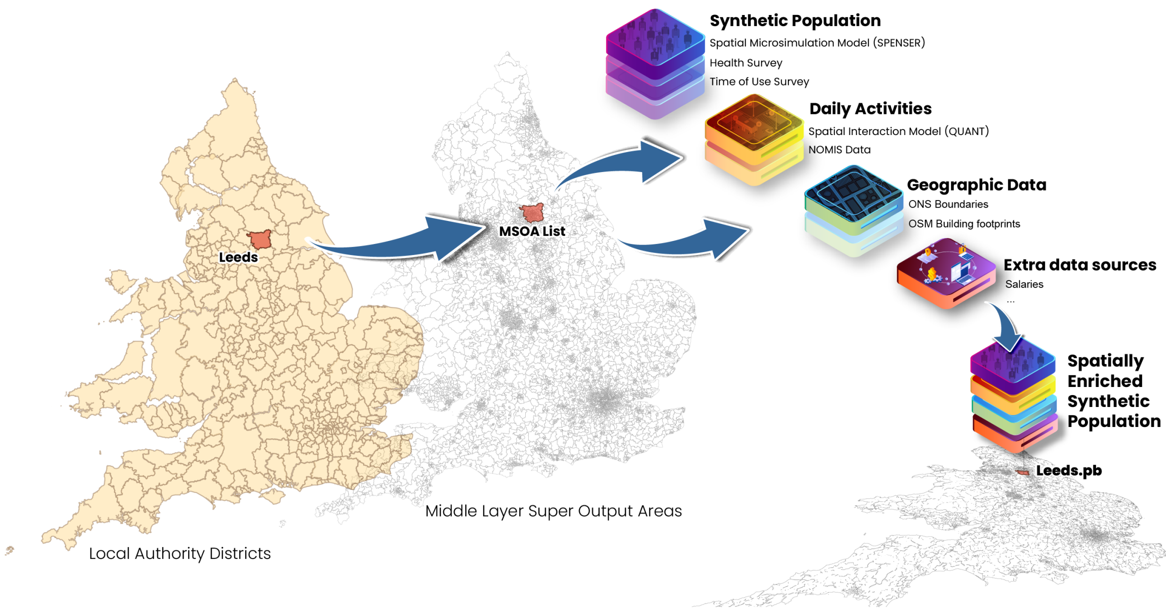
Synthetic Population Catalyst

Table of contents

1 Getting started



The Synthetic Population Catalyst (SPC) makes it easier for researchers to work with synthetic population data in England. It combines a variety of data sources and outputs a single file in protocol buffer format, describing the population in a given study area. The data includes demographic, health, and daily activity data per person, and information about the venues where people conduct activities.

You can use SPC output to catalyze your own project. Rather than join together many raw data sources yourself and deal with missing and messy data, you can leverage SPC's effort and well-documented schema.

To get started:

1. Download sample data for a county in England
2. Explore how to use the data
3. If you need a different study area, build and then run SPC

You can also download this site as a PDF and find all code on Github.

This work was supported by Wave 1 of The UKRI Strategic Priorities Fund under the EPSRC Grant EP/W006022/1, particularly the "Ecosystem of Digital Twins" and "Shocks and Resilience" themes within that grant & The Alan Turing Institute

Part I

Using SPC

2 SPC Outputs

You don't need to run SPC yourself. See config/ for the list of MSOAs covered by each study area. If you want to run SPC for a different list of MSOAs, see here.

One of the advantages of using SPC is that help researches to mimic the population characteristics and its iterations through multiples years (see for more details). So you can replicate what the population might look like across multiple periods of time. Initially check what country you would like to explore, then pick the year to get the outcome file. In case you want to explore it and see how does the data look like, and what attributes are included, load the output in our SPC Explorer and get inspired about the potential applications you could co-create using these outcomes.

- England (Available years: 2012, 2020, 2022, 2039)
- Wales (Available years: 2012, 2020, 2022, 2039)

We also included some special areas for your testing:

- special/2012/northwest_transpennine.pb.gz
- special/2020/northwest_transpennine.pb.gz
- special/2022/northwest_transpennine.pb.gz
- special/2032/northwest_transpennine.pb.gz
- special/2039/northwest_transpennine.pb.gz

If you use SPC code or data in your work, please cite using the Zenodo DOI (using the bottom-right tool to generate the citation).

2.1 Versioning

Over time, we may add more data to SPC or change the schema. Protocol buffers are designed to let combinations of new/old code and data files work together, but we don't intend to use this feature. We may make breaking changes, like deleting fields. We'll release a new version of the schema and output data every time and document it here. You should depend on a specific version of the data output in your code, so new releases don't affect you until you decide to update.

- v1: released 25/04/2022, schema

- v1.1, released 27/05/2022, schema
  - added `pwkstat`, `salary_hourly`, `salary_yearly`, and `idp`
  - reorganized `Identifiers` and `Employment` attributes
  - non-breaking change added 02/08/2022: added `bmi_new` field
- v1.2, released 29/12/2022, schema
  - switched to proto2 and made some fields optional
  - adjusted some numeric enum values to match ONS
- v2, released 09/03/2023, schema
  - new per-person and per-household fields
  - various changes to existing fields (adjusting enum number, removing the BMI enum, etc)
  - adding time-use diaries
  - expanding to Wales
  - adding multiple years of output

3 Outputs for England Counties

Check the year you would like to explore and pick the corresponding file based on the region you are interested. Remember if you want to explore the data you can load the output in our SPC explorer

- 2012:
  - bedfordshire.pb.gz
  - berkshire.pb.gz
  - bristol.pb.gz
  - buckinghamshire.pb.gz
  - cambridgeshire.pb.gz
  - cheshire.pb.gz
  - cornwall.pb.gz
  - cumbria.pb.gz
  - England/2012/derbyshire.pb.gz
  - devon.pb.gz
  - durham.pb.gz
  - east-sussex.pb.gz
  - east-yorkshire-with-hull.pb.gz
  - essex.pb.gz
  - gloucestershire.pb.gz
  - greater-london.pb.gz
  - greater-manchester.pb.gz
  - hampshire.pb.gz
  - herefordshire.pb.gz
  - hertfordshire.pb.gz
  - isle-of-wight.pb.gz
  - kent.pb.gz
  - lancashire.pb.gz
  - leicestershire.pb.gz
  - lincolnshire.pb.gz
  - merseyside.pb.gz
  - norfolk.pb.gz
  - northamptonshire.pb.gz
  - northumberland.pb.gz

- north-yorkshire.pb.gz
- nottinghamshire.pb.gz
- oxfordshire.pb.gz
- rutland.pb.gz
- shropshire.pb.gz
- somerset.pb.gz
- south-yorkshire.pb.gz
- staffordshire.pb.gz
- suffolk.pb.gz
- surrey.pb.gz
- tyne-and-wear.pb.gz
- warwickshire.pb.gz
- west-midlands.pb.gz
- west-sussex.pb.gz
- west-yorkshire.pb.gz
- wiltshire.pb.gz
- worcestershire.pb.gz

- 2020:
  - bedfordshire.pb.gz
  - berkshire.pb.gz
  - bristol.pb.gz
  - buckinghamshire.pb.gz
  - cambridgeshire.pb.gz
  - cheshire.pb.gz
  - cornwall.pb.gz
  - cumbria.pb.gz
  - derbyshire.pb.gz
  - devon.pb.gz
  - durham.pb.gz
  - east-sussex.pb.gz
  - east-yorkshire-with-hull.pb.gz
  - essex.pb.gz
  - gloucestershire.pb.gz
  - greater-london.pb.gz
  - greater-manchester.pb.gz
  - hampshire.pb.gz
  - herefordshire.pb.gz
  - hertfordshire.pb.gz
  - isle-of-wight.pb.gz
  - kent.pb.gz
  - lancashire.pb.gz

- leicestershire.pb.gz
- lincolnshire.pb.gz
- merseyside.pb.gz
- norfolk.pb.gz
- northamptonshire.pb.gz
- northumberland.pb.gz
- north-yorkshire.pb.gz
- nottinghamshire.pb.gz
- oxfordshire.pb.gz
- rutland.pb.gz
- shropshire.pb.gz
- somerset.pb.gz
- south-yorkshire.pb.gz
- staffordshire.pb.gz
- suffolk.pb.gz
- surrey.pb.gz
- tyne-and-wear.pb.gz
- warwickshire.pb.gz
- west-midlands.pb.gz
- west-sussex.pb.gz
- west-yorkshire.pb.gz
- wiltshire.pb.gz
- worcestershire.pb.gz

- 2022:
  - bedfordshire.pb.gz
  - berkshire.pb.gz
  - bristol.pb.gz
  - buckinghamshire.pb.gz
  - cambridgeshire.pb.gz
  - cheshire.pb.gz
  - cornwall.pb.gz
  - cumbria.pb.gz
  - derbyshire.pb.gz
  - devon.pb.gz
  - durham.pb.gz
  - east-sussex.pb.gz
  - east-yorkshire-with-hull.pb.gz
  - essex.pb.gz
  - gloucestershire.pb.gz
  - greater-london.pb.gz
  - greater-manchester.pb.gz

- hampshire.pb.gz
- herefordshire.pb.gz
- hertfordshire.pb.gz
- isle-of-wight.pb.gz
- kent.pb.gz
- lancashire.pb.gz
- leicestershire.pb.gz
- lincolnshire.pb.gz
- merseyside.pb.gz
- norfolk.pb.gz
- northamptonshire.pb.gz
- northumberland.pb.gz
- north-yorkshire.pb.gz
- England/2022/nottinghamshire.pb.gz
- oxfordshire.pb.gz
- rutland.pb.gz
- shropshire.pb.gz
- somerset.pb.gz
- south-yorkshire.pb.gz
- staffordshire.pb.gz
- suffolk.pb.gz
- surrey.pb.gz
- tyne-and-wear.pb.gz
- warwickshire.pb.gz
- west-midlands.pb.gz
- west-sussex.pb.gz
- west-yorkshire.pb.gz
- wiltshire.pb.gz
- worcestershire.pb.gz

- 2032:
  - bedfordshire.pb.gz
  - berkshire.pb.gz
  - bristol.pb.gz
  - buckinghamshire.pb.gz
  - cambridgeshire.pb.gz
  - cheshire.pb.gz
  - cornwall.pb.gz
  - cumbria.pb.gz
  - derbyshire.pb.gz
  - devon.pb.gz
  - durham.pb.gz

  - east-sussex.pb.gz
  - east-yorkshire-with-hull.pb.gz
  - essex.pb.gz
  - gloucestershire.pb.gz
  - greater-london.pb.gz
  - greater-manchester.pb.gz
  - hampshire.pb.gz
  - herefordshire.pb.gz
  - hertfordshire.pb.gz
  - isle-of-wight.pb.gz
  - kent.pb.gz
  - lancashire.pb.gz
  - leicestershire.pb.gz
  - lincolnshire.pb.gz
  - merseyside.pb.gz
  - norfolk.pb.gz
  - northamptonshire.pb.gz
  - northumberland.pb.gz
  - north-yorkshire.pb.gz
  - nottinghamshire.pb.gz
  - oxfordshire.pb.gz
  - rutland.pb.gz
  - shropshire.pb.gz
  - somerset.pb.gz
  - south-yorkshire.pb.gz
  - staffordshire.pb.gz
  - suffolk.pb.gz
  - surrey.pb.gz
  - tyne-and-wear.pb.gz
  - warwickshire.pb.gz
  - west-midlands.pb.gz
  - west-sussex.pb.gz
  - west-yorkshire.pb.gz
  - wiltshire.pb.gz
  - worcestershire.pb.gz

- 2039:

  - bedfordshire.pb.gz
  - berkshire.pb.gz
  - bristol.pb.gz
  - buckinghamshire.pb.gz
  - cambridgeshire.pb.gz

- cheshire.pb.gz
- cornwall.pb.gz
- cumbria.pb.gz
- derbyshire.pb.gz
- devon.pb.gz
- durham.pb.gz
- east-sussex.pb.gz
- east-yorkshire-with-hull.pb.gz
- essex.pb.gz
- gloucestershire.pb.gz
- greater-london.pb.gz
- greater-manchester.pb.gz
- hampshire.pb.gz
- herefordshire.pb.gz
- hertfordshire.pb.gz
- isle-of-wight.pb.gz
- kent.pb.gz
- lancashire.pb.gz
- leicestershire.pb.gz
- lincolnshire.pb.gz
- merseyside.pb.gz
- norfolk.pb.gz
- northamptonshire.pb.gz
- northumberland.pb.gz
- north-yorkshire.pb.gz
- nottinghamshire.pb.gz
- oxfordshire.pb.gz
- rutland.pb.gz
- shropshire.pb.gz
- somerset.pb.gz
- south-yorkshire.pb.gz
- staffordshire.pb.gz
- suffolk.pb.gz
- surrey.pb.gz
- tyne-and-wear.pb.gz
- warwickshire.pb.gz
- west-midlands.pb.gz
- west-sussex.pb.gz
- west-yorkshire.pb.gz
- wiltshire.pb.gz
- worcestershire.pb.gz

If you use SPC code or data in your work, please cite using the Zenodo DOI (using the bottom-right tool to generate the citation).

