Cartographier pour le web avec ‘bertin‘

Nicolas Lambert, Timothée Giraud, Matthieu Viry, Ronan Ysebaert

Cartographier pour le web avec `bertin`

Nicolas Lambert[1], Timothée Giraud[1], Matthieu Viry[1], Ronan Ysebaert[2]

1. UAR RIATE, CNRS
Université Paris Cité. Place Paul Ricœur. 75013 Paris
{nicolas.lambert@cnrs.fr, timothee.giraud@cnrs.fr, matthieu.viry@cnrs.fr}

2. UAR RIATE, Université Paris Cité
Université Paris Cité. Place Paul Ricœur. 75013 Paris
ronan.ysebeart@cnrs.fr

RÉSUMÉ. bertin est une bibliothèque JavaScript pour la réalisation de cartes thématiques pour le Web. La bibliothèque est bâtie principalement à partir de la bibliothèque JavaScript D3.js. Sa conception vise à permettre aux utilisateurs de créer rapidement des cartes thématiques interactives sans forcement connaître le langage JavaScript ni le framework de D3.js. Un "wrapper" de cette bibliothèque est également disponible dans les langages R et Python.

ABSTRACT. bertin is a JavaScript library for designing thematic maps for the web. The library is built mainly against the D3.js JavaScript library. Its purpose is to allow users to quickly create interactive thematic maps without necessarily knowing the JavaScript language or the D3.js framework. A wrapper of this library is also available in the R and Python languages.

MOTS-CLÉS : JAVASCRIPT, R, CARTOGRAPHIE, OBSERVABLE, VISUALISATION DE DONNÉES, GEOWEB, NOTEBOOKS

KEYWORDS: JAVASCRIPT, R, CARTOGRAPHY, OBSERVABLE, DATAVIZ, GEOWEB, NOTEBOOKS

1. Introduction

Des pionniers de la dataviz (Bertin, 1967 ; Tukey, 1977 ; Tufte *et al.*, 1983) à la révolution du Web, couplée à l'essor des démarches quantitatives (Tobler, 1970 notamment), la visualisation de données s'est peu à peu structurée en tant que pratique interdisciplinaire (Grandjean, 2022) qu'on regroupe généralement sous de terme de *visual studies*. Ces travaux anciens sont d'ailleurs très largement mobilisés aujourd'hui et nombre d'acteurs n'hésitent pas à remettre au goût du jour des instigateurs célèbres de la visualisation de données tels que Charles-Jospeh Minard (1781-1870), Emile Cheysson (1836-1910) ou W.E.B Du Bois (1868-1963) à l'aide de nouvelles technologies informatiques.

L'essor récent de la visualisation de données sur le Web a notamment été porté par les travaux de Mike Bostock en JavaScript et le développement de la bibliothèque D3.js (Bostock *et al.*, 2011). Cependant, contrairement à d'autres projets dans d'autres langages, le niveau d'abstraction relativement bas de cette bibliothèque ne vise pas à permettre de réaliser des cartes et des graphiques très rapidement mais plutôt de fournir un framework souple et ouvert pour la visualisation de données. Or cette capacité à produire des visualisations en quelques lignes de codes est très important pour articuler des analyses de données et des visualisations dans le cadre de *notebooks*. En proposant une syntaxe concise et des comportements par défaut, La bibliothèque *bertin*, présentée ici, entend répondre à cet enjeu.

2. Contexte et enjeux

2.1. Les notebooks

Les *notebooks* sont un exemple de programmation lettrée fournissant une manière de combiner langage naturel et langage informatique au sein d'un même document. Les *notebooks* sont fréquemment utilisés en sciences des données afin d'explorer ou d'analyser leur contenu et permettent de faciliter le partage mais également la reproductibilité des résultats et sorties graphiques qu'ils décrivent. Leur aspect interactif leur permet généralement de régénérer "à la volée" les différents calculs et les différentes sorties graphiques qu'ils contiennent, offrant ainsi un environnement particulièrement adapté à l'exploration de données. Dans une démarche de reproductibilité, ils sont aussi un moyen de documenter un travail scientifique (Shen, 2014).

Ce type de programmation n'est pas totalement nouveau, il a notamment été formalisé par Donald Knuth dès 1984 et des fonctionnalités *notebook* se développent dès la fin des années 80 dans plusieurs systèmes de calcul formel (MathCAD en 1987, *Mathematica* en 1988, *Mapple* en 1989). Il connaît un essor important au cours des années 2000 puis 2010 dans le domaine de la science des données, en particulier dans les solutions opensources. Les fonctionnalités *notebook* sont ainsi

introduites dans *IPython* (Pérez *et al.*, 2007) v0.12 en décembre 2011 et la première version de *knitr* (Xie, 2014) est publiée en 2012.

