

BIRKBECK, UNIVERSITY OF LONDON

MSC PROJECT REPORT

Mapping London House Prices through Well-being

Author:

Ben CANDY

Supervisor:

Dr. Alessandro PROVETTI



*A project report submitted in fulfillment of the requirements
for the degree of Master of Science*

in the

Department of Computer Science and Information Systems

September 17, 2018

Declaration of Authorship

I, Ben CANDY, confirm that:

- This report is substantially the result of my own work, expressed in my own words, except where explicitly indicated in the text.
- I give my permission for it to be submitted to the JISC Plagiarism Detection Service.
- The report may be freely copied and distributed provided the source is explicitly acknowledged.

Signed:

Date:

BIRKBECK, UNIVERSITY OF LONDON

Abstract

School of Business, Economics and Informatics
Department of Computer Science and Information Systems

Master of Science

Mapping London House Prices through Well-being

by Ben CANDY

I have always been interested in the spatial analysis of human behaviour. For this project I am going to put together a software project which will allow me to visualise this in a data-driven way. In my literature review I discovered that there are hardly any quantifiable metrics to link human happiness a real estate values in a geographical context. Therefore, I look forward to experimenting with new measures, trying to leverage the wealth of data we now have on “London Living” and Data Science methods such as Principal component analysis and dimensionality reduction.

Acknowledgements

I would like to thank my project supervisor Professor Alessandro Provetti for his support and guidance throughout this project....

Contents

Declaration of Authorship	iii
Abstract	v
Acknowledgements	vii
1 Introduction	1
1.1 Motivations and problem statement	1
1.2 Project trailer	2
2 State of the art	5
2.1 Geospatial analysis in Data Science	5
2.1.1 Point Maps	6
2.1.2 Choropleth Maps	6
2.1.3 Heat Maps	7
2.1.4 Network Maps	8
3 Data Sources	11
3.1 Datasets required to create the well-being indices	11
3.2 Static data files	11
3.2.1 London datastore	12
3.2.2 gov.uk	12
3.2.3 esri	12
3.3 APIs	12
3.3.1 Foursquare	12
3.3.2 Google Maps	13
3.4 Shape Files	13
3.4.1 London Ward shapefile	13
3.5 Data availability issues	13
3.5.1 Timeliness	13
3.5.2 Boundary changes	14
3.5.3 API rate limits	14
4 Design	15
4.1 Architecture of the specification	15
4.1.1 Archictecture diagram	15
4.1.2 Architecture choices	15
4.2 Components	16
4.2.1 Data inputs	16
4.2.2 Data importer	16
4.2.3 Data store	16
4.2.4 Main application	16
4.2.5 Data modeller	17

4.2.6	Interactive map	17
4.3	London house prices as a regression problem	17
4.4	London house prices as a classification problem	20
5	Implementation	23
5.1	Data collection and cleaning	23
5.1.1	Collection of data files	23
5.1.2	Collection from APIs	23
5.2	Well-being domain scores	24
5.3	Building a regression model	25
5.4	Building the best classifier	26
5.5	Encoding as geoJSON	28
5.6	Interactive maps	29
6	Testing and validation	31
6.1	Testing the data inputs	31
6.2	Testing the models	31
6.3	Validating the outputs	31
7	Conclusions and evaluation	33
7.1	Lessons learnt	33
7.1.1	Summary of findings	33
7.2	Possible developments	34
A	Appendix A	35
B	Appendix B - Additional tables and charts	37
C	Appendix C - Code	41
	Bibliography	61

List of Figures

1.1	Visualisation of median house prices by ward in London. Hovering over a particular ward will provide information on the well-being indices for that ward and the modelled median house prices and associated quintiles.	2
1.2	Visualisation of predicted median house prices by ward in London. This map shows the predicted values based on a linear regression model.	2
1.3	Visualisation of the quintiles of median house prices by ward in London. Hovering over a particular ward will provide information on the well-being indices for that ward and the modelled median house prices and associated quintiles.	3
1.4	Visualisation of the predicted quintiles median house prices by ward in London. This map shows the predicted values based on a logistic regression model.	3
1.5	Visualisation of the predicted quintiles median house prices by ward in London. This map shows the predicted values based on a neural network.	4
2.1	An range of the maps exploring London on the Mapping London website. This website has been created by academics from the geography department at UCL and features a range of data-driven maps using various geospatial techniques.	5
2.2	An example of a point map - the Great British Public Toilet map created by the Royal College of Art in 2014 showing instances of public lavatories across the country.	6
2.3	An example of a choropleth map - created by the City of Toronto to provide information about the neighbourhoods of urban Toronto.	7
2.4	An example of a heat map - a 2011 map created by the estate agents Zoopla to show UK average house prices.	8
2.5	An example of a network map - created by Quercia, Aiello and Schifanella in 2016, this map categorises each road in London based on the primary type of noise heard in the location.	9
4.1	Choropleth map showing median house prices for each ward in London (based on values at the end of 2017).	18
4.2	Pair-wise plot of the six well-being indices London median house prices with histograms showing the distribution of each index and house prices.	19
4.3	Choropleth map showing median house prices by for each ward in London by quintile (based on values at the end of 2017).	21

4.4	Pair-wise plot of the six well-being indices London median house prices with histograms showing the distribution of each index and house prices. Points are coloured on the basis of the quintile of the median house price within London ward.	22
5.1	Python function written to import data from Foursquare. This function iterates for latitude and longitude to traverse London and return venue information for one of three categories ("cultural", "food" or "nightlife) without exceeding daily API limits.	24
5.2	geoJSON specification showing Point, LineString and Polygon geometry objects.	29
B.1	Choropleth map showing the Education and Employment index	38
B.2	Choropleth map showing the Safety and Security index	38
B.3	Choropleth map showing the Environment index	39
B.4	Choropleth map showing the Community Vitality and Participation index	39
B.5	Choropleth map showing the Infrastructure index	40
B.6	Choropleth map showing the Health index	40

List of Tables

3.1	Indices and indicators used for the measurement of well-being for each London ward with data source and metric used	11
4.1	Correlation Matrix for the six indices and median house prices (000s)	20
5.1	Results for regression models on London median house prices	25
5.2	Results for regression models on London median house prices	26
5.3	Results for Neural Network classifier with Logistic activation function and different numbers of nodes in the hidden layer. Based on average scores using k-fold cross validation with k=7.	27
5.4	Results for Neural Network classifier with 45 nodes in the hidden layer and different activation functions. Based on average scores using k-fold cross validation with k=7.	28
5.5	Results for classification models on the quintiles of London median house prices	28

Chapter 1

Introduction

1.1 Motivations and problem statement

In recent years, the concept of well-being has been the subject of increasing interests for governments, councils, policy makers and researchers, though research on the subject has been happening since the 1970s. An interesting recent development is a report into well-being in Danish cities by (OECD, 2016) which highlights how well-being tools have become an important tool to identify the needs of citizens and the domains where demand for progress is greatest.

This idea of well-being is often defined using a set of indicators relating to a particular place. There is no universally agreed definition of exactly which factors are the best measures of well-being, however, various indexes and studies show a great deal of commonality on the types of measures that are important to the well-being of citizens.

What is less clear is whether, and how, these indicators affect the real estate values in a specific city. For example, The Greater London Authority created a set of well-being indicators and used them to create a map of well-being by London Ward in 2014 (GLA, 2014), however, this was not linked back to real estate values. In one study looking at linkages between house prices and mental wellbeing, (Ratcliffe, 2013), points out that when people can move freely, each person will maximize well-being by moving to areas that best satisfy their preferences; this results in zero correlation between area characteristics (or house prices) and wellbeing. But if people think they are paying too much for poor quality or too little for high quality, we would observe a positive relationship between area characteristics and wellbeing. The study found a positive correlation between house prices and mental wellbeing.

Another hypothesis found in the Economics literature is that real estate values are the drivers of the well-being indicators, happier people are better at generating wealth (Lyubomirsky, King, and Diener, 2005), which leaves open the possibility that areas with high well-being provide a flow of people migrating to places with higher real estate values. From the point of view of Data Science it would be interesting to consider the real estate values in bandings such as quintiles as well-being indicators may have a very different relationship with the top 20 percent of areas by real estate value than with areas in the lower four quintiles.

Work on urban scaling (Bettencourt et al., 2010) suggests that different types of well-being indicators may have different relationships with real estate values due to the non-linear nature of agglomeration. Bettencourt et al. argues that city indicators are governed by power laws rather than linear per-capita indicators. Social factors tend to be superlinear whereas material infrastructure tends to be sublinear at best. There may therefore be some level of divergence between different well-being factors. This project focuses on the city of London and tries to determine which well-being indicators show a relationship with real-estate values. The findings will be

displayed in the form of an interactive map.

This project sets out to create a set of well-being indicators for each administrative ward in London and will attempt to model median house prices based on these indicators.

1.2 Project trailer

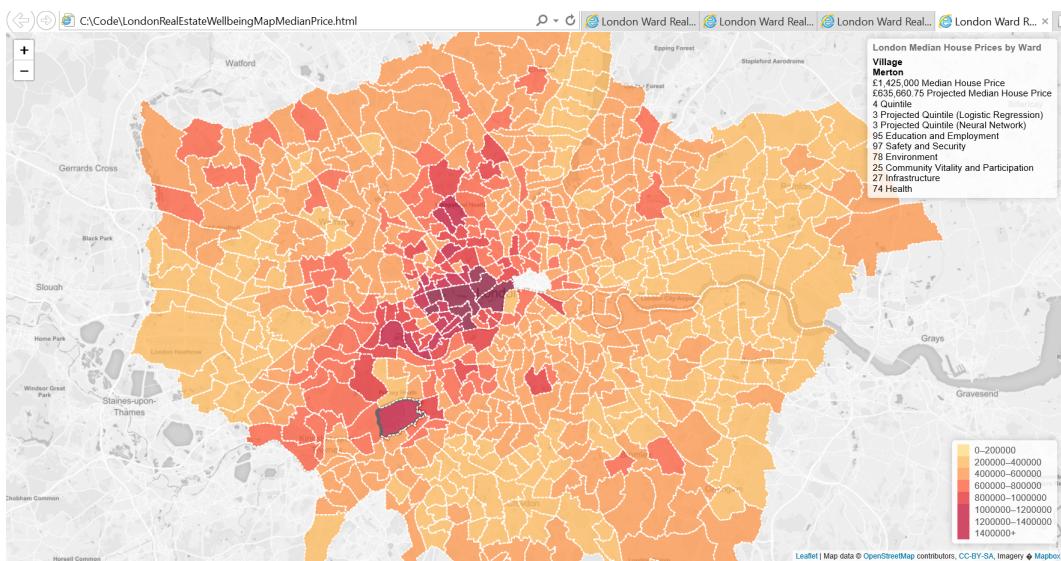


FIGURE 1.1: Visualisation of median house prices by ward in London. Hovering over a particular ward will provide information on the well-being indices for that ward and the modelled median house prices and associated quintiles.

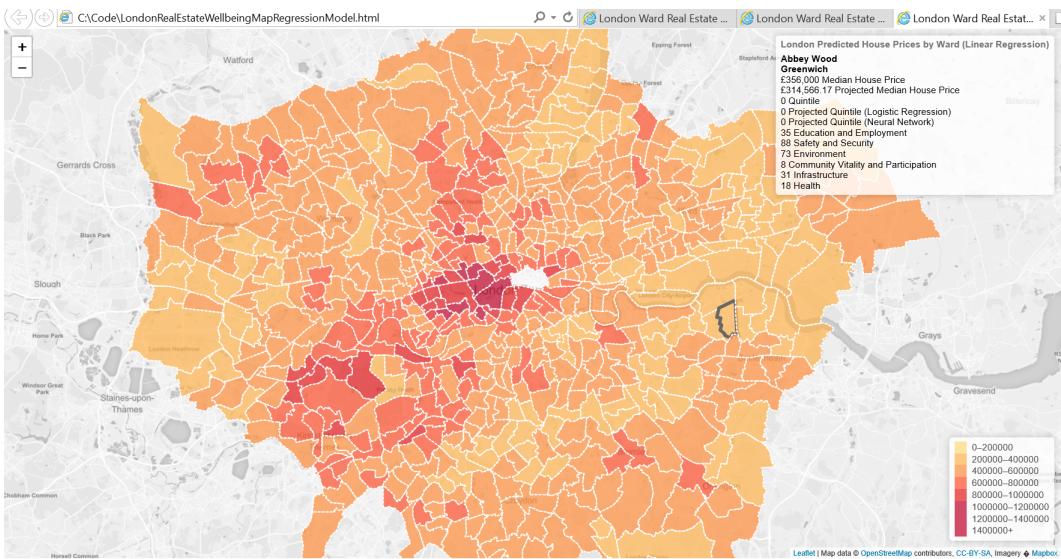


FIGURE 1.2: Visualisation of predicted median house prices by ward in London. This map shows the predicted values based on a linear regression model.

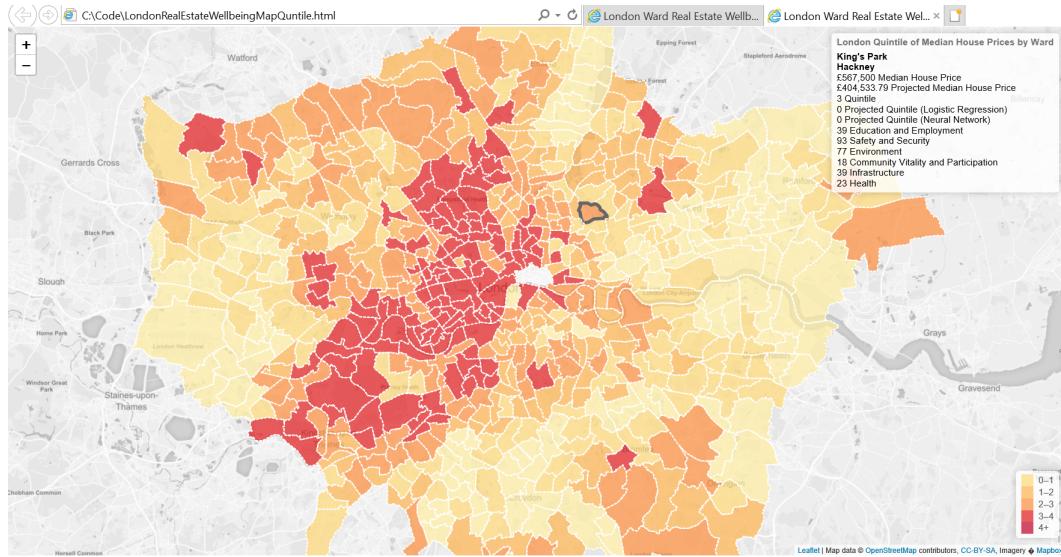


FIGURE 1.3: Visualisation of the quintiles of median house prices by ward in London. Hovering over a particular ward will provide information on the well-being indices for that ward and the modelled median house prices and associated quintiles.

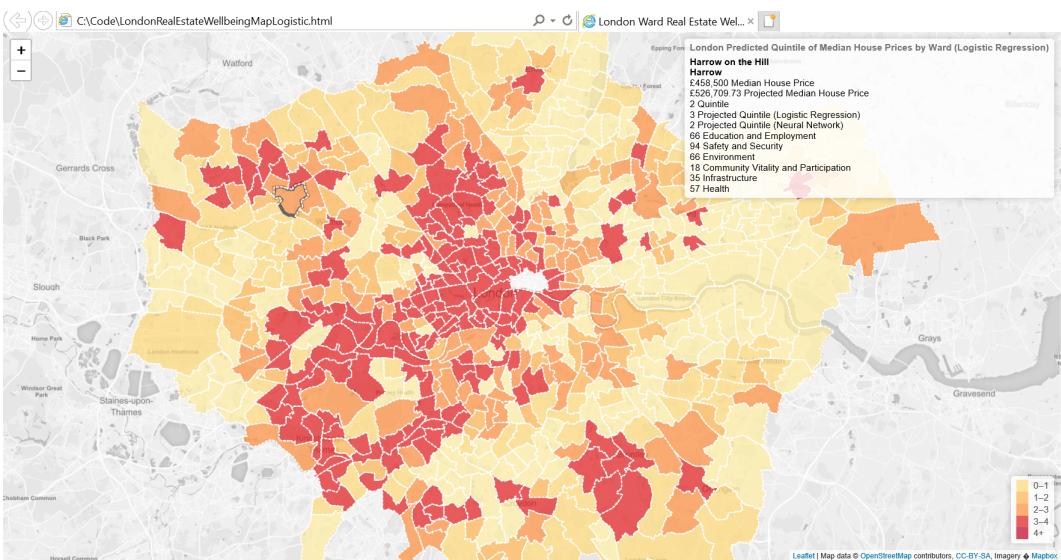


FIGURE 1.4: Visualisation of the predicted quintiles median house prices by ward in London. This map shows the predicted values based on a logistic regression model.