4 Outputs for Wales Counties

Check the year you would like to explore and pick the corresponding file based on the region you are interested. Remember if you want to explore the data you can load the output in our SPC explorer

- 2012:
  - bridgend-and-neath-port-talbot.pb.gz
  - cardiff-and-vale-of-glamorgan.pb.gz
  - central-valleys.pb.gz
  - conwy-and-denbighshire.pb.gz
  - flintshire-and-wrexham.pb.gz
  - gwent-valleys.pb.gz
  - gwynedd.pb.gz
  - isle-of-anglesey.pb.gz
  - monmouthshire-and-newport.pb.gz
  - powys.pb.gz
  - south-west-wales.pb.gz
  - swansea.pb.gz
- 2020:
  - bridgend-and-neath-port-talbot.pb.gz
  - cardiff-and-vale-of-glamorgan.pb.gz
  - central-valleys.pb.gz
  - conwy-and-denbighshire.pb.gz
  - flintshire-and-wrexham.pb.gz
  - gwent-valleys.pb.gz
  - gwynedd.pb.gz
  - isle-of-anglesey.pb.gz
  - monmouthshire-and-newport.pb.gz
  - powys.pb.gz
  - south-west-wales.pb.gz
  - Wales/2020/swansea.pb.gz
- 2022:
  - Wales/2022/bridgend-and-neath-port-talbot.pb.gz

  - Wales/2022/cardiff-and-vale-of-glamorgan.pb.gz
  - Wales/2022/central-valleys.pb.gz
  - Wales/2022/conwy-and-denbighshire.pb.gz
  - Wales/2022/flintshire-and-wrexham.pb.gz
  - Wales/2022/gwent-valleys.pb.gz
  - Wales/2022/gwynedd.pb.gz
  - Wales/2022/isle-of-anglesey.pb.gz
  - Wales/2022/monmouthshire-and-newport.pb.gz
  - Wales/2022/powys.pb.gz
  - Wales/2022/south-west-wales.pb.gz
  - Wales/2022/swansea.pb.gz

- 2032:
  - bridgend-and-neath-port-talbot.pb.gz
  - cardiff-and-vale-of-glamorgan.pb.gz
  - central-valleys.pb.gz
  - conwy-and-denbighshire.pb.gz
  - flintshire-and-wrexham.pb.gz
  - gwent-valleys.pb.gz
  - gwynedd.pb.gz
  - isle-of-anglesey.pb.gz
  - monmouthshire-and-newport.pb.gz
  - powys.pb.gz
  - south-west-wales.pb.gz
  - swansea.pb.gz

- 2039:
  - bridgend-and-neath-port-talbot.pb.gz
  - cardiff-and-vale-of-glamorgan.pb.gz
  - central-valleys.pb.gz
  - conwy-and-denbighshire.pb.gz
  - flintshire-and-wrexham.pb.gz
  - gwent-valleys.pb.gz
  - gwynedd.pb.gz
  - isle-of-anglesey.pb.gz
  - monmouthshire-and-newport.pb.gz
  - powys.pb.gz
  - south-west-wales.pb.gz
  - swansea.pb.gz

If you use SPC code or data in your work, please cite using the Zenodo DOI (using the bottom-right tool to generate the citation).

5 Using the SPC output file

Once you download or generate an SPC output file for your study area, how do you use it? Each study area consists of one `.pb` or protocol buffer file. This file efficiently encodes data following this schema. Read more about what data is contained in the output.

You can read the "protobuf" (shorthand for a protocol buffer file) in any supported language, and then extract and transform just the parts of the data you want for your model.

We have examples for Python below, but feel free to request other languages.

We have a web app using Svelte to interactively explore SPC data. Its source code is great reference for how to use the proto output.

5.1 Python

To work with SPC protobufs in Python, you need two dependencies setup:

- The protobuf library
  - You can install system-wide with `pip install protobuf`
  - Or add as a dependency to a conda, poetry, etc environment
- The generated Python library, synthpop_pb2.py
  - You can download a copy of this file into your codebase, then `import synthpop_pb2`
  - You can also generate the file yourself, following the docs: `protoc --python_out=python/ synthpop.proto`

5.1.1 Converting to Pandas data-frames and CSV

The schema expresses relationships between people, households, and venues that can't all be captured by a simple 2D table. Nevertheless, you can extract per-person information and express as a dataframe or CSV file. See this example Python script for inspiration. You can try it out:

```
# Download a file
wget https://ramp0storage.blob.core.windows.net/spc-output/v1/rutland.pb.gz
```

```
# Uncompress
gunzip rutland.pb.gz
# Convert the .pb to JSON
python3 python/protobuf_to_csv.py --input_path data/output/rutland.pb
# View the output
less people.csv
```

5.1.2 Converting .pb file to JSON format

To interactively explore the data, viewing JSON is much easier. It shows the same structure as the protobuf, but in a human-readable text format. The example below uses a small Python script:

```
# Download a file
wget https://ramp0storage.blob.core.windows.net/spc-output/v1/rutland.pb.gz
# Uncompress
gunzip rutland.pb.gz
# Convert the .pb to JSON
python3 python/protobuf_to_json.py data/output/rutland.pb > rutland.json
# View the output
less rutland.json
```

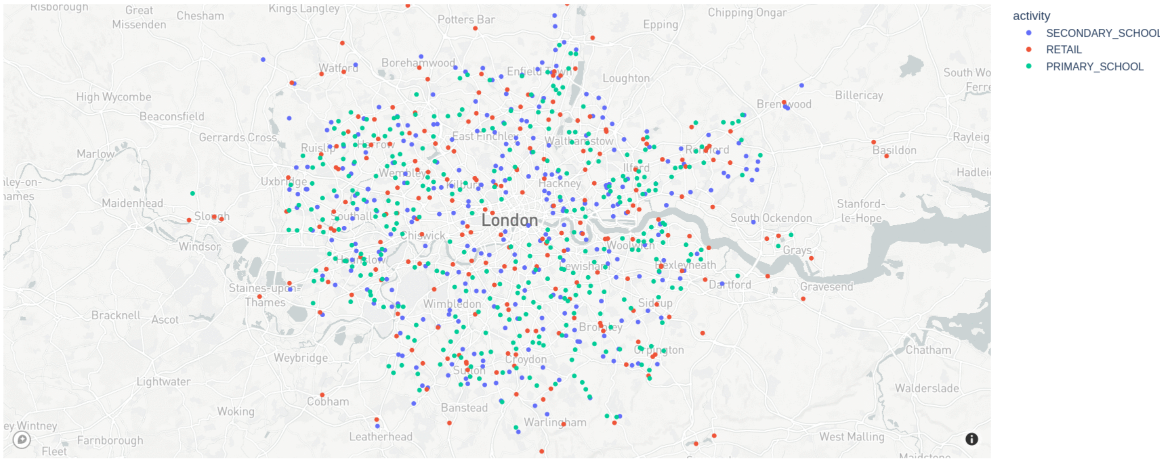
5.1.3 Converting to numpy arrays

The ASPICS project simulates the spread of COVID through a population. The code uses numpy, and this script converts the protobuf to a bunch of different numpy arrays.

Note the ASPICS code doesn't keep using the generated Python protobuf classes for the rest of the pipeline. Data frames and numpy arrays may be more familiar and appropriate. The protobuf is a format optimized for reading and writing; you don't need to use it throughout all of your model code.

5.1.4 Visualizing venues

Use this script to read a protobuf file, then draws a dot for every venue, color-coded by activity.


activity
SECONDARY_SCHOOL
RETAIL
PRIMARY_SCHOOL
London

6 Installation

You only need to compile SPC to run for a custom set of MSOAs. Just download existing output if your study area matches what we provide.

- **Rust**: The latest stable version of Rust: https://www.rust-lang.org/tools/install

6.1 Compiling SPC

```
git clone https://github.com/alan-turing-institute/uatk-spc/
cd uatk-spc
# The next command will take a few minutes the first time you do it, to build external dep
cargo build --release
```

6.2 Troubleshooting downloading

If you get an error `No such file or directory (os error 2)` it might be because a previous attempt to run SPC failed, and some necessary files were not fully downloaded. In these cases you could try deleting the `data/raw_data` directory and then running SPC again. It should automatically try to download the big files again.

If you have trouble downloading any of the large files, you can download them manually. The logs will contain a line such as `Downloading https://ramp0storage.blob.core.windows.net/nationaldata/` `to data/raw_data/nationaldata/QUANT_RAMP_spc.tar.gz`. This tells you the URL to retrieve, and where to put the output file. Note that SPC won't attempt to download files if they already exist, so if you wind up with a partially downloaded file, you have to manually remove it.

7 Creating new study areas

If the area you want to model isn't already generated, then you can follow this guide to run SPC on a custom area. You must first compile SPC.

SPC takes a newline-separated list of MSOAs in the `config/` directory as input, like this. You can generate this list from a LAD (local authority district). From the main SPC directory, run `python scripts/select_msoas.py`. Refer to `data/raw_data/referencedata/lookUp.csv` (only available after running SPC once) for all geographies available.

This script will create a new file, `config/your_region.txt`.

7.1 Run SPC for the new area

From the main directory, just run:

```
cargo run --release -- config/your_region.txt
```

This will download some large files the first time. You'll wind up with `data/output/your_region.pb` as output, as well as lots of intermediate files in `data/raw_data/`. The next time you run this command (even on a different study area), it should go much faster.

7.2 (Optional) run SPC for lots of areas

If you want to run the program over lots of areas at once and are using Mac/Linux, you can use a `for` loop in a terminal to repeatedly run SPC over all files in the config directory. For example, this will run SPC on all `.txt` files in the `config` directory:

```
for file in config/*.csv; do cargo run --release -- config/$file; done
```

7.3 Using the output

After you generate the files, see here for how to use them in your project.

If you use SPC code or data in your work, please cite using the Zenodo DOI (using the bottom-right tool to generate the citation).

Part II

Understanding SPC

8 Data schema

Here are some helpful tips for understanding the schema.