Le *notebook* se situe ainsi au centre d'un système socio-technique, comme un objet porteur de différents objectifs et de différentes fonctions : tout d'abord, et pour prendre le cas de *bertin*, par l'intégration de la réalisation cartographique au sein d'un *workflow* reproductible. Il se matérialise en amont par l'import des données et leur mise en forme, puis par l'explicitation des intentions de l'auteur et du code qui en résulte et conduisent à une représentation graphique originale.

2.2. Pourquoi JavaScript ?

De nombreux langages paraissent naturels pour l'analyse et la visualisation de données. C'est notamment le cas de Python et R, aussi largement utilisés à ces fins. Mais au regard de ces deux langages, le JavaScript n'est pas dénué d'atouts (Bostock, 2021). Tout d'abord, JavaScript est le langage du Web. Il tourne dans tous les navigateurs Web et ne nécessite aucune installation. Et si ce langage n'a pas été conçu au départ pour cela, de nombreuses bibliothèques ont été développées pour manipuler les données (*Arquero*, *tidy.js*, etc.), démontrant ainsi que ce langage peut être étendu pour mieux satisfaire les besoins inhérents aux *visual studies*. Par ailleurs, avec des bibliothèques comme D3.js[1], qui permet de manipuler canvas et SVG, JavaScript est probablement déjà aujourd'hui le langage le plus adapté pour la visualisation de données 2D. Le choix du Web est également particulièrement intéressant pour des raisons d'interopérabilité, permettant au développeur de concevoir des applications qui ne dépendent d'aucun système d'exploitation mais nécessitent uniquement un navigateur Web. Enfin, avec *WebGPU*[2] et *WebAssembly*[3], l'avenir de JavaScript semble prometteur.

Nous nous inscrivons donc ici dans une transition déjà identifiée par des auteurs du champ tels que Roth (2015): *"a broad transition in client-side web mapping away from standalone, proprietary technologies (e.g., Adobe Flash) and towards open technologies that leverage the HTML, CSS, SVG, and XML web standards (the Open Web Platform) and the JavaScript programming language"*. Cette tendance, qui n'est pas spécifique à la visualisation de données, s'est largement accélérée depuis les années 2010 en raison des améliorations de performances apportées par les navigateurs Web et leurs moteurs JavaScript (en effet la compilation *Just In Time* a été introduite dès 2008 dans les moteurs *JavaScript V8* et *SpiderMonkey*, témoignant des efforts mis en œuvre pour améliorer leurs performances). Cette tendance s'inscrit dans un processus amorcé dès les débuts du Web et déjà identifié comme pouvant améliorer l'acceptation des utilisateurs tout en évitant les écueils classiques de la distribution de logiciels (Rice et al., 1996).

[1] https://github.com/d3/d3
[2] https://www.w3.org/TR/webgpu
[3] https://www.w3.org/wasm/

2.3. Travaux connexes

Il existe de nombreuses bibliothèques JavaScript permettant de visualiser des données géographiques (figure 1). On peut néanmoins différencier les bibliothèques plutôt dédiées à la visualisation interactive des données géographiques, comme *MapLibre*, *Leaflet* ou encore *OpenLayers* qui contiennent plusieurs fonctionnalités permettant de zoomer, afficher/masquer des couches, créer des animations ; aux bibliothèques davantage orientées vers la visualisation des données statistiques suivant l'usage d'une grammaire graphique comme Observable Plot[4] ou Vega-Lite (Satyanarayan, 2016).

Fonctionnalités		OpenLayers	MapLibre GL	Leaflet	Deck.gl	Obervable Plot	Vega-Lite	bertin
Première *release*		2006	2020	2011	2015	2018	2013	2021
Famille		Carto interactive	Carto interactive	Carto interactive	Exploration de données	Visu de données	Graphique interactifs « ggplot »	Carto thématique pour le Web
Interactivitée	Zoom	Oui	Oui	Oui	Oui	Non	Non	Non
	Tooltips	Oui	Oui	Oui	Oui	Non	Oui	Oui
	Animations	Oui	Oui	Plugin	Oui	Non	Non	Non
	Élément HTML	Canvas (WebGL optionnellement)	Canvas (WebGL)	Canvas	Canvas (WebGL)	SVG	SVG	SVG
Symbologie[5]		+	++ (3D)	+	++ (3D)	++	+	+++
Habillagee	Labels	Oui	Oui	Plugin	Oui	Oui	Oui	Oui
	Projections	++	Mercator	Mercator	Mercator	+++	++	+++
	Minimap	Oui	Oui	Plugin	Oui	Non	Non	Oui
	Tuiles raster	Oui	Oui	Oui	Oui	Non	Non	Oui
	Graticules	Oui	Plugin	Plugin	Non	Non	Non	Oui
	Légende	Plugin	Plugin	Oui	Non	Oui	Oui	Oui

