Names | A Journal of Onomastics

A Note on the 'toponym' R Package: A Practical Introduction

Lennart Chevallier
Kiel University, GERMANY

Søren Wichmann
Kiel University, GERMANY

ans-names.pitt.edu
ISSN: 0027-7738 (print) 1756-2279 (web) Vol. 72 No. 3, Summer 2024 DOI 10.5195/names.2024.2617

Abstract

In this note, we describe how to install and use the 'toponym' R package, which is designed for mapping and manipulating toponymic data from the GeoNames database. This introduction will allow even unexperienced users of R to efficiently produce maps and perform simple analyses.

Keywords: toponym, hydronym, GeoNames, R, software

Introduction

The 'toponym' package essentially provides an interface to the data of GeoNames (GeoNames 2023), which is hosted at https://www.geonames.org/. The package's core functionality is to enable simple distributional displays of toponyms complying with a certain selection criterion, such as the occurrence of a certain string—defined through a regular expression—in the toponyms of one or more specific countries.

The package is written in and for the R computational environment (R Core Team 2022). Thus, the advantage is that filtered data and other output can be further processed seamlessly by the basic R functions or by the countless packages produced for fields including linguistics, geography, statistics, and so on. The disadvantage is that some prior knowledge of R (e.g., acquired through textbooks such as Cotton 2013) is required to make the most of the package. Still, by using the instructions below, even the neophyte will be able to generate useful output. For convenience all code in this note is also provided in a separate online document.[1]

Using the Package

Installation

As a first step, the 'toponym' package itself will have to be installed. Since it is currently available only on GitHub, the package will have to be installed from there, which requires prior installation of the 'devtools' package. The following commands serve to achieve this. Note that the 'greater-than' sign is not part of the R code, but simply precedes each line of code in the console. Text following the hashtags are comments for clarification, and not part of the code that needs to be supplied.

```
> install.packages("devtools")  # installs 'devtools'
> library("devtools")  # loads 'devtools'
> install_github("Lennart05/toponym")  # installs 'toponym'
> library("toponym")  # loads 'toponym'
```


ISSN: 0027-7738 (print) 1756-2279 (web) Vol. 72 No. 3, Summer 2024 DOI 10.5195/names.2024.2617

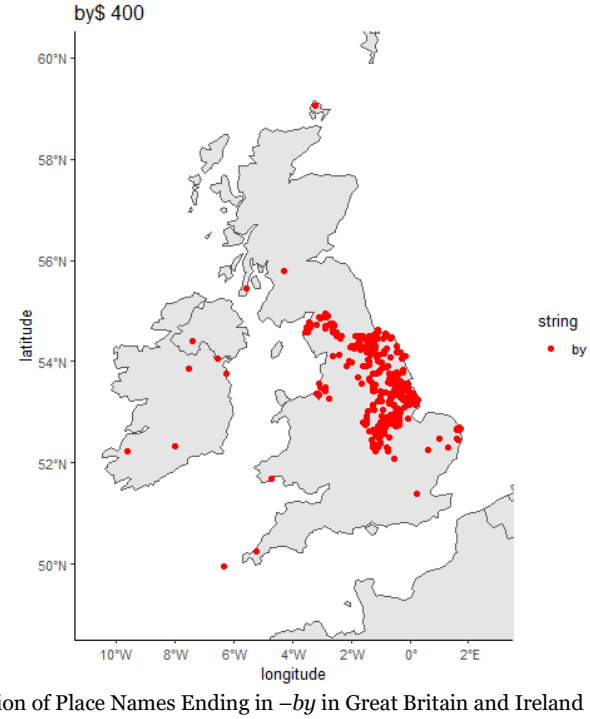

Figure 1: Distribution of Place Names Ending in *–by* in Great Britain and Ireland

Creating a Simple Map

Once the package is installed and loaded, a map such as the one in figure 1 can be created. It requires the function top() (which hints at the word 'toponym'), supplied with one or more strings to search for in GeoNames and one or more countries, in that order. A country can be indicated using either two- or three-letter ISO-3166 codes or the standard English name of the country. There is a function country(), which takes any of these three types of designation as input (e.g., country("DE")), outputting all three of them if the input is, in fact, a correct designation of some country. Another use of the function is to display a full table of all countries and designations writing country("country table"). Thus, when in doubt about how to encode country names, the user can get some orientation through the country() function. Assuming that the user knows, perhaps after a quick check, that the designations "United Kingdom" (the full country name preferred by GeoNames) and "Ireland" can also be abbreviated as "GB" and "IE", then the shortest way to write code producing figure 1 is as follows:

```
> top("by$", c("GB", "IE"))
```

Here the bit c(...) inside the expression indicates a concatenation of the two country designations. A similar concatenation would be needed to search for more than one string. If the user desires to work with one country only, such as the United Kingdom, concatenation is not necessary and the expression would simplify to top("by$", "GB"). The dollar sign at the end of the search string "by$" is part of the language of regular expressions and here serves to indicate that only toponyms ending in *-by* will be searched for (an introduction to regular expressions in R can be received by writing help(regex) in the console). For instance, a name such as *Dunbyrne* would not be included among the hits while *Baldersby* would. If the string in the command were just "by", without the dollar sign, both *Dunbyrne* and *Baldersby* would be found.