Each .pb file contains exactly one `Population` message. In contrast to datasets consisting of multiple `.csv` files, just a single file contains everything. Some of the fields in `Population` are lists (of people and households) or maps (of venues keyed by activity, or of MSOAs). Unlike a flat `.csv` table, there may be more lists embedded later. Each `Household` has a list of `members`, for example.

The different objects refer to each other, forming a graph structure. The protobuf uses `uint64` IDs to index into other lists. For example, if some household has `members = [3, 10]`, then those two people can be found at `population.people[3]` and `population.people[10]`. Each of them will have the same `household` ID, pointing back to something in the `population.households` list.

8.1 Flows: modelling daily activites

SPC models daily travel behavior of people as "flows." Flows are broken down by by an activity – shopping/retail, attending primary or secondary school, working, or staying at home. For each activity type, a person has a list of venues where they may do that activity, weighted by a probability of going to that particular venue.

Note that `flows_per_activity` is stored in `InfoPerMSOA`, not `Person`. The flows for retail and school are only known at the MSOA level, not individually. So given a particular `Person` object, you first look up their household's MSOA – `msoa = population.households[ person.household ].msoa` and then look up flows for that MSOA – `population.info_per_msoa[msoa].flows_per_activity`.

Each person has exactly 1 flow for home – it's just `person.household` with probability 1. A person has 0 or 1 flows to work, based on the value of `person.workplace`.

This doesn't mean that all people in the same MSOA share the same travel behavior. Each person has their own `activity_durations` field, based on time-use survey data. Even if two people share the same set of places where they may go shopping, one person may spend much more time on that activity than another.

See the ASPICS conversion script for all of this in action – it has a function to collapse a person's flows down into a single weighted list.

Note that per MSOA, very few venues are represented as destinations – 10 for retail and 5 for school. Only the most likely venues from QUANT are used.

8.2 Flow weights

How do you interpret the probabilities/weights for flows? If your model needs people to visit specific places each day, you could randomly sample a venue from the flows, weighting them appropriately. For retail, you may want to repeat this sampling every day of the simulation, so they visit different venues. For primary and secondary school, it may be more appropriate to sample once and store that for the simulation – a student probably doesn't switch schools daily.

Alternatively, you can follow what ASPICS does. Every day, each person logically visits all possible venues, but their interaction there (possibly receiving or transmitting COVID) is weighted by the probability of each venue.

9 Modelling methods

The principles behind the generation of the enriched SPENSER population data and behind the modelling of trips to schools and retail from QUANT are detailed in

> Spooner F et al. A dynamic microsimulation model for epidemics. Soc Sci Med., 291:114461 (2021). (DOI)

> Lomax N et al. An Open-Source Model for Projecting Small Area Demographic and Land-Use Change. Geographical analysis, 54(3), 599-622 (2022). (DOI)

In order to distribute each individual of the population to a unique physical workplace, we first created a population of all individual workplaces in England, based on a combination of the Nomis UK Business Counts 2020 dataset and the Nomis Business register and Employment Survey 2015 (see Data sources). The first dataset gives the number of individual workplace counts per industry, using the SIC 2007 industry classification, with imprecise size (i.e. number of employees) bands at MSOA level. The second dataset gives the total number of jobs available at LSOA level per SIC 2007 industry category. We found that the distribution of workplace sizes follows closely a simple 1/x distribution, allowing us to draw for each workplace a size within their band, with sum constraints given by the total number of jobs available, according to the second dataset.

The workplace 'population' and individual population are then levelled for each SIC 2007 category by removing the exceeding part of whichever dataset lists more items. This takes into account that people and business companies are likely to over-report their working availability (e.g. part time and seasonal contracts are not counted differently than full time contracts, jobseekers or people on maternity leave might report the SIC of their last job). This process can be controlled by a threshold in the parameter file that defines the maximal total proportion of workers or jobs that can be removed. If the two datasets cannot be levelled accordingly, the categories are dropped and the datasets are levelled globally. Tests in the West Yorkshire area have shown than when the level 1 SIC, containing 21 unique categories, is used, 90% of the volume of commuting flows were recovered compared to the Nomis commuting OD matrices at MSOA level.

The employees for each workplace are drawn according to the 'universal law of visitation', see

> Schläpfer M et al. The universal visitation law of human mobility. Nature 593, 522–527 (2021). (DOI)

This framework predicts that visitors to any destination follow a simple

(r,f)= K / (rf)2

distribution, where (r,f) is the density of visitors coming from a distance r with frequency f and K is a balancing constant depending on the specific area. In the context of commuting, it can be assumed that f = 1. Additionally, we only need to weigh potential employees against each other, which removes the necessity to compute explicitly K. In the West Yorkshire test, we found a Pearson coefficient of 0.7 between the predicted flows when aggregated at MSOA level and the OD matrix at MSOA level available from Nomis.

9.1 Income data

This modelling is mainly based on the 2020 revised edition of the Earnings and hours worked, region by occupation by four-digit SOC: ASHE Table 15 database from ONS. Some percentiles for employees' gross hourly salaries are provided for each full-time and part-time job according to their four-digit SOC classification per region, and separated by sex.

9.1.1 Methods

The data are far from complete (only about 15% of all possible values), especially for the highest deciles. We found that an order 3 polynomial fit was satisfactory for most categories (93.11%) to complete the partially filled SOCs. SOCs with too many missing values are given the value for the category that is immediately higher in the SOC hierarchy. Some jobs appear to have a 'ceiling' for the highest percentiles, making the polynomial fit fail. In that case, we have replaced the unknown values by the highest known value in the raw data (as there is no clear and systemic fit for these special cases). In addition, there is no information for the highest decile in all cases, which means that the highest salaries are underestimated (and exceptionally high salaries cannot be obtained). The result of this phase is four tables {male full-time, male part-time, female full-time, female part-time} containing the coefficients of the fitted order 3 polynomial, with an optional ceiling percentile when relevant.

A percentile is chosen randomly (uniformly) for each individual, and the salary is then deduced according to their full-time/part-time status, region, sex and SOC category. A basic hourly salary column is added to the unprocessed SPC data, as well as a corresponding annual salary based on their estimated hours worked per day, according to the Time Use Survey matching. In addition, we repeat this process for all individuals that are categorised as 'Self-employed' or 'Employee unspecified' by the Time Use Survey matching„ as if they were full time employees. These values are recorded in the columns `IncomeHAsIF` and `IncomeYAsIf`. We noticed that a high number of employees were given no worked hours by the Time Use Survey. We have added to the `IncomeYAsIf` column an estimation of their annual salary based on Table 15.9a:

Paid hours worked - Total 2020, and also depending on the same four variables as above (full-time/part-time status, region, sex and SOC category).

In addition, age data are made available by ONS. Part of the differences that can be observed between different age groups are already taken into account through the fact that the SOC category can evolve during a career. To take into account that dependence, we first run the above method without weighing by age. The results are shown in the age validation section below. The residual impact of age alone is then added to the model in the following way. When the percentile is drawn for a specific individual, it is morphed to fit within the usual percentage range accessible to that age category. The function that operates this morphing is inferred beforehand and takes into account the salary distribution per age computed by the previous non-age weighted iteration of the modelling (see figure - TBA - for a more detailed description of this function).

The R codes for this modelling are here.

The methods are validated in the next section. Since it is not possible to optimise every criterion at once, this next secton can also be used as a reference to re-adjust some values to match exactly the ONS estimated means for one particular criterion of interest.

9.1.2 Comparison to reference values from ONS

We compare the results of the modelling to the raw datasets from ONS.

- `Mod` for modelled
- `M` for male
- `F` for female
- `H` for hourly gross salary
- `Y` for annual gross salary
- `FT` for full-Time
- `PT` for part-Time
- Only individuals recorded as employees (i.e. not self-employed) are taken into account in this section.

Number of employees per sex and full-time/part-time classification

The numbers given by ONS vary from dataset to dataset and are reported by ONS as indicative only. For the modelled values, we give the total number of individuals with a non-zero salary in each category.

Variable	All	FT	PT	M	M FT	M PT	F	F FT	F PT
ONS tot	22-26k	16-19k	6-8k	11-13k	9-11k	1.5-2k	11-13k	6.5-7.5k	4.5-5.5k

Variable	All	FT	PT	M	M FT	M PT	F	F FT	F PT
Mod tot H	23.1k	18.5k	4.6k	11.8k	11k	0.8k	11.3k	7.5k	3.8k
Mod tot Y	17.6k	14.8k	2.8k	9.4k	8.9k	0.5k	8.2k	5.9k	2.3k

A significant number of individuals listed as working either full or part time have 0 effective worked hours per day according to the Time Use Survey matching. In those cases, an hourly salary is modelled depending on their SOC, region and sex, as for any other employee, but the annual salary will be displayed as 0. It is possible to estimate their likely true number of hours worked from the same ONS dataset (Table 15.9a: Paid hours worked - Total 2020), also depending on their sex, soc and region. This calculation has been added to the "As If" column.

Hourly gross salary per sex and full-time/part-time classification

Variable	All	FT	PT	M	M FT	M PT	F	F FT	F PT
ONS mean	17.63	18.32	13.93	18.81	19.12	14.69	16.19	17.08	13.68
ONS median	13.71	15.15	10.38	14.84	15.58	10.12	12.58	14.42	10.47
Mod mean	16.45	17.19	13.45	17.50	17.84	12.75	15.35	16.23	13.60
Mod median	13.55	14.46	10.23	14.27	14.72	9.16	12.79	14.12	10.51

The median values are quite close to the ONS values, but the mean values are always lower. This is expected, see the description of the modelling above.

Annual gross salary per sex and full-time/part-time classification

Only values > 0 are retained for these calculations.

Variable	All	FT	PT	M	M FT	M PT	F	F FT	F PT
ONS mean	31,646	38,552	13,819	38,421	42,072	14,796	24,871	33,253	13,512
ONS median	25,886	31,487	11,240	31,393	33,915	10,883	20,614	28,002	4,743
Mod mean	34,317	36,595	22,257	37,574	38,496	20,698	30,594	33,729	22,585
Mod median	28,713	30,942	17,928	31,404	32,382	17,382	25,875	29,028	18,137

The average salary for part-time employees is correct when values equal to 0 are taken into account. This suggests that the total number of hours worked for part-time employees is correct, but the way they are distributed among individuals is not. It could be due to the TUS taking a snapshot of the situation during a particular week, rather than averaging their data over the year. It appears that the TUS matching also overestimates the average number of hours worked for female employees.

Regional differences (hourly gross salary)

Region	East	East Midlands	London	North East	North West	South East	South West	West Midlands	Yorkshire and The Humber
ONS mean	16.74	15.87	23.78	15.69	16.36	17.88	16.36	16.34	15.76
ONS median	13.28	12.65	18.30	12.40	12.90	14.33	12.74	12.92	12.46
Mod mean	16.67	15.29	19.39	15.05	15.22	17.34	15.92	15.47	14.41
Mod median	13.69	12.79	16.25	12.42	12.44	14.84	13.35	12.64	12.44

The pearson correlations for mean and median between the modelled and raw values are 0.92 and and 0.93.

Hourly gross salary per one-digit SOC

1d SOC	1	2	3	4	5	6	7	8	9
ONS mean	26.77	23.38	18.29	13.42	13.35	10.87	10.94	12.23	10.77
ONS median	20.96	21.34	15.66	11.54	12.04	10.08	9.52	10.93	9.22
Mod mean	21.52	22.14	16.00	12.76	12.55	10.49	10.50	12.05	9.87
Mod median	17.22	20.66	14.12	11.46	11.34	9.71	9.59	10.82	9.12

1. Managers, directors and senior officials
2. Professional occupations
3. Associate professional and technical occupations
4. Administrative and secretarial occupations
5. Skilled trades occupations
6. Caring, leisure and other service occupations
7. Sales and customer service occupations
8. Process, plant and machine operatives
9. Elementary occupations.

