Automatically Translating Documents

Using R, Python, Quarto, and machine learning (NLP) to translate documents in bulk

2024-10-23

Frank Aragona
Washington State Department of Health
2024
Data Integration/Quality Assurance

Table des matières

1 Preface

I originally made a blog post about this work and the R Weekly Podcast recognized it in an episode (shameless plug!). This pdf contains the same information in that blog post in addition to a few more details on parameterized reports to serve as a primer on the subject.

2 Introduction

I made a simple workflow for translating the text of a markdown file into a new language. Epidemiologists, biostatisticians, and data scientists typically use Rmarkdown, Jupyter notebooks, or Quarto to make automated and parameterized reports. These tools allow you to write code within your document so that you can control things like how the document is rendered, how many documents will be output, and have a *dynamically* changing document.

For example, say you want to publish an epidemiological report on COVID-19 sequence variants over time. Since variants are updated and added daily, it would be time consuming to re-write *almost* the exact same report with slightly different text and figures every single time you want to publish. So what we do is write code alongside our text in a markdown file. This lets us render the plots created by our code *dynamically*: every time the data changes, our plots will re-render and the report will update to take those changes.

We can also automatically filter (or parameterize) or report based on a condition. Like say we want each LHJ to have their own report, it would be time consuming and error prone for use to run our code and re-write our text for **every single LHJ**. With the tools mentioned above like Quarto, **we can write one single document and set it to have multiple formats (pdf, word, html), and also filter it to divide reports based on LHJ**. Look at the example below. It takes one report and uses a parameter to divide the report by year:

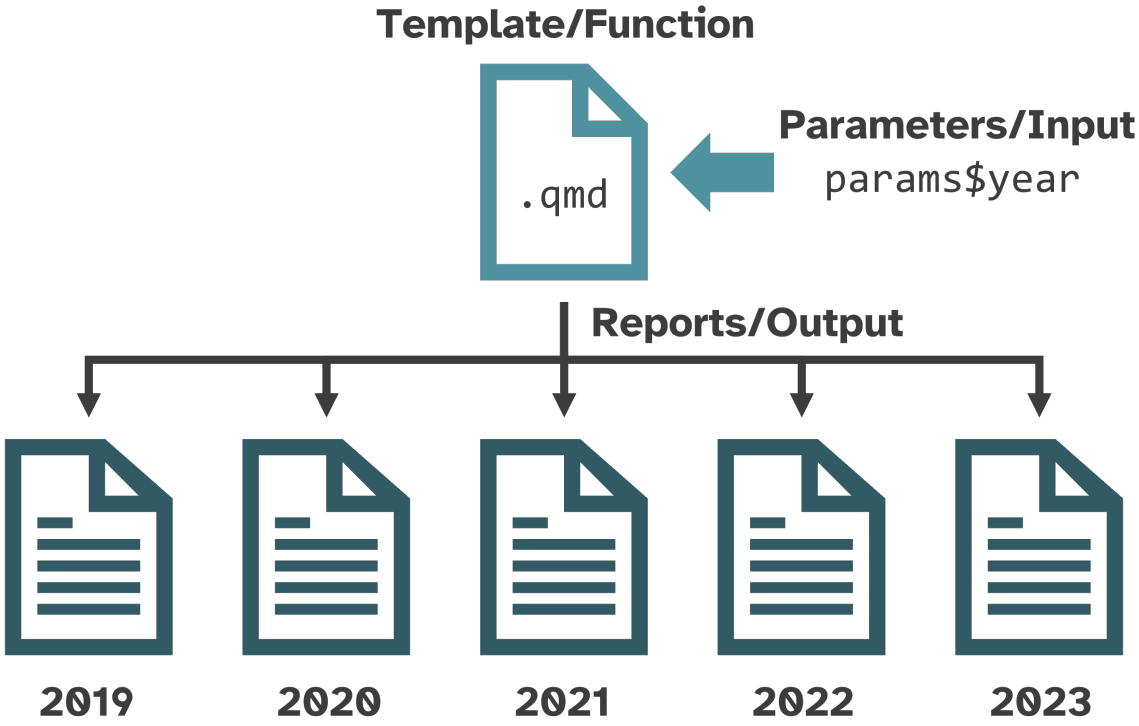

Figure 1 — https://jadeyryan.quarto.pub/cascadia-quarto/2-parameters/2-parameters.html#/like-a-custom-function

3 Translation

I built upon the idea of parameterized and automated reports and developed a workflow to additionally translate the document(s) into whatever language(s) you want.

I'll use English to Spanish as an example for the rest of this paper. Note that I won't be going over how Quarto works - I will just focus on the actual translation aspect.

4 ⊠ Hugging Face Transformers

The first step to translating text is to find a machine learning model that can translate into whatever language we want (as I did not want to build a model from scratch). I looked into a few different APIs (like Google, DeepL), but they all required a credit card on file (even free versions), an API token, and they all have a tier approach where you can only make so many API calls.

I wanted to **simplfy everything** and avoid putting my credit card into any browser. Queue **Hugging Face Transformers** - pre-trained machine learning models that you can easily use in your own projects for free:

> Transformers provides APIs and tools to easily download and train state-of-the-art pretrained models. Using pretrained models can reduce your compute costs, carbon footprint, and save you the time and resources required to train a model from scratch. These models support common tasks in different modalities, such as:
>
> Natural Language Processing: - text classification, - named entity recognition, - question answering, - language modeling, - summarization, - translation, - multiple choice, - and text generation.
>
> Computer Vision: - image classification, - object detection, - and segmentation.
>
> Audio: - automatic speech recognition - and audio classification.
>
> ⊠ Multimodal: - table question answering, - optical character recognition, - information extraction from scanned documents, - video classification, - and visual question answering.

5 Code to use Transformers

There is an R library for hugging face, but I think it requires conda to install some python libraries and I had conda issues, so I just made a more simple package that uses pip to install the python dependencies. And this package is really only for translation and for this demo.

I originally did all of this in Python, but decided to convert everything I could into R because parsing an md file is surprisingly (or unsurprisingly?) way more straightforward in R, and that's where the real magic happens.

5.1 install

- first install reticulate,
- then the package
- and then you need to install the python dependencies

reticulate:

```
# you need reticulate to use the python code
renv::install('reticulate')
```

package:

```
# install my package
remotes::install_github('edenian-prince/translatemd')
```

python dependencies:

This will install a separated virtual environment called `r-transformers` but you can rename it and specify the location of the venv if you want. It will then pip install `torch`, `transformers` and `sentencepiece`

```
translatemd::install_transformers()
```

Getting started, this video helped a ton!

https://www.youtube.com/watch?v=feA-H6blwr4

5.2 select language

Here you need to find a NLP translation model from Hugging Face. I recommend the Helsinki models

When you find a model you want, copy the entire model name and paste it as a string into the function below, like this for the spanish model:

```
translator <- translatemd::select_lang("Helsinki-NLP/opus-mt-en-
es")
```

> **Note**
>
> Note that the models are written like *from xx to yy* so in this case it's *en-es* which is *english to spanish*

5.3 translate

You can input text in the translate function and it will translate english to spanish:

```
translatemd::translate('Hello, my name is Frank')
```

```
[1] "Hola, mi nombre es Frank."
```

6 Translate an Entire Markdown File

- Parse the markdown file
- Apply the translate function to the text
- Re-write the translated markdown into a new document

6.1 md parse

The `lightparser` package is fantastic (and light!). It will take the quarto or rmarkdown file and return a tibble of its elements.

```
(parsed <- lightparser::split_to_tbl('_english.qmd'))
```

```
It seems you are currently knitting a Rmd/Qmd file. The parsing of
the file will be done in a new R session.
```

```
# A tibble: 8 × 8
   type     label              params         text   code   heading
heading_level section
  <chr>   <chr>         <list>      <nam> <lis> <chr>         <dbl>
<chr>
1 yaml    <NA>          <named list> <lgl> <lgl> <NA>            NA
<NA>
2 inline  <NA>          <lgl [1]>    <chr> <lgl> <NA>            NA
<NA>
3 heading <NA>          <lgl [1]>    <chr> <lgl> Quarto           1
Quarto
```

```
4 inline  <NA>          <lgl [1]>   <chr> <lgl> <NA>            NA
Quarto
5 heading <NA>          <lgl [1]>   <chr> <lgl> Automa…          1
Automa…
6 inline  <NA>          <lgl [1]>   <chr> <lgl> <NA>            NA
Automa…
7 block   unnamed-chunk-1 <named list> <lgl> <chr> <NA>          NA
Automa…
8 inline  <NA>          <lgl [1]>   <chr> <lgl> <NA>            NA
Automa…
```

6.2 md translate

unnest the text and apply the translate function

```
parsed_es <- parsed |>
  tidyr::unnest(cols = text) |>
  dplyr::mutate(text_es =
    purrr::map(text,translatemd::translate)
  )
```

let's see what it looks like.

```
parsed_es |>
  tidyr::unnest(cols = text_es) |>
  dplyr::select(type,text) |>
  head()
parsed_es |>
  tidyr::unnest(cols = text_es) |>
  dplyr::select(type,text_es) |>
  head()
```

```
# A tibble: 6 × 2
  type    text
  <chr>   <chr>
1 heading # Quarto
2 inline  Quarto enables you to weave together content and executable code into…
3 inline  To create the release cycle in your repo you may want to use Conventi…
4 inline  Conventional Commits are a way to format and standardize your commit …
5 inline  -  The word `feat:` can trigger a Github Action to add that commit t…
6 inline  -  and it will up-version the minor release version number.
```

```
# A tibble: 6 × 2
  type    text_es
  <chr>   <chr>
1 heading # Quarto
2 inline  Quarto le permite entretejer el contenido y el código ejecutable en u…
3 inline  Para crear el ciclo de lanzamiento en su repo es posible que desee ut…
4 inline  Commits convencionales son una forma de formatear y estandarizar sus …
5 inline  - La palabra `feat:` puede activar una acción de Github para añadir q…
6 inline  - y subirá el número de versión de lanzamiento menor.
```

6.3 write to new markdown file

clean up and write to new

```r
parsed_es_to_qmd <- parsed_es |>
  dplyr::select(-text) |>
  dplyr::rename(text = text_es)

# output to qmd
lightparser::combine_tbl_to_file(
  parsed_es_to_qmd,
  "_spanish.qmd"
)
```

7 Results

And now you have a document in English and another in Spanish

Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

To create the release cycle in your repo you may want to use Conventional Commits.

