted

to the maximum and minimum value of any shared column in the dataset, instead of the individual column being visualized. This example is shown in Figure 5.9.

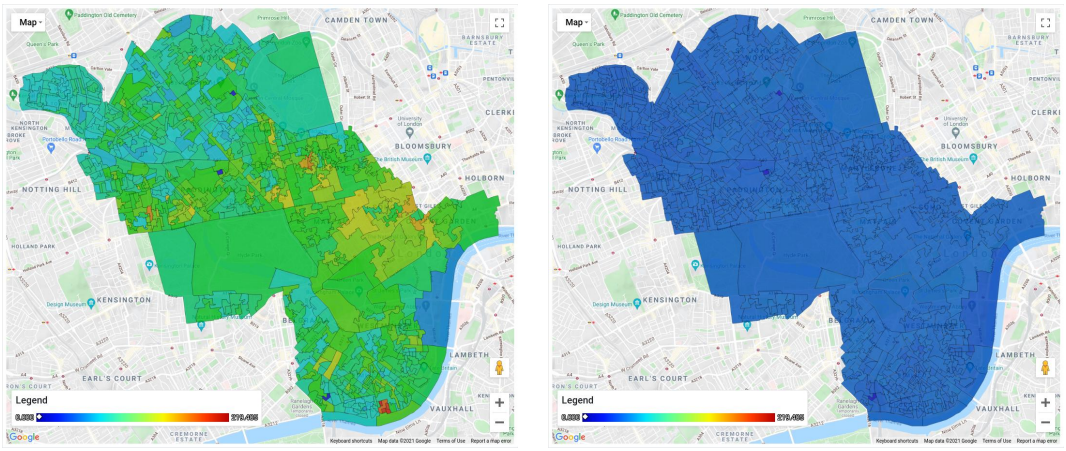


Figure 5.9: Resident spending categories shared scale columns rent (left) vs food (right).

5.2.6 Residents disposable income spending categories

This dataset is similar to the previous, showing CACI defined spending categories over households and relevant households. Some interesting results are presented in Figure 5.10 which suggest that OAs that spend heavily on social rent, do not spend heavily on structure insurance (and vice versa), however, there are OAs that break this rule in the boundaries between areas that follow the pattern.

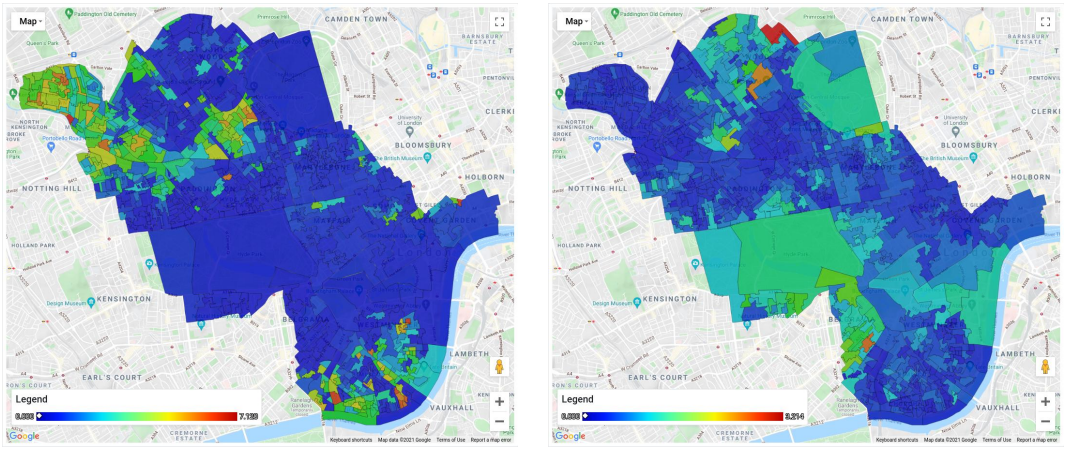


Figure 5.10: Residents disposable income spending categories social rent(left) vs structure insurance (right).

5.2.7 Residents income

The resident income dataset counts the number of residents per income band per postcode. These extend from 0-5K to 200K+ in increments of 5 or 10K (with 1K equating to £ 1000). Processing is similar to previous datasets but in this case, the *income band count per OA* column is used to create a *percentage income band count per household in OA* column. This operation is shown in Listing 5.7.

```
    # Normalize by households.
    for col in COLS:
        grouped_df["[%_in_OA] - " + col] = (grouped_df[col] / grouped_df["Total Households"]) *
     100
    ...
```

Listing 5.7: Income band by households (percentage).

Expressing the results as percentages does not change the appearance of the choropleth map but it makes the values more explainable. The resulting visualization, shown in Figure 5.11 presents a map that is more significant than the original band count map. The undesirable effect of larger OAs getting higher counts is counteracted by the normalization by the total number of households in each OA, which is a factor in the area definition of OA polygons (census areas). The normalized map shows that the relatively low-income band of 25-30K is more common in residential areas and less common in central (more expensive) parts. This follows all the other socio-economic factors in Westminster analysed so far (and still to come).

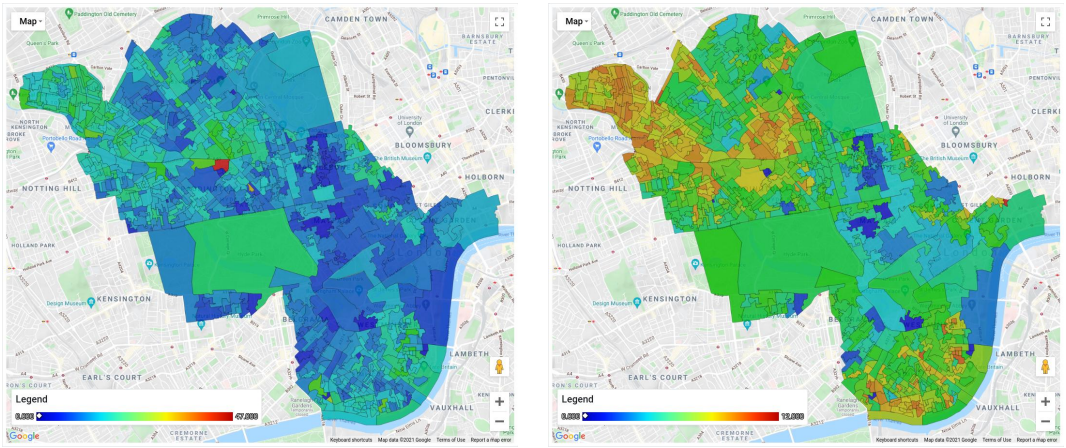
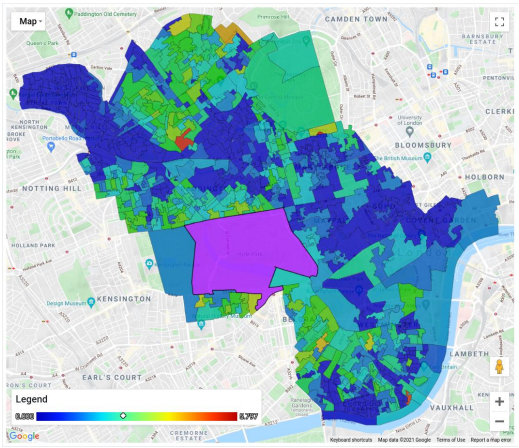


Figure 5.11: Resident income 25-30K band count (left) vs normalized by households (percentage) (right).

For this dataset-column-OA combination, side visualizations show the band distribution in the selected OA. An example of this is displayed in Figure 5.12. The results for OA E00175205 show a bimodal distribution with modes around the 40-50K and 95-140K bands respectively.



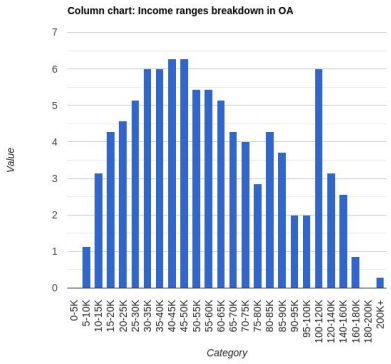


Figure 5.12: Resident income bands normalized by households (percentage) (right) and the associated column chart (left).

5.2.8 Residents Age and gender distribution

This dataset splits the population in each postcode by gender and age ranges from 0 to 65+ years old, mostly in four-year intervals. Processing, in this case, starts by defining a different grouping of age ranges, based on the education tier and job progression of people (detailed in Listing 5.8). Although the accuracy of the resulting columns decreases due to this aggregation operation, they are a lot more useful in analysing the distribution of the population based on criteria more relevant to real-world uses.

```
# Age range grouping redefinition.
AGE_GROUPS = {
    "Infant [0-4]": ["0-4"],
    "Primary student [5-9]": ["5-9"],
    "Secondary student [10-15]": ["10-14", "15"],
    "College student [16-17]" : ["16-17"],
    "Universitarian / apprentice [18-24]" : ["18-19", "20-24"],
    "Young adult [25-39]" : ["25-29", "30-34", "35-39"],
    "Middle-aged adult [40-49]": ["40-44", "45-49"],
    "Senior adult [50-64]": ["50-54","55-59","60-64"],
    "Senior [65+]" : ["65+"]
}
...
```

Listing 5.8: Age and Gender distribution age range segmentation.

Furthermore, the data is also split into three gender classes in: *Females*, *Males* and *Total (females + males)*. A high level snippet of this data structure is shown in Listing 5.9.

```
# Stores value tallies separated by gender and age.
AGE_GROUPED_COLUMNS = {
    "Females": {...},
    "Males": {...},
    "Total": {...}
}
```

```
...
```

Listing 5.9: Age and Gender distribution gender segmentation.

Listing 5.10 provides the remaining operations applied to the dataset. Counts for the new age and gender segmented columns are grouped using the original age and gender ranges, they are summed to create the *[count]* prefixed columns and finally converted from counts to the percentages that particular age and gender groups make up of the total OA population (listing line 27).

```
# Group and filter columns. Populate the AGE_GROUPED_COLUMNS map.
for g in ["Females", "Males"]:
    for c in grouped_df.columns:
        split = c.split(" ")
        if len(split) > 1 and split[0] == g:
            age_group = split[2]
            for k in AGE_GROUPS.keys():
                if age_group in AGE_GROUPS[k]:
                    temp = AGE_GROUPED_COLUMNS[g][k]
                    temp.append(c)
                    AGE_GROUPED_COLUMNS[g][k] = temp
                    temp = AGE_GROUPED_COLUMNS["Total"][k]
                    temp.append(c)
                    AGE_GROUPED_COLUMNS["Total"][k] = temp
...
COLS = []
# Use the AGE_GROUPED_COLUMNS map to compute new columns.
for gender in AGE_GROUPED_COLUMNS.keys():
    for group in AGE_GROUPED_COLUMNS[gender].keys():
        c = f"{gender} - {group}"
        COLS.append(c)
        cols_to_sum = AGE_GROUPED_COLUMNS[gender][group]
        new_df[f"[count] - {c}"] = grouped_df.loc[:, cols_to_sum].sum(axis=1)
...
# Normalize by population.
for col in COLS:
    new_df["[%_in_OA] - " + col] = (new_df["[count] - " + col] / new_df["Total population"])
  * 100
...
```

Listing 5.10: Age and Gender distribution segmentation column generation.

The *[count]* prefixed columns compare segment counts between OAs, where the *[percentage in OA]* columns compare segment percentages between OAs. They are both useful metrics for tackling different questions, and hence, produce slightly different choropleth maps, as shown in Figure 5.13.

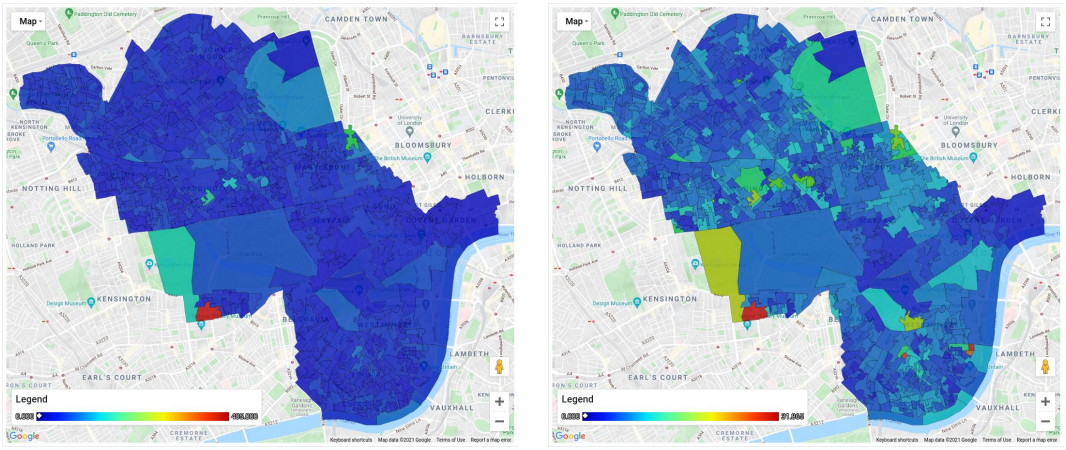


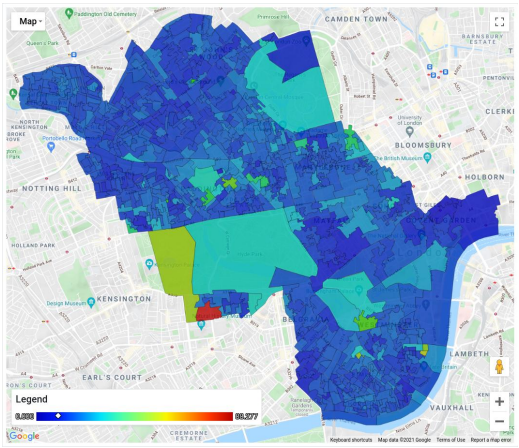
Figure 5.13: Resident distribution Females - Universitarian/apprentice [18-24] OA count (left) vs OA population percentage (right).

Clicking on any OA while having a non-total column selected for this dataset generates a side visualization presenting a value breakdown per age and gender in the OA. The histogram in Figure 5.14 shows the distribution of both males and females per age band in OA E00024092, with *Young Adults* being the predominant type for both genders.

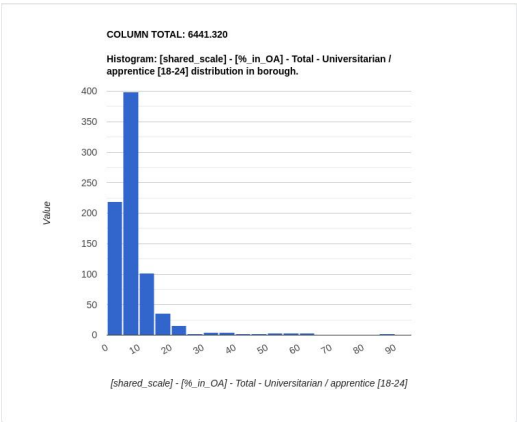
Figure 5.14: Resident distribution Male - Young adult [25-29] OA percentage (left) and associated side visualization histogram.

A useful feature of shared scale columns is that they maintain both the choropleth maps and the auxiliary histograms on the same scale, meaning that the distributions of the different types can be tracked and compared more easily. Figure 5.15 reveals how the data representation in the maps (representing data at the OA level) matches that of the histograms (representing data at the borough level). The universitarian/apprentice (a) band has the colder map with the

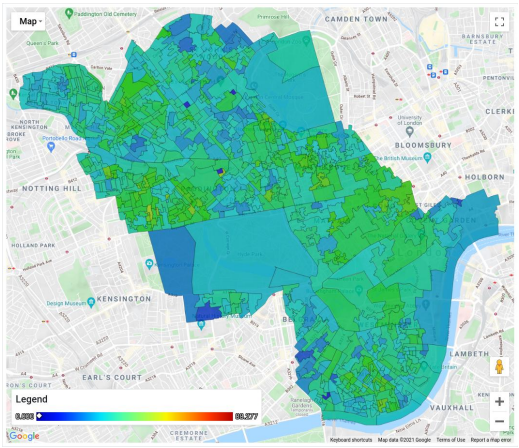
warmer spots. This is mirrored in its histogram (b) by having the most left-skewed distribution and the most outliers.



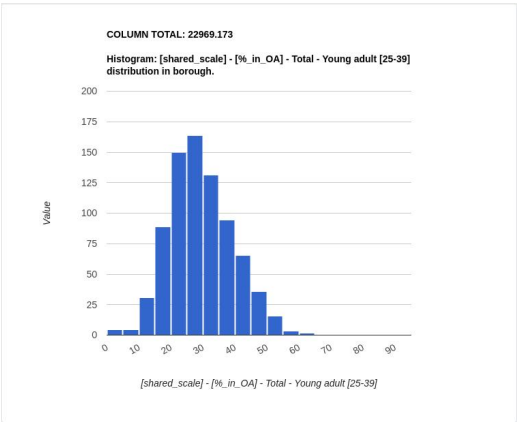
(a) Uni./appr. percentage in OA choropleth map



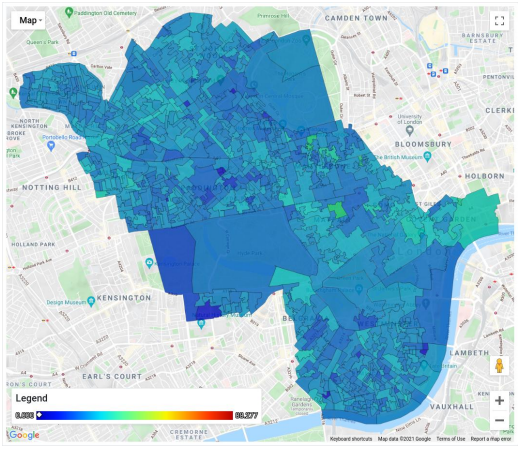
(b) Uni./appr. borough total histogram



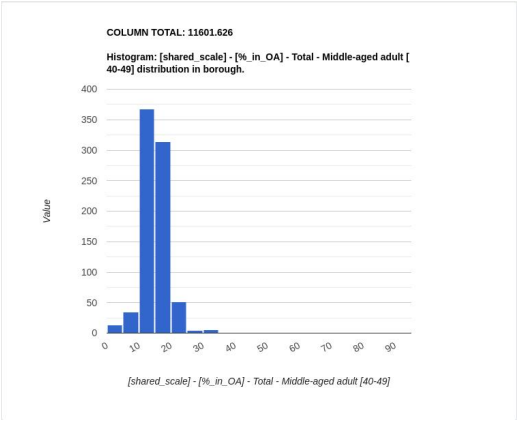
(c) Young adult percentage in OA choropleth map



(d) Young adult borough total histogram



(e) Middle-aged adult percentage in OA choropleth



(f) Middle-aged adult total borough histogram

Figure 5.15: Resident age and gender distribution shared scale column comparison.

5.3 OA Normalizing properties

The normalizing properties dataset is included in the user interface as an explanatory tool. It contains the properties that are used to normalize or disaggregate [3] the data collected (and thus aggregated) by postcode or OA. An instance of this is diving the population count of an OA by its physical size, thus disaggregating the data from *population by OA* and aggregating it by *population per size unit of OA* (i.e. a square meter). This solves the problem of physically larger sample areas usually accumulating more data than smaller ones, something that is necessary to visualize data in choropleth maps.

A practical example can be seen in Figure 5.16. Alcoholic drink sales summed by OA mostly favour those of larger size. When the same column is normalized or aggregated by population, however, higher sale numbers correlate positively with places of low resident populations (areas in green). This is a much more meaningful relationship than to OA area size.

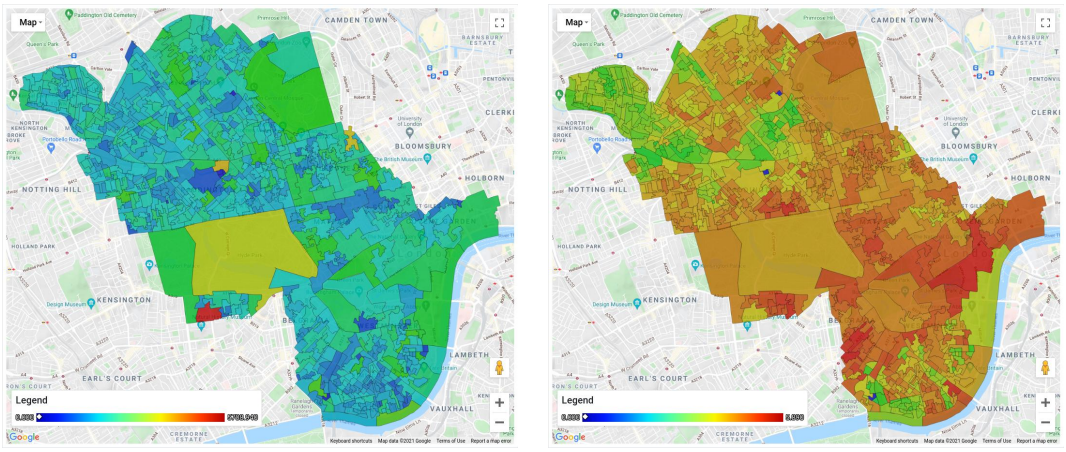


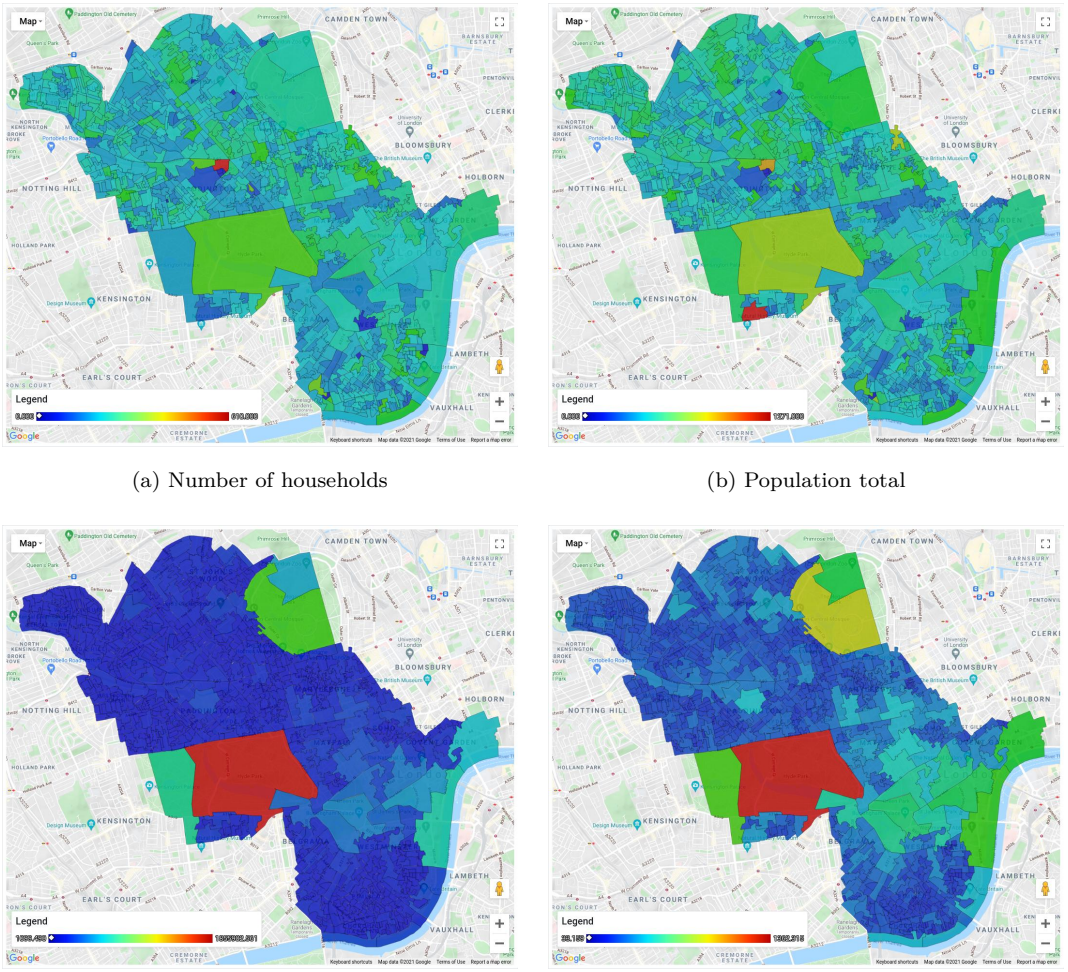
Figure 5.16: Alcohol sales sum by OA (left) vs average per person by OA (right).