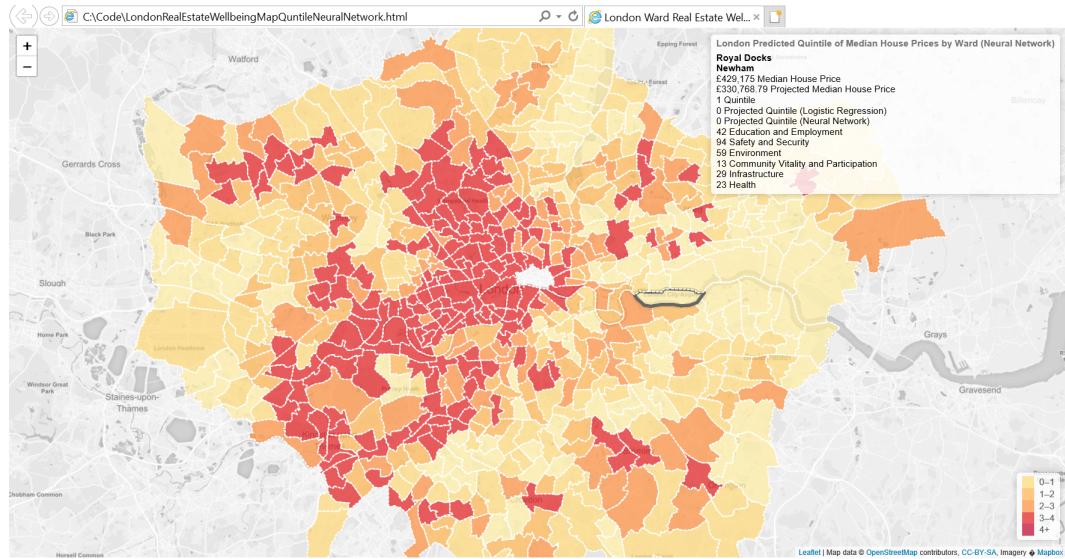


FIGURE 1.5: Visualisation of the predicted quintiles median house prices by ward in London. This map shows the predicted values based on a neural network.

Chapter 2

State of the art

2.1 Geospatial analysis in Data Science

Maps are an increasingly popular way to visualise data where some form of geographic aspect is an important element of the analysis. The website London Mapping (MappingLondon.co.uk, 2018) is a good example of the large amount of data analysis and visualisation taking place through the creation of various types of map for the city of London alone.



FIGURE 2.1: A range of maps exploring London on the Mapping London website. This website has been created by academics from the geography department at UCL and features a range of data-driven maps using various geospatial techniques.

There are several reasons why maps are a good choice for data visualisation which include:

- They provide a real-world context for the data, helping the audience to understand the analysis
- They allow users to compare data over different areas or regions at a glance

Data Science techniques play an important role in the creation of this kind of visualisation. Most map visualisations represent locations through coordinate reference systems which represent locations. These locations can be specific locations

represented by pairs of coordinates, most often (latitude, longitude), or areas which would be represented by polygons consisting of multiple coordinate pairs. Data Science techniques are required to design functions and algorithms to map, aggregate or disaggregate data between different types of geometry and different coordinate reference systems.

2.1.1 Point Maps

One mapping technique often used in interactive visualisations would be a point map where specific instances of something are plotted as points in the geo-location in which they occur. This form of mapping would use a coordinate pair, often latitude and longitude but potentially an Ordnance Survey grid reference or other coordinate reference system. An example of this would be the Great British Public Toilet Map (RCA, 2014)

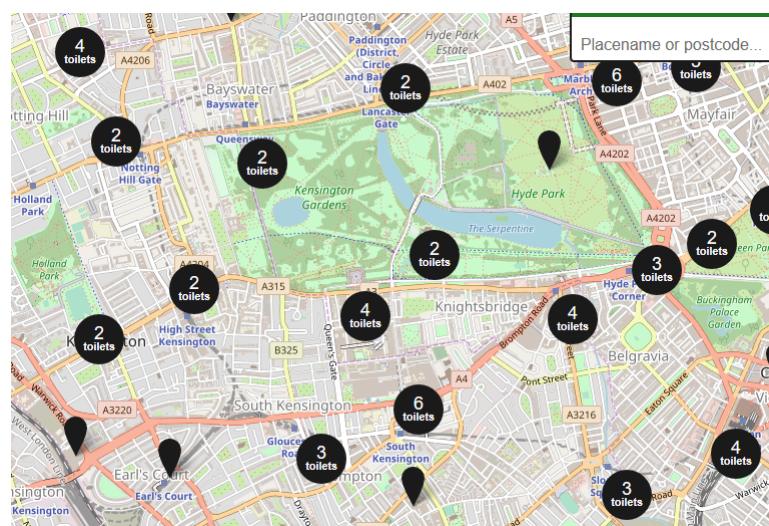


FIGURE 2.2: An example of a point map - the Great British Public Toilet map created by the Royal College of Art in 2014 showing instances of public lavatories across the country.

2.1.2 Choropleth Maps

The city of Toronto is a good example of where an informative interactive well-being map has been created (Toronto, 2018). On this map a user can click on an area and is presented with a panel showing the values for a number of measures.

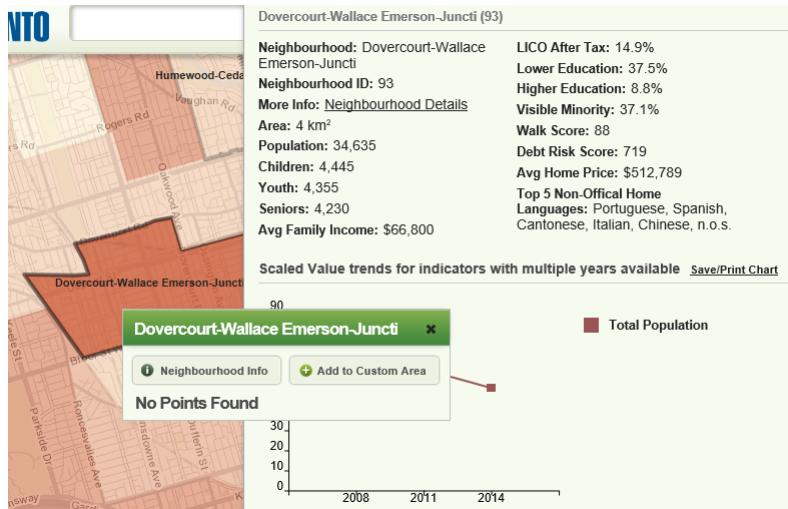


FIGURE 2.3: An example of a choropleth map - created by the City of Toronto to provide information about the neighbourhoods of urban Toronto.

The Toronto map uses a thematic mapping technique known as a Choropleth map, using shape files where polygons represent an area of space, in this instance neighbourhoods, with the colour of the polygon related to the value of the area in question. This map is an excellent design example for this project as the geometries used, in this case neighbourhoods of Toronto, are of a similar geospatial representation to London Wards. The project map will be based on median house prices and hovering over an area will display the values of the well-being indicators and the projected values of the model.

2.1.3 Heat Maps

Real estate values have also been an area of interest for mapping. In recent times these visual displays of real estate value have often been created by the commercial sector, for example, UK property price maps created by the estate agent Zoopla (Zoopla, 2018). This is an example of a heat map. Heat maps share similarities with Choropleth maps in that they assign a colour to an area based on the magnitude of a particular value. Where these two types of map differ is that heat maps assign colour to a cluster of connected points whereas for Choropleth maps, the colouration is based on a value for a polygon representing a given area, usually a administrative or political boundary.



FIGURE 2.4: An example of a heat map - a 2011 map created by the estate agents Zoopla to show UK average house prices.

2.1.4 Network Maps

An interesting alternative approach has been taken by (Quercia, Aiello, and Schifanella, 2016) who have created a series of interactive maps using crowdsourced data and image processing techniques. The locations used in the maps created by Quercia et al. use specific locations represented by images and sound recordings rather than areas of space. Here, rather than using traditional spatial analysis techniques, the maps have been created using graphs which represent London as a network. The location of the images/sounds are the nodes and the routes between them form the vertices.

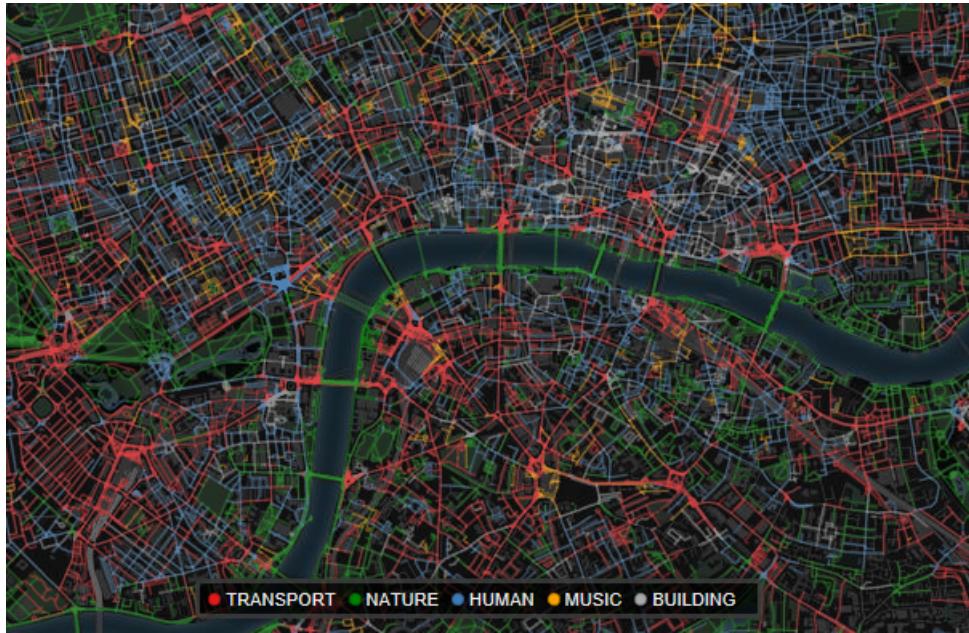


FIGURE 2.5: An example of a network map - created by Quercia, Aiello and Schifanella in 2016, this map categorises each road in London based on the primary type of noise heard in the location.

Chapter 3

Data Sources

3.1 Datasets required to create the well-being indices

To create the well-being indices the decision was made to use six indices each made up of three indicator datasets. For the purpose of the project, each index and indicator will be of equal weighting. This approach is based upon other indices of well-being where data feeds into a set of between five and ten indices. Three examples of this type of index would be the Gross National Happiness index (Bhutan Studies, 2011), the OECD Better Life Initiative (OECD, 2017) and the Canadian Index of Wellbeing (Waterloo, 2017). The approach taken was to find three indicators that both represented each index and where the data could be retrieved or transformed to be analysed at ward level.

TABLE 3.1: Indices and indicators used for the measurement of well-being for each London ward with data source and metric used

Index	Indicators	Source	Metric
Education and Employment	Average GCSE points	London datastore	Average GCSE capped point score
	Average Ofsted rating of schools	edubase @ gov.uk	Average Ofsted ratings of schools - most recent inspections (September 2018)
	Employment rates	London datastore	Rate of economically active individuals (2011)
Safety and Security	Crime rates - against the person	London datastore	Number of recorded crimes (2017)
	Crime rates - other crime	London datastore	Number of recorded crimes (2017)
	Traffic injuries	London datastore	Road collision numbers (2014)
Environment	Air pollution	London datastore	Annual mean of Nitrogen Dioxide (NO2) and Particle emissions (PM10) (2011)
	Amount of greenspace	London datastore	Percentage of ward area which is greenspace (2011)
	Access to nature	London datastore	Percentage of residential households with access to at least one open space (2011)
Community Vitality and Participation	Access to cultural space	Foursquare	Number of cultural venues in ward (August 2018)
	Amount of bars and restaurants	Foursquare	Number of bars and restaurants in ward (August 2018)
	Election turnout	London datastore	Percentage of residents voting in 2016 London election
Infrastructure	Access to public transport	London datastore	Transport for London Public Transport Accessibility Levels (2015)
	Average journey times	gov.uk	The mean percentage value of those within an hour's travel by public transport or walking of an area of 100, 500 and 500 jobs (2014)
	Population density	London datastore	Persons per square km (2013)
Health	Life expectancy	London datastore	Life expectancy at birth (2013)
	Childhood obesity	London datastore	Prevalence of obese children at age 11 (2013)
	Illness preventing employment	gov.uk	Comparative illness and disability ratio from English Indices of Deprivation (2015)

Access to open space definition ¹

Access to public transport definition ²

Illness preventing employment definition ³

3.2 Static data files

The majority of data sets are either csv or Excel files which are accessed directly via a url by the data importer application. These sources are primarily from the websites of public bodies. The data importer application makes use of the pandas module within Python to support the importing and cleaning process within a series of dataframe objects.

¹<https://data.london.gov.uk/dataset/access-public-open-space-and-nature-ward>

²<https://data.london.gov.uk/dataset/public-transport-accessibility-levels>

³<https://www.gov.uk/government/statistics/english-indices-of-deprivation-2015>

3.2.1 London datastore

The Greater London Authority Datastore (GLA, 2018) was the primary source of data for the project with significant amounts of data relating to various London metrics available. The fact that much of the data contained on the website was available at ward level was key in being able to piece together the indices.

3.2.2 gov.uk

Used for Ofsted ratings for schools, average journey times and illness affecting employment, the gov.uk website contains a varied range of datasets. Three key datasets used from this source are Edubase schools information for Ofsted ratings, 2014 Department for Transport journey time statistics and 2015 English Indices of deprivation. Without the London focus of the GLA datastore, information on gov.uk is often harder to locate and in many cases is at a higher level of aggregation than required with most information at borough or national level.

3.2.3 esri

Data for average journey times and illness affecting employment was obtained at lower super output area, a lower level of aggregation than ward. The different geographical level use in the two datasets created the need for a mapping. The open data section of esri's ArcGIS website (esri, 2017) offered this mapping document in csv format which could be incorporated into the data import software to allow an arithmetic mean to be determined at ward level.

3.3 APIs

A subset of London data is also available via various APIs. For transport information Transport for London have extensive APIs available, unfortunately none of these were able to provide any of the datasets required for the project, though other formats offered by TFL were used. The use of APIs became focussed on collecting information on venues to obtain the data for the 'community vitality' and 'access to cultural spaces' datasets. For this data Foursquare was the primary candidate with venue information available with no cost implications.

3.3.1 Foursquare

Foursquare is a social media platform based on location intelligence. The platform has information on more than 105 million locations mapped worldwide and over 50 million users per month. (Foursquare, 2018). With the extensive database of venues with location information, Foursquare could be used to look at two of the indicators which feed into the Community Vitality and Participation index:

- Access to cultural spaces
- Popularity of bars and restaurants

To measure these two indicators, Foursquare's venue API can be used to return list of venues within a radius of a specific point provided in latitude and longitude. By using the three venue types, "culture", "food" and "nightlife", a list of venues fitting those types can be captured via the API. In the implementation phase, the list

of venues containing the point location of the venues can be mapped to the relevant ward and numbers aggregated to give an indicator of volume with which to measure this category.

3.3.2 Google Maps

The places API from Google Maps was queried and code written to extract venue data. However, after obtaining some basic search results it was decided that Google's pricing policy and rate limits made this data source unfeasible for the scope of the project.

3.4 Shape Files

For choropleth mapping, shape files are needed to visualise the polygon shapes which define administrative or political boundaries. Shape files usually consist of properties of the shape such as name and size along with a geometry feature which defines the boundaries of a given shape via the coordinates of their vertices. For this project, the relevant shapefile is key to creating the final interactive map and will also be used to visualise the data and support the model building process.

3.4.1 London Ward shapefile

The key shapefile for the project is taken from the London datastore and provides the polygon geometry of the London wards. This is a .shp file where the point coordinates which make up the polygon for each ward are in the Ordnance Survey grid reference coordinate reference system. With point geometries more often listed in latitude and longitude, there will need to be some reprojection of coordinates to enable certain datasets to match with the shapefile.

3.5 Data availability issues

In part thanks to the Greater London Authority's Datastore, a wealth of information is available at London Ward level. Combining this with information from the Foursquare API and Department for Education schools data provided a significant body of information in which to model London as a multiple dataset. Data availability was such that datasets for each of the indicators feeding into the six indices were available and it was not necessary to use substitute indicators that differed significantly from those originally planned.

3.5.1 Timeliness

Some of the datasets related to different years, often linked to census years and administrative or electoral changes. Wherever possible the most recent data has been used to synchronise as closely as possible with the 2017 data used for the median house price data. For example, Emission data is from 2011 as this is the most recently published at ward level. The London Air Quality daily feed run by King's College does not have enough coverage to sufficiently differentiate over 600 wards. Ideally this data would have been from the same year as the house price information.

3.5.2 Boundary changes

Ward boundaries were re-drawn in three London Boroughs in 2014 meaning that a function had to be written that, for data pre-dating 2014, would map old ward codes to the new ward that contained the largest section of the newly defined area. Newer data would also have been helpful here but the commonality of area between the old and new codes in the mapping should ensure that the data is representative of the new area.

