DAFNI Pilot 3: Station Demand Model

Contents

Introduction to DAFNI Pilots

DAFNI will provide the National Platform to satisfy the computational needs in support of data analysis, infrastructure research and strategic thinking for the UK's long term planning and investment needs. The platform will support academic research that is aiming to provide the UK with a world-leading infrastructure system that is more: efficient, reliable, resilient and affordable. DAFNI will support big data analytics, simulation, modelling and visualisation.

DAFNI Pilots are a series of projects that run alongside the DAFNI core platform development and seek to take existing established infrastructure codes and implement them in a Cloud based environment that emulates the expected future DAFNI system. DAFNI pilot projects are submitted by the members of the DAFNI community and projects are chosen based on proposers' resource availability, benefits to DAFNI such as validating DAFNI's components, stress testing the DAFNI hardware etc. Each pilot project typically runs for 3-6 months and is supported by the DAFNI pilot team, consisting of 2-3 software developers. This will enable the following benefits to the DAFNI and its community:

- Demonstrate the capabilities of the DAFNI infrastructure.
- Feed the community requirements into improving and maturing the DAFNI infrastructure.
- Provide early access for the modellers to test their models on the DAFNI platform.
- Provide additional access to infrastructure models that may form part of the DAFNI service.
- Allow exploration of visualisation techniques useful to infrastructure modellers.
- Highlight typical data set requirements for infrastructure research.

Overview of Station Demand Model

The UK rail network is the oldest rail network in the world and comprises 15,811 km of track and features a total of 2,566 stations[1]. According to Network Rail, 1.7 billion people per year travel by rail in the UK and the number of passengers is rising by 6% each year[2]. With ever increasing demand for the rail network to be as efficient and wide-reaching as possible, it is imperative that any improvements or additions to the network are thoroughly thought out before spending the limited budget available.

Recent projects like HS2 and Crossrail have highlighted this need for thorough planning through the negative press each has received with HS2 being labelled as a "vanity project" and Crossrail facing multiple setbacks and delays. An area of particular importance is ensuring that any new stations added to the network will actually be used. There is no point in adding a new station to the network if passengers will stick to using the station that they used previously.

One of the primary factors when considering an individual's station choice is how easy it is for them to get to that station. Additional factors might include how many parking spaces are available, the frequency of service to that station and whether or not there are ticket machines, bus stations or CCTV cameras.

---

[1] Rail infrastructure, assets and environmental – 2016–17 Annual Statistical Release (PDF). Office of Rail and Road. 24 October 2017. Available at: https://orr.gov.uk/__data/assets/pdf_file/0008/25838/rail-infrastructure-assets-environmental-2016-17.pdf

[2] Passengers - Our role in getting you where you need to go. Network Rail. Available at: https://www.networkrail.co.uk/communities/passengers/

Marcus Young and Simon Blainey from the University of Southampton have developed a trip-end model which can be used to estimate the average number of trips made by rail from a particular area. Trip-end models are specifically applicable to the construction of new stations to estimate how many trips would be made from that new station. A trip-end model also incorporates additional information on demographic, socio-economic and service-related data for areas around new stations[3]. This Station Demand Model can be used to predict how many people would switch to using a newly proposed station as opposed to their previous station of choice.

One common weakness of trip-end models is in how station catchments are defined. This usually involves either using a distance or time-based buffer, or dividing the study area into zones and assigning each zone to its nearest station. However, all of these approaches produce discrete and non-overlapping catchments and suggest that anyone within a catchment will always use the same station and that stations do not compete with each other directly[4], which does not reflect actual travel behaviour. The station demand model used in this pilot addresses this problem by allocating a set of stations to each area and then using probabilities (derived through station choice models) to decide how many people would choose each of the stations.

Analysis of Software

Code

The model is written using R and is built with RStudio. The code is committed to a private GitHub repository. The R code is comprised of nearly 4,000 lines of code across about 20 files. The model also makes use of 8 SQL scripts which are used to prepare data for processing.

Input data

The model takes in a wide range of input data. This was all sent to DAFNI in a database dump created by Marcus and includes the following data:

- Polygons which show the areas attributed to different postcodes and workplace centroids in Great Britain as well as an outline of GB as a whole.
- The population per postcode from the 2011 census data.
- The average household size per Local Authority District in the UK.
- The station locations throughout GB.
- Ordnance Survey's Openroads data which contains road plotlines as well as their locations.