In the particular case of authorities in the UK, the polygonal shape of the census areas known as OAs depends mainly on the location, topography, land use, population, social makeup, and the number of households of the area being surveyed [8]. Therefore, it is established that OA area, number of households and population are valid disaggregation properties, and these are the ones presented in the project. Figure 5.17 shows these attributes mapped, and it also includes the OA area square root (in meters) property, defined as *OA effective area*.

As mentioned in subsection 3.2, population density metrics can be tailored to particular use cases. In this project, OA effective area is defined as the area (in square meters) of an OA available for physical socio-economic activities, meaning the presence of people or businesses.

Mathematically it is defined as:

$$OA\ effective\ area\ (s/m)\ =\ \sqrt{OA\ polygon\ area\ (s/m)}$$



(a) Number of households

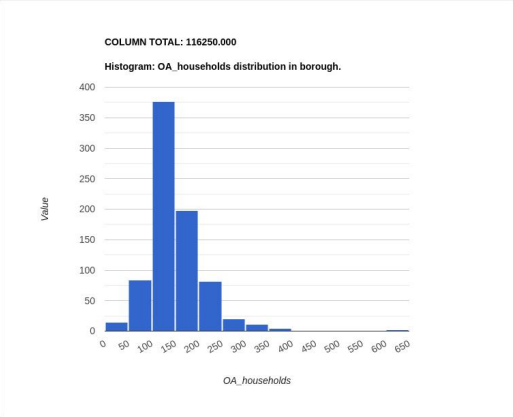
(b) Population total

(c) Area (meters)

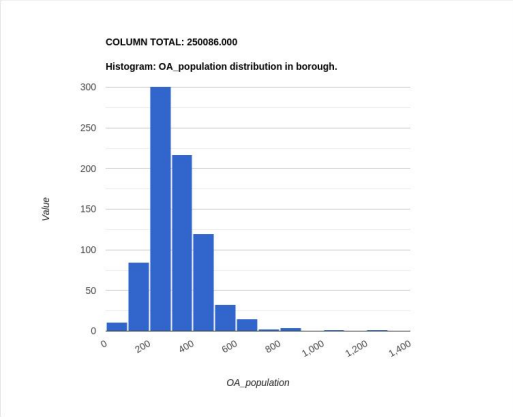
(d) Effective area (meters)

Figure 5.17: Normalizer OA properties.

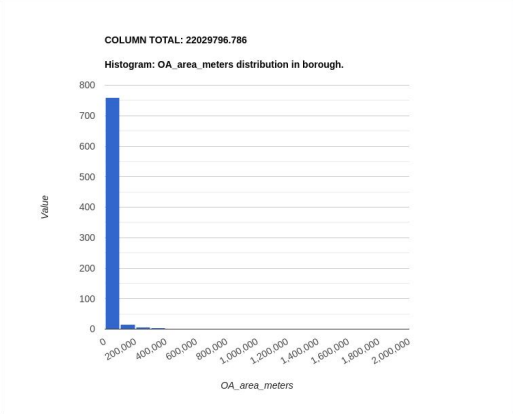
The goal of the OA effective area property is to widen (or spread out) the distribution of the OA area metric so that it becomes a better normalizer. Figure 5.18 attests that this has been achieved. The distribution of effective area is a lot closer to that of households and population in terms of range and shape. This could be attributed to the real-life metric they represent and how it is spread out amongst OAs. Generally speaking, these similarities are expected as both households and population are properties that define the size of OAs.



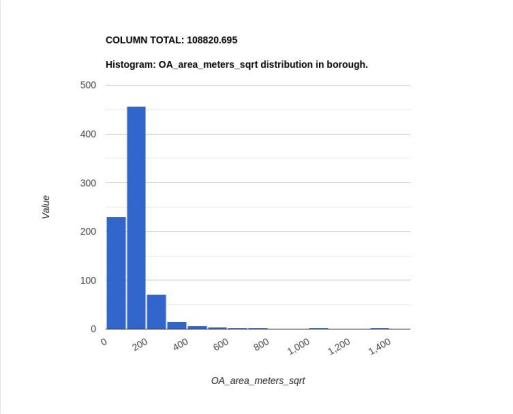
(a) Number of households distribution



(b) Population total distribution



(c) Area (meters) distribution



(d) Effective area (meters) distribution

Figure 5.18: Normalizer OA properties histograms.

Another factor worth noting is land use. Figure 5.19 shows the relevant choropleth maps with low opacity overlays to expose the map labels underneath. It is clear that most large OAs in Westminster encapsulate large natural spaces like parks, zoos or bodies of water. Normalizing by OA area heavily affects these OAs and has little to no effect on medium to small-sized ones. This is undesirable for two reasons: Firstly, natural spaces can encourage social activities so these OAs should not be affected so disproportionally; and secondly, OAs of any size that do not contain natural spaces should be affected proportionally by normalizing factors.

Effective area solves or at least lessens the impact of these issues. As it is the square root of OAs' areas, the larger their size, the less impactful their normalization value becomes. Good evidence of the effectiveness of this new property is the greater variation in normalization values between OAs of all sizes. As a result of this, both small OAs containing natural spaces and larger OAs with no natural spaces are affected proportionally to their size, and reasonably so

when compared to other OAs of similar smaller and larger sizes.

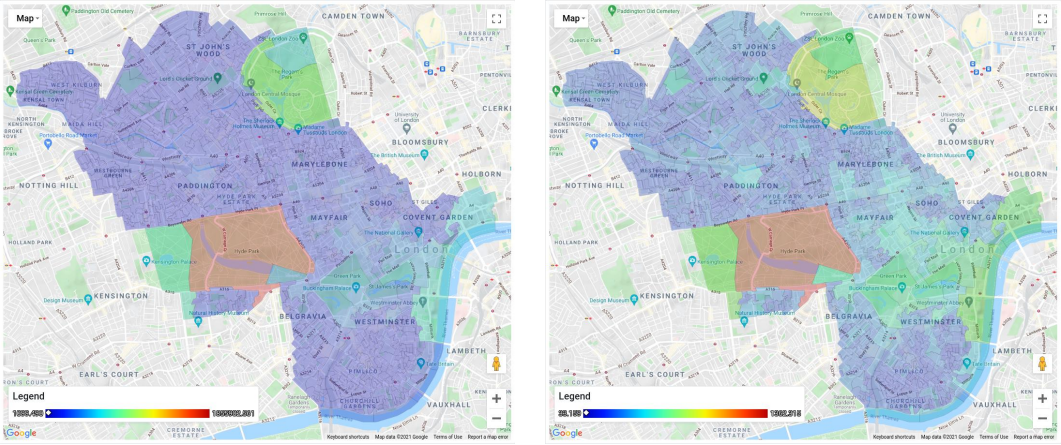
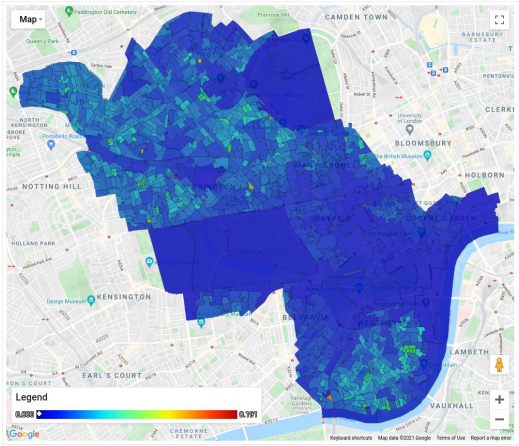


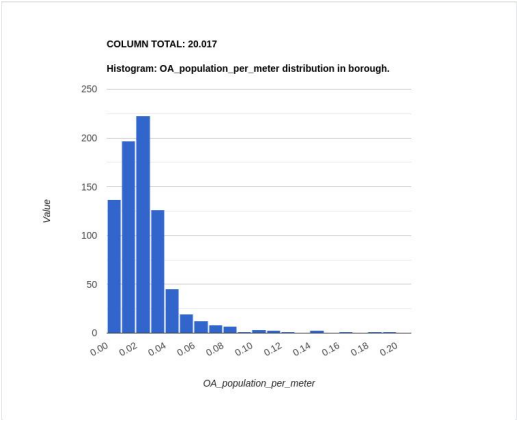
Figure 5.19: OA normalizers low opacity map: Area (left) vs Effective Area (right).

Figure 5.20 reveals these effects through a comparison of the OA population values being normalized by area and effective area. In map (a), large OAs get a population by square meter value of 0 (evidence of being dis-proportionally affected) and the smallest OAs get the highest values, becoming outliers (evidence of small OAs having a near-zero normalization factor). In contrast, map (c) shows a much smoother choropleth map, with fewer false outliers and large OAs with non-zero population values. Furthermore, the histograms created by both maps show that the area normalized data displays a distribution similar to the normalization property, indicating that it is too heavily impacted by it. On the contrary, the effective area normalized data shows a distribution very similar in shape and skew to the population total by OA histogram, indicating a good disaggregation estimate.

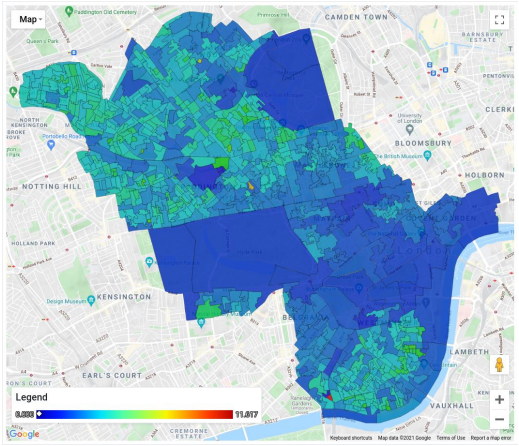
The development of this normalizing method was heavily impacted by the processing and visualization of the Places dataset. The next section further explores normalization effects applied by the OA area and OA effective area properties.



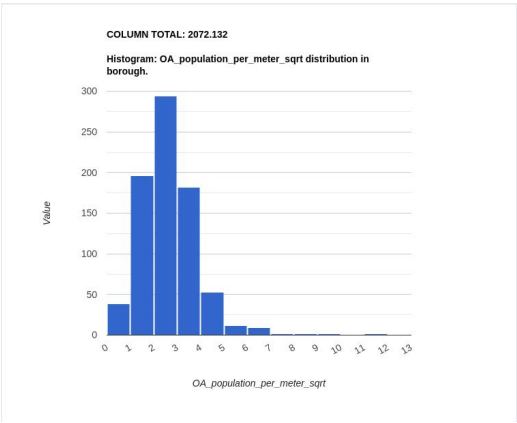
(a) Population norm. by area



(b) Population norm. by area histogram



(c) Population norm. by e. area



(d) Population norm. by e. area histogram

Figure 5.20: OA population count normalized by Area vs Effective Area.

5.4 Google Maps Places API data mining

The Places dataset is made using the Google Maps Places API. As mentioned prior, it contains, filtered by Westminster, all places, landmarks and other entities catalogued by Google Maps. This data is required for the place-based demographic population estimation method proposed in this project, but it is also an incredibly useful dataset on its own as its breadth and enrichment make it relevant to a plethora of urban planning applications.

In this project, the API is queried using areal search, where an area is defined and only entries within it are returned. The type of information attached to each entry varies, but an average case is shown in Listing 5.11. It contains geographical information, as well as the entity's description, type and other fields. The main, if not only drawback of the Places API,

is its anti-data scraping measures. Query responses are paged with a maximum of 20 results per page and capped at 3 pages of content per query. Furthermore, accessing the next page in a response requires a wait time of around 2 seconds. Because of this, the data collection strategy must make use of small search areas and properly awaited requests. Two approaches are implemented and discussed in this chapter.


```
[
    {
        "business_status": "OPERATIONAL",
        "geometry": {
            "location": { "lat": 51.51503009999999, "lng": -0.1827121 },
            "viewport": {
                "south": 51.51379046970849,
                "west": -0.184161130291502,
                "north": 51.51648843029149,
                "east": -0.181463169708498
            }
        },
        "icon": "https://maps.gstatic.com/mapfiles/place_api/icons/v1.jpg_71/
generic_business-71.jpg",
        "name": "MarieChristophe Champagne",
        "opening_hours": { "open_now": true },
        "photos": [
            {
            "height": 960,
            "html_attributions": [
                "<a href=\"https://maps.google.com/maps/contrib/109712651063378336560\">
MarieChristophe Champagne</a>"
            ],
                "width": 720
            }
        ],
        "place_id": "ChIJ4aXOynQFdkgRTSWoWTA4Kb4",
        "plus_code": {
            "compound_code": "GR88+2W London, UK",
            "global_code": "9C3XGR88+2W"
        },
        "reference": "ChIJ4aXOynQFdkgRTSWoWTA4Kb4",
        "scope": "GOOGLE",
        "types": ["food", "point_of_interest", "establishment"],
        "vicinity": "Cleveland Square, London",
        "html_attributions": []
    },
  ...
```


Listing 5.11: Google Maps Places API sample response.

5.4.1 Mining with the Javascript API

The Places Javascript API is provided by Google. It is designed to be used mostly in client-side web or mobile applications so, it is less suitable for data mining operations than some alternatives. What it does offer, is a comprehensive range of built-in API operations, and the ability to integrate with the Google Maps Javascript API and be used to visualize and verify the Places mining process (bingo!).

Listing 5.12 shows snippets of the mining method. It uses *nearbySearch* (a type of areal search) with rectangular bounds for defining the search area. It also includes the paging mechanism, accumulation of results and the sleeping procedure.

```
...
// Initialize search.
await service.nearbySearch(request, function (results, status, pagination) {
    if (results !== undefined ){
        if (results.length > 0) {
            results = results.filter(function (el) {
                return currentOaPolygonBoundsRect.getBounds().contains(el["geometry"]["
 location"].toJSON())
            })
            ...
            accumulatedResults = accumulatedResults.concat(results)
            ...
    // If present, load next page.
    if (pagination.hasNextPage) {
        pageCounter = pageCounter + 1
        pagination.nextPage()
        ...
});
// Sleep while the searching procedure is still active.
while (searching) {
    console.log("still searching")
    await sleep(500)
}
// Return results when the searching procedure becomes inactive.
return accumulatedResults
...
```

Listing 5.12: Places API data mining with the Javascript Places API.

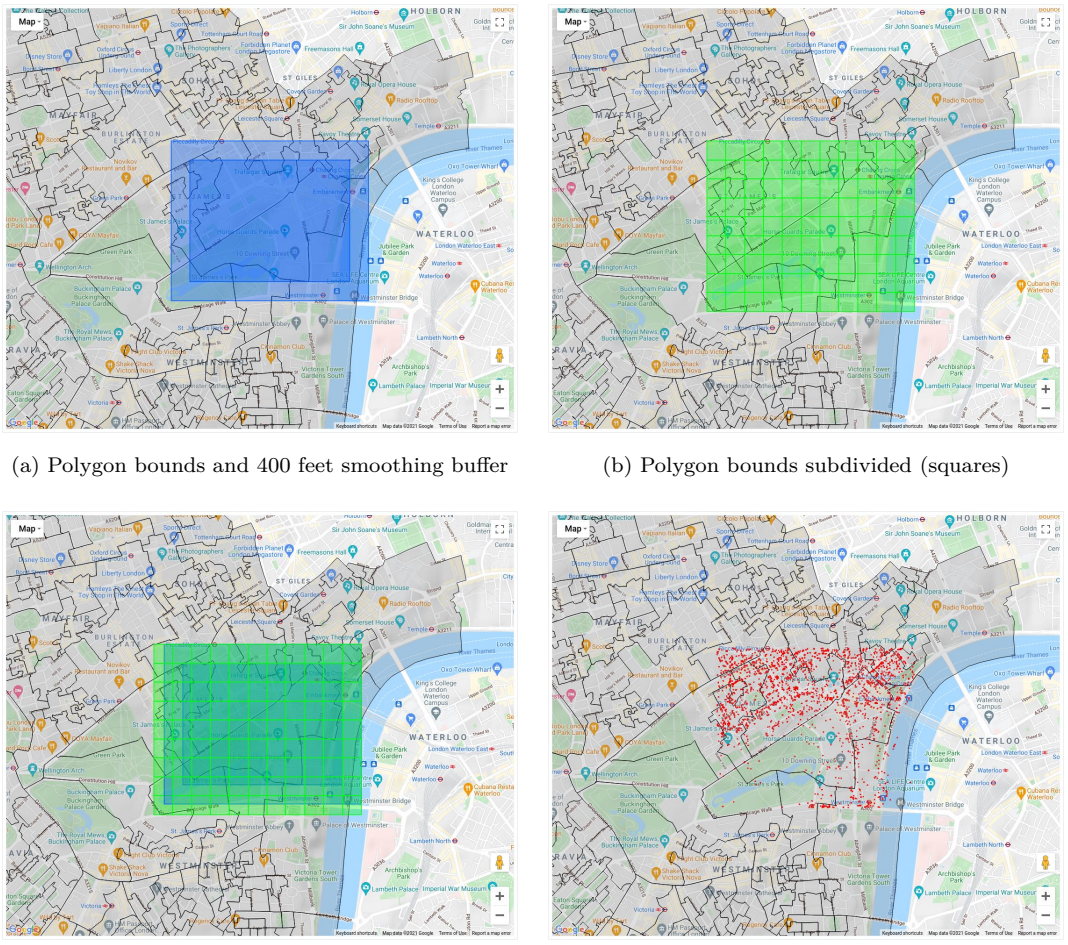
As the pure Javascript application runs on the browser, it does not have access to the file system so it requires a server-side API to allow it to save the results of the mining process. Listing 5.13 presents the definition of two endpoints to check the existence of and save newly mined results.

```
...
class OAsPlaces(Resource):
    # Get list of file names.
    def get(self):
        onlyfiles = [f for f in listdir(files_dir) if isfile(join(files_dir, f))]
        existingOAs = [x[:9] for x in onlyfiles]
        print(f"RESPONSE: {existingOAs}")
        return {'existing_OAs': existingOAs}

    # Write to JSON.
    def put(self, filename):
        body = request.form['data']
        output_json = body
        save_json_to_file(DATA_OUT_DIR, output_json, f"{filename}.json")
        return "All good!"

api.add_resource(OAsPlaces, '/<string:filename>', '/')
...
```

Listing 5.13: Python Flask server-side API for saving Places data.

The mining process is done in 5 main steps. These are visualized in Figure 5.21:

1. An OA is chosen and a rectangular area around it, bounded by the OA polygon's dimensions, is defined.
2. An extended area is built that extends the original rectangle by 400 feet or 120 meters on all sides.
3. The new area is subdivided into 120 square meter squares that fully cover it. This may result in a further extension of the search area, but results are filtered accordingly.
4. Each subdivision rectangle is queried for places and the results are saved to file under the OA's name.
5. Repeat from 1 until all OAs have been mined.



(a) Polygon bounds and 400 feet smoothing buffer

(b) Polygon bounds subdivided (squares)

(c) Polygon bounds and subdivision surplus

(d) Polygon search area results

Figure 5.21: Javascript project Places mining.

Through trial and error, 120 sqm squares were found to be suitable small enough search areas as they return less than 60 results the vast majority of the time. The search area buffer

exists as a smoothing technique. It implies that places within 120 meters of an OA border are accessible to people in that OA. The search area shape was chosen to be easily subdivided. Rectangles and circles were tested and produced similar results depending on the OA's shape. The surplus of entries each shape produces can be considered a further smoothing factor.

The user interface of this Places Mining project during execution is shown in Figure 5.22. Green squares for each OA are drawn on the map as they are mined and result logs are printed in the console window alongside. They specify the subdivision of the OA's original search area, how many results each small square has returned and in how many pages the results are delivered.

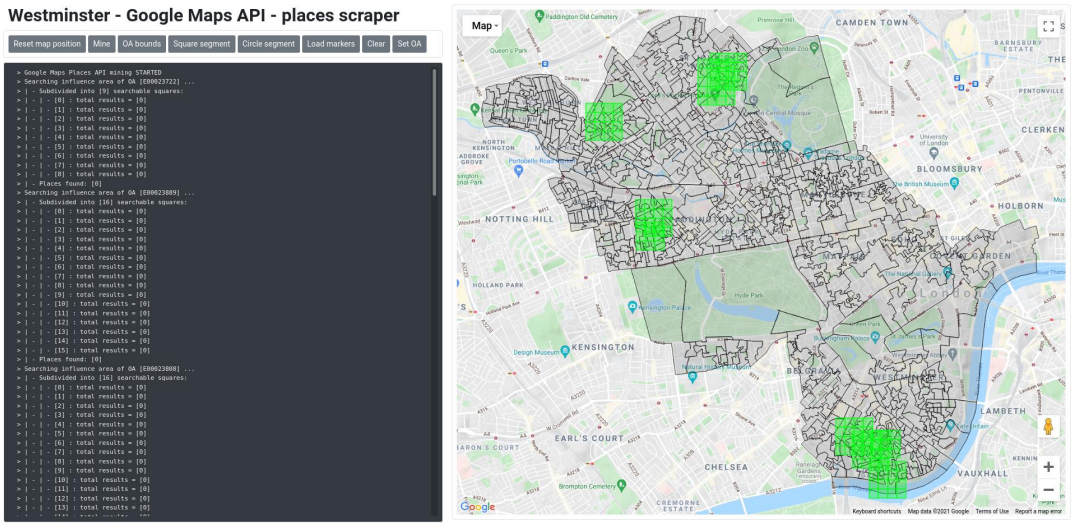


Figure 5.22: Javascript Places Mining project user interface, during execution. Note: all operations return 0 results as the image was generated with a deactivated Google Maps API key.

5.4.2 Mining with the Python API

The Google Maps Places API in Python is written by a third party and it is a much better tool for data mining, despite not offering all the API's functionality like searching using rectangular areas. It is necessary then, to subdivide the OA search areas into circles instead of squares, shown in Figure 5.23. These can now be queried using the API's radial search function which takes as parameters a geographical location and a search radius.

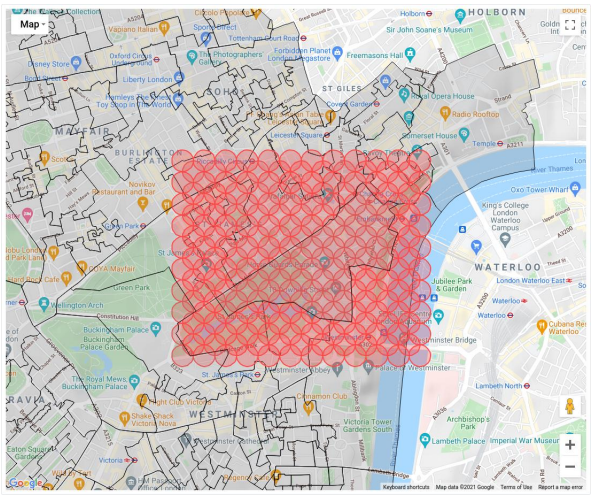


Figure 5.23: Polygon bounds subdivided (circles).