3.5.3 API rate limits

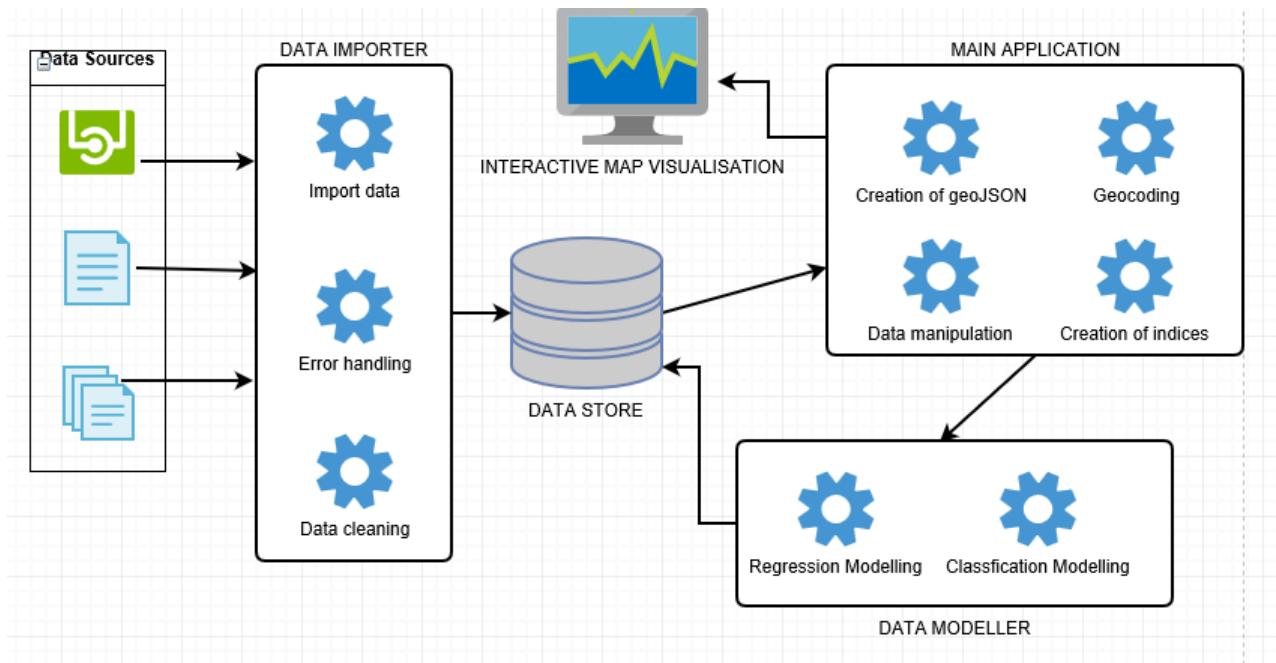
API limits and costs imposed limits on the depth of information that could be obtained from these sources. Whilst I was able to obtain the necessary Foursquare venue and venue location information, venue ratings and check-ins were subject to stringent daily limits which meant that obtaining this information would have taken weeks of daily iterations or significant costs neither of which were feasible for this project. I investigated the possibility of using Google Places API but the new pricing model introduced this year also made this unfeasible. In a commercial setting it may be possible to increase the amount of data that can be obtained from some of the APIs.

Chapter 4

Design

4.1 Architecture of the specification

4.1.1 Archictecture diagram



4.1.2 Architecture choices

For this project two main technologies are being used, Python and Javascript. The majority of the software code has been written in Python, specifically for the data import processes, for the main application which runs the analysis and geocoding and the data modelling component. Python was chosen for these components for a number of factors. Firstly for the data importer, Python has excellent capabilities for data processing allowing the data to be collected and cleaned leveraging the Pandas module. For the main application, Python's geocoding and spatial capabilities were important factors in the choice. The ability to import shape files and reproject coordinate systems leveraging Geopandas and Shapely allowed more flexibility in which datasets were suitable for the project and the option to export to GeoJSON format facilitated the data visualisation. For the visualisation stages Javascript embedded in a html file was chosen. For this final component of the project Javascript was preferred to Python due to the interactive element required on the map. With the aid of the Leaflet.js package in Javascript I was able to make use of more map features to present any end users with a more complete user experience.

4.2 Components

The software architecture design for the project has been created with the aim of being able to isolate individual elements in the interest of performance. The system is not dependent on processing a live data feed so it is important that the component that imports the data does not have to run everytime a user would wish to access the interactive map visualisation. The design means that each component can be modified and without affecting the other parts.

4.2.1 Data inputs

The data input layer is the base layer of the system. This consists of several different types of input including API feeds from the Foursquare platform, csv files collected directly from urls and other data files which have been pre-collected and loaded into the data store. This layer is the most likely to change moving forward as new and updated data sources are published or become available in alternative formats.

4.2.2 Data importer

The data importer software is written in Python and is designed as a series of functions. Each function imports one of the required datasets. Setting these up as separate functions gives the option to run imports individually. This is an important requirement of the system as certain imports take a considerable amount of time. The iterative nature of the API imports combined with daily rate limits mean that it is only feasible to run the API imports from Foursquare on at most a daily basis. As the system is not reliant on a live feed, the data import software could be run as an overnight batch process.

4.2.3 Data store

The data store is central to the architecture to safeguard performance levels. As the data imports are best suited to a batch process, the cleaned data files should be stored in the data store for the main application software to run efficiently. The data store is also used for any pre-collected data which cannot be collected directly from a url or API. This imports the data for the 18 base indicators which create the 6 well-being indices.

4.2.4 Main application

The main application software performs a variety of tasks and is written in Python. The first task that the software performs is to take the data that has been imported for the 18 base indicators from the data store. The main application then performs any data manipulation and applies a set of geocoding and spatial functions to standardise the data into a format where everything is aggregated for London Ward geometry. Having standardised, the software then creates the 6 well-being indices using functions drawing on Principal Component Analysis and mean calculation functions which also normalise the data to ensure that each score is between 0 and 100. These 6 indices are then combined with data showing median house prices and the related quintiles and passed through the regression and classification models chosen through the data modeller component. The final task performed by this component is to create a geoJSON file including the well-being index scores, median

house price information, spatial data and predicted regression and classification values. This file contains the data used in the visualisation.

4.2.5 Data modeller

The data modeller is a Python application which takes the data containing the 6 well-being indices and uses a range of modelling techniques to try and obtain the best model for the data. This is not an automated process which chooses the best score because interpretability is also a strong point of interest for this particular problem. The data modeller outputs the predicted values for the chosen model and returns them to the dataset for further use in the main application and in the interactive map. The data modeller is currently designed so that the solution developer would select the final models to be used in the map.

4.2.6 Interactive map

The final component of the software architecture is the interactive map which is built in javascript and html. This component builds a framework for the map using tiles from the MapBox service. A javascript function then takes the geoJSON file and add this as a layer to the map. This part of the architecture can be seen as the user front end as the visualisation allows users to hover over each ward to see the scores, actuals and predictions along with features such as a zoom facility.

4.3 London house prices as a regression problem

London is known for having very high real estate values but we can see from the map below that there are specific wards, particularly in the western central area, where median house prices are significantly above the Greater London area as a whole.

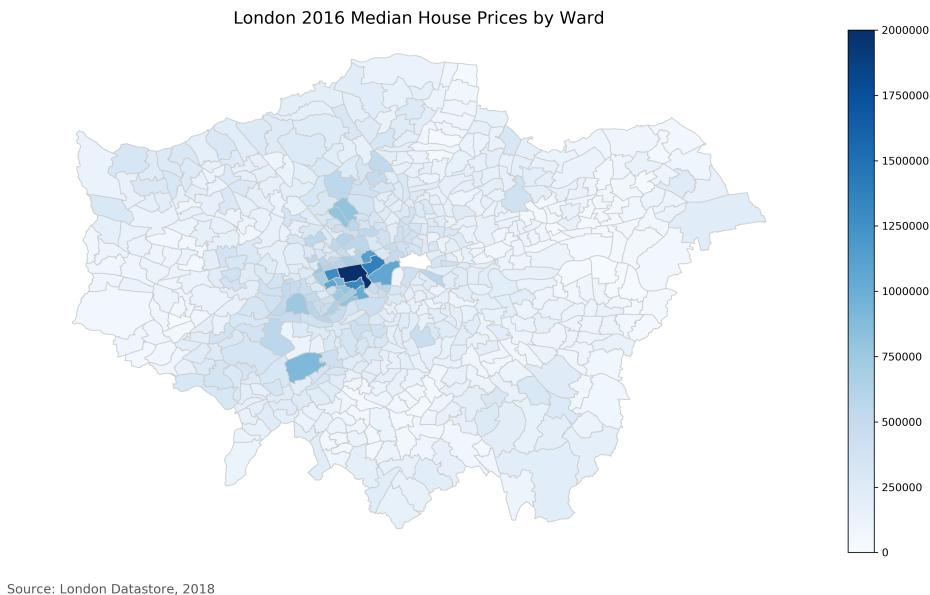


FIGURE 4.1: Choropleth map showing median house prices for each ward in London (based on values at the end of 2017).

The choropleth map would suggest a high number of wards fall at the lower end of the range of median house prices with a small number with significantly high prices. From the pairwise plot below, the shape of the histogram of median house prices by ward would suggest that a power law may best describe the distribution.

$$p(x) = x^{-4.314410441437625} \quad (4.1)$$

For the indices, the histograms show most of the six as being at least close to a normal distribution. This is less clear in the case of the Safety and Security and Community Vitality and Participation indices with Safety and Security showing a large number of high values and Community Vitality and Participation showing many low scores. This can be attributed to a small number of wards where crime is high and that wards in central London have much greater access to both cultural and social spaces as indicated on the maps below. In both cases, a power law may best describe these distributions.

This is less clear in the case of the Safety and Security and Community Vitality and Participation indices with Safety and Security showing a large number of high values and Community Vitality and Participation showing many low scores. This can be attributed to a small number of wards where crime is high and that wards in central London have much greater access to both cultural and social spaces as indicated on the maps below. In both cases, a power law may best describe these distributions.

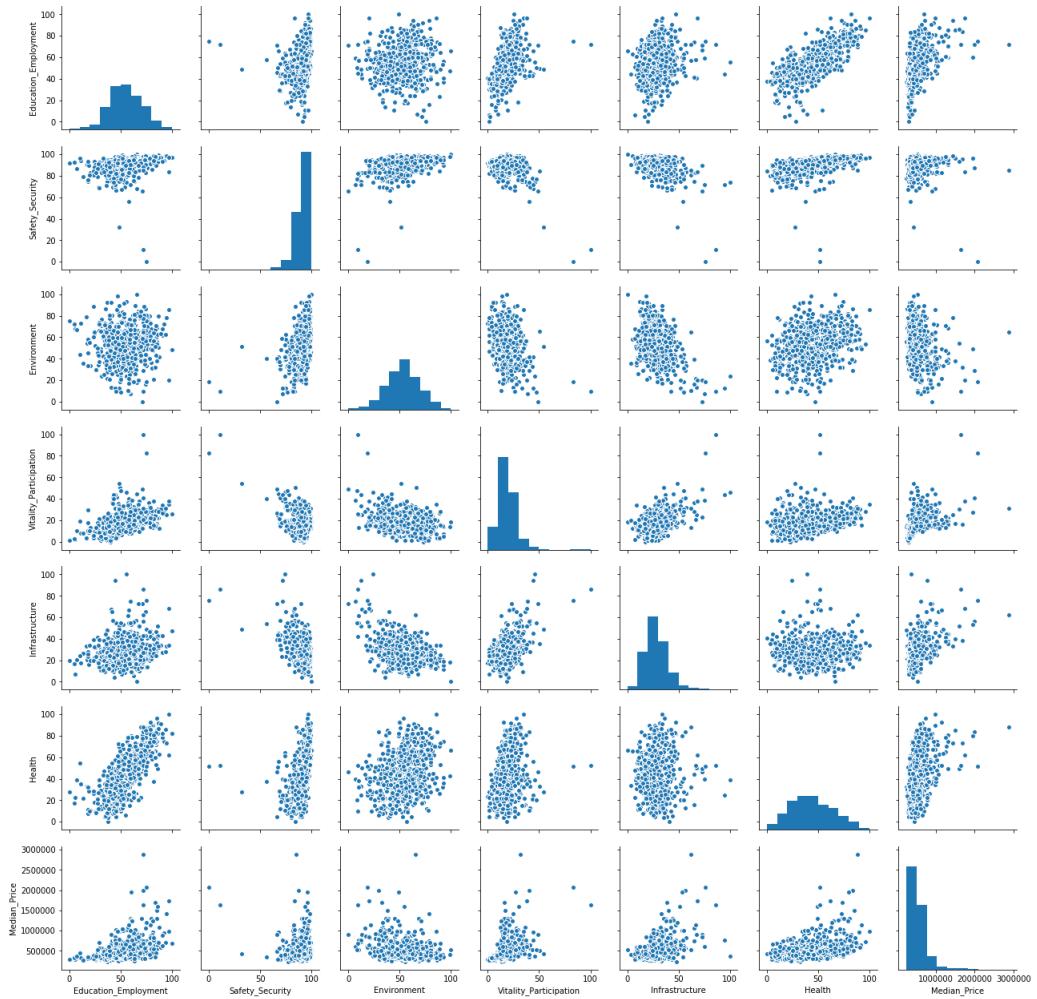


FIGURE 4.2: Pair-wise plot of the six well-being indices London median house prices with histograms showing the distribution of each index and house prices.

TABLE 4.1: Correlation Matrix for the six indices and median house prices (000s)

Correlation Matrix	Education and Employment	Safety and Security	Environment	V
Education and Employment	1	0.246739481	0.078228197	0
Safety and Security	0.246739481	1	0.394489799	-0
Environment	0.078228197	0.394489799	1	-0
Vitality and Participation	0.422030673	-0.462644114	-0.280237241	1
Infrastructure	-0.050583652	-0.418567441	-0.781988848	0
Health	0.746026055	0.382769813	0.306477755	0
Median House Price	0.447882095	-0.154480959	-0.249970294	0

The indices show a mixture of levels of correlation with each other. Environment and Infrastructure have a strong negative correlation, possibly because a central location generally has a high Infrastructure score with more transport options and shorter journey times, whereas Environment scores better in Outer London where congestion is lower. The strongest correlation is between Health and Education and Employment.

4.4 London house prices as a classification problem

The choropleth map below shows a different picture of London house prices being based on quintiles rather than monetary values, here the fifth quintile covers a larger range of values than the other quintiles due to the extremely expensive areas such as Knightsbridge and Belgravia where the median house price reaches over £2.8 million, 10 times as much as North End ward in Bexley.

For the classification problem the aim was to try to find a model that has both a good fit and high interpretability. However, models that often have high accuracy but low interpretability such as neural networks were also used to both discover and validate the best classification model for the problem.

By using the pair-wise plot and colouring the points by quintile, it is clear that there is no clear separation of quintile for any of the indices with a more gradual movement from first to fifth quintile for most indices. Environment and Safety and Security have an almost inverse relationship with the quintiles. Because the majority of the high-priced wards are centrally located, this may relate to a lack of greenspace and higher pollution centrally as well as providing high-value targets for crime.

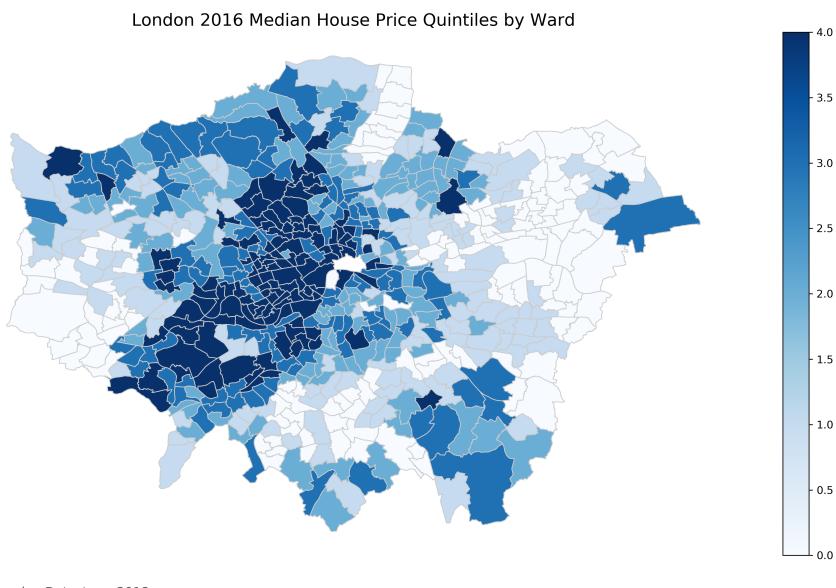


FIGURE 4.3: Choropleth map showing median house prices by for each ward in London by quintile (based on values at the end of 2017).