Configuration

The model can be configured in a number of ways. Firstly, the model can either be run in "concurrent" or "in isolation" mode. In isolation mode, each of the stations will be treated as if they were added to the network separately. This allows sensitivity analysis to be done if the same station is added more than once with varying levels of parking spaces, service frequency, etc. Concurrent mode treats the stations as if they were all being added to the network at once.

---

[3] Passenger demand forecasting for third party funded local rail schemes – Guidance Note. Department for Transport. 24 October 2011. Available at: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/3978/guidance-note.pdf

[4] Development of Integrated Demand and Station Choice Models for Local Railway Stations and Services. Marcus Young. 2017. Available at: https://eprints.soton.ac.uk/414263/1/myoung_etcpaper_2017.pdf

In either mode, stations must be added to the model run. Each station has the following configuration options:

Configuration Option	Description
ID	ID of the new station
Name	Name of the new station
Region	The region in which the new station will sit
Station NGR	Easting and Northing values for the station itself
Access NGR	Easting and Northing values for the station's access point
Category	The category of station (either E (small staffed) or F (small unstaffed))
Model abstraction for stations	A list of pre-existing station CRS codes for which abstraction results should be carried out.
Parking Spaces	The number of parking spaces available at the station
Frequency	The frequency of service to this station
Frequency group	All stations in the same frequency group will have the same frequency. This is due to them being on the same rail line and so any train passing through one station will pass through the others in the group. Different frequencies can occur if more or fewer trains stop at the new station.
CCTV	Whether or not there is CCTV at the station
Ticket Machine	Whether or not there is a ticket machine at the station
Bus Interchange	Whether or not there is a bus interchange at the station
Terminal Station	Whether or not the station is at the end of the line
Electric Services	Whether or not the station has electric services
Travelcard boundary	Whether or not the station is at a travel card boundary

In addition to station metadata, additional exogenous data can be added to a model run. This comes in the form of either adding housing, population or jobs to either a postcode or workplace centroid. The idea of this option is to allow users to update areas where there has been significant growth since the 2011 census, or is expected to occur in the near future.

The intensive part of the processing for the model actually occurs in the Postgres database to which the model is connected. This is where the catchment areas are calculated by using the Postgres extension, pgRouting[5]. To generate the choice sets for each of the postcodes within a certain service area, the drive time from each one of these postcodes to the new and existing stations must first be calculated. This process is the longest-running aspect of the model run. However, this process runs in parallel with the task being split across a number of CPU cores.

Output data

The output data is made up of two major parts: the proposed stations and the abstraction results. The proposed stations results contain polygons for the service areas of each station in the job submission as well as the probability catchment for that station. The abstraction results are only generated if the "model abstraction for stations" option was filled in during job submission. These results show the probabilistic catchments for each station in the "model abstraction for stations" list both before the new stations were added and afterwards. This gives a good representation of the

[5] pgRouting. Available at: https://pgrouting.org/

effect that the new station(s) could have on the pre-existing network in terms of drawing passengers away from one station and into another.

Pilot Scope

Objectives

The main aims for the pilot project were to:

- Provide a simple way to run the application on cloud-based infrastructure so as to form a part of DAFNI.
- Allow the user to vary the input parameters to the model through a simple user interface which does not depend on technical knowledge of the model code.
- Provide a high-powered database for the pgRouting process to run in. This should have 32 cores available to it as a minimum and should mean that the model runs much more quickly.
- Give the user a simple way to visualise the results. This will involve plotting the probability catchments on a map of the UK and presenting results tables.

Challenges

As previously mentioned, the model is entirely written using a combination of R and SQL scripts. One of the first tasks was to bundle the model into a Docker container. This turned out to be particularly challenging due to the way that the model was developed using RStudio which handles all the packaging of the different R functions. The solution was to use the rocker/verse[6] Docker image from Dockerhub to include RStudio's terminal. The model can then be installed correctly, replicating the functionality which is provided by RStudio.

The model relies heavily on various GIS data both in its inputs (OpenRoads data) and outputs (catchment polygons). This meant that the Django API had to be modified to be able to handle this correctly in order to be able to effectively store GIS data in its database and serialize this into JSON. For this, Django-rest-framework-gis[7] was used; although parts of the code had to be modified in order to be able to achieve the nesting which was required by the results data.