The mining implementation in Python is similar to the Javascript version. The Listing 5.14 contains snippets that: generate circle searchable units from a given OA, query the Places API, handle paging with sleeping, and compile the results. The response for each entry is filtered by the *fieldsToKeep* predefined list in an effort to remove unused information and reduce the file size of the results JSON files. The *existingOAs* list serves to ensure that OAs are not mined more than once in the event that the process needs to be restarted before it is fully completed.

```
fieldsToKeep = ["geometry", "name", "place_id", "types", "rating", "user_ratings_total"]
...
def getNearbyPlaces(row):
    ...
    # Avoid mining already mined OAs.
    if oa_name in existingOAs:
        print(f"- Already explored - ...")
        print(f"Total number of unique results = ...")
        OAs_counter = OAs_counter + 1
        return
    ...
    # Generate and mine searchable units.
    search_area, circles = generateSearchableUnits(row)
    ...
        places_nearby = GMAPS.places_nearby(
            location = location,    # (lat,lng)
            radius = unit_radius_meters
        )

        filtered_results = filterResults(places_nearby, search_area)
        while "next_page_token" in places_nearby:
            time.sleep(2) # it does not work with 1 second!!!
            next_page_token = places_nearby["next_page_token"]
            places_nearby = GMAPS.places_nearby(
                location = location,
                radius = unit_radius_meters,
                page_token = next_page_token
            )
    ...
```

Listing 5.14: Python Google Maps Places API mining.

The main reason for choosing the Python mining solution over the Javascript alternative was its potential to reduce mining times by using multithreading. Listing 5.15 shows the threadable function to be a Pandas lambda expression on a single row of a dataframe corresponding to an OA. The *scrape_places()* method determines the number of threads and executes the multithreaded mining function that will initialize and join the parallel executions. Each call of the threadable function creates a JSON file containing the places mined for a single OA. This lack of dependence on all other computations makes the mining operation suitable for parallelization.

As stated in the code comments, the single thread approach takes days to complete, where the multithreaded, with 100 threads, takes around 40 minutes. Mining all 783 OAs considering the OA influence area used and the way it was subdivided into searchable units, the process costs around $ 300.

```
    # A single threadable operation.
    def do_work(gdf):
        gdf.apply(lambda x: getNearbyPlaces(x), axis=1)

    # Multi threaded mining procedure.
    def multi_thread_mine(gdfs, no_threads):
        start = time.time()
        threads = no_threads
        # Define the jobs.
        jobs = []
        for i in range(0, threads):
            thread = multiprocessing.Process(target=do_work, args=(gdfs[i],))
            jobs.append(thread)
        # Start the threads.
        for j in jobs:
            j.start()
        # Ensure all of the threads have finished.
        for j in jobs:
            j.join()
        ...

    # Executor method. Multitheaded method disabled by default.
    def scrape_places():
        gdf = pd.read_csv(INPUT_DATA_DIR + geo_file)
        no_threads = 100
        gdfs = np.array_split(gdf, no_threads)
        single_thread_mine(gdfs) # runs in TOO MANY seconds.
        # multi_thread_mine(gdfs, no_threads) # runs in around 40 minutes.
    ...
```

Listing 5.15: Python Google Maps Places API multithreaded mining.

5.4.3 Places dataset aggregation and normalization

Once the raw Places data has been collected, it can be processed. The first step is to combine the 783 Places files by OA into a single JSON file (without any information loss), and in the

process, filter out the Google Maps place types in Listing 5.16. These types are political, routing or locality labels which are not necessary. The new file *OA_places.json* is stored in the *focused_data* directory.

```
filter_mask = ["route", "locality", "sublocality", "sublocality_level_1", "neighborhood", "
 political"]

# Filter an element out (return True) if it is found in the filter mask.
def filter_func(e):
    if e in filter_mask:
        return False
    else:
        return True
...
```

Listing 5.16: Places data file cleaning.

The next step is to create the places tally file. This is a reduction operation on the new unified places by OA file. Instead of an entry for each unique place under an OA in the JSON file, the places tally by OA CSV file stores the counts of each place type of all the places associated with an OA. Sample entries are shown in Listing 5.17. This operation reduces the file's information complexity and also its size, allowing it to be loaded and visualized in the user interface. Rendering a coloured polygon per OA is far less taxing than rendering thousands of individual place markers.

```
# JSON file OA entry.
"E00023433": {
    "ChIJhQM_I5cadkgRZqIiW9shw_I": {
        "geometry": {
            "location": {"lat": 51.5397297, "lng": -0.1732535},
            "viewport": {
                "northeast": {"lat": 51.5410596302915, "lng": -0.171967119708498},
                "southwest": {"lat": 51.5383616697085, "lng": -0.174665080291502}
            }
        },
        "name": "Compass Creatives",
        "place_id": "ChIJhQM_I5cadkgRZqIiW9shw_I",
        "types": ["point_of_interest", "establishment"]
    },
    ...

# CSV file OA entry
E00023433,1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0, ...
```

Listing 5.17: OA entry: JSON file vs CSV file.

This dataset is available in the UI and as shown in Figure 5.24, it illustrates quantities (place counts) which as discussed before, are affected by the range in OA polygon size when trying to display distribution information in choropleth maps. This occurs for place types of high counts, like cafes, and also low-frequency ones like gas stations. The issue can be resolved by normalizing the dataset, in this case using OA effective area. The new columns now show the

number of places per effective area square meter in an OA. Listing 5.18 contains the operation being done.

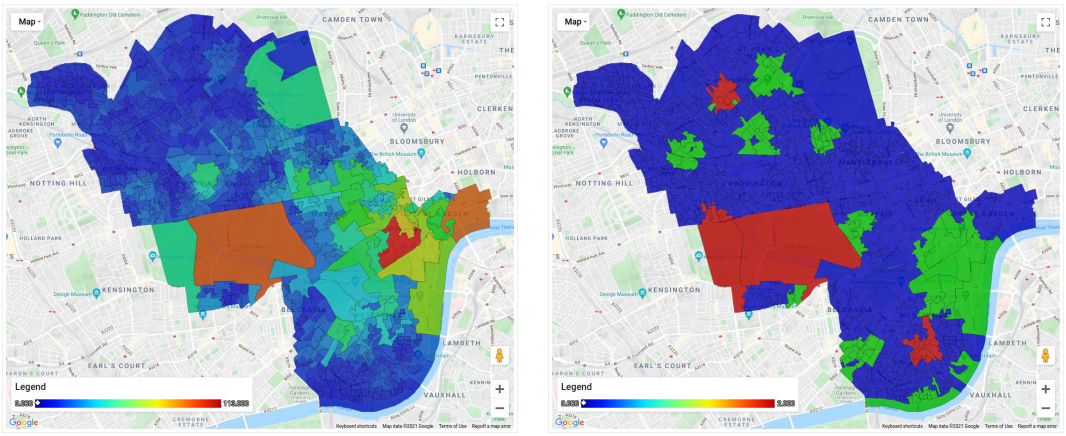


Figure 5.24: Place counts: Cafes (left) and Gas stations (right).

```
# Add njormalizer values to the places tally dataset.
merged = pd.merge(normalizers, oa_type_tally, on="OA")

# Normalize by OA effective area.
normalized_by_effective_area = pd.DataFrame()
normalized_by_effective_area["OA"] = merged["OA"]
for c in merged.loc[:, "accounting":"zoo"]:
    normalized_by_effective_area[c] = merged[c] / merged["OA_area_meters_sqrt"]

# Add shared columns.
normalized_by_effective_area = add_shared_scale_columns(normalized_by_effective_area)
...
```

Listing 5.18: OA place count column normalization.

The effects of the operation are visualized in Figure 5.25. Just as before, the data has been successfully disaggregated by OA and as a result, it is better at showing the spread of data points throughout the borough. Not only are the high concentration areas better-defined, but the values assigned have a higher degree of accuracy, allowing green areas in the gas station count to take on a range of clearly distinguishable colours in its normalized version.

In terms of accuracy, when compared to a map of place markers instead of a heatmap of OAs, the results mirror each other. Figure 5.26 reveals the spread of casinos and art galleries in choropleth map form versus a simple Google Maps search using the same place types as keywords. The map views are not in the same position and the search results only show a limited number of markers, but it is clear that the information is not being distorted or misrepresented.

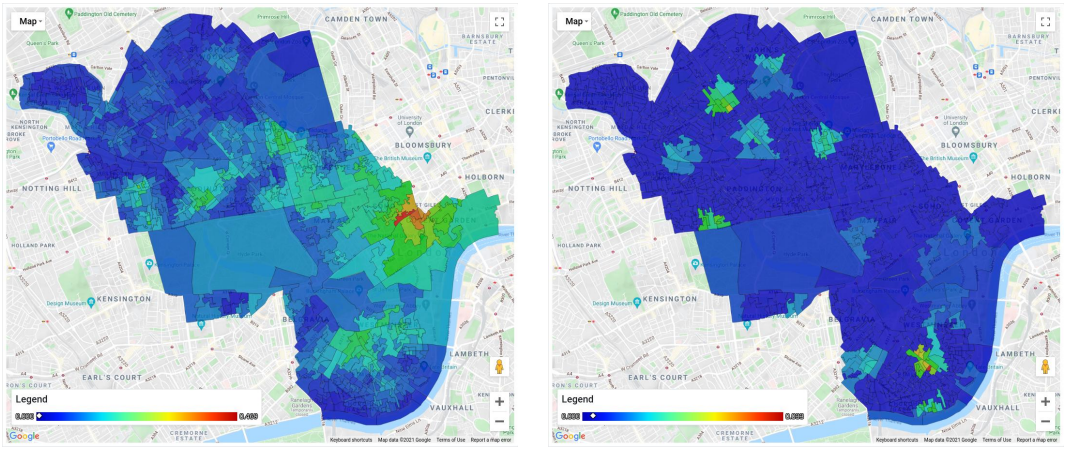
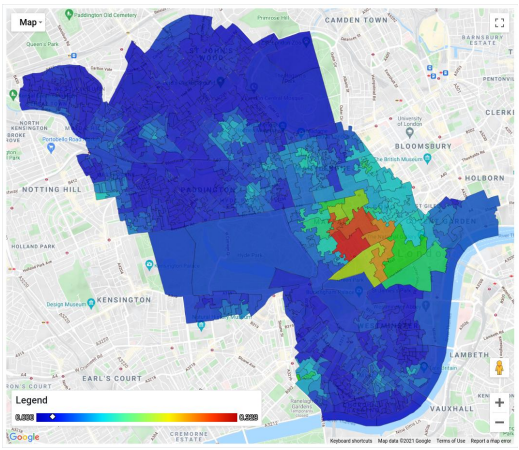
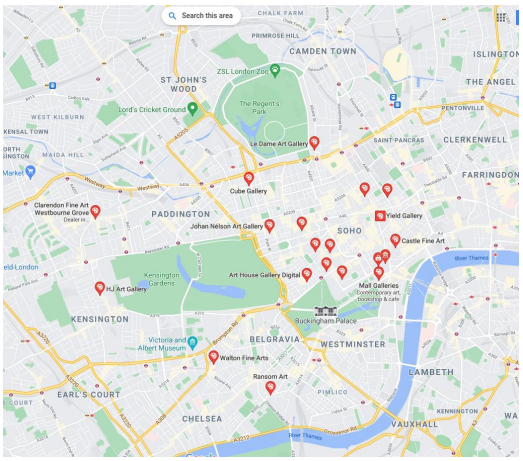


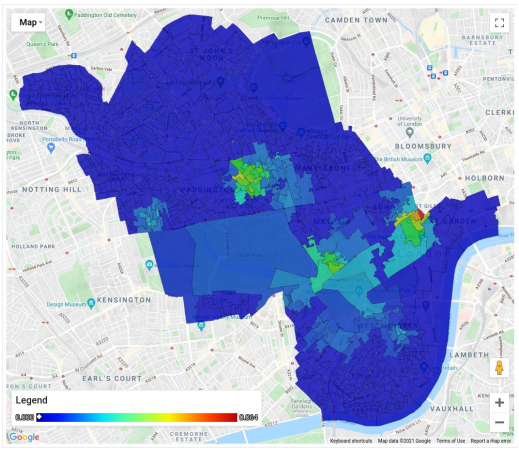
Figure 5.25: Place counts in OA per effective area meter: Cafes (left) and Gas stations (right).



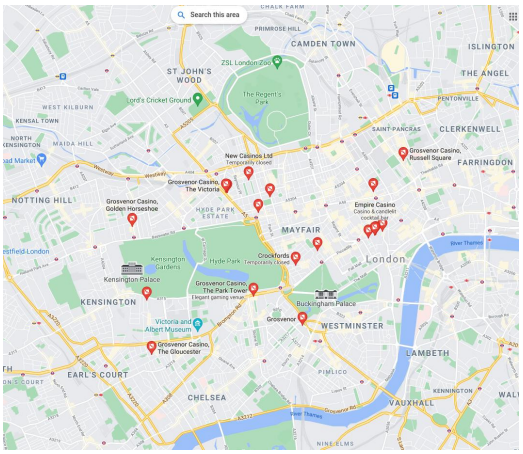
(a) Art galleries distribution



(b) Art galleries place markers



(c) Casinos distribution

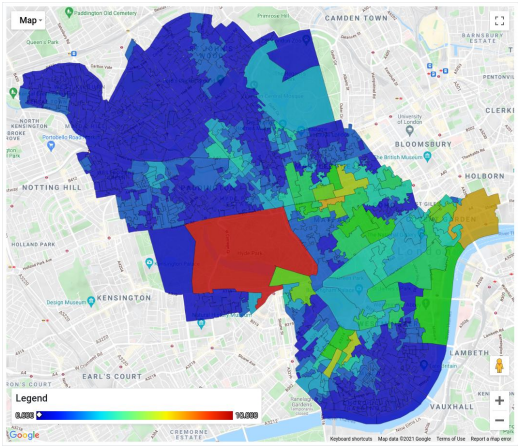


(d) Casinos place markers

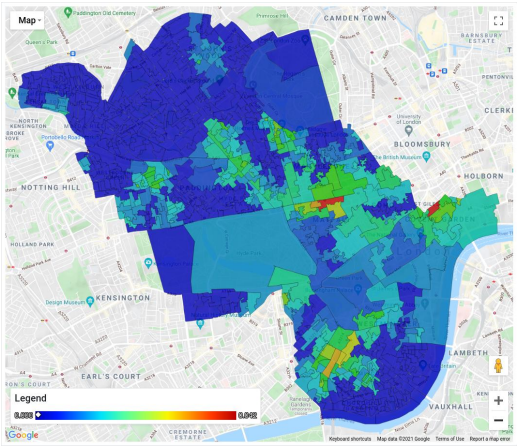
Figure 5.26: Places heatmap by OA vs Google Maps search marker map.

As mentioned previously, many normalization factors are used in the project. OA effective area produces the best results for the Places dataset but other options were considered. Figure 5.27 contrasts the effects of normalizing the count by OA of shopping malls by effective area, as well as households per square meter and households per square meter (bounded), which features a restricted range.

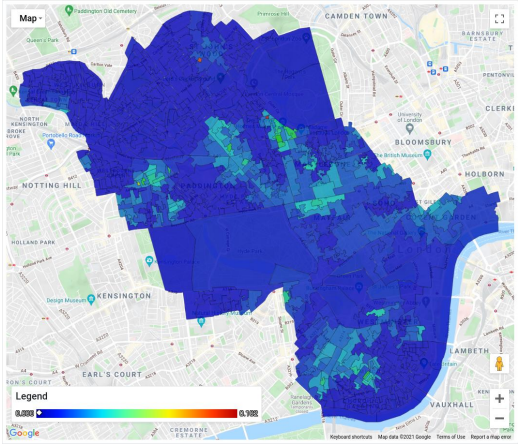
Comparing the results amongst themselves and against place marker maps, OA effective area (b) is by far the best normalization factor for the Places dataset. Households per square meter (c) introduces the variable of housing, which might not be relevant in this domain (and could add bias to the results), and the bounded solution (d) artificially tries to eliminate extreme values to narrow the resulting distribution.



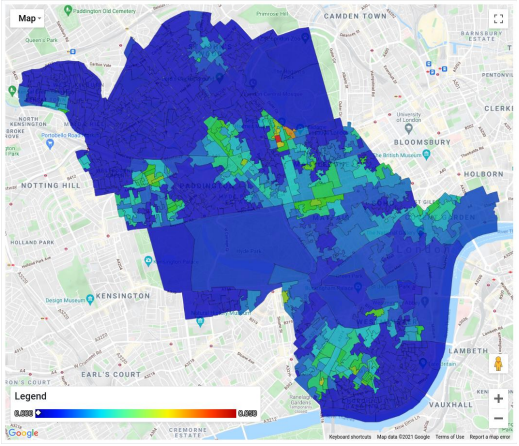
(a) Count by OA



(b) Effective area square meter



(c) Households per square meter



(d) Households per square meter (bounded)

Figure 5.27: Normalization factors comparison: Shopping malls.

5.5 Daytime population demographic distribution estimation analysis

The population estimation component of this project aims to find the distribution and propose a segmentation of the daytime population in Westminster. A number for the estimated population can be set by hand and the models built can be used to convert the derived distributions into people counts for each demographic. The subpopulations are defined by a principal reason that grants visitor status in Westminster:

- **Worker:** working in any kind of employment and in any kind of establishment. From office buildings to pop-up markets.
- **Student:** studying at any level of academia. From early years to higher education.
- **Tourist:** engaging in any activity that could be considered tourism, with more emphasis on particularly touristic activities. From visiting monuments to coffee shops.
- **Shopper:** shopping with more emphasis on non-food items. From clothing and electronic devices to home furniture.
- **Leisurer:** engaging in any activity that could be considered leisure with more emphasis on non-touristic activities. Includes attending restaurants, parks, cinemas and pubs.
- **Chorer:** people (mostly locals) consuming non-recreational or food-shopping services. Covers the cases not included in the other categories like grocery shopping, tradesperson service usage and local government engagement.

5.5.1 Demographic distributions by place type

The inputs to the population estimation function are: The places tally per OA dataset discussed before, and a demographic distribution by place file that defines the demographic makeup of each place type (shown in Listing 5.19). The columns include the six people types and a *relevance* factor that is used to weigh the physical size or capacity of each place type. In this case, the demographic values add up to 100 for each place type because they can easily be understood as percentages, but this is not strictly necessary. These files are also referred to as *models* and they are the user input in the population estimation process. They can be made to increase the accuracy of a population model predicting the current circumstances in the borough, but also modified to analyse hypothetical situations that would cause a change in the distribution of demographics at places. A recent and definitely suitable example would be the aftermath of the Covid-19 pandemic. With schools running at reduced capacity and many

people choosing to work from home, how does this change the number and spread of workers and students in Westminster? How have flying restrictions changed the spread of tourists? Have cultural hotspots remained the same pre and post-pandemic?

```
place_type,relevance,worker_perc,student_perc,tourist_perc,shopper_perc,leisurer_perc,
 chorer_perc
accounting,2,55,5,5,5,10,20
airport,2,25,5,55,5,5,5
amusement_park,3,10,10,10,25,10,35
aquarium,3,5,10,35,10,35,5
art_gallery,3,10,5,30,10,40,5
atm,1,10,10,25,25,15,15
bakery,2,10,10,25,20,30,5
bank,2,20,10,10,10,10,40
bar,2,10,10,15,5,55,5
...
```

Listing 5.19: Demographic distribution by place type (granular model).

5.5.2 Demographic distributions by Output Area

The steps of the demographic distributions calculation process are shown in Figure 5.28. The end product is a group of columns (one for each demographic type) containing proportions and population values by *OA* and by *OA effective area square meter*, at the scope of individual OAs and also the entire borough (5., 6., 7. and 8.).

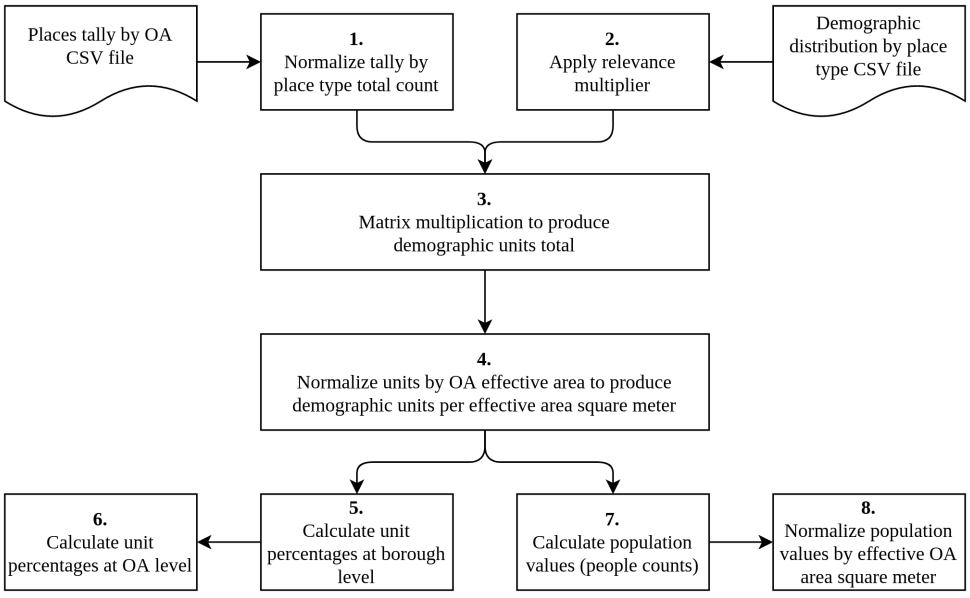


Figure 5.28: Demographic distribution columns computation process.

1. Place count normalization

Place count normalization is a technique designed to create an inversely proportional relationship between a place type's frequency and its weight in calculations. Places of rare types

(like museums) are likely to be more influential than frequent ones (like cafes). Listing 5.20 shows the operation being done. All type column counts are divided by the square root of their column total. The square root function is used here again to narrow the divisor's distribution and avoid very small weight values.

```
    # Recalculate type counts, dividing each by a metric of their column total.
    types_counts = tally.loc[:, "accounting":].sum()
    for c in tally.columns[1:]:
        tally[c] = tally[c] / math.sqrt(types_counts[c])
```

Listing 5.20: Place count normalization.

2. Relevance multiplier

The relevance value encodes the people traffic a place encounters, but can also be used to represent how likely a place is to retain people in its vicinity and have them interact with their surroundings. A museum would have high relevance, while a bus station would have low relevance. Listing 5.21 shows that each demographic percentage is simply multiplied by the relevance value of the associated place type.

```
    # Recalculate distributions, multiplying each row by its relevance.
    dists = dists.sort_values(by=["place_type"])
    for c in dists.columns[2:]:
        dists[c] = dists[c] * dists["relevance"]
```

Listing 5.21: Applying the relevance multiplier.

A good example of this feature is showcased in Figure 5.29. The shopper distribution starts to make more sense when the relevance of shopping malls, clothing stores and electronic stores is set to a value higher than 1. Instead of being centred around a university and hospital hub, the hotspot now lays over the Soho and Covent Garden areas, known for their shopping.