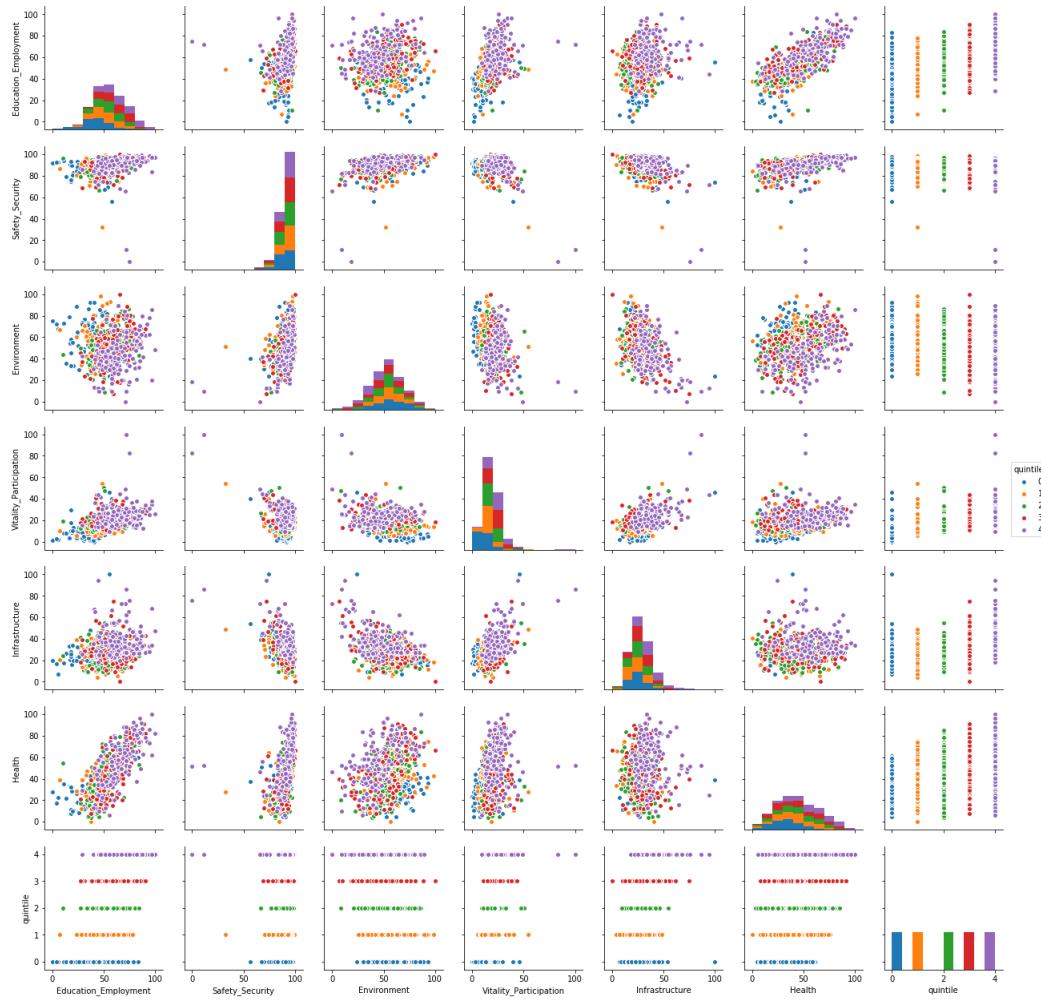


FIGURE 4.4: Pair-wise plot of the six well-being indices London median house prices with histograms showing the distribution of each index and house prices. Points are coloured on the basis of the quintile of the median house price within London ward.

Chapter 5

Implementation

5.1 Data collection and cleaning

The key foundation of the project was being able to collect the data that would allow the creation of the well-being domain scores. Collecting the required data and shaping it into the formats that were needed was, in places, one of the most challenging aspects of the project and the one that required the largest proportion of time. This part of the project presented several challenges that needed to be overcome.

5.1.1 Collection of data files

For ward level data, the Greater London Authority's datastore is a primary data source. A significant proportion of the data for the creating of the well-being indicators could be accessed via this source.

5.1.2 Collection from APIs

The API that was key to the success of the project was that of social media platform Foursquare. To represent community vitality, the aim was to create a measure for the restaurants and bars in each ward. Foursquare is a platform based around venue information so holds the information required for this measure. Foursquare also holds information on cultural venues. Therefore the measure of 'Access to Cultural Space' could also be obtained from this platform. To use the API to obtain venue information you are required to search for venues within a specified radius of a certain point, given in latitude and longitude. The challenge with Foursquare was the the API rate limits which limits each search to a maximum of 49 searches. There are also daily and monthly rate limits to stay within. To obtain results of all bars, restaurants and cultural venues across Greater London with these limits an iterative algorithm had to be created which could travel accross London by latitude and longitude, taking in small enough areas so as to not exceed 49 results a time. This function can be seen in figure 5.1. To achieve this, a latitude and longitude bounding box was created around London and a nested loop iteration of 50 points of latitude and 50 points of longitude was undertaken. The areas created by this had some overlap so as not to have missing areas of the capital. With a counter to check if the search limit of 49 venues was exceeded, this search algorithm put the list of venues into a pandas dataframe and de-duplicated for where areas had overlapped.

Because of the rate limits, running the algorithm for such a large area could only be done on a daily basis.

```

def importVenue(category, filename):
    df_Foursquare = pd.DataFrame()
    FrameList = []
    limit_reached = 0
    for i in range(1,50):
        for j in range(15,750,15):
            client_id = "X01Q0I002J0E0K0X24HBOYPHW2DSWJH5E1BN2AOZG2NRZVPK"
            client_secret = "G3GSIAZR4WXLYOHW4VP1JHTAMLH23EZXCWZDVPCTYY2R04V"
            lat = (51.2 + (i/100.0))
            long = (-0.25 + (j/1000.0))
            if category == 'cultural':
                category_id = '4dd4b7104d754a06370d81259' # cultural space
            elif category == 'restaurant':
                category_id = '4dd4b7105d754a06374d81259' # food
            elif category == 'bar':
                category_id = '4dd4b7105d754a06376d81259' # drink
            distance = 450
            requested_keys = ["categories","id","location","name"]
            url = "https://api.foursquare.com/v2/venues/search?ll=%s,%s&intent=browse&radius=%s&categoryId=%s&limit=49&client_id=%s&client_secret=%s&v=%s"
            % (lat, long, distance, category_id, client_id, client_secret, time.strftime("%Y%m%d"))
            resp = requests.get(url)
            dataResp = resp.json()
            if dataResp["response"]["venues"] != []:
                data = pd.DataFrame(dataResp["response"]["venues"])
                df_FoursquareIteration = pd.DataFrame(data)
                if len(df_FoursquareIteration) == 49:
                    limit_reached = limit_reached + 1
                df_FoursquareIteration["categories"] = df_FoursquareIteration["categories"].apply(lambda x: dict(x[0])['name'])
                df_FoursquareIteration["lat"] = df_FoursquareIteration["location"].apply(lambda x: dict(x)[("lat")])
                df_FoursquareIteration["long"] = df_FoursquareIteration["location"].apply(lambda x: dict(x)[("lng")])
                FrameList.append(df_FoursquareIteration)
    df_Foursquare = pd.concat(FrameList)
    df_Foursquare.drop_duplicates(subset=['id'],keep=False)
    if limit_reached > 0:
        print('Limit Reached')
        print(limit_reached)
    columns = ['name','categories','lat','long']
    df_culture = pd.DataFrame(df_Foursquare, columns = columns)
    df_culture.to_csv(filename, index=False, encoding='utf-8')

```

FIGURE 5.1: Python function written to import data from Foursquare. This function iterates for latitude and longitude to traverse London and return venue information for one of three categories ("cultural", "food" or "nightlife") without exceeding daily API limits.

5.2 Well-being domain scores

The motivation for creating composite index scores 6 different measures of well-being was two-fold. Firstly creating a score for each index over a pre-defined scale would allow users of the front-end interactive map compare scores across wards and give those scores some contextual meaning and significance. With 6 indices rather than the original 18 datasets, the amount of information presented to the user would not be too large to prevent a non-technical audience from interpreting the information on the map. The second reason for creating the indices was to help to try and create an interpretable model rather than one requiring significant dimension reduction. Whilst some less interpretable models will be tested to validate the quality of the quintile classification model, ideally the project and the user front end should allow us to draw some conclusions about any link between well-being and median house prices in London.

For each index three appropriate datasets were chosen and then reduced to give a single score for each ward. The 3 datasets for each index were standardised using a function in Python and those which related to a negative indicator such as emissions or crime were multiplied by a value of -1. Using this technique of standardisation gave equal weighting to each of the three indicators. Because of the equal weighting given to each indicator, these can be seen as substitutable within the context of the index which is an important factor in the choice of technique used to reduce the indicators to a score. With indicators that can be assumed to be substitutable suitable techniques would include Principal Component Analysis (PCA) and an arithmetic mean as discussed by Pareto and Adriano (Pareto, 2013). For the 6 indices, a mixture of the two techniques have been used. For those indices where the 3 indicators have some level of correlation (>0.4) the first Principal Component has been used to try and extract the largest amount of variance within the three indicators. Where the indicators for an index have low levels of correlation for at least one of the three

datasets, the arithmetic mean has been used. This approach gives us three indices which use PCA (Safety and Security, Infrastructure and Health) and three which use the mean (Education and Employment, Environment and Community Vitality and Participation).

This approach tries to maximise the information contained in all of the datasets. To create an interpretable and comparable score for each domain the resulting single scores have been normalised and then rescaled to be between 0 and 100, this is the score that is presented in the user front end.

5.3 Building a regression model

To begin the task of trying to model the median house prices from the well-being indices, a simple multiple linear regression model was used with all six indices as predictors. The results show that this model explains less than half the variance with an R squared value of 0.39. There is enough to suggest that some form of linear model may provide a reasonable model of this problem. To validate the model K-fold cross validation was used with k=7 with accuracy scores and mean squared errors average across the seven iterations.

The summary statistics for a simple linear model would suggest some level of statistical significance for all of the indices with Community Vitality and Participation and Education and Employment less significant than the others. Infrastructure and Health are the most significant indices here and they are also the most influential on the model. Safety and Security is the only index where there is a negative impact on house prices in the model.

TABLE 5.1: Results for regression models on London median house prices

Predictor	coef	std err	t	p
const	46.9752	135.66	0.346	0.729
Education_Employment	1.4541	0.759	1.915	0.056
Safety_Security	-4.5924	1.391	-3.301	0.001
Environment	2.0667	0.712	2.902	0.004
Vitality_Participation	2.2165	1.227	1.807	0.071
Infrastructure	9.2899	0.769	12.075	0
Health	6.2779	0.624	10.064	0

Next a polynomial model was tried with degree two. This model explained almost none of the variance with an R sq value of very close to 0. The mean squared error also increased dramatically. The polynomial model does not seem to be a strong candidate for solving the problem.

The next models involved some feature reduction based on the p values of the indices in the original linear model. This meant that first Education and Employment were removed and then Community Vitality and Participation. This resulted in a small improvement in the model fit but little improvement in the MSE.

With the median house price distribution suggesting a power law, a log transformation was performed on the data to try and linearise it with the linear regression model then re-run. This transformation dramatically increased the R sq value of the

TABLE 5.2: Results for regression models on London median house prices

Model	R squared	Average MSE
Linear regression	0.396	35326.75
Linear regression with polynomial degree 2	0.007	59013.81
Linear regression with 5 dimensions	0.473	33907.16
Linear regression with 4 dimensions	0.443	34608.08
Linear regression on log-transformed data	0.641	4944.06

model, to 0.64, an increase of 0.25 from the equivalent model without the transformation. The model run on the log transformed data explains an additional 25 percent of the variance within the model. The exponent of the MSE is also a substantial reduction. This returned the following model:

$$\log(\text{Medianprice}) = 0.00300393 * I1 + 0.00201057 * I2 + 0.00191012 * I3 + 0.0081844 * I4 + 0.01301158 * I5 + 0.00742296 * I6 \quad (5.1)$$

where I1 = Education and Employment, I2 = Safety and Security, I3 = Environment, I4 = Community Vitality and Participation, I5 = Infrastructure and I6 = Health

The equation shows that Infrastructure has the largest impact on the median house prices in the model followed by Community Vitality and Participation and then Health. All indices have a positive impact on median house prices in the model, Environment is the least influential index

As the best performing regression model this will be used for prediction in the final visualisation.

When the predictions were given the inverse transformation and validated against the actual values, the majority of wards showed that this model had good predictive capability although there were a handful of wards where the prediction had a significant error margin. The wards with significant errors tended to be those which had a much higher or lower score in a particular domain as compared to the neighbouring areas.

5.4 Building the best classifier

The first model investigated was a logistic regression model. With strong interpretability well-performing logistic regression model would provide a good solution to the problem. The model had a classification success rate of 44 percent, showing a reasonable level of predictive capability across a five value classification problem. The confusion matrices provided by the seven cross-validation iterations show that model performs strongly in the first and fifth quintiles but finds it much more difficult to classify points in the second, third and fourth quintiles. From the pair-wise plots this is unsurprising as the range of median house prices is much smaller for the middle three quintiles and many of the well-being indicators are of similar magnitude.

For the second model a k-nearest neighbours classifier was used, this achieved a 40 percent classification success rate and was outperformed by the logistic regression model.

TABLE 5.3: Results for Neural Network classifier with Logistic activation function and different numbers of nodes in the hidden layer.
Based on average scores using k-fold cross validation with k=7.

Number of nodes in hidden layer	Test Score
5	0.469
10	0.477
15	0.474
20	0.490
25	0.458
30	0.471
35	0.482
40	0.451
45	0.502
50	0.488
55	0.498
60	0.496
65	0.483
70	0.501
75	0.482
80	0.480
85	0.456
90	0.490
95	0.471
100	0.490

A model using a Gaussian Naive Bayes algorithm performed very slightly above the logistic regression, however, given the better interpretability of the logistic regression model, the 0.2 percent gain would not be enough to justify using this model for the solution.

The next model investigated was a decision tree, with an average classification percentage of 38, this performed significantly below the rate of the logistic regression model.

Although a single decision tree was the worst of the classifiers used so far, the next models investigated were ensemble models. A random forest was tried on the data. With 10 trees this performed below the level of the logistic regression model, but when tuned to use 100 trees, this produced the best classification score so far at just under 46 percent. The random forest was therefore a candidate to use as the classification model. Another ensemble method, gradient boosting, also performed better than the logistic regression model but was not able to match the random forest's correct classification rate.

The final model used was a Neural Network using a single hidden layer. A process of tuning this model involved trying four activation functions, of which the logistic function performed best. Different values were used for the number of nodes in the hidden layer with 45 giving the highest classification rate. By using a tuned Neural Network, a 50 percent classification rate was achieved. This was a significant improvement on the other models used.

To take advantage of both interpretability and classification success rates, it was decided to display the predicted quintile values for both the Logistic Regression

TABLE 5.4: Results for Neural Network classifier with 45 nodes in the hidden layer and different activation functions. Based on average scores using k-fold cross validation with k=7.

Activation Function	Test Score
identity	0.380
logistic	0.485
tanh	0.445
relu	0.446

TABLE 5.5: Results for classification models on the quintiles of London median house prices

Model Type	Training Score	Test Score
Logistic Regression	0.470	0.440
Decision Tree	1.000	0.388
K-Nearest Neighbours	0.614	0.407
Random Forest	1.000	0.459
Gradient Boosting	0.987	0.448
Naive Bayes	0.465	0.442

model and the Neural Network in the final map visualisation. The results of testing each of the different models using k-fold cross validation with k=7 were:

5.5 Encoding as geoJSON

To use the information from the creation of the well-being domains and the predictions obtained from the models in a javascript based visualisation, the information needed to be converted from a pandas dataframe in Python to a geoJSON file in javascript. A geoJSON file is a variation on the JSON format as defined below:

Whilst a relatively straightforward task within the geopandas module in Python, geoJSON accepts coordinates in latitude and longitude and the shapefile being used has the coordinates defining the ward polygons in Ordnance Survey grid references. To change this to latitude and longitude required the creation of a function that would take each polygon in the ward geometry in the geo data frame and break this down into the individual points which define the polygon. Once separated into coordinate pairs, these pairs could be reprojected to be latitude and longitude. The function was then required to recombine all of the coordinate pairs into the correct polygon geometries.

```

1 | def geometricTransform(gdf):
2 |     point_list = []
3 |     poly_list = []
4 |     for index, row in gdf.iterrows():
5 |         for pt in list(row['geometry'].exterior.coords):
6 |             inProj = Proj(init='epsg:27700') # lat long
7 |             outProj = Proj(init='epsg:4326') # OS National Grid
8 |             new_Point = transform(inProj, outProj, pt[0], pt[1])

```

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {"type": "Point", "coordinates": [102.0, 0.5]},
      "properties": {"prop0": "value0"}
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [102.0, 0.0], [103.0, 1.0], [104.0, 0.0], [105.0, 1.0]
        ]
      },
      "properties": {
        "prop0": "value0",
        "prop1": 0.0
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
            [100.0, 1.0], [100.0, 0.0] ]
        ]
      },
      "properties": {
        "prop0": "value0",
        "prop1": {"this": "that"}
      }
    }
  ]
}
```

FIGURE 5.2: geoJSON specification showing Point, LineString and Polygon geometry objects.

