D4.4 Library for AGRICORE data visualisation purposes

agricore

Deliverable Number	D4.4
Lead Beneficiary	AAT
Authors	AAT, IDE
Work package	WP4
Delivery Date	M33 (May 2022)
Dissemination Level	Public

www.agricore-project.eu

The Agricore project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No. 816078

Document Information

Project title	Agent-based support tool for the development of agriculture policies
Project acronym	AGRICORE
Project call	H2020-RUR-04-2018-2019
Grant number	816078
Project duration	1.09.2019-31.8.2023 (48 months)
Deliverable Authors	Juan Carlos Castillo Alcántara (AAT), Rubén García Hoyos (AAT)
Deliverable Reviewers	Obdulia Parra (CAAND), Fernando Dorado (IDE), Dimitrios Natos (AUTH), Pablo Báez (IDE)

Version History

Version	Description	Organisation	Date
0.1	Compiling information and tools	AAT	01 Feb 2022
0.2	ToC definition	AAT	08 Mar 2022
0.3	Selection of the charting tool	AAT	15 Mar 2022
0.4	ToC reviewed	IDE	12 Apr 2022
0.5	Apache ECharts for data visualisation	AAT, IDE	26 Apr 2022
0.6	Development of sections	AAT	09 May 2022
0.7	Review of the deliverable	IDE, AUTH, CAAND	24 May 2022
0.8	Changes implemented	AAT	30 May 2022
0.9	Deliverable exported, formatted and completed	IDE	31 May 2022

Disclaimer

All the contributors to this deliverable declare that they:

- Are aware that plagiarism and/or literal utilisation (copy) of materials and texts from other Projects, works and deliverables must be avoided and may be subject to disciplinary actions against the related partners and/or the Project consortium by the EU.
- Confirm that all their individual contributions to this deliverable are genuine and their own work or the work of their teams working in the Project, except where is explicitly indicated otherwise.
- Have followed the required conventions in referencing the thoughts, ideas and texts made outside the Project.

Executive Summary

AGRICORE is a research project funded by the European Commission under the RUR-04-2018 call, part of the H2020 programme, which proposes an innovative way to apply agent-based modelling to improve the capacity of policymakers to evaluate the impact of agricultural-related measurements under and outside the framework of the Common Agricultural Policy (CAP).

Within the AGRICORE suite of tools, its main graphical user interface (GUI) can be found. Developed as a cross-platform desktop application, this tool enables users to configure and launch simulations on agricultural policies using synthetic populations and the Agent-based model (ABM). In such simulations, the analysis of the effect of the different policy measures is carried out through the computation of a series of key performance indicators (KPIs). Once finished the simulations, these KPIs will be used to assist in the analysis and visualisation of the results, representing them graphically in interactive charts and maps that will help users and policy makers to draw conclusions and support decision-making processes.

This document presents an analysis of different tools for the visualisation of data in charts and maps with the aim of providing the platform with the most suitable tools based on the objectives, requirements and characteristics of the AGRICORE project. The document begins with an introductory chapter that analyses the need to explore these tools and why they are necessary for the project. Then, it continues with a chapter explaining and comparing, with advantages and disadvantages, each of the tools selected and analysed. Finally, it explains the tool chosen for data visualisation in AGRICORE.

A third chapter explains how the chosen solution will be used in the platform and finally, a concluding chapter summarises the conclusions drawn and explores the next steps to be taken.

Abbreviations

Abbreviation	Full name
ABM	Agent Based Model
API	Application Programming Interface
ARDIT	Agricultural Research Data Index Tool
CSS	Cascading Style Sheets
DOM	Document Object Model
EU	European Union
GL	Graphics Library
GUI	Graphical User Interface
HTML	HyperText Markup Language
JS	JavaScript
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
NUTS	Nomenclature of Territorial Units for Statistics
OpenGL	Open Graphics Library
SVG	Scalable Vector Graphics

List of Figures

List of Tables

Table of Contents

1 Introduction

The present document covers the analysis conducted for the selection of the tool/tools necessary for the visualisation of data in graphs and maps within the AGRICORE project graphical user interface, which provides the user with the required functionalities to configure and launch simulations for the agricultural policy assessment. The impact assessment is translated into a series of measures that could affect jointly: a) the already implemented policies (ex post analysis) and b) policies in the design phase (ex-ante analysis). It can be divided into three categories (socio-economic, environmental and delivery of ecosystem services) from which a set of key performance indicators (KPIs) are computed during the simulations so they can displayed with the results of them. To that end, the AGRICORE interface needs a visualisation section where users could visualise and analyse the results of the completed simulations in a simple and user-friendly way. Figure 1 shows the cyclical process of policy evaluation in the AGRICORE project through the agent-based simulations.

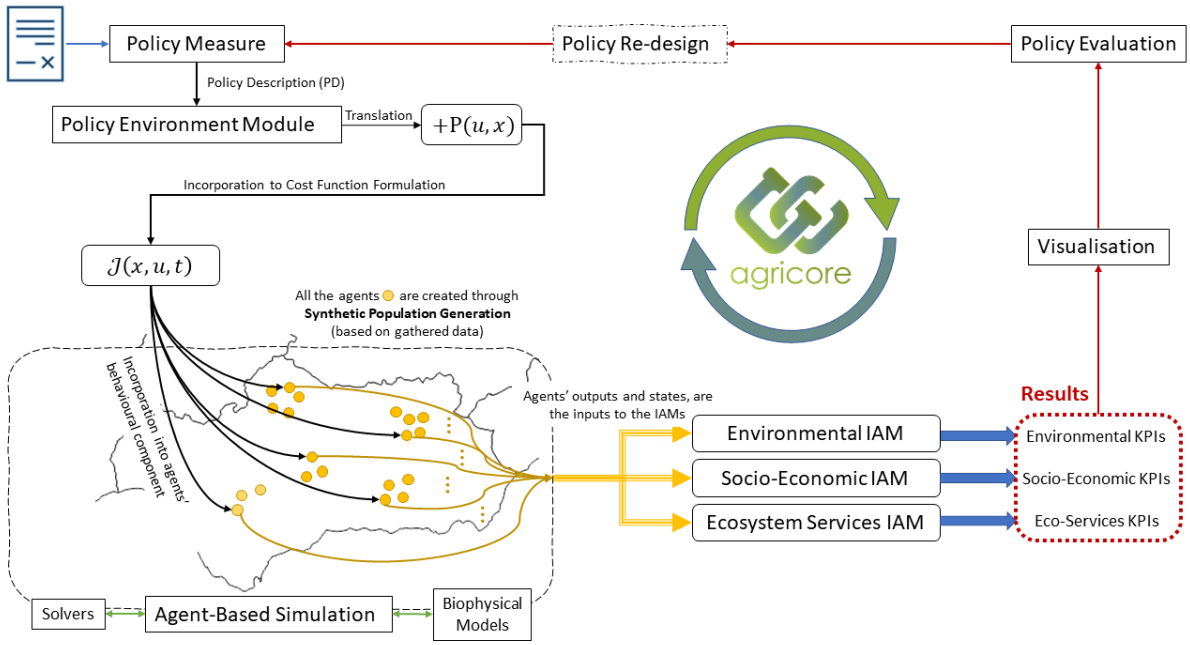


Figure 1 Cycle of Policy Evaluation using the AGRICORE tool

Figure 2 shows the different modules that compose the AGRICORE architecture, where the AGRICORE GUI (D.16) will centralise the user interaction with the AGRICORE suite linking directly to other project modules such as the ARDIT (D.1), the data warehouse (D.2) or the module for the synthetic population generation (D.5) and with the activation, once launched from the interface, of the simulation processes through the ABM simulation module (D.6). A detailed description of all AGRICORE modules can be found on deliverable 'D6.1 - AGRICORE architecture and interfaces'.

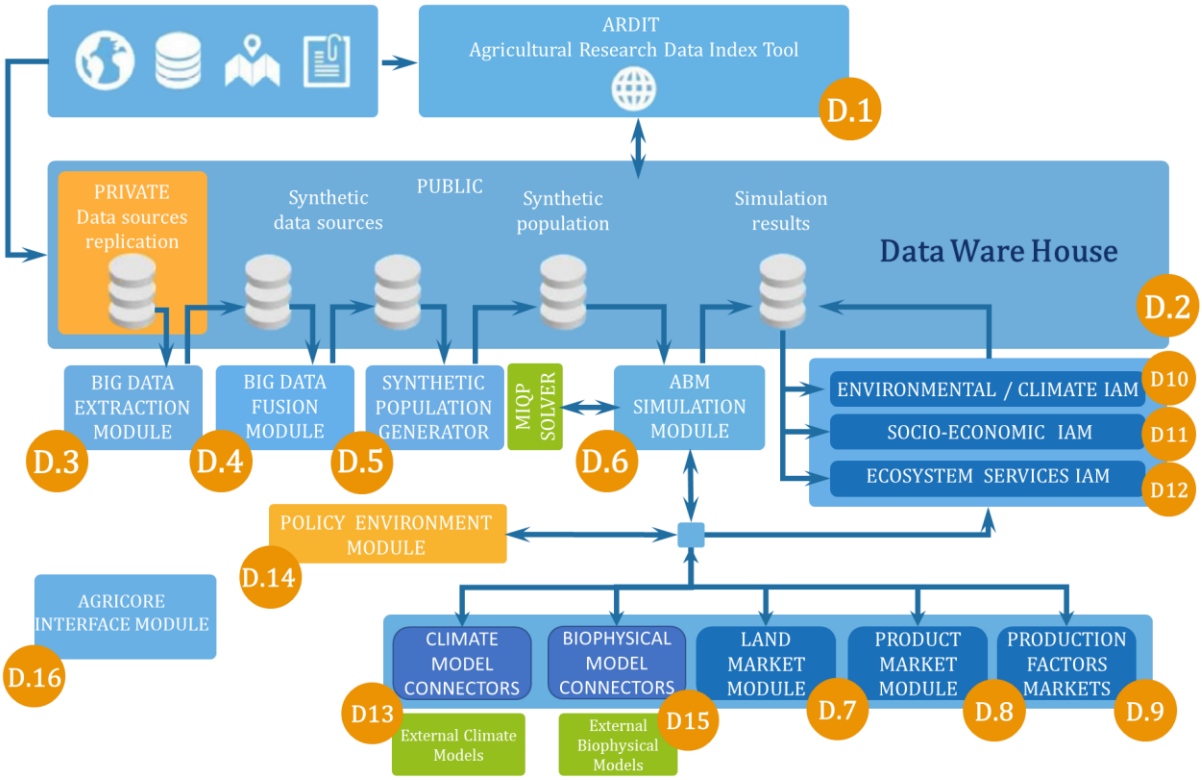


Figure 2 AGRICORE Modular Architecture

Developed as a cross-platform desktop application with Electron [1], the AGRICORE interface provides a platform to interact with the AGRICORE suite from any modern operating system (Windows, macOS or Linux based systems). Based on Node.js to build the server-site and Chromium for the user interface, Electron allows to build entire applications using only JavaScript, HTML, CSS and the same frameworks, tools or libraries used for website development. In this way, current popular frontend frameworks such as Angular [2], React [3] or Vue.js [4] can be used for building the user interface.

The AGRICORE deliverable 'D4.3 - Validated design for the AGRICORE interface' includes the mockup design of the screens that compose the platform and the definition of the requirements from which can be extracted the functionalities and characteristics of the visualisation section of the simulation results:

"***AG.D16.FR.001-4. Visualisation Module:*** *users must be able to visualise the results from a finished simulation. These results can be automatically immediately displayed after the end of a simulation, or uploaded by the user using a previously obtained results file. It should be possible to download each visualisation individually as an image file or all the visualisation together as a PDF report. Advanced users must have the possibility of opening a Jupyter IDE to create their own customised visualisations by directly processing the raw result data*"

The following Figure 3 depicts the mockup designed for the visualisation module of the AGRICORE interface:

Figure 3 Visualisation section in the AGRICORE interface

2 Comparative tool analysis

The first step for the selection of the visualisation tool(s) consisted in the compilation and comparative analysis of the most popular ones. Then, the analysis was divided into two parts, the first one, for tools dedicated to charting data and the second one, for tools that allow the construction of interactive and detailed maps of different types. The specific selection of the tools was based on previously established requirements and needs by the AGRICORE project. These include:

- The tool(s) should be free of charge, open-source or a free version of them could be used without time or use limit.
- Given that the platform will be developed using a JavaScript-based framework the tool(s) to be selected must be compatible with this technology. Therefore, special emphasis will be placed on tools and libraries developed in this programming language.
- As the platform's user interface will be developed using Angular, the tool(s) selected must have easy integration with this framework.
- The tool(s) must have a small learning curve but, at the same time, they must be as complete as possible, i.e. they should include as many charts, map types and options as possible.
- Given the requirement for downloading the visualisations as an image, it would be desirable to choose a tool that allows or facilitates the export of the generated graphs or maps.
- As the visualisation module aims to help and support policy makers in analysing the simulation results, the tool(s) to be selected must be highly interactive to allow the user an in-depth data exploration.

2.1 Tools for visualisation graphics

This subchapter comprises all the tools selected for displaying graphs that allow different types of charts to be plotted on the screen. Unlike the tools for geo-referenced data, the task description in the project's Grant Agreement document did not specify any examples, so all tools were chosen based on the requirements detailed above. These include:

- Chart.js
- D3.js
- Recharts
- Ngx-charts
- Apache ECharts

2.1.1 Chart.js

Chart.js [5] is an open-source charting JavaScript library maintained by its user community which offers a great solution for representing data graphically in web-based applications. Some of its main features include:

- Is one of the most popular JavaScript charting libraries with a large community of users.
- Offers 8 different chart types: bar, line, area, circular, bubble, radar, polar and scattering. Allowing to mix different chart types.
- Allows to customise the styles if the charts.

- Has advanced animations when displaying data or interacting with charts.
- Include plugins that enhance the performance of charts with large data sets.
- Being a JavaScript library, it has full integration with React and Angular.
- Uses HMTL5 canvas to draw the graphics with a full responsive design that adapts to the screen size.
- Supports modern browsers.

The following Figure 4 and Figure 5 depict two working examples with Chart.js [6]:

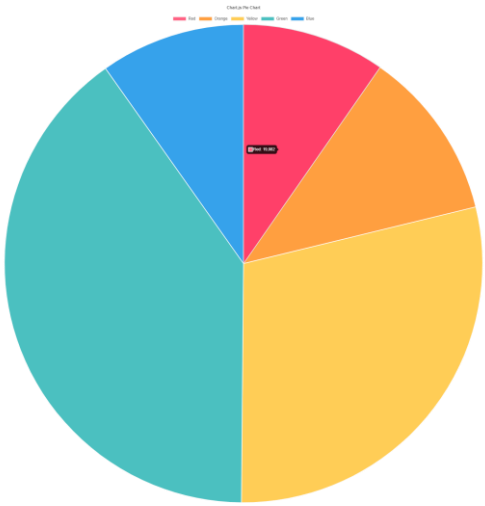


Figure 4 Example of pie chart in Chart.js

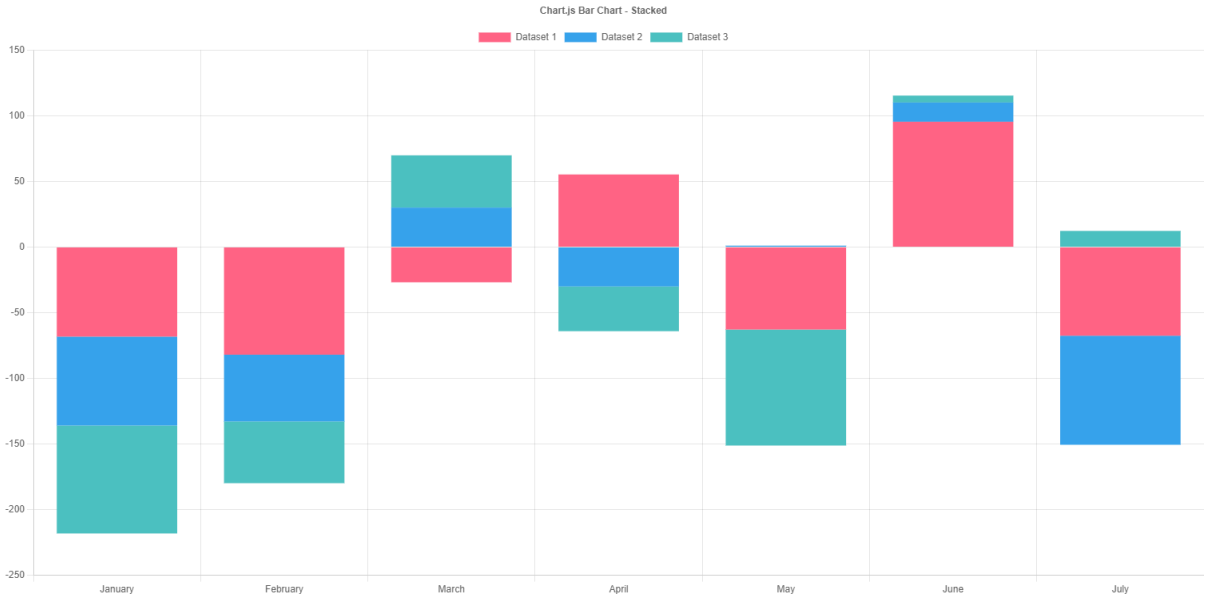


Figure 5 Example of bar chart in Chart.js

2.1.2 D3.js

D3.js [7], Data-Driven Documents, is a JavaScript library developed by Mike Bostock used to manipulate documents based on data and visualise data in web applications. This library provides features to make data interactive using HTML, SVG and CSS, binding the data to the Document Object Model (DOM) and supporting large datasets to obtain a fully interactive and animated solution. Among its main features this library allows:

- The graphical representation of information in different types of charts: line, bar, radial, area, network, hierarchy, scatterplot and more types of charts.
- Representation of geographical data in maps, supporting GeoJSON, TopoJSON or shapefiles.
- Interaction features: the library includes interaction methods like dragging, brushing or zomming to create fully interactive graphics.
- Animation features: the library also includes animated transitions over data using data joins, interpolators or easings.
- Data analysis features include tools for quantitative analysis, such as data transformation, random number generation or hexagonal binning.
- Including labels, legends, axes, titles, guides and essays to help users providing full explanations about the graphics and the visualisation.

In addition, D3 provides community-developed modules which can increase the functionalities of the library.

In general, it is a very complete solution for data visualisation in web environments (Figure 6), however, it is a library that has a high learning curve and requires time for developers to dominate it completely, as it is necessary to have knowledge not only in HTML, CSS, JavaScript, but also in SVG or canvas.

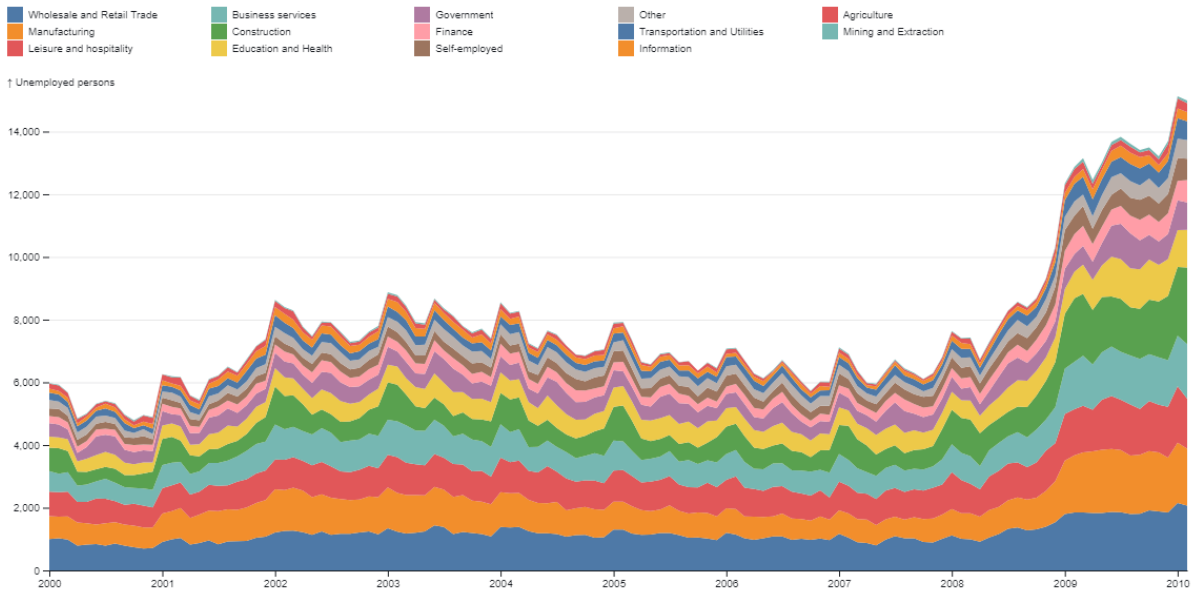


Figure 6 Example of stacked area chart in D3.js

2.1.3 Recharts

Recharts [8] is an open-source chart library built with React and D3.js which provides a simple and easy way to integrate charts (Figure 7, Figure 8) into applications built with this library. Its main features are:

- Is a React exclusive library but it has a very simple deployment and use in this technology thanks to the use of components to represent the charts.
- Provides native SVG support with a lightweight dependency on some D3.js submodules.
- It offers different chart types: line, area, bar, scatter, pie, radar or treemap.

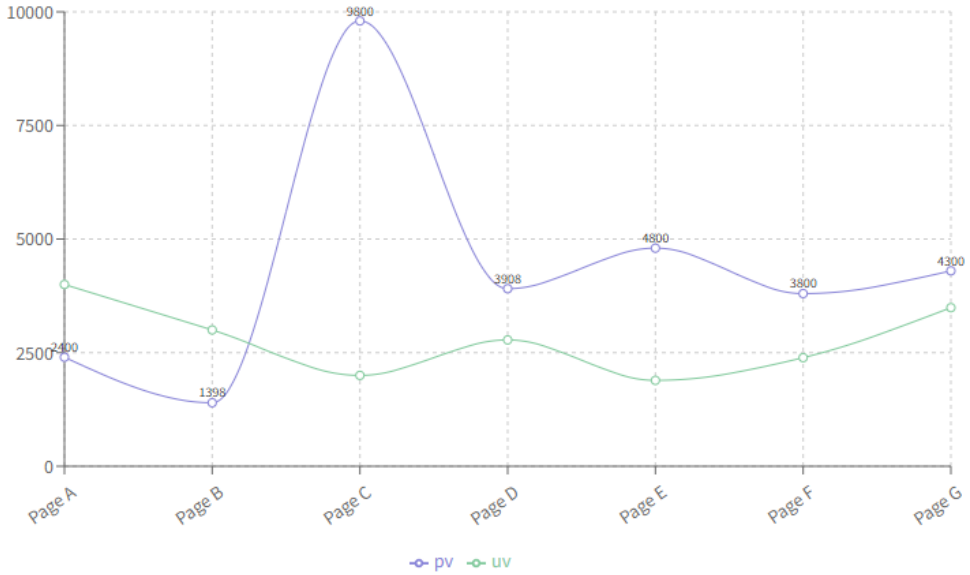


Figure 7 Example of line chart in Recharts

Figure 8 Example of combined chart in Recharts

2.1.4 Ngx-charts

Ngx-charts [9] is an open-source Angular charting library developed by Swimlane which offers a similar solution to Recharts with React. It is a simple and easily integrated library with SVG support based on the D3.js library for math functions, scales and axis shape generators (Figure 9, Figure 10). It offers different types of charts (bar, pie, line, area, bubble, box, tree map or gauge among others) with the possibility to edit their styles completely through CSS.

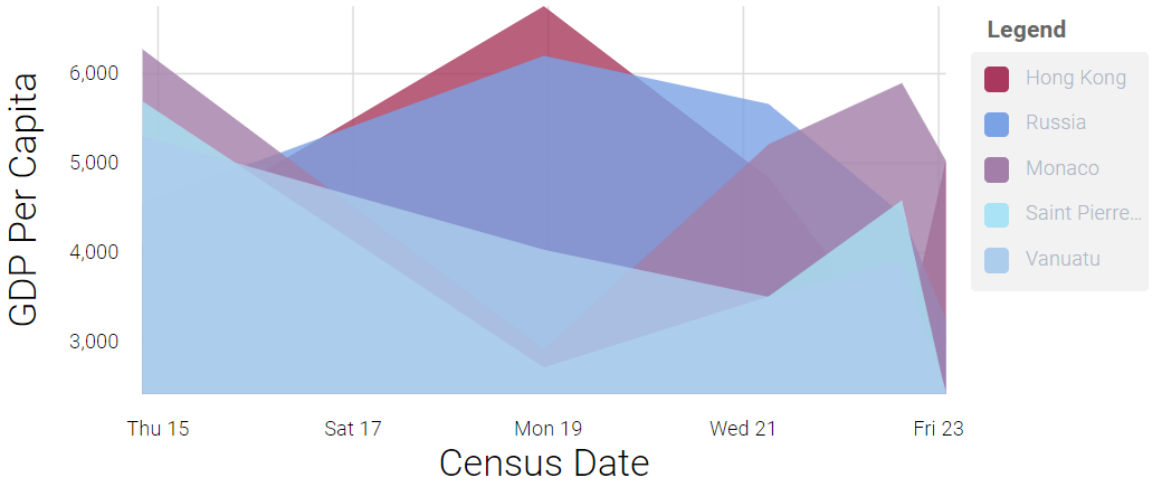


Figure 9 Example of stacked area chart in Ngx-charts

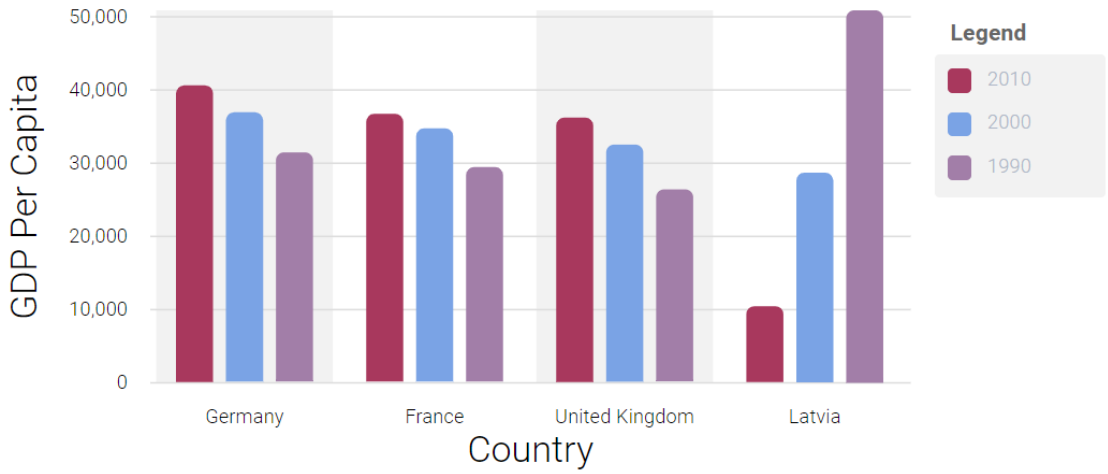


Figure 10 Example of bar chart in Ngx-charts

2.1.5 Apache ECharts

ECharts [10] is an open-source JavaScript visualisation library that, unlike the libraries explained above, allows the possibility of working with maps and the representation of information on them. The library is based on ZRender, a graphic rendering engine that allows the creation of fully interactive and highly customisable charts (Figure 11, Figure 12). Its main features are:

- Provides more than 20 charts and map types which can be combined to use.
- Fully interactive and animated charts and maps, combining the use of canvas or SVG for the rendering of the data.
- Provides modern browsers, mobile and PC support for efficient interaction and rendering of data on different platforms.
- Allows working with data in multiple formats, e.g. with matrices or key-value maps.
- Supports visualisation and rendering of large data sources (more than 10 million elements) through streaming data with WebSockets, enabling efficient interaction, zooming and transformations on charts and maps.
- Has a web tool in which the user can select the different charts and components he/she wants to work with, making the library download package lighter and customizable [11].
- Has a timeline component that allows for modifying the content displayed in the charts according to the time dimension.
- Provides a toolbox which includes features to export the charts as images, change the chart type or zoom in.

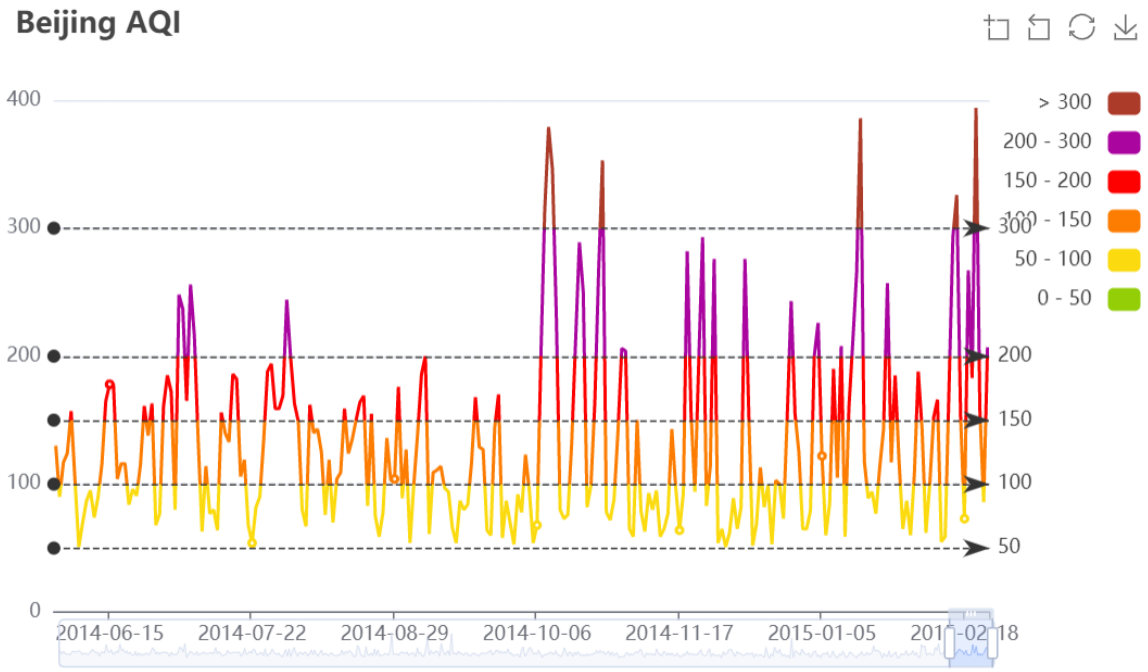


Figure 11 Example of line chart in Apache ECharts

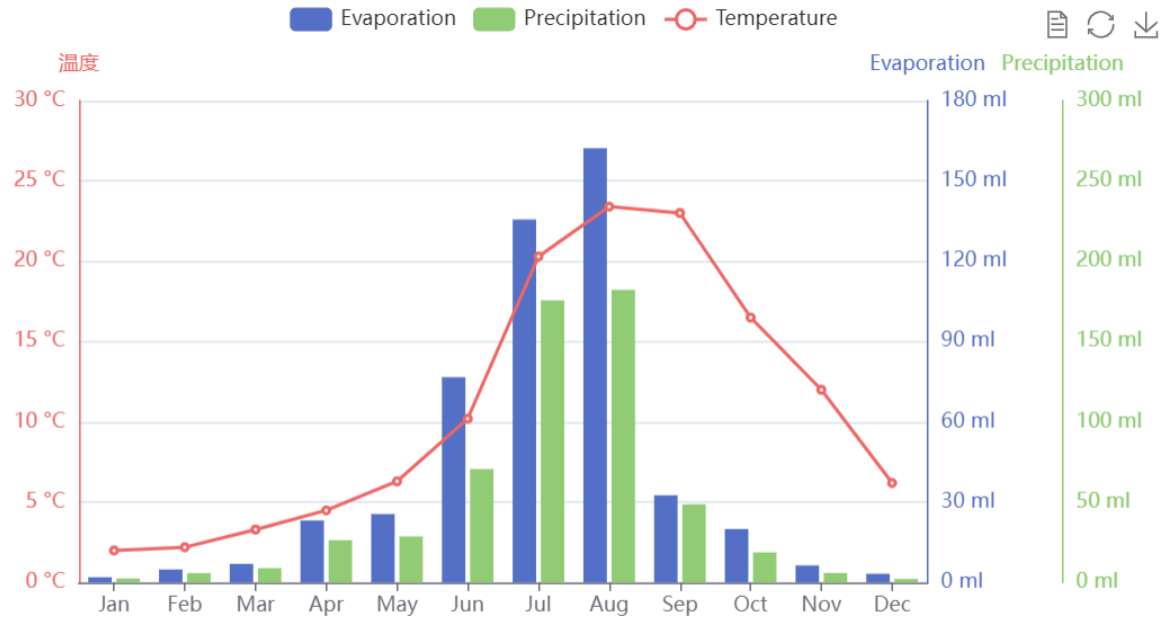


Figure 12 Example of a combined line and bar chart in Apache ECharts

2.2 Tools for geo-referenced data

This subchapter comprises all the tools selected for displaying maps in the visualisation module of the AGRICORE interface. In contrast to the charting tools, the project's Grant Agreement document included a list of tools selected to carry out this process. That list includes:

- Google Maps
- Polymaps
- ArcGIS
- OpenHeatMap
- TargetMap

In addition, other tools selected during the comparative analysis phase have been added:

- Mapbox GL JS
- Apache ECharts maps
- Leaflet
- OpenLayers

2.2.1 Google Maps

Being Google Maps one of the most popular services for map visualisation, this tool has a JavaScript API [12] that users can use to build fully interactive, dynamic and customisable solutions (Figure 13, Figure 14). Among its main features the Google Map's JavaScript API has:

- The possibility to display different types of maps: road, satellite, hybrid, terrain or custom.
- Custom styled markers can be added to the maps with the possibility to show info windows that increase the context and information about the markers.
- Event handling on user interaction with the map.
- Display GeoJSON and other data types on the map.
- Polygons and other different types of shapes can be drawn on the map.
- The tool also supports the possibility to visualise heatmaps showing the density of data at geographical points.

Since this is a commercial service provided by Google, in order to make use of this tool it is necessary to generate an API key that must be added to the code of the applications where the maps need to be used, in order to identify and authenticate the platform, and therefore, for Google to be able to accept requests to the maps API. On the other hand, the Google Maps Platform is free as long as its use does not exceed a cost of 200$ per month, which would be equivalent to 28,500 map loads per month at the most.

Figure 13 Polygon drawing with Google Maps JavaScript API

Figure 14 Heatmap with Google Maps JavaScript API

2.2.2 Polymaps

Polymaps [13] is a free JavaScript library developed by Stamen and SimpleGeo for making dynamic and interactive maps (Figure 15, Figure 16) like Google Maps using SVG and OpenStreetMap as a base. Its functionalities are very similar to those of Google Map Platform, as its main objective is to serve as a completely free alternative of it, without any kind of usage limits. So, Polymaps allows users to display also different types of maps (satellite, road, terrain) and display different types of information on them (shapes and polygons, GeoJSON, markers, density data).

The main problem with Polymaps is that its official GitHub repository has not been updated since mid-2011, which suggests that the project has been abandoned and is no longer supported by the developers.

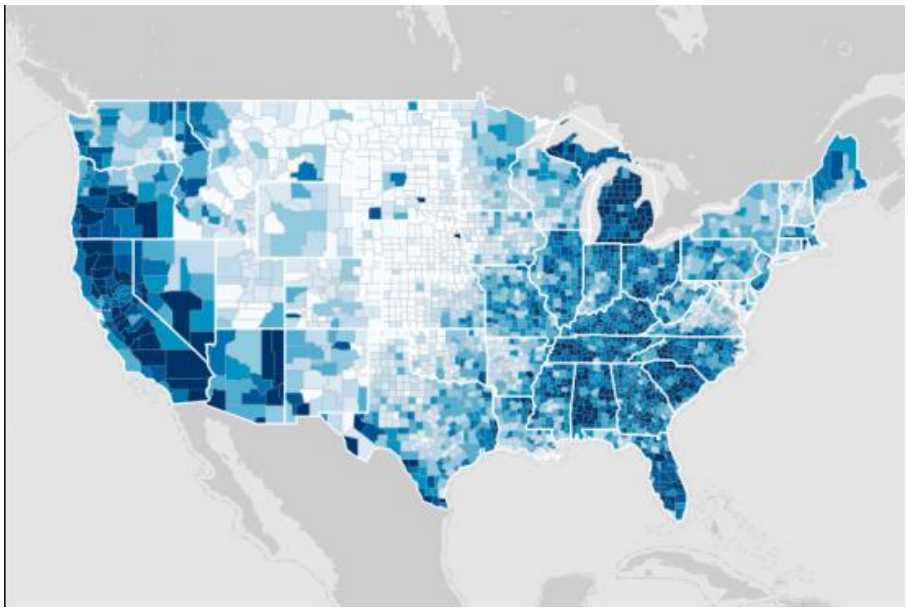


Figure 15 Example of an interactive map with Polymaps

Figure 16 Pavement quality in San Francisco using Polymaps

2.2.3 ArcGIS

ArcGIS [14] is a complete framework developed by Esri that provides a suite of tools for creating maps, compiling geographic information, creating or managing geographic databases, solving spatial analysis problems, or simply visualising information on maps. In the context of map use and creation, ArcGIS allows the generation of a wide variety of maps (Figure 17, Figure 18), which can be accessed from browsers, mobile apps, exported in high resolution to be printed, included on reports or simply embedded in other platforms. As a basis, the system has a comprehensive collection of maps of various types including topographic, terrain, hydrology, land use and geology maps.

Within the ArcGIS toolset, the ArcGIS API for JavaScript [15] provides similar functionalities to all of the technologies discussed in this chapter, including the possibility to draw polygons, shapes or markers, add interactive elements, popups or layers at different levels. However, despite being one of the most professional solutions available, to use the API it is necessary to create an ArcGIS developer account which has certain usage limits. In addition, it is a more complex option than other open-source libraries.

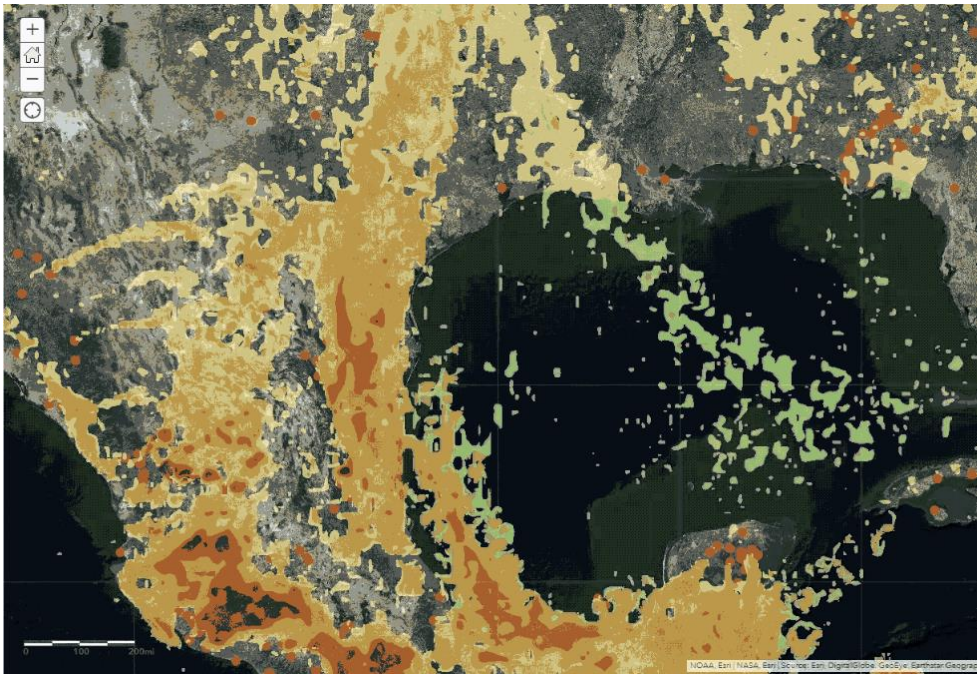


Figure 17 Example of map using ArcGIS

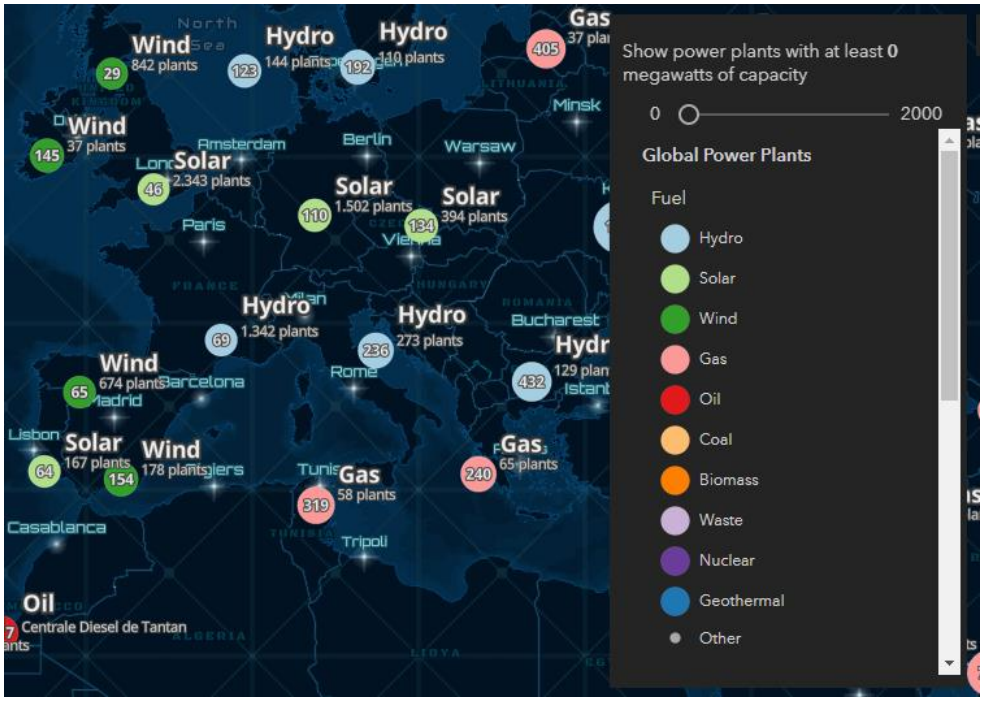


Figure 18 Interactive maps using ArcGIS

2.2.4 Mapbox GL JS

Mapbox GL JS [16] is a JavaScript library for building maps and web applications using Mapbox's modern mapping technology. Using OpenGL for rendering interactive vector mosaic maps (Figure 19) and styles, this library provides the following features:

- Geographic data visualisation and animation.
- Querying and filtering data on the map.
- Draw layers using different type of data (vector tiles, raster tiles, GeoJSON) and Mapbox styles.
- 3D visualisations and animations.
- Adding custom markers and interactive popups programatically.

In addition, the library provides plugins created and maintained by the Mapbox developer community that extend and enhance the core library features and functionalities. For example, there are plugins for the user interface to extend the map controls or the draw and exportation tools; plugins to enhance the map rendering or others that can help with the integration of Mapbox GL JS in external frameworks (React, Angular, Vue.js).

Like Google and ArcGIS tools and although the use of the Mapbox GL JS library is free, a user account is required to obtain the API keys to access and connect to other Mapbox tools that work underneath the GL library, such as the map provider. For this reason, the free version of Mapbox GL JS has a limit of 50.000 map loads per month (being a map load every time it is initialised on the screen).

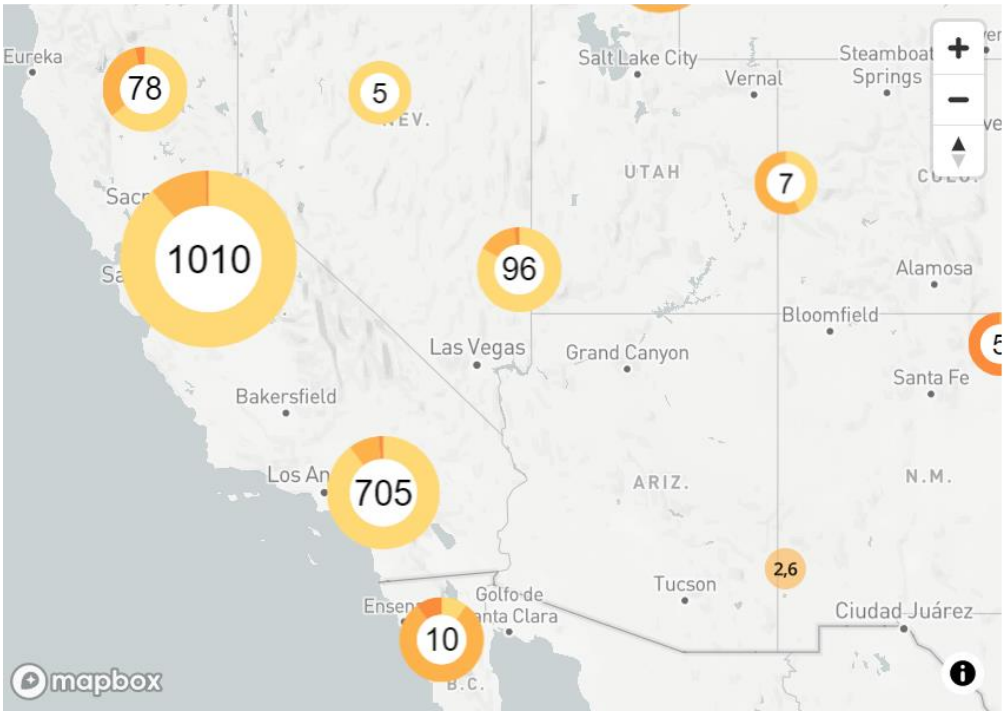


Figure 19 Example of cluster map using Mapbox GL JS

2.2.5 Apache ECharts

The open-source library ECharts also provides solutions for the representation of geographic information on maps with the possibility to generate fully interactive maps (Figure 20, Figure 21) with route and polygon tracing, heatmaps, marker placement, etc. In addition, the library supports different type of maps, but the main disadvantage is that these maps must be provided by the user through SVG or GeoJSON files that allow it to be rendered or drawn on screen respectively. The library also supports the use of satellite map services, by default, it only allows the use of Baidu Maps, a Chinese map service, but there are plugins developed by the community for the integration of other types of services such as Google Maps.

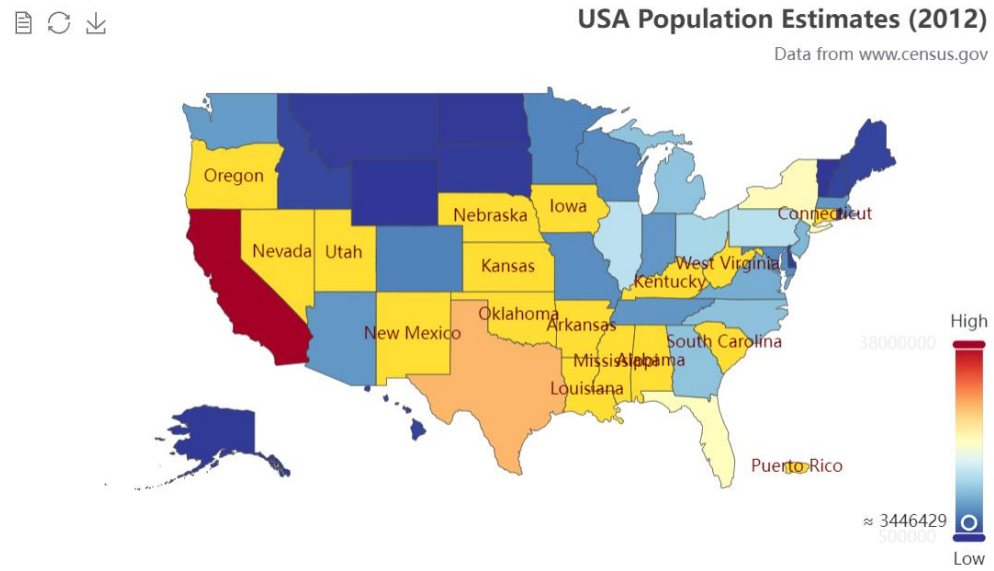


Figure 20 Interactive map using Apache ECharts

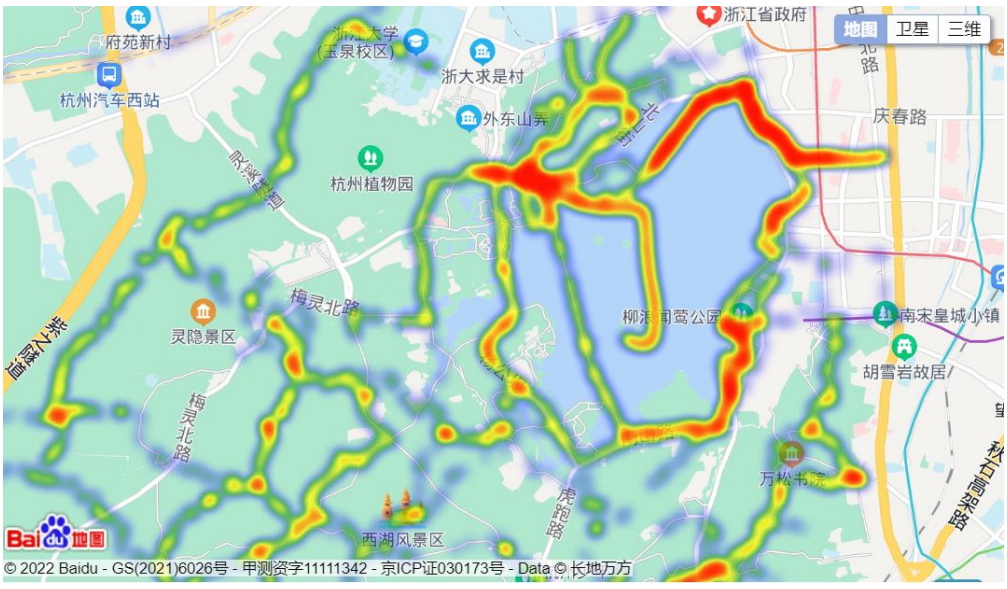


Figure 21 Heatmap using Apache ECharts

2.2.6 Leaflet

Leaflet [17] is an open-source JavaScript mapping library developed by Vladimir Agafonkin which allows creating fully interactive maps for mobile and web applications using OpenStreetMap (Figure 22). Among its main features, this library provides a light, simple, easy-to-use and user-friendly solution that also allows drawing polygons, shapes and custom markers on the map, loading GeoJSON data, adding interactive popups or customising the styles of the entire map content. In addition, given its open-source nature, the library can be complemented with third-party plugins that provide additional functionalities such as heatmaps, animate markers and maps adding a temporal dimension to them, load data from CSV files, etc.

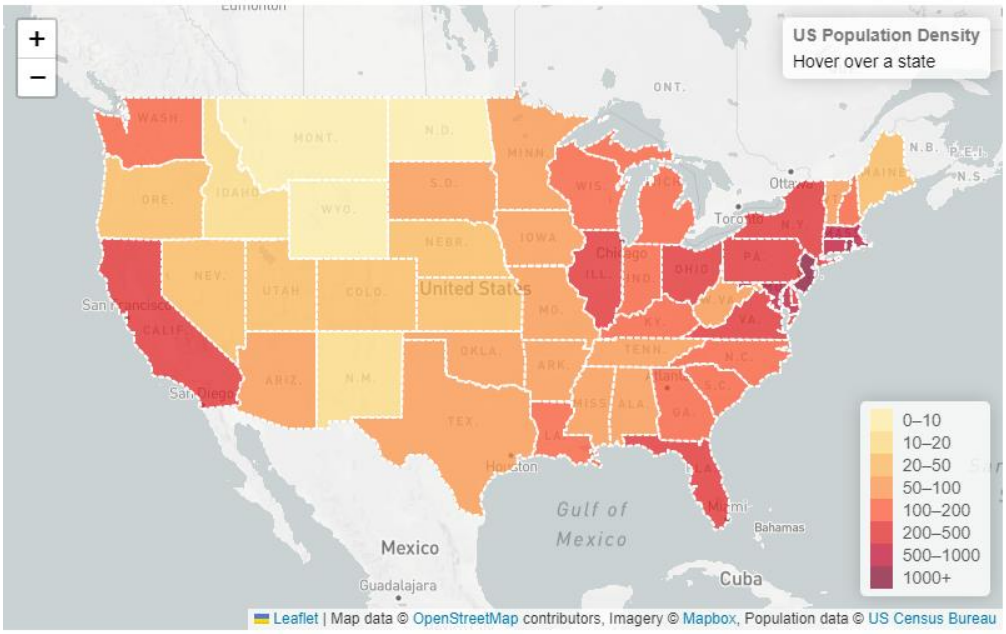


Figure 22 Interactive map using Leaflet

2.2.7 OpenLayers

OpenLayers [18] is another open-source JavaScript library to include a map component with georeferences (Figure 23) on any web page which, like the other libraries and tools above, allows the drawing of different elements on maps. The main difference is that it enables the use of several different mapping services as a basis, such as Google Maps, Bing Maps, Mapbox or OpenStreetMap among others. The main disadvantage is that OpenLayers diverges somewhat from the simplicity offered by other libraries as it seeks to go deeper into the use of geographic information and requires knowledge of GIS (Geographic Information System). In addition, and like Leaflet, OpenLayers can be complemented with multiple community-developed plugins that provide additional and advanced functionalities.

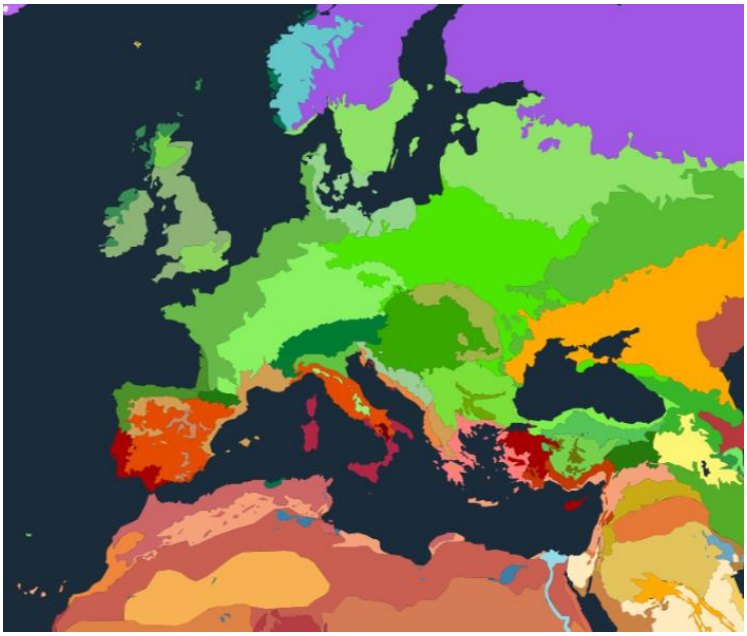


Figure 23 Example of GeoJSON data load on OpenLayers

2.2.8 OpenHeatMap

OpenHeatMap is a web application developed by Pete Warden that allows the generation of heatmaps with density data in geographic points (Figure 24). Its functioning is very simple, the user can include an Excel or CSV file with the density data in the correct format and the platform automatically generates a map that can be exported as an image or embedded in a website. However, this tool has two main problems, the first is that it is a web application accessible through a browser, it has no API and therefore could not be integrated into the AGRICORE platform and the second is that it has not been possible to access its official website during the analysis process. So, is possible that its developers have stopped supporting or maintaining it.

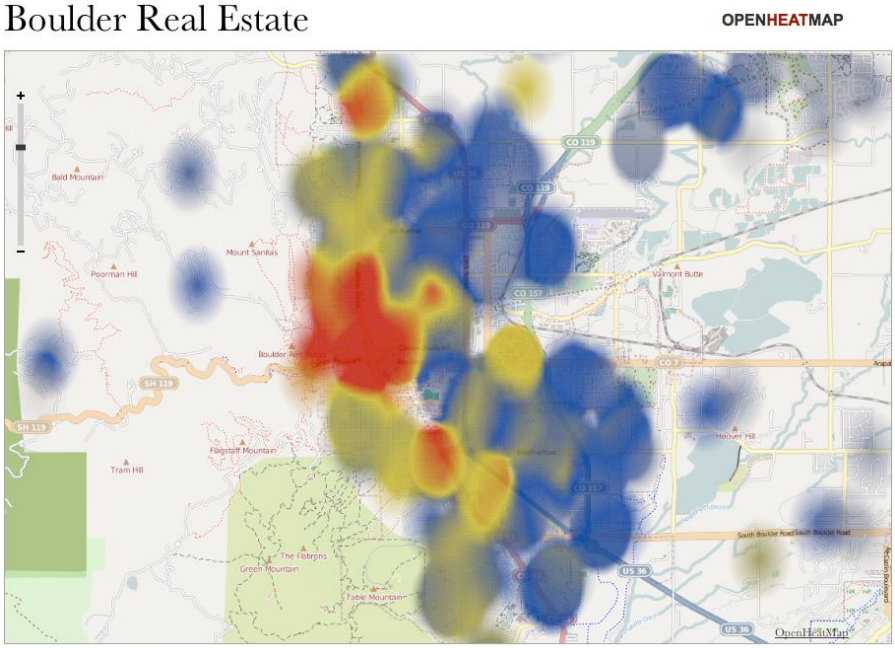


Figure 24 Heatmap using OpenHeatMap

2.2.9 TargetMap

TargetMap is another web application developed by MapGenia S.L that allows the creation of fully detailed maps and filling them in with the data of the user's choice (Figure 25). Among its main features, this tool allows the user to work by adding content to the map directly or by loading data from Excel files. In addition, it has functionalities to export the generated maps. However, like OpenHeatMap, this tool is a web application without a specific API to work on so, its integration with the AGRICORE GUI is not possible.

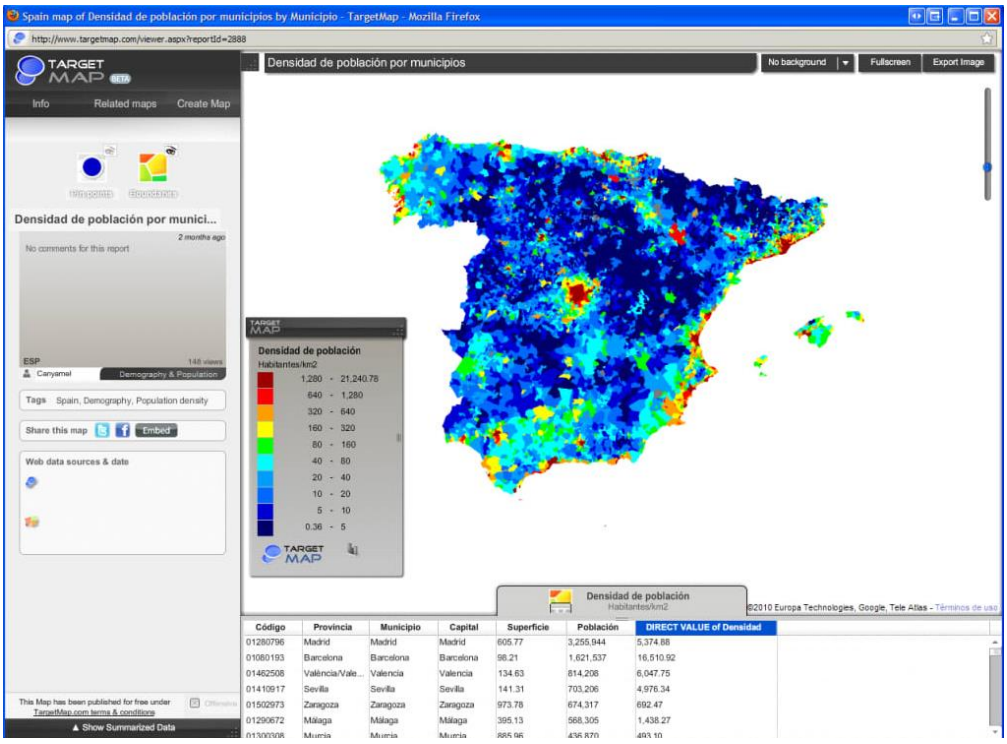


Figure 25 Interactive map using TargetMap

2.3 Results obtained

For the charting tools, as most of them provided similar features, the choice was focused on those that were more complete and offered more advanced functionalities. The following Table 1 shows a comparative summary of all the charting tools analysed:

Table 1: Comparative summary of all the charting tools analysed

Tool	Advantages	Disadvantages
Chart.js	• Simple and light. • Open-source. • Very popular with a great community of users and developers. • Has implementations for integration into different development languages and frameworks.	• Has only 8 chart types. • Although it covers the most common types (bar, line, pie), it can be limited in certain occasions. • Uses only HMTL5 canvas as rendering technology.
D3.js	• High customization and interactivity. • Supports a lot of different types of data for charting. • Special for advanced charting.	• It is a complex solution with a high learning curve. • Requires a lot of code to develop each chart completely.
Recharts	• Open-source. • Simple and easy integration with React.	• Restricted to React. • Has some dependencies on D3.js • Uses only SVG as rendering technology. • In terms of styles, by default, there are more interactive and visually more complete options.
Ngx-charts	• Open-source. • Simple and easy integration with Angular.	• Restricted to Angular. • Uses only SVG as rendering technology. • By default, it has basic but fully customisable styles.
Apache ECharts	• Provides a more complete set of charts which can be combined. • Can use both SVG and HTML5 canvas as rendering technologies. • Provides a toolbox which includes the possibility to export the charts as images. • Support both charting and mapping. • Allows the rendering of large data sources. • Provides features that enhance its use in mobile and PC applications.	• Does not have official implementations for its integration into frameworks such as Angular or React, but there are others developed by its user community.

For the mapping tools, OpenHeatMaps, TargetMap and Polymaps tools were automatically discarded because the first two are in themselves two web platforms, a priori, without an accessible API. So, the integration with the AGRICORE platform would therefore be impossible. In addition, the three tools do not seem to have been supported for several years, so the choice of Polymaps would be wrong for the future if this library is not going to receive any further updates. On the other hand, the Google, Mapbox and ArcGIS tools have the major drawback that their free use is limited. Although some of them provide quite acceptable limits, there is the

possibility of reaching them and losing access to their services from the AGRICORE platform. For this reason, the choice of mapping tool focused especially on open-source libraries.

The following Table 2 shows a comparative summary of all the mapping tools analysed:

Table 2: Comparative summary of all the mapping tools analysed

Tool	Advantages	Disadvantages
Google Maps JavaScript API	• Very popular, it is one of the most widely used map services. • Possibility to draw shapes, polygons, markers, heatmaps, load data from different formats.	• Being an external service, the free use of its API is limited. • Has no direct functionality to export the generated map as an image. Instead, other tools have to be used and the functionality must be developed (Google Maps Static API or html2canvas). • In the lack of plugins, for some functionalities, it may be necessary to develop them programmatically.
Polymaps	• Open-source. No usage limits. • Functionalities and features very similar to other alternatives that are not completely free.	• The project's official GitHub repository has not been updated since 2011.
ArcGIS API for JavaScript	• One of the most complete and professional solutions available. • Allows connections to other ArcGIS tools which enhances its functionalities. • The API has an endpoint to export the map in image format.	• Being an external service, the free use of its API is limited. • Has a higher learning curve. • Can be a complicated solution for simple maps.
Mapbox GL JS	• Carefully designed base maps. • Allows the rendering of large data sources. • The documentation is very complete. • Can be extended with official plugins developed by its user community. • Has implementations for its full integration in frameworks like Angular, React or Vue.js.	• Being an external service, the free use of its API is limited. • Can be a complicated solution for simple maps. • Has no direct functionality to export the generated map as an image, but it can be developed programmatically.
Apache ECharts	• Open-source. • Allows the rendering of large data sources. • Provides a toolbox which includes the possibility to export the maps as images. • Simple and very interactive solution.	• It is necessary to provide the maps to the library through GeoJSON files or SVG images. • By default, it only allows the use of Baidu Maps as map provider, but there are plugins developed by its user community to add Google Maps or other mapping services. • Has no official implementations for frameworks like Angular, but there are solutions developed by the user community.
Leaflet	• Open-source. • Light, simple and easy-to-use library. • Functionalities and features very similar to other alternatives that are not completely free.	• The documentation only contains basic examples. • The core functionalities only support the loading of data in GeoJSON format. Other formats can be provided by plugins.

	• Can be extended with plugins developed by its user community.	• Has no official implementations for frameworks like Angular, but there are solutions developed by the user community (ngx-leaflet).
OpenLayers	• Open-source. • Functionalities and features very similar to other alternatives that are not completely free. • Enables the use of several different mapping services as a basis (Google Maps, Bing Maps, Mapbox, OpenStreetMap) • Can be extended with plugins developed by its user community. • By default it supports the loading of data in different formats.	• Is likely to require knowledge of GIS. • API documentation could be very large.

Of all the possible options, **Apache ECharts** was finally chosen as the charting solution, since of all the tools selected, it is the one that provides the greatest number of functionalities that are well suited to the needs and requirements defined by the project, especially in terms of interactivity and the possibility of downloading the graphics as images. For maps, there were two options: either to use **Leaflet** together with plugins as an open-source option with no usage limits, or to use **Apache ECharts** for maps as well. Choosing the same library for both solutions greatly simplifies the project's tool dependencies, so it is a good option to lighten the deployment and maintenance of the tools. On the other hand, there are some drawbacks when using ECharts for the maps however, as will be explained in the next section, these problems have been addressed based on the requirements of the AGRICORE GUI.

Following the decision to use the open-source **Apache ECharts** library as the main solution to carry out the visualisation process in the AGRICORE interface and given the features highlighted in the comparison above, it was decided that the development of an own library for the generation of interactive graphics and maps was unnecessary due to the reasons explained below:

1. The fact of using a single library for the whole visualisation process eliminates the need to integrate different tools which, a priori, might not be compatible within a single tool in order to make them compatible. Therefore this layer of complexity is removed.
2. The development of a visualisation library that has a dependency on a single library whose purpose is the same, would only add greater complexity to the architecture of the project or application.
3. The ECharts API provides a large number of features that guarantee the coverage of the vast majority of use cases that may arise. Developing a layer on top of the library would mean, on the one hand, selecting a subset of these functionalities and adapting them to the needs of the project, a subset that could change over the life cycle of the project and would therefore require maintenance. Or, on the other hand, if all the functionalities are kept in this layer it would require an unnecessary work to develop something that can be used directly.
4. The ECharts library is maintained by **The Apache Software Foundation**. In case of errors or problems, users are guaranteed to have a dedicated development team and a great community of users.
5. Being an open-source library, it is possible to extend its functionalities using plugins developed by its community or to simply download the source code and make the necessary modifications.

6. Generally, it is not recommended to make direct use of vanilla JS libraries in frameworks such as Angular that use TypeScript as programming language, as these libraries work directly on the HTML DOM and can interfere with Angular's manipulation of it. For this reason, libraries usually have their corresponding implementation in the main web development frameworks, languages or tools which facilitate the integration. The development of an own library using ECharts would not imply the creation of a general-purpose library because it would be specifically linked to JavaScript projects. Therefore, if anyone wants to use this library in any other platform that does not use JavaScript, it would be necessary to develop and maintain its own implementation, something completely unnecessary since there are already implementations of this library in other languages such as Java or Python, developed by its user community.

3 Apache ECharts as library for data visualisation

Using Angular as the main framework upon Electron for the construction of the user interface of the AGRICORE platform, it will make use of the implementation of ECharts library for this framework (ngx-echarts [19]). At the time of writing this document, ngx-echarts is in its version 8, compatible with Angular versions equal to or higher than 13 and Apache ECharts versions equal to or higher than 3 (currently in its version 5). The following Table 3 shows a comparison of versioning between Angular, ngx-echarts and Apache ECharts:

Table 3: Versioning between Angular, Ngx-echarts and Apache ECharts

Angular version	ngx-echarts version	Apache ECharts version
>=13	v8.x	<=v5.x
>=11	v7.x	<=v5.x
>=10, <11	v6.x	<=v4.x
>=6, <10	v5.x	<=v4.x

From the above it can be concluded that is recommended to select the most recent versions of Angular in order to use Apache ECharts in its most stable version, v5.

The use of ECharts applied to the AGRICORE GUI can be divided according to the generation of charts or maps. For charting, there would not be any inconvenience or limitation in producing the charts so, the main objective in this case is to use as much of the functionalities available in the library API and allow as many types of charts as possible that will depend on the KPIs selected in the simulation setup process. In the case of maps, there are certain limitations therefore, the procedure to be applied will be as follows:

1. Since the library, by default, supports only SVG and GeoJSON maps and it is necessary for the user to provide the maps for rendering, the GeoJSON option will be used exclusively for the representation of maps. SVG maps have the disadvantage that when trying to place interactive elements on them, their internal positioning does not depend on geographic coordinates, making it very complex to work with them.
2. Since the library also, by default, provides support to the external Chinese mapping service Baidu, this will not be used as it does not provide support for languages other than Chinese. However, the possibility to add an external mapping service linked to the ECharts library in the future, if required, will be kept in order to represent information using geographic coordinates. In this case, it could be a matter of combining ECharts with Leaflet or Google Maps using plugins developed by the community, being the first option the most suitable due to its open-source character and unlimited use.
3. Taking into account the previous points and based on the requirements and needs regarding the representation of geographic information on the platform, it will be decided to use files in GeoJSON format to render the maps and make them interactive using Apache ECharts features. For this purpose, a collection of GeoJSON files will be provided to the library, to represent the map of the countries that compose the European Union and individual maps of each country present on it. To create this collection of GeoJSONs there are different tools and utilities available:
    a. A plugin developed by the ECharts user community which provides a list of 214 GeoJSON files from different countries and regions of the world [20].
    b. Web applications that allow users to generate GeoJSON files by selecting the countries they want to include or by drawing polygons to select the regions that wish to be added [21] [22].

c. Through official EU repositories, platforms with geographical databases or open-source project repositories on GitHub, which include collections of GeoJSON files to represent different countries at different levels of territorial or regional division [23].

Figure 26 and Figure 27 represent some screenshots of the utilities that allow building the collection of GeoJSON files for the representation of geographic information in the platform:

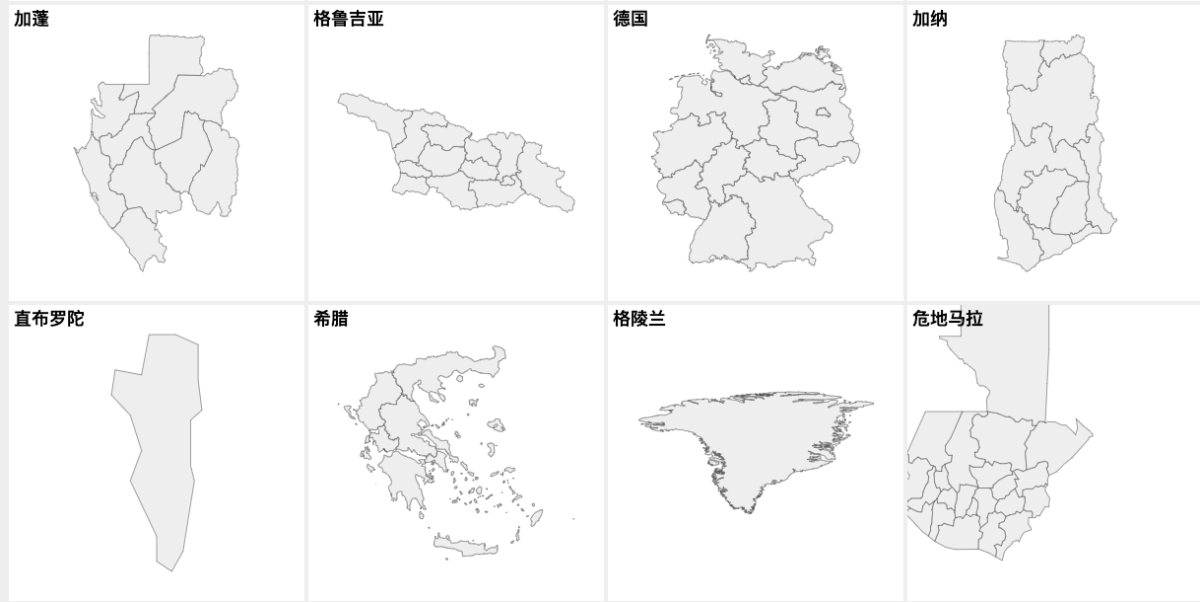


Figure 26 Example of maps given by ECharts countries plugin

Figure 27 Example of GeoJSON file generation using a web application

3.1 Developing charts and maps with Apache ECharts

The use of the ECharts API within the library is summarised in four objects or properties composed of a number of sub-properties each. The full documentation of each property and all usage details can be found in the official documentation of this tool [24]. The four objects are the following:

- **Echarts**: is a global object which allows actions to be performed outside the scope of a specific ECharts instance. Some of its most important functions are "init" which creates a new EchartsInstance and "registerMap" which allows to add a SVG or GeoJSON map as an asset to the library.
- **EchartsInstance**: represents the instance of an individual chart/map and the operations that can be performed on it. One of its main functions is "setOption" which is the main method of this library allowing the definition of the default configuration, data, animation, interaction and styles for the charts, maps, toolboxes, axis, etc.
- **Action**: list of actions supported by ECharts. For example:
    - Highlight or select specific data on the charts by default, applying a specific style for them.
    - Set the default zoom or timeline values.
- **Events**: comprises functions to deal with event-handling or user interaction with charts and maps. For example, mouse events, handling interactions when clicking on the legend, the toolbox, the timeline, zooming, etc.

It is important to clarify that when mentioning the use of the ECharts API, this is a procedure that is done entirely locally. The library is downloaded and installed as a project dependency module, so unlike Google Maps, Mapbox or ArcGIS no requests are made to external services/APIs using an authentication token. In ECharts it is a whole process that runs locally on the user's computer.

The basic process of generating graphics with the ECharts library has the following steps:

1. In the HTML file, a new section (div) is added to render the chart or map.

```
<div id="main" style="width: 600px;height:400px;"></div>
```

2. In the component file, the '"*div*" is assigned to ECharts, creating a new EchartsInstance on it.

```
var myChart = echarts.init(document.getElementById('main'));
```

3. (Only for maps) In the case of maps there is an intermediate step which consists of registering the asset file on SVG or GeoJSON format to be rendered on screen, transferring it to ECharts. In addditon, in this step it is possible to configure the positioning of territories such as islands or archipelagos on the map.

```
echarts.registerMap('USA', usaJson, {
    Alaska: {
      left: -131,
      top: 25,
      width: 15
    },
    Hawaii: {
      left: -110,
      top: 28,
      width: 5
    },
    'Puerto Rico': {
      left: -76,
      top: 26,
      width: 2
    }
});
```

4.

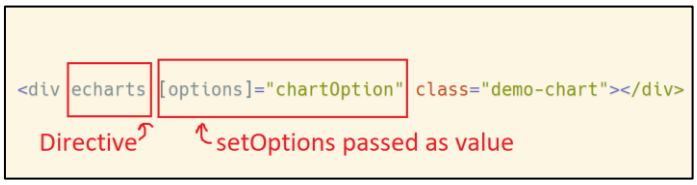
5. The initial configuration and style of the chart/map is defined including the axis, toolbox, animations, data, etc.

```
var option = {
    title: {
      text: 'ECharts Getting Started Example'
    },
    tooltip: {},
    legend: {
      data: ['sales']
    },
    xAxis: {
      data: ['Shirts', 'Cardigans', 'Chiffons', 'Pants', 'Heels', 'Socks']
    },
    yAxis: {},
    series: [
      {
        name: 'sales',
        type: 'bar',
        data: [5, 20, 36, 10, 10, 20]
      }
    ]
 };
```

6. Finally, the configuration set is passed to the ECharts instance via the "*setOptions*" method and the chart/map is rendered on screen.

```
myChart.setOption(option);
```

Using the Angular implementation simplifies the process above, as it relieves the developer of creating an ECharts instance to render every chart by using Angular directives, which are an Angular feature that allow changing the appearance or behaviour of DOM elements and components. The directive would eliminate the whole process of initialising the ECharts instance and only the options object would have to be set with the appropriate structure so that it can be received as an attribute by the directive to set the "*setOptions*" method.



Regarding the use of charts and maps in the AGRICORE interface, the construction of the charts will depend on the KPIs selected during the configuration phase of the simulation, as these will determine which type of charts will be the most appropriate to visualise the data, what information to include in the axes, legends, etc. Regarding the maps, following the methodology described above, interactive maps will be created according to the regions selected in the synthetic population chosen for the simulation. Thus, it can be maps representing the European Union or a specific country. Also, it is added the possibility to display and make interactive on the maps different regions at NUTS [25] level. Being possible the following levels:

- **NUTS 1**: major socio-economic regions.
- **NUTS 2**: basic regions for the application of regional policies.
- **NUTS 3**: small regions for specific diagnoses.

Thanks to GeoJSON files and the Apache ECharts library it is possible to create maps where these regions can be fully interactive, being able to be clicked on and show popups with related information extracted from the simulation results. Figure 28 and Figure 29 show some examples and test done during the development of this deliverable representing the use of different GeoJSON files to render an interactive map with the library.

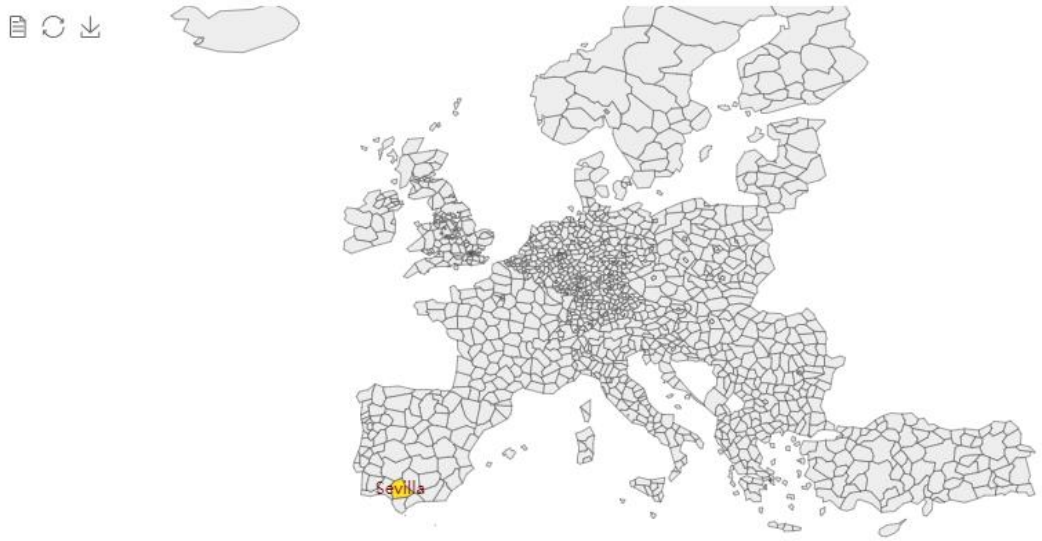


Figure 28 Example of an interactive ECharts map representing Europe NUTS values

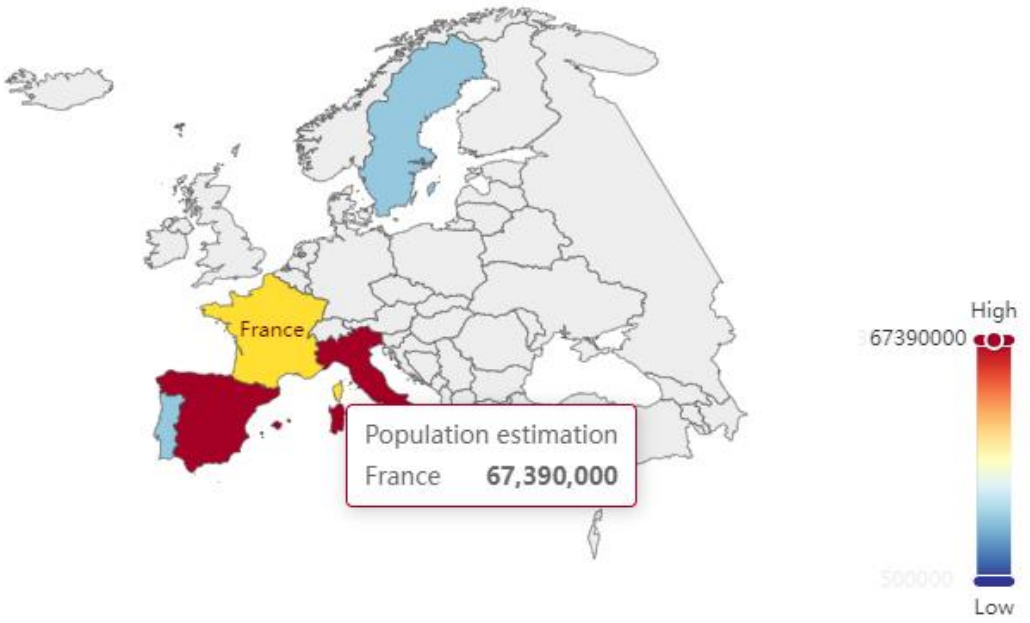


Figure 29 Example of an interactive ECharts map representing Europe countries

3.2 Prometheus monitoring system

In the description of the task involving this deliverable in the project's Grant Agreement document, it is specified the combination of the selected visualisation tool(s) with the use of Prometheus [26]. This tool is an open-source system commonly used for monitoring events and alerts on servers or applications, including those with complex microservices-based architectures. Prometheus receives data and metrics via streams and stores them in a database as time series in the format *<timestamp, value>*. Being the "*timestamp*" the instant of time at which the "*value*" was recorded. Typically the type of metrics to monitor are counters, gaugers, histograms and summaries and are related to CPU and memory usage, error diagnosis, service problems or any other property depending on the application or system to be monitored.

The Grant Agreement suggests making use of this tool in the context of **big data** however, there are some limitations and problems in this regard. Prometheus is a solution particularly useful for monitoring numerical series in real time and this clashes with the objectives of the project, since the AGRICORE GUI does not work with real-time data or data streams. The visualisation section in the platform takes the results of the completed simulations to represent them graphically, so it works with data already generated and stored.

However, by removing the real-time data component, Prometheus could be used for the visualisation of KPIs that may evolve over time, but this would only add more complexity to the platform because the Prometheus system would have to be integrated or connected with Apache ECharts for the representation of a limited number of charts or maps. In addition, the ECharts library already offers a solution for the representation of data that can evolve over time thanks to its timeline component and also supports the processing of large data sets using WebSockets, so the issue with big data would also be covered.

4 Conclusions and next steps

The deliverable D4.4 'Library for AGRICORE data visualisation purposes' is a continuation of the work developed in D4.3 'Validated design for the AGRICORE interface' and part of the next steps in the development of the AGRICORE GUI with deliverable D4.6 'AGRICORE interface'. This document provides the analysis carried out in order to identify the most suitable tools for the visualisation of the simulations results obtained in the AGRICORE main interface. The conclusion of this comparative analysis was to use the open-source Apache ECharts library as solution for both charts and maps, which provides AGRICORE with a high customisable and interactive tool to help users in the analysis of the data and results obtained. Although ECharts has some drawbacks when working with maps, the procedure for working with geographical data has been described in the previous chapters, procedure which eliminates the use of external services that may have future limitations of use but, at the same time, opens the door to the inclusion of possible plugins to extend its functionality.

In addition, following the decision to use a single tool for the complete visualisation of the data in the platform and given that it is open-source, the reasons why it has been decided not to build a dedicated library for the project for this purpose have been explained. Instead, a direct use of the library will be made which relieves from the need to implement additional functionalities or to perform maintenance on them.

Finally, the next steps involving the development of the AGRICORE interface include:

- The development of the user interface itself through the implementation of the designs and functionalities described in D4.3 and D4.4.
- The integration of Apache ECharts on the platform which will use Angular over Electron framework to build the user interface. Additionally, Ngx-echarts, the Apache ECharts implementation for Angular, will be used to simplify the integration and facilitate the use of the library.

5 References

[1] ^ O. Foundation, "Electron framework official site." [Online]. Available: https://www.electronjs.org/

[2] ^ Google, "Angular framework official site." [Online]. Available: https://angular.io/

[3] ^ M. Platforms, "React library official site." [Online]. Available: https://en.reactjs.org/

[4] ^ E. You, "Vue.js framework official site." [Online]. Available: https://vuejs.org/

[5] ^ C. j. community, "Charts.js library official site." [Online]. Available: https://www.chartjs.org/

[6] ^ C. j. community, "Charts.js working samples." [Online]. Available: https://www.chartjs.org/docs/latest/samples/information.html

[7] ^ D. j. community Mike Bostock Jeffrey Heer, Vadim Ogievetsky, "D3.js library official site." [Online]. Available: https://d3js.org/

[8] ^ R. Group, "Recharts library official site." [Online]. Available: https://recharts.org/en-US

[9] ^ Swimlane, "Ngx-charts library official site." [Online]. Available: https://swimlane.github.io/ngx-charts/#/ngx-charts/bar-vertical

[10] ^ T. A. S. Foundation, "Apache ECharts library official site." [Online]. Available: https://echarts.apache.org/en/index.html

[11] ^ T. A. S. Foundation, "Apache ECharts online builder." [Online]. Available: https://echarts.apache.org/en/builder.html

[12] ^ Google, "Google Maps JavaScript API documentation." [Online]. Available: https://developers.google.com/maps/documentation/javascript

[13] ^ S. SimpleGeo, "Polymaps library official site." [Online]. Available: http://polymaps.org/

[14] ^ Esri, "ArcGIS official site." [Online]. Available: https://www.esri.com/en-us/arcgis/about-arcgis/overview?rsource=%2Fsoftware%2Farcgis

[15] ^ Esri, "ArcGIS JavaScript API documentation." [Online]. Available: https://developers.arcgis.com/javascript/latest/

[16] ^ Mapbox, "Mapbox GL JS library official site." [Online]. Available: https://www.mapbox.com/mapbox-gljs

[17] ^ V. Agafonkin, "Leaflet library official site." [Online]. Available: https://leafletjs.com/

[18] ^ O. community, "OpenLayers library official site." [Online]. Available: https://openlayers.org/

[19] ^ Xieziyu, "Ngx-echarts. Apache ECharts implementation for Angular." [Online]. Available: https://www.npmjs.com/package/ngx-echarts

[20] ^ Jaska, "Echarts-countries-js. Apache ECharts plugin for countries and regions." [Online]. Available: https://echarts-maps.github.io/echarts-countries-js/

[21] ^ Mapbox, "Geoson.io tool official site." [Online]. Available: https://geojson.io/#map=2/20.0/0.0

[22] ^ Ashkyd, "Geojson-maps tool official site." [Online]. Available: https://geojson-maps.ash.ms/

[23] ^ Eurostat, "Geographical information and maps." [Online]. Available: https://ec.europa.eu/eurostat/web/gisco/geodata/reference-data/administrative-units-statistical-units/countries

[24] ^ T. A. S. Foundation, "Apache ECharts API official documentation." [Online]. Available: https://echarts.apache.org/en/api.html#echarts

[25] ^ Eurostat, "NUTS - Nomenclature of territorial units for statistics." [Online]. Available: https://ec.europa.eu/eurostat/web/nuts/background

[26] ^ C. N. C. Foundation, "Prometheus official site." [Online]. Available: https://prometheus.io/

Apart from the references mentioned along the document, the current deliverable took into account following inputs:

- AGRICORE Proposal: project proposes a novel tool for improving the current capacity to model policies dealing with agriculture by taking advantage of the latest progresses in modeling approaches and ICT.
- AGRICORE Grant Agreement ANNEX 1 Part A and B, Research and Innovation action, Number-816078: Official Grant Agreement of the AGRICORE project, which defined the terms and conditions of the project, as well as the main requirements of the project.

Deliverable Number	Deliverable Title	Lead beneficiary	Type	Dissemination Level	Due date	Reason
D4.3	Validated design for the AGRICORE interface	AAT	Report	Public	M27	Review the requirements defined and the designs realised for the AGRICORE interface
D6.1	AGRICORE Architecture and Interfaces	IDE	Report	Public	M24	Understand the connection between different modules of the project and the AGRICORE interface