The first time data for a particular country is queried, that data will be downloaded from GeoNames. Thus, on such occasions the user should be online and have a bit of patience. For subsequent queries concerning the same country the data will already have been downloaded to the user's computer; as a consequence, access to the data will be quicker.

DOI 10.5195/names.2024.2617 Vol. 72 No. 3, Summer 2024 ISSN: 0027-7738 (print) 1756-2279 (web)

More on the top() Function

The simplicity of the bit of code just given is deceptive in the sense that there are several other instructions to the function which need not be made explicit because they come as defaults. Moreover, the names of the main, mandatory parameters of the function (such as strings, countries) need not be indicated as long as the instructions come in a specific order. For instance, if the search string(s) do(es) not precede the country designation(s) it is necessary to write out the parameter names, as follows:

```
> top(countries=c("GB", "IE"), strings="by$")
```

The order of the parameters and their default settings can be viewed by writing help(top). Any parameter appearing in the list of "additional parameters" on the help page needs to be explicit—that is, the parameter name must be given. From the information given it can, for example, be gleaned, that a map similar to what was produced earlier but with blue instead of red dots and inclusion of highest-level administrative borders could be produced using the following code, which would also output the filtered rows of GeoNames to a file with comma-separated values in the working directory:

```
> top("by$", c("GB", "IE"), color="blue", regions=1, csv=TRUE)
```

When the search string consists of Latin characters, the column in which to search for matches will by default be the [name] column. But it is also possible to search in the [asciiname] and/or [alternatename] columns by specification of the parameter column. If the search string contains one or more non-Latin characters, however, the default automatically changes to the [alternatename] column, which is the only one potentially containing non-Latin characters. For instance, if users want to search for a Cyrillic string such as ^Влад in the data for Russia (^ indicates that it is a leading string), it is not necessary to indicate that the relevant column is [alternatenames]. The following line of code will suffice:

```
> top(strings="^Влад", countries="RU")
```

If, in contrast, it would be of interest to look up the Latin-character string ^*Vlad* in the [alternatenames] column, then column specification would be needed:

```
> top(strings="^Vlad", countries="RU",
column="alternatenames")
```

Finding Frequent Toponym Strings

For finding the most common leading or trailing strings of a certain length, there is a dedicated function called topFreq(). This might be used as an initial, exploratory step. For instance, in a study of the generic parts of US hydronyms, perhaps along the lines of Zelinski (1955) or Campbell (1991), users might start out using this function, varying the length of the endings in order to eventually capture all generic terms. In the following example, a list of the 24 most frequent US endings (type = "$") of length 5 characters belonging to the class of hydronyms (feat.class = "H") is output. The function is blind to what is inside the ending and includes the space character if one occurs among the last five letters.

```
> topFreq(countries="US", len=5, limit=24, type="$",
feat.class="H")
toponyms
```

Creek$	Lake$	ranch$	Well$	pring$	rvoir$	Pond$	Tank$
140723	67253	38744	37526	30691	21242	19265	15696
Brook$	Ditch$	River$	Fork$	rings$	Cove$	lough$	Canal$
11734	8896	8219	4694	3953	3931	3787	3730
Bayou$	Drain$	Swamp$	Lakes$	Falls$	Wash$	Bend$	s Run$
3438	3414	3055	2702	2385	2093	1895	1884

Instead of specifying one or more countries, one or more countries *and* a polygon of interest within the country (or countries) may be specified. The polygon may represent any geographical region on Earth defined through some spanning coordinates. Queries will then be restricted to this polygon. For presentational purposes, we have predefined a polygon for the Flanders region of Belgium. This is constructed as a data frame called flanders_polygon with the columns lats and lons containing numbers representing the coordinates. Users may

ISSN: 0027-7738 (print) 1756-2279 (web) Vol. 72 No. 3, Summer 2024 DOI 10.5195/names.2024.2617

create a polygon themselves, for instance by combining vectors called lats and lons into a data frame, giving that as input to the function. An example of the use of topFreq() with the predefined polygon follows. The output consists of the eight most frequent leading strings (type="^") of cities and villages (the default toponym class) of length 5 in Flanders.

```
> topFreq(countries="BE", len=5, type="^", limit=8,
polygon=flanders_polygon)
toponyms

^Sint-    ^Molen    ^Klein    ^Kruis    ^Nieuw    ^Dries     ^Steen    ^Broek
   117        78        76        45        44        43         42        39
```