Deciding on the architecture for the backend system was an additional challenge. As previously mentioned, the majority of the intensive processing occurs in a database. In addition to this, the database must be pre-loaded with a large amount of pre-processed data. Including all of this data in the image itself would make an extremely large image size. Downloading this data each time the model runs would also take a long time. A solution was therefore to add all of the pre-processed data to a central database where all job processing would happen. This database has 32 cores, 768GB of memory and is backed by a flash-based storage area network. The database for the station demand model is 6GB when uncompressed.

One disadvantage of this approach is that if a user were to run a job and request 32 cores then another user would have their job waiting until the previous job completes. This is something to think about going forward when we have other models which make use of databases to do their processing. One alternative would be to keep the database dump on fast storage on the server and to then pull this in to a fresh database each time the model was run. This would ensure that a model run would only ever be waiting if all of the Kubernetes resources had been exhausted.

---

[6] Rocker/verse Docker image. Rocker. Available at: https://hub.docker.com/r/rocker/verse

[7] Django rest framework GIS. Available at: https://github.com/djangonauts/django-rest-framework-gis

Outcomes

This pilot builds upon the previous backend and middleware which was developed in the previous pilots. However, the introduction of GIS data and nested parameter submission required a refactor of the backend to be able to support this new type of model. This refactor puts the backend in a much better state to be able to work with a wider variety of models as it no longer requires models to be in a generic format.

As well as the updates to the backend infrastructure, a front end interface to the model was developed using Vue JS. A new approach was also taken in developing the front end as NUXT was used for this pilot in an effort to unify the front end with the direction taken by the core team in their front end design.

In summary, this pilot project has provided the following:

- A web interface built in the Vue.js JavaScript framework. This interface allows remote access to the models without any need to install software on the local computer. All the user settable parameters are exposed through this interface and can be altered before a simulation is submitted. The server side code runs on a machine within DAFNI and allows computational jobs to be submitted onto DAFNI cloud resources.
- A database has been set up to store the job parameters and results from simulation jobs. This allows a series of jobs to be submitted by the user and then compared to see the effects of the changes to parameters. This allows a persistent store of simulations.
- For submitting jobs to the DAFNI cloud infrastructure the set of APIs developed for previous pilots have been extended to allow them to work with the requirements of the station demand model. The database had to be updated to store the new parameters and results and the job queue adapted to allow different images to run as required.
- The visualisation requirements mainly involved plotting probability catchments on a map of the UK. These plots are accompanied by a table which details how many people would begin to use a new station based on the results of the model run. For abstraction analysis, the number of potentially lost entries/exits at each modelled existing station is also shown.

A typical job submission process is shown in Figures 1, 2, and 3. This shows the process of setting configuration options, adding new stations and adding exogenous data.