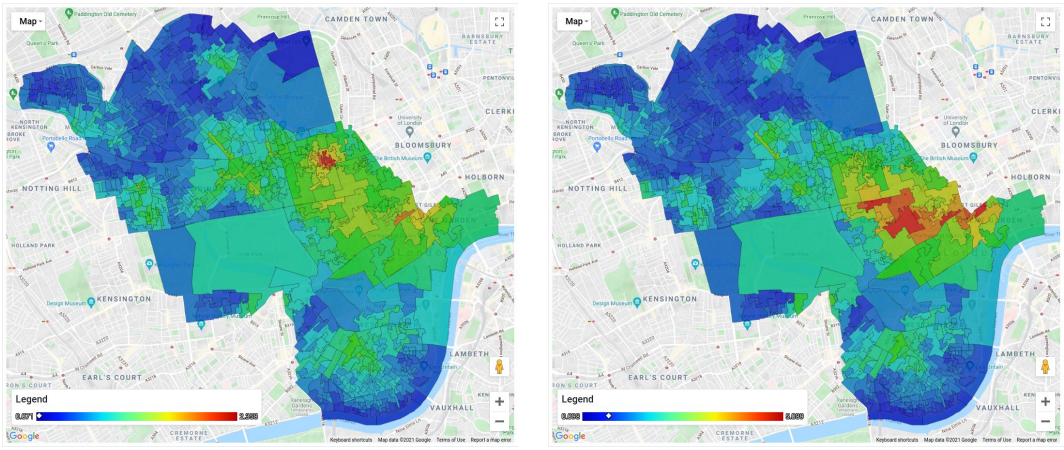


Figure 5.29: Shopper demographic distributions: no multiplier (left) vs relevance multiplier (right).

3. Demographic units calculation and normalization

This step calculates a value for all demographics in each OA. This is done by multiplying a demographic's distribution value for each place type, by the tally value of that place type in a given OA, then summing the results. Listing 5.22 contains the core of the operation. The result is a dataset containing demographic distributions per OA expressed in units, a temporary value that can be turned into more explainable results.

```
    for oa in oas:  # For each of the 783 OAs.
        # Count of each place type in the OA.
        oa_place_count = tally.loc[tally["OA"] == oa].values.tolist()[0][1:]
        ...
        for demo_type in demo_types:    # For each of the 6 demographic types.
            # Distribution values of the demo type for all place types.
            demo_place_percs = dists[demo_type].values.tolist()
            # Get a list of the products of all pairs of a and b.
            multiplied = [a * b for a, b in zip(oa_place_count, demo_place_percs)]
            # Sum all the products.
            summed = round(sum(multiplied), 3)
            ...
```

Listing 5.22: Dataset multiplication.

The dataset is then normalized by effective area to produce visualizable columns of units per effective area square meter (shown in Figure 5.30).

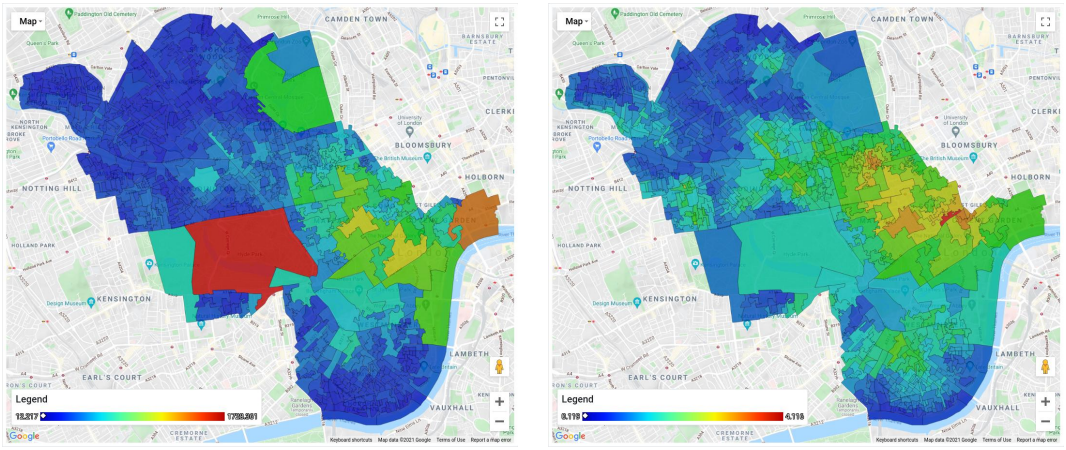


Figure 5.30: Worker demographic distribution: Denormalized (left) vs Normalized (right).

4. Units conversion to population percentages

The temporary metric of units is converted in Listing 5.23 to percentage values at the OA and borough level. This is done by dividing unit cells by either demographic totals in the borough or in individual OAs.

```
    ...
    # Calculate the percentage the units of a type in an OA out of the total units of that type
     in the borough (all the OAs).
```

```
    for focus_col in RELEVANT_COLUMNS:
        col_sum = normalized_by_effective_area[focus_col].sum(axis = 0)
        normalized_by_effective_area[f"[%_of_borough_total] - {focus_col}"] = (
     normalized_by_effective_area[focus_col] / col_sum) * 100
    ...
    # Calculate the percentage the units of a type in an OA out of the total units of any types
     in that OA.
    normalized_by_effective_area["[per_effective_area_square_meter] - total_units"] =
     normalized_by_effective_area[cols_to_sum].sum(axis = 1)

    for col in cols_to_sum:
        normalized_by_effective_area[f"[%_of_OA_total] - {col}"] = (normalized_by_effective_area
     [col] / normalized_by_effective_area["[per_effective_area_square_meter] - total_units"]) *
     100
    ...
```

Listing 5.23: Unit conversion to OA and borough percentages.

Figure 5.31 shows the resulting choropleth maps using the demographic of tourists. The (left) map indicates that 0.6 % of all tourists in Westminster are found in the OA in red, whereas the (right) map indicates that the OA in red has a demographic composition of 27.9 % tourists.

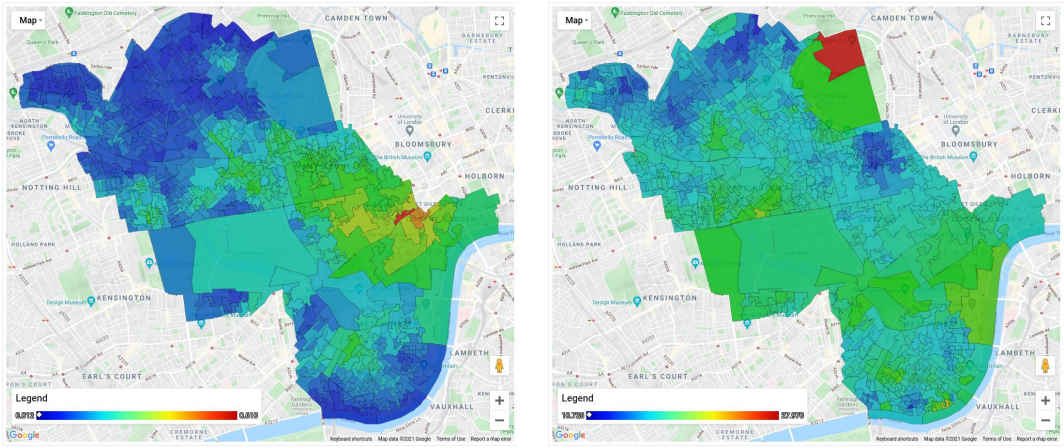


Figure 5.31: Tourist demographic percentages: In borough (left) vs In OA (right).

5. Units conversion to population values

The conversion from units to people counts per OA requires the setting of a daytime population value. In this project, Westminster's total daytime population is estimated at 1,000,000 using the source [18]. The process is done 7 steps:

1. Calculate the total sum of units per demographic.
2. Calculate the total sum of units in the borough.
3. Calculate the proportion that each demographic makes up of the total unit sum.
4. Define the total daytime population value.

5. Multiply the total daytime population value by the proportions of each demographic from step 3.
6. For each cell in the dataset, divide its value by the column total and multiply the results by the daytime population value assigned to the cell's demographic.
7. Normalize the dataset by dividing all cell values by their OA's effective area.

The visualized results for the student demographic are shown in Figure 5.32. The maps are aesthetically the same as their units versions, but now the legend values indicate people counts per effective area square meter instead.

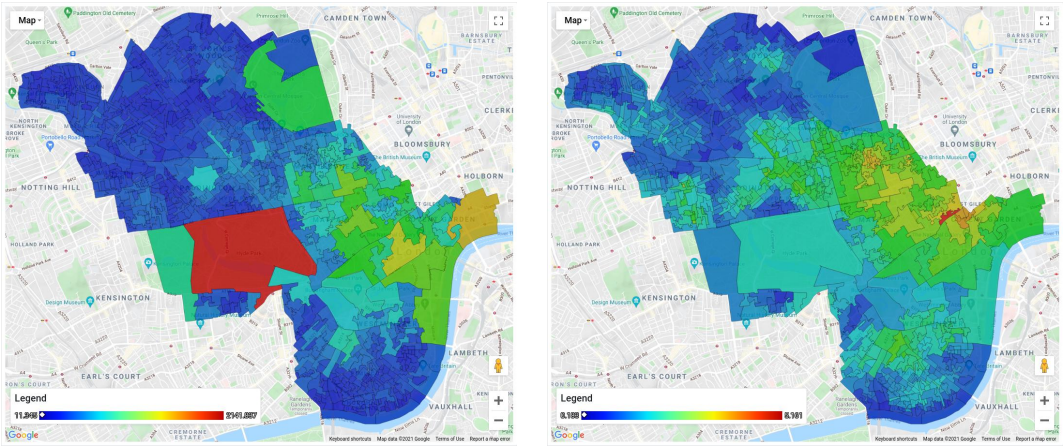


Figure 5.32: Student demographic population values: Denormalized (left) vs Normalized (right).

5.5.3 Demographic distributions alternative models

During the development of the project, many models were developed to test new features, experiment with hypothetical scenarios and increase the accuracy of the estimation method. This section presents four memorable ones.

5.5.3.1 Granular model

The granular model, (results in Appendix section C.1) was created by manually assigning demographic percentage values to each of the 103 place types. The model is liberal because it encourages demographics to act outside their principal activity by assigning relatively high values to non-primary demographics at all place types, with the minimum assignable value set at 5 %. An example of this is the *post_office* type. Although it is arguably a mostly chorer type, it assigns 15 % to workers and students, and 10 % to shoppers and leisurers.

This strategy leads to a model that favours areas with a high volume of places of all types, and this is evident in the demographic maps, where students, tourists and leisurers have the same dominant OA and very similar distributions. This is reinforced by the total distribution shown in Figure 5.33 which again, mimicks the same patterns.

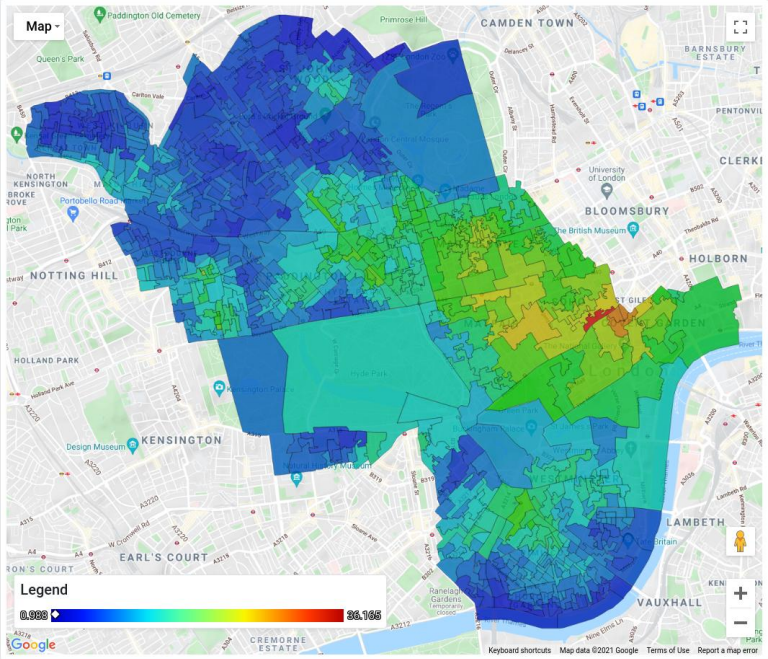


Figure 5.33: Granular model total population distribution estimate.

5.5.3.2 Supertypes model

The supertypes model, (results in Appendix section C.2) was created to test an approach aimed at making the models more easily editable by hand. The 103 place types are arranged into 14 groups (shown in Listing 5.24). Demographic distribution percentages can then be defined for each supertype and these inputs are used by a script to generate the actual model file that is used to produce the population estimate.

```
# Place types groupings into supertypes.
religion_0 = ["hindu_temple", "place_of_worship", "synagogue", "mosque", "church"]
store_1 = ["home_goods_store", "convenience_store", "liquor_store", "book_store", "
 shopping_mall", "grocery_or_supermarket", "supermarket", "hardware_store", "florist", "
 clothing_store", "department_store", "bicycle_store", "shoe_store", "movie_rental", "
 drugstore", "jewelry_store", "electronics_store", "pet_store", "furniture_store"]
chore_2 = ["storage", "moving_company", "car_dealer", "plumber", "car_repair", "painter", "
 travel_agency", "electrician", "post_office", "real_estate_agency", "car_wash", "locksmith",
  "laundry", "general_contractor", "roofing_contractor", "car_rental", "insurance_agency", "
 cemetery", "funeral_home"]
academia_3 = ["university", "secondary_school", "school", "primary_school", "library"]
medicine_4 = ["dentist", "veterinary_care", "pharmacy", "hospital", "physiotherapist"]
transport_5 = ["parking", "bus_station", "train_station", "subway_station", "transit_station
 ", "taxi_stand"]
```

```
    hospitality_6 = ["cafe", "bar", "bakery", "meal_takeaway", "restaurant", "meal_delivery", "
     food"]
    generals_7 = ["point_of_interest", "establishment", "premise", "health", "doctor", "store"]
    legal_8 = ["courthouse", "city_hall", "local_government_office", "lawyer", "embassy"]
    attraction_9 = ["aquarium", "amusement_park", "lodging", "night_club", "casino", "
     movie_theater", "stadium", "bowling_alley", "zoo", "campground", "park"]
    industry_10 = ["finance", "accounting"]
    service_11 = ["atm", "bank", "police", "gas_station", "fire_station", "airport"]
    selfcare_12 = ["beauty_salon", "spa", "hair_care", "gym"]
    tourist_attraction_13 = ["tourist_attraction", "museum", "art_gallery"]
```

Listing 5.24: Supertypes model type groupings.

The demographic values, in this case, are set less liberally than in the granular model, but still with a minimum value of 5 %. The relevance value is set to 1 for all supertypes. Figure 5.34 shows that the model conserves the overall trends of the granular model but performs better at identifying multiple hotspots for particular demographics. This can be due to more consistency in the assignment of percentage value as a result of the grouping of individual place types.

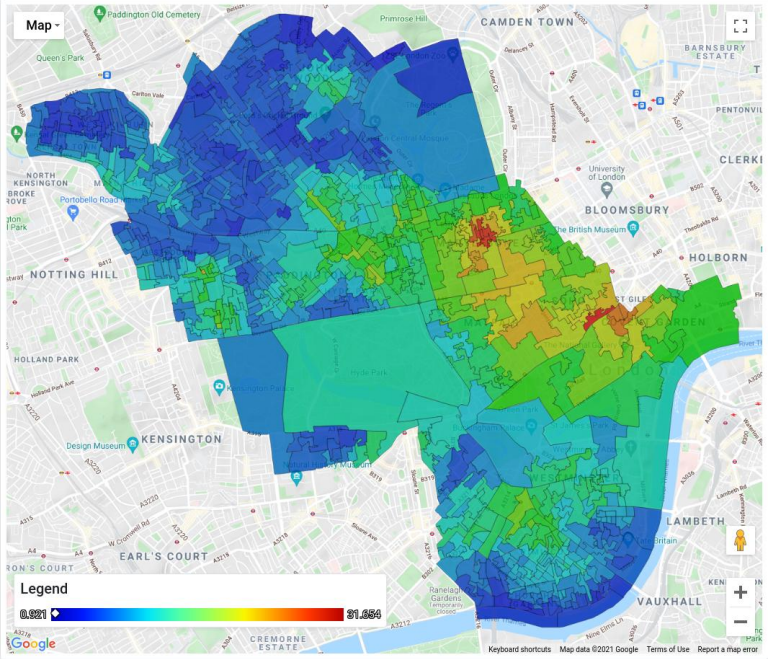


Figure 5.34: Supertypes model total population distribution estimate.

5.5.3.3 Supertypes attractors model

The attractors model experiments with more elaborate model building techniques. It splits the place supertypes into Attractors and Services. Attractors are all the places that force or highly motivate their target audience to travel to them. They also lack the ability to relocate or adapt to their audience's behaviour. This includes offices, tourist attractions and shopping

malls. Services are places that do not create people traffic by themselves but can reach out to their target audience and cater to their needs. Here we have most of the hospitality industry. As Listing 5.25 shows, this model attempts to estimate demographic distributions without the influence of any Service supertypes (which just includes *hospitality_7*). It does this by using a relevance function that assigns a value of 0 to all non-Attractor supertypes. This has the effect of negating any influence those types have on the final result values.

```
# Place supertypes attractor/service grouping.
ATTRACTOR_SUPERTYPES = ["religion_0","store_1","chore_2","academia_3","medicine_4","
 transport_5","generals_7","legal_8","attraction_9","industry_10","service_11","selfcare_12",
  "tourist_attraction_13"]

# Assign a relevance score of 1 to attractor supertypes and 0 to all others.
def assign_relevance_score_attractors(x):
    if x in ATTRACTOR_SUPERTYPES:
        return 1
    return 0
...
```

Listing 5.25: Attractors supertype grouping and relevance function.

The model results in Appendix section C.2 reveal the effects of this change. The areas of high hospitality concentration are much less prevalent in all demographic maps, shifting the focus onto other key areas and making them more visible. Tourists and leisurers retain the high hospitality areas as their single hotspots, indicating that those areas also contain other influencing factors. This causes their distributions to narrow in range and reveal other potential areas that without considering hospitality, appear more appealing to these demographics.

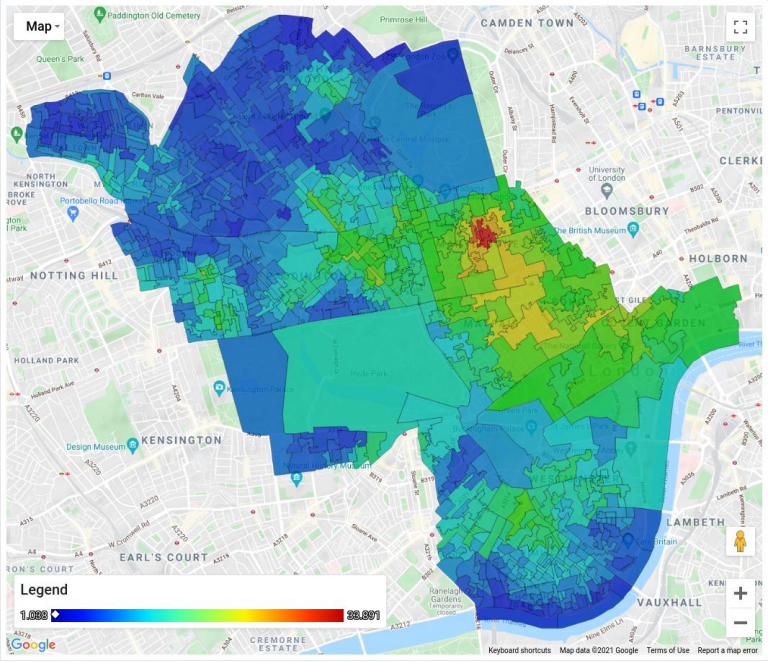


Figure 5.35: Supertypes attractors model population total distribution estimate.

The model's total distribution map in Figure 5.35 features the fading out of Westminster's hospitality epicentre around the Soho, Covent Garden and Mayfair areas. This proves that simulation of hypothetical scenarios is doable.

5.5.3.4 Supertypes discriminant model

The discriminant model is the best estimate of Westminster's population put forward in this project. The name comes from its demographic distribution by place value assignment strategy, where demographics are reluctant to act outside their principal activities. An example of this is the *legal_8* supertype setting the worker percentage to 95 % and all others to 1 %. Discriminant scraps the Attractors/Services segmentation and instead uses the relevance by supertype mapping shown in Listing 5.26.

```
# Relevance value assignment function for the supertypes discriminant model.
def assign_relevance_score_discriminant(x):
    SUPERTYPE_RELEVANCE_MAP = {
        "religion_0":2,
        "store_1":1,
        "chore_2":1,
        "academia_3":2,
        "medicine_4":1,
        "transport_5":1,
        "hospitality_6":0.75,
        "generals_7":0.5,
        "legal_8":2,
        "attraction_9":2,
        "industry_10":3,
        "service_11":1,
        "selfcare_12":1,
        "tourist_attraction_13":3
    }

    return SUPERTYPE_RELEVANCE_MAP[x]
...
```

Listing 5.26: Discriminant supertype relevance function.

The combination of these two strategies allows the model to not have to ignore very influential types, while at the same time stopping them from converging all the distributions together. As the results in Appendix section C.4 show, hospitality dominates tourists and leisurers, while still proportionally influencing all other classes accordingly.

This model produces discriminant distributions for all demographics. Each one is different from all others in both the spread and epicentre location, which is a good indicator that each demographic is being modelled independently. The results also match more precisely with the real world. The students' choropleth map is a good exhibit of this. It correctly simulates the broad spread of schools of all levels across Westminster, with its hotspot correctly laying over the campuses of Westminster University.

Another important feature of the discriminant model is that it is tailored to roughly match the reported number of workers in Westminster, estimated at 558,900 [14]. By adjusting the model, it is able to estimate 502,350 workers with the shown distribution in Figure 5.36 (in demographic counts per effective area square meter). Although this adjustment for population value does not have to affect the resulting distributions, it might be a great indicator of the optimal model parameter values.

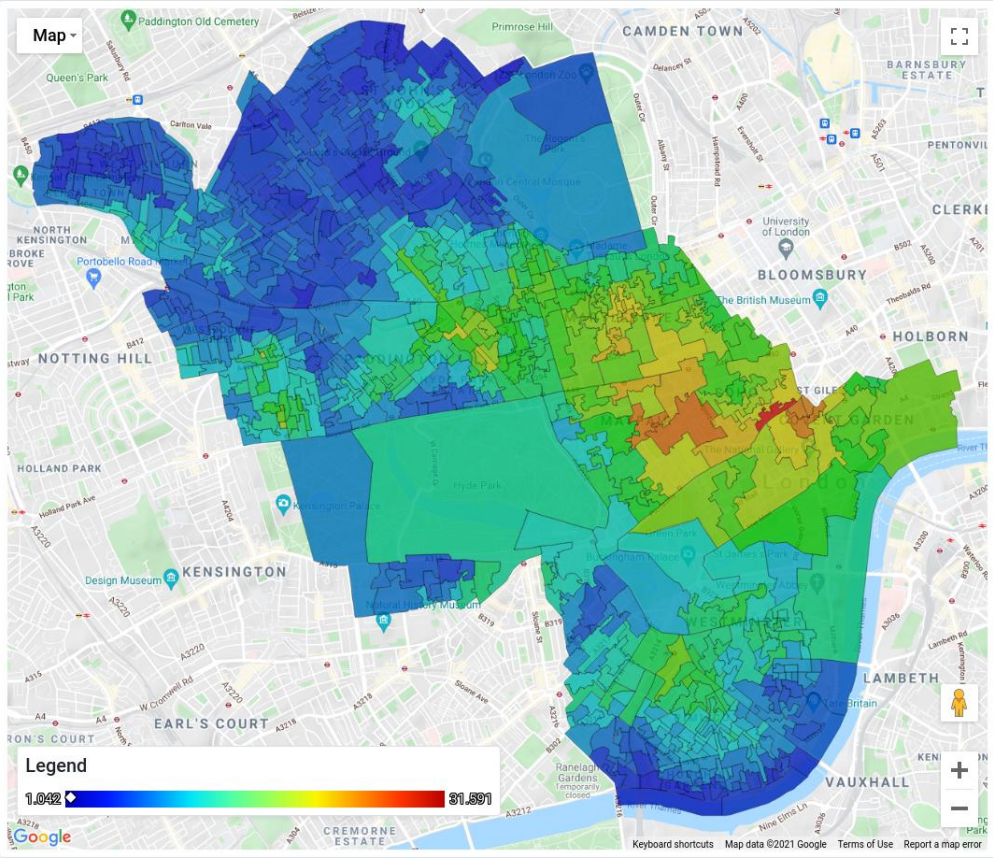


Figure 5.36: Supertypes discriminant model total population distribution estimate.