The pearson correlations for mean and median between the modelled and raw values are 0.98 and 0.98.

Hourly gross salary per age

The reference for this table is: Table 6.5a Hourly pay - Gross 2020

Table before weighting by age:

Age	16-17	18-21	22-29	30-39	40-49	50-59	60+
ONS mean	7.21	9.59	14.09	18.13	20.04	19.12	16.32
ONS median	6.36	9.00	12.26	15.08	15.89	14.39	12.17
Mod mean	12.77	14.96	16.33	16.93	16.83	16.66	16.29
Mod median	10.93	12.71	13.88	14.02	13.96	13.85	13.65

The pearson correlations for mean and median between the modelled and raw values are 0.92 and 0.92.

Table after weighting by age:

Age	16-17	18-21	22-29	30-39	40-49	50-59	60+
ONS mean	7.21	9.59	14.09	18.13	20.04	19.12	16.32
ONS median	6.36	9.00	12.26	15.08	15.89	14.39	12.17
Mod mean	9.05	11.15	14.87	17.35	17.96	17.47	15.41
Mod median	8.20	9.51	12.86	14.41	14.78	14.43	12.56

The pearson correlations for mean and median between the modelled and raw values are 0.99 and 0.99.

9.2 BMI data

Body Max Index (BMI) is calculated for each individual from the Health Survey for England 2019 (access needs to be requested to the UK Data Service). This calculation is completely indepedent from the PSM to the HSE 2017, and therefore the new BMI values will not fit within the categories indicated by this earlier PSM. As the BMI variable is not necessarily independent from the other health variables (diabetes etc.), the new variable should only be used for studies where all other variables are considered equal. The new variable is continuous (a float).

According to the HSE 2019, the distribution of BMI values should follow figure 1. Socio-economic category was discarded for the modelling as it is not independent from the other variables and “mixed” and “other” ethnicities have been merged due to small sample sizes.

Figure 1. BMI per age. Columns represent ethnicity (White, Black, Asian, Other), and the rows sex (female, male).

The distribution for each age group is a gamma distribution. See figure 2.

Figure 2. Distribution of BMI values for white females aged 30-34.

Due to small sample sizes, the BMI is calculated for each individual depending on their age according to a gamma distribution whose mean is the mean for the corrresponding age, sex and ethnicity (thick line in figure 1), but whose variance is only determined by the total variance by sex and ethnicity. The resulting BMI where validated for Bedfordshire, and correlations of 0.93 and 0.97 were found between the mean and variance of the modelled data compared to those for the reference HSE 2019 data. See figure 3. The distribution per age, as in figure 1, were also validated.

Figure 3. Modelled mean and variance compared to the reference mean and variance from the HSE 2019 data for each of the eight categories of figure 1.

The R codes for this modelling are here.

10 Data sources

The data is sorted around the 2011 Middle-layer Super Output Area (MSOA) geographical unit. These units where created for census collection and are designed to be relatively homogeneous, with an average population size of 8000. Any list of MSOAs in England can be run, with the exception of the MSOAs forming the City of London (i.e the London borough called the City, not London as a whole).

The data from Open Street Map (OSM) is downloaded directly from https://www.openstreetmap.org. Everything else is hosted as local copies on one Azure repository that interacts automatically with the model, and divided into utilities, county level data and national data.

`lookUp.csv`

The look-up table links different geographies together. It is used internally by the model, but can also help the user define their own study area. `MSOA11CD`, `MSOA11NM`, `LAD20CD`, `LAD20NM`, `ITL321CD`, `ITL321NM`, `ITL221CD`, `ITL221NM`, `ITL121CD`, `ITL121NM` are all standard denominations fully compatible with ONS fields of the same name. They are based on ONS lookups. See ONS documentation for more details. `CTY20NM` and `CCTY20NM` are custom denominations for the counties of England (used to sort the county level population data) and the ceremonial counties of England respectively. Their spelling may vary in different data sources and the field `CTY20NM` is not compatible with the ONS field of the same name (which excludes all counties that are also unitary authorities). `GoogleMob` and `OSM` are different spellings for the counties of England used by Google and OSM for their data releases.

10.1 County level data

Contains 47 files, each representing the population in 2020 of one of the counties of England mentioned above, and named

`pop_<county_name>.gz`

This data is based on the 2011 UK census, the Time Use Survey 2014-15 and the Health Survey for England 2017. The SPENSER (Synthetic Population Estimation and Scenario Projection) microsimulation model (reference) distributes a synthetic population based on the census at MSOA scale and projects it to 2020 according to estimates from the Office for National Statistics (ONS). This information was enriched with some of the content of the other two datasets through propensity score matching (PSM) by Prof. Karyn Morrissey (Technical University of Denmark). The rest of the datasets can be added *a posteriori* from the identifiers provided.

The fields currently contained are:

- `idp`: a unique global individual identifier across all counties
- `MSOA11CD`: MSOA code where the individual lives
- `hid`: household identifier, includes communal establishments
- `pid`: identifier linking to the 2011 Census
- `pid_tus`: identifier linking to the Time Use Survey 2015
- `pid_hse`: identifier linking to the Health Survey for England 2017
- `sex`: 0 female; 1 male
- `age`: in years
- `origin`: 1 White; 2 Black; 3 Asian; 4 Mixed; 5 Other
- `nssec5`: National Statistics Socio-economic classification:
  - 1: Higher managerial, administrative and professional occupations
  - 2: Intermediate occupations
  - 3: Small employers and own account workers
  - 4: Lower supervisory and technical occupations
  - 5: Semi-routine and routine occupations
  - 0: Never worked and long-term unemployed
- `soc2010`: Previous version of the Standard Occupational Classification
- `sic1d07`: Standard Industrial Classification of Economic Activities 2007, 1st layer (number corresponding to the letter in alphabetical order)
- `sic2d07`: Standard Industrial Classification of Economic Activities 2007, 2nd layer
- `pwkstat`: Employment status according to the TUS
- Proportion of 24h spent doing different daily activities:
  - `punknown` + `pwork` + `pschool` + `pshop` + `pservices` + `pleisure` + `pescort` + `ptransport` = `pnothome`
  - `phome` + `pworkhome` = `phometot`
  - `pnothome` + `phometot` = 1
- `IncomeX`: hourly (X = "H") and annual (X = "Y") income for employees, see modelling methods for more details
- `cvd`: has a cardio-vascular disease (0 or 1)
- `diabetes`: has diabetes (0 or 1)

- `bloodpressure`: has high blood pressure (0 or 1)
- `BMIvg6`: Body Mass Index:
  - Not applicable
  - Underweight: less than 18.5
  - Normal: 18.5 to less than 25
  - Overweight: 25 to less than 30
  - Obese I: 30 to less than 35
  - Obese II: 35 to less than 40
  - Obese III: 40 or more
- `bmiNew` is a float estimated value for the BMI of the individual. These were drawn directly according to age, sex and ethnicity and are completely independent of the above values (see modelling methods).
- `lng`: longitude of the MSOA11CD centroid
- `lat`: latitude of the MSOA11CD centroid

Some other fields were kept from and for other specific projects but are not from official sources and should generally not be used.

10.2 National data

`businessRegistry.csv`

Contains a breakdown of all business units (i.e. a single workplace) in England at LSOA scale (smaller than MSOA), estimated by the project contributors from two nomis datasets: UK Business Counts - local units by industry and employment size band 2020 and Business Register and Employment Survey 2015. Each item contains the `size` of the unit and its main `sic1d07` code in reference to standard Industrial Classification of Economic Activities 2007 (number corresponding to the letter in alphabetical order). It is used to compute commuting flows.

The R codes to compute this file are here.

`MSOAS_shp.tar.gz`

Is a simple shapefile taken from ONS boundaries.

`QUANT_RAMP.tar.gz`

See: Milton R, Batty M, Dennett A, dedicated RAMP Spatial Interaction Model GitHub repository. It is used to compute the flows towards schools and retail.

`timeAtHomeIncreaseCTY.csv`

This file is a subset from Google COVID-19 Community Mobility Reports, cropped to England. It describes the daily reduction in mobility, averaged at county level, due to lockdown and other COVID-19 restrictions between the 15th of February 2020 and 15th of April 2022. Missing values have been replaced by the national average. These values can be used directly to reduce `pnothome` and increase `phometot` (and their sub-categories) to simulate more accurately the period.

The R codes to process these data are here.

Part III

Advanced

11 Developer guide

The site is built with Quarto. You can iterate on it locally: `cd docs; quarto preview`

11.1 Code hygiene

We use automated tools to format the code.

```
cargo fmt

# Format Markdown docs
prettier --write *.md
prettier --write docs/*.qmd --parser markdown
```

Install prettier for Markdown.

11.2 Some tips for working with Rust

There are two equivalent ways to rebuild and then run the code. First:

```
cargo run --release -- devon
```

The `--` separates arguments to `cargo`, the Rust build tool, and arguments to the program itself. The second way:

```
cargo build --release
./target/release/aspics devon
```

You can build the code in two ways – **debug** and **release**. There's a simple tradeoff – debug mode is fast to build, but slow to run. Release mode is slow to build, but fast to run. For the ASPICS codebase, since the input data is so large and the codebase so small, I'd recommend always using `--release`. If you want to use debug mode, just omit the flag.

If you're working on the Rust code outside of an IDE like VSCode, then you can check if the code compiles much faster by doing `cargo check`.

11.3 Docker

We provide a Dockerfile in case it's helpful for running, but don't recommend using it. If you want to, then assuming you have Docker setup:

```
docker build -t spc .
docker run --mount type=bind,source="$(pwd)"/data,target=/spc/data -t spc /spc/target/rele
```

This will make the `data` directory in your directory available to the Docker image, where it'll download the large input files and produce the final output.

12 Code walkthrough

SPC is implemented in Rust, and its code can be found here. This is an unusual implementation choice in the data science world, so this page has some notes about it.

The code-base makes use of some techniques that may be generally applicable to other projects, independent of the language chosen.

12.0.1 Split code into two stages

Agent-based models and spatial interaction models require some kind of input. Often the effort to transform external data into this input can exceed that of the simulation component. Cleanly separating the two problems has some advantages:

- iterate on the simulation faster, without processing raw data every run
- reuse the prepared input for future projects
- force thinking about the data model needed by the simulation, and transform the external data into that form

SPC is exactly this first stage, originally split from ASPICS when further uses of the same population data were identified.

12.0.2 Explicit data schema

Dynamically typed languages like Python don't force you to explicitly list the shape of input data. It's common to read CSV files with `pandas`, filter and transform the data, and use that throughout the program. This can be quick to start prototyping, but is hard to maintain longer-term. Investing in the process of writing down types:

- makes it easier for somebody new to understand your system – they can first focus on **what** you're modeling, instead of how that's built up from raw data sources
- clarifies what data actually matters to your system; you don't carry forward unnecessary input
- makes it impossible to express invalid states

- One example is here – per person and activity, there's a list of venues the person may visit, along with a probability of going there. If the list of venues and list of probabilities are stored as separate lists or columns, then their length may not match.

- reuse the prepared input for future projects

There's a variety of techniques for expressing strongly typed data:

- protocol buffers or flatbuffers
- JSON schemas
- Python data classes and optional type hints
- statically typed languages like Rust

12.0.3 Type-safe IDs

Say your data model has many different objects, each with their own ID – people, households, venues, etc. You might store these in a list and use the index as an ID. This is fine, but nothing stops you from confusing IDs and accidentally passing in venue 5 to a function instead of household 5. In Rust, it's easy to create "wrapper types" like this and let the compiler prevent these mistakes.