Defining a Polygon

Besides the Flanders polygon, the only other predefined polygon is for the historical Danelaw area of England (danelaw_polygon). In order to help users define their custom polygons, we provide a function, createPolygon(), which allows users to specify the spanning coordinates defining a polygon through mouse clicks. Besides invoking the function, it is also necessary to save the output in an (arbitrarily named) object. If, for instance, users are particularly interested in Sumba Island in Indonesia, then users might type the following command.

```
> p <- createPolygon("ID")
```

This will trigger a map of all of Indonesia. Left-clicking with the mouse or touchpad, users can then produce some dots spanning the island. When the last dot (which should not repeat the initial one) has been produced, there are two different ways to end the procedure, depending on the R environment. In the standard RGui that comes with the basic installation of R, users should right click on the mouse or touchpad and then select "stop" in the little menu that emerges. In RStudio, users should instead press escape in order to finish creating the polygon. In the example just given, the selected coordinates will be saved in the object called p. Subsequently that data frame may be fed to functions that accept coordinates, namely top(), topFreq(), topComp(), and topZtest().

If the area of interest corresponds to a high-level administrative unit there is a good chance that it can be retrieved using createPolygon() without any need for mouse-clicks, because the function accepts a parameter region_name, specifying an official administrative unit. An overview of administrative units that can be specified is provided by the function country(), which returns accepted region designations if the additional parameter regions is set to 1 (e.g., country("MX", regions = 1)). More information is provided at https://gadm.org/. In the following example, the element Chi will come to hold coordinates defining the contours of the Mexican state of Chihuahua:

```
> Chi <- createPolygon(countries="MX",
region_name="Chihuahua", retrieve=TRUE)
```

For verification of the contents of Chi, users can simply use R's native plotting function, doing plot(Chi). The contours of Chihuahua will then appear.

Strings Specific to a Region

For determining whether a certain area exhibits toponyms containing strings that are characteristic of that area, the function topComp() offers a means of initial exploration. It is designed to compare the number of occurrences of a certain leading or trailing string (the latter being default) within an area with the number of occurrences in a given country as a whole. It will output strings ordered by their proportional frequency, such that more characteristic strings appear on the top of the list. The maximal number of different strings to output is controlled by a parameter called limit and a cut-off ratio is controlled by a parameter called rat. For instance, a ratio cut-off of 0.7 means that only those cases will be shown where at least 70% of all occurrences of the string in the country appear in toponyms inside the region. The following example applies this cut-off to a search for two-letter endings that are the most distinctive with the historical Danelaw area of Great Britain.

DOI 10.5195/names.2024.2617 Vol. 72 No. 3, Summer 2024 ISSN: 0027-7738 (print) 1756-2279 (web)

```
> topComp(countries="GB", len=2, rat=.7, limit=100,
  polygon=danelaw_polygon)
```

Dataframe data_top_100 saved in global environment.

```
1     by$
2     pe$
```

	toponym	ratio_perc	frequency
1	by$	82.03	324/395
2	pe$	78.77	167/212

Although a display of as many as 100 cases was allowed for, just two strings passed the criterion. The most distinctive is *-by*, for which more than 82% of the occurrences are within Danelaw. This is in fact an old Scandinavian (North Germanic) lexical root meaning 'farmstead, village, settlement' (Mills 2003: 802). Its distribution was displayed in figure 1. More exploration would reveal that most occurrences of *-pe* belong to *thorpe*, which is another Scandinavian lexical root, the same as *thorp*, meaning 'secondary settlement, dependent outlying farmstead or hamlet' (Mills 2003: 814). One way of carrying out this further exploration is to use the function topCompOut(), which, when supplied with the same parameters as topComp(), will produce one distributional map and one data frame with all the pertinent GeoNames data per toponym string. The data frame containing the instances of *-pe*, which will be called data_pe, can be accessed through standard R commands. For instance, the length() function applied to the data_pe$name column will reveal that there are 212 occurrences of *-pe* in the data, and when applied to searches for the string *thorpe* it will reveal 166 occurrences, as follows:

```
> length(data_pe$name)
[1] 212
> length(grep("thorpe", data_pe$name))
[1] 166
```

If the user wanted to extract only the *thorpe*-cases from data_pe and produce a map of those, then the mapper() function may be used. (Obviously, another way is to use top(), inputting the string *thorpe*, but here we want to illustrate mapper(), which was made for the purpose of producing a map of the coordinates in an edited [user-defined] data frame.). Using some native R techniques to extract only strings containing *thorpe*, users can edit the data_pe data frame as follows:

```
> thorpe <- data_pe[grep("thorpe", data_pe$name),]
```

The edited version of data_pe, now called thorpe, can be passed to mapper() to produce a map of populated places containing the string *thorpe* (map not shown here), as follows:

```
> mapper(thorpe)
```

This function accepts parameters not available to top() through which plot and legend titles may be controlled, respectively called title and legend_title. We foresee that mapper() could become an open-ended stage for the development of map-creation functionalities, with more parameters added in the future.