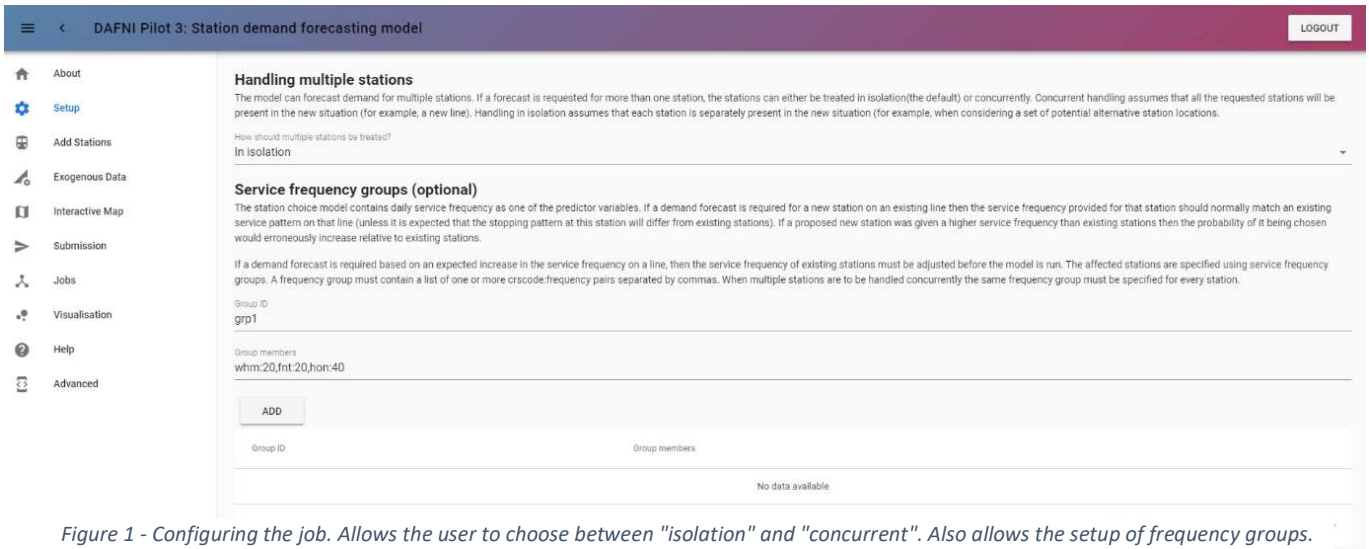

Figure 1 - Configuring the job. Allows the user to choose between "isolation" and "concurrent". Also allows the setup of frequency groups.

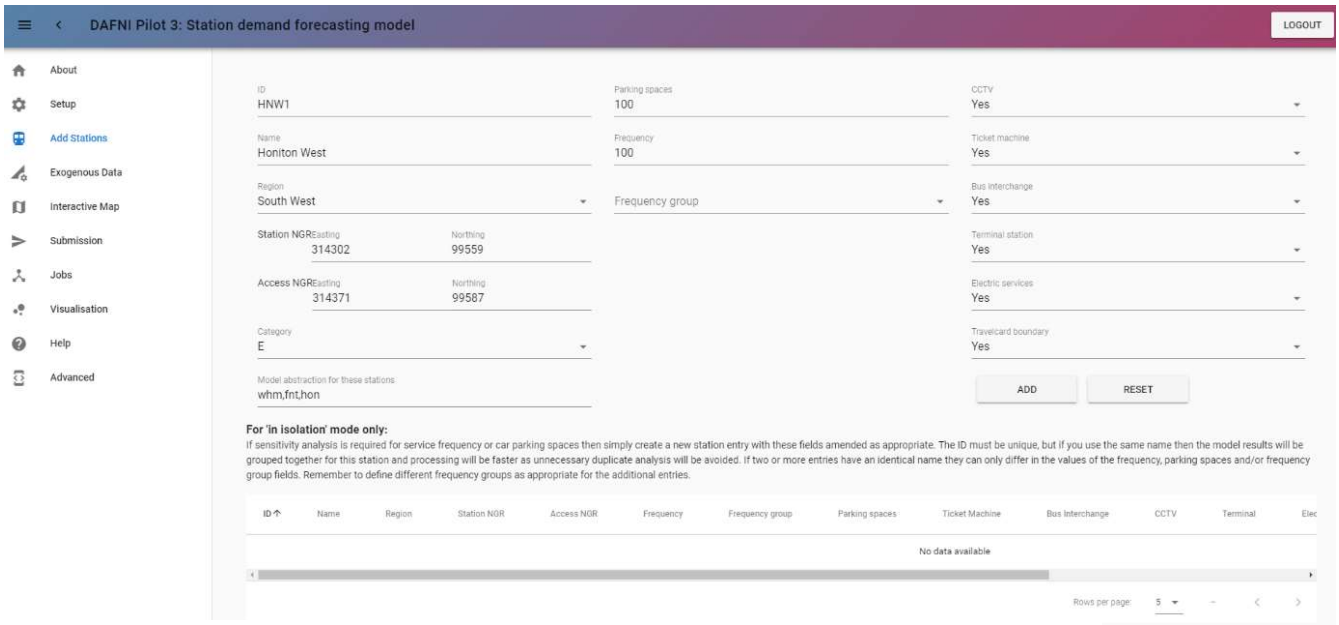

Figure 2 - Adding stations to the model run. Allows the user to configure each station individually.