FIGURE *1. tableau comparatif des bibliothèques JavaScript de cartographie*

La bibliothèque *bertin* est d'avantage à rapprocher de ces deux dernières, qui partagent un peu la même philosophie avec des références claires à l'écosystème D3.js. *bertin* se démarque néanmoins en étant spécifiquement dédiée à la cartographie thématique. Bien plus que les autres bibliothèques précédemment citées, l'effort est apporté à la multiplicité des fonctions de représentations proposées, suivant les préceptes de sémiologie graphique initiés par Jacques Bertin

[4] https://github.com/observablehq/plot
[5] Variété des options de symbologie proposées

et la mise en scène de la carte (habillage de la carte, gestion des projections, paramétrage avancé de la légende).

3. La bibliothèque *bertin*

bertin est une bibliothèque JavaScript consacrée à la réalisation de cartes statistiques vectorielles dont le développement a débuté en novembre 2021. La bibliothèque a été nommée ainsi en hommage au géographe français Jacques Bertin (1918 - 2010) et ses travaux fondateurs sur la sémiologique graphique (Bertin, 1967 ; Bertin, 1973) dont l'influence est aujourd'hui encore majeure dans le domaine de la visualisation de données. Néanmoins, la bibliothèque ne propose en aucun cas un décalque de la sémiologie graphique de Jacques Bertin, mais plutôt des méthodes de représentations classiques telles que présentées dans les différents manuels de cartographie (Lambert et Zanin, 2020).

3.1. Description technique

La bibliothèque repose sur de nombreuses dépendances open source, principalement liées à l'écosystème D3.js mais également sur des bibliothèques spatiales telles que *jsts*[6], *turf*[7] et *proj4js*[8]. Elle est publiée sous licence MIT et en développement actif. 152 versions ont été publiées entre novembre 2021 et novembre 2022 (version 1.5.9). La bibliothèque *bertin* est maintenue par une personne, mais compte à ce jour 7 contributeurs externes. Le volume du code représente 8535 lignes de codes dans la dernière version soit 272 Ko. Le code source et la documentions sont accessibles sur github : https://github.com/neocarto/bertin

3.2. Principes

Le principe de la bibliothèque *bertin* est de proposer un outil permettant de réaliser rapidement des cartes thématiques variées sans faire appel à la programmation en JavaScript ni directement à la bibliothèque D3.js.

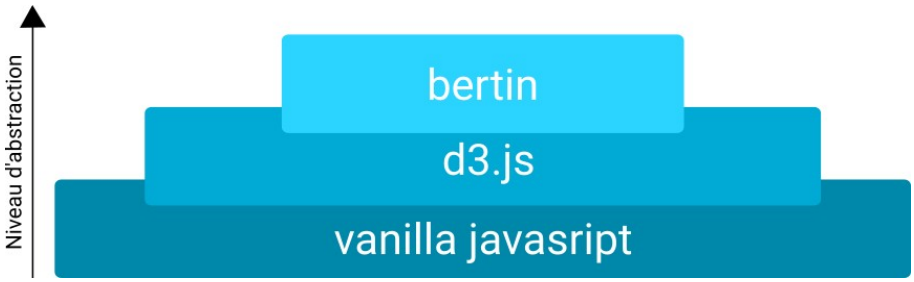


FIGURE 2. *niveau d'abstraction de la bibliothèque bertin*

[6] https://github.com/bjornharrtell/jsts
[7] https://github.com/Turfjs/turf
[8] https://github.com/proj4js/proj4js

Le but est d'avoir un outil de cartographie dans l'écosystème JavaScript directement appréhendable par des étudiants en géographie sans compétences préalables en développement Web, mobilisable dans des *notebooks*. Le niveau d'abstraction de la bibliothèque (figure 2) est donc volontairement très élevé, et mobilise une grammaire qui fait directement référence au vocabulaire métier de la cartographie thématique.

La principale fonction de la bibliothèque est *draw()* qui permet de dessiner des cartes thématiques (figure 3). Le fonction prend comme seul argument un objet JavaScript contenant l'ensemble des informations nécessaires au dessin de la carte. L'objet se divise en 2 parties : des paramètres généraux (propriété "params" de l'objet donné en argument, optionnel) définissant l'emprise de la carte, sa projection, sa couleur de fond, ses marges et sa taille ; et un *array* (propriété "layers" de l'objet donné en argument) contenant l'ensemble des couches à afficher. Chaque couche est définie par un type spécifique (bubble, spikes, header, minimap...) qui détermine le type de rendu. L'ordre d'écriture des couches détermine l'ordre d'affichage .