The preceding examples of data frame manipulation have given some hints at how general knowledge of R will strengthen the utility of the package presented here. But such base R functions as length() and grep(), which are not specific to our package, exceed what we can cover here.

A last function intended to aid in the exploration of locally distributed place names is topZtest(). This checks whether the number of occurrences of a given string in a given region is significantly greater than in the rest of the country, taking into account the total number of attested place names within and outside the region. For instance, users might run the following line of code, which will output many five-letter endings whose occurrences within Danelaw represent 50% or more occurrences of each of the various strings in the United Kingdom:

ISSN: 0027-7738 (print) 1756-2279 (web) Vol. 72 No. 3, Summer 2024 DOI 10.5195/names.2024.2617

```
> topComp(countries="GB", limit=100, len=5, rat=.5,
polygon=danelaw_polygon)
```

Among the output strings is *stead*, with 46 occurrences inside Danelaw and 75 in all of the United Kingdom. Next, the topZtest() function can be run as follows:

```
> topZtest(strings="stead$", countries="GB",
polygon=danelaw_polygon)
```

Under the hood, the function invokes R's prop.test() function, employing it to test whether the proportion of occurrences of the final element *stead* to occurrences of all place names not containing it inside Danelaw is significantly greater than the corresponding proportion outside Danelaw, using the Pearson's chi-squared test. In this particular case, the test would indicate that the probability of rejecting the null hypothesis of a random distribution is exceedingly small ($p < .0000001$). We can conclude that *stead* is characteristic of Danelaw toponyms. It is perhaps worthwhile digging into more if we are interested in the Viking connection. But this does not mean anything else. It may well be the case, as etymologists seem to agree (e.g., Mills 2003: 814), that *stead* is Old English and not Scandinavian. This statistical test has been implemented as an exploratory tool, and it cannot supplant careful interpretation.

Other Functions

Above, we have dealt with the main functions of the package. To remain succinct, we did not discuss the full range of parameters and their respective options, but everything is documented in the R help pages of the package. Additionally, there are a couple of utility functions that have not been mentioned previously. The function getData() allows users to update the GeoNames data accessible to the package. The main intended use is to control the date of access to GeoNames database, setting it to the current date. Another function, ortho(), lists all symbols and characters found in specific columns in the data for one or more countries. Other functions are only for internal purposes, being used by other functions already described.

Outlook

We do not have concrete plans for extensions and revisions of the package but envisage improvements following feedback from users. We will strive to maintain backward compatibility such that code making use of the package will still produce the same results after updates. Possible updates might include more options for the shapes of underlying maps being used, for instance allowing for the display of features of the landscape.

Notes

[1] https://github.com/Sokiwi/ToponymNote

Funding Details

This work was supported by the Deutsche Forschungsgemeinschaft (German Research Foundation) under Germany's Excellence Strategy, Grant number EXC 2150 390870439.

References

Campbell, Jon C. 1991. "Stream Generic Terms as Indicators of Historical Settlement Patterns". *Names* 39, no. 4: 333–366. https://doi.org/10.1179/nam.1991.39.4.333

Cotton, Richard. 2013. *Learning R*. Sebastopol, CA: O'Reilly.

Mills, A. D. 2003. *A Dictionary of British Place-Names*. Oxford: Oxford University Press.

R Core Team. 2022. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

Zelinski, Wilbur. 1955. "Some Problems in the Distribution of Generic Terms in the Place-Names of the Northeastern United States." *Annals of the Association of American Geographers* 45, no. 4: 319–349.

Notes on Contributors:

Lennart Chevallier holds a BA in Empirical Linguistics & Political Science and is currently pursuing a Master's degree in Language & Variation and Political Science at Kiel University. He is interested in typology, language description, languages of South Asia, and toponomastics.

Søren Wichmann is a Postdoctoral Fellow at Kiel University (since 2021). He has previously held positions at University of Copenhagen, Leiden University, Kazan Federal University, and Max Planck Institute for Evolutionary Anthropology. He specializes in historical linguistics, descriptive linguistics, language typology, quantitative methods, and Mesoamerican languages and writing systems.

Correspondence to: Lennart Chevallier, ISFAS, CAU, Leibnizstraße 10, 24118 Kiel, Germany; Email: mail@lchevallier.de

ISSN: 0027-7738 (print) 1756-2279 (web) Vol. 72 No. 3, Summer 2024 DOI 10.5195/names.2024.2617