5.6 Total borough population estimation

The total population dataset combines all six visitor demographic populations with the resident's population to produce a total population distribution estimation. These results are shown in Figure 5.37.

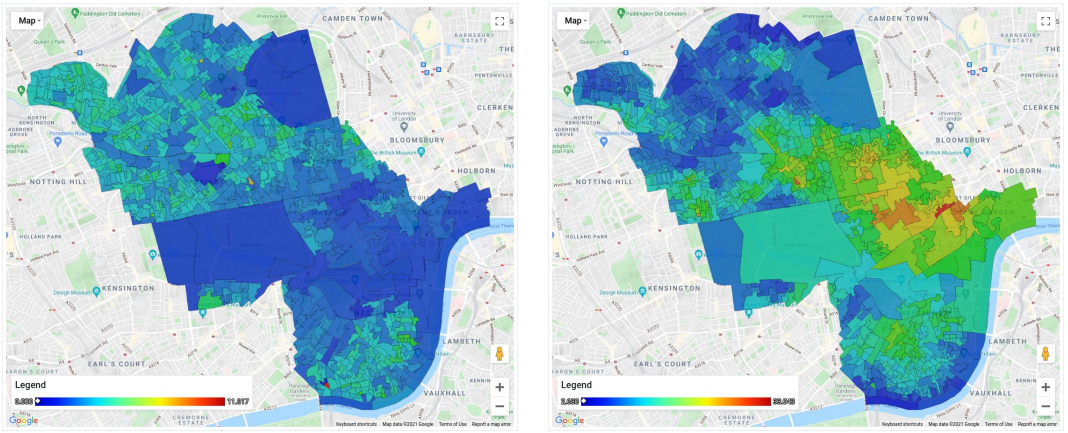


Figure 5.37: 24 hour population per effective area: Residents (left) vs Total (right).

5.7 Supply and demand indicator metrics

The processed data and information described thus far is highly versatile and has plenty of applications. At the request of WCC, new business optimal positioning metrics is the city planning problem tackled in this project. Two methods were developed to investigate the relationship between the supply of services in an area and the demand of all or any combination of audiences. Out of the available 103 types of places, the supply and demand dataset only includes bars, cafes and restaurants, as these are the most common business license types in Westminster.

5.7.1 Supply and demand distribution metric

This metric is computed by the process shown in Listing 5.27. The distributions per effective area square meter of places and demographics are normalized to a range of 0 to 1. The normalized demand value is then subtracted from the normalized supply value for each OA.

```
    # Normalize them all out of 1.
    for col in cols_to_keep:
        if col != "OA":
            col_min = df[col].min()
            col_max = df[col].max()
            df[f"[normalized_to_[0-1]] - {col}"] = ((df[col] - col_min) / (col_max - col_min))
    ...
    # Calculate distribution differences.
    for type_col in ["bar", "cafe", "restaurant"]:
        for demo_col in DEMO_TYPES:
            df[f"[supply - demand] - [normalized_[0-1]_proportions] - {type_col}_{demo_col}"] =
     df[f"[normalized_to_[0-1]] - {type_col}"] - df[f"[normalized_to_[0-1]] - [
     per_effective_area_square_meter] - {demo_col}_count"]
```

```
...
```

Listing 5.27: Supply and demand distribution metric calculation.

The results are not very mathematically significant because they are simply juxtaposing the normalized distributions of two variables. There is no clear indicator of what level of supply satisfies what level of demand and the metric can be affected by outliers and very differently shaped distributions. What the metric is good at, is visualizing and comparing the interactions between the distributions of the selected place type and demographic in a single map. As shown in Figure 5.38, colder colours indicate areas of high concentration of the demand demographic and low concentration of the supply place type. Values close to 0 indicate equivalent concentrations.

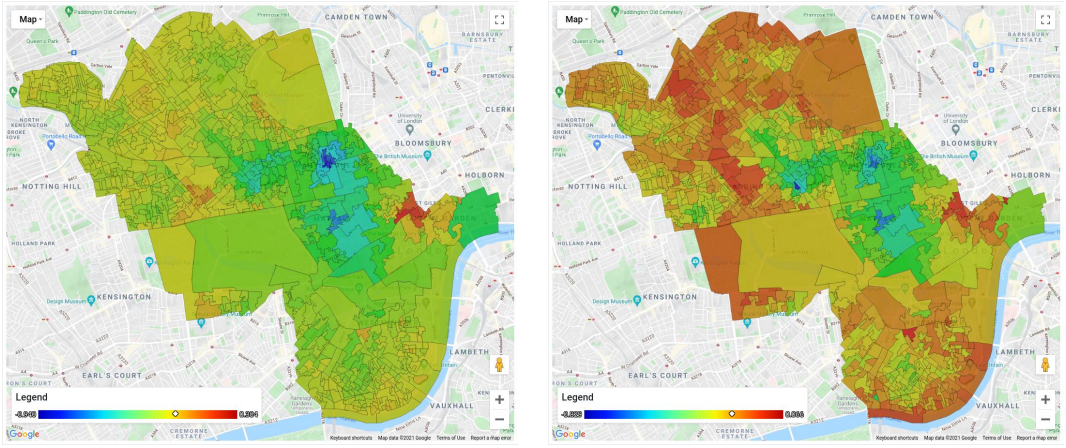


Figure 5.38: Supply - Demand distribution metric: Cafes and workers (left) and Cafes and total population (right).

5.7.2 Supply and demand index metric

This metric is proportion based. The demographic count by effective area square meter in each OA is divided by the place count by effective area square meter. The divisor in this case, as noted in Listing 5.28, is thresholded with a minimum value of 0.02 to restrict outlier values.

```
# Calculate supply and demand index.
for type_col in ["bar", "cafe", "restaurant"]:
    df[f"{type_col}_or_limit"] = df[type_col].apply(lambda x: max(x, 0.02))

for type_col in ["bar", "cafe", "restaurant"]:
    for demo_col in DEMO_TYPES:
        df[f"[supply_demand_index] - {type_col}_{demo_col}"] = (df[f"[
 per_effective_area_square_meter] - {demo_col}_count"] / df[f"{type_col}_or_limit"])
...
```

Listing 5.28: Supply and demand index metric calculation.

This method is not designed to show the separate hotspots of supply and demand, but it does indicate areas of proportionally high demand. Figure 5.39 encodes areas of high demand in warmer colours. Cold colours indicate areas where there is low demand or demand is being met by supply. Mathematically speaking, this method is more robust than the previous. It does not necessitate information about what specific level of supply would satisfy demand because it is simply plotting the results of the index computation for all the OAs. If an OA where supply and demand are balanced is identified, then all the OAs with higher index values can be classified as having demand, and all others as not having demand. Without this information, the OAs highlighted in red are simply the most likely areas to exhibit demand.

The results shown do align with the real world as adding the resident population triggers demand for cafes in numerous residential areas with low counts of cafe businesses.

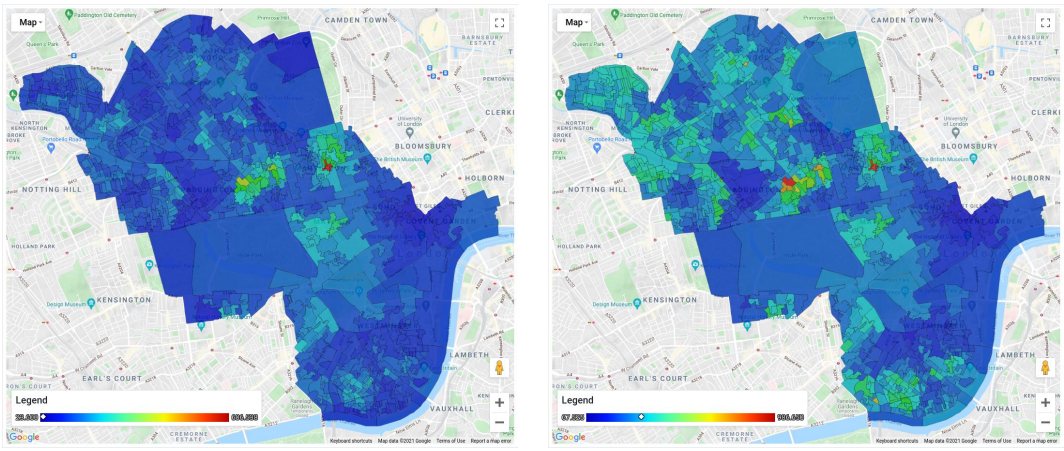


Figure 5.39: Supply and Demand index metric: Cafes and workers (left) vs Cafes and total population (right).

5.8 Dataset metadata file

The dataset metadata file is a system design feature necessary to allow the user interface to visualize large amounts of data and guarantee application performance. It achieves this by providing the UI with metadata information, accessible in constant time lookups, that would otherwise require the processing of entire datasets to retrieve. It also facilitates the integration of new datasets into the system, since the way the data is processed by the UI and visualized can be controlled through changing metadata file entries either by code or by hand. Details are described in this section.

5.8.1 Column typing and range definition

The metadata file contains an entry for each dataset and each column. Listing 5.29 shows the PTAL dataset with its two columns. The *type_registered* associated with each column is the type assigned to it by the Pandas library when converted to a dataframe. The *type_practial* is a custom field that can take any value, usually, something relevant to the user interface's data loading module. Numerical columns get assigned minimum and maximum values, while categorical ones get a range of labels.

```
...
{
    "[OA]_PTAL_directory.csv": {
        "column_data": [
            {
                "name": "Public_Transport_Accessibility_Index",
                "type_registered": "float64",
                "type_practical": "number_float",
                "range": null,
                "min": 3.918,
                "max": 79.42
            },
            {
                "name": "Public_Transport_Accessibility_Level",
                "type_registered": "object",
                "type_practical": ["PTAL", "default"],
                "range": ["1b", "2", "3", "4", "5", "6a", "6b"],
                "min": null,
                "max": null
            }
        ]
    }
    ...
},
...
```

Listing 5.29: Dataset metadata file snippet.

In the user interface, this file is loaded as a (hash)map and it serves to populate the dataset and column selector dropdowns, generate the appropriate scales, and populate legends with labels or range values. The latter two operations done for the more than 400 columns in the final datasets would impact performance significantly, but with the metadata file, all the necessary information is accessed in constant time, leading to a one-time loading wait of around 3 seconds upon startup.

5.8.2 Shared scales

The feature of shared scale columns is made possible by the metadata file. For a dataset, all columns prefixed with *[shared_scale]* have their min and max values set to the min and max of the group. When the user interface generates scales for these columns, they will all have

the same range and encoded colours. Figure 5.40 highlights shared columns in the Population dataset.

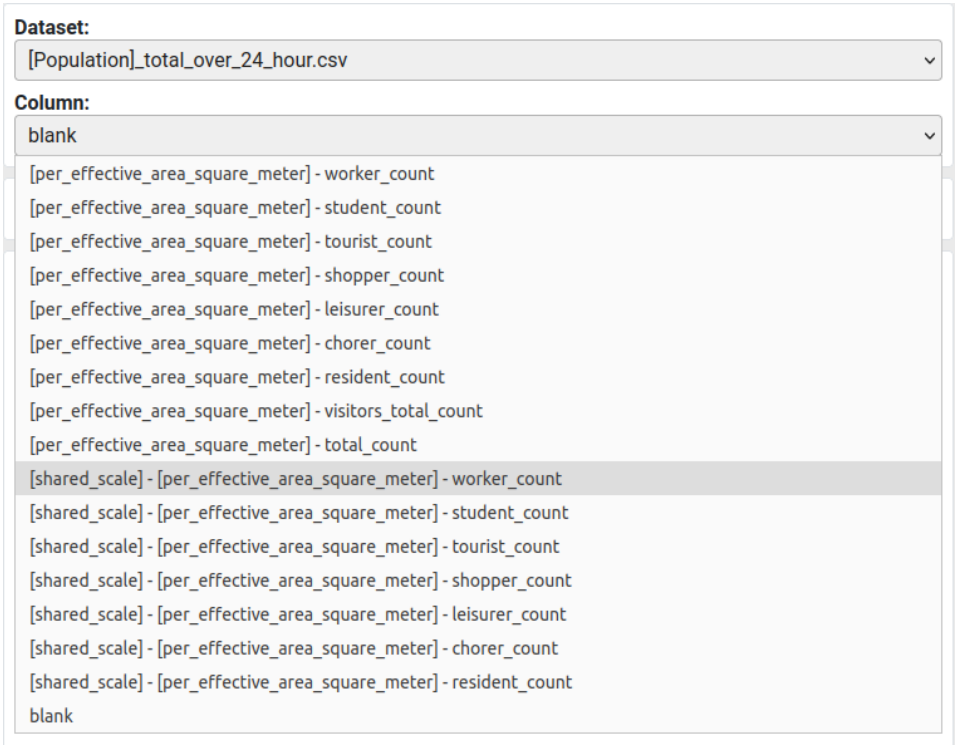


Figure 5.40: Shared columns in column selector dropdown menu.

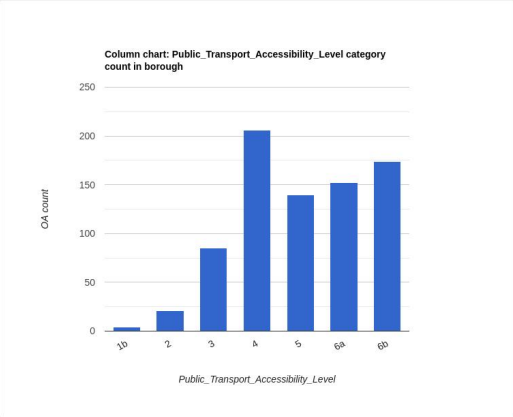
5.8.3 Column hiding

The metadata file is also used to hide columns from the user's view but retain them in their datasets for use in behind-the-scenes components of the user interface. Most categorical columns are accompanied by a frequency count column that is not visualizable in the choropleth map but is used for generating auxiliary visualizations. Listing 5.30 shows how such a column is hidden in the PTAL dataset.

```
# Columns to be excluded from the metadata file.
DATASETS_SKIP_COLUMNS = {
    "[OA]_Normalizing_properties.csv": [],
    "[OA]_PTAL_directory.csv" : ["Public Transport Accessibility Level_frequency_count"],
    "[OA]_Street_value_directory.csv" : ["Banding Description"],
...
}
...
```

Listing 5.30: Metadata file column hiding.

Figure 5.41 shows the frequency count column being used to generate a side category count and a data table when OAs are clicked and one of the two accessible columns is selected.



Property	Value
OA	E00175206
Public_Transport_Accessibility_Index	35.842
Public_Transport_Accessibility_Level	6a
Public Transport Accessibility Level_frequency_count	{'6a': 93, '6b': 26, '5': 5, '3': 3, '1a': 2, '2': 1}

Figure 5.41: Column hiding in the PTAL dataset: Histogram (left) and Data table (right).

Chapter 6

Interactive Data Visualization and User Interface Implementation

6.1 User interface

The user interface features the layout in Figure 6.1. Components match the UI structure diagram presented in Chapter 4.

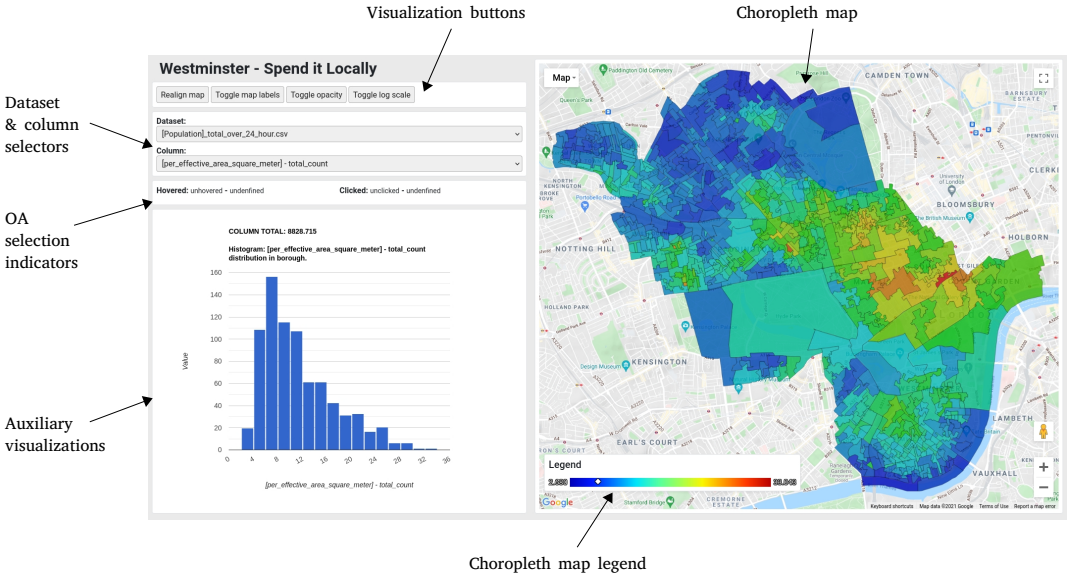


Figure 6.1: Google Maps based user interface layout.

6.1.1 Dataset scale & legend generation

An important feature of the user interface is its ability to dynamically produce scales and legends that match the range of the data visualized through the choropleth map. This makes these visualizations easier to analyse and it also enables new datasets to benefit from this feature without any additional work.

D3 scale objects are used to handle scales in the system as they have built-in functionality to deal with unknown values, automatically map domain objects to appropriate colours and expose functions to perform this computation on demand. As shown in Listing 6.1, scale objects are built depending on the *type_practical* of columns defined in the metadata file.

```
...
if (c["type_practical"] === "string") {
    // Categorical scale.
    scale = d3.scaleOrdinal()
        .domain(c["range"])
        .range(d3.schemePaired)
        .unknown(blankPathFill);
} else if (c["type_practical"] === "number_int" || c["type_practical"] === "number_float"){
    // Numeric continuous scale.
    scale = d3.scaleLinear()
        .domain([c["min"], c["max"]])
        .range([continuousLinearLow, continuousLinearHigh])
        .unknown(blankPathFill);
...
```

Listing 6.1: User interface dynamic scale generation.

Some categorical columns require specific colours and labels. These are defined in the *projectColours.js* file under the data structure shown in Listing 6.2.

```
// Dataset column label and color mapping.
const colourMappingsSimple = {
    "PTAL":{
        "default":{
            "6b": {
                "colour": "#b62419",
                "name": "6b"
            },
            "6a": {
                "colour": "#ff3f35",
                "name": "6a"
            },
...
```

Listing 6.2: User interface colour mapping definition.

Numerical columns are visualized on the tricolour scale seen before. It includes 10 steps ranging from blue to teal, to green, to yellow to red. Compared to the continuous scale of the D3 project (Figure 6.4), it is a significant improvement. The changes in hue and tint, combined with the larger dynamic range allows it to visually distinguish a larger range of values, thus making the choropleth maps easier to digest by human eyes.

All scale information is stored in the global *Scales Map* object. It maps each dataset and column combination to a scale object, shown in Figure 6.2. This object is constructed once upon startup and accessing it is constant time.

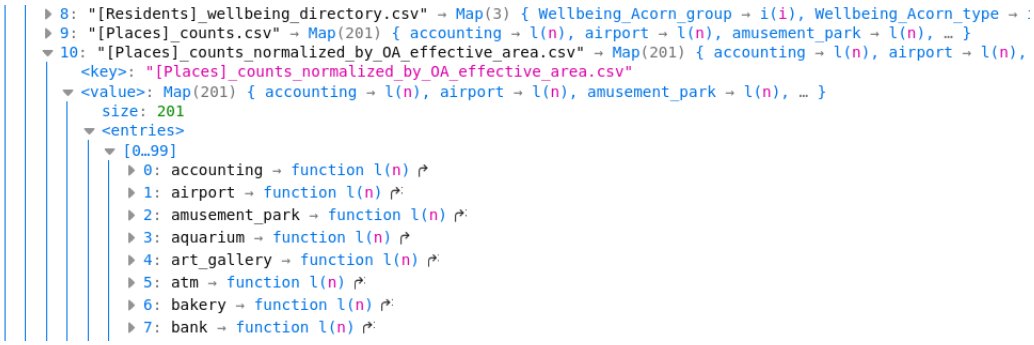


Figure 6.2: User interface Scales Map.

6.1.2 Internal data storage object structure

The *Data Map* object organizes data in a way such that the user interface runs smoothly while handling over 400 columns that are visualizable on demand. During the loading process, all tabular datasets are combined and stored in this map object indexed by: OA code, dataset name, column name and mapping to a single column value. Figure 6.3 shows a snippet of this data structure.

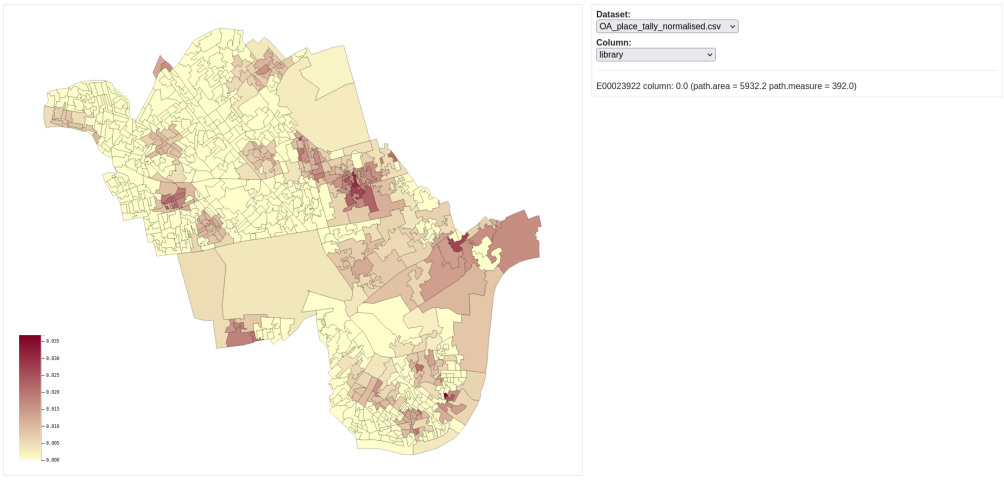
Figure 6.3: User interface Data Map.

The way data is accessed can now be constant if retrieved using the indexing pattern provided. Most data accessing operations adhere to this strategy, including the choropleth maps, the legends, the selected OA indicators and the auxiliary visualizations. Furthermore, the fact

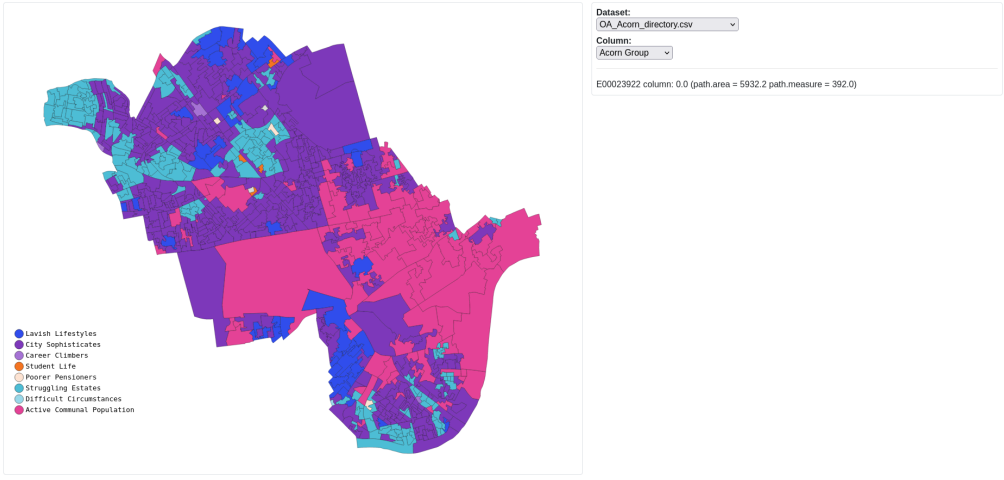
that this object is persistent in memory during execution avoids the reloading of datasets and further improves performance.