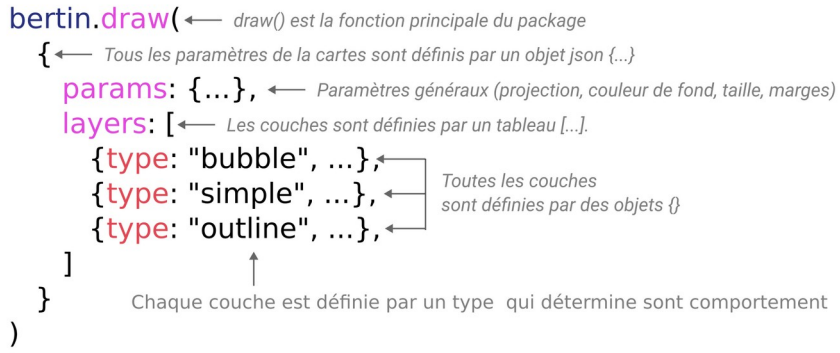


FIGURE *3. la fonction draw()*

Un type custom permet également d'intégrer à la carte tout type de contenu via des fonctions spécifiques construites par l'utilisateur, permettant ainsi pleinement l'interopérabilité de la bibliothèque avec D3.js notamment. La fonction renvoie un élément HTML de type SVG et éventuellement un générateur qui produit une nouvelle valeur chaque fois que l'utilisateur interagit avec la carte. Plusieurs paramètres par défaut sont définis de façon empirique pour déterminer un comportement par défaut optimal et ainsi alléger la syntaxe minimale nécessaire à la génération d'une carte.

Au final, la bibliothèque *bertin* permet de construire des visualisations cartographiques finalisées directement publiables sur des sites Web ou

téléchargeables au format SVG. Ci-dessous, un exemple de carte réalisée avec *bertin* pour le journal l'Humanité.fr[9] (figure 4).

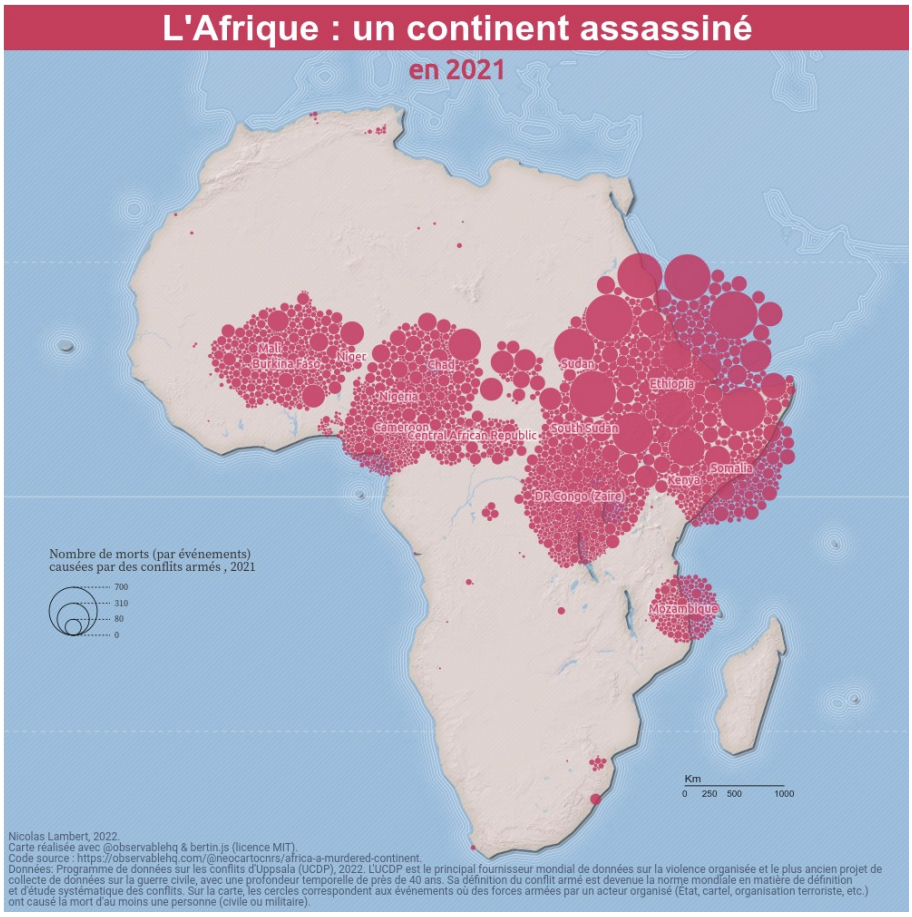


FIGURE *4. carte « finalisée » réalisée avec bertin*

3.3. Types de cartes

Les types de couches définies dans la bibliothèque *bertin* peuvent être séparées en 2 catégories : les couches de symbologie d'une part et les couches d'habillage d'autre part (figure 5). Les couches de symbologie prennent systématiquement en entrée un fichier GeoJSON contenant les données statistiques et les géométries. Les autres paramètres permettent de définir la personnalisation de la couche à afficher (couleur, taille du symbole, épaisseur des traits, etc.). Les types d'habillages ont principalement un rôle esthétique : ombrage, graticule, hachures, tuiles raster…