This technique is also useful when preparing external data. GTFS data describing public transit routes and timetables contains many string IDs – shapes, trips, stops, routes. As soon as you read the raw input, you can store the strings in more precise types that prevent mixing up a stop ID and route ID.

12.0.4 Idempotent data preparation

If you're iterating on your initialisation pipeline's code, you probably don't want to download a 2GB external file every single run. A common approach is to first test if a file exists and don't download it again if so. In practice, you may also need to handle unzipping files, showing a progress bar while downloading, and printing clear error messages. This codebase has some common code for doing this in Rust. We intend to publish a separate library to more easily call in your own code.

12.0.5 Logging with structure

It's typical to print information as a complex pipeline runs, for the user to track progress and debug problems. But without any sort of organization, it's hard to follow what steps take a long time or encounter problems. What if your logs could show the logical structure of your pipeline and help you understand where time is spent?

```
[192.30s] [get_info_per_msoa] Loading buildings from data/raw_data/countydata/OSM/west-yorkshire-latest-free/
[192.64s] [get_info_per_msoa] Found 474,207 buildings from data/raw_data/countydata/OSM/west-yorkshire-latest-free/gis_
osm_buildings_a_free_1.shp
[192.70s] [get_info_per_msoa] Matching 474,207 points to 299 polygons. Building R-Tree...
[194.22s] [calculate_lockdown_per_day] Calculating per-day lockdown values
[194.24s] [load_events] Loading events data
[194.25s] [initialisation] By the end, Memory usage: 1.53GiB
[200.89s] [Writing snapshot] Merging flows for all activities

  212.24s          initialisation WestYorkshireLarge
    31.18ms          grab_raw_data
    192.04s          creating population
      8.20s            read_individual_time_use_and_health_data
        4.35s            Reading "data/raw_data/countydata/tus_hse_west-yorkshire.csv"
        3.83s            Creating households
      152.38s          create_commuting_flows
      8.30s            setup_venue_flows Retail
        6.59s            Copying flows to people Retail
      7.47s            setup_venue_flows Nightclub
        6.63s            Copying flows to people Nightclub
      8.68s            setup_venue_flows PrimarySchool
        6.58s            Copying flows to people PrimarySchool
      7.00s            setup_venue_flows SecondarySchool
        6.50s            Copying flows to people SecondarySchool
    2.03s            get_info_per_msoa
    24.48ms          calculate_lockdown_per_day
    251.20µs         load_events "model_parameters/eventDataConcerts.csv"
    1.07s            Writing population to "data/processed_data/WestYorkshireLarge/rust_cache.bin"
    16.93s           Writing snapshot
```

The screenshot above shows a summary printed at the end of a long pipeline run. It's immediately obvious that the slowest step is creating commuting flows.

This codebase uses the tracing framework for logging, with a custom piece to draw the tree. (We'll publish this as a separate library once it's more polished.) The tracing framework is hard to understand, but the main conceptual leap over regular logging framworks is the concept of a **span**. When your code starts one logical step, you call a method to create a new span, and when it finishes, you close that span. Spans can be nested in any way – `create_commuting_flows` happens within the larger step of `creating population`.

12.0.6 Determinism

Given the same inputs, your code should always produce identical output, no matter where it's run or how many times. Otherwise, debugging problems becomes very tedious, and it's more difficult to make conclusions from results. Of course, many projects have a stochastic element – but this should be controlled by a random number generator (RNG) seed, which is part of the input. You vary the seed and repeat the program, then reason about the distribution of results.

Aside from organizing your code to let a single RNG seed influence everything, another possible source of non-determinism is iteration order. In Rust, a `HashMap` could have different order every time it's used, so we use a `BTreeMap` instead when this matters. In Python, dictionaries are ordered. Be sure to check for your language.

12.1 Protocol buffers

SPC uses protocol buffers v2 for output. This has some advantages explained the "explicit data schema" section above.

Note that we chose proto2 instead of proto3, because proto3 doesn't support required fields. This is done to allow schemas to evolve better over time, but this isn't a feature SPC makes use of. There's no need to have new code work with old data, or vice versa – if the schema is updated, downstream code should adapt accordingly and use the updated input files.

Note also that protocol buffers don't easily support type-safe wrappers around numeric IDs, so downstream code has to be careful not to mix up household, venue, and person IDs. For this reason, SPC internally doesn't use the auto-generated protobuf code until the very end of the pipeline. It's always possible to be more precise with native Rust types, and convert to the less strict types later.

12.2 An example of the power of static type checking

Imagine we want to add a new activity type to represent people going to university and higher education. SPC already has activities for primary and secondary school, so we'll probably want to follow those as a guide. In any language, we could search the codebase for relevant terms to get a sense of what to update. In languages like Python without an up-front compilation step, if we fail to update something or write blatantly incorrect code (such as making a typo in variable names or passing a list where a string was expected), we only find out when that code happens to run. In pipelines with many steps and large input files, it could be a while before we reach the problematic code.

Let's walk through the same exercise for SPC's Rust code. We start by adding a new `University` case to the Activity enum. If we try to compile the code here (with `cargo check` or an IDE), we immediately get 4 errors.

```
error[E0004]: non-exhaustive patterns: `University` not covered
  --> src/init/quant.rs:38:44
   |
38 |         let (population_csv, prob_sij) = match activity {
   |                                                ^^^^^^^^ pattern `University` not covered
   |
  ::: src/lib.rs:129:1
   |
129 | / pub enum Activity {
130 | |     Retail,
131 | |     PrimarySchool,
132 | |     SecondarySchool,
... |
135 | |     University,
    | |     ---------- not covered
136 | | }
    | |_- `Activity` defined here
    |
    = help: ensure that all possible cases are being handled, possibly by adding wildcards or more
match arms
    = note: the matched value is of type `Activity`
```

Three of the errors are in the QUANT module. The first is here. It's immediately clear that for retail and primary/secondary school, we read in two files from QUANT representing venues where these activities take place and the probability of going to each venue. Even if we were unfamiliar with this codebase, the compiler has told us one thing we'll need to figure out, and where to wire it up.

```
error[E0004]: non-exhaustive patterns: `University` not covered
   --> src/protobuf.rs:135:11
    |
135 |     match activity {
    |           ^^^^^^^^ pattern `University` not covered
    |
   ::: src/lib.rs:129:1
    |
129 | / pub enum Activity {
130 | |     Retail,
131 | |     PrimarySchool,
132 | |     SecondarySchool,
... |
135 | |     University,
    | |     ---------- not covered
136 | | }
    | |_- `Activity` defined here
    |
    = help: ensure that all possible cases are being handled, possibly by adding wildcards or more
match arms
    = note: the matched value is of type `Activity`
```

The other error is in the code that writes the protobuf output. Similarly, we need a way to represent university activities in the protobuf scheme.

Extending an unfamiliar code-base backed by compiler errors is a very guided experience. If you wanted to add more demographic attributes to people or energy use information to households, you don't need to guess all of the places in the code you'll need to update. You can just add the field, then let the compiler tell you all places where those objects get created.

13 Performance

The following tables summarizes the resources SPC needs to run in different areas.

year	study_area	num_msoas	num_households	num_people	pb_file_size	runtime	commuting_runtime	memory_usage
2012	England/bedfordshire	74	245,166	647,272	256.83 MiB	10 seconds	3 seconds	849.02 MiB
2020	England/bedfordshire	74	272,875	674,044	271.65 MiB	9 seconds	3 seconds	922.88 MiB
2022	England/bedfordshire	74	309,706	703,582	277.74 MiB	9 seconds	3 seconds	929.80 MiB
2032	England/bedfordshire	74	309,706	703,582	277.74 MiB	9 seconds	3 seconds	929.80 MiB
2039	England/bedfordshire	74	329,061	715,797	278.39 MiB	11 seconds	3 seconds	927.77 MiB
2012	England/berkshire	107	342,167	890,543	356.04 MiB	14 seconds	7 seconds	1.06 GiB
2020	England/berkshire	107	365,905	918,258	373.35 MiB	14 seconds	7 seconds	1.10 GiB
2022	England/berkshire	107	394,446	941,655	368.37 MiB	14 seconds	7 seconds	1.08 GiB
2032	England/berkshire	107	394,446	941,655	368.37 MiB	14 seconds	7 seconds	1.08 GiB
2039	England/berkshire	107	408,604	949,986	367.21 MiB	14 seconds	7 seconds	1.08 GiB
2012	England/bristol	55	182,299	448,233	173.74 MiB	6 seconds	2 seconds	527.23 MiB
2020	England/bristol	55	196,940	470,039	183.99 MiB	6 seconds	2 seconds	547.49 MiB

year	study_area	num_msoas	num_households	num_people	pp_file_size	runtime	commuting_runtime	memory_usage
2022	England/bristol	55	216,197	503,014	192.51 MiB	7 seconds	2 seconds	559.78 MiB
2032	England/bristol	55	216,197	503,014	192.51 MiB	7 seconds	2 seconds	559.78 MiB
2039	England/bristol	55	227,770	521,371	199.72 MiB	7 seconds	2 seconds	573.40 MiB
2012	England/buckinghamshire	99	99,235	261,340	108.30 MiB	5 seconds	1 second	310.38 MiB
2020	England/buckinghamshire	99	108,999	271,050	114.31 MiB	5 seconds	1 second	400.87 MiB
2022	England/buckinghamshire	99	123,578	278,548	112.39 MiB	5 seconds	1 second	393.64 MiB
2032	England/buckinghamshire	99	123,578	278,548	112.39 MiB	5 seconds	1 second	393.64 MiB
2039	England/buckinghamshire	99	130,393	281,773	112.14 MiB	5 seconds	1 second	391.52 MiB
2012	England/cambridgeshire	98	327,257	832,980	323.35 MiB	11 seconds	5 seconds	1013.16 MiB
2020	England/cambridgeshire	98	348,522	863,250	341.16 MiB	12 seconds	5 seconds	1.03 GiB
2022	England/cambridgeshire	98	377,634	907,166	348.75 MiB	12 seconds	5 seconds	1.03 GiB
2032	England/cambridgeshire	98	377,634	907,166	348.75 MiB	12 seconds	5 seconds	1.03 GiB
2039	England/cambridgeshire	98	392,478	924,170	351.39 MiB	12 seconds	5 seconds	1.04 GiB
2012	England/cheshire	139	441,084	1,042,065	402.14 MiB	16 seconds	7 seconds	1.13 GiB
2020	England/cheshire	139	464,134	1,070,597	416.35 MiB	16 seconds	7 seconds	1.46 GiB
2022	England/cheshire	139	489,476	1,125,198	425.27 MiB	16 seconds	7 seconds	1.47 GiB