6.2 Interactive data visualization

As mentioned before, an alternative application made in D3 was also created to evaluate its suitability. Figure 6.4 presents sample visualizations of numeric and categorical columns. The numerical scale uses a different bicolour colour mapping, and the toolbox div is placed on the opposite side of the choropleth map.



(a) Numeric column



(b) Categorical column

Figure 6.4: D3-based user interface layout.

The choropleth maps generated using D3 lacked the Google Maps labelling and terrain features and therefore were not used in the final solution. The rest of the application, however, makes use of D3's backend functionality like file loading, the update pattern for managing dropdown menus and scale objects.

6.2.1 Choropleth map visualization

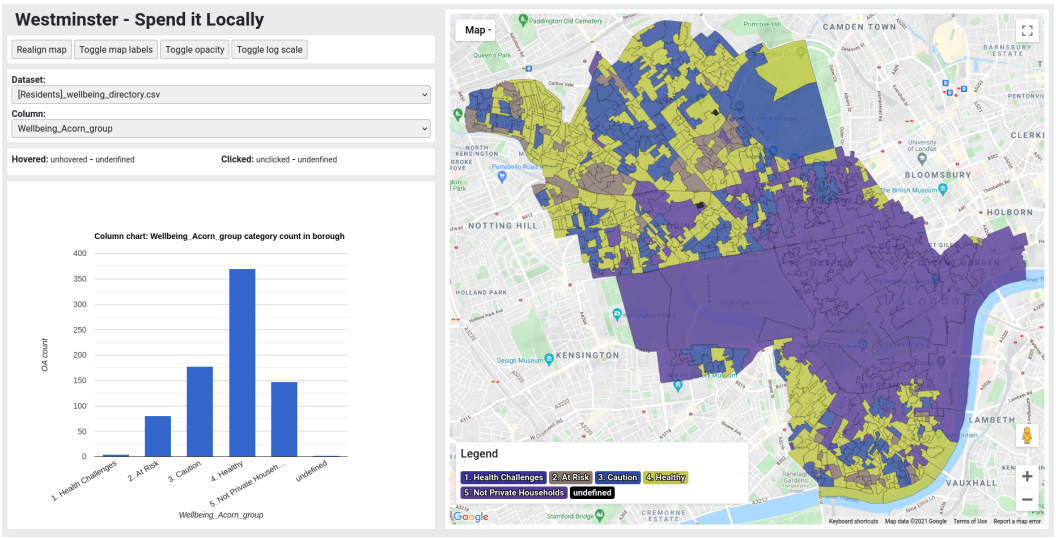
The authority level chosen for the Westminster map is OA. The lower level authority is postcode, but the borough contains over 30,000 of them and visualization was not practical. The higher-level authority is wards, of which there are 20 in the area. This count is too low and would cause aggregation results to be broad, lose too much detail and become unusable.

Choropleth maps are generated by setting the data property of a Google Maps map object to the Westminster OAs TOPOJSON file. The polygon colours are then applied via styles. Listing 6.3 shows the *styleContinuous* method used for applying numerical colour overlays, and the *styleCategorical* used for categorical columns. Both of these are also tasked with generating the appropriate legends.

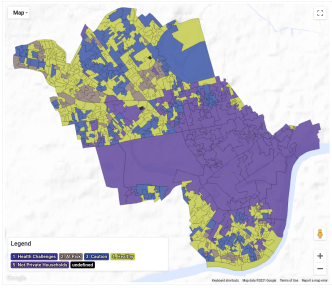
```
...
/*
Populates the map with the geojson boundary data and applies the default style.
 */
function populateMap(map, data) {
    map.data.addGeoJson(data);

    map.data.setStyle(function(feature) {
        ...
        return ({
            fillColor: color,
            strokeWeight: strokeWeight,
            strokeColor: strokeColor,
            strokeOpacity: OPACITY,
            fillOpacity: OPACITY,
        })
    })
}

/*
Style applied to the map when a column of type numeric continuous is selected. It uses the
 blue-turquoise-green-yellow-red continuous scale in the legend.
 */
function styleContinuous(feature){...}

/*
Style applied to the map whenever a categorical typed column is selected. Colors depend on
 the OA values and their respective color mappings.
 */
function styleCategorical(feature){...}
...
```

Listing 6.3: User interface choropleth map operations.

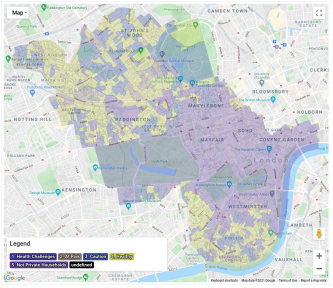
The visualization control buttons at the top of the toolbox are used to change the appearance of both the choropleth map and the auxiliary visualizations. They can recenter the map, remove Google Maps information overlays, reduce the opacity of the polygon colour overlays and change the numeric axis of any currently displayed side visualization to a logarithmic scale. These are all very useful when doing manual data analysis on the user interface. Figure 6.5 shows them in action.



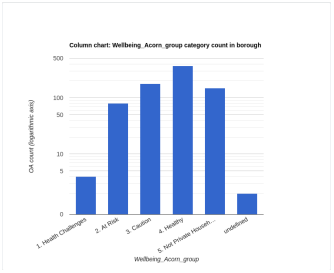
(a) Starting state



(b) Toggle map labels



(c) Toggle opacity



(d) Toggle log scale

Figure 6.5: User interface visualization control buttons.

6.2.2 Auxiliary visualizations

If the choropleth map offers a view of all individual OAs in the borough, auxiliary visualizations offer a view of all the data attached to a single OA. Practically, they can perform any data operation possible, but the *Data Map* indexing pattern is generally followed. Through side visualizations, data is visualized in formats other than a choropleth map. These include histograms, bar charts, pie charts and deviations of these. Some unseen auxiliary visualizations

in this report so far are shown in Appendix D.

The UI uses a regular expression case statement to map combinations of datasets, columns and selected OAs to methods in the *anon* file that use the Google Charts library to generate the visualizations. Regexes are used to drastically reduce the number of statements needed, as column names can be matched in groups. A snippet dealing with the Places dataset is shown in Listing 6.4.

```
/*
Triggers the rendering of a different auxiliary visualization for any combination of
 selected dataset, column and OA.
*/
function applyToVisualizationMap(key) {
    switch(key) {
        // Places datasets.
        case (key.match(/\[Places].*#\[shared_scale].*#yes_oa/) || {}).input:
            dataTableSelectedOA()
            break;
        case (key.match(/\[Places].*#.*#yes_oa/) || {}).input:
            barChartPlacesSelectedOA()
            break;
...
```

Listing 6.4: Auxiliary visualizations mapping.

Chapter 7

Results Analysis and Evaluation

The results of the data science process (in the context of the problem statement) and the deliverable data science application are evaluated in this chapter by recalling the established system requirements from Chapter 2 and reviewing the findings in the scope of Westminster and the associated literature. For more detailed results and analysis on individual system components, refer to Chapter 5.

7.1 Functional requirements

All functional requirements were met.

Data science

- ***F-DS 1:*** *Process and visualize all provided resident datasets:*
  All the datasets containing resident data from the company CACI are processed and visualized.
- ***F-DS 2:*** *Collect, process and visualize Places data from Google Maps Places API:*
  Google Maps Places data is mined through Python scripts, normalized in various ways, and visualized.
- ***F-DS 3:*** *Derive, process and visualize demographic distributions for each demographic:*
  Four alternative models are put forward that estimate the demographic distributions in Westminster. All visualized in the user interface.
- ***F-DS 4:*** *Derive, process and visualize all relevant population demographic totals:*
  An estimate of Westminster's total daytime and worker population is used in the su-

pertypes discriminant model to not only estimate demographic distributions but also demographic counts.

- ***F-DS 5:*** *Derive, process and visualize supply/demand indicator data:*
  Two methods revealing the dynamic between supply and demand are implemented. Referred to as distribution and proportion-based.

- ***F-DS 6:*** *All data visualization instances have a geographical component:*
  All dataset columns and thus all selectable information in the user interface is linked to an OA, which has a geographical component and therefore is visualizable in the choropleth map.

User interface

- ***F-UI 1:*** *The system is web-based:*
  The user interface is a pure Javascript application run on a browser.

- ***F-UI 2:*** *Contains visualizations for all dataset-column combinations available:*
  All columns are visualizable and visualized through the choropleth map and auxiliary visualizations, although some are more complex than others.

- ***F-UI 3:*** *Geographical visualizations are interactive:*
  The choropleth map supports zooming, panning, label toggling, polygon overlay opacity toggling, OA selection via hovering and also OA selection via clicking.

- ***F-UI 4:*** *Non-geographical visualizations are interactive:*
  Auxiliary visualizations have interactive elements. Hovering highlights chart elements and clicking displays further information.

- ***F-UI 5:*** *Geographical visualizations support continuous and categorical legends:*
  Numerical legends adjust to the range of the current column and the categorical legends are built through label and colour mappings.

- ***F-UI 6:*** *When applicable, presents data in a borough and OA context:*
  The choropleth map usually presents the borough view and the auxiliary visualizations cover the OA view. However, when appropriate, one or both of these patterns are broken.

- ***F-UI 7:*** *When applicable, visualizations can make use of dynamic scales:*
  Columns in the choropleth map generate legends that dynamically adjust to the column's range. Auxiliary visualizations also modify axis ranges to improve chart legibility.

- ***F-UI 8:*** *When applicable, dynamic scales can be disabled:*

  In certain scenarios, columns are paired with shared scale versions of themselves. These force both the choropleth map and side visualizations to use the defined group range.

7.2 Non-functional requirements

All non-functional requirements were met.

- ***NF 1:*** *The data processing pipeline is executed fully in under 6 hours:*

  The estimated execution time of the data processing pipeline is 1 to 1.5 hours, with most of that time being taken up by the mining of the Places dataset.

- ***NF 2:*** *The user interface application runs smoothly on a browser on an average machine:*

  The use of the metadata file, the data map and the scales map create a highly responsive user interface.

- ***NF 3:*** *An attempt has been made to delegate processing away from the web application:*

  The Python-based processing pipeline performs data reduction operations, computes deliverable columns and generates the metadata file to avoid the user interface having to do as much computing as possible.

- ***NF 4:*** *An attempt has been made to make the user interface intuitive to use:*

  The UI is a single page application with only eight interactable components, the layout is static and front-end web frameworks were used to present a modern and familiar appearance.

- ***NF 5:*** *The system can process and visualize new datasets relatively easily:*

  Placing new datasets in the correct folder and adding this path to the list of visualizable datasets is all that is necessary.

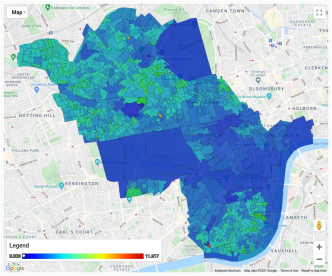
- ***NF 6:*** *The system is suitable for automation and deployment:*

  The user interface is already in a deployable format (web application). The data processing project builds a sequential pipeline that lends well to deployment and usage through cloud processing providers like Amazon's AWS or Microsoft's Azure.

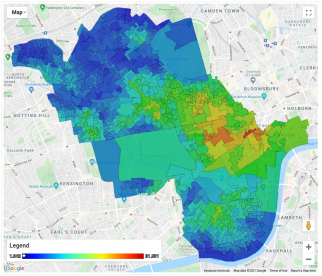
7.3 Established problem statement solution

The provided application contains data that could fuel analysis in many areas and at various depths. Focusing on the problem statement, the daytime population of Westminster is subdivided into six subpopulations and their distributions estimated. Their combination produces models showcasing the borough's total population distribution as well as the respective demographic people counts. The models are created using the counts of places of each OA and model files that specify the demographic breakdown of each place type. These files are user parameters and allow the population models to be customizable. This can lead to simulations of the current state of the borough or the exploration of hypothetical scenarios.

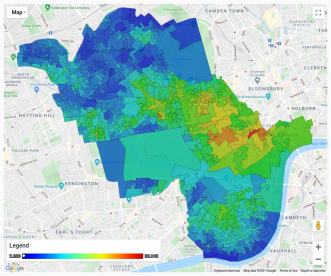
Model accuracy is difficult to gauge, as such analysis in the borough of Westminster or any authority including Westminster is not present in the publicly available literature, with a slight exception. Figure 7.1 presents the total population estimation given in the 2018 City Profile report. No information about how it was computed is provided, just a breakdown of the nighttime, daytime and combined population visualized in Westminster maps divided into wards. These results concur with the ones presented in the project, and the comparison highlights the benefit of having a lower level subdivision of the sample area. OAs in the project choropleth maps more accurately depict the distribution within the much larger ward polygons.



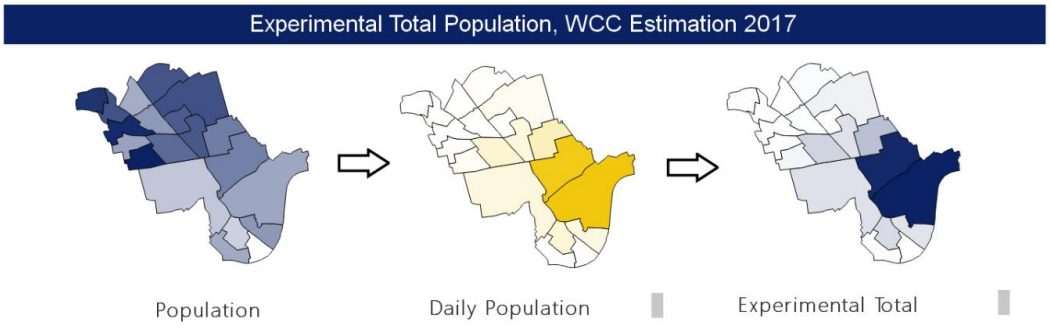
(a) Nighttime population



(b) Daytime population



(c) Total population



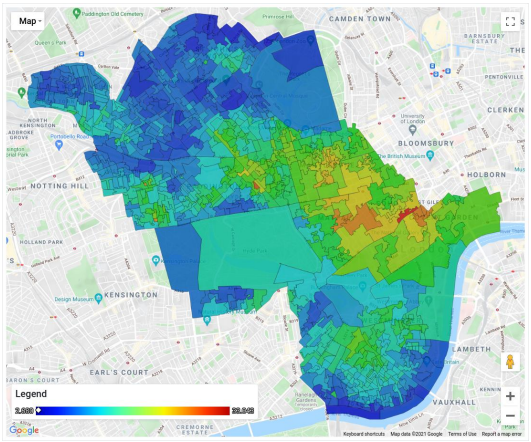
(d) City Profile 2018 population estimation. Sourced from 2017 data

Figure 7.1: Population estimation comparison with the 2018 City Profile report.

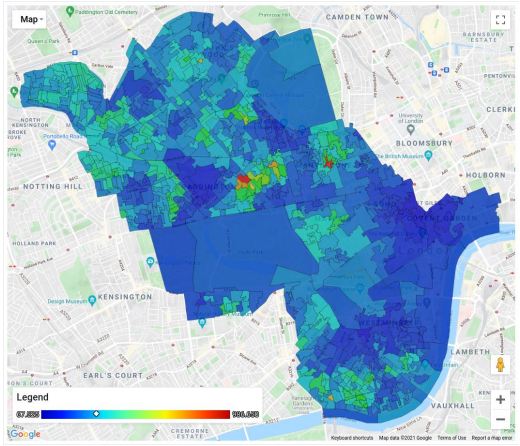
Despite the lack of similar works, it is possible to extrapolate further conclusions from related studies and on the grounds of general city knowledge.

In the decomposed demographic distribution maps of the discriminant model in Appendix C.4, the Student distribution correlates closely with that of academic centres in Westminster. The distribution of Shoppers has a large epicentre in the Oxford Street area, which is recognized as the busiest shopping street in Europe. The spreads of Leisurers and Tourists are expectedly similar in their concentration around the Soho and Covent Garden areas, however, Tourists have a noticeable bias towards riverside attractions in the north bank and sightseeing spots parallel to The Mall leading to Buckingham Palace and Hyde Park Corner. Leisurers seem to be more focused around the Leicester Square area with alternative preferences for less touristic areas like Bayswater and Pimlico. Workers showcase a bimodal distribution, with hotspots around Mayfair (standing for upscale retailers and the financial sector), and Marylebone (containing numerous hospitals, health care centres and a thriving high street). Interestingly, many other well defined active areas are distinguishable, including The Strand, Victoria, Paddington and St John's Wood; perhaps indicating the natural clustering of workplace hubs. Lastly, Chorers appear to be reasonably well spread out in the borough, with their hotspot over the Edgware Road area. This is can be an indicator that the area contains a high concentration of services and tradespeople, perhaps due to lower operating costs in a relatively central location.

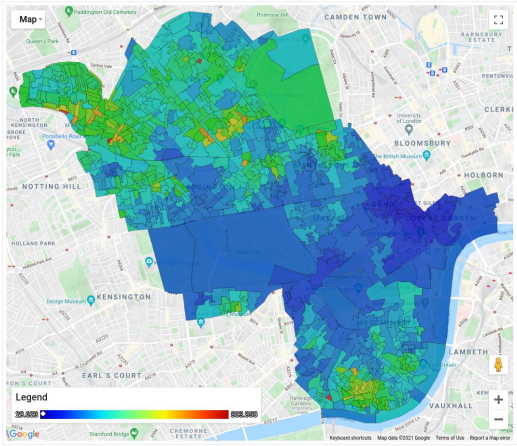
In regards to the optimal location metrics for new establishments, considering map (a) in Figure 7.2, the total population distribution combines the preferences of the aforementioned demographics, plus the resident's population. The supply and demand index metrics highlight optimal places for new businesses in warm colours. Areas of high index values and low population indicate low demand and very low supply, whereas areas of low index values and high population indicate high demand and plentiful supply. These are both plausible socio-economic situations for a business to flourish in and a decision for an owner to make: quiet and low competition vs busy and high competition. A Goldilocks scenario in the majority of cases would be an area of high index value and high people traffic. Such areas are highlighted in map (b) where a relatively high population density and low concentration of cafes manifest around Hyde Park Estate, Marylebone, Mayfair, Lisson Grove and Churchill Gardens.



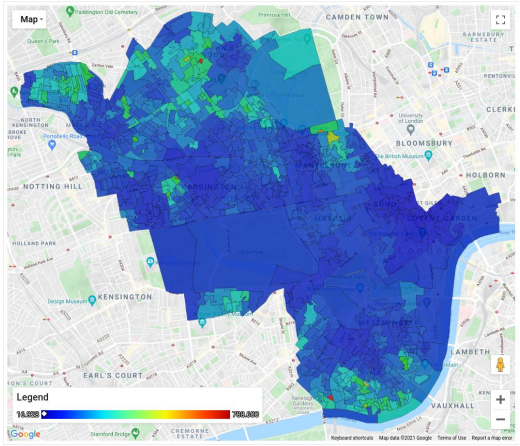
(a) Total population distribution



(b) Supply and demand index for cafes



(c) Supply and demand index for bars



(d) Supply and demand index for restaurants

Figure 7.2: Supply and demand index for the total population distribution estimate.

Restaurants follow a similar trend to cafes but with fewer recommended areas, indicating a lower demand for this type of establishment or most likely, a better-served niche of the hospitality industry. Bars produce results that are perhaps uninformed by the nature and audience of such places. A more accurate supply and demand index distribution can be obtained by looking at individual demographics rather than the total population. This also applies to the other place types.

Comparing the data science pipeline results of the residents' data to those presented by the sources that study Westminster's local population ([19], [22] and others) leads to an agreement in the vast majority of conclusions. After all, the source data originates from UK census data of recent years which is not likely to change dramatically. The studies touch on the population makeup, the local economy, education status, transport accessibility, wellbeing factors and

more. For the topics covered in this report, the system generates arguably more accurate results than the literature as aggregation areas are set to more granular OAs rather than wards. Furthermore, the visualization devices of interactive choropleth maps and auxiliary charts are superior to the static report format used in the literature. In terms of content, the same conclusions are presented, leading to the verdict that the processing and visualization of resident's data in the CACI datasets is done effectively.

7.4 System limitations

Although the project satisfies its requirements and proposes a reasonable solution to the problem statement, the constraints of resources and time create numerous areas for improvement.

7.4.1 Static population estimate

The population estimate although it is divided into nighttime and daytime subpopulations, lacks a greater resolution in the temporal dimension. The estimate is given as a total over a 24-hour period which limits the usefulness of the results as many applications require a more granular breakdown of population by hours or even minutes. This also causes the potential analysis of people movements to be impossible to simulate.

7.4.2 Mobility data

In the literature, Places data is regularly used as an enhancing feature to the base estimation done with mobility data. Although the aim of this project is to produce an estimate without the use of mobility data, the model's accuracy can be evaluated and perhaps improved with the addition of such data. People traffic per OA can become another covariate used in the computation of demographic distributions.

7.4.3 Google Maps Places API results capacity property

The Places API entries do not contain clear information about the size and capacity of establishments. These properties are currently modelled by the relevance multiplier and the normalization of the places tally, but these methods do not work based on individual places. The ability to set relevance multipliers for individual entries rather than for place types or supertypes can give more control over the model's behaviour. Currently, there is a trade-off being made in favour of convenience at the cost of accuracy.

7.4.4 Authority aggregation level

The map of Westminster is divided into OAs and all the data is aggregated as such. As discussed in Chapter 3, the disaggregation of data from their census areas is important for increasing its accuracy, visualization potential and analytical versatility. The method used in this project is the normalization of data by derived normalization factors like OA effective area. This produces good results, but introduces inaccuracies in the data based on the quality of the normalization factors. It also prevents the data from being visualized at lower granularities (like postcodes) as this would require a renormalization by the appropriate authority level.

7.4.5 Places dataset mining technique

Places are tallied by OA using areal search. All search areas are quadrilaterals that fully contain the OA being searched. Depending on the shape of the OA's polygon this method might over or underestimate the number of places that should be attributed to that OA (relative to the area it covers). This information retrieval and aggregation method could be described as somewhat unstable or over-dependent on polygon shape. The smoothing factor of 400 feet further worsens unstable estimates as it simply extends the rectangular search areas.

The smoothing buffer is another area for improvement. It is set at 400 feet to reduce the mining time and costs of the operation, but in reality, more than 120 meters is still within the wildly accepted range of walking distance and therefore warrants further analysis. Expanding it could result in the smoothing of OA hotspots and the removal of outliers.

7.4.6 User interface

The user interface is a candidate for numerous quality of life improvements but here are three suggested by WCC during a feedback session. Firstly, the visualization of multiple choropleth maps at the same time or the overlaying of multiple columns on the same map. Secondly, the selection of multiple OAs simultaneously and the dynamic rescoping of all visualizations to information regarding the selected group. Lastly, the generation of reports or other means to allow the sharing of visualization results.

Chapter 8

Legal, Social, Ethical and Professional Issues

The theory developed in this report and the practical application delivered have the potential to become commercially viable and be used by WCC or other authorities to inform urban planning strategies. In this case, additional work is required to ensure more comprehensive compliance with the Code of Conduct of the British Computer Society [25].

Although the project and its component parts have been manually verified and gone through multiple stages of user testing, it lacks a comprehensive automated testing suite that is essential to ensure the correctness, security and stability of the system. Given that the application contains sensitive data, derives information private to WCC, and mines potentially expensive datasets, it is in the interest of the public and of WCC that unauthorised access is restricted.