[9] https://www.humanite.fr/en-debat/regard-de-cartographe/l-afrique-un-continent-assassine-775086

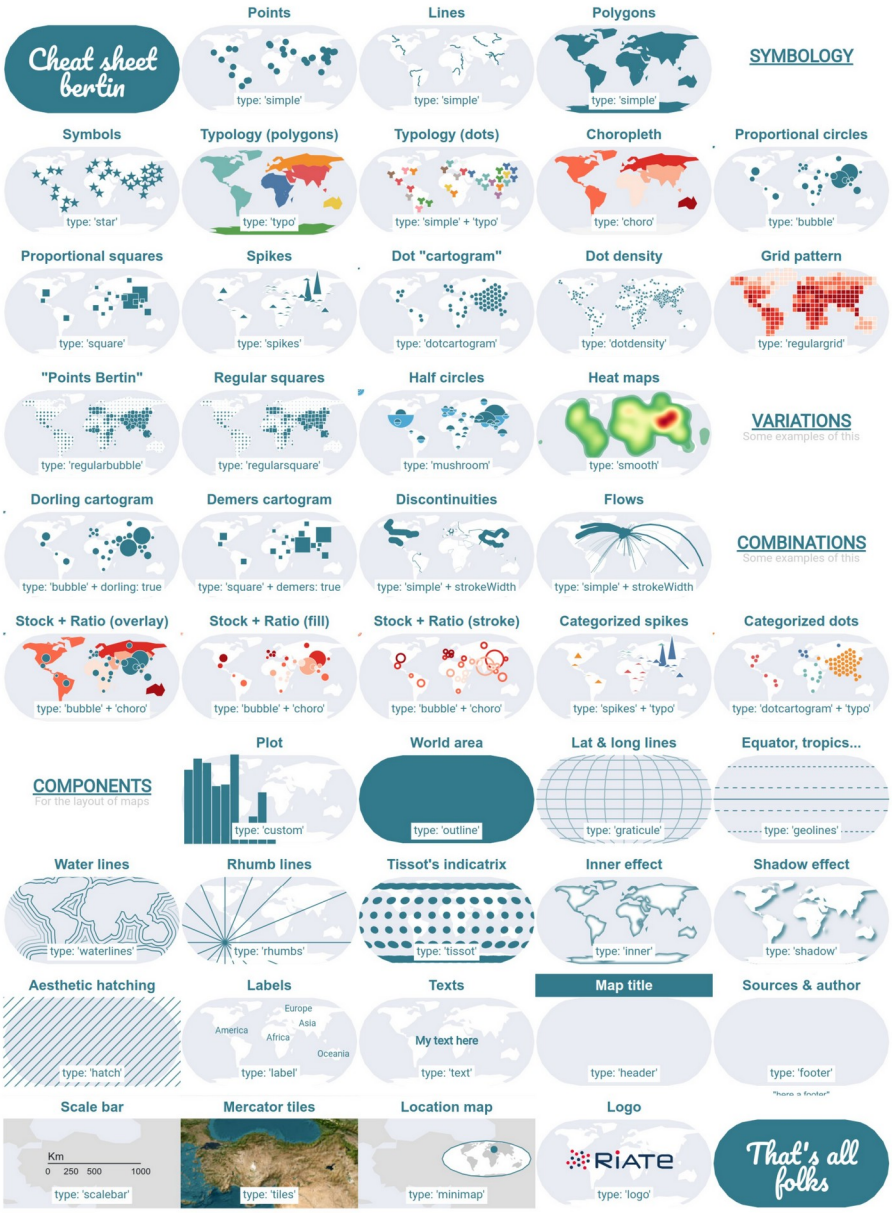


FIGURE 5. *les types de couches*

3.4. Une documentation abondante pour une diffusion large

En sus de la documentation sur Github et npm, *bertin* est agrémenté d'une collection Observable[10] constituée d'une trentaine de *notebooks* qui permettent d'introduire la bibliothèque par une série de cas d'usage. Ils présentent la philosophie et les principales fonctionnalités de la bibliothèque, en jouant de façon didactique sur les différents arguments des fonctions proposées. Ces *notebooks* rappellent à la fois la façon d'initier une représentation (import des données, jointures attributaires), de jouer sur la symbologie, en combinant parfois plusieurs propositions ("stock + typo") et de mettre en œuvre des éléments avancés de mise en page (afficher des tuiles raster, waterlines, minimap, tooltips, labels).

