ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

SEMESTER PROJECT

Impresso - Metadata mining of large collections of historical newspapers

Author:
Justine WEBER

Supervisors:
Dr. Maud EHRMANN
Dr. Matteo ROMANELLO

Professor:
Prof. Frédéric KAPLAN

Digital Humanities Laboratory
EPFL

January 9, 2020

Abstract

Extracting relevant information from a large dataset and communicating it well, is the main challenge of a data scientist. Having a huge amount of data it great, but to get something out of it, one needs to compute statistics and display results in a forthcoming way.

Impresso: Media Monitoring of the Past is a driving project, aiming at making accessible a large corpus of 200 years historical newspapers. This large scale project is the perfect example of need for extracting and visualizing descriptive statistics on a large dataset. This is the purpose of this project.

Contents

Abstract **iii**

1 Introduction **1**

2 Dataset **3**
- 2.1 Dataset formats . . . 3
- 2.2 Issues table . . . 3
  - 2.2.1 Content . . . 4
  - 2.2.2 Statistics . . . 4
- 2.3 Content items table . . . 4
  - 2.3.1 Content . . . 4
  - 2.3.2 Statistics . . . 5
- 2.4 Targeted aspects . . . 5

3 The impresso_stats package **7**
- 3.1 Dependencies . . . 7
- 3.2 Code structure . . . 7
- 3.3 Dataset expansion . . . 8
- 3.4 Main functions . . . 9
  - 3.4.1 Plot issues . . . 9
  - 3.4.2 Plot content items . . . 11

4 Project retrospective **15**
- 4.1 Chronology . . . 15
  - 4.1.1 Issues - time analysis . . . 15
  - 4.1.2 Licences - adding another *facet* . . . 16
  - 4.1.3 Content items - using a new platform . . . 16
  - 4.1.4 Content items - function design . . . 16
  - 4.1.5 Finalizing . . . 16
- 4.2 Challenges . . . 17
  - 4.2.1 User interface . . . 17
  - 4.2.2 Maintainable code . . . 17
  - 4.2.3 Diverse . . . 18
- 4.3 Limits . . . 18
- 4.4 Extensions . . . 18

5 Conclusion **19**

A Introductory tutorial notebook **21**

B Issues tutorial notebook **41**

C Content items tutorial notebook **63**

1 Introduction

This project is part of project Impresso. Impresso - *"Media monitoring of the past. Mining 200 years of historical newspapers"* - was undertaken in 2017 and aims to enable critical text mining of newspaper archives. Laboratory DHLab of EPFL, supervised by Prof. Kaplan, is one main actor of the project, together with other entities such as The Luxembourg Centre for Contemporary and Digital History (C^2DH) of the University of Luxembourg's, and the Institute of Computational Linguistics of the University of Zürich. One of the goal of this project is to enable historians to access this large newspaper corpus, which has been created from digitizing a huge collection of historical newspapers (mainly Swiss and Luxembourgian for the moment), and to allow them to get information from it. An interface [1] is also in development, to enable wider access to the collection and interesting visualization.

The project presented in this report contributes to these objectives, by providing a python library to produce and visualize descriptive statistics about this newspaper corpus. This library is intended to be used by historians who know the basis of code, and the team working on Impresso, in order to get information on the dataset in an intuitive, fast and universal way. Statistics have been done a two levels : issues and content items, and mainly concern their frequency.

The following chapter (2) gives insights on the dataset, chapter 3 is more technical, presenting the library and introducing how it works, and chapter 4 summarizes the project in its context (chronology, challenges, limits).

As appendices to the report, one can find three jupyter notebooks, constituting tutorials for a future user of the library. It is highly encouraged to have a look at them, for those who intend to use it.

All material is available online [2]. This is especially where you can download the package and the tutorial jupyter notebooks.

[1] Link to the Impresso interface : https://impresso-project.ch/app/

[2] Link to the github repository : https://github.com/dhlab-epfl-students/impresso-metadata-explorer/

2 Dataset

The `impresso_stats` library has been created based on analysis of the Impresso corpus.

The dataset of the newspapers corpus is accessible through several platforms, in different formats. Part of it is available on SQL, part of it is stocked on S3. S3 is a storage service offered by Amazon. It stands for simple storage service and provides cloud storage [1]. Analysis at the issues level has been done based on the SQL dataset (version available in January 2020), and analysis at the content items level has been done based on a pre-exported S3 dataset.

Results and statistics presented in this report are based on the work done in the context of this project, on the impresso dataset. Users of the library can actually load their own dataset and use the functions from impresso_stats, as long as their dataset is "impresso-like", meaning that it has the same characteristics as presented in this chapter.

This chapter aims to give the reader a precise idea on what the dataset looks like, what is its shape and its size, in order to better understand what the library is useful for, and also how to use it on their own dataset.

2.1 Dataset formats

First of all, let's give a brief introduction on the format of the dataset, expected as input to the library's functions. Just as the dataset is available in two different formats, we load it in two different dataframe formats.

The libraries used are pandas and dask. Pandas is a very common library for loading dataframes in python. Dask can be described in a simplistic way, as a distributed version of pandas. It implements a kind of lazy evaluation, thus enabling powerful queries on large datasets.

The table of content items being much larger than the issues one, it cannot be loaded as a pandas dataframe on an ordinary computer. This is the reason why we use the dask library.

To summarize, the issues table (and other light tables that may be used, such as the newspapers table or tables on properties) is loaded as a **pandas dataframe**, and the content items tables is loaded as a **dask dataframe**.

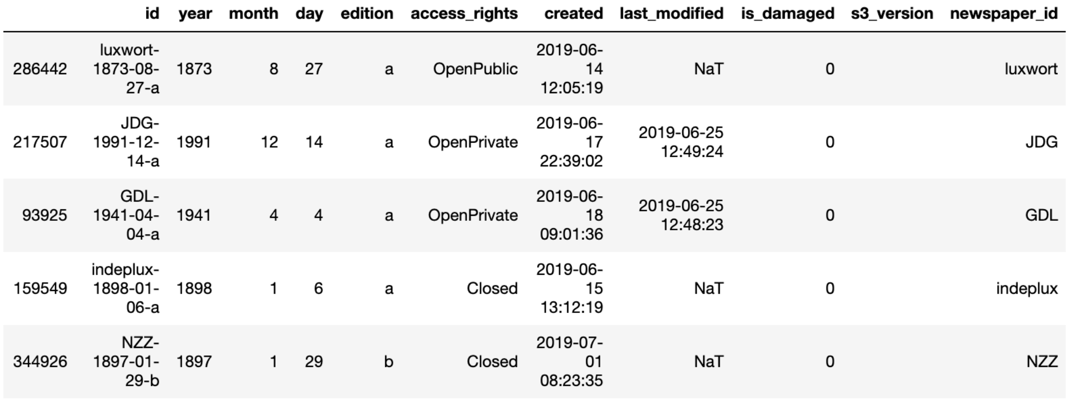
2.2 Issues table

An issue is a publishing of a newspaper at a certain date. Each newspaper has several issues and each issue refers to a newspaper (by its ID).

[1] `https://aws.amazon.com/fr/s3/`

2.2.1 Content

As said in the introduction, the issues table is loaded from SQL, as a pandas dataframe. Figure 2.1 gives an insight on what it looks like.



	id	year	month	day	edition	access_rights	created	last_modified	is_damaged	s3_version	newspaper_id
286442	luxwort-1873-08-27-a	1873	8	27	a	OpenPublic	2019-06-14 12:05:19	NaT	0		luxwort
217507	JDG-1991-12-14-a	1991	12	14	a	OpenPrivate	2019-06-17 22:39:02	2019-06-25 12:49:24	0		JDG
93925	GDL-1941-04-04-a	1941	4	4	a	OpenPrivate	2019-06-18 09:01:36	2019-06-25 12:48:23	0		GDL
159549	indeplux-1898-01-06-a	1898	1	6	a	Closed	2019-06-15 13:12:19	NaT	0		indeplux
344926	NZZ-1897-01-29-b	1897	1	29	b	Closed	2019-07-01 08:23:35	NaT	0		NZZ

FIGURE 2.1: Issues table snapshot.

Some characteristics are particularly important for the library to work properly : the columns' names and the format of their values (typically `string` or `integer` for numbers).

Only some of the columns are needed : `id`, `years`, `access_rights`, `newspaper_id`. The rest is not useful for the library yet. The `year` column contains `integers`. The rest (for the columns we focus on) are `strings`.

2.2.2 Statistics

- Number of rows (i.e. issues) : 441'861.
- Dates : ranging from 1740 to 2018.
- Newspapers : 47 different IDs.
- Access rights : 3 different types -'closed', 'openpublic' and 'openprivate'.

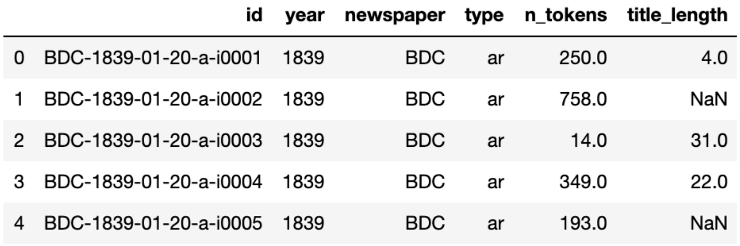
2.3 Content items table

A content item refers to a unit contained in a newspaper page. Content items are therefore at a "lower level" than issues, and even than pages. It corresponds to an element, extracted from the scanned image of a newspaper page, constituting an item on its own. It can be for example an advertisement, an article, or a weather forecast.

Each content item has a reference to a certain issue. The ID of the issue can be extracted from the ID of the content item.

2.3.1 Content

The content item table is loaded from a pre-exported S3 dataset, as a dask dataframe. The following figure (2.2) gives an insight on what it looks like.



	id	year	newspaper	type	n_tokens	title_length
0	BDC-1839-01-20-a-i0001	1839	BDC	ar	250.0	4.0
1	BDC-1839-01-20-a-i0002	1839	BDC	ar	758.0	NaN
2	BDC-1839-01-20-a-i0003	1839	BDC	ar	14.0	31.0
3	BDC-1839-01-20-a-i0004	1839	BDC	ar	349.0	22.0
4	BDC-1839-01-20-a-i0005	1839	BDC	ar	193.0	NaN

FIGURE 2.2: Content items table snapshot.

As for the issues table, the columns' names and the format of their values are particularly important for the library to be workable. Column `n_tokens` is not needed. The values of the column `year` are `integers`, values of columns `title_length` are `floats`, and values of other columns are `strings`.

2.3.2 Statistics

- Number of rows (i.e. content items) : 47'876'994 (10 times more than the issues table).
- Dates : ranging from 1738 to 2018.
- Newspapers : 76 different IDs.
- Type : 9 different values - ar, img, ad, section, picture, page, tb, ob, w.
- Title length : ranging from 1 to 19'306 characters (can also have value `NaN` when not applicable, i.e. when no title).

2.4 Targeted aspects

The impresso_stats package focuses on some specific aspects on the newspapers which are the following:

- issue frequency through time
- licences
- content items frequency
- title length

This list has been established incrementally throughout the project These statistics are basic, but essential to understand the corpus.

In particular, they can be useful for finding anomalies in the dataset, such as missing data or labelling errors.
Another great utility of having descriptive statistics on these characteristics, is to optically analyze the evolution of publications through time, compare newspapers or detect singularities for example.

The rest of the work, namely finding the reasons behind these numbers and the possible anomalies, is up to historians. Based on the provided statistics, they will be able to get insights on what is interesting to study in depth, and play with the library's functions in order to get more and more precise results.

The original list can be extended to other aspects of the dataset, including statistics on the number of pages (per issues), the number of words or characters (per page), the OCR quality, each analysed through time, and per newspaper.

These kind of statistics have in fact been established drawing on the work from BnF (*Bibliothèque nationale de France*). They have managed a similar project on their own dataset, that is, six national and regional French newspapers (1814-1945, 880,000 pages, 150,000 issues) from BnF press collections. Articles on this project, with plots on the statistics they have provided are available online [2].

[2] *Data Mining Historical Newspaper Metadata, Old news teaches history*, Jean-Philippe Moreux (@altomator), BNF : http://altomator.github.io/EN-data_mining/

3 The impresso_stats package

The impresso_stats package is intended to be used by historians as well as computer scientists, for diverse purpose : detecting missing data, spotting differences among newspapers, visualizing the evolution of publications over time, ... The possibilities are many.

The package gathers a set of functions, made for providing statistics on the dataset and visualize them. Most functions which are intended to be used perform a `group-by` and `aggregrate` operation (typically `count` or `mean`), return the aggregated dataframe, and display a bar plot of the result.

The goal of these functions globally is to access information on the dataset in an intuitive, fast and universal way.

This chapter contains indications on the way one can use the functions to get interesting statistics, and snapshots of the results that can be obtained. The purpose of this chapter is not to provide a full tutorial of the package. For that purpose three *tutorial notebooks* are available online, and put as appendix to the report : *introductory tutorial (1)* - Appendix A, *issues tutorial (2)* - Appendix B, *content items tutorial (3)* - Appendix C.

3.1 Dependencies

Before diving into the package's functions, it is worth introducing the main python libraries used in the package.

- Pandas and dask have already been introduced (cf. 2.1) and are the two main python libraries used for loading, querying and handling the dataset.
- Matplotlib and seaborn are the two libraries used for plotting. The latter is based on the former and is more adapted for handling pandas dataframes.

  As shown in the tutorials, it is recommended to import the seaborn library and set the style of it, for better rendering.
- Other libraries such as SQLAlchemy, or NumPy are also useful.

3.2 Code structure

Code is structured in a way which is adapted to packages. There are three python files in the package, containing functions depending on their use.

- **`sql.py`**: this file contains all functions related to loading data from SQL. Two functions are worth knowing :
  - `db_engine()` : the function creating the SQL engine from environment variables (i.e. connecting to the database and interpreting the query).

  - `read_table(table_name, engine)` : the function loading a certain SQL table passed as parameter, through the engine passed as parameter.

- **`helpers.py`**: this file contains all functions handling dataframes (dask or pandas). It includes functions for expanding dataframes (cf. 3.3), querying them (e.g. getting the newspapers IDs which have a certain property), filtering them, and others. Since these functions are not the ones intended to be used the most by users of the package, we will not go through them in this report. The function that is mostly worth looking at, is `filter_df`. It takes a dataframe as parameter (dask or pandas) and some filtering parameters, and returns a filtered dataframe according to those. It is used as helper function to the visualization functions.

- **`visualization.py`**: this file contains all functions for visualization, i.e. functions intended to be called by users of the package, which aggregate the dataframe, return it and display a barplot.

  It is made of three sections : the first one for functions on issues, the second one about licences, the last one for content items. The main functions of this file are presented in the following section (cf. 3.4).

  For the sake of clarity, the file also contains many helper functions, which are called one by another to allow greater modularity of the code.

The details regarding how functions work and which other function they call are important to understand, for one who would maintain the package, or wishes to adapt it to some particularities of their own dataset. It is not needed for users who simply wish to get statistics on their dataset, such as historians.

3.3 Dataset expansion

Before describing the key functions of the library, let's briefly introduce essential helper functions, which expand the basic dataframes (i.e. add new columns) useful for our analysis:

- <u>Decades</u>

  A function of the library enables adding a column 'decade' to both dataframes: `decade_from_year_df()`. Documentation is given directly in the code. Succinctly, it takes a dask or pandas dataframe as input and returns the same dataframe, with a new column with the name 'decade'. The values inside the column are `integers`. Having the possibility to analyse and plot by decade is very useful, especially when analysing over a large period of time. It is recommended to add this column to the dataframe.

- <u>Access rights</u>

  As presented on the tables snapshot (figures 2.1 and 2.2 in chapter 2), there is no access right information at the level of content items. Yet it may be interesting to analyse the access rights for content items. A function of the library enables adding a column 'access_rights' to the content items dataframe: `licenses_ci_df()`. It takes as input a dask dataframe having the exact same format as presented in figure 2.2, with an extra decade column, and returns the same dataframe with a new 'access_rights' column.

3.4 Main functions

3.4.1 Plot issues

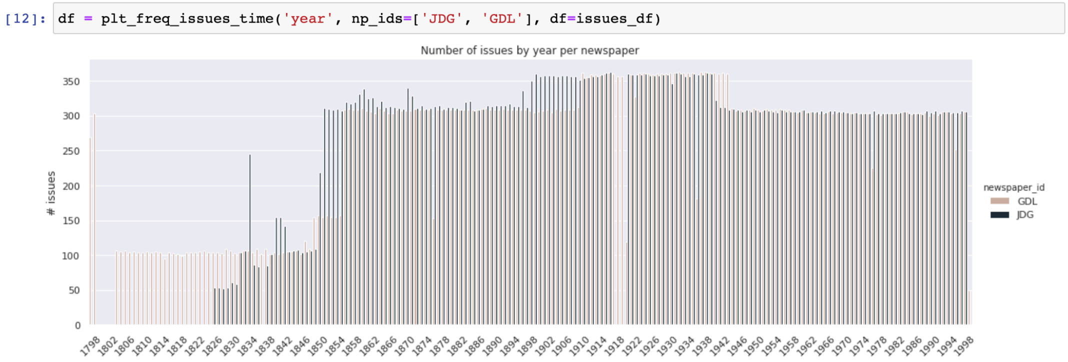
- Issues frequency through time

  The function for issues frequency returns an aggregated dataframe with three columns : a time column, a newspaper column and a count column, and displays a histogram. Let's see what this function looks like and what it takes as parameters. An example of output is shown on figure 3.1.

```
plt_freq_issues_time(time_gran,
                     start_date,
                     end_date,
                     np_ids,
                     country,
                     df,
                     ppty,
                     ppty_value,
                     batch_size)
```

  We won't go through all the parameters as it will be tedious and is already explained in detail in appendix B. Let's describe the most important ones briefly:

  - time_gran can take string values 'year' or 'decade' depending on the time granularity the user wants.
  - df is the dataframe (it must be a pandas dataframe in this case) of issues, on which to perform aggregation and to plot. It must have the same format as described in 2.2. It is an optional parameter. When not specified, calling the function also loads the dataset.
  - batch_size (optional) takes an integer value for plotting by batch when a lot of data needs to be displayed (see appendix B for details about group plotting).
  - all other parameters are optional and are filtering parameters which can be applied to the dataframe (see appendix A for details about filtering).



```
[12]: df = plt_freq_issues_time('year', np_ids=['JDG', 'GDL'], df=issues_df)
```

FIGURE 3.1: Number of issues through time for newspapers *Gazette de Lausanne* and *Journal de Genève*, by calling function plt_freq_issues_time

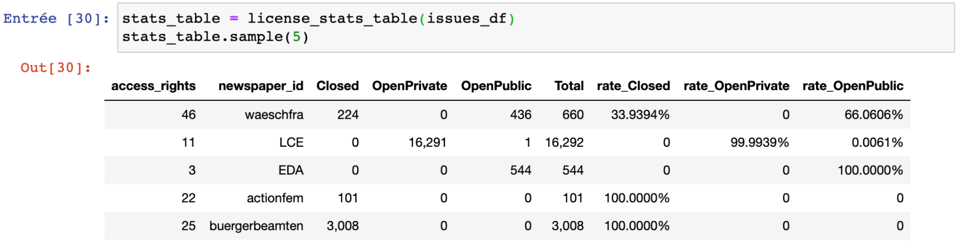
- Access rights frequency

  The access right value (also called licence), specifies the conditions for having access to a document (here an issue). The access right is defined at the level of issues. Typically an issue can by accessible to anyone, or only to people having signed a NDA.