Before the system can become publicly hosted on the internet, it is required that the datastores are stored securely. This includes not loading visualizable datasets into the user interface as static resources. Currently, these are accessible through the DOM but ideally, they would be accessed through secured API calls.

In preparation for this undertaking, extensive research in the discipline of urban planning, Westminster as a case study, and in particular the topic of population estimation was done. Great effort was invested in the application of the scientific method, the verification of the obtained results, and the explanation of how they were computed. Supporting evidence is provided and alternative solutions are critically evaluated.

Credit is given to the authors of the inspiring literature and to the creators of the technologies that make up the system. In return, the source code and this report are made publicly

available. Code commenting and documentation files are provided to facilitate the reusability and understanding of the codebase.

This report clarifies the origins of the publicly available raw datasets used. The private CACI datasets were provided by WCC, who also supervised this project. The deliverable application aims to influence decisions that could mould the borough of Westminster. This can affect all its citizens, but only through the authority of WCC, who I submit to. It is recommended that all decisions based on the data produced by this system are supported by additional material and reviewed by official decision-makers.

Chapter 9

Conclusion and Future Work

The objective of this project was to build an information repository and visualization system that would allow Westminster City Council to analyse the socio-economic state of the borough and provide a strong evidence base to advise urban planning strategies aimed at closing the gap between retailers and consumers.

The produced system processes over 40 datasets and visualizes over 400 columns. The information contained provides a comprehensive breakdown of resident's demographic makeup, social behaviour and spending tendencies. The Places dataset classifies and geographically plots all amenities in the borough through 103 explainable place types. The demographic distribution models estimate the nighttime, daytime and total population distribution of Westminster split into six well-defined subpopulations. Lastly, supply and demand metrics establish an algorithmic recommendation system for the optimal positioning of new enterprises depending on the business type and the target audience.

The mined and aggregated Places data has stood out as an incredibly versatile and useful dataset. A catalogue of all establishments, geographical locations and points of interest, with additional metadata associated with all entries is invaluable in urban planning operations. Furthermore, such data might be difficult to access by WCC through other sources and in the best of cases, it will lack data enrichment features like place type classifications provided by Google Maps. The parameterised mining solution presented in this project is a great starting point for future developments.

The population models offered provide a detailed total population estimation, which by the accounts of the available literature, has not been done in Westminster before. The use of Places data instead of mobility or census information as the core of the estimate presents this method

as a potential alternative for the computation of daytime population distributions. The level of user input and customisation through demographic distributions by place files allows the models to take both a descriptive and a predictive role, which expands their range of use cases.

The supply and demand metrics are examples of prescriptive data science tasks that can be performed using the Places dataset and a comprehensive population distribution estimate. In this project, only a few hospitality industry business types are covered, but this principle can be applied to anything: from building schools next to high residential population areas to placing advertisement boards in regions of high target demographic traffic.

9.1 Lessons learned

The most important takeaway from this project is that population distribution estimation using Places data works. Adding on, the Google Maps Places API yields very useful datasets and the disaggregation of information by census area is an important step in the data processing pipeline. The task of visualizing geographical data can be effectively tackled using Google's Javascript API and Charts library. Lastly, the project has revealed numerous insights into the socio-economic state of Westminster and the data science methods used to deduce such information.

9.2 Future work

This section offers potential solutions to the project's limitations raised in Chapter 6.

9.2.1 Improved population estimate

An increase in the temporal resolution of the population estimate would be helpful. This can take the form of occupancy curves per demographic per place type, along with temporal relevance values. The resulting choropleth maps would then have a time-series component to cycle through the various temporal distributions of all demographics.

Further processing of the Places dataset could also improve overall model accuracy. The filtering of overlapping place types (i.e. food and restaurant) and the ability to assign relevance values by place ID or proximity to other places can create a more explainable estimate.

9.2.2 Places data enrichment

Continued enrichment of the Places dataset can provide more information to the population models. Certain entities like establishments or businesses contain links to websites that could be used to scrape useful information such as operational hours, customer capacity, number of employees, seasonal events and perhaps unique identifiers that set them apart from similar entities.

9.2.3 Addition of mobility data

It would be scientifically prudent to perform an equivalent population estimate study using mobility data and compare the results. It might be beneficial then to combine both models, in what would constitute a more traditional approach, where places information is used to optimize the accuracy of the mobility-based model. It will be interesting to see how the segmentation of the population into the six demographic types impacts the results.

9.2.4 Implicit data disagregation

An important step in improving the accuracy and explainability of most results showcased in this project would be to replace the aggregation level of the data by OA with a more granular and uniform alternative. A proposal inspired by [3] is to split the borough of Westminster into a grid of 25 square meter squares and then both: disaggregate already aggregated census data to this property and collect new data exclusively at this scale. Details on the former are available in the literature.

This strategy would remove the need for normalization factors as all search areas would be the same size. It would also allow for variable authority aggregation on-demand as the raw data can be grouped by postcodes, output areas and wards efficiently.

9.2.5 Alternative Places mining techniques

The compilation of the Places dataset can be improved both in time and content. A better arrangement of search areas that cover the total searchable area can be found, which would lead to reduced mining times. More importantly, the search area of each OA can be better defined to increase the accuracy, smoothing and stability of the results.

Using the aforementioned 25 square meter Westminster subdivision, an area that mimics the shape of an OA polygon can be built out of these uniform units. More informed smoothing

adjustments can then be made, such as expanding the search area depending on the polygon shape or at variable rates depending on the population distribution within the OA.

9.2.6 General user interface improvements

Giving the user more control over how the data is displayed on the screen is a simple engineering task that has a significant effect on how impactful the entire information delivery system can be. The ability to view more than one choropleth map at a time, perhaps four like in Figure 9.1, would make it a lot easier to compare data. Overlaying multiple columns on a single map would have a similar effect, and both of these features combined would complement each other and dramatically increase the versatility of the UI. Area selection can be useful for filtering regions and examining them in isolation. Combined with multicolumn overlays and multiple choropleth maps, not only areas, but also columns can be: filtered, aggregated and simultaneously compared with other selections or borough totals. User interface improvements alone would constitute a big step forward in the usability of the system.

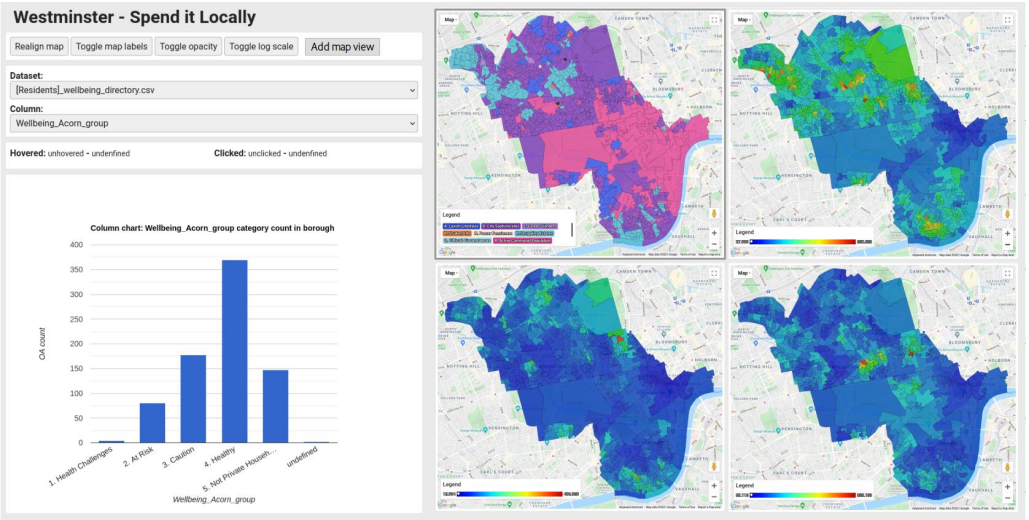


Figure 9.1: User interface multiple choropleth map concept.

References

[1] Acorn user guide. URL: https://www.caci.co.uk/sites/default/files/resources/Acorn%20User%20Guide%202020.pdf.

[2] Budhendra Bhaduri. Population distribution during the day, 2008.

[3] Budhendra L. Bhaduri, Edward A. Bright, Amy N. Rose, Cheng Liu, Marie L. Urban, and Robert N. Stewart. Data driven approach for high resolution population distribution and dynamic models, 2014.

[4] BusinessTown. Factors to consider when choosing a business location. URL: https://businesstown.com/factors-consider-choosing-business-location/.

[5] Caci. URL: https://www.caci.co.uk/.

[6] Data-driven documents. URL: https://d3js.org/.

[7] Helen Susannah Moat Federico Botta and Tobias Preis. Quantifying crowd size with mobile phone and twitter data, 2015.

[8] Office for National Statistics. Census geography - an overview of the various geographies used in the production of statistics collected via the uk census. URL: https://www.ons.gov.uk/methodology/geography/ukgeographies/censusgeography.

[9] Geopandas. URL: https://geopandas.org/.

[10] Google maps javascript api. URL: https://developers.google.com/maps/documentation/javascript/overview.

[11] Google maps javascript api. URL: https://developers.google.com/chart.

[12] Google maps python api. URL: https://github.com/googlemaps/google-maps-services-python.

[13] Westminster City Council Labour Group. What is the future for westminster? URL: https://www.westminsterlabour.org.uk/westminster-news/2020/08/21/what-is-the-future-for-westminster/.

[14] GLA Intelligence. Commuting in london - july 2014. URL: https://londondatastore-upload.s3.amazonaws.com/Zho%3Dttw-flows.pdf.

[15] Sally Lauckner. How to choose a business location: 8 factors to consider. URL: https://www.nerdwallet.com/article/small-business/business-location.

[16] Xiang Liu and Philo Pöllmann. Dynamic population estimation using anonymised mobility data, 2020.

[17] Mapshaper. URL: https://mapshaper.org/.

[18] City of Westminster. City of westminster. URL: https://nla.london/members/city-of-westminster.

[19] City of Westminster. City of westminster profile 2018. URL: https://www.westminster.gov.uk/sites/default/files/city_profile.pdf.

[20] pandas. URL: https://pandas.pydata.org/.

[21] Nirav N. Patel, Forrest R. Stevens, Zhuojie Huang, Andrea E. Gaughan, Iqbal Elyazar, and Andrew J. Tatem. Improving large are population mapping using geotweet densities, 2016.

[22] Clare Yeowart Rachel Findlay and Angela Kail. Understanding social needs in westminster. URL: https://www.thinknpc.org/wp-content/uploads/2018/07/Report-for-Grosvenor-and-the-Westminster-Foundation.pdf.

[23] Robert Riddell. Sustainable urban planning, 2004.

[24] Matt Rosenberg. Population density information and statistics. URL: https://www.thoughtco.com/population-density-overview-1435467.

[25] British Computer Society. Bcs code of conduct. URL: https://cdn.bcs.org/bcs-org-media/2211/bcs-code-of-conduct.pdf.

[26] Enrico Steiger, René Westerholt, Bernd Resch, and Alexander Zipf. Twitter as an indicator for whereabouts of people? correlating twitter with uk census data, 2015.

[27] Inc. The Staff of Entreperneur Media. 10 things to consider when choosing a location for your business. URL: https://www.entrepreneur.com/slideshow/299849.

[28] Erin Yurday. Average uk household cost of food. URL: https://www.nimblefins.co.uk/average-uk-household-cost-food.

Appendix A

Source code

The project source code can be found at:

anon

Appendix B

Raw data sources

B.1 CACI datasets

These datasets were provided by Westminster City Council as part of the Spend it Locally project. They originate from the data provider company CACI:

WCC - CACI Paycheck Disposable Income - Feb2020.xlsx `www.google.com`

- https://www.caci.co.uk/datasets/

WCC - Population by Age and Gender - Feb2020.xlsx

- https://www.caci.co.uk/datasets/

Westminster City Council - Westminster Acorn directory - February 2020.xlsx

- https://www.caci.co.uk/datasets/

Westminster City Council - Westminster COICOP directory - February 2020.xlsx

- https://www.caci.co.uk/datasets/

Westminster City Council - Westminster Paycheck directory - February 2020.xlsx

- https://www.caci.co.uk/datasets/

Westminster City Council - Westminster PTAL directory - February 2020.xlsx

- https://www.caci.co.uk/datasets/

Westminster City Council - Westminster StreetValue directory - February 2020.xlsx

- https://www.caci.co.uk/datasets/

Westminster City Council - Westminster Wellbeing Acorn directory - February 2020.xlsx

- https://www.caci.co.uk/datasets/

B.2 UK authority hierarchy datasets

These datasets include information about the authority hierarchy in the UK:

OA_Ward_LA_2011.csv

- Authority mapping from OAs to LSOAs, wards and MSOAs in the UK. Originally from the UK Data Service, Census Support.
- https://borders.ukdataservice.ac.uk/easy_download.html

OAs_geojson.json

- Authority boundary geographical data for all Output Areas in the UK. Originally from the UK Data Service, Census Support.
- https://borders.ukdataservice.ac.uk/easy_download.html

UK_Postcode_to_Output_Area_Hierarchy_with_Classifications.csv

- Authority hierarchy information about postcodes in the UK. Includes a large range of higher level authorities and also some authority classifications. Originally from the UK Data Service, Census Support.
- https://borders.ukdataservice.ac.uk/pcluts.html

B.3 Google Maps Places API

The Place types dataset is constructed from data available in the Google Maps Places API documentation:

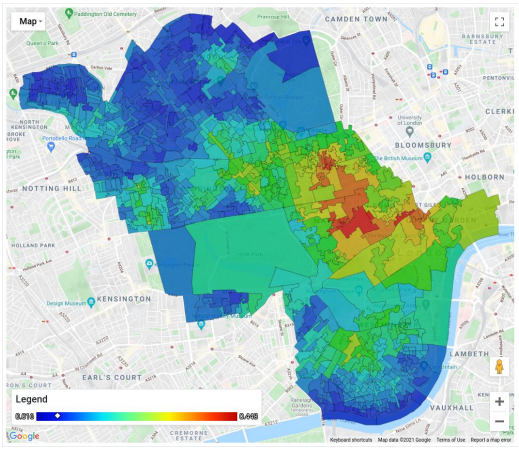
palce_types.csv

- https://developers.google.com/maps/documentation/places/web-service/supported_types

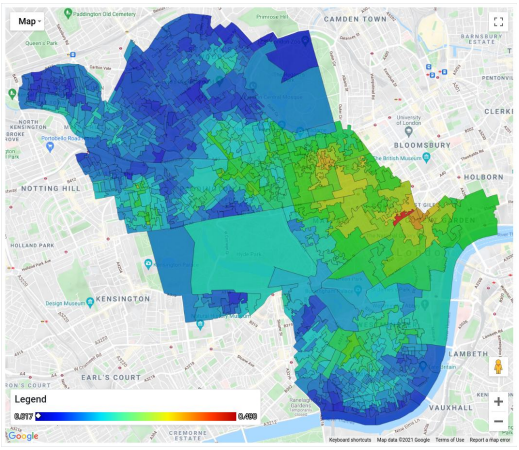
Appendix C

Demographic distribution models

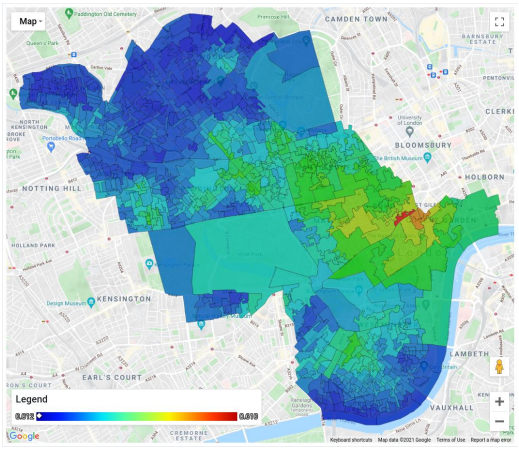
C.1 Granular model



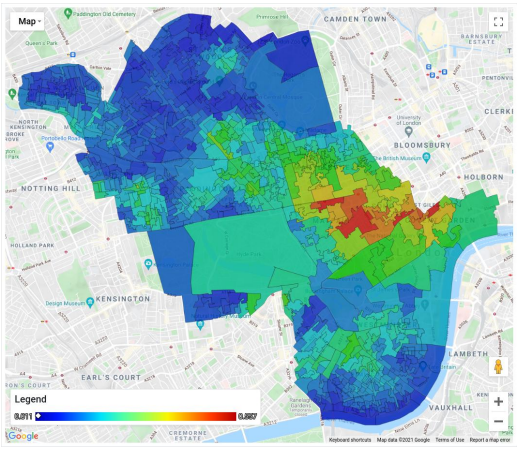
(a) Worker



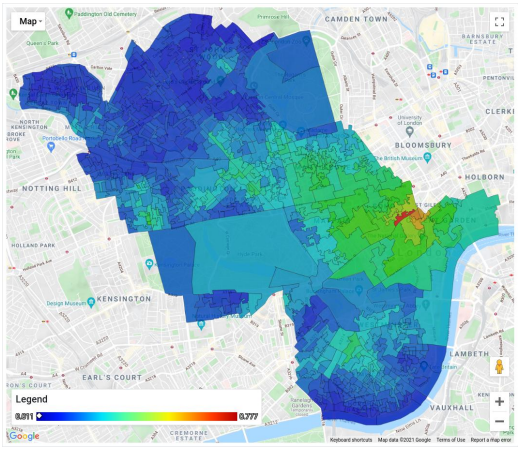
(b) Student



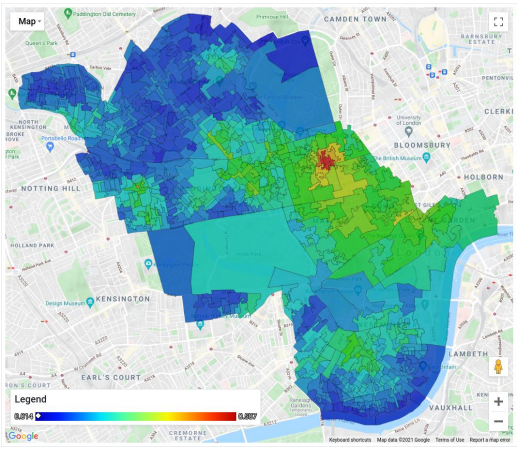
(c) Tourist



(d) Shopper



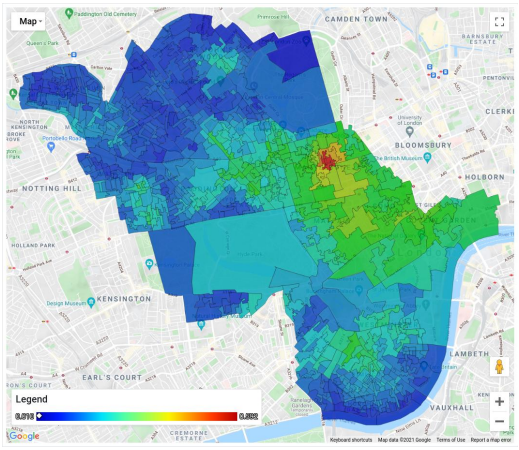
(e) Leisurer



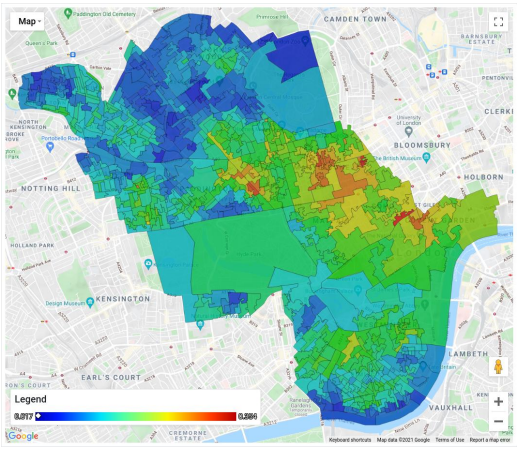
(f) Chorer

Figure C.1: Granular model.

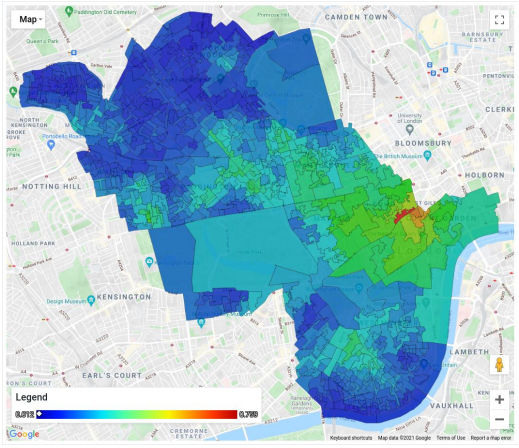
C.2 Supertypes model



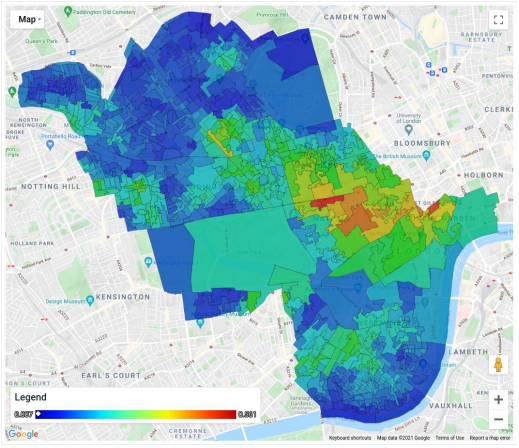
(a) Worker



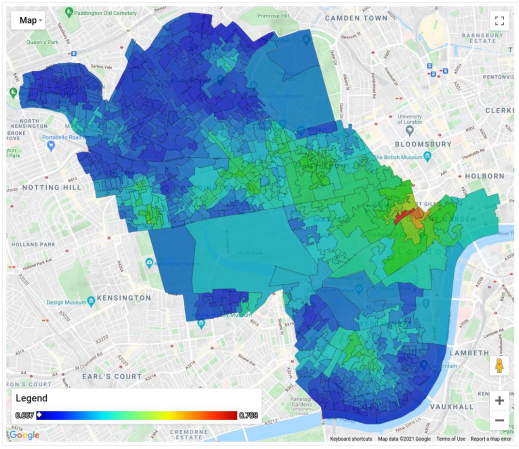
(b) Student



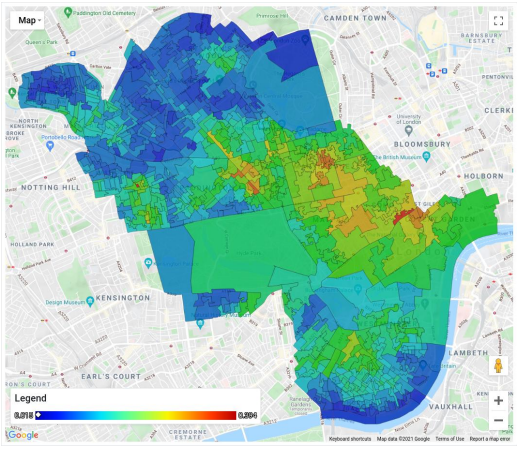
(c) Tourist



(d) Shopper



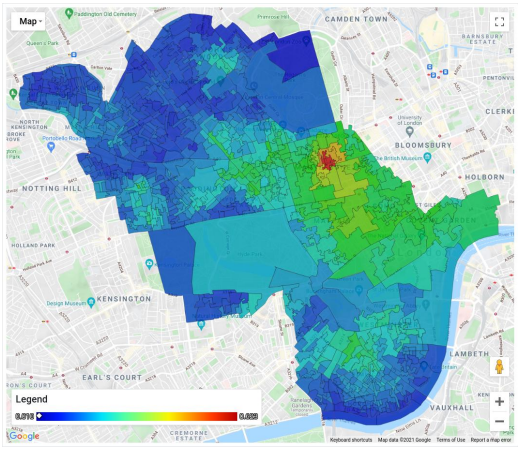
(e) Leisurer



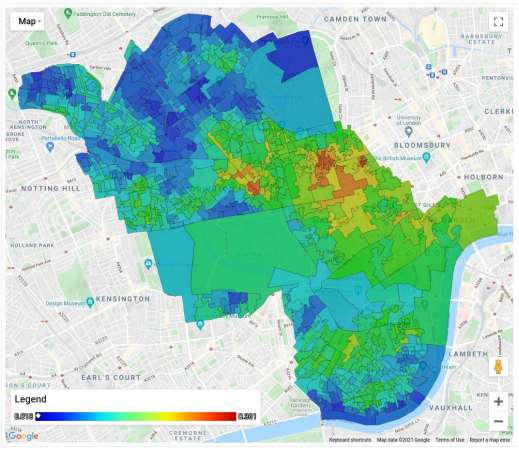
(f) Chorer

Figure C.2: Supertypes model.