year	study_area	num_msoas	num_households	num_people	pbf_file_size	runtime	commuting_runtime	memory_usage
2032	England/cheshire	139	489,476	1,125,198	425.27 MiB	16 seconds	7 seconds	1.47 GiB
2039	England/cheshire	139	501,501	1,149,515	431.10 MiB	16 seconds	7 seconds	1.48 GiB
2012	England/cornwall	74	232,659	549,616	208.07 MiB	8 seconds	2 seconds	742.92 MiB
2020	England/cornwall	74	247,105	577,414	219.74 MiB	8 seconds	3 seconds	764.95 MiB
2022	England/cornwall	74	270,134	634,940	233.36 MiB	8 seconds	3 seconds	828.45 MiB
2032	England/cornwall	74	270,134	634,940	233.36 MiB	8 seconds	3 seconds	828.45 MiB
2039	England/cornwall	74	280,546	658,610	239.72 MiB	8 seconds	3 seconds	838.14 MiB
2012	England/cumbria	64	222,586	498,624	188.03 MiB	7 seconds	2 seconds	547.33 MiB
2020	England/cumbria	64	226,893	499,873	188.73 MiB	7 seconds	2 seconds	548.52 MiB
2022	England/cumbria	64	230,206	499,840	183.18 MiB	7 seconds	2 seconds	533.99 MiB
2032	England/cumbria	64	230,206	499,840	183.18 MiB	7 seconds	2 seconds	533.99 MiB
2039	England/cumbria	64	231,202	498,475	181.58 MiB	7 seconds	2 seconds	530.96 MiB
2012	England/derbyshire	131	436,276	1,035,356	397.75 MiB	15 seconds	7 seconds	1.12 GiB
2020	England/derbyshire	131	459,743	1,064,406	409.76 MiB	16 seconds	8 seconds	1.44 GiB
2022	England/derbyshire	131	489,764	1,122,078	419.52 MiB	16 seconds	7 seconds	1.45 GiB
2032	England/derbyshire	131	489,764	1,122,078	419.52 MiB	16 seconds	7 seconds	1.45 GiB
2039	England/derbyshire	131	505,314	1,152,518	429.01 MiB	16 seconds	8 seconds	1.47 GiB

year	study_area	num_msoas	num_households	num_people	pp_file_size	runtime	commuting_runtime	memory_usage
2012	England/devon	156	494,106	1,165,952	438.60 MiB	18 seconds	8 seconds	1.49 GiB
2020	England/devon	156	523,033	1,212,387	459.44 MiB	18 seconds	8 seconds	1.53 GiB
2022	England/devon	156	567,011	1,304,874	478.71 MiB	19 seconds	9 seconds	1.64 GiB
2032	England/devon	156	567,011	1,304,874	478.71 MiB	19 seconds	9 seconds	1.64 GiB
2039	England/devon	156	589,178	1,342,775	488.23 MiB	19 seconds	9 seconds	1.66 GiB
2012	England/durham	117	390,472	911,601	349.78 MiB	12 seconds	5 seconds	1.03 GiB
2020	England/durham	117	407,828	930,184	359.59 MiB	12 seconds	5 seconds	1.05 GiB
2022	England/durham	117	425,611	952,801	356.63 MiB	12 seconds	5 seconds	1.04 GiB
2032	England/durham	117	425,611	952,801	356.63 MiB	12 seconds	5 seconds	1.04 GiB
2039	England/durham	117	434,593	959,555	357.66 MiB	12 seconds	5 seconds	1.04 GiB
2012	England/east-sussex	102	355,257	827,703	313.71 MiB	12 seconds	5 seconds	987.31 MiB
2020	England/east-sussex	102	380,894	853,970	324.02 MiB	12 seconds	6 seconds	1006.13 MiB
2022	England/east-sussex	102	423,181	895,907	329.56 MiB	12 seconds	5 seconds	1008.58 MiB

year	study_area	num_msoas	num_households	num_people	pop_file_size	runtime	commuting_runtime	memory_usage
2032	England/east-sussex	102	423,181	895,907	329.56 MiB	12 seconds	5 seconds	1008.58 MiB
2039	England/east-sussex	102	446,000	915,014	335.45 MiB	12 seconds	5 seconds	1020.75 MiB
2012	England/east-yorkshire-with-hull	75	255,848	593,271	227.49 MiB	8 seconds	3 seconds	778.75 MiB
2020	England/east-yorkshire-with-hull	75	262,609	602,286	233.14 MiB	8 seconds	3 seconds	835.04 MiB
2022	England/east-yorkshire-with-hull	75	272,805	613,721	230.34 MiB	8 seconds	3 seconds	824.50 MiB
2032	England/east-yorkshire-with-hull	75	272,805	613,721	230.34 MiB	8 seconds	3 seconds	824.50 MiB
2039	England/east-yorkshire-with-hull	75	277,770	617,357	230.45 MiB	8 seconds	3 seconds	825.00 MiB
2012	England/essex	211	722,974	1,786,316	690.77 MiB	30 seconds	19 seconds	2.06 GiB
2020	England/essex	211	773,454	1,857,205	726.02 MiB	32 seconds	20 seconds	2.13 GiB
2022	England/essex	211	858,552	1,981,994	761.40 MiB	33 seconds	20 seconds	2.19 GiB
2032	England/essex	211	858,552	1,981,994	761.40 MiB	33 seconds	20 seconds	2.19 GiB
2039	England/essex	211	906,640	2,042,404	777.71 MiB	33 seconds	21 seconds	2.21 GiB
2012	England/gloucestershire	107	365,240	889,836	344.16 MiB	13 seconds	5 seconds	1.02 GiB
2020	England/gloucestershire	107	392,643	933,909	362.90 MiB	13 seconds	6 seconds	1.06 GiB
2022	England/gloucestershire	107	432,216	1,025,077	389.56 MiB	14 seconds	6 seconds	1.10 GiB

year	study_area	num_msoas	num_households	num_people	pp_file_size	runtime	commuting_runtime	memory_usage
2032	England/gloucestershire	107	432,216	1,025,077	389.56 MiB	14 seconds	6 seconds	1.10 GiB
2039	England/gloucestershire	107	453,383	1,068,484	403.87 MiB	14 seconds	6 seconds	1.43 GiB
2012	England/greater-london	983	3,283,305	8,581,245	3.27 GiB	12 minutes	11 minutes	11.80 GiB
2020	England/greater-london	983	3,574,266	8,983,777	3.48 GiB	12 minutes	11 minutes	12.22 GiB
2022	England/greater-london	983	3,997,548	9,452,049	3.55 GiB	12 minutes	11 minutes	12.25 GiB
2032	England/greater-london	983	3,997,548	9,452,049	3.55 GiB	12 minutes	11 minutes	12.25 GiB
2039	England/greater-london	983	4,229,017	9,688,506	3.58 GiB	12 minutes	11 minutes	12.95 GiB
2012	England/greater-manchester	346	1,128,371	2,745,455	1.05 GiB	70 seconds	53 seconds	3.56 GiB
2020	England/greater-manchester	346	1,192,547	2,840,431	1.10 GiB	71 seconds	54 seconds	3.66 GiB
2022	England/greater-manchester	346	1,272,689	2,974,954	1.13 GiB	73 seconds	55 seconds	3.69 GiB
2032	England/greater-manchester	346	1,272,689	2,974,954	1.13 GiB	74 seconds	56 seconds	3.69 GiB
2039	England/greater-manchester	346	1,319,090	3,049,727	1.15 GiB	76 seconds	58 seconds	3.73 GiB
2012	England/hampshire	225	733,611	1,810,518	698.03 MiB	32 seconds	20 seconds	2.07 GiB

year	study_area	num_msoas	num_households	num_people	pbf_file_size	runtime	commuting_runtime	memory_usage
2020	England/hampshire	225	777,116	1,861,250	721.62 MiB	32 seconds	20 seconds	2.12 GiB
2022	England/hampshire	225	836,451	1,931,669	728.97 MiB	32 seconds	20 seconds	2.12 GiB
2032	England/hampshire	225	836,451	1,931,669	728.97 MiB	33 seconds	20 seconds	2.12 GiB
2039	England/hampshire	225	867,417	1,960,190	735.50 MiB	33 seconds	20 seconds	2.13 GiB
2012	England/herefordshire	23	79,083	188,362	72.21 MiB	4 seconds	1 second	234.89 MiB
2020	England/herefordshire	23	83,238	195,194	74.71 MiB	4 seconds	1 second	239.36 MiB
2022	England/herefordshire	23	89,574	209,784	77.63 MiB	4 seconds	1 second	242.83 MiB
2032	England/herefordshire	23	89,574	209,784	77.63 MiB	4 seconds	1 second	242.83 MiB
2039	England/herefordshire	23	92,605	216,508	79.43 MiB	4 seconds	1 second	245.69 MiB
2012	England/hertfordshire	153	457,276	1,160,155	458.65 MiB	19 seconds	11 seconds	1.56 GiB
2020	England/hertfordshire	153	494,661	1,190,043	477.18 MiB	19 seconds	11 seconds	1.59 GiB
2022	England/hertfordshire	153	546,573	1,219,124	476.55 MiB	19 seconds	11 seconds	1.67 GiB
2032	England/hertfordshire	153	546,573	1,219,124	476.55 MiB	19 seconds	11 seconds	1.67 GiB
2039	England/hertfordshire	153	575,179	1,233,573	476.97 MiB	19 seconds	11 seconds	1.67 GiB
2012	England/isle-of-wight	18	61,636	139,732	53.88 MiB	3 seconds	1 second	188.79 MiB
2020	England/isle-of-wight	18	65,140	143,268	54.99 MiB	3 seconds	1 second	190.45 MiB

year	study_area	num_msoas	num_households	num_people	pb_file_size	runtime	commuting_runtime	memory_usage
2022	England/isle-of-wight	18	70,496	151,582	55.55 MiB	3 seconds	1 second	200.99 MiB
2032	England/isle-of-wight	18	70,496	151,582	55.55 MiB	3 seconds	1 second	200.99 MiB
2039	England/isle-of-wight	18	72,968	154,841	56.14 MiB	3 seconds	1 second	202.13 MiB
2012	England/kent	220	718,544	1,793,702	700.10 MiB	29 seconds	17 seconds	2.08 GiB
2020	England/kent	220	781,933	1,873,451	737.20 MiB	30 seconds	18 seconds	2.15 GiB
2022	England/kent	220	875,515	2,008,857	773.24 MiB	31 seconds	19 seconds	2.21 GiB
2032	England/kent	220	875,515	2,008,857	773.24 MiB	32 seconds	19 seconds	2.21 GiB
2039	England/kent	220	926,571	2,069,087	788.47 MiB	35 seconds	19 seconds	2.23 GiB
2012	England/lancashire	191	619,861	1,476,469	571.94 MiB	24 seconds	14 seconds	1.83 GiB
2020	England/lancashire	191	640,196	1,511,896	589.78 MiB	24 seconds	14 seconds	1.87 GiB
2022	England/lancashire	191	663,637	1,567,396	594.49 MiB	24 seconds	14 seconds	1.87 GiB
2032	England/lancashire	191	663,637	1,567,396	594.49 MiB	24 seconds	14 seconds	1.87 GiB
2039	England/lancashire	191	674,387	1,591,908	600.02 MiB	25 seconds	14 seconds	1.88 GiB
2012	England/leicestershire	120	370,305	958,470	373.02 MiB	14 seconds	7 seconds	1.08 GiB

year	study_area	num_msoas	num_households	num_people	pop_file_size	runtime	commuting_runtime	memory_usage
2020	England/leicestershire	120	397,467	1,016,632	397.28 MiB	14 seconds	7 seconds	1.13 GiB
2022	England/leicestershire	120	438,413	1,118,737	426.12 MiB	15 seconds	8 seconds	1.48 GiB
2032	England/leicestershire	120	438,413	1,118,737	426.12 MiB	15 seconds	8 seconds	1.48 GiB
2039	England/leicestershire	120	459,655	1,164,678	440.87 MiB	16 seconds	8 seconds	1.50 GiB
2012	England/lincolnshire	134	449,394	1,064,403	403.05 MiB	15 seconds	7 seconds	1.43 GiB
2020	England/lincolnshire	134	475,646	1,098,403	419.31 MiB	15 seconds	7 seconds	1.46 GiB
2022	England/lincolnshire	134	507,295	1,152,299	427.55 MiB	15 seconds	7 seconds	1.47 GiB
2032	England/lincolnshire	134	507,295	1,152,299	427.55 MiB	16 seconds	7 seconds	1.47 GiB
2039	England/lincolnshire	134	523,548	1,172,923	430.83 MiB	16 seconds	7 seconds	1.47 GiB
2012	England/merseyside	184	603,483	1,399,209	533.96 MiB	20 seconds	11 seconds	1.75 GiB
2020	England/merseyside	184	632,617	1,435,755	553.33 MiB	20 seconds	11 seconds	1.79 GiB
2022	England/merseyside	184	665,766	1,498,518	570.21 MiB	20 seconds	10 seconds	1.82 GiB
2032	England/merseyside	184	665,766	1,498,518	570.21 MiB	21 seconds	11 seconds	1.82 GiB