  The access rights of the issues in our database can have three values : 'open public', 'open private', 'closed'. There are three main functions for access rights statistics:

  1. `license_stats_table`
  2. `multiple_ar_np`
  3. `plt_licences`

  The first function (`license_stats_table`) takes as parameter a dataframe a returns a table containing statistics on the different access right types of issues for each newspaper. Figure 3.2 gives an example of the statistics table one can get.



Entrée [30]:

```
stats_table = license_stats_table(issues_df)
stats_table.sample(5)
```

Out[30]:

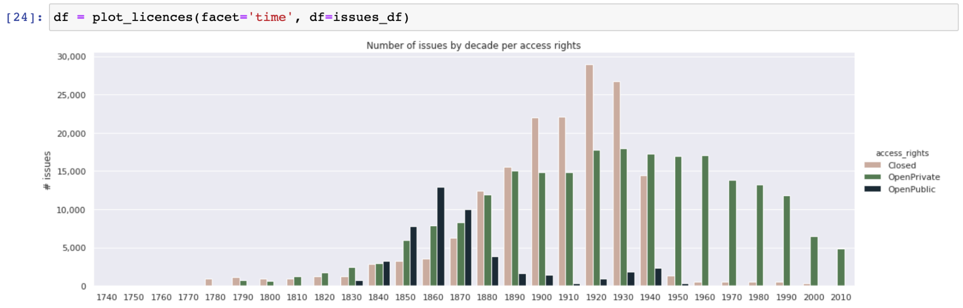
access_rights	newspaper_id	Closed	OpenPrivate	OpenPublic	Total	rate_Closed	rate_OpenPrivate	rate_OpenPublic
46	waeschfra	224	0	436	660	33.9394%	0	66.0606%
11	LCE	0	16,291	1	16,292	0	99.9939%	0.0061%
3	EDA	0	0	544	544	0	0	100.0000%
22	actionfem	101	0	0	101	100.0000%	0	0
25	buergerbeamten	3,008	0	0	3,008	100.0000%	0	0

FIGURE 3.2: Statistics on issues access rights per newspapers, by calling function `license_stats_table`

  Actually in the impresso dataset, most newspapers have only one type of access right : all their issues are either open public, open private, or closed. There are, however, a few exceptions, and function 2 (`multiple_ar_np`) returns the list of newspaper IDs for which their issue have at least two different access right types, from a pandas dataframe having columns 'access_rights' and 'newspaper_id'. It can be interesting to combine this function, filtering and function 1 (`license_stats_table`), to get statistics on the newspapers which have several access right types only (cf. Tutorial 2 as appendix B for example).

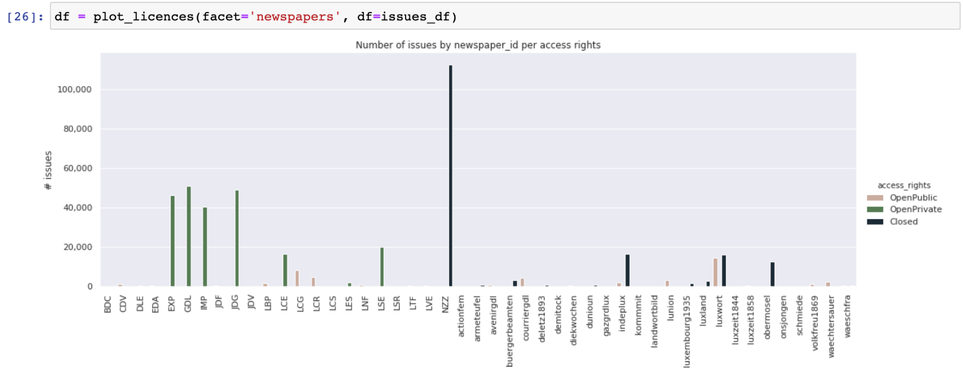
  Finally, function 3 (`plt_licences`) displays a bar plot of the number of issues per access right type, either through time (decade only) or by newspaper. Parameter 'facet' specifies which one to chose (taking values 'time' or 'newspaper'). Figures 3.3 and 3.4 present one example of each.

See tutorial notebook as Appendix B.



```
[24]: df = plot_licences(facet='time', df=issues_df)
```

FIGURE 3.3: Number of issues per access right type through time, by calling function `plt_licences`



```
[26]: df = plot_licences(facet='newspapers', df=issues_df)
```

FIGURE 3.4: Number of issues per access right type per newspaper, by calling function `plt_licences`

3.4.2 Plot content items

The functions made for analysis at the level of content items have been created after those at the level of issues. They enable greater modularity and more plotting options.

Regarding content items, two measures can be analysed : the **content items frequency** and the **average title length of content items**. Both measures have two main functions : `plt_freq_ci` and `plt_freq_ci_filter` for content items frequency, `plt_avg_tl` and `plt_avg_tl_filter` for average title length. Both groups of functions work exactly the same way and only differ in the way they aggregate the dataframe (count operation for frequency versus mean operation for average). As their name suggests, functions having suffix `_filter` take filtering parameters. Apart from that, they work exactly the same as the simple functions. Calling function `plt_freq_ci_filter`, without passing it any filter parameters, comes down to calling `plt_freq_ci` (same for `plt_avg_tl_filter`).

In this section we will present how function `plt_freq_ci_filter` works, and provide examples of what can be obtained using it. Exploration of the details and of other functions is left to users of the package.

Here is what this function looks like and what it takes as parameters:

```
plt_freq_ci_filter(df,
                   grouping_col,
                   asc,
                   hide_xtitle,
                   log_y,
                   types,
                   start_date,
                   end_date,
                   np_ids,
                   country)
```

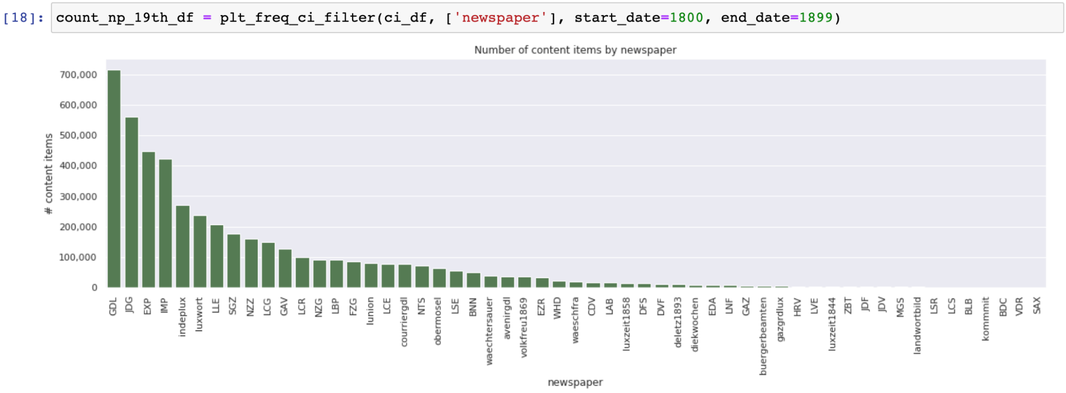
We distinguish two types of plotting for content items : ***one-dimensional* plotting** and ***two-dimensional* plotting**. Both can be obtained by calling the same function, changing one parameter : `grouping_col`. This parameter takes a list containing either one or two parameters, depending on whether one wishes to plot in *one* or *two* dimensions respectively. The values in the list are column names.

One-dimensional analysis stands for analysis aggregated a one level. For example, analysing the content items frequency at *one-dimension*, means computing the content items frequency by year, or by newspapers for example, but one at a time only. Based on that definition, one can easily deduce what *two-dimensional* analysis means: analysis aggregated at two levels. It would be for example computing the number of content items by newspapers, per year (or any other combination of two characteristics).

- One-dimensional analysis

  Based on the dataframes presented in chapter 2, the values which can be used as parameter `grouping_col` are the following: year, decade, newspaper, type, licenses.

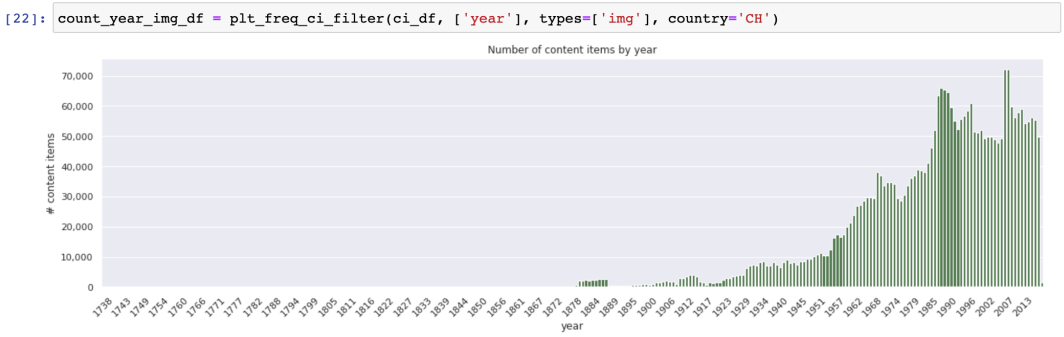
  As already explained earlier, the number of possibilities is huge, as you can apply a lot of filters and combine them. Here are some examples.



```
[18]: count_np_19th_df = plt_freq_ci_filter(ci_df, ['newspaper'], start_date=1800, end_date=1899)
```

FIGURE 3.5: Number of content items per newspapers in the 19th century, by calling function `plt_freq_ci_filter`

Figure 3.5 shows how one would get information on newspapers that have the largest number of content items during the 19th century.



```
[22]: count_year_img_df = plt_freq_ci_filter(ci_df, ['year'], types=['img'], country='CH')
```

FIGURE 3.6: Number of images in Swiss journals through time (year granularity), by calling function `plt_freq_ci_filter`

Figure 3.6 shows how one would get information on the number of images which have been published in Swiss newspapers, per year.

- Two-dimensional analysis

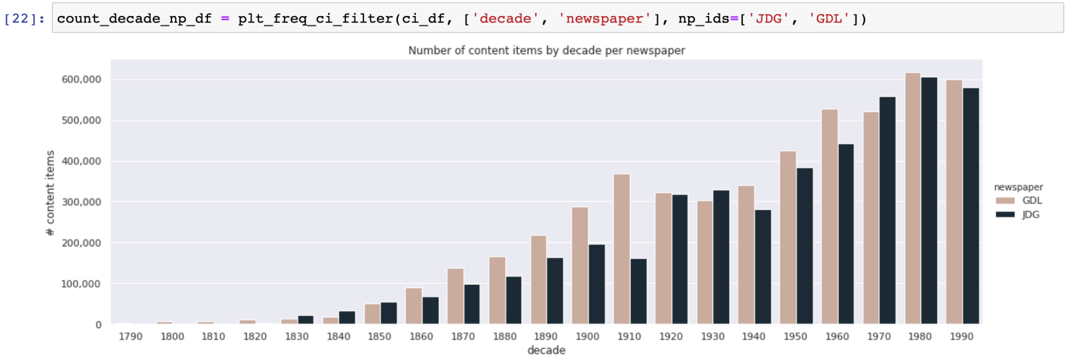
  For two-dimensional analysis, parameter `grouping_col` needs to be a list of length 2. The order in which the values are put in the list matters ! The first one will correspond to the x-axis, and the second one to the category (hue on the plot). Time features (years or decades) shouldn't be used as the second element in the list (as it doesn't make sense to use it as a category).

  Also when plotting in two-dimensions, more data is shown on a single plot. It is then necessary (at least extremely recommended) to use filters in that case, in order to reduce the amount of data to be plotted.

  Figure 3.7 shows how one would compare the number of content items for journals *Gazette de Lausanne* and *Journal de Genève* by decade.

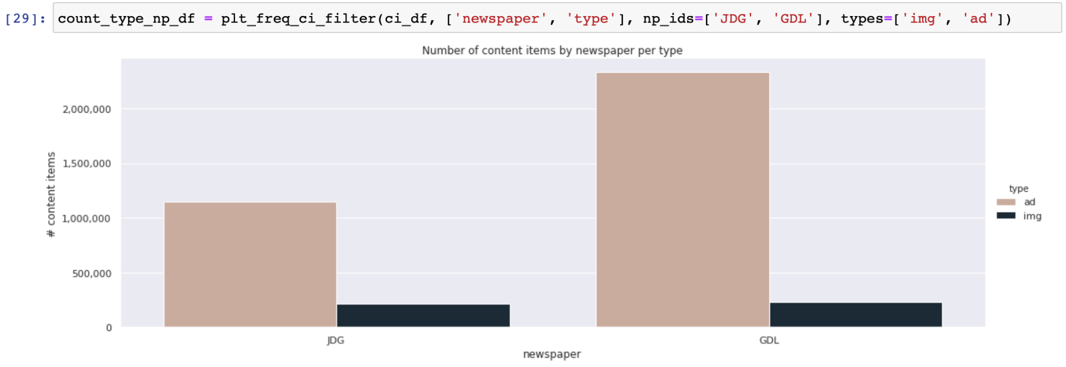
  Figure 3.8 shows how one would compare the number of images to the number of advertisements published in journals *Gazette de Lausanne* and *Journal de Genève* through the whole dataset (all time range).

Not all functionalities have been presented here. For example, you can produce the same plots in logarithmic scale. More details are given in the notebook as Appendix C.



```
[22]: count_decade_np_df = plt_freq_ci_filter(ci_df, ['decade', 'newspaper'], np_ids=['JDG', 'GDL'])
```

FIGURE 3.7: Number of content items by decade for journals *Gazette de Lausanne* (light) and *Journal de Genève* (dark), by calling function `plt_freq_ci_filter`



```
[29]: count_type_np_df = plt_freq_ci_filter(ci_df, ['newspaper', 'type'], np_ids=['JDG', 'GDL'], types=['img', 'ad'])
```

FIGURE 3.8: Number of images (dark) compared to advertisements (light) in journals *Gazette de Lausanne* and *Journal de Genève* (over the whole dataset), by calling function `plt_freq_ci_filter`

4 Project retrospective

This chapter aims to describe the external context of the project : the chronology, the faced challenges, the limits of the library and finally what could be done in the future.

From a more personal point of view, this chapter reflects upon what I thought about when creating this library, how I think people may use it, and what I think could have been done better and deserves some improvements.

4.1 Chronology

This project has been started in September 2019 and ended in January 2020. The original goal was to provide statistics on newspapers metadata, and make the results available to those who would use it (historians or computer scientists), in some way. One idea was to integrate the functionalities to the Impresso interface later on.

We have decided first to create generic functions which would enable visualization in a jupyter notebook.

Understanding the chronology of the project can help understand why functions have been built in this way, and why they may be built in different ways.

The indications given on the time period that each part lasted are approximated. They include some time that may have been "lost" because of setting up issues. It is intended to give the reader a better idea of the proportion of time that was spent on each part and of the pace at which the project has progressed.

4.1.1 Issues - time analysis

$\approx$ 4 weeks (17 sept. 2019 - 15 oct. 2019)

The first step has been to explore the dataframe, creating statistics and plotting into a **jupyter notebook**, using the pandas and seaborn libraries, and loading data from SQL. The focus was on issues throughout the first half of the project.

Once interesting work on issues was well underway, i.e. once I had obtained interesting plots and created the functionality to plot by batches (cf. Appendix B), work on **code organizing** was undertaken: first by creating modular functions, second by creating python files gathering those functions. At this point I also started **documenting** the code, through docstring and type hints.

The global development process for most functions, from this moment on, was the following : writing code in a jupyter notebook and doing some tests, once it worked correctly write it as a function which takes parameters and test it, finally transfer the function to the corresponding python file and import it in the jupyter notebook.

If necessary the functions would be modified later on, directly in the python file to enable more functionalities or change some parameters.

4.1.2 Licences - adding another *facet*

≈ 3 weeks (15 oct. 2019 - 05 nov. 2019)

During this first part, the focus was only on statistics through time. Then analysis on licences was started. This introduced a new "dimension" for statistics: not only per year (or decade), but also per newspapers. The way of specifying which of the two *facets* to choose was done through a parameter `facet` taken by the function.

The **filtering function** (cf. Appendix A) was created at this time, as it became more and more useful to have a standard way to specify which newspapers to choose for all functions (in that moment, for licence function and basic issues frequency function).

4.1.3 Content items - using a new platform

≈ 2 weeks (05 nov. 2019 - 19 nov. 2019)

So far, all work was done locally on an ordinary laptop. Then we decided to focus on a lower level : content items. As the dataframe was much larger, from then on computation was done on a computing cluster node. I also started working with S3 and trying to use it myself to export data, but we have decided that focusing on analysis rather than exporting data was more important, and that we wouldn't have time to do both. Therefore, I worked on a pre-exported S3 dataset.

This is also the time where I have started using and learning dask. Surprisingly positively, the majority of the functions that had already been created in the first part, made for pandas dataframes, needed very few adaptations in order to work with dask dataframes (in particular the filtering function).

4.1.4 Content items - function design

≈ 2 weeks (19 nov. 2019 - 3 dec. 2019)

In the second half of the project (i.e. work on content items), more time has been devoted to decide how to design functions in a way that would be both intuitive for a user of the library, and understandable for someone who would maintain it.

The functions made in the second part are therefore more modular : they allow more "facets" than the two basic ones (newspapers and time). The code is also built using more helper functions, in order to avoid repetition and provide good structure.

4.1.5 Finalizing

≈ 3 weeks + extra-time post-semester (from 3 dec. 2019)

The next step has been to package this python module, by creating a `__init__.py` file and enable the possibility to download it, through the `setup.py` file.

Finally, once most functions had been done, they have been enhanced to enable more functionalities, for example plotting in logarithmic scale, specifying plotting settings. Filtering has also been enhanced to adapt to content items and enable filtering by type.

Functions for analysing average title length have been created in the end, and have been built based on the code structure for content items frequency.

4.2 Challenges

The challenges related to this project have been encountered in various areas.

4.2.1 User interface

As the goal was for the functions to be used by people who may not be computer scientists, it was very important to put myself in their shoes and try to determine what was the best way to create an intuitive and easy-to-use library. A big work has been done on function design, to create a good user experience.

This concern has been mainly handled during the second part of the project. A good example is the way the functions for content items have been built. There was a lot of options, for example:

1. Having a unique function which would have had one more parameter, specifying whether to plot the average title length, or the content items frequency.
2. Having one function for plotting average title length and one for plotting content items frequency (the implemented option).
3. Having one function for *one-dimensional* plotting (1D) and one for *two-dimensional* plotting (2D) - both having a 'facet' parameter specifying whether to plot average title length or content items frequency .
4. Having four distinct functions : one for 1D average title length, one for 2D average title length, one for 1D content items frequency, one for 2D content items frequency.
5. Any other option, since the possibilities are infinite.

The question was : what would be more intuitive for the users ? As information on content items frequency and average title length is very different and would probably be needed for different reasons, option 2 has been chosen because it seemed more intuitive for a historian.

4.2.2 Maintainable code

Another aspect was to create a library which is maintainable : not only nice for external users but also for internal users (people modifying the code). For that purpose it was important to focus on code organization.

Following the example above, since the back-end is very similar for both functions (only the operation to aggregate the `group-by` is different - and a few plotting settings such as the title), a helper function called by both functions has been created. It takes a parameter "facet", which can be either "freq" or "avg".

A typical user of the library would not see these details. This is only useful for someone maintaining the library or adapting it for a specific use and has been designed to make sure he would have as little to change as possible.

4.2.3 Diverse

A few other challenges have been faced during this project and are very typical for coding projects.

It concerns for example technical issues for connecting to the database or the cluster and setting up the environment. Thanks to the support of DHLab, these have been easily handled and have not stuck me for too long.

Another kind of difficulty is taking up new libraries such as seaborn, for which it can be hard to find clear documentation. The consequence of this is that one needs to try a lot of different things before making it work and understanding it. The positive side from that kind of situation is that it also teaches how to learn something on our own. Other examples that have been faced through this project are : handling multiindex pandas dataframe, or filling time gaps when plotting frequency by time, in the case of absence of publications for some years / decades.

4.3 Limits

It has already been introduced earlier in this report that things have been learned progressively throughout the project. Implementations done at the end of the project are more mature than those done at the beginning. Some work has been done to homogenize a bit. For example, functions' names have been adapted, and plotting settings have been unified for all functions so that they all call the same helper functions.

However some more work should be done to standardize everything, for example by designing functions on issues similarly to what has been done for content items and thus enable plotting in one or two dimensions, and enabling the possibility to read from the database in all functions or in none of them.

4.4 Extensions

According to the limits that have been described right above, here is what could or should be done to maintain and extend this library (in this priority order according to me).

1. Unify all and in particular design the issues function to be similar to the content items functions.
2. Create tests for helping maintenance.
3. Complete the pipeline for content items, so that the function itself fetches data from S3. That way it wouldn't be needed to have a pre-exported dataframe.
4. Provide functions for "cross-level" analysis, such as average on content items statistics per issues. For example, create a function for getting the average number of images (content-items level) per issue (issues level), per newspaper over time. The function would be modular and enable any similar average.
5. Adapt the functions output to integrate this information to the Impresso interface, and work together with the designers.

5 Conclusion

The outcome of this project is the creation of a python package, providing a way to get and visualize descriptive statistics about the Impresso newspaper corpus.

The impresso_stats package has not had the opportunity to be tested yet, and will probably need some adjustments and improvements. Coming from a computer science background, some functionalities which seem intuitive to me, may not be for future users, evenn though effort have been put toward this.

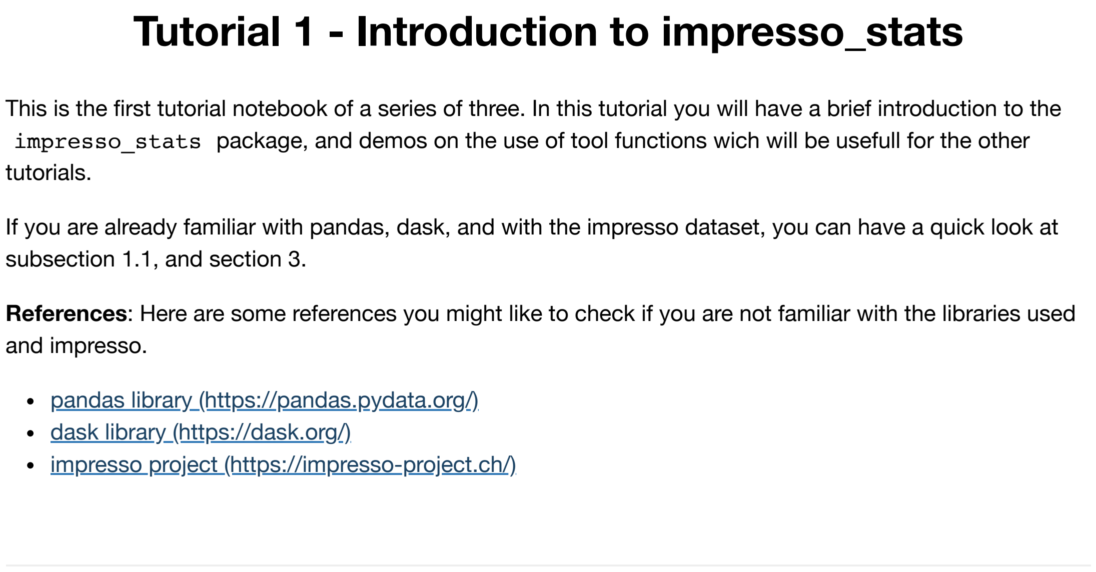
Work done through this project - and especially on content items - constitutes a good basis for enriching the impresso_stats package and provide many more functionalities.

This project has been a great opportunity to participate in the Impresso project, produce something useful and work with a large dataset of concrete data. I hope this work will be useful to the Impresso team at DHLab, and will one day be adapted to the Impresso interface.

To conclude, I would like to thank the whole DHLab for their support throughout the project, and in particular my two supervisors Dr. Maud Ehrmann and D. Matteo Romanello, who defined the lead of the project, and have always been available and gave me valuable advice, whenever it was needed.

A Introductory tutorial notebook

This is the introductory tutorial, presenting the dataset and the main helper functions.



Tutorial 1 - Introduction to impresso_stats

This is the first tutorial notebook of a series of three. In this tutorial you will have a brief introduction to the `impresso_stats` package, and demos on the use of tool functions wich will be usefull for the other tutorials.

If you are already familiar with pandas, dask, and with the impresso dataset, you can have a quick look at subsection 1.1, and section 3.

References: Here are some references you might like to check if you are not familiar with the libraries used and impresso.

- pandas library (https://pandas.pydata.org/)
- dask library (https://dask.org/)
- impresso project (https://impresso-project.ch/)

Table of content

The impresso_stats package

Setup

1 Dataset overview

- 1.1 Issues table
- 1.2 Content item table

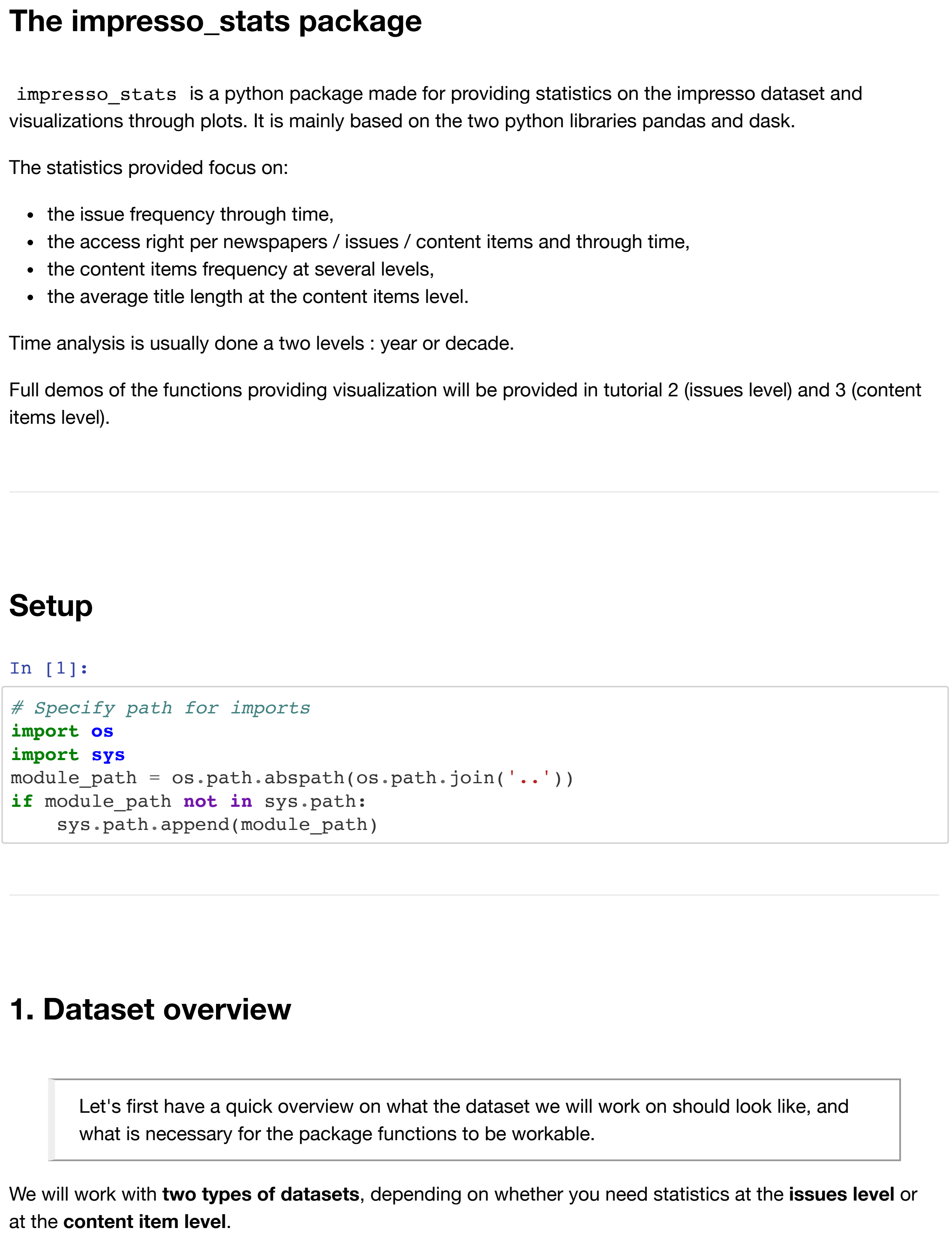
2 Statistics

- 2.1 Issues table
- 2.2 Content item table

3. Tool functions

- 3.1 Expanding
  - 3.1.1 Decades
  - 3.1.2 Access rights at content item level
- 3.2 Filtering
  - 3.2.1 Filtering by date
  - 3.2.2 Filtering by newspaper ID
  - 3.2.3 Filtering by country
  - 3.2.4 Filtering by property

4. impresso_stats



The impresso_stats package

`impresso_stats` is a python package made for providing statistics on the impresso dataset and visualizations through plots. It is mainly based on the two python libraries pandas and dask.

The statistics provided focus on:

- the issue frequency through time,
- the access right per newspapers / issues / content items and through time,
- the content items frequency at several levels,
- the average title length at the content items level.

Time analysis is usually done a two levels : year or decade.

Full demos of the functions providing visualization will be provided in tutorial 2 (issues level) and 3 (content items level).

Setup

In [1]:

```
# Specify path for imports
import os
import sys
module_path = os.path.abspath(os.path.join('..'))
if module_path not in sys.path:
    sys.path.append(module_path)
```

1. Dataset overview

> Let's first have a quick overview on what the dataset we will work on should look like, and what is necessary for the package functions to be workable.

We will work with **two types of datasets**, depending on whether you need statistics at the **issues level** or at the **content item level**.

1.1 Issues table (SQL > pandas)

For the analysis at the issues level, data has been loaded from SQL. Here are some tool functions that can be useful if you also wish to load data from SQL.

Read Table

The `read_table` function is usefull for loading a table from SQL to pandas. You can use it if you have the database in SQL format. Otherwise you should load your database the way you are used to, to have it as a pandas dataframe.

The `db_engine` function is needed for creating the sql engine from you environment variables. Check in the code directly to see how your environment varibles should be named.

In [2]:

```
from impresso_stats.sql import db_engine, read_table
```

In [3]:

```
engine = db_engine()
```

As most function are made to do analysis at the issue level (or content item level but we will see that later), we focus on the `issues` table, but you can use this function to load any other impression SQL table (e.g. `newspapers` table).

In [4]:

```
issues_df = read_table('impresso.issues', engine)
#newspapers_df = read_table('impresso.newspapers', engine)
```

Snapshot

In [5]:

```
issues_df.head()
```

Out[5]:

	id	year	month	day	edition	access_rights	created	last_modified	is_damaged
0	actionfem-1927-10-15-a	1927	10	15	a	Closed	2019-06-15 12:22:38	NaT	0
1	actionfem-1927-11-15-a	1927	11	15	a	Closed	2019-06-15 12:22:38	NaT	0
2	actionfem-1927-12-15-a	1927	12	15	a	Closed	2019-06-15 12:22:38	NaT	0
3	actionfem-1928-01-15-a	1928	1	15	a	Closed	2019-06-15 12:22:41	NaT	0
4	actionfem-1928-02-15-a	1928	2	15	a	Closed	2019-06-15 12:22:41	NaT	0

The issues table we will use must contain at least the 4 following columns :

- `id`
- `year`
- `access_right`
- `newspaper_id`

1.2 Content item table (S3 > dask)

For analysis at the content item level, data has been loaded from a prepared dataset exported from s3. We will show you here how this prepared dataset should look like.

As it is much larger than the issues dataset, here we load data as a **dask dataframe** (and not a pandas dataframe).
See here (https://dask.org/) for full documentation.

We also use tools from the **impresso_commons** package.
Full code and documentations can be found here (https://github.com/impresso/impresso-pycommons).

In [6]:

```
import dask.dataframe as ddf
```

In [7]:

```
# Usefull for reading the dask dataframe in the right format.
from impresso_commons.utils.s3 import IMPRESSO_STORAGEOPT
```

In [8]:

```
# Put the variables corresponding to your own location and file name
PATH = '/scratch/students/justine/'
FILE = 's3-impresso-stats'

ci_df = ddf.read_csv(
    PATH+FILE+"/content-item-stats/*",
    storage_options=IMPRESSO_STORAGEOPT)
```

Snapshot

In [9]:

```
ci_df.head()
```

Out[9]:

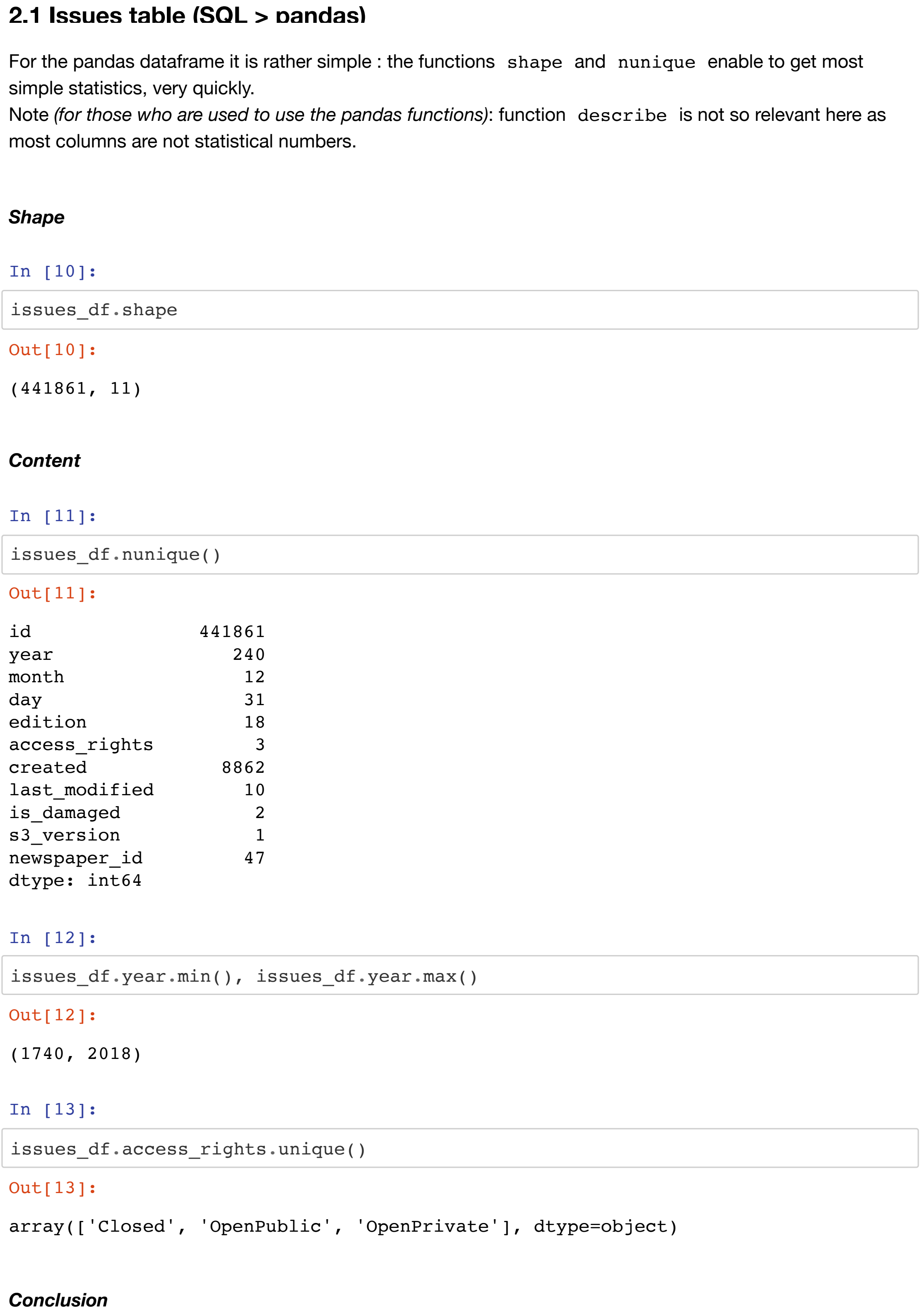
	id	year	newspaper	type	n_tokens	title_length
0	BDC-1839-01-20-a-i0001	1839	BDC	ar	250.0	4.0
1	BDC-1839-01-20-a-i0002	1839	BDC	ar	758.0	NaN
2	BDC-1839-01-20-a-i0003	1839	BDC	ar	14.0	31.0
3	BDC-1839-01-20-a-i0004	1839	BDC	ar	349.0	22.0
4	BDC-1839-01-20-a-i0005	1839	BDC	ar	193.0	NaN

The content item table we will use must contain at least the 5 following columns :

- `id`
- `year`
- `newspaper`
- `type`
- `title_length`

2. Dataset statistics

> **This sections aims to show you how to retrieve simple statistics about the datasets, and give you an idea on our dataset scale.**



2.1 Issues table (SQL > pandas)

For the pandas dataframe it is rather simple : the functions `shape` and `nunique` enable to get most simple statistics, very quickly.
Note *(for those who are used to use the pandas functions)*: function `describe` is not so relevant here as most columns are not statistical numbers.

Shape

In [10]:

```
issues_df.shape
```

Out[10]:

```
(441861, 11)
```

Content

In [11]:

```
issues_df.nunique()
```

Out[11]:

```
id                441861
year                 240
month                 12
day                   31
edition               18
access_rights          3
created             8862
last_modified         10
is_damaged             2
s3_version             1
newspaper_id          47
dtype: int64
```

In [12]:

```
issues_df.year.min(), issues_df.year.max()
```

Out[12]:

```
(1740, 2018)
```

In [13]:

```
issues_df.access_rights.unique()
```

Out[13]:

```
array(['Closed', 'OpenPublic', 'OpenPrivate'], dtype=object)
```

Conclusion

- The issues dataframe contains 441'861 lines (and 11 columns), each corresponding to one issue.
- Issues date from 1740 to 2018.
- Issues belong to 47 different newspapers.
- There are 3 types of access rights: 'closed', 'openpublic' and 'openprivate'.

2.2 Content item table (S3 > dask)

The content item table is much bigger and statistics are therefor much longer to compute (which is why it is stored as a dask dataframe).
The following cells may take quite some time to run.

Shape

In [14]:

```
ci_df.shape[0].compute()
```

Out[14]:

```
47876994
```

Content

In [15]:

```
ci_df.type.unique().compute()
```

Out[15]:

```
0          ar
1         img
2          ad
3     section
4     picture
5        page
6          tb
7          ob
8           w
Name: type, dtype: object
```

In [16]:

```
ci_df.newspaper.unique().compute()
```

Out[16]:

```
0                BDC
1                BLB
2                BNN
3                CDV
4                CON
          ...
71          schmiede
72         tageblatt
73       volkfreu1869
74      waechtersauer
75         waeschfra
Name: newspaper, Length: 76, dtype: object
```

In [17]:

```
ci_df.year.max().compute()
```

Out[17]:

```
2018
```

In [18]:

```
ci_df.year.min().compute()
```

Out[18]:

```
1738
```

In [19]:

```
ci_df.title_length.max().compute()
```

Out[19]:

```
19306.0
```

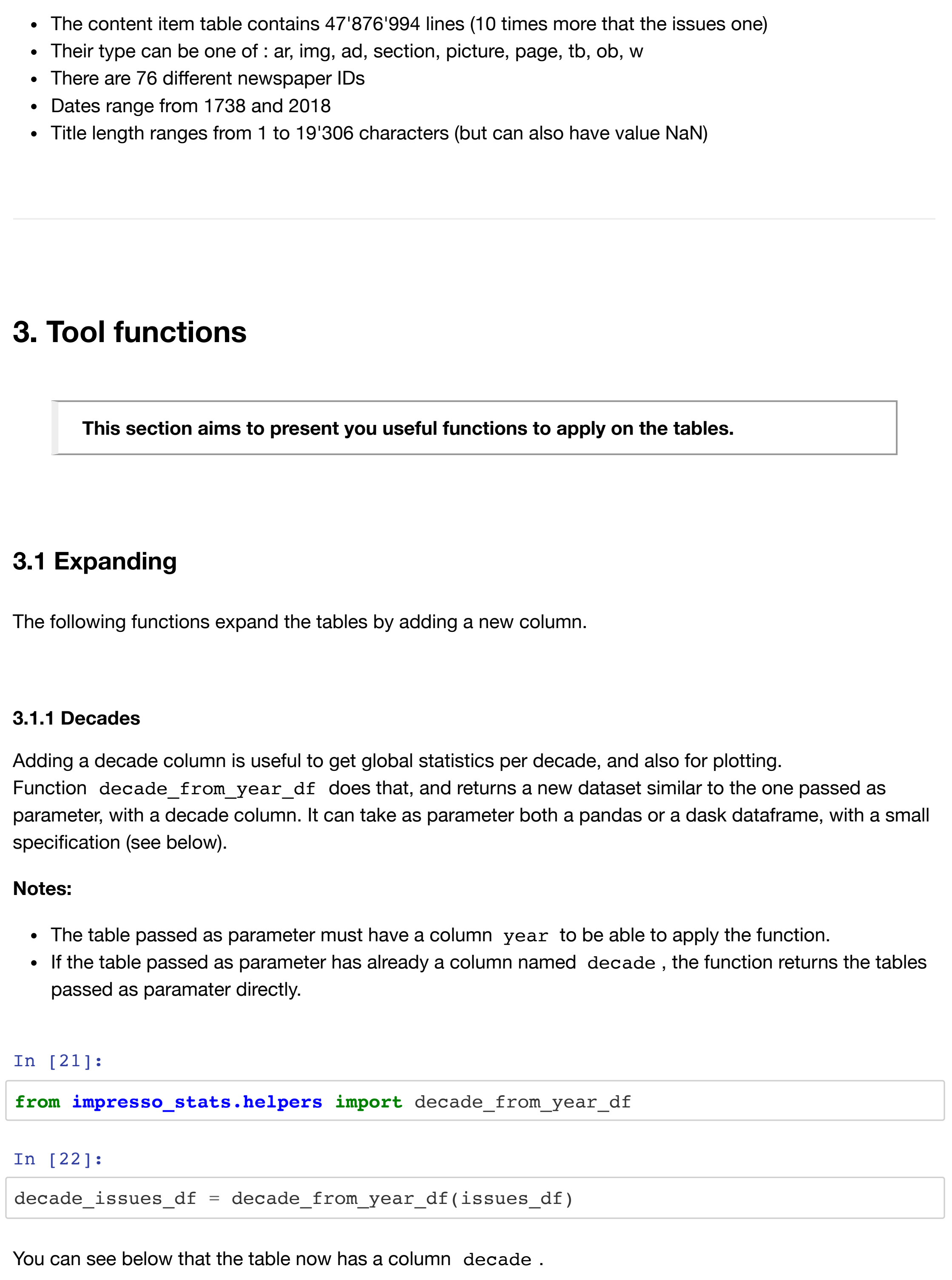
In [20]:

```
ci_df.title_length.min().compute()
```

Out[20]:

```
1.0
```

Conclusion



- The content item table contains 47'876'994 lines (10 times more that the issues one)
- Their type can be one of : ar, img, ad, section, picture, page, tb, ob, w
- There are 76 different newspaper IDs
- Dates range from 1738 and 2018
- Title length ranges from 1 to 19'306 characters (but can also have value NaN)

3. Tool functions

> **This section aims to present you useful functions to apply on the tables.**

3.1 Expanding

The following functions expand the tables by adding a new column.

3.1.1 Decades

Adding a decade column is useful to get global statistics per decade, and also for plotting.
Function `decade_from_year_df` does that, and returns a new dataset similar to the one passed as parameter, with a decade column. It can take as parameter both a pandas or a dask dataframe, with a small specification (see below).

Notes:

- The table passed as parameter must have a column `year` to be able to apply the function.
- If the table passed as parameter has already a column named `decade` , the function returns the tables passed as paramater directly.

In [21]:

```
from impresso_stats.helpers import decade_from_year_df
```

In [22]:

```
decade_issues_df = decade_from_year_df(issues_df)
```

You can see below that the table now has a column `decade` .

In [23]:

```
decade_issues_df.head()
```

Out[23]:

	id	year	month	day	edition	access_rights	created	last_modified	is_damaged
0	actionfem-1927-10-15-a	1927	10	15	a	Closed	2019-06-15 12:22:38	NaT	0
1	actionfem-1927-11-15-a	1927	11	15	a	Closed	2019-06-15 12:22:38	NaT	0
2	actionfem-1927-12-15-a	1927	12	15	a	Closed	2019-06-15 12:22:38	NaT	0
3	actionfem-1928-01-15-a	1928	1	15	a	Closed	2019-06-15 12:22:41	NaT	0
4	actionfem-1928-02-15-a	1928	2	15	a	Closed	2019-06-15 12:22:41	NaT	0

With dask dataframe: When the parameter passed is a dask dataframe, it needs to be specified through the parameter `dask_df` , by setting it to true.

In [24]:

```
decade_ci_df = decade_from_year_df(ci_df, dask_df=True)
```

3.1.2 Access rights at content item level

The content item table doesn't contain information about access rights, as the access right type is defined at the issues level. In order to get the access right for each content item, you need to join the two tables. The function `licenses_ci_df` does it directly for you.

It loads the issues table from SQL, extracts the issues ID in the table passed as parameter, and merges the two.

Notes:

- This function is **very sensitive to any change in the table's shape or in the column names** and needs specific entries :
  - the table passed as parameter must have the following columns (named exactly that way): `'id'` , `'newspaper'` , `'decade'` , `'year'` , `'month'` , `'day'` , `'type'` , `'n_tokens'` , `'title_length'` , `'access_rights'` . Else you will need to modify the function directly. (In particular, the `decade` column is needed here)
  - its columns `id` must have a specific format (same as the one on the snapshot presented above), from which we can extract the ID of the issue.
  - the `issues` table (which is loaded directly in the function) must have the same columns as the one presented earlier in the notebook.
- The **number of entries in the merged table is lower that the number of entries in the original content item dataframe**. In fact some newspaper ID are not represented in the issues dataframe (which contains the access rights) and those will therefore be dropped in the merged dataframe as we don't have information on the access right for those (the function does an inner join for this reason). >> Therefore we recommend using the table output by the function, only when you are doing an analysis related to access rights. For any other analysis we recommend using the original `ci_df` in order to keep as much information as possible.

In [25]:

```
from impresso_stats.helpers import licenses_ci_df
```

In [26]:

```
ci_licences_df = licenses_ci_df(decade_ci_df)
```

In [27]:

```
ci_licences_df.shape[0].compute()
```

Out[27]:

```
33707113
```

The resulting dataframe has 33'707'113 entries : around 14mio less than the original.

You can see below that the table now has a column `access_rights` .

In [28]:

```
ci_licences_df.head()
```

Out[28]:

	id	newspaper	decade	year	month	day	type	n_tokens	title_length	access_rights
0	BDC-1839-01-20-a-i0001	BDC	1830	1839	1	20	ar	250.0	4.0	OpenPublic
1	BDC-1839-01-20-a-i0002	BDC	1830	1839	1	20	ar	758.0	NaN	OpenPublic
2	BDC-1839-01-20-a-i0003	BDC	1830	1839	1	20	ar	14.0	31.0	OpenPublic
3	BDC-1839-01-20-a-i0004	BDC	1830	1839	1	20	ar	349.0	22.0	OpenPublic
4	BDC-1839-01-20-a-i0005	BDC	1830	1839	1	20	ar	193.0	NaN	OpenPublic

3.2 Filtering

Function `filter_df` takes as parameter a table (dask or pandas) and filtering parameters, and returns a tuple : the filtered table and the IDs of the newspapers which have been kept. It can take the following filters :

- `start_date` and `end_date` : indicate the **years** between which you can to keep the entries.
    - you need to specify **either both either none**: if you specify only one the filter won't be applied.
    - you must have `start_date <= end_date` (of course).
- `np_ids` : indicate the specific list of newspapers you want to keep.
    - the list can either be specified by a python list, or by a pandas series.
- `country` : indicate the country indicator of the newspapers you want to keep (typically 'CH' or 'LU').
- `ppty` and `ppty_value` : indicate the property of the newspapers you want to keep.

Note: attributes `country` and `ppty` are defined at the newspaper level (higher level than issues or content items). When specified, the function will load data from the SQL tables `newspapers_metadata` and `meta_properties` . Filtering with these characteristics is done based on the information in those tables. If they are not up-to-date, the output may not be correct.

For later: this function is used as a helper function to most visualization functions (those which enable filtering) you will see in tutorials 2 and 3.

In [29]:

```
from impresso_stats.helpers import filter_df
```

Examples

Examples are shown with `issues_df` for a matter of computation time, but work exactly the same way with a dask dataframe (`ci_df` - in which case a dask dataframe is returned as 1st value of the return tuple).

We provide examples with one specific filter, so that you see how to use each of them. Of course you can **combine** them and specify several filters at the same time.

3.2.1. Filtering by date

Here we keep the issues between 1932 and 1956 (**comprised**).

In [30]:

```
output_table, np_ids = filter_df(issues_df, start_date=1932, end_date=1956)
```

In [31]:

```
output_table.shape
```

Out[31]:

```
(83930, 11)
```

In [32]:

```
output_table.head()
```

Out[32]:

	id	year	month	day	edition	access_rights	created	last_modified	is_damaged
40	actionfem-1932-01-15-a	1932	1	15	a	Closed	2019-06-15 12:22:53	NaT	0
41	actionfem-1932-02-15-a	1932	2	15	a	Closed	2019-06-15 12:22:53	NaT	0
42	actionfem-1932-03-15-a	1932	3	15	a	Closed	2019-06-15 12:22:53	NaT	0
43	actionfem-1932-04-15-a	1932	4	15	a	Closed	2019-06-15 12:22:53	NaT	0
44	actionfem-1932-05-15-a	1932	5	15	a	Closed	2019-06-15 12:22:53	NaT	0

Here are the newspaper IDs which have been kept, i.e. the newspapers which have issues published between 1932 and 1956.

In [33]:

```
np_ids
```

Out[33]:

```
array(['actionfem', 'demitock', 'dunioun', 'EXP', 'GDL', 'IMP',
       'indeplux', 'JDG', 'LCE', 'LES', 'LSE', 'luxembourg1935',
       'luxland', 'luxwort', 'NZZ', 'obermosel', 'onsjongen'],
      dtype=object)
```

3.2.2. Filtering by newspaper IDs

Here we keep only two newspapers : *Gazette de Lausanne* (*GDL*) and *Journal de Genève* (*JDG*). (In that case the second return value `np_ids` is not very useful).

In [34]:

```
output_table, np_ids = filter_df(issues_df, np_ids=['GDL', 'JDG'])
```

In [35]:

```
output_table.shape
```

Out[35]:

```
(100201, 11)
```

In [36]:

```
output_table.head()
```

Out[36]:

	id	year	month	day	edition	access_rights	created	last_modified	is_damaged
60257	GDL-1798-02-01-a	1798	2	1	a	OpenPrivate	2019-06-18 07:08:55	2019-06-25 12:48:23	0
60258	GDL-1798-02-02-a	1798	2	2	a	OpenPrivate	2019-06-18 07:08:55	2019-06-25 12:48:23	0
60259	GDL-1798-02-03-a	1798	2	3	a	OpenPrivate	2019-06-18 07:08:55	2019-06-25 12:48:23	0
60260	GDL-1798-02-04-a	1798	2	4	a	OpenPrivate	2019-06-18 07:08:55	2019-06-25 12:48:23	0
60261	GDL-1798-02-05-a	1798	2	5	a	OpenPrivate	2019-06-18 07:08:55	2019-06-25 12:48:23	0

3.2.3. Filtering by country

Here we keep only the Swiss newspapers - with country value 'CH'. If you want to get the Luxembourgian newspapers, use value 'LU'.

In [37]:

```
output_table, np_ids = filter_df(issues_df, country='CH')
```

In [38]:

```
output_table.shape
```

Out[38]:

```
(356196, 11)
```

In [39]:

```
output_table.head()
```

Out[39]:

	id	year	month	day	edition	access_rights	created	last_modified	is_damaged
2012	BDC-1839-01-20-a	1839	1	20	a	OpenPublic	2019-06-17 11:53:25	NaT	0
2013	BDC-1839-01-21-a	1839	1	21	a	OpenPublic	2019-06-17 11:53:25	NaT	0
2014	BDC-1839-01-23-a	1839	1	23	a	OpenPublic	2019-06-17 11:53:25	NaT	0
2015	BDC-1839-01-26-a	1839	1	26	a	OpenPublic	2019-06-17 11:53:25	NaT	0
2016	BDC-1839-01-28-a	1839	1	28	a	OpenPublic	2019-06-17 11:53:25	NaT	0

3.2.4. Filtering by property

Here is the list of properties you can use :
(To have more information on the propertie values, load the tables `meta_properties` , `newspapers_metadata` , `languages` , or `newspapers_languages` .)

['language', 'archivalHolder', 'countryCode', 'currentPeriodicity', 'deliveredIssueNb', 'deliveredPageNb', 'editor', 'expectedIssueNb', 'expectedPageNb', 'formerPeriodicity', 'founder', 'ingestedIssueNb', 'ingestedPageNb', 'isbn', 'longTitle', 'noteGenealogy', 'notePublicationDates', 'otherTitle', 'periodicity', 'polOrientation', 'pressType', 'provinceCode', 'publicationDates', 'publicationPlace', 'publisher', 'relatedUrl', 'variantTitle', 'provenanceId', 'provenanceSource', 'provenanceUri', 'logoFilename', 'availabilityEta']

Language
Keep newspapers written in French.

In [40]:

```
output_table, np_ids = filter_df(issues_df, ppty='language', ppty_value='fr')
```

In [41]:

```
output_table.shape
```

Out[41]:

```
(306841, 11)
```

In [42]:

```
output_table.head()
```

Out[42]:

	id	year	month	day	edition	access_rights	created	last_modified	is_damaged
0	actionfem-1927-10-15-a	1927	10	15	a	Closed	2019-06-15 12:22:38	NaT	0
1	actionfem-1927-11-15-a	1927	11	15	a	Closed	2019-06-15 12:22:38	NaT	0
2	actionfem-1927-12-15-a	1927	12	15	a	Closed	2019-06-15 12:22:38	NaT	0
3	actionfem-1928-01-15-a	1928	1	15	a	Closed	2019-06-15 12:22:41	NaT	0
4	actionfem-1928-02-15-a	1928	2	15	a	Closed	2019-06-15 12:22:41	NaT	0

Canton

Keep newspapers from the canton of Vaud (Switzerland - VD).

In [43]:

```
output_table, np_ids = filter_df(issues_df, ppty='provinceCode', ppty_value='VD'
)
```

In [44]:

```
output_table.shape
```

Out[44]:

```
(100201, 11)
```

In [45]:

```
output_table.head()
```

Out[45]:

	id	year	month	day	edition	access_rights	created	last_modified	is_damaged
60257	GDL-1798-02-01-a	1798	2	1	a	OpenPrivate	2019-06-18 07:08:55	2019-06-25 12:48:23	0
60258	GDL-1798-02-02-a	1798	2	2	a	OpenPrivate	2019-06-18 07:08:55	2019-06-25 12:48:23	0
60259	GDL-1798-02-03-a	1798	2	3	a	OpenPrivate	2019-06-18 07:08:55	2019-06-25 12:48:23	0
60260	GDL-1798-02-04-a	1798	2	4	a	OpenPrivate	2019-06-18 07:08:55	2019-06-25 12:48:23	0
60261	GDL-1798-02-05-a	1798	2	5	a	OpenPrivate	2019-06-18 07:08:55	2019-06-25 12:48:23	0

4. impresso_stats

This was the first tutorial of a sequence of three. The following tutorials are:

- Issues*tutorial*(2) : tutorial on how to use fonctions of the library for statistics at the issues level.
- Content_items*tutorial*(3) : tutorial on how to use functions of the library for statistics at the content items level.

B Issues tutorial notebook

This is the tutorial focused on issues exploration.

Tutorial 2 - Analysis at the issue level

This is the 2nd tutorial notebook, following the 1st one: `Introductory_tutorial_(1)` .
This tutorial aims to explain you how to use the functions made for analysis **at the issue level**.

Analysis at the issue level has been focused on issue frequency through time, and access right statistics (license type).

If you do not understand some parts of the notebook, in particular for any part regarding loading datasets, transformation of tables etc, please refer to the first tutorial notebook 1 : `Introductory_tutorial_(1)` .

Table of Content

Introduction

Setup

1. Issue frequency

- 1.1 Naive example
- 1.2 Loading data
- 1.3 Filtering
  - 1.3.1 Specify newspapers
  - 1.3.2 Specify year bounds
- 1.4 Group plotting

2. Licences

- 1.1 Stats table
- 1.2 Plotting

Introduction

There are 2 groups of functions that you may use for the issues analysis :

- Analysis of **issues frequency** : `plt_freq_issues_time`
- Analysis of **access rights frequency** at the issues level : `multiple_ar_np` , `license_stats_table` , `plot_licences`

Both functions `plt_freq_issues_time` and `plot_licences` display a plot, and return a pandas dataframe.

The other two functions for access rights help providing statistics.

Setup

In [1]:

```
%load_ext autoreload
%autoreload 2
```

In [2]:

```
# Specify path for imports
import os
import sys
module_path = os.path.abspath(os.path.join('..'))
if module_path not in sys.path:
    sys.path.append(module_path)
```

Plotting setup

For better rendering, we recommand importing the seaborn python library (https://seaborn.pydata.org/) and using a grid (white or dark, depending on your preference).

In [3]:

```
import seaborn as sns
sns.set(style="darkgrid")
#sns.set(style="whitegrid")
```

1. Issue frequency

The main function regarding issue frequency is `plt_freq_issues_time` .
It computes the number of issues by newspaper by year (or by decade, depending on parameter `time_gran`), and displays it on a plot. The result of the operation (a pandas dataframe) is also returned.

You can pass the function several **filters** (see tutorial (1)), about years, newspapers, country, properties. It will call the function `filter_df` , presented in the 1st tutorial.

This function is "all in one". What we mean by that is that it does all for you :

- loading the sql issues table as a pandas dataframe,
- adding the necessary columns (e.g. decade),
- grouping by and counting

You only need to specify one parameter : `time_gran` , which can be either `'year'` or `'decade'` , depending on how precise ou want the frenquency statistic to be.

In [4]:

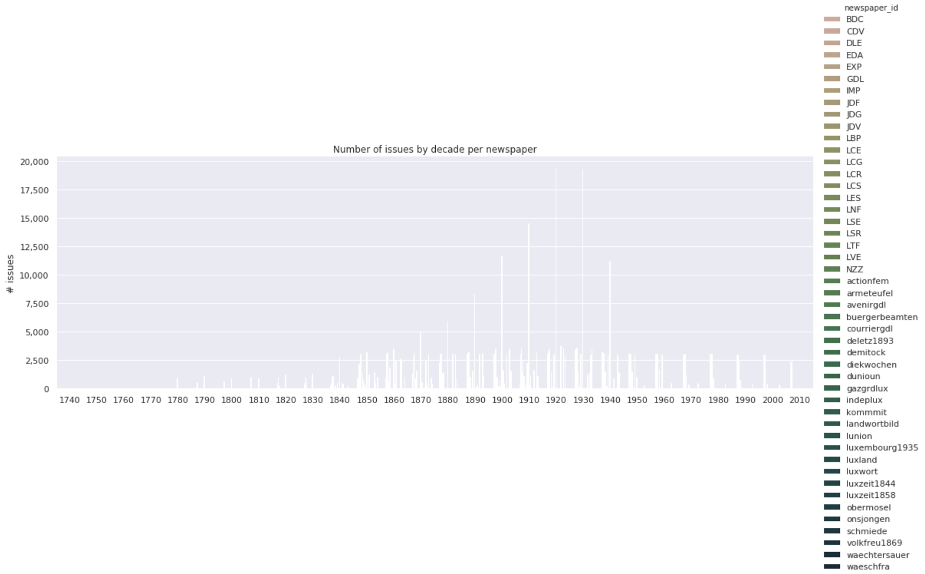
```
from impresso_stats.visualization import plt_freq_issues_time
```

1.1 Naive example

Let's first call the function in the simplest way, to give you an idea on what it does. The minimum you have to specify is the "time granularity", i.e. `'year'` or `'decade'` .

In [5]:

```
df = plt_freq_issues_time('decade')
```



Snapshot of the returned dataframe

In [6]:

```
df.head()
```

Out[6]:

	newspaper_id	decade	count
0	BDC	1740	0.0
1	BDC	1750	0.0
2	BDC	1760	0.0
3	BDC	1770	0.0
4	BDC	1780	0.0

Explanation

As there is a lot of newspapers (and also a lot of decades), this plot shows nothing: we can't see the colors. You have two options to make it more readable : **plot by batch** or **filter** (or both !).

The two options and explained below.

Remark

As said above, the function does all for you, but each time you call it, it reloads the table, adds the decade columns etc. If you are going to try several visualization (as we will do now), we recommend that you load the table as a pandas dataframe yourself, and do all necessary preprocessing ourself once, and then **pass the final dataframe as parameter to the function**. It will be computationnally lighter (and thus faster).

1.2 Loading data

If any part of this section seem unclear, please refer to the 1st tutorial notebook `Introductory_tutorial_(1).ipynb` .

In [7]:

```
from impresso_stats.sql import db_engine, read_table
from impresso_stats.helpers import decade_from_year_df
```

In [8]:

```
engine = db_engine()
issues_df = read_table('impresso.issues', engine)
```

In [9]:

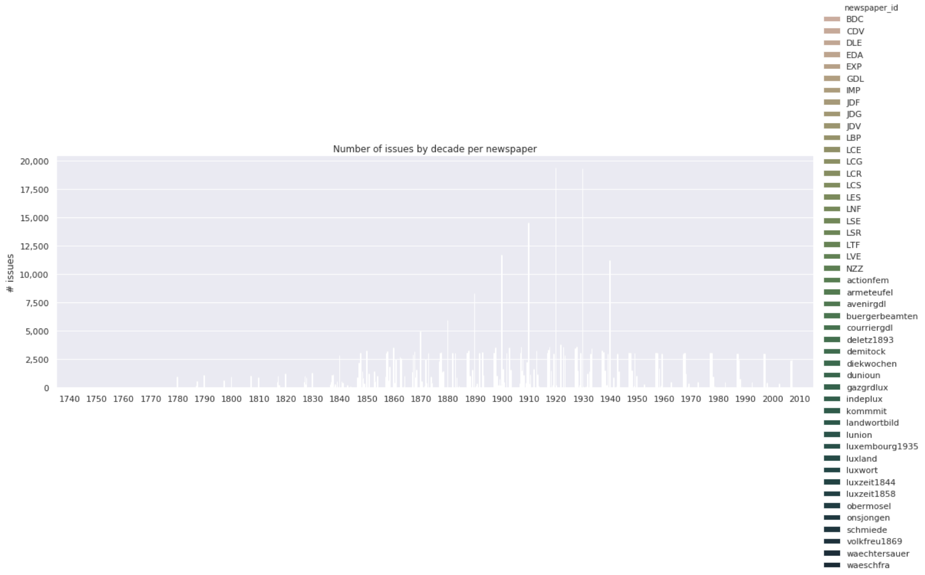
```
issues_df = decade_from_year_df(issues_df)
```

Calling the function with a dataframe

Now that you have loaded the dataframe, you would call the function the same way as before, but with a new parameter.

In [10]:

```
# This does exactly the same as above.
df = plt_freq_issues_time('decade', issues_df)
```



1.3 Filtering

You can apply any kind of filter, as shown in tutorial (1).

Let's see some examples.

1.3.1 Specify newspapers

Let's say you would like to know the number of issues published for two specific newspapers (on the same plot) : *Gazette de Lausanne (GDL)* and *Journal de Genève (JDG)*, for each year.

Remark

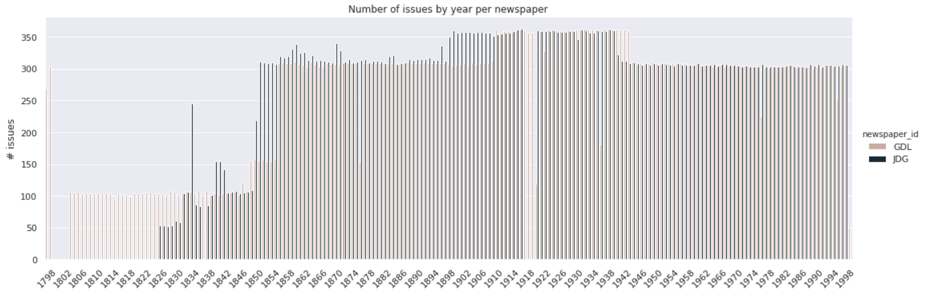
As you only have two newspapers here, it is fine to plot by year, but if you have more and the newspapers have issues over a lot of years, if may not render well. You may have to chose specific years, or plot by decade.

In [11]:

```
sns.set_style('darkgrid')
```

In [12]:

```
df = plt_freq_issues_time('year', np_ids=['JDG', 'GDL'], df=issues_df)
```



Explanation

As there are issues for a lot of years, only one year on 4 is displayed on the x axis.
In this example, several periods might seem interesting :

- (a) *JDG* seems to have several gaps : around 1834/1838, around 1914/1922
- (b) *GDL* seems to have particular years around : 1874, 1920, 1936, ... where you see drops in the number of issues.
- (c) between 1846 and 1858, it may be interesting to compare the number issues for both journals as they evolve differently.

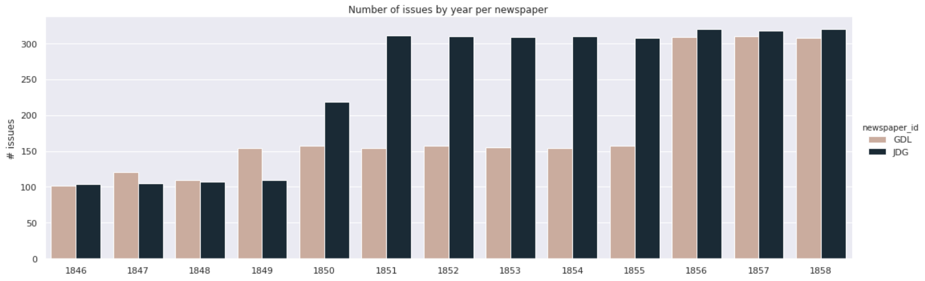
You might want to see more closely what happens and at which date exactly. For that purpose you need to "zoom" on the plot, by specifying the years you would like to see. Let's see an example below.

1.3.2 Specify year bounds

Let's see how to "zoom" for (c).

In [13]:

```
df = plt_freq_issues_time('year', np_ids=['JDG', 'GDL'], \
                          df=issues_df, start_date=1846, end_date=1858)
```



Explanation

Here you can see more clearly that :

- between 1848 and 1856 the number of issues for both journals has been multipled by 3 (from 100 to 300)
- the *Journal de Genève* was the first to increase its number of issues, progressively : 100 in 1849 / 200 in 1850 / 300 in 1851 (approximately)
- the *Gazette de Lausanne* first increased a little bit its number of issues from 100 to approximately 150 in 1849, and then more drastically 1856, to reach the same number as *Journal de Genève*, 5 years later.

It may be interesting to see if one increase could have influenced the other, if the same phenomenon has happenned around the same years for other journals, if it can be related to historical events, etc.

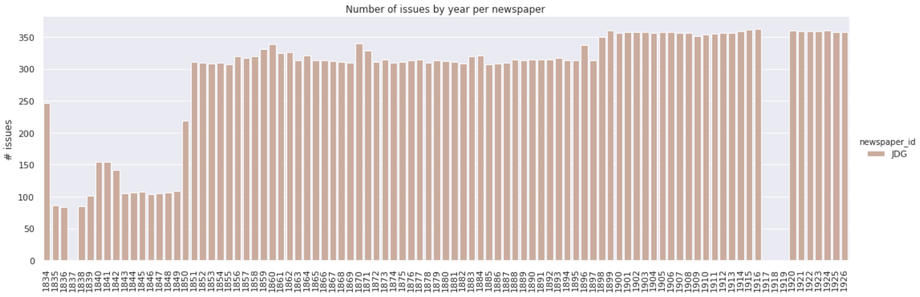
If you look again at the first plot comparing the two journals, you can see that a similar phenomenon seem to happen around 1894/1910, and 1938/1946.

You now have the tools to investigate deeper on this !

Let's see how to zoom for (a)

In [14]:

```
df = plt_freq_issues_time('year', np_ids=['JDG'], df=issues_df, start_date=1834,
end_date=1926)
```



Explanation

Here you can see more clearly which years are missing :

- 1837
- between 1917 and 1919 (included)

It may be interesting to see why there years are missing : did *Journal de Genève* not publish any issue during these years for a particular reason ? Are they missing from the dataset ? And in that case, why ?

1.4 Group plotting

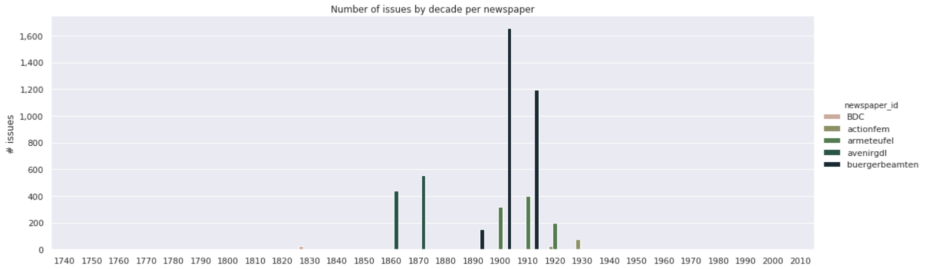
In section 1.3, you have seen how to filter. The other option for plotting a lot of data, if you do not want to drop any information is to **plot "by batch"**.

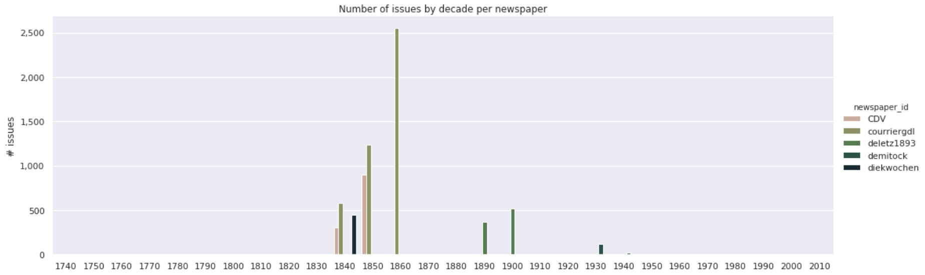
By that we mean that you can specify a parameter `batch_size` (typically between 2 and 6), indicating the number of newspapers you want to be displayed on each plot.

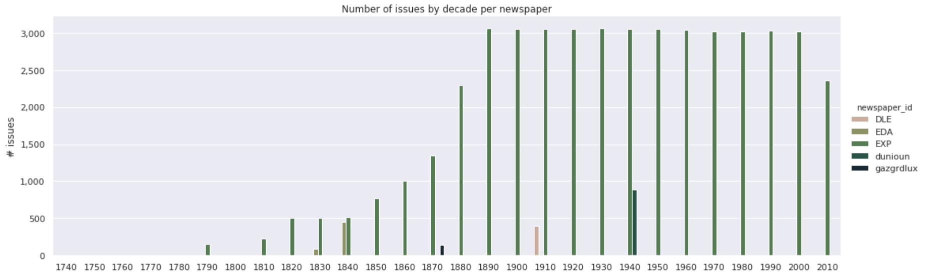
Let's see an example for better understanding.

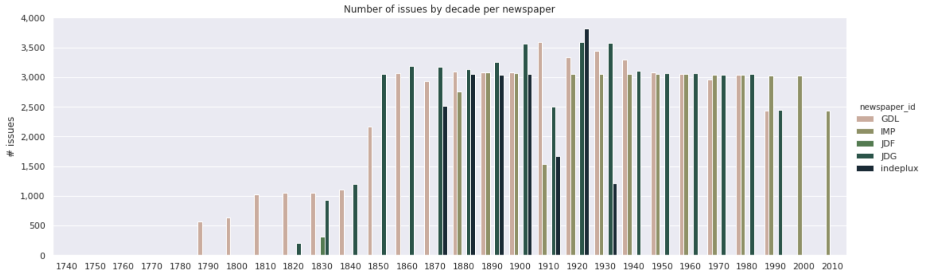
```
In [15]:
```

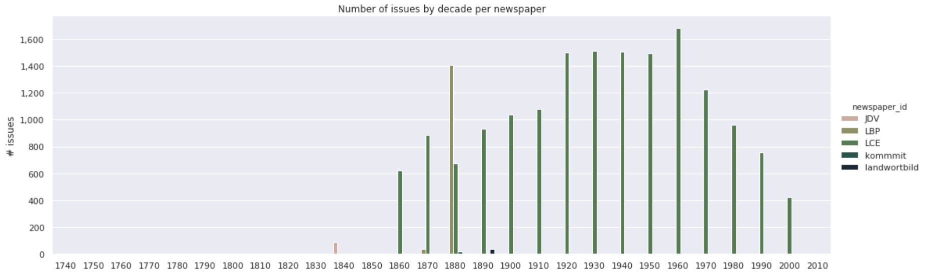
```
df = plt_freq_issues_time('decade', issues_df, batch_size=5)
```

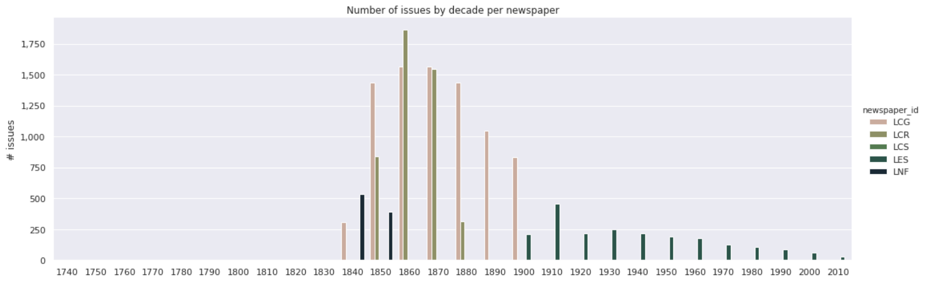

Number of issues by decade per newspaper
# issues
0
200
400
600
800
1,000
1,200
1,400
1,600
1740 1750 1760 1770 1780 1790 1800 1810 1820 1830 1840 1850 1860 1870 1880 1890 1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000 2010
newspaper_id
BDC
actionfem
armeteufel
avenirgdl
buergerbeamten

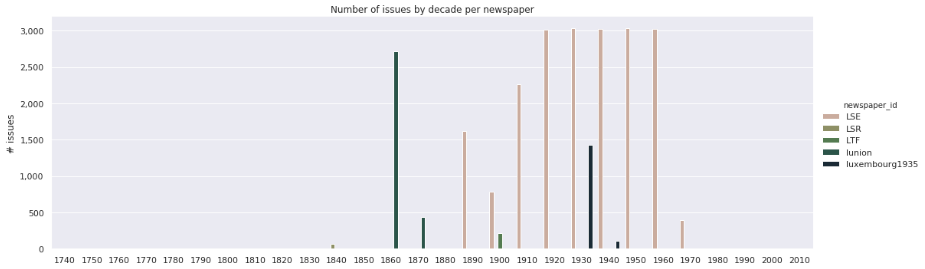

Number of issues by decade per newspaper
# issues
0
500
1,000
1,500
2,000
2,500
1740 1750 1760 1770 1780 1790 1800 1810 1820 1830 1840 1850 1860 1870 1880 1890 1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000 2010
newspaper_id
CDV
courriergdl
deletz1893
demitock
diekwochen

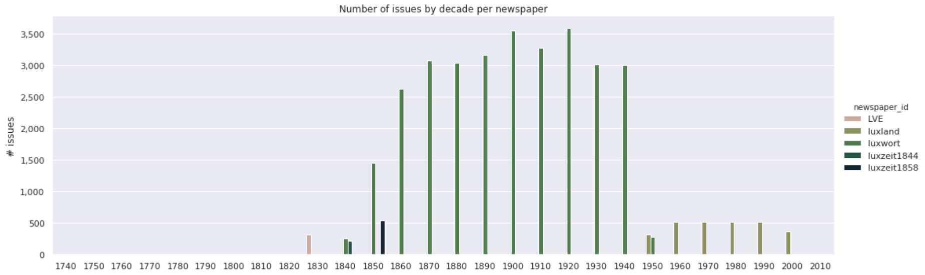

Number of issues by decade per newspaper
# issues
0
500
1,000
1,500
2,000
2,500
3,000
1740 1750 1760 1770 1780 1790 1800 1810 1820 1830 1840 1850 1860 1870 1880 1890 1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000 2010
newspaper_id
DLE
EDA
EXP
dunioun
gazgrdlux

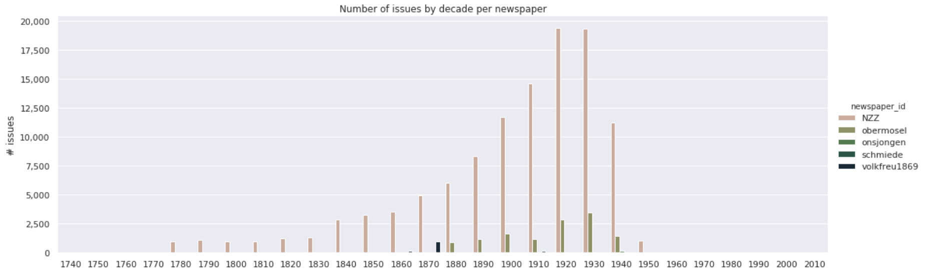

Number of issues by decade per newspaper
# issues
0
500
1,000
1,500
2,000
2,500
3,000
3,500
4,000
1740 1750 1760 1770 1780 1790 1800 1810 1820 1830 1840 1850 1860 1870 1880 1890 1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000 2010
newspaper_id
GDL
IMP
JDF
JDG
indeplux

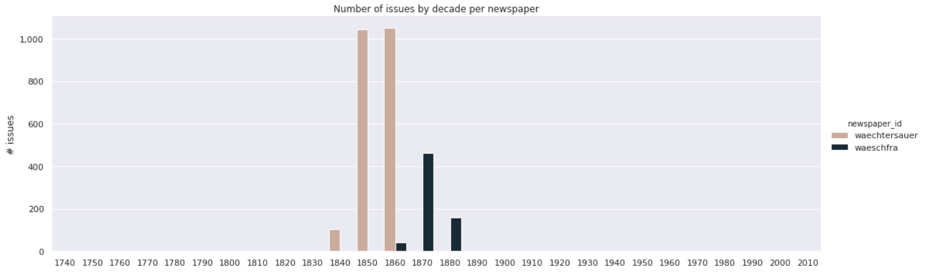

Number of issues by decade per newspaper
# issues
0
200
400
600
800
1,000
1,200
1,400
1,600
1740 1750 1760 1770 1780 1790 1800 1810 1820 1830 1840 1850 1860 1870 1880 1890 1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000 2010
newspaper_id
JDV
LBP
LCE
kommmit
landwortbild


Number of issues by decade per newspaper
# issues
0
250
500
750
1,000
1,250
1,500
1,750
1740 1750 1760 1770 1780 1790 1800 1810 1820 1830 1840 1850 1860 1870 1880 1890 1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000 2010
newspaper_id
LCG
LCR
LCS
LES
LNF


Number of issues by decade per newspaper
# issues
0
500
1,000
1,500
2,000
2,500
3,000
1740 1750 1760 1770 1780 1790 1800 1810 1820 1830 1840 1850 1860 1870 1880 1890 1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000 2010
newspaper_id
LSE
LSR
LTF
lunion
luxembourg1935


Number of issues by decade per newspaper
# issues
0
500
1,000
1,500
2,000
2,500
3,000
3,500
1740 1750 1760 1770 1780 1790 1800 1810 1820 1830 1840 1850 1860 1870 1880 1890 1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000 2010
newspaper_id
LVE
luxland
luxwort
luxzeit1844
luxzeit1858


Number of issues by decade per newspaper
# issues
0
2,500
5,000
7,500
10,000
12,500
15,000
17,500
20,000
1740 1750 1760 1770 1780 1790 1800 1810 1820 1830 1840 1850 1860 1870 1880 1890 1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000 2010
newspaper_id
NZZ
obermosel
onsjongen
schmiede
volkfreu1869


Number of issues by decade per newspaper
# issues
0
200
400
600
800
1,000
1740 1750 1760 1770 1780 1790 1800 1810 1820 1830 1840 1850 1860 1870 1880 1890 1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000 2010
newspaper_id
waechtersauer
waeschfra

Explanation

The x axis labelling is the same for all, whereas the y axis depends on each plot.

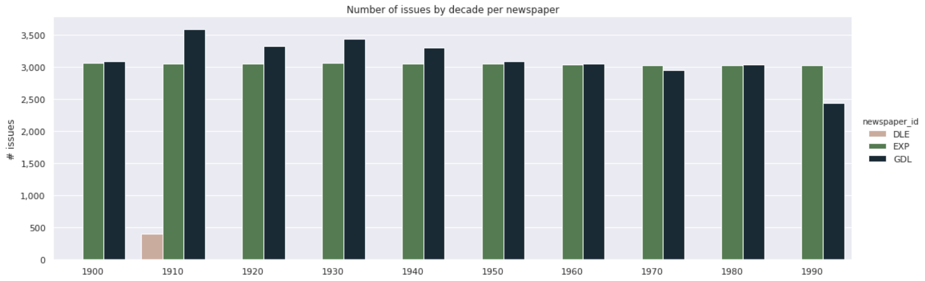
The ordering in which newspapers are grouped is simply the alphabetical order. Therefore as you can see on the plots above, the groups may not be very significants :

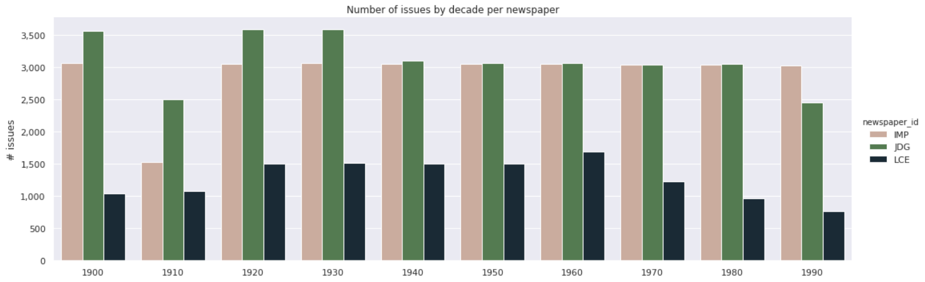
- there are several plots on which the years on which newspapers have published issues do not overlap at all,
- there are several plots on which the number of issues published for the different newspaper is very different and not easily comparable (e.g. if one is on average 10 times larger that the others).

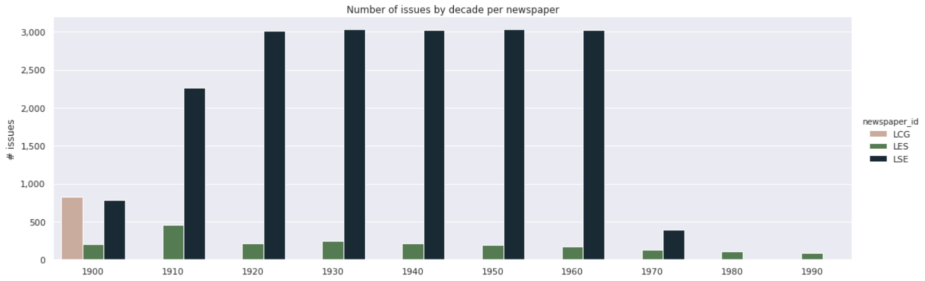
Using filtering combined with batches can lead to better results. You may want to see only Swiss newspapers, only during the 19th century for example (see below).

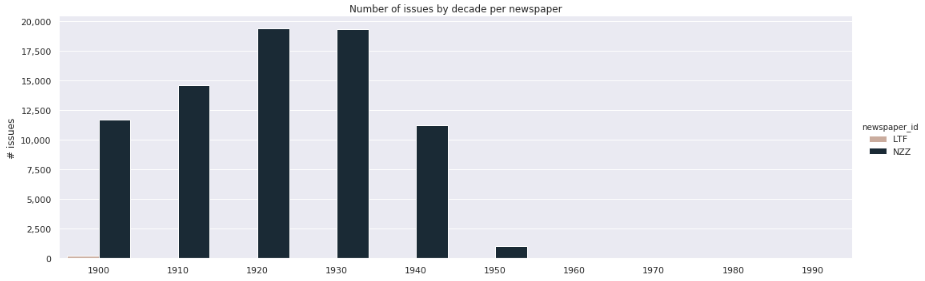
In [16]:

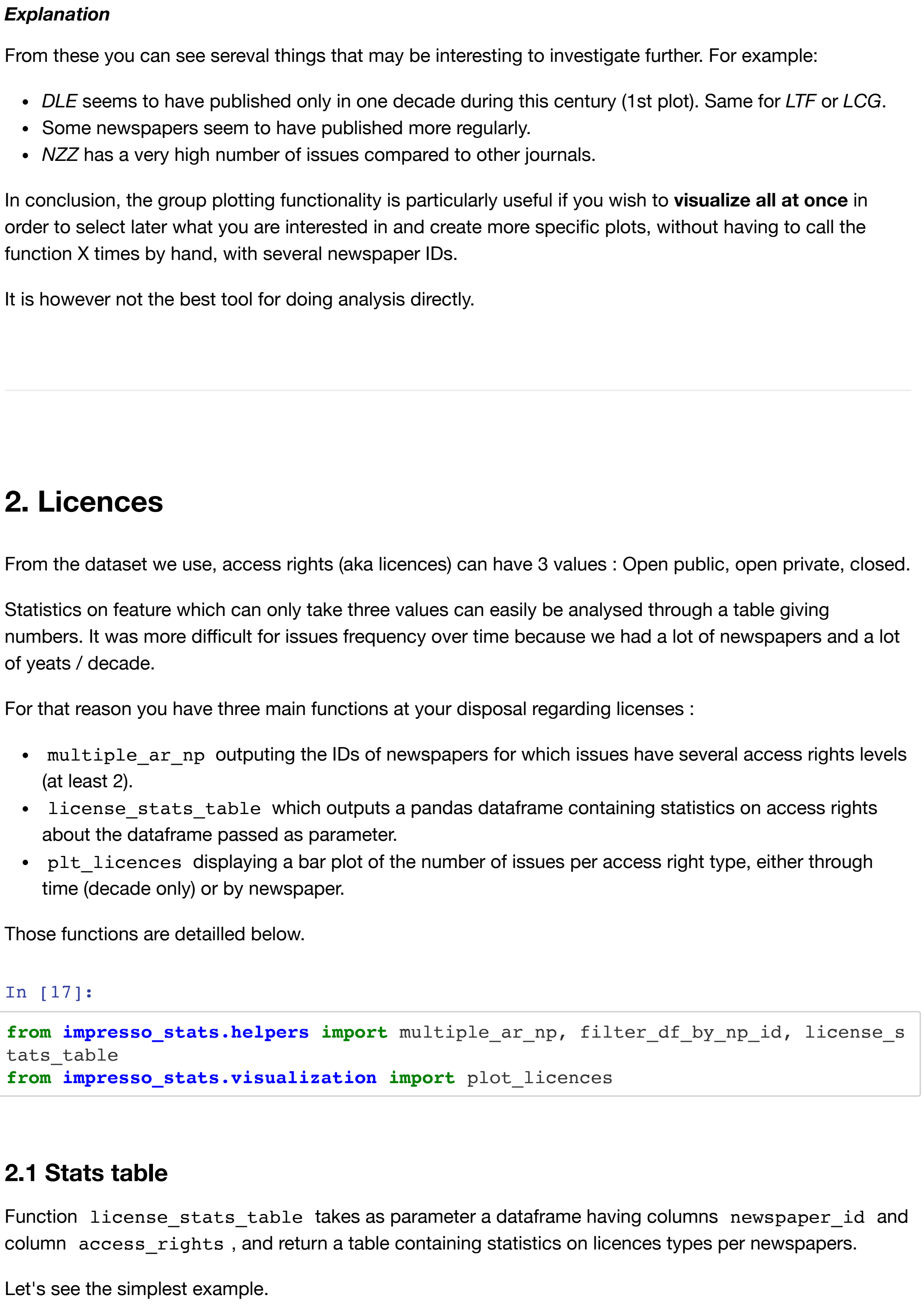
```
df = plt_freq_issues_time('decade', start_date=1900, end_date=1999, df=issues_df
, batch_size=3, country='CH')
```











Explanation

From these you can see sereval things that may be interesting to investigate further. For example:

- *DLE* seems to have published only in one decade during this century (1st plot). Same for *LTF* or *LCG*.
- Some newspapers seem to have published more regularly.
- *NZZ* has a very high number of issues compared to other journals.

In conclusion, the group plotting functionality is particularly useful if you wish to **visualize all at once** in order to select later what you are interested in and create more specific plots, without having to call the function X times by hand, with several newspaper IDs.

It is however not the best tool for doing analysis directly.

2. Licences

From the dataset we use, access rights (aka licences) can have 3 values : Open public, open private, closed.

Statistics on feature which can only take three values can easily be analysed through a table giving numbers. It was more difficult for issues frequency over time because we had a lot of newspapers and a lot of yeats / decade.

For that reason you have three main functions at your disposal regarding licenses :

- `multiple_ar_np` outputing the IDs of newspapers for which issues have several access rights levels (at least 2).
- `license_stats_table` which outputs a pandas dataframe containing statistics on access rights about the dataframe passed as parameter.
- `plt_licences` displaying a bar plot of the number of issues per access right type, either through time (decade only) or by newspaper.

Those functions are detailled below.

In [17]:

```
from impresso_stats.helpers import multiple_ar_np, filter_df_by_np_id, license_s
tats_table
from impresso_stats.visualization import plot_licences
```

2.1 Stats table

Function `license_stats_table` takes as parameter a dataframe having columns `newspaper_id` and column `access_rights` , and return a table containing statistics on licences types per newspapers.

Let's see the simplest example.

In [18]:

```
stats_table = license_stats_table(issues_df)
```

In [19]:

```
stats_table
```

Out[19]:

access_rights	newspaper_id	Closed	OpenPrivate	OpenPublic	Total	rate_Closed	rate_
0	BDC	0	0	23	23	0	
1	CDV	0	0	1,200	1,200	0	
2	DLE	0	402	0	402	0	
3	EDA	0	0	544	544	0	
4	EXP	0	46,196	8	46,204	0	
5	GDL	0	51,062	8	51,070	0	
6	IMP	0	40,227	1	40,228	0	
7	JDF	0	0	319	319	0	
8	JDG	0	49,131	0	49,131	0	
9	JDV	0	92	0	92	0	
10	LBP	0	0	1,440	1,440	0	
11	LCE	0	16,291	1	16,292	0	
12	LCG	0	0	8,185	8,185	0	
13	LCR	0	0	4,561	4,561	0	
14	LCS	0	0	2	2	0	
15	LES	0	2,120	0	2,120	0	
16	LNF	0	0	924	924	0	
17	LSE	0	20,182	1	20,183	0	
18	LSR	0	0	71	71	0	
19	LTF	0	0	217	217	0	
20	LVE	0	0	312	312	0	
21	NZZ	112,676	0	0	112,676	100.0000%	
22	actionfem	101	0	0	101	100.0000%	
23	armeteufel	916	0	0	916	100.0000%	
24	avenirgdl	0	0	995	995	0	
25	buergerbeamten	3,008	0	0	3,008	100.0000%	
26	courriergdl	0	0	4,367	4,367	0	
27	deletz1893	887	0	0	887	100.0000%	
28	demitock	134	0	0	134	100.0000%	
29	diekwochen	0	0	444	444	0	
30	dunioun	891	0	0	891	100.0000%	
31	gazgrdlux	141	0	0	141	100.0000%	
32	indeplux	16,447	0	1,903	18,350	89.6294%	
33	kommmit	22	0	0	22	100.0000%	
34	landwortbild	39	0	0	39	100.0000%	
35	lunion	0	0	3,144	3,144	0	
36	luxembourg1935	1,539	0	0	1,539	100.0000%	

access_rights	newspaper_id	Closed	OpenPrivate	OpenPublic	Total	rate_Closed	rate_
37	luxland	2,772	0	0	2,772	100.0000%	
38	luxwort	16,027	0	14,328	30,355	52.7986%	
39	luxzeit1844	0	0	215	215	0	
40	luxzeit1858	0	0	538	538	0	
41	obermosel	12,563	0	0	12,563	100.0000%	
42	onsjongen	139	0	0	139	100.0000%	
43	schmiede	159	0	0	159	100.0000%	
44	volkfreu1869	0	0	1,087	1,087	0	
45	waechtersauer	0	0	2,199	2,199	0	
46	waeschfra	224	0	436	660	33.9394%	

Explanation

This table is quite sparse, you can see a lot of zeros : most newspapers have issues having a simple access right type.

The most interesting way to use function `license_stats_table` , is on the dataframe containing only newspapers which have issues with several access right types. Typically you would find those newspapers by calling `multiple_ar_np` , then filter you dataframe for keeping only the interesting newspapers, and finally call `license_stats_table` to have a summary of statistics.

Let's see this example below:

In [20]:

```
# Get the IDs of newspapers having
np_multiple_ar = multiple_ar_np(issues_df)
```

In [21]:

```
# This function as a type of filtering. One may also call function filter_df
filtered_df = filter_df_by_np_id(issues_df, np_multiple_ar)
```

In [22]:

```
stats_table_2 = license_stats_table(filtered_df)
```

In [23]:

```
stats_table_2
```

Out[23]:

access_rights	newspaper_id	Closed	OpenPrivate	OpenPublic	Total	rate_Closed	rate_Ope
0	EXP	0	46,196	8	46,204	0	9
1	GDL	0	51,062	8	51,070	0	9
2	IMP	0	40,227	1	40,228	0	9
3	LCE	0	16,291	1	16,292	0	9
4	LSE	0	20,182	1	20,183	0	9
5	indeplux	16,447	0	1,903	18,350	89.6294%	
6	luxwort	16,027	0	14,328	30,355	52.7986%	
7	waeschfra	224	0	436	660	33.9394%	

Explanation

This table is much condensed and contains much interesting insights.

Papers which have issues with several access rights can be grouped into three groups :

- Group 1 : Those for which almost 100% of their issues have the same access right level and for which other access right permissions are very rare. It is actually the case for all swiss newspapers which have several access rights : EXP, GDL, IMP, LCE and LSE. For the last three, only one issue among all has a different access right. We could investigate on whether this is an outlier and what it is due to. Note also that they all have the same access right majority : OpenPrivate (and minority : OpenPublic).
- Group 2 : Those for which most of their issues have the same access right level. It is the case for indeplux : almost 90% of its issues have access right Closed, whereas the left 10% are OpenPublic.
- Group 3 : Those for which the difference between the two access right levels is more balanced. It is the case for luxwort (53% closed vs 47% open public) and waeschfra (34% closed vs 66% open public).

Based on these observations, we can investigate on the following questions :

- Where do the rare values in group 1 come from ? Errors from the dataset ? Particular issue-s ?
- Are the access right levels in group 2 and 3 linked to some other property of the issues or to the date in time ?

2.2 Plotting

Function `plot_licences` takes as parameter:

- a `facet` : can be either of value `time` or value `newspaper` , depending on which dimension you wish to aggregate on, and have as x axis.
- a dataframe with columns 'access_rights' and another of value the same as the `facet` parameter.

It returns the aggregated dataframe and displays a plot on access right types. As for issues frequency through time, you can pass filtering parameters to the function in order to sharpen you results. Exploration of these filters is left to you !

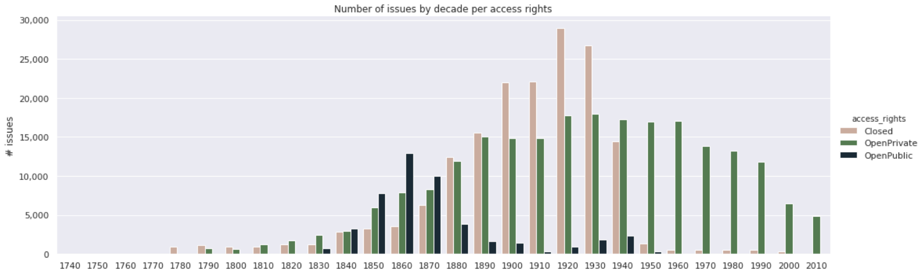
See the following examples.

Time facet

Note: *The only time granularity enabled here is decade.*

In [24]:

```
df = plot_licences(facet='time', df=issues_df)
```



In [25]:

```
df.head()
```

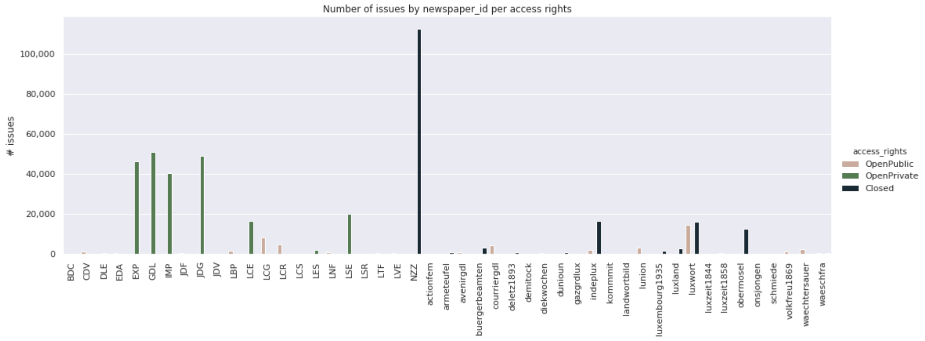
Out[25]:

	access_rights	decade	count
0	Closed	1740	0.0
1	Closed	1750	0.0
2	Closed	1760	0.0
3	Closed	1770	0.0
4	Closed	1780	940.0

Newspaper facet

In [26]:

```
df = plot_licences(facet='newspapers', df=issues_df)
```



In [27]:

```
df.head()
```

Out[27]:

	newspaper_id	access_rights	count
0	BDC	OpenPublic	23
1	CDV	OpenPublic	1200
2	DLE	OpenPrivate	402
3	EDA	OpenPublic	544
4	EXP	OpenPrivate	46196

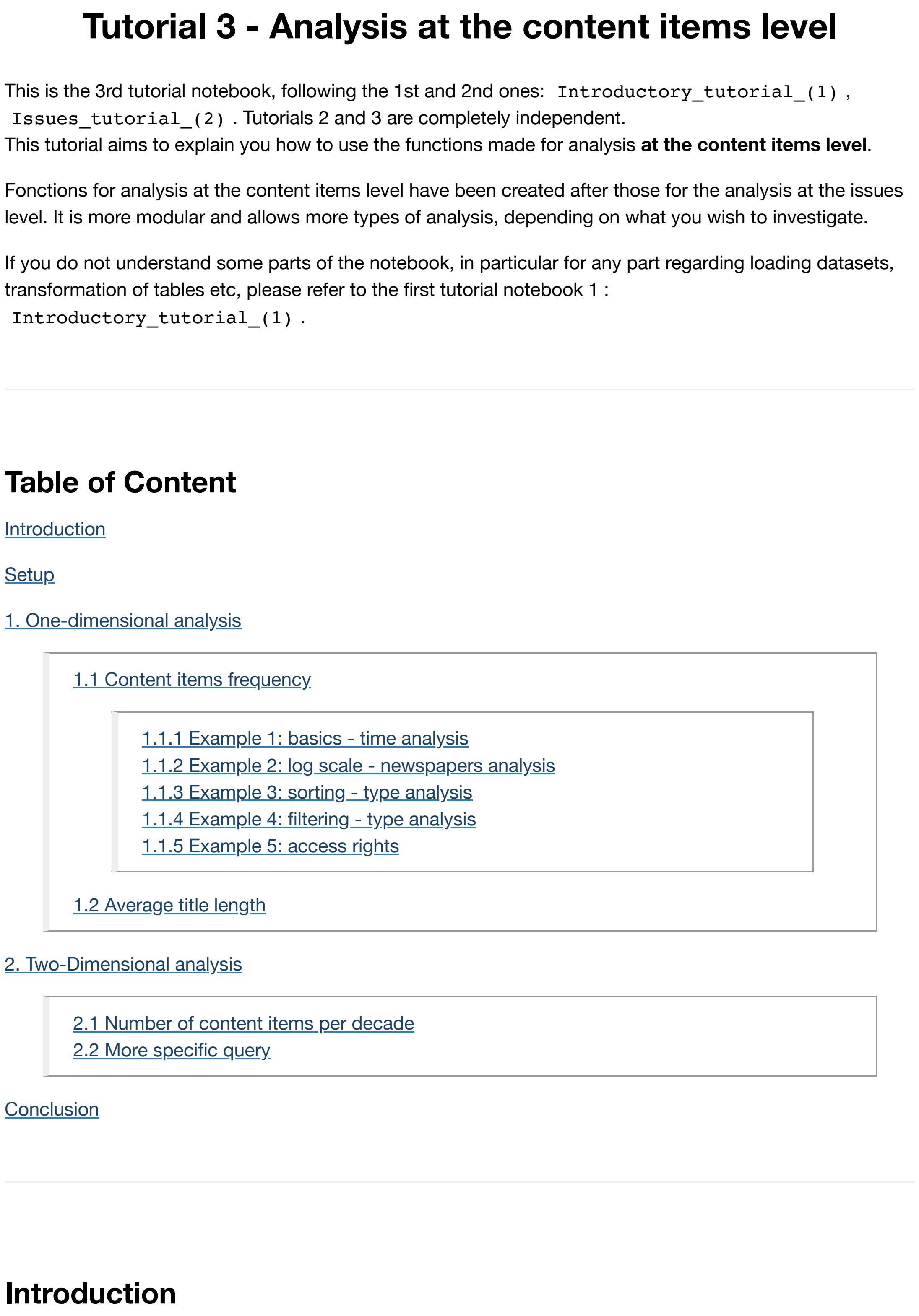
Explanation

This plot shows that there are lots of disparities among journals, but is not very representative. In these cases the statistics table on access rights may give better information.

On can also try to plot in logarithmic scale, as well as applying several filters.

C Content items tutorial notebook

This is the last tutorial, focused on content items analysis.



Tutorial 3 - Analysis at the content items level

This is the 3rd tutorial notebook, following the 1st and 2nd ones: `Introductory_tutorial_(1)` , `Issues_tutorial_(2)` . Tutorials 2 and 3 are completely independent.
This tutorial aims to explain you how to use the functions made for analysis **at the content items level**.

Fonctions for analysis at the content items level have been created after those for the analysis at the issues level. It is more modular and allows more types of analysis, depending on what you wish to investigate.

If you do not understand some parts of the notebook, in particular for any part regarding loading datasets, transformation of tables etc, please refer to the first tutorial notebook 1 : `Introductory_tutorial_(1)` .

Table of Content

Introduction

Setup

1. One-dimensional analysis

1.1 Content items frequency

1.1.1 Example 1: basics - time analysis
1.1.2 Example 2: log scale - newspapers analysis
1.1.3 Example 3: sorting - type analysis
1.1.4 Example 4: filtering - type analysis
1.1.5 Example 5: access rights

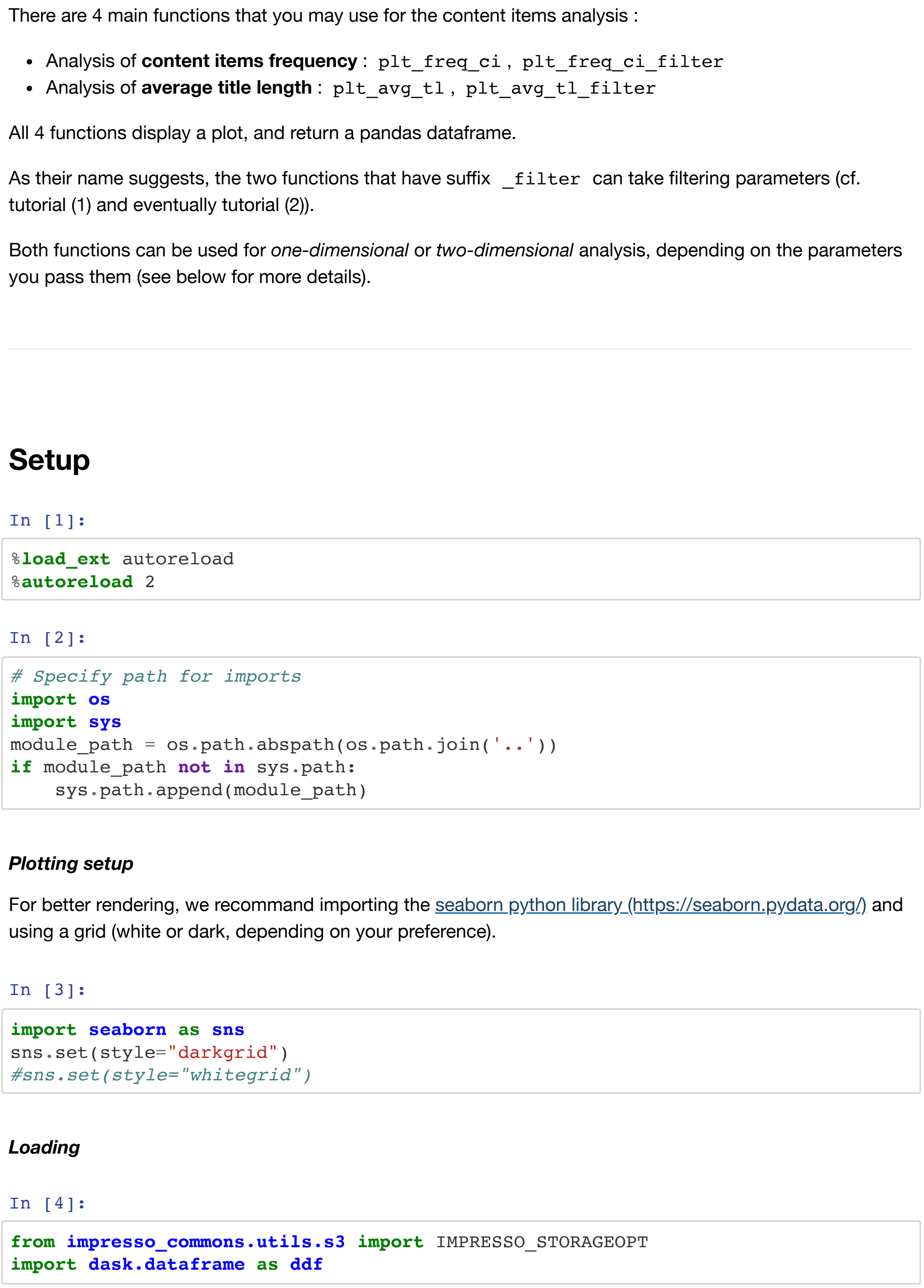
1.2 Average title length

2. Two-Dimensional analysis

2.1 Number of content items per decade
2.2 More specific query

Conclusion

Introduction



There are 4 main functions that you may use for the content items analysis :

- Analysis of **content items frequency** : `plt_freq_ci` , `plt_freq_ci_filter`
- Analysis of **average title length** : `plt_avg_tl` , `plt_avg_tl_filter`

All 4 functions display a plot, and return a pandas dataframe.

As their name suggests, the two functions that have suffix `_filter` can take filtering parameters (cf. tutorial (1) and eventually tutorial (2)).

Both functions can be used for *one-dimensional* or *two-dimensional* analysis, depending on the parameters you pass them (see below for more details).

Setup

In [1]:

```
%load_ext autoreload
%autoreload 2
```

In [2]:

```
# Specify path for imports
import os
import sys
module_path = os.path.abspath(os.path.join('..'))
if module_path not in sys.path:
    sys.path.append(module_path)
```

Plotting setup

For better rendering, we recommand importing the seaborn python library (https://seaborn.pydata.org/) and using a grid (white or dark, depending on your preference).

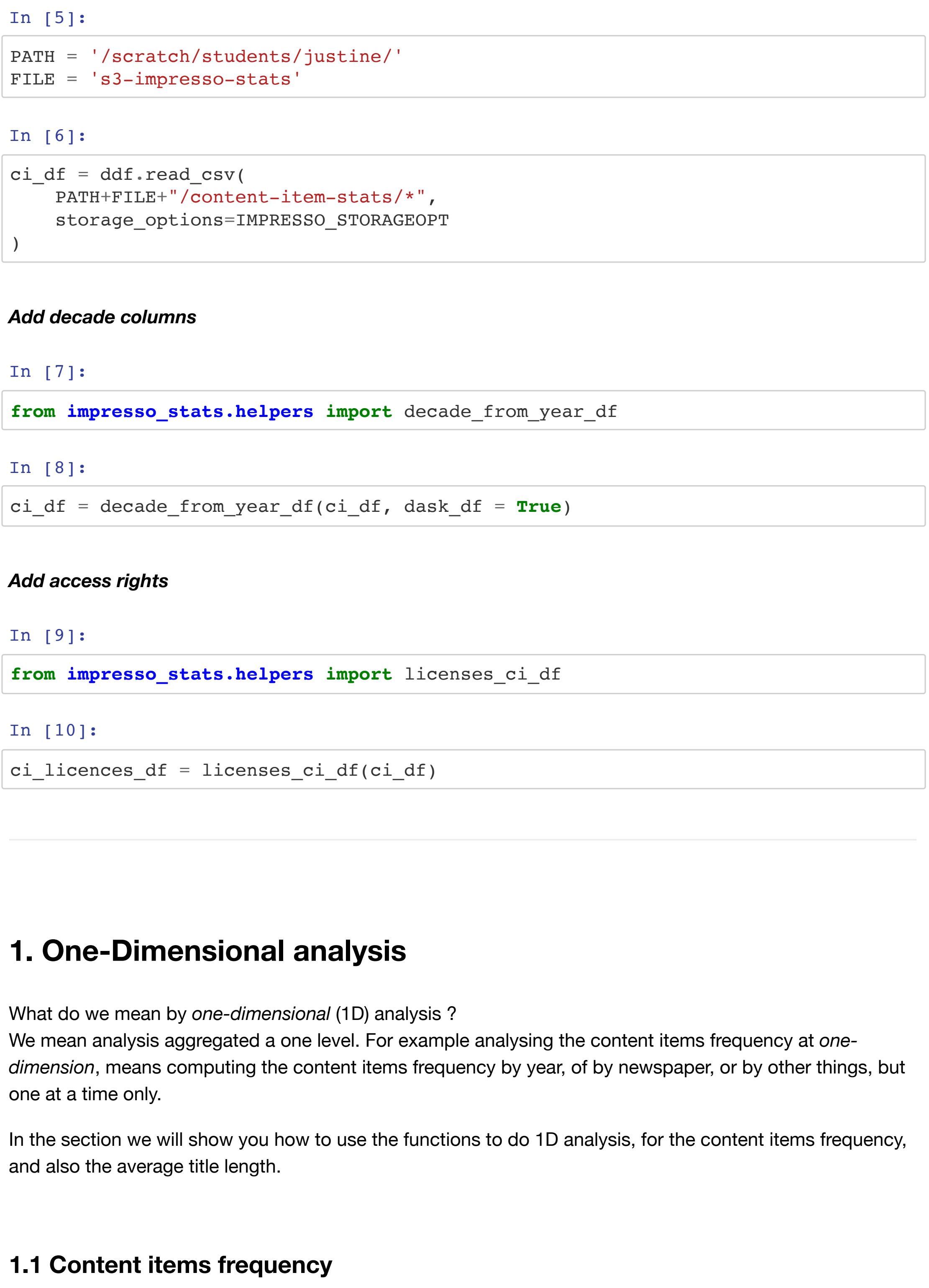
In [3]:

```
import seaborn as sns
sns.set(style="darkgrid")
#sns.set(style="whitegrid")
```

Loading

In [4]:

```
from impresso_commons.utils.s3 import IMPRESSO_STORAGEOPT
import dask.dataframe as ddf
```



In [5]:

```
PATH = '/scratch/students/justine/'
FILE = 's3-impresso-stats'
```

In [6]:

```
ci_df = ddf.read_csv(
    PATH+FILE+"/content-item-stats/*",
    storage_options=IMPRESSO_STORAGEOPT
)
```

Add decade columns

In [7]:

```
from impresso_stats.helpers import decade_from_year_df
```

In [8]:

```
ci_df = decade_from_year_df(ci_df, dask_df = True)
```

Add access rights

In [9]:

```
from impresso_stats.helpers import licenses_ci_df
```

In [10]:

```
ci_licences_df = licenses_ci_df(ci_df)
```

1. One-Dimensional analysis

What do we mean by *one-dimensional* (1D) analysis ?
We mean analysis aggregated a one level. For example analysing the content items frequency at *one-dimension*, means computing the content items frequency by year, of by newspaper, or by other things, but one at a time only.

In the section we will show you how to use the functions to do 1D analysis, for the content items frequency, and also the average title length.

1.1 Content items frequency

Let's start with function `plt_freq_ci` , and its derivative `plt_freq_ci_filter` .

You need to pass at least two parameters to the functions :

- `df` : your dataframe
- `grouping_col` : the features by which you want to aggregate

The `df` parameter is a dask dataframe, and has do to have at least a column `id` and a column with the same name as `grouping_col` .
The `grouping_col` parameter should be a list.

Aggregation levels

Usgin the dataset we have loaded above (and presented in tutorial (1)), here is the list of parameters you can give to the function for `grouping_col` :

- year
- decade
- newspaper
- type
- licenses

In [11]:

```
from impresso_stats.visualization import plt_freq_ci, plt_freq_ci_filter
```

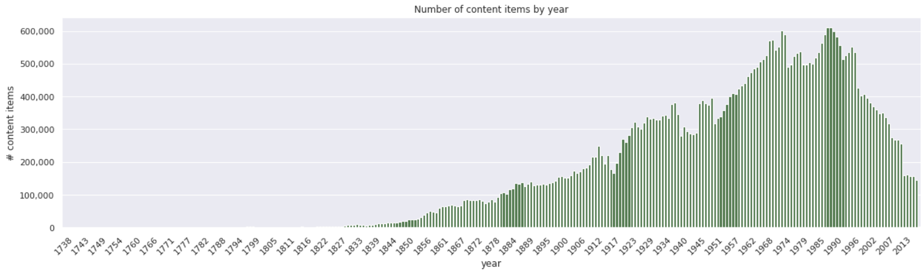
1.1.1 Example 1 : basics (time)

Let's show you a simple use of the function.

The following cell plots the number of content items by year (through the whole dataset).

In [12]:

```
count_year_df = plt_freq_ci(ci_df, ['year'])
```



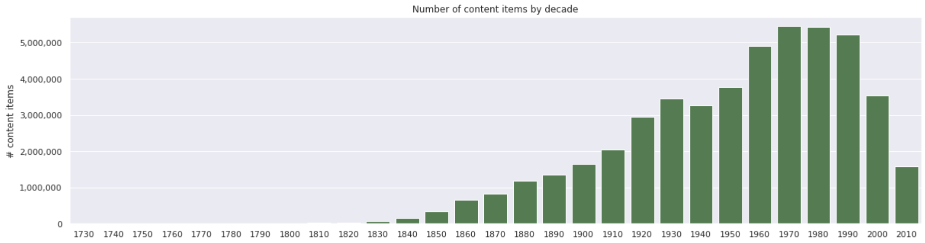
Explanation

There si not much to say here as the plot is quite understandable from itself.
If you do not want to display the x axis title "year", you can set the parameter `hide_xtitle` as True (see below).

The following cell plots the number of content items by decade (through the whole dataset), hiding the x axix title.

In [13]:

```
count_decade_df = plt_freq_ci(ci_df, ['decade'], hide_xtitle=True)
```

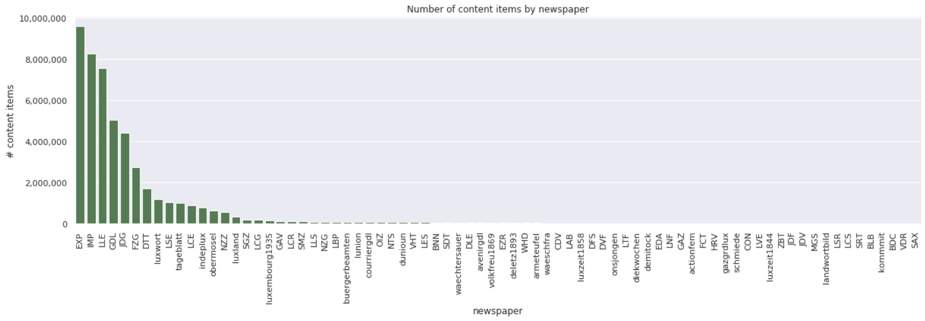


1.1.2 Example 2: log scale (newspapers)

You may want to visualize which newspapers have more / less content items than others.
The following cell plots the number of content items by newspaper.

In [14]:

```
count_np_df = plt_freq_ci(ci_df, ['newspaper'])
```

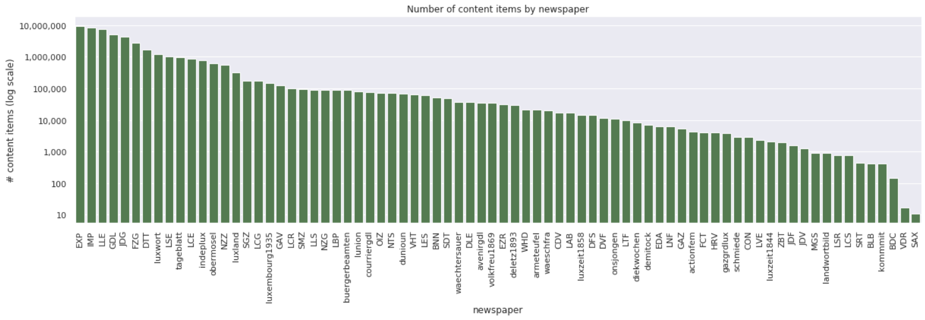


Explanation

From this plot you can see approximately well the 15 first newspapers. The difference in the number of content item is so large with other newspapers that the plot doesn't render well.
You may want to plot in log scale.

In [15]:

```
count_np_log_df = plt_freq_ci(ci_df, ['newspaper'], log_y=True)
```



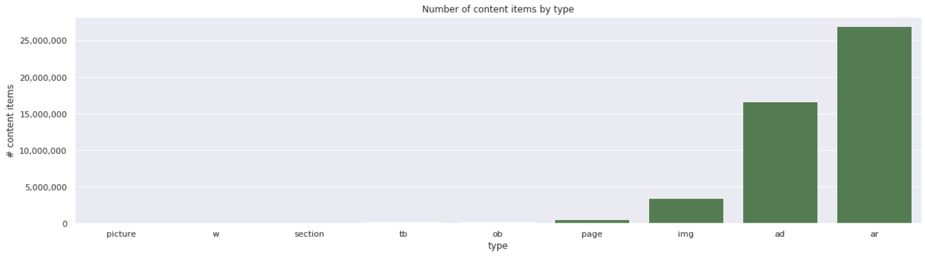
1.1.3 Example 3: sorting (type)

By default, on the plot the x axis label are ordered in descending order (or chronological if we are using time values).

If you prefer plotting in acsending, here is how you should do.

In [16]:

```
count_type_df = plt_freq_ci(ci_df, ['type'], asc=True)
```

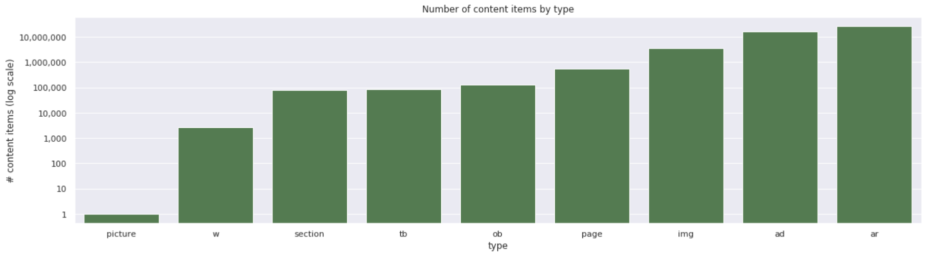


Explanation

You can combine several functionalities. Here as for years above, you may like to plot log scale.

In [17]:

```
count_type_log_df = plt_freq_ci(ci_df, ['type'], log_y=True, asc=True)
```



1.1.4 Example 4: filtering

You can apply filters on the dataframe before computing and plotting the content item frequency. For that purpose you should use function `plt_freq_ci_filter` . If you call this function without passing it any parameter, it simply does the same as `plt_freq_ci` .

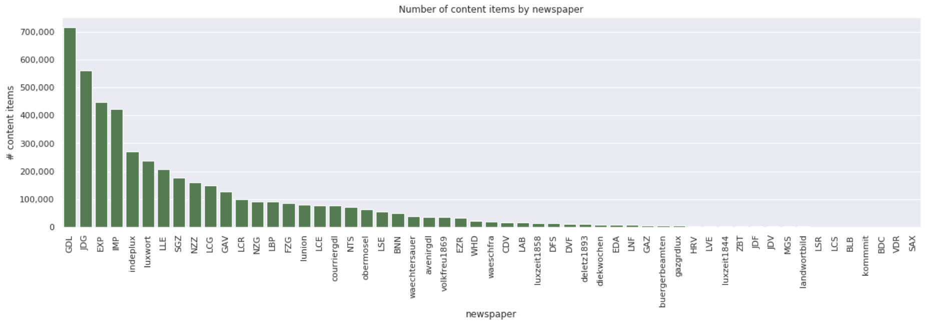
For more information and examples on filter, see Tutorial 2.

Filtering over a period of time

Let's say you would like to know what newspapers have the larger number of content items during the 19th century.

In [18]:

```
count_np_19th_df = plt_freq_ci_filter(ci_df, ['newspaper'], start_date=1800, end
_date=1899)
```



Explanation

You can compare this plot to the one obtained in 1.1.2 and see that is quite different: there are less newspapers and they don't appear in the same order.

GDL and *JDG* are the two newspapers that have the larger number of content items un the 19th century, whereas *EXP* which was the largest overall arrives in third place.

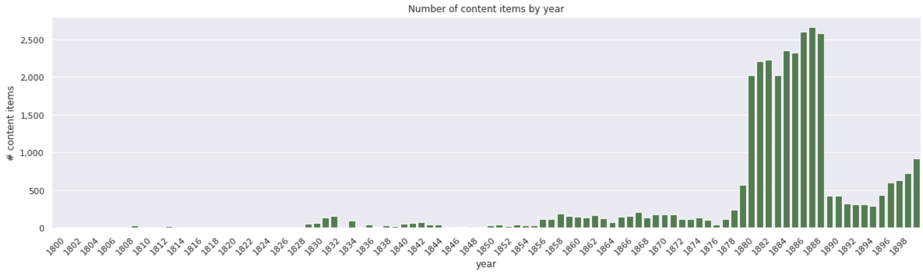
Filtering for content items

You can filter content items with the same parameters as issues (see Tutorial 2), but you can also **filter them by content item type** (article, image, page, weather, ...). You can specify several types by putting them in sequence.

Let's say you would like to know how many **images** were published in swiss journals, during the 19th century.

```
In [19]:
```

```
count_year_img_df = plt_freq_ci_filter(ci_df, ['year'], types=['img'], country=
'CH', start_date=1800, end_date=1899)
```



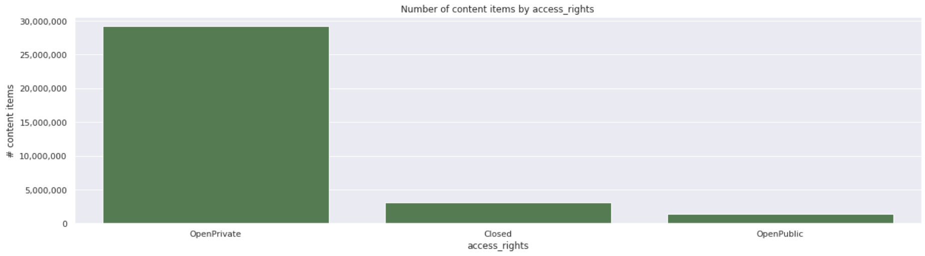
You can also combine several types.

1.1.5 Example 5: access rights

If you want to have statistics on the number of content items per access right, you should be careful to use the dataframe `ci_licences_df` , having a column `access_rights` (and not `ci_df`).
Note: Do not forget that this dataframe has less rows than the original one (cf. Tutorial 1).

In [20]:

```
count_licenses_df = plt_freq_ci(ci_licences_df, ['access_rights'])
```



1.2 Average title length

The average title length can be computed and plotted using function `plt_avg_tl` and its derivative `plt_avg_tl_filter` .

This function works exactly the same way as `plt_freq_ci` presented above, so we will only show you one simple example here, and let you explore by yourself.

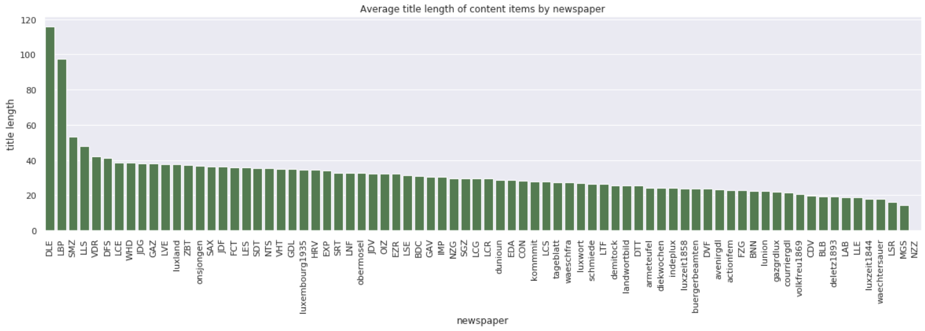
The `df` parameter is a dask dataframe, and has to have at least a column `title_length` and a column with the same name as `grouping_col` .

In [21]:

```
from impresso_stats.visualization import plt_avg_tl, plt_avg_tl_filter
```

In [22]:

```
tl_per_np = plt_avg_tl(ci_df, ['newspaper'])
```



2. Two-Dimensional analysis

In comparision to *one-dimensional* analysis, by *two-dimensional* analysis we mean analysis aggregated a two levels. For example analysing the content items frequency at *two-dimensions*, means computing the content items frequency by year per newspaper, or by type per access_rights, or by other things, but grouping by two at a time.

When passing a list of length 2 as parameter `grouping_col` to the functions, it will produce a two-dimensional analysis. **The order of the parameters in the list matters !**
The first one will correspond to the x axis, and the second one to the category (hue on the plot).

Note: *You should not use a time feature (decade of year) as second element of the list. It should always be the label used in x axis and not as a category.*

In the section we will show you how to use the functions to do 2D analysis, for the content items frequency. You can also use it for the average title length, but exploration if left to you !

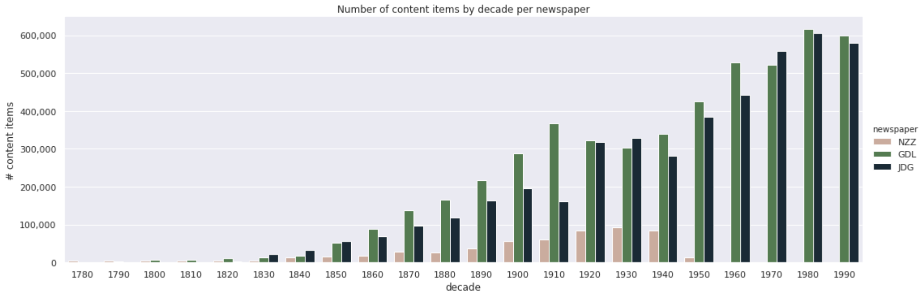
Note: *When plotting in 2D,* ***filtering becomes necessary*** *in order to avoid having too much data to plot on a single graph (the limit is set to a total of 350 bars for now - you can change it in the file* `visualization.py` *).*
In particular the second element of the list, used as category, should not have too many values possible (not more than 5/6), and should have less values than the element chosen as first one.
For example, if you wish to plot the number of content items per newspaper by year, you will need to select some newspapers (we recommend 2 or 3, until 5 maximum), and eventually some year bounds.

2.1 Number of content items per decade

Let's say you are interested in comparing the number of content items for three journals : *Neue Zürcher Zeitung (NZZ)*, *Journal de Genève (JDG)* and *Gazette de Lausanne (GDL)*. You could first start looking at the decade to avoid having to many bars on the plot.

In [23]:

```
count_decade_np_df = plt_freq_ci_filter(ci_df, ['decade', 'newspaper'], np_ids=[
'NZZ', 'JDG', 'GDL'])
```



Explanation (optional: concerns issues - see tutorial 2)

NZZ seems to have less content items than JDG and GDL. One interesting fact is that the number of content items increases through time for JDG and GDL, whereas for NZZ it increases until 1930 and then starts decreasing.

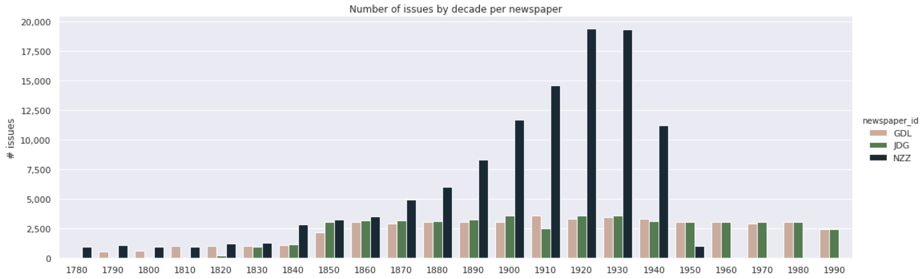
It may be interesting to compare this tendancy to the issues frequency !

In [24]:

```
from impresso_stats.visualization import plt_freq_issues_time
```

In [25]:

```
issues_comparison = plt_freq_issues_time('decade', np_ids=['NZZ', 'JDG', 'GDL'])
```



Explanation

Two interesting facts:

- **The tendancy for NZZ :** for issues as well as content items, the tendancy for NZZ is the same. It increases until 1930 and then starts decreasing until having no publication at all from the 60s. It is probable that from the 60s the journal stopped publishing, which is not the case for JDG and GDL and explains the difference.
- **Comparison of newspapers at the level of issues VS. at the level of content items :** on the second plot, we can see that at some point the number of issues for NZZ is way larger than for JDG and GDL (at least 4 times higher). Conversly, on the first plot, the number of content items was always way less for NZZ (always at least 3 times less) than for the two other newspapers. It suggests than the number of content items in issues of JDG and GDL was way larger than in issues of NZZ. Does it come from the dataset or from a particularity of the newspapers ? This would be interesting to analyse.

This encourages providing a function for having the average number of content items per issues through time, or per newspapers. This function is not provided yet in the library, but would be a nice extension to have.

2.2 More specific query

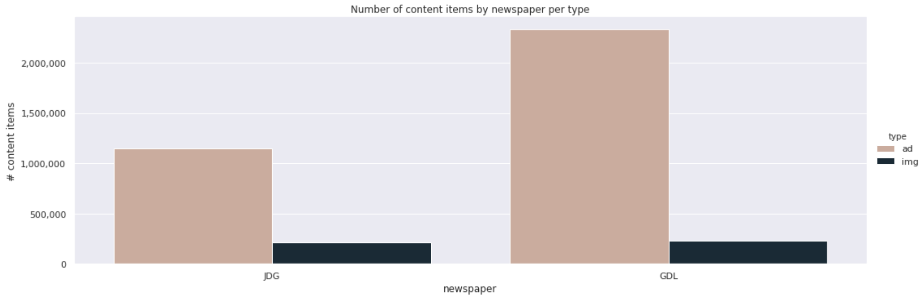
Given the great modularity of the function, given by filtering and changing the dimensions on which to plot, you can ask more specific queries.

The example below shows how to compare the number of ads to the number of images, between two newspapers : JDG and GDL, all years combined.

You could also indicate specific year bounds, in order to have a more precise result.

In [26]:

```
count_type_np_df = plt_freq_ci_filter(ci_df, ['newspaper', 'type'], np_ids=['JD
G', 'GDL'], types=['img', 'ad'])
```



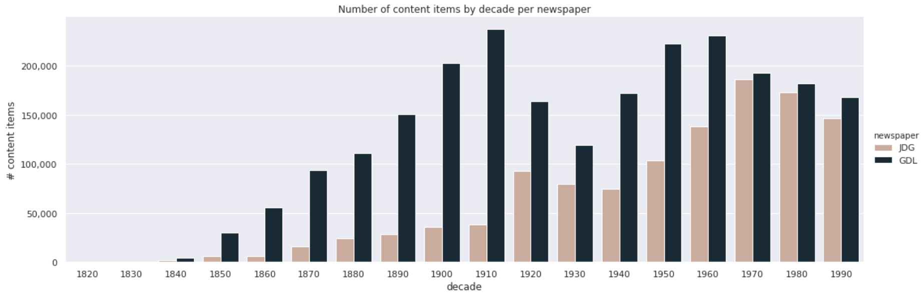
Explanation

- The number of images for the two newspapers seems to be the same
- The number of ads in GDL is twice as much as for JDG. It could be interesting to see if it is a general tendency, or if it comes from a precise periode where GDL published a lot more ads than JDG.
- For both newspapers, the number of ads is way bigger that the number of images.

Let's see the evolution in the number of ads through time for the two newspapers.

In [27]:

```
count_decade_np_df = plt_freq_ci_filter(ci_df, ['decade', 'newspaper'], np_ids=[
'JDG', 'GDL'], types=['ad'])
```



Explanation

This is interesting : we see that in the last three decades JDG and GDL approximately published the same number of ads. But from 1840 until 1910, the number of ads in GDL increased a lot: from 5'000 in 1840 (you can generate a zoomed plot to see precisely) to approximately 240'000 in 1910 (almost 50 times more !), whereas the number of ads in JDG was stable (increased a little). This is why the overall number of ads for GDL is twice as big as for JDG, and it comes mainly from this period, not from a general tendency.

Conclusion

This is the end of the tutorials. I hope it has given you a good idea on how to use the functions of the impresso_stats library and how they could be useful for you.

```
In [ ]:
```