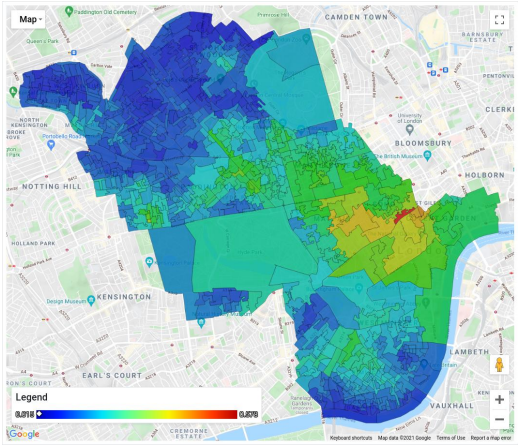
C.3 Supertypes attractors model



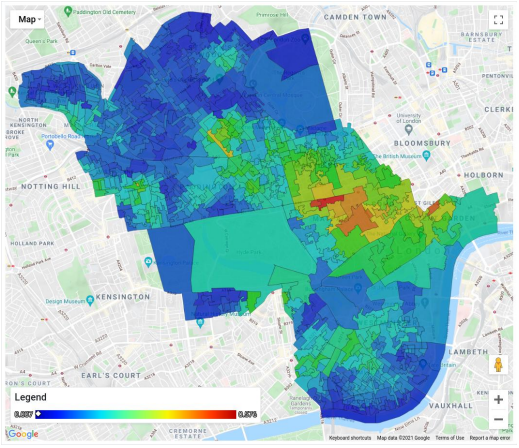
(a) Worker



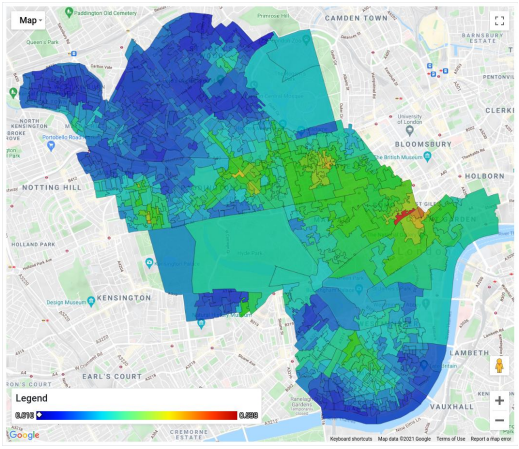
(b) Student



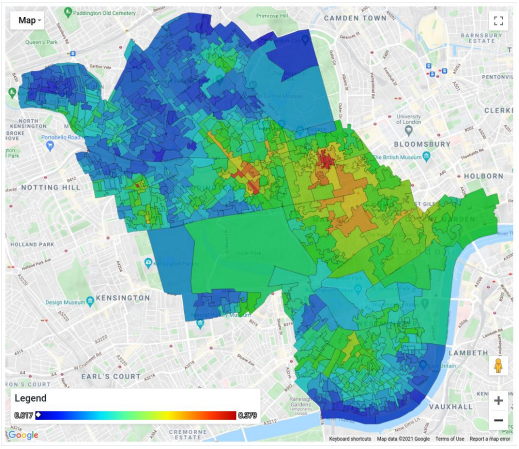
(c) Tourist



(d) Shopper



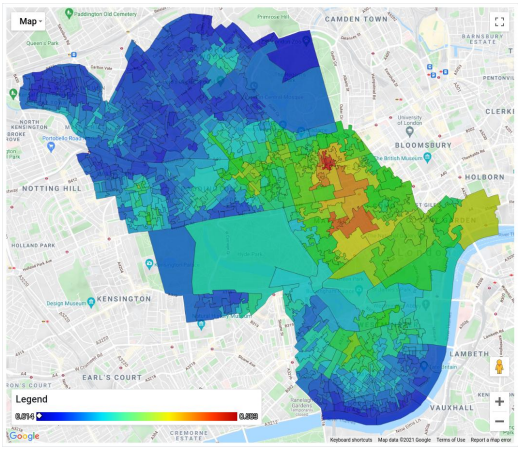
(e) Leisurer



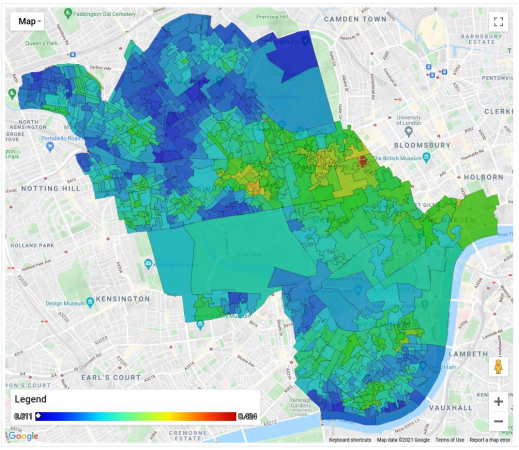
(f) Chorer

Figure C.3: Supertypes attractors model.

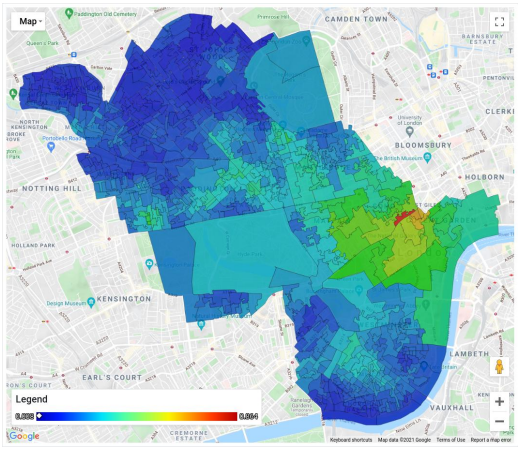
C.4 Supertypes discriminant model



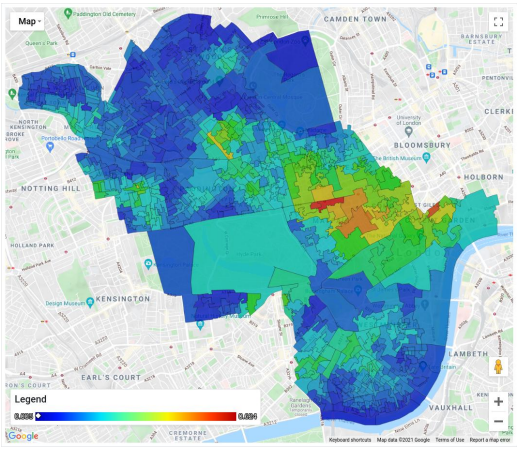
(a) Worker



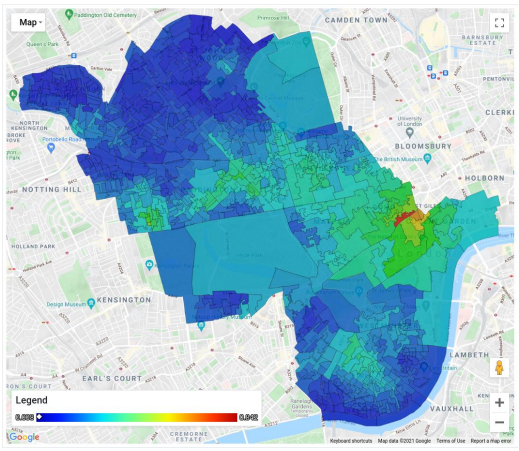
(b) Student



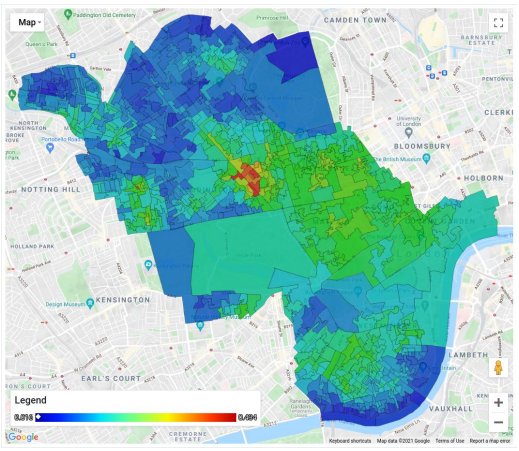
(c) Tourist



(d) Shopper



(e) Leisurer



(f) Chorer

Figure C.4: Supertypes discriminant model.

Appendix D

Additional auxiliary visualizations

D.1 Place count bar chart

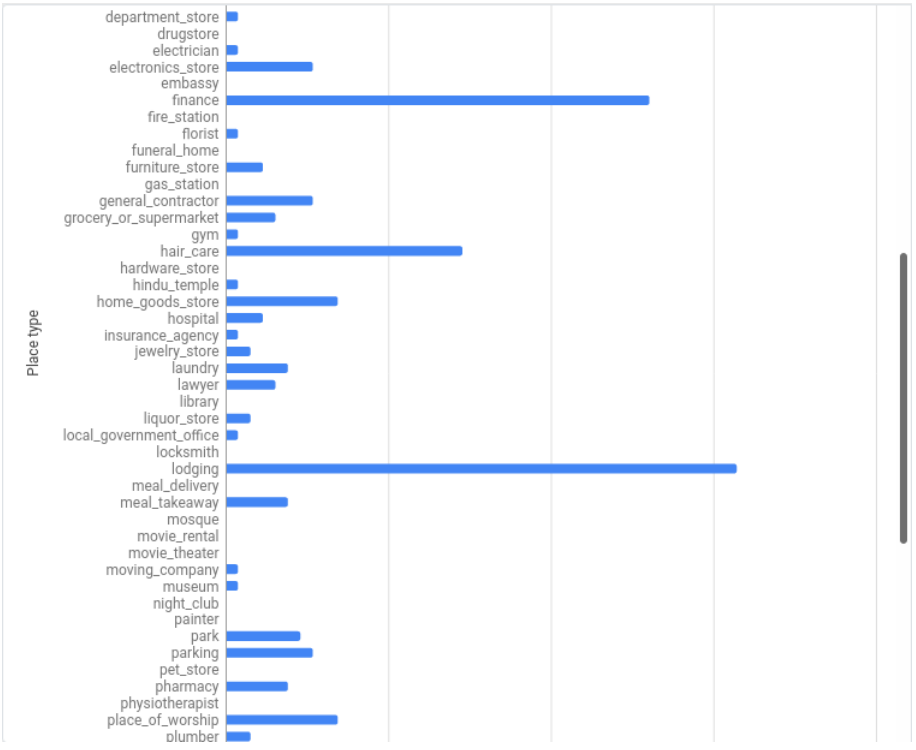


Figure D.1: Place count bar chart. Displayed when any column from a [Places] dataset is selected an OA is clicked. It presents the value assigned to each place type at that OA.

D.2 Demographic distribution column chart

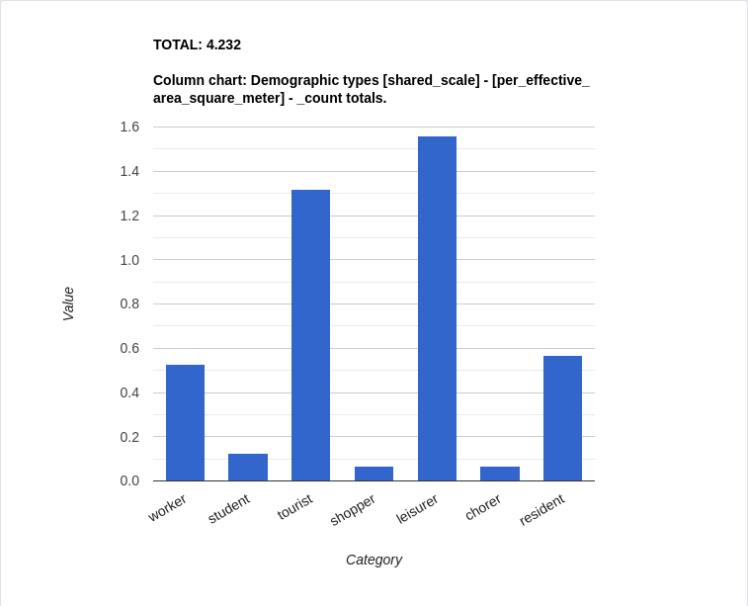


Figure D.2: Demographic distribution pie chart. Displayed when a [shared_scale] column from the [Population] dataset is selected and an OA is clicked. It presents a column with the values of all demographic types in that OA.

D.3 Demographic distribution pie chart

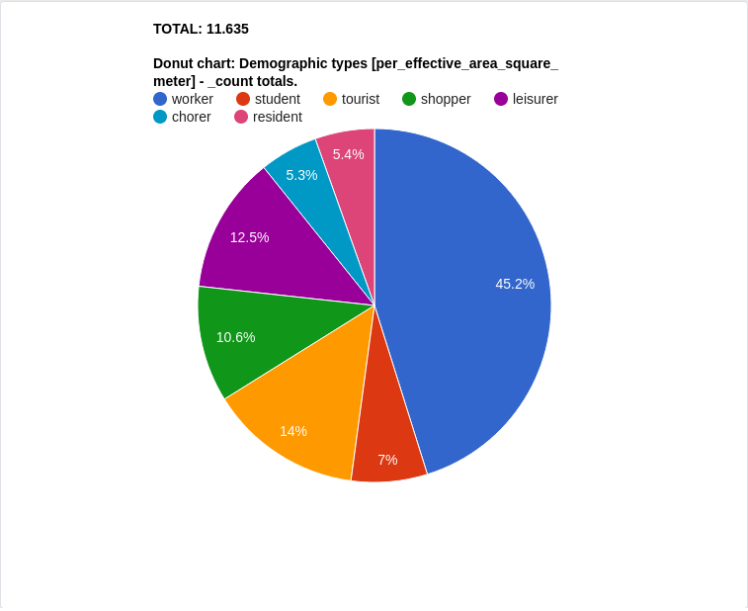


Figure D.3: Demographic distribution pie chart. Displayed when a non-total column from the [Population] dataset is selected and an OA is clicked. It presents a pie chart split into the percentages of all demographic types in that OA.

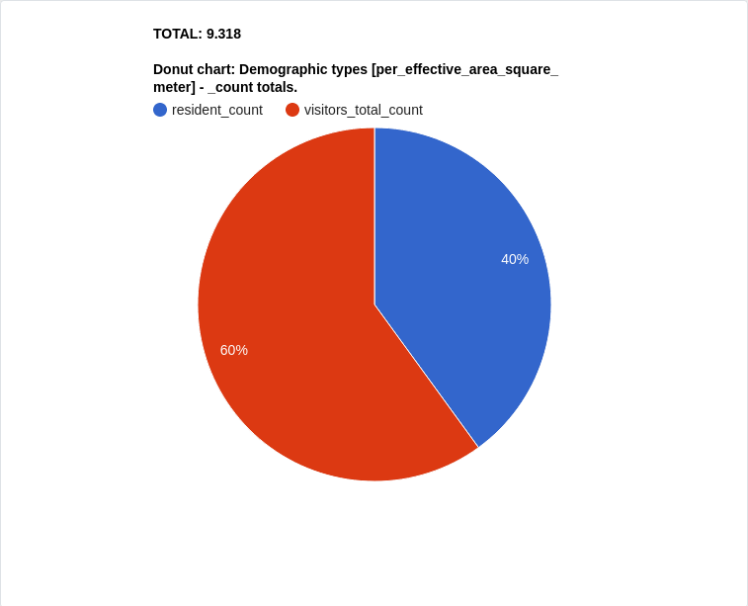


Figure D.4: Demographic distribution pie chart. Displayed when a total column from the [Population] dataset is selected and an OA is clicked. It presents a pie chart split into the resident and visitor percentages in that OA.

D.4 Supply and demand distribution nested column chart

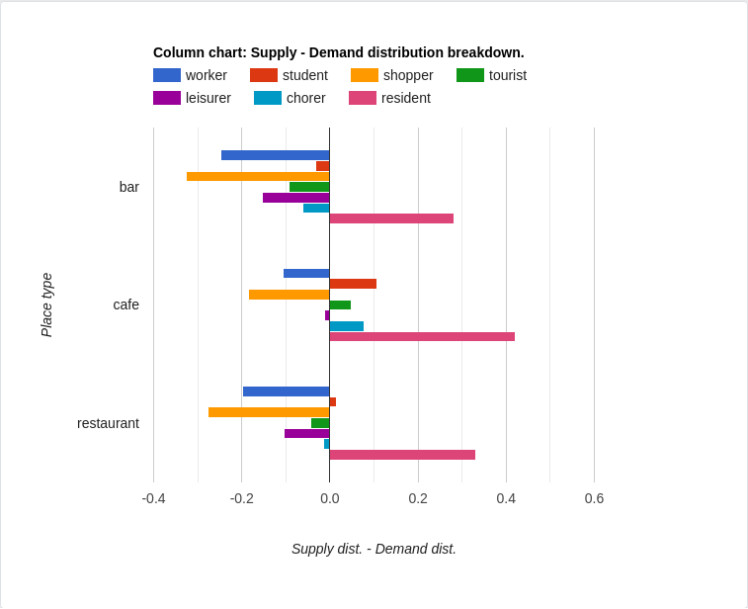


Figure D.5: Supply and demand distribution nested column chart. Displayed when a [supply - demand] column in the [Supply_demand] dataset is selected and an OA is clicked. It presents the result, for each demographic, of the operation of supply dist. minus demand dist. at that OA.

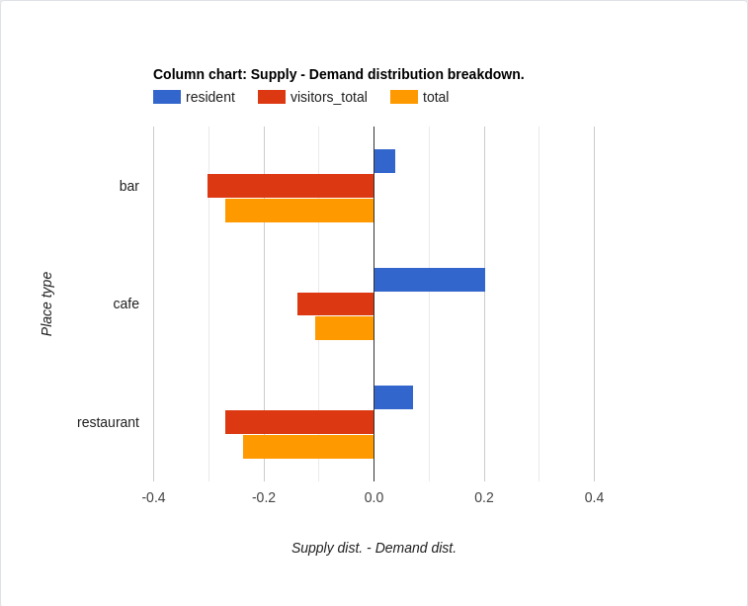


Figure D.6: Supply and demand distribution nested column chart. Displayed when a [supply - demand] total column in the [Supply_demand] dataset is selected and an OA is clicked. It presents the result (for residents, visitors and residents + visitors) of the operation of supply dist. minus demand dist. at that OA.

D.5 Supply and demand index nested bar chart

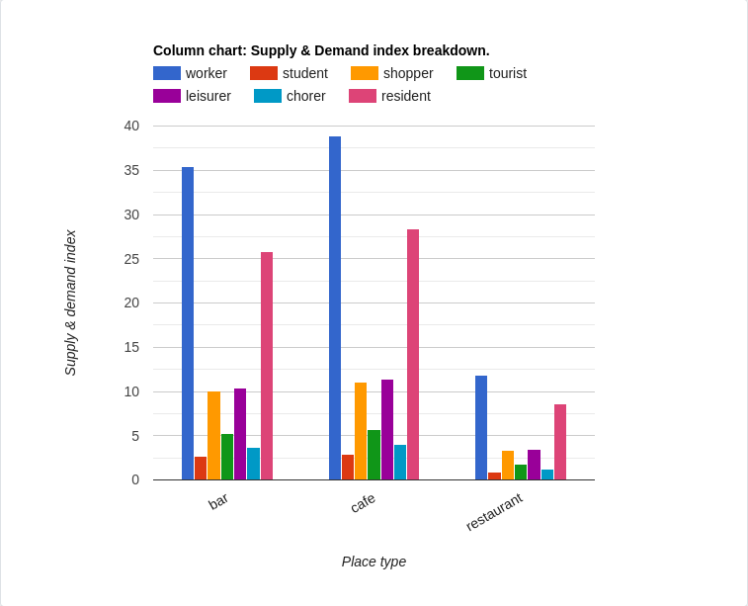


Figure D.7: Supply and demand index nested bar chart. Displayed when a [supply_demand_index] column in the [Supply_demand] dataset is selected and an OA is clicked. It presents the supply and demand index values per demographic in that OA.

Appendix E

System setup guide

A Data Science system to analyse the socio-economic state of The City of Westminster in London, United Kingdom. Consists of a data processing pipeline written in Python and a user interface and data visualization system written in Javascript. The system deals with resident census data, a places catalogue, population demographic distribution estimate models and supply & demand metrics.

E.1 Running instructions

In the delivered version of the project, the datasets in the data processing project are not present due to legal reasons. The user interface does have the datasets it requires to run, so the executions steps can be followed starting at step 6.

E.1.1 Preparation steps

1. Create a Google Cloud Console account.
2. Create a Google Cloud project.
3. Enable the Google Maps: Places, Javascript and Maps APIs. Instructions at https://developers.google.com/maps/documentation/javascript/places.
4. Generate an API key for the project.

(Data processing pipeline)

5. Create the file `data_processing/src/focused_data/api.py` and set its contents to `api_key = "[API_KEY]"`.

(User interface and visualization web application)

6. Create the file `user_interface/website_ui/credentials/google_maps_api_credentials.js` and set its contents to `const apiKey = "[API_KEY]"`.

E.1.2 Execution steps

(Data processing pipeline)

1. Ensure all the required raw data files are found in the appropriate subdirectories under `data_processing/raw_data`.
2. Execute the `run_scrape_places.py` script.
3. Execute the `run_focused.py` script.
4. Execute the `run_processed.py` script.
5. Execute the `copy_dataset_files.py` script.

(User interface and visualization web application)

6. Ensure all the required data files are found in the `user_interface/website_ui/data` directory.
7. Launch the web application project.

E.1.3 Web application launching

For development, testing and demonstrations, it is sufficient to launch the app in a local development server. Many alternatives exist but a working procedure is detailed below:

1. Download and install the Visual Studio Code (VSCode) code editor at https://code.visualstudio.com/.
2. In the Extensions pane on the left hand side tool bar, install the Live Server extension. Can also be found at https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer.
3. Open the project root `Spend-it-Locally` with VSCode.
4. Open the `index.html` file in the main code editing window.
5. Click on the *Go Live* button in the bottom right of the editor window.