```

9     point_list.append(new_Point)
10    poly = shapely.geometry.Polygon(point_list)
11    point_list = []
12    poly_list.append(poly)
13    gdf['geometry'] = gpd.GeoSeries(poly_list)
14    return(gdf)
15
16 # output geoJSON to javascript file defining a js variable for
17 # interactive map
18 def toGeoJSON(gdf):
19     json_file = gdf.to_json()
20     with open('dataset.js', 'w') as openfile:
21         openfile.write('var dataset = ' + json_file + ';')
```

The information was then in the required format to be exported to a geoJSON file.

5.6 Interactive maps

For the production of the interactive map which serves as the user front end for this project, the information was moved from the Python environment via geoJSON and into javascript functions embedded into html. This decision was made on the basis of the additional functionality and interactivity that could be smoothly added to the map in javascript. With less experience of javascript as a programming language, the leaflet.js tutorial (Agafonkin, 2017) was an excellent starting point.

The map has key functionality added. This includes a zoom facility and hovering over any of the 625 wards will display the median house price value and quintile, the scores for each of the six well-being domains and the predictions from both the regression and classification models.

Chapter 6

Testing and validation

6.1 Testing the data inputs

The data input software collects information from several urls where open data is published, particularly on the Greater London Authority datastore. To ensure that the removal or changing of one of these urls does not cause the software to crash the data importer has been written as a series of functions so that each import can be run individually and unit testing can be done. This involves the use of error handling to inform the user that a specific data import has failed so that this can be isolated and investigated.

During the course of the project one link changed twice and one was removed. Because of the way the software was written, this could be easily identified and resolved. With the data importer writing information to the data store once imported, a failure in any of the import processes will not prevent the rest of the software from running if the data store has previously been populated.

For API requests, empty records are ignored and the import process will continue to run and return the populated records.

6.2 Testing the models

To test the quality and accuracy of the data models, cross-validation was employed. In each case k-fold cross-validation was the chosen method and was run with $k=7$ for all models. All tests on the models were run using the same starting point to ensure that the experiments were replicable.

6.3 Validating the outputs

To validate the outputs and the values showing on the map, a spot-checking technique was used. I took wards where I had an idea as to what the values should look like such as the ward in which I reside and the ward in which I work. This method alerted me to an early problem with the Education and Employment index where not all schools were being included in the import. This validation technique also confirmed that the information on the interactive map was correct. I was also able to cross-reference some of the information, for instance, Zoopla's heatmap which featured in section 2 of the report was a valuable resource to confirm that the house price choropleth maps were showing the correct values as it was relatively straightforward to compare locations between the two.

Chapter 7

Conclusions and evaluation

7.1 Lessons learnt

The project showed that well-being indicators can seem to go some way towards providing us with a model for house prices with the prediction capability and success of both the regression and classification models considerably above what would be expected from a set of unrelated data. The regression model tended to struggle more on the bottom and top quintiles whereas the classification models found the middle three quintiles more difficult to predict. The median house prices by ward definitely seem to follow a power law and the regression model was more effective when using a log transformation to make the data more linear. Infrastructure was the most influential of the well-being indices suggesting that London residents value good public transport access and short journey times to work. Wards which had a high score for infrastructure when their neighbours did not were amongst the hardest for the model to predict. With median house prices generally highest in the western central areas of London, there was less influence from the well-being scores which tended to be higher for Outer London wards. For the Environment indicator in particular, being away from the centre of London helped wards score highly due to lower pollution levels from less traffic and commerce in general. There is further potential for the link between house prices and well-being to be investigated with the use of more datasets from a greater range of sources.

The approach to the project ensured that a data-driven visualisation of the findings could be produced which was suitable for a range of users. The reduction of 18 datasets into six indices meant that the models were more interpretable than had all 18 been used to create the models. The use of geospatial analysis to analyse the data and present the results gives an excellent visual representation of what the data is showing.

I have enjoyed working on this project and would be really keen to take this work further perhaps looking at some of the extensions outlined in section 7.2. By completing the project, I have significantly improved my fluency in Python especially around the use of Pandas and Geopandas. The geospatial data elements of the project are ones that I have found particularly interesting and I hope to be able to make more use of these in the future.

7.1.1 Summary of findings

- There is some evidence of a link between well-being and house prices
- Regression models were unable to make good predictions for high-value wards with the dataset assembled

- Classification models predicted the top and bottom quintiles well but struggled more in the middle of the range
- More work is needed to produce a model of house prices based on well-being indicators that has very high levels of accuracy
- Innovative ways of data collection are required to improve the range available at ward level and at lower levels of aggregation

7.2 Possible developments

An extension of this project that would be an interesting addition to the results already displayed would be to look at whether changes in the well-being indices were related to changes in house prices. Such analysis could potentially provide insight on whether increasing house prices fuel rises in well-being indicators or whether improvements in local well-being increase demand for houses in a particular area. Correlation here would support causation, either in the case of increased demand for houses where "wellness" is high or that when house prices rise, an area will gentrify and well-being indicators will rise as a result. To achieve this, more data would need to be found that could represent all of the well-being indicators for regular intervals over a significant period of time. Given that some of the datasets are only released every few years, this may require looking at some innovative solutions of data collection with crowdsourcing and image analysis possible avenues of investigation. For instance, recent work around the analysis of social media photos has shown how cultural capital has a strong relationship with house prices and how cultural inequality can widen economic inequality as neighbourhoods become gentrified (Hristova, Aiello, and Quercia, 2018). This project has also focused on house prices, yet the rental market may also have an impact on these along with changes to how properties are used such as the increase of companies such as airbnb (Quattrone et al., 2016) Another area to look at would be how the flow of residents, particularly between neighbouring wards, can affect the house prices. For instance, if the perception of a particular area declines or potentially undesirable changes occur (eg. large construction projects), how does the inflow and outflow affect the prices. This may be an area where the use of complex networks could assist with analysis. The other area of interest would be to go beyond the ward geographic area and use the Lower Super Output Area (LSOA) which are small census areas of around 1,500 people, grouped in such a way that the areas are seen to have similar characteristics. Whilst the full range of data used in the project is not currently available at LSOA, two of the datasets used were mapped upwards from this level and would be a logically direction for this work to develop. If being run as a commercial application, a paying subscription to the Foursquare and/or Google API would allow further analysis around assessing venue popularity to feed into the Community Vitality and Participation domain.

Appendix A

Appendix A

Write your Appendix content here.

Appendix B

Appendix B - Additional tables and charts

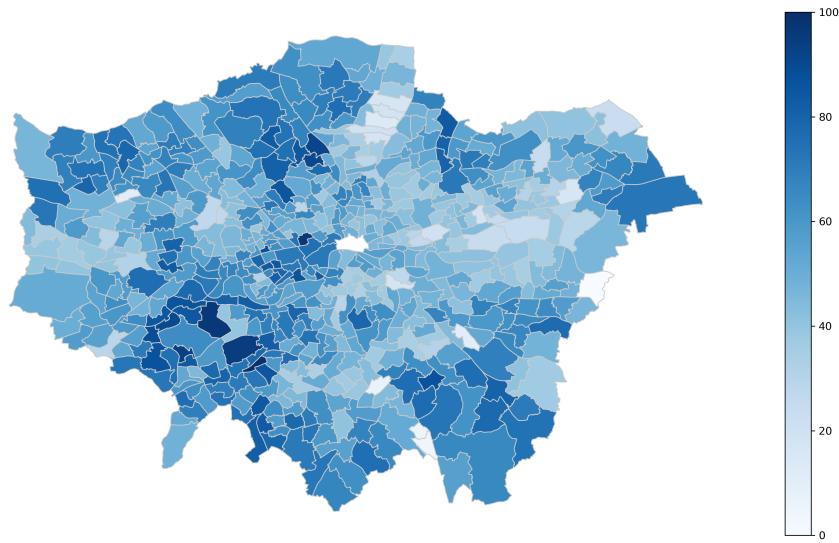


FIGURE B.1: Choropleth map showing the Education and Employment index

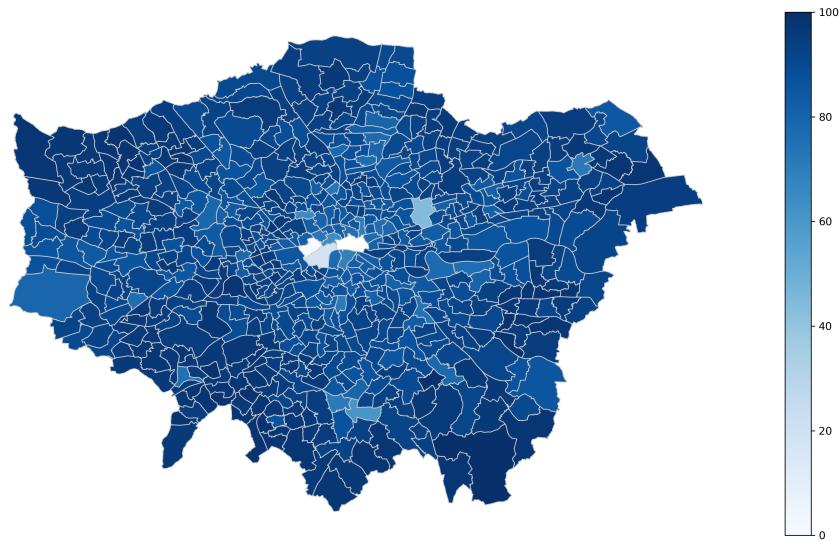


FIGURE B.2: Choropleth map showing the Safety and Security index

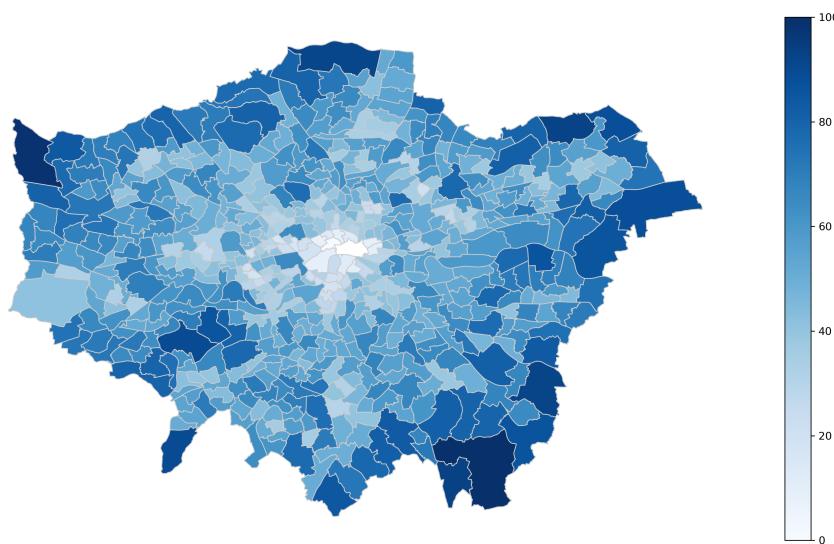


FIGURE B.3: Choropleth map showing the Environment index

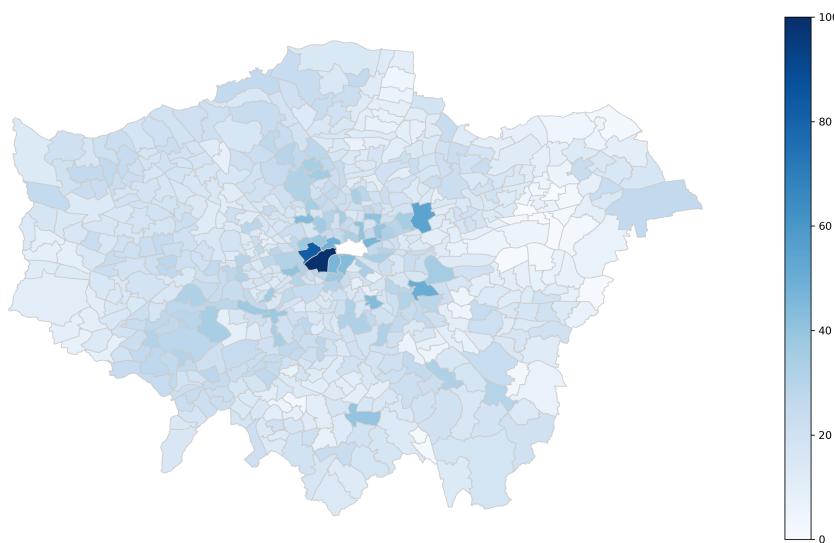


FIGURE B.4: Choropleth map showing the Community Vitality and Participation index

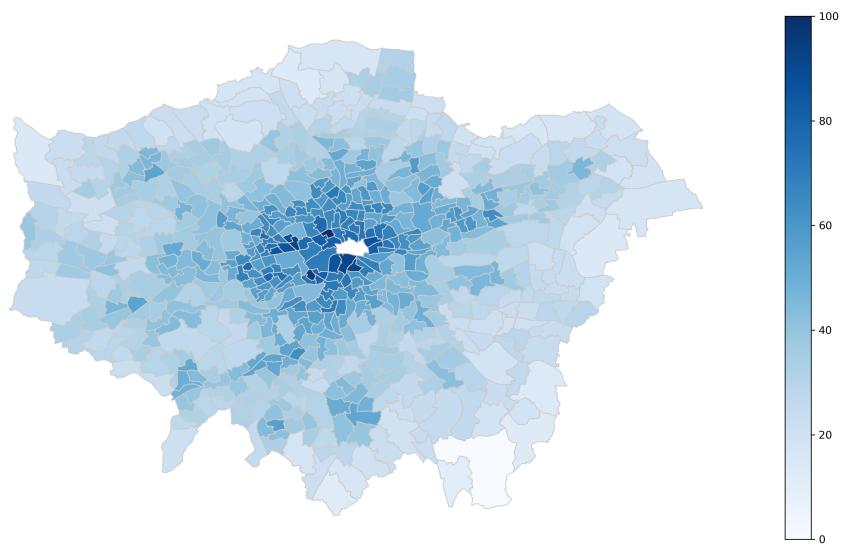


FIGURE B.5: Choropleth map showing the Infrastructure index

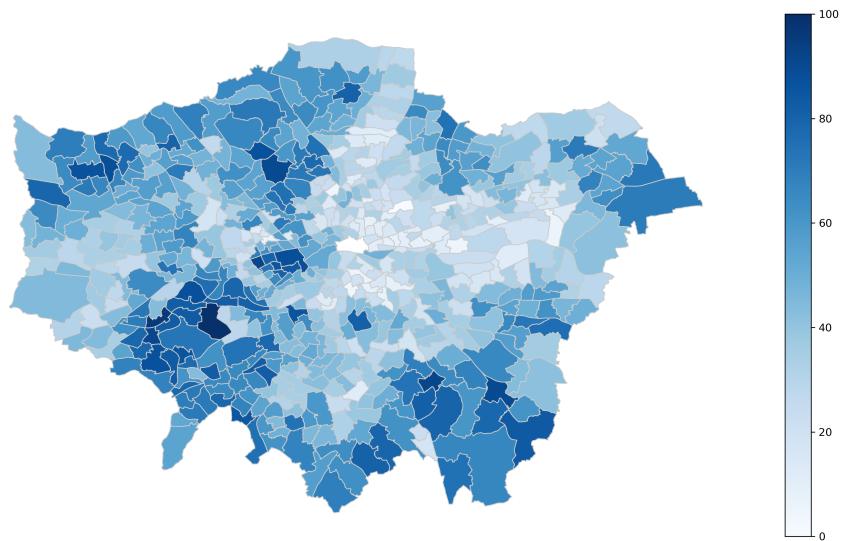


FIGURE B.6: Choropleth map showing the Health index

Appendix C

Appendix C - Code

Main Application

```

1 import pandas as pd
2 import numpy as np
3 import geopandas as gpd
4 from sklearn.preprocessing import StandardScaler, MinMaxScaler
5 from sklearn.decomposition import PCA
6 from shapely.geometry import Point, Polygon
7 from pyproj import Proj, transform
8 import shapely
9
10
11 def standardise(df):
12     # standardises values in a dataframe and returns new values to the
13     # data frame
14     STscaler = StandardScaler()
15     scaled_values = STscaler.fit_transform(df)
16     df.loc[:, :] = scaled_values
17     return(df)
18
19 def tablestyle(df):
20     th =
21         [('font-size','11px'),('text-align','centre'),('font-weight','bold')
22         ,('color','#6d6d6d'),('background-color','#f7f7f9')]
23     td = [('font-size','11px')]
24     styles = [dict(selector='th',props=th),dict(selector='td',props=td)]
25     df.style.set_table_styles(styles)
26     return(df)
27
28 # calculates domain score (out of 100) using Principal Component
29 # Analysis
30 # used where there is significant covariance for the three indicators
31 def domain_indicator_PCA(df, indicator_name):
32     # function runs PCA on 3 indicators in a domain and returns a score
33     # for domain out of 100
34     NMscaler = MinMaxScaler()
35     pca = PCA(n_components=2)
36     principalComponents = pca.fit_transform(df)
37     df['Indicator'] = pd.Series(principalComponents[:,0],index=df.index)
38     normComponents = NMscaler.fit_transform(df)
39     df_norm = df
40     df_norm.loc[:, :] = normComponents
41     indicator = df_norm.loc[:, 'Indicator']*100
42     indicator.rename(columns={'Indicator':indicator_name},inplace=True)

```

```

39     return(indicator)
40
41 # calculates domain score (out of 100) using arithmetic mean
42 # used where the three indicators have low covariance
43 def domain_indicator_Mean(df, indicator_name):
44     df['Indicator'] = (df.iloc[:,0] + df.iloc[:,1] + df.iloc[:,2])/3
45     NMscaler = MinMaxScaler()
46     normComponents = NMscaler.fit_transform(df)
47     df_norm = df
48     df_norm.loc[:,:] = normComponents
49     indicator = df_norm.loc[:, 'Indicator']*100
50     indicator.rename(columns={'Indicator':indicator_name},inplace=True)
51     return(indicator)
52
53 # maps points to London ward polygons - projects lat long point co-ords
54 # to OS map grid co-ords
55 def mapLatLongToWard(df):
56     # takes latitude and longitude of a place
57     # converts co-ordinate reference system to match shapefile
58     # maps point to ward polygon
59     inProj = Proj(init='epsg:4326') # lat long
60     outProj = Proj(init='epsg:27700') # OS National Grid
61     df['points'] = df.apply(lambda row: Point(transform(inProj, outProj,
62                                                 row['long'],row['lat'])), axis=1)
63     del (df['lat'], df['long'])
64     df = gpd.GeoDataFrame(df, geometry='points')
65     shapes = 'datastore/Shapes/London_Ward.shp'
66     wards = gpd.GeoDataFrame.from_file(shapes)
67     wards = wards[['GSS_CODE','geometry']]
68     df = gpd.tools.sjoin(df, wards, how='left')
69     df.dropna(subset=['index_right'], inplace=True)
70     return(df)
71
72 # maps pre-2014 wards to best-fit post-2014 wards
73 # affects pre-2014 wards in Hackney, Kensington & Chelsea and Tower
74 # Hamlets
75 def old_to_new_wards(df,column_name):
76     new_wards = ['E05009367','E05009368','E05009369','E05009370',
77                         'E05009371','E05009372','E05009373',
78                         'E05009374','E05009375','E05009376','E05009377',
79                         'E05009378','E05009379','E05009380',
80                         'E05009381','E05009382','E05009383','E05009384',
81                         'E05009385','E05009386','E05009387',
82                         'E05009388','E05009389','E05009390','E05009391',
83                         'E05009392','E05009393','E05009394',
84                         'E05009395','E05009396','E05009397','E05009398',
85                         'E05009399','E05009400','E05009401',
86                         'E05009402','E05009403','E05009404','E05009405',
87                         'E05009317','E05009318','E05009319',
88                         'E05009320','E05009321','E05009322','E05009323',
89                         'E05009324','E05009325','E05009326',
90                         'E05009327','E05009328','E05009332','E05009333',
91                         'E05009329','E05009330','E05009331',
92                         'E05009334','E05009335','E05009336']
93     old_wards = ['E05000231','E05000232','E05000234','E05000235',
94                         'E05000236','E05000237','E05000238',
95                         'E05000249','E05000239','E05000233','E05000239',
96                         'E05000239','E05000233','E05000239'],
97

```