Au final, cet important travail de documentation vise à favoriser l'appropriation de *bertin* par un public pas nécessairement familier du langage JavaScript et du framework D3.js ; et rendre ainsi possible son usage par une multiplicité d'utilisateurs dans des contextes de création très variés.

4. Contexte d'utilisation

4.1. Notebooks (Observable et Quarto)

Comme n'importe quelle bibliothèque JavaScript front-end, *bertin* peut être directement importé et utilisé dans une page Web pour y insérer une carte interactive. Mais son utilisation fait particulièrement sens dans le cadre de *notebooks Observable* (Perkel, 2021), la bibliothèque étant pensée pour s'intégrer parfaitement avec cet écosystème, et notamment son aspect réactif. Les cartes dessinées avec *bertin* peuvent fonctionner comme des générateurs. Avec l'ajout du paramètre *viewof*, chaque couche de la carte peut servir d'*input* pour communiquer avec d'autres éléments du *notebook*. Cette fonctionnalité est particulièrement adaptée à la réalisation de tableaux de bord (ou *dashboards*) et proposer au lecteur de la carte interactive qui en résulte d'être acteur de son exploration et son interprétation.

Nous retrouvons la même logique dans Quarto (Allaire, J. *et al.*, 2022), avec la possibilité supplémentaire d'articuler des traitements et des analyses en R ou en Python avec des visualisations en JavaScript. Par ailleurs, l'aspect réactif de l'observable JavaScript et le système de *panels* proposé par Quarto, permet également de réaliser rapidement et avec très peu de code, des *single page applications* basées sur *bertin*.

4.2. R

Nous travaillons également à rendre la bibliothèque *bertin* disponible dans le logiciel libre R (R Core Team, 2022). Le logiciel R est un logiciel et un langage de programmation particulièrement adapté à la création de chaîne de traitements reproductibles. Son écosystème spatial est à la fois riche et robuste (Bivand, 2021) et

[10] https://observablehq.com/collection/@neocartocnrs/bertin

il est déjà assez facile d'y manipuler, traiter et représenter des données géographiques. Le système d'extensions du logiciel R (les *packages*) permet d'enrichir les fonctionnalités du logiciel. Ces extensions peuvent notamment être des *wrappers* autour de bibliothèques écrites dans d'autres langages, par exemple en C, C++ ou JavaScript. Le package *sf* (Pebesma, 2018) est le package de référence pour réaliser les opérations d'import, de manipulation et de géotraitement. Ce package sert de fondation à plusieurs packages de cartographie thématique tels que *tmap* (Tennekes, 2018) et *mapsf* (Giraud, 2022) mais également à d'autres packages (*leaflet*, *mapview*, *mapdeck*) qui sont eux dédiés à la création de cartes interactives (*slippy maps*) et s'appuient sur des bibliothèques JavaScript pour fonctionner.

Nous avons donc créé le package *bertin* (Giraud et Lambert, 2023) qui sert d'interface entre R et la bibliothèque JavaScript. Ce faisant nous rendons disponible directement dans R les représentations offertes par *bertin* (JavaScript). Ce portage en R permet de créer les cartes de qualité en SVG ou en HTML directement dans une chaîne de traitement reproductible. D'un point de vue plus technique, le package *bertin* s'appuie sur le package *htmlwidgets* (Vaidyanathan *et al.*, 2023) dont le but est de créer des *widgets* en HTML et sur *sf* pour s'intégrer facilement dans l'écosystème des package pré-existant. Nous avons choisi de proposer chaque types de représentation dans des fonctions distinctes, cette méthode permet d'exposer la documentation du package de manière plus détaillée, plus intégrée et plus idiomatique. Par ailleurs ce package nous permet d'intégrer facilement *bertin* dans des applications *shiny* (Chang *et al.*, 2022) et de proposer des applications interactives de cartographie thématique sans maîtriser le JavaScript. Des tests sont actuellement en cours pour valider l'utilisation de *bertin* avec *shiny*.

4.2. Python / Jupyter

Enfin, des utilisateurs de *bertin* contribuent également à rendre cette bibliothèque disponible dans un environnement Python, et plus particulièrement dans l'environnement interactif Jupyter (Kluyver *et al.*, 2016) qui permet notamment de réaliser des *notebooks* en Python. Ainsi l'extension *ipybertin*[11] permet d'appeler facilement la bibliothèque *bertin* depuis des *notebooks* interactifs écrit en Python. *ipybertin* utilise le mécanisme d'extension proposé par Jupyter Notebook et le widget *DOMWidget* proposé par *ipywidget*, la bibliothèque de références proposant des *widgets* HTML interactifs pour les *notebooks* Jupyter.