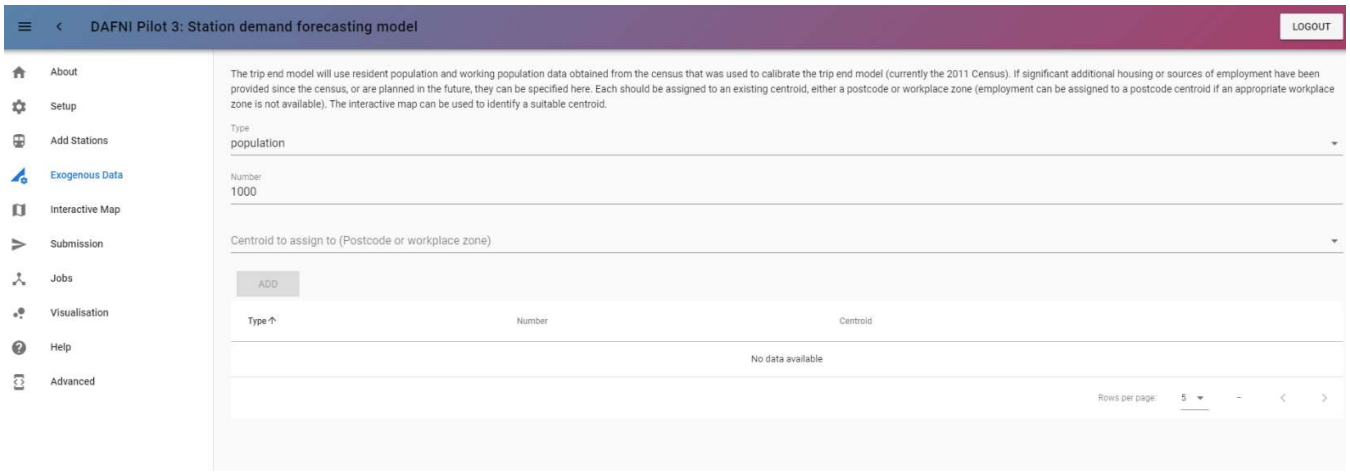

Figure 3 – Adding exogenous data to the model run. Allows a user to add population, households and/or jobs to a model run.

After a job has been submitted and the run has completed, results can be accessed on the visualisation page. Figure 4 shows the probabilistic catchment for Penryn station after having added a station near Helston (although this is only a test scenario so the figure is only meant as a guide to what the visualisation looks like).

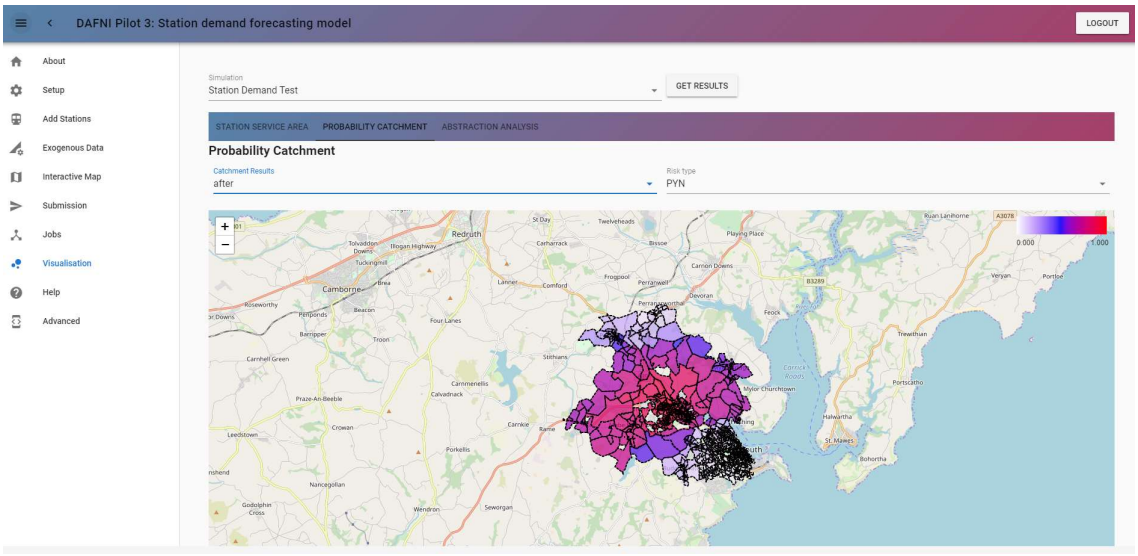

Figure 4 – Visualising the results of a test run. Showing the probabilistic catchment around Penryn station.

The visualisation page can also show the abstraction analysis results in a table. This table gives a more precise view of how many trips there are at each station before and after the addition of a new station. This table is shown in figure 5.

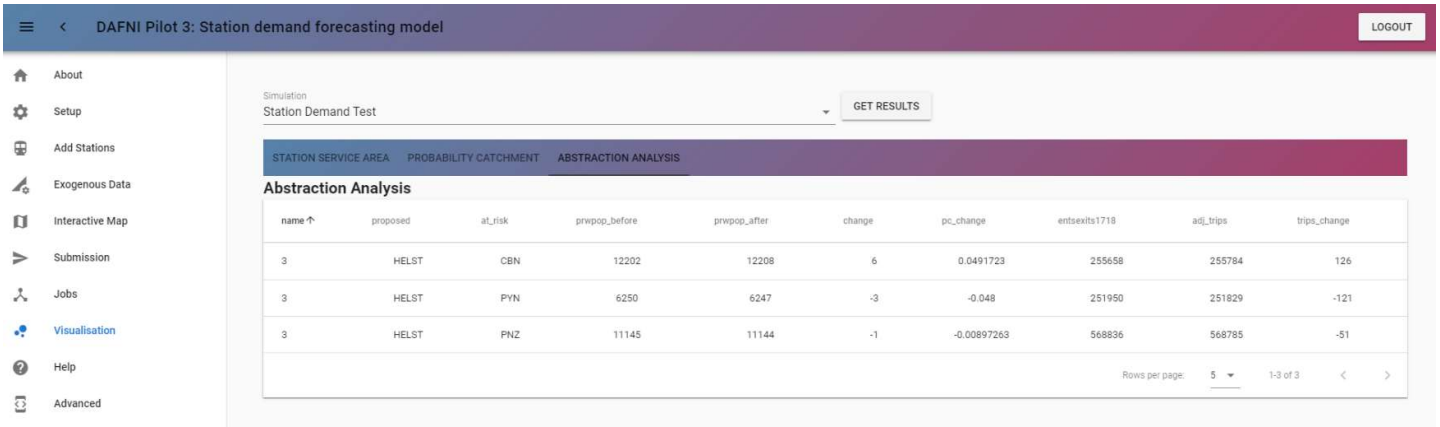

Figure 5 – Viewing the abstraction analysis results in a table.

Benefits

The main benefits from pilot 3 were:

- Showing that R-based models can easily be adapted to run in the same backend framework that was developed for the previous pilots.
- Demonstrating that DAFNI can provide a significant speed-up for running models compared to a typical workstation PC by simply providing a large amount of compute resource to the model.
- The provision of an easy to use interface to the model will help to make the software much more accessible to non-technical users who wish to investigate the model's behaviour.
- The usage of GIS data and the need to store such data has helped to define how this will be done effectively moving forward.

- The demonstration of efficient use of PostGIS and pgRouting with Open Roads data may prove useful to future projects on DAFNI.
- In order to get user feedback from the pilot, we opened up the user interface to the modellers. This process has ensured that we have a good understanding of how to do this and we can now do the same for previous pilots.

Lessons Learned

For the Platform

- The knowledge gained about GIS data (and in particular about PostGIS and Django rest framework GIS) will undoubtedly be invaluable for the core team moving forward. As more data is added into the NID, there will be a requirement to be able to store and download GIS data. After extensive research during this pilot, it is clear that a PostGIS database will be needed and that Django Rest Framework GIS would be a good choice for serializing this data. Also, the additions that had to be made to the DRFG package will also be useful for the core team.
- We will have to think about how to handle models which make use of intensive processing on the database. The solution for this pilot is not ideal and this would form a good case study for how to improve moving forward. The discussions with our systems administrator seem to point at storing the database dump on fast storage and then recreating the database each time. This will add extra run time to the model but it will have the benefit of being able to run more instances of the model in parallel without any slowdown.
- The interactive map for this model has many more points plotted on it than any of the previous models. The previous solution (used in pilot 1) was to use Mapbox to plot all of the points on the map. This turned out not to scale well when plotting millions of points (one point for every postcode and workplace centroid). The interactive map for this model now uses Leaflet instead and this is much more scalable.

For the Pilots

- The earlier we start to make the front end available to the piloteers (i.e. the researchers providing the model) the better. In this case, the front end was probably made available after it should have been which meant that valuable feedback from the piloteers might have been unable to make it into the final product. In future, we should try and operate in a more agile way where the piloteers actually get access to the UI as opposed to just being shown it through a video call.
- Tagged releases of the piloteers' code base are incredibly useful. We should set out at the outset of future pilots that models will need to be tagged for release prior to the end of the pilot. This will ensure that we always have a solid base which will not change meaning that this repository can always be pulled from and we can expect the same code.
- It is useful for the piloteers to be able to see the logging output from a model run. This allows them a much better insight into whether the model is running correctly. For the purposes of the pilot, a very rudimentary logging system was implemented which simply posts the set of logs after a model run has finished. This could be made much more effective by streaming the logs to the API as they occur. This would give the user a live output of their model run.