```

93     'E05000236','E05000241','E05000242',
94         'E05000245','E05000235','E05000246','E05000243',
95     'E05000247','E05000248','E05000244',
96         'E05000382','E05000383','E05000384','E05000387',
97     'E05000385','E05000386','E05000389',
98         'E05000388','E05000389','E05000391','E05000392',
99     'E05000393','E05000394','E05000395',
100         'E05000396','E05000397','E05000398','E05000399',
101     'E05000581','E05000575','E05000576',
102         'E05000577','E05000578','E05000578','E05000583',
103     'E05000583','E05000579','E05000580',
104         'E05000582','E05000575','E05000586','E05000587',
105     'E05000584','E05000585','E05000573',
106         'E05000584','E05000588','E05000589']
107 frames = []
108 df_newvalues = pd.DataFrame()
109 for i in range(0,len(new_wards)):
110     new_frame = pd.DataFrame(df.loc[old_wards[i],:])
111     new_frame.rename(columns={old_wards[i]:column_name},inplace=True)
112     new_frame['ward'] = new_wards[i]
113     new_frame.set_index('ward', inplace=True)
114     frames.append(new_frame)
115 df_newvalues = pd.concat(frames, sort=False)
116 final_frames = [df, df_newvalues]
117 df_newwards = pd.concat(final_frames, sort=False)
118 return(df_newwards)
119
120 # function to transform ordnance survey polygon geometry to lat long
121 # co-ords
122 # required when passing to javaScript visualisation
123 def geometricTransform(gdf):
124     point_list = []
125     poly_list = []
126     for index, row in gdf.iterrows():
127         for pt in list(row['geometry'].exterior.coords):
128             inProj = Proj(init='epsg:27700') # lat long
129             outProj = Proj(init='epsg:4326') # OS National Grid
130             new_Point = transform(inProj, outProj, pt[0], pt[1])
131             point_list.append(new_Point)
132             poly = shapely.geometry.Polygon(point_list)
133             point_list = []
134             poly_list.append(poly)
135     gdf['geometry'] = gpd.GeoSeries(poly_list)
136     return(gdf)
137
138 # output geoJSON to javascript file defining a js variable for
139 # interactive map
140 def toGeoJSON(gdf):
141     json_file = gdf.to_json()
142     with open('dataset.js', 'w') as openfile:
143         openfile.write('var dataset = ' + json_file + ';')
144
145 # house prices
146 df_HousePrices = pd.read_csv('datastore/HousePrices.csv')
147 df_HousePrices['quintile'] = pd.qcut(df_HousePrices['Year ending Dec
2017'],5,labels=False)

```

```

146 df_HousePrices['thousands'] = df_HousePrices['Year ending Dec
147     2017']//1000
148 house_price_columns = ['New code', 'Ward name', 'Borough name', 'Year
149     ending Dec 2017', 'quintile', 'thousands']
150 df_HousePrices = pd.DataFrame(df_HousePrices, columns =
151     house_price_columns)
152
153 # read shape file for ward polygons
154 shapes = 'datastore/Shapes/London_Ward.shp'
155 map_df = gpd.read_file(shapes)
156
157 # Domain One - Education and Employment
158 # employment
159 df_employment = pd.read_csv('datastore/Employment.csv')
160 df_employment.set_index('Ward_Code', inplace=True)
161 df_employment = df_employment.drop(['K04000001'])
162 df_employment = df_employment.drop(['E92000001'])
163 df_employment = df_employment.drop(['E12000007'])
164 df_employment = standardise(df_employment)
165 df_employment = old_to_new_wards(df_employment, 'Employ_Rate')
166 # average GCSE scores
167 df_GCSE = pd.read_csv('datastore/GCSE.csv')
168 df_GCSE.set_index('Ward_Code', inplace=True)
169 df_GCSE = df_GCSE.drop(['K04000001'])
170 df_GCSE = df_GCSE.drop(['E92000001'])
171 df_GCSE = df_GCSE.drop(['E12000007'])
172 df_GCSE = standardise(df_GCSE)
173 df_GCSE = old_to_new_wards(df_GCSE, 'Av_GCSE_Points')
174 # average Ofsted rating for schools
175 df_schoolslondonopen = pd.read_csv('datastore/Schools.csv')
176 df_schoolsborough =
177 df_schoolslondonopen.pivot_table(values='Ofsted_Rating',
178     index='DistrictAdministrative (code)', fill_value=0)
179 df_schools =
180 df_schoolslondonopen.pivot_table(values='Ofsted_Rating',
181     index='AdministrativeWard (code)', fill_value=0)
182 df_schoolward = pd.merge(map_df, df_schools, left_on='GSS_CODE',
183     right_index=True, how='outer')
184 df_schools =
185 pd.merge(df_schoolward, df_schoolsborough, left_on='LB_GSS_CD',
186     right_index=True, how='outer')
187 df_schools['Ofsted_Rating_x'].fillna(df_schools['Ofsted_Rating_y'],
188     inplace=True)
189 final_columns = ['GSS_CODE', 'Ofsted_Rating_x']
190 df_schools = pd.DataFrame(df_schools, columns = final_columns)
191 df_schools.set_index('GSS_CODE', inplace=True)
192 df_schools = standardise(df_schools)
193 # merge the 3 indicators to create the domain
194 df_edu = pd.merge(df_GCSE, df_schools, left_index=True, right_index=True)
195 df_eduemploy = pd.merge(df_employment, df_edu, left_index=True,
196     right_index=True)
197 # check covariance and correlation matrices
198 df_eduemploy_cov = df_eduemploy.cov()
199 df_eduemploy_corr = df_eduemploy.corr()
200 df_eduemploy_corr.to_csv('datastore/eduemploycorr.csv')
201 ## run principal component analysis to create domain score

```

```
193 Education_Employment =
194     domain_indicator_Mean(df_eduemploy,'Education_Employment')
195
196 # Domain Two - Safety and Security
197 # crime (against the person and locality crime)
198 df_crime = pd.read_csv('datastore/Crime.csv')
199 df_crime_pivot = df_crime.pivot_table(index='WardCode', columns='Major
200     Category', values='2017',aggfunc='sum')
201 person_columns = ['Robbery','Sexual Offences','Violence Against the
202     Person']
203 area_columns = ['Burglary','Criminal Damage','Drugs','Theft and
204     Handling']
205 df_crimeperson = pd.DataFrame(df_crime_pivot, columns = person_columns)
206 df_crimeperson = standardise(df_crimeperson)
207 df_crimearea = pd.DataFrame(df_crime_pivot, columns = area_columns)
208 df_crimearea = standardise(df_crimearea)
209 # traffic danger
210 df_traffic = pd.read_csv('datastore/Traffic.csv')
211 df_traffic.set_index('Ward_Code',inplace=True)
212 df_traffic = standardise(df_traffic)
213 df_traffic['traffic_fatality_score'] = (df_traffic['Fatal'] +
214     df_traffic['Serious'] + df_traffic['Slight'])/3
215 del df_traffic['Fatal']
216 del df_traffic['Serious']
217 del df_traffic['Slight']
218 # merge the 3 indicators to create the domain
219 df_crime = pd.merge(df_crimeperson, df_crimearea, left_index=True,
220     right_index=True)
221 df_safetysecurity = pd.merge(df_crime,df_traffic, left_index=True,
222     right_index=True)
223 # check covariance and correlation matrices
224 df_safetysecurity_cov = df_safetysecurity.cov()
225 df_safetysecurity_corr = df_safetysecurity.corr()
226 df_safetysecurity_corr.to_csv('datastore/safetysecuritycorr.csv')
227 # run principal component analysis to create domain score
228 Safety_Security =
229     100-domain_indicator_PCA(df_safetysecurity,'Safety_Security')
230
231 # Domain Three - Environment
232 # emissions
233 df_emissions = pd.read_csv('datastore/Emissions.csv')
234 df_emissions.set_index('Ward_Code',inplace=True)
235 df_emissions = df_emissions.drop(['K04000001'])
236 df_emissions = df_emissions.drop(['E92000001'])
237 df_emissions = df_emissions.drop(['E12000007'])
238 df_emissions = standardise(df_emissions)
239 df_emissions['emission'] = (df_emissions['PM10'] +
240     df_emissions['NO2'])/2
241 df_emissions = pd.DataFrame(df_emissions['emission']*-1) # emissions
242     negative so *-1
243 df_emissions = old_to_new_wards(df_emissions,'emission')
244 # greenspace
245 df_greenspace = pd.read_csv('datastore/Greenspace.csv')
246 df_greenspace.set_index('Ward_Code',inplace=True)
247 df_greenspace = df_greenspace.drop(['K04000001'])
248 df_greenspace = df_greenspace.drop(['E92000001'])
249 df_greenspace = df_greenspace.drop(['E12000007'])
```

```

240 df_greenspace = standardise(df_greenspace)
241 df_greenspace = old_to_new_wards(df_greenspace,'Greenspace%')
242 # nature
243 df_nature = pd.read_csv('datastore/Nature.csv')
244 df_nature.set_index('Ward_Code',inplace=True)
245 df_nature = standardise(df_nature)
246 df_nature = old_to_new_wards(df_nature,'Nature_Access')
247 # merge the 3 indicators to create the domain
248 df_green = pd.merge(df_nature,df_greenspace, left_index=True,
249     right_index=True)
250 df_environment = pd.merge(df_green,df_emissions, left_index=True,
251     right_index=True)
252 # check covariance and correlation matrices
253 df_environment_cov = df_environment.cov()
254 df_environment_corr = df_environment.corr()
255 df_environment_corr.to_csv('datastore/environmentcorr.csv')
256 # create domain indicator using arithmetic mean
257 Environment = domain_indicator_Mean(df_environment,'Environment')

258 # Domain Four - Community Vitality and Participation
259 # election turnout
260 df_turnout = pd.read_csv('datastore/ElectionTurnout.csv')
261 df_turnout.set_index('Ward_Code', inplace=True)
262 df_turnout = standardise(df_turnout)
263 # access to cultural spaces
264 cultural_venues = pd.read_csv('datastore/CulturalVenues.csv')
265 cultural_locations = mapLatLongToWard(cultural_venues)
266 df_culturepivot =
267     cultural_locations.pivot_table(index='GSS_CODE',
268         values='name',aggfunc='count')
268 df_cultureaccess = standardise(df_culturepivot)
269 # access to food and drink establishment
270 bars = pd.read_csv('datastore/Bars.csv')
271 restaurants = pd.read_csv('datastore/Restaurants.csv')
272 food_drink_list = [bars, restaurants]
273 bars_restaurants = pd.concat(food_drink_list)
274 # specify co-ordinate reference systems
275 bar_restaurant_locations = mapLatLongToWard(bars_restaurants)
276 df_barrestaurantpivot =
277     bar_restaurant_locations.pivot_table(index='GSS_CODE',
278         values='name',aggfunc='count')
278 df_barrestaurant = standardise(df_barrestaurantpivot)
279 df_barrestaurant.rename(columns={'name':'name_y'},inplace=True)
280 # merge the 3 indicators to create the domain
281 df_participation = df_turnout.join(df_cultureaccess, how='outer')
282 df_vitalityparticipation = df_participation.join(df_barrestaurant,
283     how='outer')
284 df_vitalityparticipation['name'].fillna(0.0, inplace=True)
285 df_vitalityparticipation['name_y'].fillna(0.0, inplace=True)
286 df_vitalityparticipation['Turnout'].fillna(0.0, inplace=True)
287 # check covariance and correlation matrices
288 df_vitalityparticipation_cov = df_vitalityparticipation.cov()
289 df_vitalityparticipation_corr = df_vitalityparticipation.corr()
290 df_vitalityparticipation_corr.to_csv('datastore/vitalityparticipationcorr.csv')
291 # create domain indicator using arithmetic mean
292 Vitality_Participation =
293     domain_indicator_Mean(df_vitalityparticipation,'Vitality_Participation')

```

```
291 # Domain Five - Infrastructure
292 # access to public transportation
293 df_pta = pd.read_csv('datastore/TransportAccess.csv')
294 df_pta.set_index('Ward Code', inplace=True)
295 df_pta = standardise(df_pta)
296 # average journey times
297 df_journeys = pd.read_csv('datastore/JourneyTimes.csv')
298 df_journeypivot = df_journeys.pivot_table(index='GSS_CODE',
299                                         values='JourneyTime')
300 df_journeys = standardise(df_journeypivot)*-1
301 # population density
302 df_density = pd.read_csv('datastore/PopDensity.csv')
303 df_density.set_index('Code', inplace=True)
304 df_density = standardise(df_density)*-1
305 df_density =
306     old_to_new_wards(df_density, 'Population_per_square_kilometre')
307 # merge the 3 indicators to create the domain
308 df_transportation = pd.merge(df_pta, df_journeys, left_index=True,
309                               right_index=True)
310 df_infrastructure = pd.merge(df_transportation, df_density,
311                               left_index=True, right_index=True)
312 # check covariance and correlation matrices
313 df_infrastructure_cov = df_infrastructure.cov()
314 df_infrastructure_corr = df_infrastructure.corr()
315 df_infrastructure_corr.to_csv('datastore/infrastructurecorr.csv')
316 # create domain indicator using PCA
317 Infrastructure =
318     domain_indicator_PCA(df_infrastructure, 'Infrastructure')

319 # Domain Six - Health
320 # life expectancy
321 df_expectancy = pd.read_csv('datastore/LifeExpectancy.csv')
322 df_expectancy.set_index('Ward_Code', inplace=True)
323 df_expectancy = standardise(df_expectancy)
324 df_expectancy = old_to_new_wards(df_expectancy, 'Life_Ex')
325 # childhood obesity
326 df_obesity = pd.read_csv('datastore/ChildObesity.csv')
327 df_obesity.set_index('Code_x', inplace=True)
328 df_obesity = standardise(df_obesity)*-1
329 # illness affecting work
330 df_illness = pd.read_csv('datastore/Illness.csv')
331 df_illnesspivot
332 = df_illness.pivot_table(index='GSS_CODE', values='Comparative illness
333     and disability ratio indicator')
334 df_illness = standardise(df_illnesspivot)*-1
335 # merge the 3 indicators to create the domain
336 df_prehealth = pd.merge(df_illness, df_obesity, left_index=True,
337                         right_index=True)
338 df_health = pd.merge(df_prehealth, df_expectancy, left_index=True,
339                         right_index=True)
340 # check covariance and correlation matrices
341 df_health_cov = df_health.cov()
342 df_health_corr = df_health.corr()
343 df_health_corr.to_csv('datastore/healthcorr.csv')
344 # build domain indicator using PCA
345 Health = domain_indicator_PCA(df_health, 'Health')
```

```

340 # build data set for modelling
341 DataSet =
342 pd.merge(df_HousePrices, pd.DataFrame(Education_Employment), left_on =
343     'New code', right_index = True)
344 DataSet.rename(columns={0:'Education_Employment'},inplace=True)
345 DataSet = pd.merge(DataSet, pd.DataFrame(Safety_Security), left_on =
346     'New code', right_index = True)
347 DataSet.rename(columns={0:'Safety_Security'},inplace=True)
348 DataSet = pd.merge(DataSet, pd.DataFrame(Environment), left_on = 'New
349     code', right_index = True)
350 DataSet.rename(columns={0:'Environment'},inplace=True)
351 DataSet = pd.merge(DataSet, pd.DataFrame(Vitality_Participation),
352     left_on = 'New code', right_index = True)
353 DataSet.rename(columns={0:'Vitality_Participation'},inplace=True)
354 DataSet = pd.merge(DataSet, pd.DataFrame(Infrastructure), left_on =
355     'New code', right_index = True)
356 DataSet.rename(columns={0:'Infrastructure'},inplace=True)
357 DataSet = pd.merge(DataSet, pd.DataFrame(Health), left_on = 'New code',
358     right_index = True)
359 DataSet.rename(columns={0:'Health','Year ending Dec
360     2017 ':'Median_Price'},inplace=True)
361 DataSet.to_csv('datastore/DataSet.csv', index=False)
362
363 df_predictions = pd.read_csv('datastore/predictions.csv')
364
365 df_datasetmapping =
366 gpd.GeoDataFrame(pd.merge(map_df, df_predictions, left_on='GSS_CODE',
367     right_on='New code'))
368
369 df_datasetmapping.to_csv('datastore/datasetmapping.csv')
370
371 df_js = geometricTransform(df_datasetmapping)
372
373 toGeoJSON(df_js)

```

Data Modeller