year	study_area	num_msoas	num_households	num_people	pbf_file_size	runtime	commuting_runtime	memory_usage
2039	England/merseyside	184	685,165	1,528,037	577.48 MiB	21 seconds	11 seconds	1.83 GiB
2012	England/norfolk	110	374,491	882,793	333.07 MiB	12 seconds	5 seconds	1017.16 MiB
2020	England/norfolk	110	397,770	916,799	348.41 MiB	12 seconds	5 seconds	1.02 GiB
2022	England/norfolk	110	432,187	982,755	362.27 MiB	13 seconds	5 seconds	1.04 GiB
2032	England/norfolk	110	432,187	982,755	362.27 MiB	13 seconds	5 seconds	1.04 GiB
2039	England/norfolk	110	450,068	1,013,214	371.39 MiB	13 seconds	5 seconds	1.06 GiB
2012	England/northamptonshire	91	289,575	720,263	284.39 MiB	10 seconds	4 seconds	941.33 MiB
2020	England/northamptonshire	91	316,553	762,382	304.36 MiB	10 seconds	4 seconds	981.14 MiB
2022	England/northamptonshire	91	352,529	828,003	320.81 MiB	11 seconds	5 seconds	1005.64 MiB
2032	England/northamptonshire	91	352,529	828,003	320.81 MiB	11 seconds	5 seconds	1005.64 MiB
2039	England/northamptonshire	91	370,555	855,812	328.03 MiB	11 seconds	5 seconds	1016.86 MiB
2012	England/northumberland	40	138,928	315,894	120.66 MiB	5 seconds	1 second	423.11 MiB
2020	England/northumberland	40	143,516	322,616	121.94 MiB	5 seconds	1 second	423.87 MiB
2022	England/northumberland	40	148,792	333,456	122.06 MiB	5 seconds	1 second	421.48 MiB
2032	England/northumberland	40	148,792	333,456	122.06 MiB	5 seconds	1 second	421.48 MiB

year	study_area	num_msoas	num_households	num_people	pbf_file_size	runtime	commuting_runtime	memory_usage
2039	England/northumberland	40	150,259	337,186	122.24 MiB	5 seconds	1 second	421.48 MiB
2012	England/north-yorkshire	138	460,050	1,085,067	413.05 MiB	16 seconds	7 seconds	1.45 GiB
2020	England/north-yorkshire	138	478,639	1,107,928	423.18 MiB	16 seconds	7 seconds	1.47 GiB
2022	England/north-yorkshire	138	499,392	1,134,723	420.60 MiB	16 seconds	7 seconds	1.45 GiB
2032	England/north-yorkshire	138	499,392	1,134,723	420.60 MiB	16 seconds	7 seconds	1.45 GiB
2039	England/north-yorkshire	138	509,099	1,143,895	421.52 MiB	16 seconds	7 seconds	1.46 GiB
2012	England/nottinghamshire	138	460,022	1,123,005	432.35 MiB	16 seconds	8 seconds	1.49 GiB
2020	England/nottinghamshire	138	486,163	1,169,489	453.68 MiB	16 seconds	8 seconds	1.53 GiB
2022	England/nottinghamshire	138	522,944	1,248,804	473.35 MiB	17 seconds	8 seconds	1.56 GiB
2032	England/nottinghamshire	138	522,944	1,248,804	473.35 MiB	17 seconds	8 seconds	1.56 GiB
2039	England/nottinghamshire	138	543,291	1,281,812	482.21 MiB	18 seconds	9 seconds	1.66 GiB
2012	England/oxfordshire	86	261,235	671,997	260.43 MiB	9 seconds	3 seconds	852.84 MiB
2020	England/oxfordshire	86	274,908	695,490	271.62 MiB	10 seconds	4 seconds	918.90 MiB
2022	England/oxfordshire	86	293,368	729,866	275.40 MiB	10 seconds	4 seconds	919.34 MiB

year	study_area	num_msoas	num_households	num_people	pop_file_size	runtime	commuting_runtime	memory_usage
2032	England/oxfordshire	86	293,368	729,866	275.40 MiB	10 seconds	4 seconds	919.34 MiB
2039	England/oxfordshire	86	303,035	743,227	277.51 MiB	10 seconds	4 seconds	922.19 MiB
2012	England/rutland	5	14,912	38,314	16.37 MiB	3 seconds	1 second	54.07 MiB
2020	England/rutland	5	16,698	40,381	17.09 MiB	3 seconds	1 second	57.95 MiB
2022	England/rutland	5	18,198	44,193	18.26 MiB	3 seconds	1 second	60.08 MiB
2032	England/rutland	5	18,198	44,193	18.26 MiB	3 seconds	1 second	60.08 MiB
2039	England/rutland	5	18,914	45,659	18.71 MiB	3 seconds	1 second	61.20 MiB
2012	England/shropshire	62	197,768	483,414	186.29 MiB	7 seconds	2 seconds	550.97 MiB
2020	England/shropshire	62	211,035	508,233	195.76 MiB	7 seconds	2 seconds	568.62 MiB
2022	England/shropshire	62	228,285	558,755	207.29 MiB	7 seconds	2 seconds	740.58 MiB
2032	England/shropshire	62	228,285	558,755	207.29 MiB	7 seconds	2 seconds	740.58 MiB
2039	England/shropshire	62	236,015	581,476	213.23 MiB	7 seconds	2 seconds	749.82 MiB
2012	England/somerset	124	329,040	790,346	303.62 MiB	11 seconds	4 seconds	970.03 MiB
2020	England/somerset	124	353,976	822,271	317.43 MiB	11 seconds	4 seconds	996.71 MiB
2022	England/somerset	124	388,675	880,441	331.61 MiB	12 seconds	4 seconds	1018.53 MiB
2032	England/somerset	124	388,675	880,441	331.61 MiB	12 seconds	4 seconds	1018.53 MiB
2039	England/somerset	124	406,157	906,545	339.87 MiB	12 seconds	4 seconds	1.01 GiB

year	study_area	num_msoas	num_households	num_people	pbf_file_size	runtime	commuting_runtime	memory_usage
2012	England/south-yorkshire	172	566,664	1,372,435	528.11 MiB	20 seconds	11 seconds	1.75 GiB
2020	England/south-yorkshire	172	597,694	1,418,840	548.59 MiB	21 seconds	11 seconds	1.79 GiB
2032	England/south-yorkshire	172	637,411	1,493,544	563.91 MiB	21 seconds	11 seconds	1.81 GiB
2039	England/south-yorkshire	172	659,843	1,531,313	575.31 MiB	22 seconds	12 seconds	1.83 GiB
2012	England/staffordshire	143	464,441	1,111,144	425.27 MiB	16 seconds	8 seconds	1.47 GiB
2020	England/staffordshire	143	486,645	1,139,752	437.51 MiB	16 seconds	8 seconds	1.49 GiB
2022	England/staffordshire	143	510,634	1,188,857	444.87 MiB	17 seconds	8 seconds	1.50 GiB
2032	England/staffordshire	143	510,634	1,188,857	444.87 MiB	17 seconds	8 seconds	1.50 GiB
2039	England/staffordshire	143	522,882	1,215,006	452.94 MiB	17 seconds	8 seconds	1.52 GiB
2012	England/suffolk	90	136,142	327,349	128.13 MiB	5 seconds	1 second	440.37 MiB
2020	England/suffolk	90	146,277	333,781	130.90 MiB	5 seconds	1 second	445.14 MiB
2022	England/suffolk	90	159,882	344,534	130.76 MiB	5 seconds	1 second	442.14 MiB
2032	England/suffolk	90	159,882	344,534	130.76 MiB	5 seconds	1 second	442.14 MiB
2039	England/suffolk	90	166,718	350,358	132.54 MiB	5 seconds	1 second	446.24 MiB
2012	England/surrey	151	458,108	1,168,112	456.50 MiB	21 seconds	13 seconds	1.55 GiB

year	study_area	num_msoas	num_households	num_people	pbf_file_size	runtime	commuting_runtime	memory_usage
2020	England/surrey	151	480,930	1,195,509	472.89 MiB	21 seconds	13 seconds	1.58 GiB
2022	England/surrey	151	518,720	1,214,557	467.03 MiB	21 seconds	13 seconds	1.56 GiB
2032	England/surrey	151	518,720	1,214,557	467.03 MiB	21 seconds	13 seconds	1.56 GiB
2039	England/surrey	151	538,941	1,221,227	464.71 MiB	21 seconds	13 seconds	1.64 GiB
2012	England/tyne-and-wear	145	483,909	1,119,030	427.35 MiB	15 seconds	7 seconds	1.47 GiB
2020	England/tyne-and-wear	145	501,383	1,143,194	439.09 MiB	15 seconds	7 seconds	1.50 GiB
2022	England/tyne-and-wear	145	521,777	1,168,078	440.03 MiB	15 seconds	7 seconds	1.49 GiB
2032	England/tyne-and-wear	145	521,777	1,168,078	440.03 MiB	14 seconds	6 seconds	1.49 GiB
2039	England/tyne-and-wear	145	532,652	1,177,340	441.36 MiB	15 seconds	7 seconds	1.58 GiB
2012	England/warwickshire	108	361,467	896,673	347.44 MiB	13 seconds	6 seconds	1.03 GiB
2020	England/warwickshire	108	392,639	958,833	373.63 MiB	14 seconds	6 seconds	1.08 GiB
2022	England/warwickshire	108	432,682	1,061,955	405.95 MiB	15 seconds	7 seconds	1.44 GiB
2032	England/warwickshire	108	432,682	1,061,955	405.95 MiB	14 seconds	7 seconds	1.44 GiB

year	study_area	num_msoas	num_households	num_people	file_size	runtime	commuting_runtime	memory_usage
2039	England/warwickshire	108	454,732	1,112,230	424.10 MiB	15 seconds	7 seconds	1.47 GiB
2012	England/west-midlands	314	958,034	2,477,391	990.27 MiB	56 seconds	38 seconds	3.24 GiB
2020	England/west-midlands	314	1,002,273	2,572,395	1.01 GiB	58 seconds	40 seconds	3.33 GiB
2022	England/west-midlands	314	1,046,146	2,664,228	1.04 GiB	60 seconds	41 seconds	3.37 GiB
2032	England/west-midlands	314	1,079,612	2,706,242	1.04 GiB	61 seconds	41 seconds	3.55 GiB
2039	England/west-midlands	314	1,128,890	2,787,990	1.07 GiB	62 seconds	42 seconds	3.59 GiB
2012	England/west-sussex	100	348,766	836,646	321.17 MiB	11 seconds	5 seconds	1004.45 MiB
2020	England/west-sussex	100	375,837	871,029	337.76 MiB	12 seconds	5 seconds	1.01 GiB
2022	England/west-sussex	100	419,347	931,573	350.11 MiB	12 seconds	5 seconds	1.03 GiB
2032	England/west-sussex	100	419,347	931,573	350.11 MiB	12 seconds	5 seconds	1.03 GiB
2039	England/west-sussex	100	442,292	958,567	356.77 MiB	12 seconds	5 seconds	1.04 GiB
2012	England/west-yorkshire	299	921,242	2,271,833	893.87 MiB	46 seconds	31 seconds	3.05 GiB
2020	England/west-yorkshire	299	963,460	2,339,931	930.47 MiB	48 seconds	33 seconds	3.12 GiB