Conventional Commits are a way to format and standardize your commit messages, which can be used to then automate the repo's release cycle. For example, one conventional naming method is to label any commit associated with a new feature as `feat:` plus a commit message.

- The word `feat:` can trigger a Github Action to add that commit to your changelog under the **Features** header,
- and it will up-version the minor release version number.
- So if you are on release 1.0.0, a new `feat` will up-version the cycle to 1.1.0
- Commit titles that start with the word `fix:` as in a bug fix will up-version the patch number of the, i.e. 1.0.0 to 1.0.1

Automating The Release Cycle

You should consider automating your release cycle so that your project cycle is consistent and predictable. There are many different ways to approach this.

Some repos have semi-automatic cycles where there is some manual component of releasing their software, whereas others are fully automated. Manual releases can work too for some scenarios.

Here's a code chunk

```
mtcars |>
  select(mpg)|>
  slice(1)
```

Quarto

Quarto le permite entretejer el contenido y el código ejecutable en un documento terminado. Para obtener más información sobre Quarto, consulte https://quarto.org.

Para crear el ciclo de lanzamiento en su repo es posible que desee utilizar Commits convencionales.

Commits convencionales son una forma de formatear y estandarizar sus mensajes de commit, que se puede utilizar para automatizar el ciclo de lanzamiento de la repo. Por ejemplo, un método convencional es etiquetar cualquier commit asociado con una nueva característica como `feat:` más un mensaje de commit.

- La palabra `feat:` puede activar una acción de Github para añadir que se compromete a su registro de cambios bajo el encabezado **Características**,
- y subirá el número de versión de lanzamiento menor.
- Así que si usted está en la versión 1.0.0, un nuevo `feat ' subirá la versión del ciclo a 1.1.0
- Commit títulos que comienzan con la palabra `fix:` como en una corrección de errores subirá la versión del número de parche del, es decir, 1.0.0 a 1.0.1

Automatizando el Ciclo de Liberación

Usted debe considerar automatizar su ciclo de lanzamiento para que su ciclo de proyecto sea consistente y predecible.Hay muchas maneras diferentes de abordar esto.

Algunas repos tienen ciclos semiautomáticos donde hay algún componente manual de la liberación de su software, mientras que otras están completamente automatizadas. Las releases manuales también pueden funcionar para algunos escenarios.

Aquí hay un trozo de código.

```
mtcars |>
  select(mpg)|>
  slice(1)
```

8 Discussion

Quarto and Rmarkdown provide an excellent way to blend code and text for publishing automated, reproducible, and parameterized reports. I built on top of these capabilites and added a workflow for translating the language of the documents they output. The workflow uses R, Python, and natural language processing/machine learning to break down the document, run it through a language model, translate, and then convert the document back into a readable form. This workflow could be adapted to translate documents in bulk, that is, when an epidemiologist needs to publish a report in many different formats, parameters, and/or on a regular cadence, they could also translate all the documents they output seamlessly with this code.

9 Bugs

I've caught a few bugs to this approach and you maybe even noticed some!

1. A `#` got removed in the translate - look at the section called `Automating the Release Cycle`. Since the `#` got removed it is no longer a header 😐
2. The `lightparser` package has a reported bug with quarto chunk yaml parameters. Here it converted `#| eval: false` into `#| eval: no`, but we know that the `#| eval: false` should not be treated as text. Hopefully this is fixed

I recommend going through the document and looking for bugs like these! Some manual edits to the translated qmd file may be necessary.

I have a fix for these bugs but have not implemented them at the moment.

10 Full Script Example

```
# install
translatemd::install_transformers()

# select language
translator <- translatemd::select_lang("Helsinki-NLP/opus-mt-en-
es")

# parse your qmd
(parsed <- lightparser::split_to_tbl('english.qmd'))

# translate the qmd
parsed_es <- parsed |>
  tidyr::unnest(cols = text) |>
 dplyr::mutate(text_es = purrr::map(text,translatemd::translate))

# write to a new qmd
parsed_es_to_qmd <- parsed_es |>
  dplyr::select(-text) |>
  dplyr::rename(text = text_es)

# output to qmd
lightparser::combine_tbl_to_file(
  parsed_es_to_qmd,
  "_spanish.qmd"
)
```