```

1 import pandas as pd
2 import numpy as np
3 from sklearn import linear_model
4 from sklearn.preprocessing import PolynomialFeatures
5 from sklearn.model_selection import KFold
6 from sklearn.metrics import mean_squared_error
7 import statsmodels.api as sm
8 import seaborn as sns
9 import powerlaw
10 from sklearn.linear_model import LogisticRegression
11 from sklearn.neural_network import MLPClassifier
12 from sklearn.model_selection import KFold
13 from sklearn.tree import DecisionTreeClassifier
14 from sklearn.neighbors import KNeighborsClassifier
15 from sklearn.ensemble import GradientBoostingClassifier
16 from sklearn.ensemble import RandomForestClassifier
17 from sklearn.naive_bayes import GaussianNB
18 import math

```

```
20 from sklearn.metrics import classification_report,
21 confusion_matrix, mean_squared_error, accuracy_score
22
23
24 np.random.seed(100)
25
26 # create datasets and specify validation model
27 dataset = pd.read_csv('files/DataSet.csv')
28 validation = KFold(n_splits=7, shuffle=True, random_state=None)
29
30 columns = ['Education_Employment', 'Safety_Security', 'Environment',
31             'Vitality_Participation',
32             'Infrastructure', 'Health', 'thousands']
33 DataVars = pd.DataFrame(dataset, columns=columns)
34 y = np.ravel(pd.DataFrame(dataset['thousands']))
35 X = DataVars
36 del X['thousands']
37
38 pl_fit = powerlaw.Fit(y)
39
40 pl_alpha = pl_fit.power_law.alpha
41 pl_sigma = pl_fit.power_law.sigma
42
43 # use statsmodel for model summary
44 X_sm = sm.add_constant(X)
45 stats_model = sm.OLS(y, X_sm)
46 stats_model_estimate = stats_model.fit()
47 print(stats_model_estimate.summary())
48
49
50 # linear regression
51 linarr_score = 0
52 linarr_error = 0
53
54 for train_index, test_index in validation.split(X):
55     X_train, X_test = np.array(X)[train_index], np.array(X)[test_index]
56     y_train, y_test = np.array(y)[train_index], np.array(y)[test_index]
57     lm = linear_model.LinearRegression()
58     model = lm.fit(X_train, y_train)
59     predictions = lm.predict(X_test)
60     linarr_score = linarr_score + model.score(X_test, y_test)
61     linarr_error = linarr_error + mean_squared_error(y_test,
62                                                       predictions)
63 linarr_score = linarr_score/7
64 linarr_error = linarr_error/7
65
66 lr_list = pd.DataFrame([['Linear Regression', linarr_score,
67                         linarr_error]])
68
69 linarr_score = 0
70 linarr_error = 0
71
72 poly2 = PolynomialFeatures(degree=2)
73 X2 = poly2.fit_transform(X)
74 for train_index, test_index in validation.split(X2):
```

```

73     X2_train, X2_test = np.array(X2)[train_index],
74             np.array(X2)[test_index]
75     y_train, y_test = np.array(y)[train_index], np.array(y)[test_index]
76     clm = linear_model.LinearRegression()
77     model = clm.fit(X2_train, y_train)
78     predictions = clm.predict(X2_test)
79     linearr_score = linearr_score + model.score(X2_test, y_test)
80     linearr_error = linearr_error + mean_squared_error(y_test,
81             predictions)
82     linearr_score = linearr_score/7
83     linearr_error = linearr_error/7
84
85 # reduce features
86 del X['Vitality_Participation']
87 linearr_score = 0
88 linearr_error = 0
89
90 for train_index, test_index in validation.split(X):
91     X_train, X_test = np.array(X)[train_index], np.array(X)[test_index]
92     y_train, y_test = np.array(y)[train_index], np.array(y)[test_index]
93     lm = linear_model.LinearRegression()
94     model = lm.fit(X_train, y_train)
95     predictions = lm.predict(X_test)
96     linearr_score = linearr_score + model.score(X_test, y_test)
97     linearr_error = linearr_error + mean_squared_error(y_test,
98             predictions)
99     linearr_score = linearr_score/7
100    linearr_error = linearr_error/7
101
102 lr_list_poly2 = pd.DataFrame([['Linear Regression (poly 2)',

103                               linearr_score, linearr_error]])
104
105 del X['Education_Employment']
106 linearr_score = 0
107 linearr_error = 0
108
109 for train_index, test_index in validation.split(X):
110     X_train, X_test = np.array(X)[train_index], np.array(X)[test_index]
111     y_train, y_test = np.array(y)[train_index], np.array(y)[test_index]
112     lm = linear_model.LinearRegression()
113     model = lm.fit(X_train, y_train)
114     predictions = lm.predict(X_test)
115     linearr_score = linearr_score + model.score(X_test, y_test)
116     linearr_error = linearr_error + mean_squared_error(y_test,
117             predictions)
118     linearr_score = linearr_score/7
119     linearr_error = linearr_error/7
120
121 lr_list_4dim = pd.DataFrame([['Linear Regression (reduced to 4 dim)',

122                               linearr_score, linearr_error]])
123
124 # try a log transformation of the data
125 dataset = pd.read_csv('files/DataSet.csv')
126 validation = KFold(n_splits=7, shuffle=True, random_state=None)

```

```
123
124 columns = ['Education_Employment', 'Safety_Security', 'Environment',
125     'Vitality_Participation',
126     'Infrastructure', 'Health', 'Median_Price']
127 DataVars = pd.DataFrame(dataset, columns=columns)
128 y = np.log(np.ravel(pd.DataFrame(dataset['Median_Price'])))
129 X = DataVars
130 #X.apply(np.log)
131 #X.replace(np.nan,0)
132 del X['Median_Price']
133
134 for train_index, test_index in validation.split(X):
135     X_train, X_test = np.array(X)[train_index], np.array(X)[test_index]
136     y_train, y_test = np.array(y)[train_index], np.array(y)[test_index]
137     lm = linear_model.LinearRegression()
138     model = lm.fit(X_train, y_train)
139     predictions = lm.predict(X_test)
140     linearr_score = linearr_score + model.score(X_test, y_test)
141     linearr_error = linearr_error + mean_squared_error(y_test,
142         predictions)
143 linearr_score = linearr_score/7
144 linearr_error = linearr_error/7
145
146 lr_list_logtrans = pd.DataFrame([['Linear Regression (Log Transform)', 
147     linearr_score, linearr_error]])
148
149 model_results = pd.concat([lr_list, lr_list_poly2, lr_list_5dim,
150     lr_list_4dim, lr_list_logtrans])
151 model_results.rename(columns={0:'Model Type',1:'Av. R sq',2:'Av.
152     MSE'},inplace=True)
153
154 print(model.coef_, model.intercept_)
155
156 new_data = pd.DataFrame(dataset)
157 new_data['predicted'] = np.exp(pd.Series(lm.predict(X)))
158
159 model_results.to_csv('files/regression.csv', index=False)
160
161 np.random.seed(100)
162 # create datasets and specify validation model
163 dataset = pd.read_csv('files/DataSet.csv')
164 validation = KFold(n_splits=7, shuffle=True, random_state=None)
165
166 columns = ['Education_Employment', 'Safety_Security', 'Environment',
167     'Vitality_Participation',
168     'Infrastructure', 'Health', 'quintile']
169
170 DataVars = pd.DataFrame(dataset, columns=columns)
171 y = np.ravel(pd.DataFrame(dataset['quintile']))
172 X = DataVars
173 del X['quintile']
174
175 classification_models = [LogisticRegression(),
176     DecisionTreeClassifier(), KNeighborsClassifier(),
177     RandomForestClassifier(n_estimators=100),
178     GradientBoostingClassifier(), GaussianNB()]
```

```

171 classification_model_labels = ['Logistic Regression', 'Decision Tree',
172     'K-Nearest Neighbours',
173     'Random Forest', 'Gradient Boosting', 'Naive
174     Bayes']
175
176 result_list = []
177 # classification models
178 for i in range(0,len(classification_models)):
179     np.random.seed(100)
180     classification_score = 0
181     training_score = 0
182     prediction_score = 0
183     for train_index, test_index in validation.split(X):
184         X_train, X_test = np.array(X)[train_index],
185             np.array(X)[test_index]
186         y_train, y_test = np.array(y)[train_index],
187             np.array(y)[test_index]
188         classifier = classification_models[i]
189         model = classifier.fit(X_train, y_train)
190         predictions = model.predict(X_test)
191         for j in range(0,len(y_test)):
192             prediction_score = prediction_score +
193             ((model.predict(X_test)[j]-y_test[j])*(model.predict(X_test)[j]-y_test[j]))
194             training_score = training_score + accuracy_score(y_train,
195                 model.predict(X_train))
196             classification_score = classification_score +
197                 accuracy_score(y_test, predictions)
198             result = [classification_model_labels[i], training_score/7,
199                 classification_score/7, math.sqrt(prediction_score/7)]
200             result_list.append(result)
201 result_df = pd.DataFrame(result_list)
202 result_df.rename(columns={0:'Model Type',1:'Training Score',2:'Test
203     Score',3:'Prediction Error'}, inplace=True)
204 result_df.to_csv('files/classificationResults.csv', index=False)
205
206 # tune Neural Network for hidden layer
207 np.random.seed(100)
208 neural_size_results = []
209 for i in range(1,21):
210     classification_score = 0
211     for train_index, test_index in validation.split(X):
212         X_train, X_test = np.array(X)[train_index],
213             np.array(X)[test_index]
214         y_train, y_test = np.array(y)[train_index],
215             np.array(y)[test_index]
216         nn =
217             MLPClassifier(hidden_layer_sizes=(i*5),activation='logistic',max_iter=1000)
218         network = nn.fit(X_train, y_train)
219         predictions = nn.predict(X_test)
220         classification_score = classification_score +
221             accuracy_score(y_test, predictions)
222         neural_size = [i*5,classification_score/7]
223         neural_size_results.append(neural_size)
224 neuralsize_df = pd.DataFrame(neural_size_results)
225 neuralsize_df.rename(columns={0:'Nodes in hidden layer',1:'Test
226     Score'}, inplace=True)
227 neuralsize_df.to_csv('files/neuralsizeResults.csv', index=False)

```

```

215
216 # tune Neural Network for best activation function
217 np.random.seed(100)
218 activation_functions = ['identity', 'logistic', 'tanh', 'relu']
219 neural_act_results = []
220 for j in range(0,len(activation_functions)):
221     classification_score = 0
222     for train_index, test_index in validation.split(X):
223         X_train, X_test = np.array(X)[train_index],
224             np.array(X)[test_index]
225         y_train, y_test = np.array(y)[train_index],
226             np.array(y)[test_index]
227         nn =
228             MLPClassifier(hidden_layer_sizes=(45),activation=activation_functions[j],max_iter=1000)
229         network = nn.fit(X_train, y_train)
230         predictions = nn.predict(X_test)
231         classification_score = classification_score +
232             accuracy_score(y_test, predictions)
233         neural_act = [activation_functions[j],classification_score/7]
234         neural_act_results.append(neural_act)
235     neuralact_df = pd.DataFrame(neural_act_results)
236     neuralact_df.rename(columns={0:'Activation Function',1:'Test Score'},
237         inplace=True)
238     neuralact_df.to_csv('files/neuralactivationResults.csv', index=False)
239
240 chosen_classifier = LogisticRegression()
241 chosen_model = chosen_classifier.fit(X, y)
242 chosen_predictions = chosen_model.predict(X)
243 new_data['Logistic'] = pd.Series(chosen_predictions)
244
245 neural_classifier =
246     MLPClassifier(hidden_layer_sizes=(45),activation='logistic',max_iter=1000)
247 neural_model = neural_classifier.fit(X, y)
248 nn_predictions = neural_model.predict(X)
249 new_data['NeuralNetwork'] = pd.Series(nn_predictions)
250
251 new_data.to_csv('files/predictions.csv')

```

Data Importer

```

1
2 import pandas as pd
3 import numpy as np
4 import time
5 import requests
6 import geopandas as gpd
7
8 def importHousePrices():
9     XLS_HOUSE_PRICE =
10        'https://data.london.gov.uk/download/average-house-prices/fb8116f5-06f8-42e0-aa6c-b0b1bd69c'
11        /land-registry-house-prices-ward.xls'
12    try:
13        df_HousePricesAll =
14            pd.read_excel(XLS_HOUSE_PRICE,'Median',skiprows=[1])
15    except:
16        print('Cannot open data file - London House Prices')
17        df_HousePrices = pd.DataFrame(df_HousePricesAll.iloc[1:,:])

```

```

17     df_HousePrices.to_csv('datastore/HousePrices.csv',index=False)
18     return(df_HousePrices)
19
20 def importEmploymentRates():
21     XLSX_WARD_ATLAS =
22     'https://files.datapress.com/london/dataset/ward-profiles-and-atlas/2015-09-24T14:48:35'
23     try:
24         df_ward_atlas2 = pd.read_excel(XLSX_WARD_ATLAS, 'iadatasheet2',
25                                         skiprows=2)
26     except:
27         print('Cannot open data file - Ward Atlas Sheet 2')
28     employment_columns = ['New Code','Economically active: % In
29                           employment.1']
30     df_employment = pd.DataFrame(df_ward_atlas2, columns =
31                                   employment_columns)
32     df_employment.rename(columns=
33                           {'New Code':'Ward_Code','Economically active: % In
34                           employment.1':'Employ_Rate'},inplace=True)
35     df_employment.to_csv('datastore/Employment.csv',index=False)
36     return(df_employment)
37
38 def importGCSE():
39     XLSX_WARD_ATLAS =
40     'https://files.datapress.com/london/dataset/ward-profiles-and-atlas/2015-09-24T14:48:35'
41     try:
42         df_ward_atlas3 = pd.read_excel(XLSX_WARD_ATLAS, 'iadatasheet3',
43                                         skiprows=2)
44     except:
45         print('Cannot open data file - Ward Atlas Sheet 3')
46     GCSE_columns = ['New Code','2014.2']
47     df_GCSE = pd.DataFrame(df_ward_atlas3, columns = GCSE_columns)
48     df_GCSE.rename(columns={'New
49                           Code':'Ward_Code','2014.2':'Av_GCSE_Points'},inplace=True)
50     df_GCSE.to_csv('datastore/GCSE.csv',index=False)
51     return(df_GCSE)
52
53 def importSchools():
54     # from https://get-information-schools.service.gov.uk/Downloads
55     CSV_SCHOOLS =
56         'http://ea-edubase-api-prod.azurewebsites.net/edubase/edubasealldata20180908.csv'
57     try:
58         iter_csv = pd.read_csv(CSV_SCHOOLS, iterator=True,
59                               chunksize=1000)
60         df_schoolsall = pd.DataFrame()
61         df_schoolsall = pd.concat([chunk[chunk['GOR (name)'] ==
62                                     'London'] for chunk in iter_csv])
63     except:
64         print('Cannot open data file - SchoolsData')
65     df_schoolslondonopen =
66         df_schoolsall[(df_schoolsall['EstablishmentStatus (code)'] == 1)]
67     df_schoolslondonopen['OfstedRating (name)'].replace('', np.nan,
68                               inplace=True)
69     df_schoolslondonopen.dropna(subset=['OfstedRating (name)'],
70                                 inplace=True)
71     df_schoolslondon = pd.DataFrame(df_schoolslondonopen)
72     df_schoolslondon.rename(columns={'OfstedRating
73                                     (name)':'Ofsted_Rating'}, inplace=True)

```

```
61     ofsted_ratings =
62     {'Ofsted_Rating': {'Outstanding': 2, 'Good': 1, 'Requires improvement': -1,
63     'Serious Weaknesses': -2, 'Special Measures': -2, 'Inadequate': -2}}
64     df_schoolslondon.replace(ofsted_ratings, inplace=True)
65     df_schoolslondon.to_csv('datastore/Schools.csv', index=False)
66     return(df_schoolslondon)
67
68 def importCrime():
69     CSV_CRIME =
70     'https://data.london.gov.uk/download/recorded_crime_summary/866c05de-c5cd-454b-8fe5-9e7c77ea
71     MPS%20Ward%20Level%20Crime%20%28most%20recent%202024%20months%29.csv'
72     try:
73         df_crime = pd.read_csv(CSV_CRIME)
74     except:
75         print('Cannot open data file - Crime by Ward')
76     df_crime['2017'] =
77         df_crime['201701']+df_crime['201702']+df_crime['201703']+df_crime['201704']+
78     df_crime['201705']+df_crime['201706']+df_crime['201707']+df_crime['201708']+df_crime['201709']+
79     +df_crime['201710']+df_crime['201711']+df_crime['201712']
80     df_crime.to_csv('datastore/Crime.csv', index=False)
81     return(df_crime)
82
83 def importTraffic():
84     XLSX_TRAFFIC_FATALITIES =
85     'https://files.datapress.com/london/dataset/
86     road-casualties-severity-borough/road-casualties-severity-lsoa-msoa-ward.xls'
87     try:
88         df_traffic_f = pd.read_excel(XLSX_TRAFFIC_FATALITIES, 'Ward 2014
89             only', skiprows=1)
90         df_traffic_f2 = pd.read_excel(XLSX_TRAFFIC_FATALITIES,
91             'Ward(pre2014)', skiprows=1)
92     except:
93         print('Cannot open data file - Traffic Fatalities by Ward')
94     traffic_columns = ['Unnamed: 0','1 Fatal.9','2 Serious.9','3
95         Slight.9']
96     traffic_columns2 = ['Ward Code','Fatal.4','Serious.4','Slight.4']
97     df_traffic_fatalities = pd.DataFrame(df_traffic_f, columns =
98         traffic_columns)
99     df_traffic_fatalities2 = pd.DataFrame(df_traffic_f2, columns =
100         traffic_columns2)
101     df_traffic_fatalities.rename(columns=
102         {'Unnamed: 0':'Ward_Code','1 Fatal.9':'Fatal','2
103         Serious.9':'Serious','3 Slight.9':'Slight'},
104         inplace=True)
105     df_traffic_fatalities2.rename(columns={'Ward
106         Code':'Ward_Code','Fatal.4':'Fatal','Serious.4':'Serious','Slight.4':'Slight'},
107         inplace=True)
108     df_traffic_fatalities = pd.concat([df_traffic_fatalities,
109         df_traffic_fatalities2])
110     df_traffic_fatalities.to_csv('datastore\Traffic.csv', index=False)
111
112 def importEmissions():
113     XLSX_WARD_ATLAS =
114     'https://files.datapress.com/london/dataset/ward-profiles-and-atlas/2015-09-24T14:48:35/ward
115     profiles-and-atlas.xls'
116     try:
117         df_ward_atlas5 = pd.read_excel(XLSX_WARD_ATLAS, 'iadatasheet5',
118             skiprows=2)
```

```
108 except:
109     print('Cannot open data file - Ward Atlas Sheet 5')
110 emissions_columns = ['New Code','2011.4','2011.6']
111 df_emissions = pd.DataFrame(df_ward_atlas5, columns =
112     emissions_columns)
113 df_emissions.rename(columns={'New
114     Code':'Ward_Code','2011.4':'PM10','2011.6':'NO2'},inplace=True)
115 df_emissions.to_csv('datastore/Emissions.csv', index=False)
116 return(df_emissions)
117
118 def importGreenspace():
119     XLSX_WARD_ATLAS =
120     'https://files.datapress.com/london/dataset/ward-profiles-and-atlas/2015-09-24T14:48:35Z'
121     try:
122         df_ward_atlas5 = pd.read_excel(XLSX_WARD_ATLAS, 'iadatasheets5',
123             skiprows=2)
124     except:
125         print('Cannot open data file - Ward Atlas Sheet 5')
126     greenspace_columns = ['New Code','2014.4']
127     df_greenspace = pd.DataFrame(df_ward_atlas5, columns =
128         greenspace_columns)
129     df_greenspace.rename(columns={'New
130     Code':'Ward_Code','2014.4':'Greenspace%'},inplace=True)
131     df_greenspace.to_csv('datastore/Greenspace.csv', index=False)
132
133 def importNature():
134     CSV_NATURE_ACCESS =
135     'https://files.datapress.com/london/dataset/access-public-open-space-and-nature-ward/
136     public-open-space-nature-ward_access-to-nature.csv'
137     try:
138         df_nature_access = pd.read_csv(CSV_NATURE_ACCESS)
139     except:
140         print('Cannot open data file - Nature Access by Ward')
141     nature_columns = ['Ward','% homes with good access to nature']
142     df_nature = pd.DataFrame(df_nature_access, columns = nature_columns)
143     df_nature.rename(columns=
144     {'Ward':'Ward_Code', '% homes with good access to
145         nature':'Nature_Access'},inplace=True)
146     df_nature.to_csv('datastore/Nature.csv', index=False)
147
148 def importElection():
149     XLSX_ELECTION =
150     'https://files.datapress.com/london/dataset/london-elections-results-2016-wards-boroughs/
151     /2016-05-27T10:46:12/gla-elections-votes-all-2016.xlsx'
152     try:
153         df_election = pd.read_excel(XLSX_ELECTION, 'Wards & Postals',
154             skiprows=2)
155     except:
156         print('Cannot open data file - Election Turnout by Ward')
157     df_turnout = df_election[(df_election['Unnamed: 2'] == 'Ward')]
158     turnout_columns = ['Unnamed: 4','% Turnout']
159     df_turnout = pd.DataFrame(df_turnout, columns = turnout_columns)
160     df_turnout.rename(columns={'Unnamed: 4':'Ward_Code', '%
161         Turnout':'Turnout'},inplace=True)
162     df_turnout.to_csv('datastore/ElectionTurnout.csv', index=False)
163     return(df_turnout)
```

```

157 def importVenue(category, filename):
158     df_Foursquare = pd.DataFrame()
159     FrameList = []
160     limit_reached = 0
161     for i in range(1,50):
162         for j in range(15,750,15):
163             client_id = "XO1QQIQ02JE0EKQXZ4HBOYPHW2DSWJH5EIBN2AOZG2NRZVPK"
164             client_secret =
165                 "GJGSIAZR4WXLYOHW4VP1JHTAMLH23EZXCWZDVPCZTYY2RQ4V"
166             lat = (51.2 + (i/100.0))
167             long = (-0.25 + (j/1000.0))
168             if category == 'cultural':
169                 category_id = '4d4b7104d754a06370d81259' # cultural space
170             elif category == 'restaurant':
171                 category_id = '4d4b7105d754a06374d81259' # food
172             elif category == 'bar':
173                 category_id = '4d4b7105d754a06376d81259' # drink
174             distance = 450
175             requested_keys = ["categories", "id", "location", "name"]
176             url = "https://api.foursquare.com/v2/venues/search?ll=%s,%s&intent=browse&radius=%s&categoryId=%s&limit=49&client_id=%s&client_secret=%s&v=%s" % (lat, long, distance, category_id, client_id, client_secret,
177                 time.strftime("%Y%m%d"))
178             resp = requests.get(url)
179             dataResp = resp.json()
180             if dataResp["response"]["venues"] != []:
181                 data =
182                     pd.DataFrame(dataResp["response"]["venues"])
183                     df_FoursquareIteration = pd.DataFrame(data)
184                     if len(df_FoursquareIteration) == 49:
185                         limit_reached = limit_reached + 1
186                     df_FoursquareIteration["categories"] =
187                     df_FoursquareIteration["categories"].apply(lambda x:
188                         dict(x[0])['name'])
189                     df_FoursquareIteration["lat"] =
190                     df_FoursquareIteration["location"].apply(lambda x: dict(x)[ "lat"])
191                     df_FoursquareIteration["long"] =
192                     df_FoursquareIteration["location"].apply(lambda x: dict(x)[ "lng"])
193                     FrameList.append(df_FoursquareIteration)
194                     df_Foursquare = pd.concat(FrameList)
195                     df_Foursquare.drop_duplicates(subset=['id'], keep=False)
196                     if limit_reached > 0:
197                         print('Limit Reached')
198                         print(limit_reached)
199                         columns = ['name', 'categories', 'lat', 'long']
200                         df_culture = pd.DataFrame(df_Foursquare, columns = columns)
201                         df_culture.to_csv(filename, index=False, encoding='utf-8')

202 def importTransportAccess():
203     CSV_TRANSPORT_ACCESS =
204     'https://files.datapress.com/london/dataset/public-transport-accessibility-levels/
205     2018-02-20T14:44:30.58/Ward2014%20AvPTAI2015.csv'
206     try:
207         df_transport_access = pd.read_csv(CSV_TRANSPORT_ACCESS)
208     except:
209         print('Cannot open data file - Transport Access by Ward')
210         transport_access_columns = ['Ward Code', 'AvPTAI2015']

```

```

210     df_transport_access = pd.DataFrame(df_transport_access, columns =
211                                         transport_access_columns)
212     df_transport_access.to_csv('datastore/TransportAccess.csv',
213                               index=False)
214     return(df_transport_access)
215
216
217 def importJourneyTimes():
218     XLS_JOURNEYTIMES = 'files/jts0501.xls'
219     CSV_LSOA_MAPPING =
220     'https://opendata.arcgis.com/datasets/07a6d14d4a0540769f0662f4d1450bae_0.csv'
221     try:
222         df_journeys = pd.read_excel(XLS_JOURNEYTIMES, 'JTS0501',
223                                     skiprows=6)
224     except:
225         print('Cannot open data file - JourneyTimes')
226     try:
227         df_lsoa_ward = pd.read_csv(CSV_LSOA_MAPPING)
228     except:
229         print('Cannot open data file - LSOA to Ward Mapping')
230     journey_columns = ['LSOA_code','100EmpPTt','500EmpPTt','5000EmpPTt']
231     df_journeys = pd.DataFrame(df_journeys, columns = journey_columns)
232     df_journeys['JourneyTime'] =
233     (df_journeys['100EmpPTt'] + df_journeys['500EmpPTt'] +
234      df_journeys['5000EmpPTt'])/3.0
235     df_journeys = pd.merge(df_journeys, df_lsoa_ward,
236                           left_on='LSOA_code', right_on='LSOA11CD')
237     shapes = 'datastore/Shapes/London_Ward.shp'
238     wards = gpd.read_file(shapes)
239     df_journeys = pd.merge(wards, df_journeys, left_on='GSS_CODE',
240                           right_on='WD15CD')
241     df_journeys.to_csv('datastore/JourneyTimes.csv', index=False)
242
243
244 def importPopulationDensity():
245     CSV_POP_DENSITY=
246     'https://files.datapress.com/london/dataset/land-area-and-population-density-ward-and-bo
247 /2018-03-05T10:54:05.31/housing-density-ward.csv'
248     try:
249         df_pop_density = pd.read_csv(CSV_POP_DENSITY)
250     except:
251         print('Cannot open data file - Population Density by Ward')
252         df_density = df_pop_density[(df_pop_density['Year'] == 2017)]
253         density_columns = ['Code','Population_per_square_kilometre']
254         df_density = pd.DataFrame(df_density, columns = density_columns)
255         df_density.to_csv('datastore/PopDensity.csv', index=False)
256     return(df_pop_density)
257
258
259 def importLifeExpectancy():
260     CSV_LIFE_EX =
261     'https://files.datapress.com/london/dataset/life-expectancy-birth-and-age-65-ward
262 /2016-02-09T14:25:25/life-expectancy-ward-at-Birth.csv'
263     try:
264         df_expectancy = pd.read_csv(CSV_LIFE_EX)
265     except:
266         print('Cannot open data file - Life Expectancy by Ward')
267         df_life_ex = df_expectancy[(df_expectancy['Geography'] == 'Ward')]

```

```
260 lifeex_columns = ['Ward','2010-14;Female;Life expectancy at  
261     birth','2010-14;Male;Life expectancy at birth']  
262 df_life_ex = pd.DataFrame(df_life_ex, columns = lifeex_columns)  
263 df_life_ex.rename(columns=  
264 {'Ward':'Ward_Code', '2010-14;Female;Life expectancy at  
265     birth':'Female', '2010-14;Male;Life expectancy at birth':'Male'}  
266 ,inplace=True)  
267 df_life_ex['Life_Ex'] = (df_life_ex['Female'] + df_life_ex['Male'])/2  
268 del df_life_ex['Female']  
269 del df_life_ex['Male']  
270 df_life_ex.to_csv('datastore/LifeExpectancy.csv', index=False)  
271 return(df_life_ex)  
272  
273 def importChildObesity():  
274     XLSX_CHILD_OBESITY =  
275     'https://files.datapress.com/london/dataset/prevalence-childhood-obesity-borough/2015-09-21T  
276     /MSOA_Ward_LA_Obesity.xlsx'  
277     try:  
278         df_childobesity = pd.read_excel(XLSX_CHILD_OBESITY,  
279             '2011-12_2013-14', skiprows=3)  
280     except:  
281         print('Cannot open data file - Childhood Obesity by Ward')  
282     df_childobesity.rename(columns={'Unnamed: 13':'Obese%'},inplace=True)  
283     obesity_columns = ['Geog Level','Code','LA code','Obese%']  
284     df_childobesity = pd.DataFrame(df_childobesity, columns =  
285         obesity_columns)  
286     df_obesityborough =  
287         pd.DataFrame(df_childobesity[(df_childobesity['Geog Level'] ==  
288             'LA')])  
289     df_obesityborough.rename(columns={'Obese%':'BoroughObese%'},inplace=True)  
290     borough_columns = ['Code','BoroughObese%']  
291     df_obesityborough = pd.DataFrame(df_obesityborough, columns =  
292         borough_columns)  
293     df_childobesity =  
294         pd.DataFrame(df_childobesity[(df_childobesity['Geog Level'] ==  
295             'Ward')])  
296     df_childobesity = pd.merge(df_childobesity, df_obesityborough,  
297         left_on='LA code', right_on='Code')  
298     df_childobesity['Obese%'].replace('s',  
299         df_childobesity['BoroughObese%'], inplace=True)  
300     ward_columns = ['Code_x', 'Obese%']  
301     df_childobesity = pd.DataFrame(df_childobesity, columns =  
302         ward_columns)  
303     df_childobesity.to_csv('datastore/ChildObesity.csv', index=False)  
304     return(df_childobesity)  
305  
306 def importIllness():  
307     XLSX_ILLNESS =  
308     'https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/467775  
309     /File_8_ID_2015_Underlying_indicators.xlsx'  
310     CSV_LSOA_MAPPING =  
311     'https://opendata.arcgis.com/datasets/07a6d14d4a0540769f0662f4d1450bae_0.csv'  
312     try:  
313         df_illness = pd.read_excel(XLSX_ILLNESS, 'ID 2015 Health Domain')  
314     except:  
315         print('Cannot open data file - Illness by Ward')  
316     try:
```

```
305     df_lsoa_ward = pd.read_csv(CSV_LSOA_MAPPING)
306 except:
307     print('Cannot open data file - LSOA to Ward Mapping')
308 df_illness =
309 pd.merge(df_illness, df_lsoa_ward, left_on='LSOA code (2011)',
310         right_on='LSOA11CD')
311 shapes = 'datastore/Shapes/London_Ward.shp'
312 wards = gpd.read_file(shapes)
313 df_illness = pd.merge(wards, df_illness, left_on='GSS_CODE',
314         right_on='WD15CD')
315 df_illness.to_csv('datastore/Illness.csv', index=False)
316 return(df_illness)
```

Bibliography

- Agafonkin, Vladimir (2017). URL: <https://leafletjs.com/examples/choropleth/>.
- Bettencourt, Luís M. A. et al. (2010). "Urban Scaling and Its Deviations: Revealing the Structure of Wealth, Innovation and Crime across Cities". In: *PLoS ONE* 5.11. DOI: [10.1371/journal.pone.0013541](https://doi.org/10.1371/journal.pone.0013541).
- Bhutan Studies, Centre for (2011). URL: <https://ophi.org.uk/policy/national-policy/gross-national-happiness-index/>.
- esri (2017). URL: <https://hub.arcgis.com/pages/open-data>.
- Foursquare (2018). URL: <https://foursquare.com/about>.
- GLA (2014). *the Mayor of London and the London Assembly*. URL: <https://data.london.gov.uk/dataset/london-ward-well-being-scores>.
- (2018). *the Mayor of London and the London Assembly*. URL: <https://data.london.gov.uk/>.
- Hristova, Desislava, Luca M. Aiello, and Daniele Quercia (2018). "The New Urban Success: How Culture Pays". In: *Frontiers in Physics* 6, p. 27. ISSN: 2296-424X. DOI: [10.3389/fphy.2018.00027](https://doi.org/10.3389/fphy.2018.00027). URL: <https://www.frontiersin.org/article/10.3389/fphy.2018.00027>.
- Lyubomirsky, Sonja, Laura King, and Ed Diener (2005). "The Benefits of Frequent Positive Affect: Does Happiness Lead to Success?" In: *Psychological Bulletin* 131.6, 803–855. DOI: [10.1037/0033-2909.131.6.803](https://doi.org/10.1037/0033-2909.131.6.803).
- MappingLondon.co.uk (2018). URL: <http://mappinglondon.co.uk/>.
- OECD (2016). *Well-being in Danish Cities - en*. URL: <http://www.oecd.org/publications/well-being-in-danish-cities-9789264265240-en.htm>.
- (2017). *How's life?* URL: <http://www.oecdbetterlifeindex.org/>.
- Pareto, Adriano (Apr. 2013). "Methods for Constructing Composite Indices: One for All or All for One?" In: LXVII, pp. 67–80.
- Quattrone, Giovanni et al. (2016). "Who Benefits from the "Sharing" Economy of Airbnb?" In: *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pp. 1385–1394. DOI: [10.1145/2872427.2874815](https://doi.org/10.1145/2872427.2874815). URL: [http://doi.acm.org/10.1145/2872427.2874815](https://doi.acm.org/10.1145/2872427.2874815).
- Quercia, Daniele, Luca Maria Aiello, and Rossano Schifanella (2016). "The Emotional and Chromatic Layers of Urban Smells". In: *CoRR* abs/1605.06721. arXiv: [1605.06721](https://arxiv.org/abs/1605.06721). URL: [http://arxiv.org/abs/1605.06721](https://arxiv.org/abs/1605.06721).
- Ratcliffe, Anita (2013). "Wealth Effects, Local Area Attributes, and Economic Prospects: On the Relationship between House Prices and Mental Wellbeing". In: *Review of Income and Wealth* 61.1, 75–92. DOI: [10.1111/roiw.12075](https://doi.org/10.1111/roiw.12075).
- RCA (2014). *The Great British Public Toilet Map*. URL: <https://www.toiletmap.org.uk/>.
- Toronto, City of (2018). *Neighbourhoods and Communities*. URL: <https://www.toronto.ca/city-government/data-research-maps/neighbourhoods-communities/>.
- Waterloo, University of (2017). *Home | Canadian Index of Wellbeing*. URL: <https://uwaterloo.ca/canadian-index-wellbeing/>.
- Zoopla (2018). *Heatmap of UK property values*. URL: <https://www.zoopla.co.uk/heatlmaps/>.