Comme dans le cas du package R *bertin*, *ipybertin* permet de créer des cartes thématiques mobilisant toutes les fonctionnalités de *bertin* et ce sans saisir le moindre code JavaScript. Toutefois, contrairement au package R *bertin*, l'API de *ipybertin* suit l'API proposée par *bertin* en exposant un simple objet *Map* encapsulant la fonction *draw* et acceptant un dictionnaire Python (un type de données similaire aux objets JavaScript attendus par *bertin*) contenant les propriétés *params* et *layers*.

[11] https://github.com/davidbrochart/ipybertin

5. Conclusion

Alors que la philosophie de D3.js est de mettre à disposition un framework complet permettant de développer des visualisations de données sur mesure pour le Web, la bibliothèque *bertin* permet de produire rapidement un certain nombre de types de cartes interactives dans une syntaxe concise. Sans chercher à remplacer d'autres projets JavaScript similaires comme *Observable Plot* ou *Vega-Lite*, la bibliothèque *bertin* vise à contribuer, en complémentarité, à la constitution d'un écosystème spatial en JavaScript particulièrement pertinent pour les *visual studies* via des *notebooks*. Cet écosystème semble se constituer autour de bibliothèques de plutôt haut niveau permettant d'effectuer des tâches plus ou moins complexes en une ou quelques lignes des tâches. D'autres travaux vont dans ce sens (*geotoolbox*, *statsbreaks*, *go-cart-wasm*...).

Bibliographie non numérotée et références

Allaire J., Teague C., Scheidegger C., Xie Y., Dervieux C., (2022). Quarto (Version 1.2) [Computer software]. https://doi.org/10.5281/zenodo.5960048

Bertin J. (1967), Sémiologie graphique. Les diagrammes. Les réseaux. Les cartes, Paris/La Haye, Mouton ; Paris, Gauthier-Villars.

Bertin J. (1973), Sémiologie graphique. Les diagrammes, les réseaux, les cartes, Paris/La Haye, Mouton ; Paris, Gauthier-Villars.

Bivand R.S. (2021). Progress in the R ecosystem for representing and handling spatial data. Journal of Geographical Systems, 23(4), 515-546. https://doi.org/10.1007/s10109-020-00336-0

Bostock M. (2021). Script for Data Analysis, https://towardsdatascience.com/javascript-for-data-analysis-2e8e7dbf63a7

Bostock M., Ogievetsky V., Heer J. (2011). D³ Data-Driven Documents, IEEE, Volume: 17, https://ieeexplore.ieee.org/abstract/document/6064996

Chang W., Cheng J., Allaire J., Sievert C., Schloerke B., Xie Y., Allen J., McPherson J., Dipert A., Borges B., (2022). _shiny: Web Application Framework for R_. R package version 1.7.4, <https://CRAN.R-project.org/package=shiny>.

Giraud T. (2022). mapsf: Thematic Cartography. https://CRAN.R-project.org/package=mapsf

Giraud T., Lambert N., (2023). bertin: Thematic Cartography with 'bertin.js'_. https://github.com/riatelab/bertin/, https://riatelab.github.io/bertin/.

Grandjean M. (2022). La visualisation de données, entre usages démonstratifs et heuristiques.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Willing, C. (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt (Eds.), Positioning and Power in Academic Publishing: Players, Agents and Agendas (pp. 87–90).

Lambert N., Zanin C., (2020). Practical handbook of thematic cartography: principles, methods, and applications. CRC Press

Pebesma E. (2018). Simple Features for R: Standardized Support for Spatial Vector Data. The R Journal 10 (1), 439-446, https://doi.org/10.32614/RJ-2018-009

Pérez F., Granger B.E., (2007). IPython: A System for Interactive Scientific Computing, Computing in Science and Engineering, vol. 9, no. 3, pp. 21-29, doi:10.1109/MCSE.2007.53. URL: https://ipython.org

Perkel J.M. (2021). Reactive, reproducible, collaborative: computational notebooks evolve, https://www.nature.com/articles/d41586-021-01174-w

R Core Team. (2022). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rice J., Farquhar A., Piernot P., Gruber T., (1996). "Using the Web instead of a window system". In: Proceedings of the SIGCHI conference on Human factors in computing systems common ground - CHI '96. the SIGCHI conference. Vancouver, British Columbia, Canada: ACM Press, pp. 103–110. isbn: 978-0-89791-777-3. doi: 10.1145/238386.238442.