year	study_area	num_msoas	num_households	num_people	pb_file_size	runtime	commuting_runtime	memory_usage
2022	England/west-yorkshire	299	1,021,830	2,434,902	945.77 MiB	48 seconds	33 seconds	3.13 GiB
2032	England/west-yorkshire	299	1,021,830	2,434,902	945.77 MiB	49 seconds	33 seconds	3.13 GiB
2039	England/west-yorkshire	299	1,053,859	2,481,358	957.40 MiB	49 seconds	33 seconds	3.32 GiB
2012	England/wiltshire	89	285,600	704,491	274.58 MiB	9 seconds	3 seconds	921.08 MiB
2020	England/wiltshire	89	309,159	735,088	288.20 MiB	9 seconds	3 seconds	947.43 MiB
2022	England/wiltshire	89	335,400	774,105	292.69 MiB	10 seconds	3 seconds	949.16 MiB
2032	England/wiltshire	89	335,400	774,105	292.69 MiB	10 seconds	3 seconds	949.16 MiB
2039	England/wiltshire	89	348,866	792,075	296.40 MiB	10 seconds	3 seconds	955.08 MiB
2012	England/worcestershire	85	240,958	578,628	221.47 MiB	8 seconds	3 seconds	770.62 MiB
2020	England/worcestershire	85	255,594	601,116	231.59 MiB	8 seconds	3 seconds	790.42 MiB
2022	England/worcestershire	85	274,309	644,922	241.99 MiB	8 seconds	3 seconds	849.84 MiB
2032	England/worcestershire	85	274,309	644,922	241.99 MiB	8 seconds	3 seconds	849.84 MiB
2039	England/worcestershire	85	283,275	666,303	248.38 MiB	8 seconds	3 seconds	861.38 MiB
2012	special/northwest_transpennine	829	2,653,096	6,416,497	2.45 GiB	5 minutes	4 minutes	7.74 GiB
2020	special/northwest_transpennine	829	2,788,624	6,616,117	2.56 GiB	5 minutes	4 minutes	7.95 GiB
2022	special/northwest_transpennine	829	2,960,285	6,908,374	2.62 GiB	5 minutes	4 minutes	8.02 GiB

year	study_area	num_msoas	num_households	num_people	pbf_file_size	runtime	commuting_runtime	memory_usage
2032	special/northwest_transpennine	829	2,960,285	6,908,374	2.62 GiB	5 minutes	5 minutes	8.02 GiB
2039	special/northwest_transpennine	829	3,058,114	7,059,122	2.66 GiB	5 minutes	5 minutes	8.08 GiB
2012	Wales/bridgend-and-neath-port-talbot	38	119,725	283,159	108.21 MiB	5 seconds	1 second	382.24 MiB
2020	Wales/bridgend-and-neath-port-talbot	38	123,909	289,896	111.10 MiB	5 seconds	1 second	387.45 MiB
2022	Wales/bridgend-and-neath-port-talbot	38	124,921	292,227	111.51 MiB	4 seconds	1 second	387.72 MiB
2032	Wales/bridgend-and-neath-port-talbot	38	128,601	301,529	113.58 MiB	5 seconds	1 second	390.82 MiB
2039	Wales/bridgend-and-neath-port-talbot	38	129,740	307,260	114.33 MiB	5 seconds	1 second	391.29 MiB
2012	Wales/cardiff-and-vale-of-glamorgan	63	199,208	484,182	187.17 MiB	6 seconds	2 seconds	558.19 MiB
2020	Wales/cardiff-and-vale-of-glamorgan	63	214,676	499,272	194.70 MiB	7 seconds	2 seconds	572.89 MiB
2022	Wales/cardiff-and-vale-of-glamorgan	63	218,981	502,763	196.11 MiB	6 seconds	2 seconds	576.04 MiB
2032	Wales/cardiff-and-vale-of-glamorgan	63	240,112	522,526	199.42 MiB	7 seconds	2 seconds	577.84 MiB
2039	Wales/cardiff-and-vale-of-glamorgan	63	254,162	531,549	201.82 MiB	7 seconds	2 seconds	737.29 MiB
2012	Wales/central-valleys	38	124,691	296,581	115.15 MiB	5 seconds	1 second	396.20 MiB
2020	Wales/central-valleys	38	130,072	301,907	117.77 MiB	18 seconds	1 second	400.97 MiB
2022	Wales/central-valleys	38	131,383	303,557	118.40 MiB	8 seconds	1 second	424.47 MiB
2032	Wales/central-valleys	38	136,404	310,032	118.04 MiB	5 seconds	1 second	421.13 MiB
2039	Wales/central-valleys	38	138,735	314,703	119.17 MiB	5 seconds	1 second	423.02 MiB
2012	Wales/conwy-and-denbighshire	30	92,732	211,205	80.50 MiB	4 seconds	1 second	251.47 MiB
2020	Wales/conwy-and-denbighshire	30	95,314	213,302	81.56 MiB	4 seconds	1 second	253.63 MiB

year	study_area	num_msoas	num_households	num_people	pp_file_size	runtime	commuting_runtime	memory_usage
2022	Wales/conwy-and-denbighshire	30	95,881	214,182	81.85 MiB	4 seconds	1 second	254.21 MiB
2032	Wales/conwy-and-denbighshire	30	97,683	218,122	81.11 MiB	4 seconds	1 second	251.17 MiB
2039	Wales/conwy-and-denbighshire	30	97,687	220,933	80.92 MiB	4 seconds	1 second	249.76 MiB
2012	Wales/flintshire-and-wrexham	38	122,180	288,696	113.32 MiB	5 seconds	1 second	393.63 MiB
2020	Wales/flintshire-and-wrexham	38	127,660	292,056	114.58 MiB	5 seconds	1 second	395.27 MiB
2022	Wales/flintshire-and-wrexham	38	129,007	292,644	115.03 MiB	5 seconds	1 second	396.56 MiB
2032	Wales/flintshire-and-wrexham	38	134,527	292,817	112.37 MiB	5 seconds	1 second	410.92 MiB
2039	Wales/flintshire-and-wrexham	38	136,425	293,540	112.22 MiB	5 seconds	1 second	410.77 MiB
2012	Wales/gwent-valleys	46	144,178	341,543	132.17 MiB	5 seconds	1 second	451.03 MiB
2020	Wales/gwent-valleys	46	148,386	344,566	132.83 MiB	5 seconds	1 second	450.89 MiB
2022	Wales/gwent-valleys	46	149,374	345,498	132.72 MiB	5 seconds	1 second	450.23 MiB
2032	Wales/gwent-valleys	46	151,842	347,976	130.50 MiB	5 seconds	1 second	442.86 MiB
2039	Wales/gwent-valleys	46	151,729	350,397	130.58 MiB	5 seconds	1 second	443.03 MiB
2012	Wales/gwynedd	17	52,926	122,595	48.28 MiB	3 seconds	1 second	141.47 MiB
2020	Wales/gwynedd	17	55,064	124,569	49.28 MiB	3 seconds	1 second	143.70 MiB
2022	Wales/gwynedd	17	55,683	125,030	49.20 MiB	3 seconds	1 second	143.45 MiB
2032	Wales/gwynedd	17	58,372	128,844	49.81 MiB	3 seconds	1 second	143.80 MiB
2039	Wales/gwynedd	17	59,746	130,948	50.64 MiB	3 seconds	1 second	145.62 MiB
2012	Wales/isle-of-anglesey	9	30,797	69,919	27.65 MiB	3 seconds	1 second	96.81 MiB
2020	Wales/isle-of-anglesey	9	31,366	69,845	27.85 MiB	3 seconds	1 second	97.39 MiB

year	study_area	num_msoas	num_households	num_people	file_size	runtime	commuting_runtime	memory_usage
2022	Wales/isle-of-anglesey	9	31,488	69,864	27.91 MiB	3 seconds	1 second	97.71 MiB
2032	Wales/isle-of-anglesey	9	31,601	69,502	27.09 MiB	3 seconds	1 second	95.51 MiB
2039	Wales/isle-of-anglesey	9	31,337	69,423	26.91 MiB	3 seconds	1 second	95.37 MiB
2012	Wales/monmouthshire-and-newport	31	100,402	240,491	94.44 MiB	4 seconds	1 second	280.40 MiB
2020	Wales/monmouthshire-and-newport	31	104,394	250,185	98.11 MiB	4 seconds	1 second	286.97 MiB
2022	Wales/monmouthshire-and-newport	31	105,481	253,282	99.27 MiB	4 seconds	1 second	289.03 MiB
2032	Wales/monmouthshire-and-newport	31	109,752	265,785	102.21 MiB	4 seconds	1 second	371.39 MiB
2039	Wales/monmouthshire-and-newport	31	111,246	273,319	103.90 MiB	4 seconds	1 second	373.82 MiB
2012	Wales/powys	19	59,028	132,725	51.21 MiB	4 seconds	1 second	185.06 MiB
2020	Wales/powys	19	59,972	132,328	50.60 MiB	4 seconds	1 second	183.38 MiB
2022	Wales/powys	19	60,190	132,467	50.46 MiB	4 seconds	1 second	182.88 MiB
2032	Wales/powys	19	59,586	133,010	49.63 MiB	4 seconds	1 second	180.64 MiB
2039	Wales/powys	19	57,969	133,514	49.35 MiB	4 seconds	1 second	179.80 MiB
2012	Wales/south-west-wales	50	165,004	383,260	145.79 MiB	5 seconds	1 second	474.32 MiB
2020	Wales/south-west-wales	50	170,327	385,937	146.53 MiB	5 seconds	1 second	474.47 MiB
2022	Wales/south-west-wales	50	171,623	386,901	147.00 MiB	5 seconds	1 second	476.10 MiB
2032	Wales/south-west-wales	50	175,897	392,107	145.19 MiB	6 seconds	1 second	469.31 MiB
2039	Wales/south-west-wales	50	176,482	394,303	144.53 MiB	6 seconds	1 second	467.47 MiB
2012	Wales/swansea	31	104,423	242,128	93.10 MiB	4 seconds	1 second	276.14 MiB
2020	Wales/swansea	31	110,304	247,820	95.72 MiB	4 seconds	1 second	281.38 MiB

year	study_area	num_msoas	num_households	num_people	pb_file_size	runtime	commuting_runtime	memory_usage
2022	Wales/swansea	31	111,940	249,098	96.10 MiB	4 seconds	1 second	282.16 MiB
2032	Wales/swansea	31	119,141	257,653	98.28 MiB	4 seconds	1 second	285.53 MiB
2039	Wales/swansea	31	123,450	262,306	99.93 MiB	4 seconds	1 second	366.61 MiB

Notes:

- `pb_file_size` refers to the size of the uncompressed protobuf file in `data/output/`
- The total `runtime` is usually dominated by matching workers to businesses, so `commuting_runtime` gives a breakdown
- Measuring memory usage of Linux processes isn't straightforward, so `memory_usage` should just be a guide
- These measurements were all taken on one developer's laptop, and they don't represent multiple runs. This table just aims to give a general sense of how long running takes.
  - That machine has 24 cores, which matters for the parallelized commuting calculation.
- The time *usually* doesn't include downloading or decompressing raw data. For some areas, it might!
- `scripts/collect_stats.py` produces the table above