Roth R.E., Donohue R.G., Sack C.M., Wallace T.R., Buckingham T.M.A., (2015). "A Process for Keeping Pace with Evolving Web Mapping Technologies". In: Cartographic Perspectives 78, pp. 25–52. issn: 1048-9053. doi: 10.14714/CP78.1273. url: https://cartographicperspectives.org/index.php/journal/article/view/cp78-roth-et-al

Satyanarayan A., Moritz D., Wongsuphasawat K., Heer J. (2016). Vega-lite: A grammar of interactive graphics. IEEE transactions on visualization and computer graphics, 23(1), 341-350. https://idl.cs.washington.edu/files/2017-VegaLite-InfoVis.pdf https://vega.github.io/vega/about/vega-and-d3/

Shen H. (2014). Interactive notebooks: Sharing the code. Nature, 515(7525), 151-152.

Tennekes M. (2018). "tmap: Thematic Maps in R." _Journal of Statistical Software_, *84*(6), 1-39. doi:10.18637/jss.v084.i06 <https://doi.org/10.18637/jss.v084.i06>.

Tobler W.R. (1970). A computer movie simulating urban growth in the Detroit region. Economic geography, 46(sup1), 234-240.

Tufte E., Graves-Morris P., (1983). The visual display of quantitative information. Diagrammatik-Reader. Grundlegende Texte aus Theorie und Geschichte. Berlin: De Gruyter, 219-230.

Tukey J.W. (1977). Exploratory data analysis (Vol. 2, pp. 131-160).

Vaidyanathan R., Xie Y., Allaire J., Cheng J., Sievert C., Russell K., (2023). _htmlwidgets: HTML Widgets for R_. R package version 1.6.1, <https://CRAN.R-project.org/package=htmlwidgets>.

Xie Y. (2014). "knitr: A Comprehensive Tool for Reproducible Research in R." In Stodden V, Leisch F, Peng RD (eds.), Implementing Reproducible Computational Research. Chapman and Hall/CRC. ISBN 978-1466561595, http://www.crcpress.com/product/isbn/9781466561595.

Annexes

Annexe 1. Exemple de syntaxe utilisée par la bibliothèque bertin

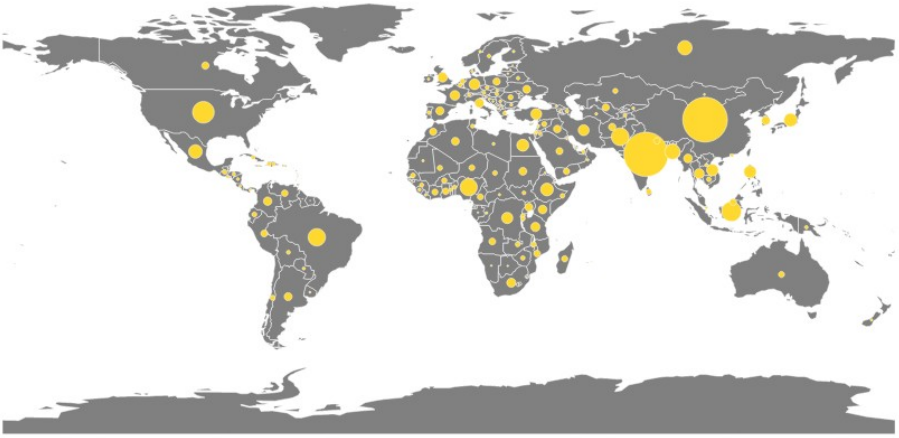
```
bertin.draw({
  params: { width: 500, projection: "Bertin1953" },
  layers: [
    { geojson: world, fill: "#ffc94d" },
    { type: "graticule" },
    { type: "outline" },
    { type: "header", text: "Carte du Monde" }
  ]
})
```

Annexe 2. concision du code grâce aux paramètres par défaut

Annexe 3. Exemple de syntaxe dans R

Example

```
library(bertin)
library(sf)
world <- st_read(system.file("gpkg/world.gpkg", package = "bertin"),
                 layer = "world", quiet = TRUE)
bt_param(width = 800)|>
  bt_layer(data = world, fill = "#808080") |>
  bt_bubble(data = world, values = "pop", k = 20) |>
  bt_draw() |>
  bt_save("map.svg")
```



Annexe 4. Exemple de syntaxe dans l'environnement Python

In [9]:
```
options = {
    "params": {"projection": "d3-geo-projection.geoEckert3()"},
    "layers": [
        {
            "type": "layer",
            "geojson": f'bertin.merge({world}, "ISO3", {data}, "id")',
            "fill": {
                "type": "choro",
                "values": "gdppc",
                "nbreaks": 7,
                "method": "quantile",
                "colors": "RdYlGn",
                "leg_round": -3,
                "leg_title": "GDP per inh\n(in $)",
                "leg_x": 100,
                "leg_y": 200,
            },
            "tooltip": ["$name", "$gdppc", "(current US$)"],
        },
        {"type": "graticule"},
        {"type": "outline"},
    ]
}

Map(options=options)
```

Out[9]: