Department of Information Engineering and Computer Science

Bachelor's degree in
Computer Science

Final Dissertation

INTERACTIVE DASHBOARDS FOR MEASURING WIKIPEDIA COMMUNITY HEALTH

Technologies, contextualization and implementation of the Community Health Metrics dashboards

Supervisor	Co-Supervisor	Student
Alberto Montresor	Cristian Consonni	Paolo Aliprandi
	Marc Miquel-Ribé	
	David Laniado	

Academic year 2021/2022

Special thanks

I would like to thank the people that constantly helped and supported me during this incredible journey. They are the people who make my life better and worthy to be enjoyed and who helped me became the man I am. And most importantly, they inspire me into becoming the man I will be tomorrow. Hence I want to thank my family: my parents, my sisters and my grandparents. They are always there for me and gave me everything I have. I will never be able to thank you enough.
I want to thank my amazing girlfriend. She made me a happier person than I could ever hope to be. Most importantly, she inspires me constantly into wanting to improve myself. For me, for her, for us.
I also want to thank professor Montresor, for being a great professor and a mentor, who also gave me the opportunity of participating in this project.
I think that being grateful, is the most beautiful feeling ever.

Contents

Abstract

This dissertation describes in detail the activity performed during my two-month internship period at the Big Data & Data Science Department of Eurecat – Centro Tecnológico de Catalunya, which was supervised by Cristian Consonni, Marc Miquel-Ribé and David Laniado.
The Community Health Metrics project aims at measuring and understanding the current Wikipedia health status, in order to bring awareness among the Wikipedia editors of the different communities. Hence the realization of visual and interactive dashboards: since Wikipedia relies on its active editors, these dashboards are meant to be seen especially by them. It is important to render Wikipedia health status data in a way that is easy and straight-forward, in order to make them understandable even by non-technical people.
The project wants to share knowledge in order to have a positive impact on Wikipedia health status.
I have contributed at the project by:

- handling and analyzing over 20 GB of data from Wikipedia database
- experimenting and discovering different kind of visualization and graphs for each health metric
- mainly, coding and creating Python scripts to visualize and plot Wikipedia Vital Signs' data
- studying different approaches for the code structure
- adapting and deploying the scripts from a local architecture to a remote server architecture
- studying how to improve the frontend user experience
- defining and generating automated description to enhance the dashboards readability

The database I analyzed was given to me by the EURECAT researchers: it is generated every month by observing the interactions of the Wikipedia editors in the different communities. Then, this data are grouped by the language of the communities, vital sign analyzed and other factors that will be later explained.
Given this database, I created different Python scripts to generate the visual dashboards that graphically shows the data took from the database and the automatic captions, called highlights.
I implemented the visualization of the Wikipedia vital signs, six metric defined to study the health status of Wikipedia.
Each dashboard presented its unique challenge. From pre-elaborating the data in order to feed them correctly to the graph, from engineering the correct query statement to retrieve the more comprehensible data even to deciding which visualization to use to display a certain data class.
In this dissertation we are going to discuss in details the implementation of each visualization and how it contributes to the dashboards, briefly mentioning the technologies employed and a minimal contextualization of the meaning of this metric. In the end, we will analyze the results emerged from these dashboards.

1 Introduction

Wikipedia is the largest encyclopedia available on the Internet. We could argue that it is the largest collection of knowledge ever created in the entire human history. No other library or oral knowledge could ever stand the comparison in size. It is also important to note that Wikipedia is available everywhere, from anyone in almost any language. Wikipedia, in fact, makes the knowledge accessible to anyone, in a democratic way. This is a remarkable way of spreading wisdom.
For all the above reasons is fair to say that Wikipedia has a tremendous importance in our modern society. Such an important source of knowledge requires an adequate maintenance and care, in order to ensure its safety and its prosperity.

1.1 Context and motivation

Everybody deserves the possibility to learn, to know, to explore the human knowable. But in order to provide this amazing opportunity to whoever asks for it, there must be someone to take care of it. This is the Wikipedia trade-off: if something is open and reachable by everyone, it must be provided and taken care of by many people. In order to ensure the democracy of the project, everyone must join it.
Without its fundamental active users, its contributors, Wikipedia ceases to be what it is now.
There can't be readers without writers.
Driven by these very motivations, the Wikimedia Community Health Metrics projects aims at taking care of the online encyclopedia, collect information about it and spread them to share knowledge among Wikipedians.

1.2 Project description

The very goal of the Wikimedia Community Health project is to measure and understand the Wikipedia community health status [6]. The CHM project wants to monitor it, because if we can understand something, we can interact with it; we can change it; and hopefully, we can improve it. In fact, the project does not limit itself to an ephemeral comprehension of the wellness of Wikipedia. It wants to give recommendation in order to ensure its prosperity and spread the information about it. These tips are for everybody who interacts with Wikipedia but are especially meant for its active users.
In order to make these data understandable by everyone, even by non-technical people, visual dashboards were created. These graphics make the information about Wikipedia health easy to read and their comprehension straight forward.
But before measuring Wikipedia health, we must define it. Like doctors can measure the body vital signs (heartbeat, cerebral activity, oxygen levels,...) so can we: the project defines six metrics that can be analyzed and studied. These are called Wikipedia Vital Signs.

1.3 Vital Signs

Vital Signs are well defined concepts that cast the light on different aspects of the Wikipedia community.
Thanks to the contribute of each Vital Sign, we can determine the overall health status of Wikipedia. More precisely, we will measure the vital signs of each language community. Since Wikipedia is the sum of all its parts, we can understand its general wellness measuring the health status of each small parts.
The project envisions six vital signs that will be monitored:

- Retention
- Stability
- Balance
- Special functions
- Admin flags
- Global community participation

These metrics, defined by the Eurecat project team, provides a powerful description of the Wikipedia communities health status.
The first three are about the Wikipedia active users: the editors. While the last three are about more technical aspects.
We are given multiple indicators to have a better understanding of different important aspects of the Wikipedia community.

1.4 Visualizations

The Vital Signs indicators are accurately and individually shown in the interactive dashboards.
Each of the Vital Sign has (at least) one peculiar charts that shows how this value has changed over time. This is what we are really interested in: to visualize and understand how the metrics evolved throughout the Wikipedia life.
Depending on the metric, our attention will be directed towards the rise or the fall of a certain values. Is it good that this particular metric increased? Should it have decreased? These questions find their answers in the highlights. We can find them right under the graphs, they help the user to better understand the metric visualized.
Both the graphs and the highlights are dynamic: the dashboards presents some interactive components that allow the user to change different aspects of the visualizations and of the highlights:

- **Language**: Since we want to study the diverse Wikipedia language communities, the dashboards allow the user to select among every language available on Wikipedia. The user then obtain one chart for each language selected, to let comparison between different language happen.
- **Time**: We want to understand the evolution in time of the Wikipedia health status. In fact, we can change the time aggregation of the data from yearly to monthly, in order to have a general or accurate view. We can also select the time period analyzed through an intuitive window that can slide left or right, allowing to scroll through the years (or month).
- **Data**: For each Vital signs there are peculiar parameter that can be selected, in order to let the user obtains the right chart with the right type of data desired. The ultimate goal is to spread knowledge and understanding.
- **Value**: Finally, we can decide if we want to see absolute values or their percentage values, to fulfill every research task. Absolute values are more useful if we want to conduct a quantitative analysis on singular language communities.
  Percentage values should be preferred in case of a qualitative analysis to compare different language communities.

Every graph is also interactive as well: when passing the cursor over the figures a little text box appears, with useful information about the specific coordinates selected. It is also possible to slide a window left or right, to analyze a specific time period and let the user decide how long it should be.
Everything is immediate and intuitive to provide the best user experience while letting the user study and understand the Wikipedia health metrics.

In the next chapters, we will get a closer look at these visual dashboards: how they work and which conclusions we can infer from them.

2 Technologies and Architecture

This chapter describes the Python modules used to build and run the Dashboards, and how they are structured.
The vital signs dashboard have been realized as a dynamic website that allow user interaction.
Let us now describe the role of each Python module.

2.1 Plotly

Plotly is a Python graphing library that provides a wide variety of charts for visualising data. Since we get to plot different data, we need different charts to provide the best suited graph for each data we want to visualize. Each of them is interactive.
We remind that the goal of the project is to make the user understands what is going on with Wikipedia health metrics.
The power of Plotly is that it allows the graphs to change which data column to show, so the user has the possibility to see exactly the data he or she is interested in: from the time period to the value type. To create the graphs we use two different sub-modules of Plotly:

- **Graph objects [1]:** The figures created, manipulated and rendered by this plotly Python library are represented by tree-like data structures which are automatically serialized to JSON. These trees are composed of named nodes called "attributes". The `plotly.graph` objects module, imported as `go`, contains an automatically-generated hierarchy of Python classes which represent non-leaf nodes in this figure schema. The term "graph objects" refers to instances of these classes.
- **Plotly express [2]:** The plotly express module, imported as `px`, contains functions that can create entire figures at once. It is a built-in part of the plotly library, and is the starting point for creating most common figures. Every Plotly Express function uses graph objects internally and returns a plotly `go.Figure` instance. The module `px` was preferred due to its simplicity and more expressive syntax.

In the visual dashboards, we find four different type of graphs:

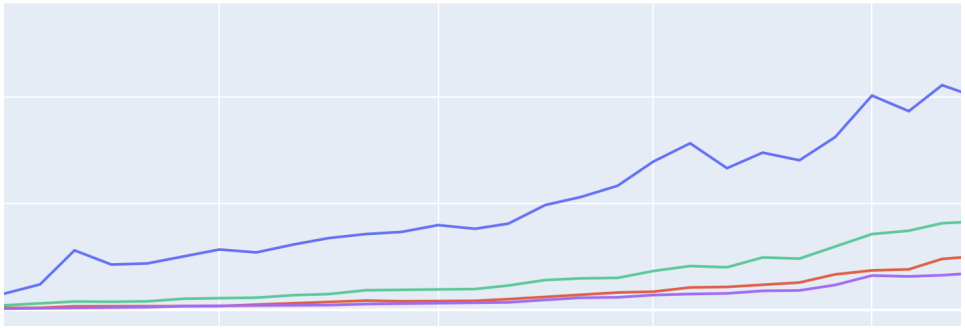

Figure 2.1: Line chart

In Figure 2.1 we see the example of a line chart. We use them when we want to move the focus on the evolution of our data in time, especially when comparing different classes of data.
In this example we can see the evolution in time of the number of active editors of different language communities. In the next chapter we will focus more on this concept.

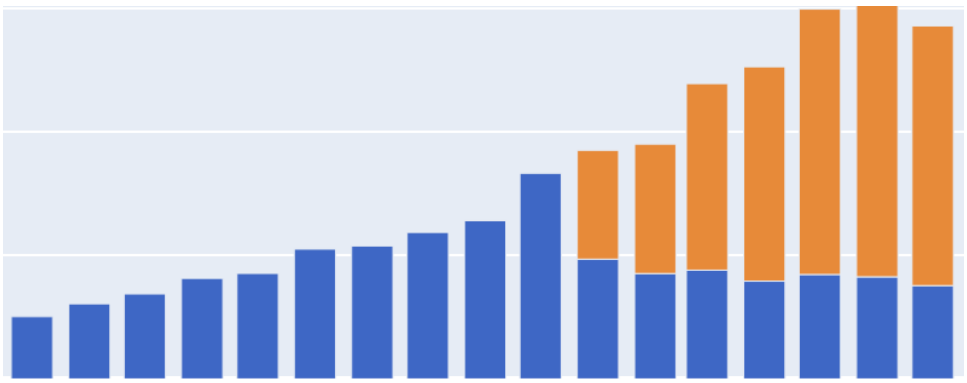

Figure 2.2: Stacked bar chart

The most frequent graph is the stacked bar chart seen in Figure 2.2. It is particularly suited to analyze the data distribution and focus on the quantity of a certain value with respect to other values. Different classes of data sharing some common attributes get stacked to render the idea of their quantitative difference. In this example we see the distribution of two different generations of editors through the years.

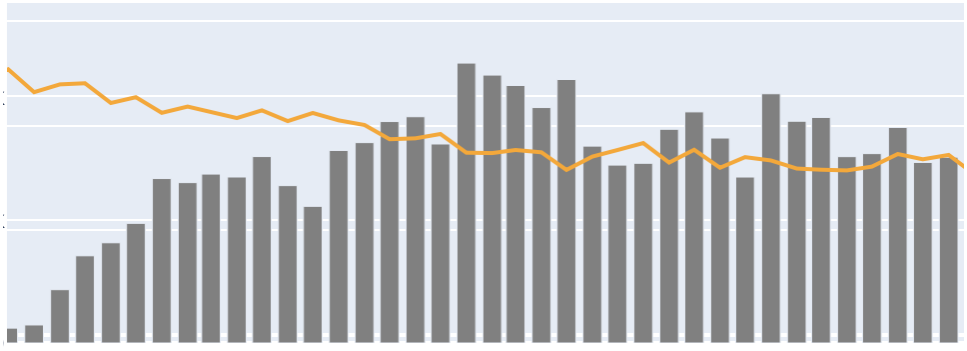

Figure 2.3: Dual-axis graph: line chart and bar chart

The dual-axis graph from Figure 2.3 helps conveying multiple information at the same time. In the dashboards, it is used in its line chart plus bar chart configuration, thus, it helps the user to analyze the distribution of a certain data class and also compare it to the evolution in time of another class of data. In this example, we can see the distribution of registered editors with respect to the evolution in time of the number of editors that remain active after a certain temporal threshold. Again, this will be explained later.

Figure 2.4: Treemap chart

Finally, the Treemap chart. It is a special graph used to represents the quantitative distribution of certain data classes with respect to all the other data classes considered in the current analysis.
It is preferred to a pie chart visualization because the comparison between rectangular shapes is clearer than the contrast between circular sectors. Each colorful square can be expanded, to reveal other information about the current object, in this case the current language.[7]
In the dashboards it is used to visualize the Meta-Wiki participation of the all the Wikipedia language communities.

2.2 Dash

Dash is the web framework on which the plotly apps runs. It is tailored to create visual dashboards in an extremely simple way. [3]
The entire dashboards layout is built with Dash. There is no HTML, no CSS nor JavaScript.
The layout is, however, built with components of the HTML Python module that allows to place HTML classic components, like `div` container or radio buttons, and handling them with a JSON-like syntax. These components directly commmunicate with the plotly graphs thanks to the Dash functions, or callbacks, that feed the information coming from them to the plotly charts, making them interactive. Not only the graphs are interactive, the whole page is: by selecting the desired vital sign, the layout will automatically display the respective dashboard.

Listing 2.1: Structuring the Dash layout

```
import dash
from dash import Dash, dcc, html, dbc

html.Div(
        html.P('Select Yearly or Monthly Time aggregation'),

        html.Br(),

        html.Div(
            dcc.Dropdown(
                id='language',
                multi=True,
                value='English (en)',
                style={'width':'490px'}
            ), style={'display':'inline-block','width':'500px'}
        )
    )
```

From the code snippet above, we can clearly see the absence of any HTML code. The whole layout is implemented using Python scripts like this.
Dash has many libraries as well, such as dash core components (`dcc`) and dash bootstrap components (`dbc`), that provides HTML components with pre-built CSS style, in order to help us build an homogeneous and elegant web page. Since we want to focus on the dashboards, the layout is designed to be minimal.
From the example above, in fact, we can see a drop down in a HTML `Div`. This element comes from the `dbc` library and provides a simple custom drop down menus:

- **id**: since it is an HTML element, we can set its identifier.
- **multi**: we can set if it must contains more than one choice.
- **value**: it allows us to set the default value, in this case the English language.
- **style**: we can add other information about the CSS style.

All these graphic components hides more than what we see: they are the interface through which we can interact with the Plotly charts. We can bind these components with functions, called Dash callbacks, that communicate directly with the Graph objects.
We also said that Dash is responsible for the whole business logic of the web applications. Let us take a look at the callbacks and how they are defined.

Listing 2.2: Dash callback for URL update

```
import dash
from urllib.parse import urlparse, parse_qsl, urlencode

@app.callback(Output('page-content', 'children'),
              inputs=[Input('url', 'href')])
def page_load(href):
    if not href:
        return []
    state = parse_state(href)
    return layout1
```

From the code snippet above we get to see the definition of a Dash callback. It works just like a function: we define the input and the output structure, and we write the code to execute. The main thing to notice is that we bind this function to the Dash application with the `@app.callback` string. In this example we are looking at the function that allows the sharing of the visualization with other users: the parameters that are read from the graph callbacks are embedded in the URL, in order to let the page update its HTML components' value with the values read from the URL. In this way, the callbacks will always read the correct parameters. We will read the default ones only if we are visiting the dashboard as a new visitor.
In the next chapter, we will have a closer look at this mechanism.

2.3 Pandas

Pandas is the mediator between front-end and back-end code. [4]
It is a Python module that is in touch both with Plotly and with the SQL database, managing the data flow between these two components.
It is mainly used to encapsulate the Result sets coming from the SQL queries asked to the Eurecat Database. Since these data are still too raw, we need to process them in a way that they can work properly with plotly: we use Pandas objects called Data frames.
The Pandas data frames are extremely useful since they allow us to grab the raw data and envelope them into these objects that are much more malleable than their raw form: they have a lot of methods used to handle them. Let us take a look at how data frames are created:

Listing 2.3: Creating a Pandas data frame

```
import pandas as pd
import sqlite3

conn = sqlite3.connect(database)

SQL_query = '''select *
            from vital_signs_metrics
            where topic = 'active_editors'
                and year_year_month = '''+time_type+'''
                and m1_calculation = 'threshold'
                and m1_value = '''+user_type+'''
                and langcode IN (%s) '''%languages

df = pd.read_sql_query(SQL_query, conn)
print(df[:100])
df['percentage']=((df['m2_count']/df['m1_count'])*100).round(2)
```

First of all, we specify the database with the Sqlite3 connection object `conn`. Then, we create a parameterized SQL query.
The parameters of this query are:

- **time type**: specifies the time aggregation of the final chart, that can be yearly or monthly.
- **user type**: specifies if we talk about active editors or very active editors.

- **languages**: determines the desired languages on which we want to do our research.

The query is then passed to the database and, at the same time, the result set is put into the data frame `df`. We then print it to ensure that it is the correct desired data set.
Finally, we create a new data frame column from already existing ones. In the example above, we create a percentage column.
If we wanted to do this via SQL, we would have to change the query and thus, we would have put two different things at the same time, violating the Separation of Concern principle[8].
Instead, with this method we can launch a high level query and then refining it with pandas. This is much more convenient since we can separate the creation of the raw result set and then the refining phase, making it easier to eventually debug.
Later one we will have an in-depth view on how pandas and plotly work together.

2.4 Architecture

Now that the Python module have been described, this section will illustrate how they are structured.

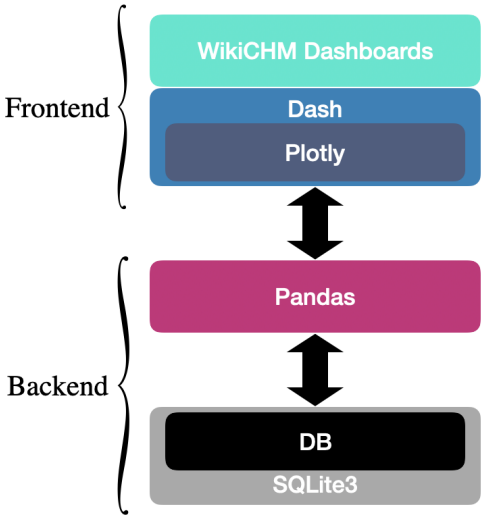

Figure 2.5: Dashboards architecture

The Figure 2.5 is a high-level definition of the architecture of the visual dashboards.
Every level of this structure has been made with ad-hoc Python modules.
Let us describe them with a top-down approach:

- **Plotly**: every dashboard's graph is made with the Python library for data visualization. It is contained into Dash web framework to allow the interaction between the user and the HTML components. It is what the user actually sees and interacts with.

- **Dash**: the charts run in a web container, realized with a Python module that allows the creation of web applications. Dash automatically manages every web-related aspects such as the building of the entire layout that can totally change at run-time, creating the website to which the user can access to.
  Dash is also responsible for the functions that makes the dashboard interactive and for the dynamic layout.

- **Pandas**: the data plotted are handled with the widely-used data manipulation Python library, handling and managing the exchange of data between the underlying DBMS and the above layers. It is the mediator between the front-end dashboards and the back-end data.

- **SQLite3**: this low-level module provides the connection with the relational database, created by the Eurecat researchers, and mainly acts as its DBMS. It is the responsible for the raw extraction of the data.

Every layer of the structure dialogues with its neighbour. The data flows bottom up: from the database to the Wiki Community Health Metrics dashboards.
Each component captures the output of the underlying layer, elaborates it, and feed its own result to the overlying one. All these Python modules have been chosen for their mutual compatibility, in particular Python, Dash and Pandas to ensure a smooth and robust workflow.

3 Implementation

The Vital Signs Dashboard shows us different graphs with different data visualization about specific aspects of the health status of Wikipedia. Despite their overall similarity, each of them is built differently from the others due to its peculiarity.
In this chapter we are going to have an in-depth look at how the code is structured and at the actual implementation of each different dashboard.

3.1 Business logic

The main idea behind every charts is the following: we read the input from the HTML components, we process it in some way to make it compliant with the SQL queries, then we capture the outputs and feed them to the plotly chart.
This whole interaction between the Dash layout and the Plotly charts is done with Dash asynchronous functions: the callbacks. Each graph has its callback that is linked to some of the HTML components and each time the user interacts with one of them, the function is called.
The same is done for the highlights since they also need information about the current visualization in order to adapt the captions.
There is also a higher level of dynamics, regarding the whole layout: in the previous chapter we have looked at an example of a Dash function that changes the layout according to the URL parameter. This whole mechanism can be described as follows:

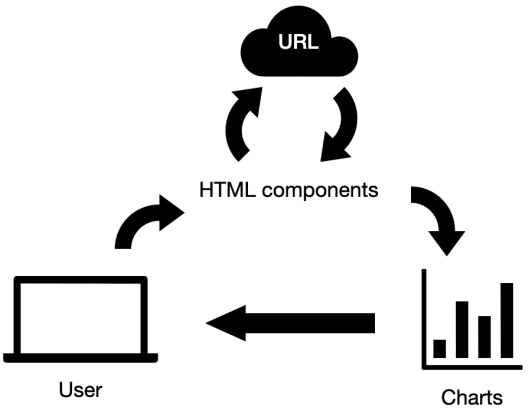

Figure 3.1: Dash callback loop

The user interacts with the HTML components, setting some parameters. When they change value, a function is activated that embed each parameter into the URL. As soon as the URL changes, a function parses it, retrieving the parameter and applying the change to the same HTML components that activated the loop. The main goal of this is that by changing the vital sign from the drop down menu, the layout is changed accordingly. Also, the respective vital sign callback is called, kick-starting the creation of the dashboard that will return the graph[1].
In this way, our visualization coordinates are preserved: if we select some parameters for a certain

[1]For some vital signs there is more than one graph, even with one language only

vital sign, like Italian and French on Very active editors, these choices will be maintained when we change vital sign, allowing the comparison of the same communities under different health metrics.
There is also another effect: by sharing our link of the current page with someone else, they will see exactly our same visualization. We remind that the goal of the project is, in fact, sharing knowledge.
We will now see each callback of the different Vital signs.

3.2 Editors

The first dashboard is about editors: the users that are actively involved in the Wikipedia project by writing and editing the Wikipedia pages.
Based on the number of their monthly contributions we can split them into two categories: active editors and very active editors.
Active editors are the users that edits at least five times per month, while Very active editors are the ones who edit at least 100 times per month.
From the graph in Figure 3.2 we see the number of active and very active editors in different language communities. Since we are interested into seeing how these numbers changed over time, a line chart is provided.
We are also interested into comparing these values across different languages. In fact, the graph allows the visualization of different contemporary traces representing the evolution of the number of editors in different languages (with different colors).

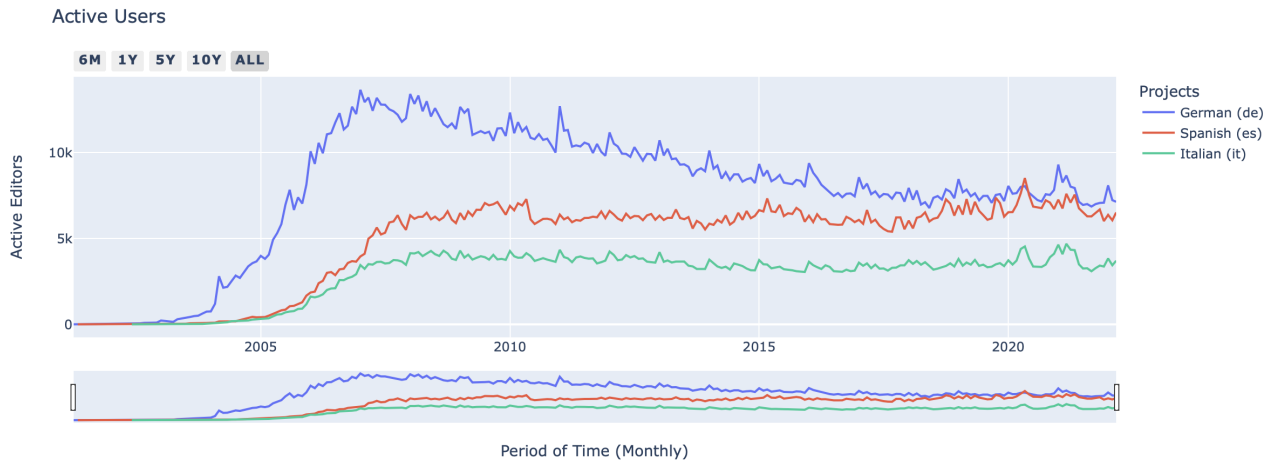

Figure 3.2: Active editors dashboard

3.2.1 Editors implementation

The Active editors dashboard is perhaps the simplest one.
The graph directly reads which information to display from the interactive components "Language", "Editors" and "Time aggregation".
First, the function reads the language into a list. Then, the whole languages names like "Italian (it), German (de)" are converted through a dictionary, called `language_names` into their respective language codes, to make it compliant with the SQL queries. Furthermore, an additional string is created called `params`. This string is very important because in the final query we will search the languages contained in this parameter. We have obtained our first parameter.
We then proceed to capture the type of editors the user wants to visualize. This can be done by simply filtering the input number called `user_type`. If it is equal to five, then they are active editors, otherwise they are very active editors. In reality, the input number will be passed to the query untouched, since the HTML component has a numeric values behind the literal label. This is our second parameter.
Finally, we want to know if to display a monthly charts or a yearly one, and the process is identical to the one described for the numeric value of the editors.
Now, after having collected these three parameters, we can build a parameterized query by having a

fixed statement and inserting in the string the variables containing the parameters. We then run the query to the database using pandas, yielding a data frame containing the desired information. This object is now passed to the actual plotly charts at creation time.

Listing 3.1: Creating a line chart

```
fig = px.line(
            dataframe=df,
            x='year_month',
            y='m1_count',
            color=df['language_name'],
            height=500,
            width=1200,
            title=incipit+' Users',
            labels={
                    "m1_count": incipit+" Editors ",
                    "year_month": time_text,
                    "language_name": "Projects",
                },
        )
```

From the example above we can see the code responsible for the creation of a line chart. It is done using plotly express, the sub-module of plotly.
We can also see how we specify the data source, the data frame, and which data column we want to put on the x-axis and on the y-axis.
We can also see some other keywords that are useful to improve the visualization of the graph: we can set a title to enhance its readability, we can specify its size and we can also create a map that provides more readable labels for the data columns.
Finally the dashboard is ready to be deployed.

3.3 Retention

The first vital sign is Retention and it measures the percentage of new editors that survives 60 days[2] after the first edit. Survivors are meant as those users which keep editing after the time threshold. We are interested in the capability of a certain community of keeping engaged its editors, in order to understand if and why editors are leaving Wikipedia.
In this dashboard we can see the retention rate of one or more language communities, along the timely distribution of the registered users. From the Figure 3.3, we can see that the graph is actually a dual-axis chart: on the left y-axis, as bars, we have the number of registered editors, while on the right y-axis we have the retention rate displayed as an orange line.

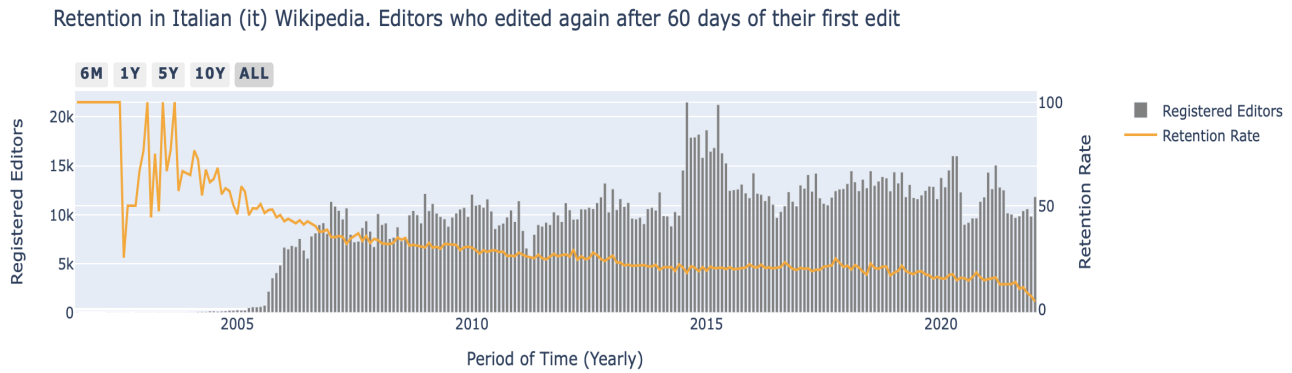

Figure 3.3: Retention dashboard

[2]Retention period is not fixed but can be chosen from different time periods: 24 hours, 30, 60, 365 and 730 days

3.3.1 Retention implementation

The retention dashboard is quite different from the previous dashboard and from the following ones since it is not a normal chart, but a dual chart. Due to the high-level nature of plotly express, this module was not useful to code a dual-axis chart. Thus, the graph objects module was used; in particular the `make_submodule()` method.

Listing 3.2: Creating a dual axis: line chart and bar chart

```
fig = make_subplots(
    specs=[[{"secondary_y": True}]])

fig.add_bar(
    x=df1['year_month'],
    y=df1['m1_count'],
    name="Registered Editors",
    marker_color='gray')

fig.add_trace(
    go.Scatter(
        x=df2['year_month'],
        y=df2['retention'],
        name="Retention Rate",
        hovertemplate='%{y:.2f}%',
        marker_color='orange'),
        secondary_y=True)
```

Here we can see the code responsible for the realization of the Retention dual axis.
First we create a figure with a secondary vertical axis. Then, we set which type of charts will populate this figure: we add a bar chart, for registered editors, and a line chart, for the retention rate, with `go`. To enhance the readability of the latter, we use the keyword `hovertemplate` to show the retention values when the cursor passes over the line.
We remind that our goal is to visualize Wikipedia communities health values and to compare them. In fact, when we select multiple languages in the drop down menu, one Retention graph for each language will appear. This feature is similar to the one of the following graph, but from the back-end prospective it is fundamentally different.
We will later discuss the use of a fundamental keyword, `facet_row`, but for our current understanding we can limit to the fact that the mechanism of the Retention graphs is unlike the others.
To duplicate the Retention graphs based on the number of input languages, a for loop has been applied: when we dynamically return the layout, we also call the function related to that vital sign and, in this case, we return more than one retention graph, each one with a different language from the input.

Listing 3.3: For-loop to return multiple charts

```
array=[]
res = html.Div(children=array)

for x in language:
    fig=retention_graph(x, retention_rate, len(params))
    array.append(fig)

return res
```

In this way we by-pass the problem of go not supporting the `facet_row` keyword. We simply create multiple graphs and we return it all, yielding the visualization in Figure 3.4.

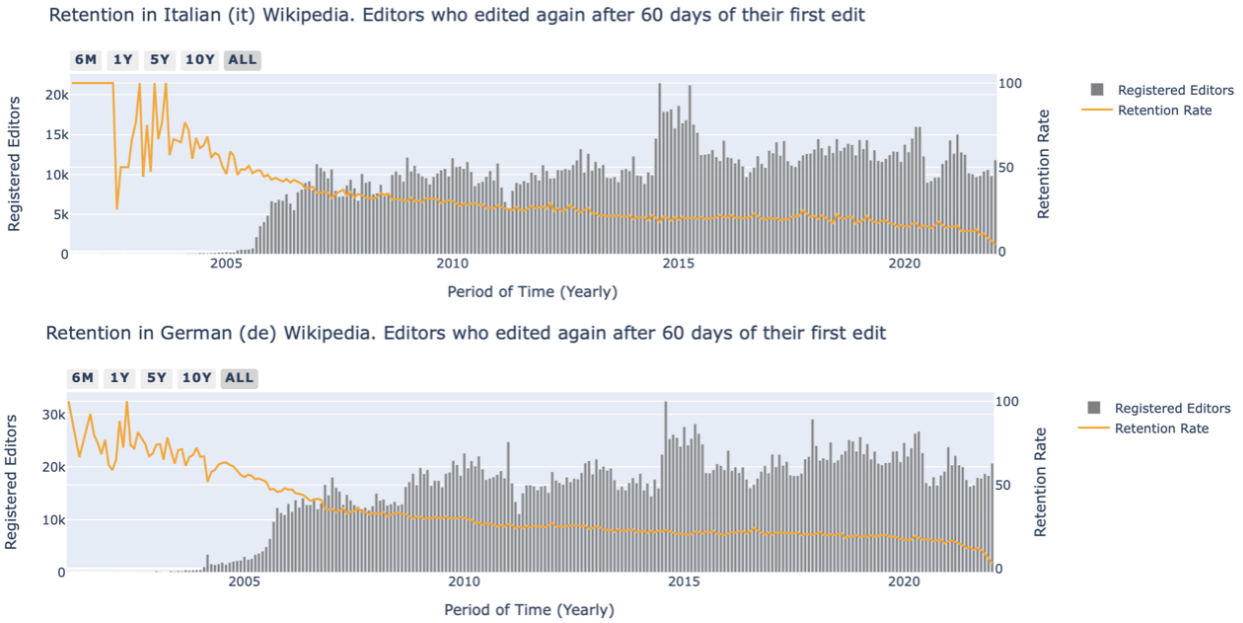

Figure 3.4: Multiple retention rates

Here we can see an example of the retention graphs when multiple language are selected, in this case Italian and German.
This method has some drawbacks though: mainly, the visualization is not as compact as it would be with the `facet_row` method, in fact the titles and the legends are repeated.

3.4 Stability

The second vital sign is Stability and it measures the the persistence of active and very active editors, as well as by the succession of the various generations of editors over time. It is a useful indicator since we want to know if a certain language community is able to attract new editors, called "fresh", and at the same time if it has solid community composed by long-term engaged editors.
For this metric it is convenient to classify the editors based on the number of months for which they kept editing Wikipedia's pages. We have different bin, from one month to more than 24 months. The editors that keep editing for one month in a row are defined as "fresh editors".
In the stacked bar chart below we can see that each bin of continuous activity is colored differently.

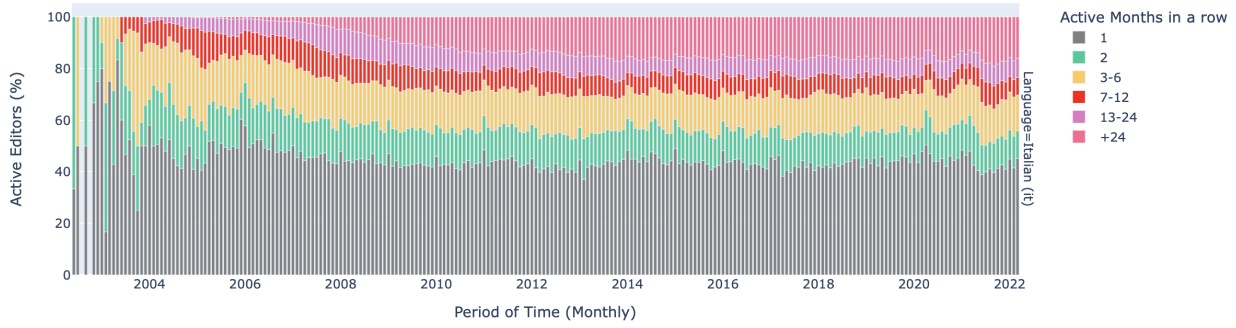

Figure 3.5: Stability dashboard

The persistence of each class of editors is represented in percentage, but it could be shown by absolute values as well through the radio buttons of the HTML components.

3.4.1 Stability implementation

The stability metric is the one that introduces the most common graph in the dashboards: the stacked bar chart.

Despite it being different from the line chart, this type of visualization has been realized in a very similar way. First, we query the data from the database, reading the input parameter for the languages, the type of editors that can be active or very active, and the type of value that we want to see, percentage or absolute. This parameter, called `value_type`, will not be included in query, but it is necessary for the correct visualization of the chart: if the user wants to have the absolute values, nothing happens; instead, for a percentage view we have to create a new data column by dividing the number of a certain class of editors by the total number of editors in that month.
After having retrieved the data, we create the final plot using the plotly express `px.bar()` method.

Listing 3.4: Creating a stacked bar chart

```
fig = px.bar(df,
             x='year_month',
             y=value_type,
             color='m2_value',
             text=value_type,
             facet_row=df['language_name'],
             width=1200,
             height=height_value,
             color_discrete_map={"1": "gray",
                                 "2": "#00CC96",
                                 "3-6": "#FECB52",
                                 "7-12": "red",
                                 "13-24": "#E377C2",
                                 "24+": "#636EFA"},
             labels={"year_month": time_text,
                     "perc": incipit+" Editors (%)",
                     "m2_value": "Active Months in a row",
                     "m2_count": incipit+" Editors",
                     "language_name":"Language"})
```

We can observe two important keywords: `color_discrete_map` and `facet_row`. The first one is about the style of the chart. We set a dictionary that maps our data column to specific colors that we decide, in order to have the full control on the final visualization. In this example it is important to avoid using similar colors in order to ensure the contrast between each editor class. The mapping can be seen in the legend on the right in Figure 3.5.
The second keyword, `facet_row`, is essential for the dashboards. We previously talked about this keyword in retention, explaining how we mimed its mechanism with a for-loop.
The `facet_row` attribute [3] allows us to set a data column from the data frame, in order to multiply the graph based on each different value of that column. In this example and in the majority of the dashboards we use the `langcode` data column. In this way, we will have a different visualization for each input language, allowing the comparison of the vital signs across the language communities. A meaningful milestone for the WikiCHM project.

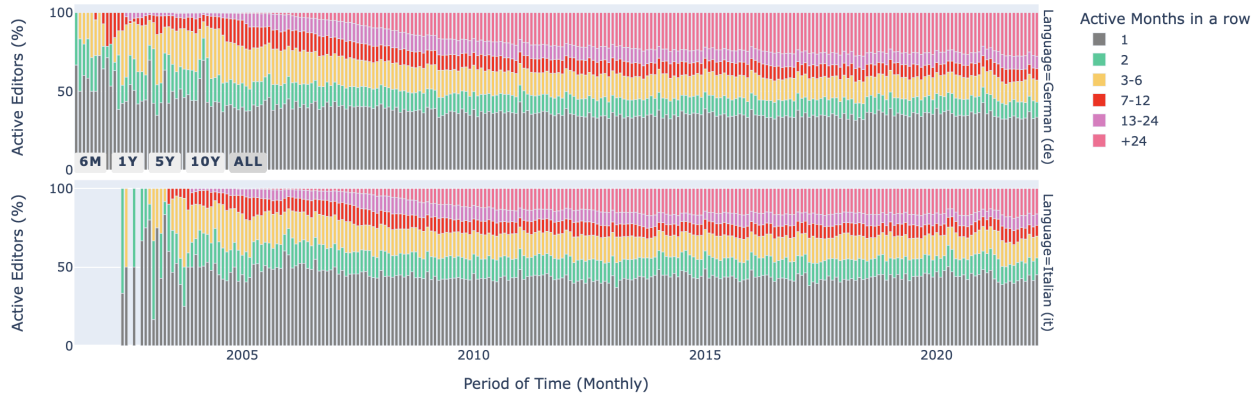

Figure 3.6: Faceted stability

[3] facet row is an attribute specific of plotly express

In Figure 3.6 can observe the result of the, so called, "faceted graph" that yields a more compact view than the for-loop method: for example, there is only one legend for all the charts and each of them is identified by the language on the right.
We can also observe that the size of the two graphs is smaller compared to the previous Figure 3.5. This is due to the fact that having many languages, thus many different graphs at their maximum size, would rapidly crowd the page in a way that would prevent making comparison between the languages at the top and the ones at the bottom. We in fact decided to dynamically reduce the size of the graphs, preventing them from becoming too small to be analyzed. However, a maximum of five languages at a time is recommended.

3.5 Balance

The third vital sign is community Balance. It measures the ability of a certain community of maintaining an equal proportion of old and new contributors. Previously, in stability, we monitored the distribution of some community's editors. This time, we measure the interaction of the community towards the editors, in particular how well a certain language manage the phenomenon known as "generational change". For a better understanding of this metric, it is useful to define the concept of "generation of editors": a generation is considered to be five years. This time we discriminate the editors based on their temporal belonging, not on their continuous contribution.
In this visualization we can see different colors representing the different generations, from the birth of Wikipedia until our current days (and beyond). The stacked bar chart is very convenient: for a given year, or month, we can observe the actual distribution of different generation of user, enabling the possibility of doing quantitative analysis. As was said before, the metrics support absolute and percentage values. In this case, a percentage visualization is recommended.

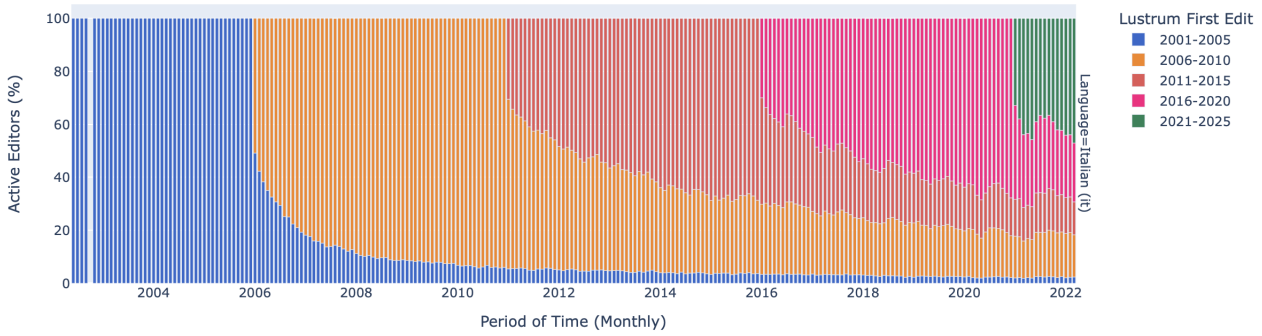

Figure 3.7: Balance dashboard

3.5.1 Balance implementation

The function related to the Balance metric is not much different from the stability one. They work in a very similar way. We take the opportunity to re-explain the basic workflow of each callbacks.
The Dash function, linked to the Dash app, takes as inputs some components' value of the web app layout. In this case, we consider the selected languages, if we talk about active or very active editors, absolute or percentage values and the timely aggregation of the chart, that can be yearly or monthly; usually, a yearly chart produces more general and meaningful representation of the data and its trends, while a monthly time aggregation favors a discrete and finer-grain data analysis.
Then, after having processed the input in order to make them SQL-compliant, we feed them to a parameterized query, that will produce a result set later captured in a Pandas data frame, convenient for the plotly charts.
Finally, we create the figure, in this case a stacked bar chart, that is coded exactly as a common bar chart. We first set the core attributes, like which data column must be placed on the x-axis, y-axis and the `facet_row`, and then the style attributes, like the title, the labels and the size.
Once we have this ready, we return the figure to the function responsible for the dynamic layout, that will actually display it in the appropriate HTML `div`.

3.6 Special functions

The fourth vital sign is called Special functions. It is about editors which hold a certain role that allows them to perform special actions in Wikipedia. In particular, we have two distinct roles: Technical functions and Community coordination functions.
As said in the first chapter, the first three metrics were about editors. We point out that from now on the dashboards are about vital signs related to Wikipedia specific functions like, in this case, the technical and coordination functions.
Since we have two different classes of functions, we have two different graphs. Two different bar charts that, as the previous vital sign, discriminate the editors based on their generation. The colors are in fact the same.
It is evident how the two visualizations look very similar, except for the data distribution. In the dashboards they are paired with a descriptive title.

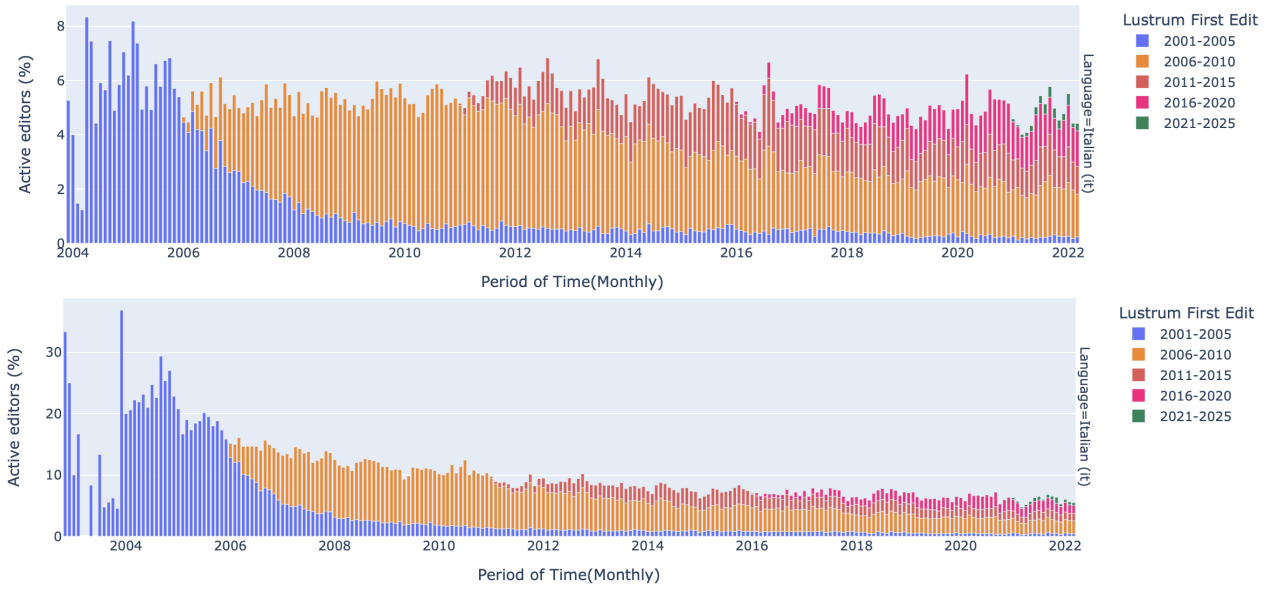

Figure 3.8: Technical and Coordination functions dashboards

3.6.1 Special functions implementation

These two different dashboards have two different callbacks. The two functions works exactly like the other, reading inputs, making a query and creating a figure. To be noted the mechanism with which we return two different graphs in the same layout.
We said that the Dash layout contains many HTML components, like the `div` container. There are two of them because some health metric, like Special functions, have two charts to be displayed. In this way we take in account this necessity. The functions will output in their respective containers. Also the faceted graph will appear in their intended place.

3.7 Admin flags

The fifth vital sign is about another kind of specific Wikipedia aspects. This time it is about Administrators. This metric analyze the distribution in time of editors which holds a position of Wikipedia administrator, with respect to their generation[4].
For this metric the dashboards display three different bar charts.
The first one represent the absolute distribution of a certain admin flag over the years. The flag is chosen through the HTML interactive interface.
Looking at the charts in Figure 3.9 we move to the right to find the second graph: it is a compact view of the first one, since it sums up the editors number discriminating on their generation. The

[4]The concept of generation is the same for Balance and Specialists

same color as the previous metrics has been kept to enhance the readability and preserve continuity.
At the bottom of the Figure 3.9, the third graph is found. It is about the percentage distribution of the previously mentioned admin flag. The title of the graphs changes as the admin flag chosen changes.
This time we do not divide the editors in different class, because we are interested only in the percentage distribution.

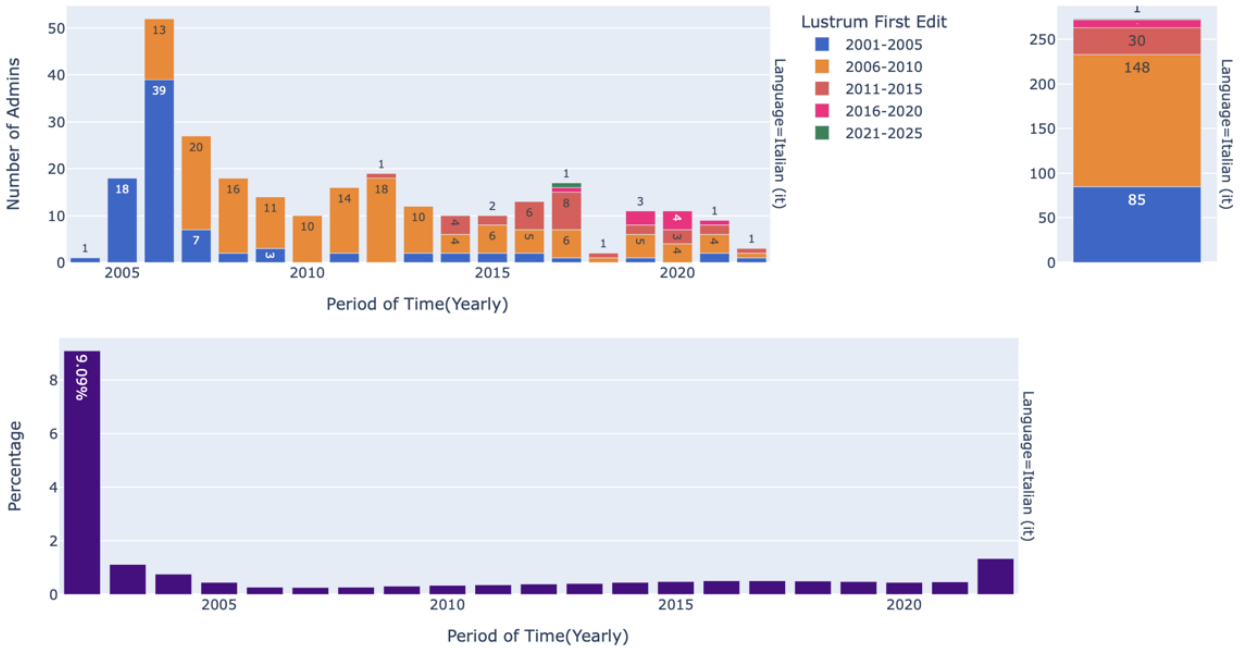

Figure 3.9: Admin flags dashboards

3.7.1 Admin flags implementation

The admin flags' callback has an interesting aspect regarding on which data column we decide to `GROUP BY`. We will not explain how the stacked bar chart are created to avoid time-consuming repetition.
We focus on the small chart on the right. It may seem like an extension of the first chart and thus bound to it. But in reality it is an independent figure.
The figure itself is created aside, but its data frame is related to the one used by the first graph.
For the latter, we retrieve the data about admin editors and we group by the time and by the generation, in order to have the data about different temporal belonging on the same column of the x-axis. Instead, for the second graph we group by only for the generation, to which we refer as `m2_value` in the python script. Furthermore, since we are only interested in one-dimension of our data, the absolute values, a new empty data column is created and it is used as the x-axis of this figure, yielding a compact view of our data.

3.8 Global community

The sixth and last vital sign brings the attention to the Global community of Wikipedia.
Since Wikipedia is a community project, the participation of each local community in the global community is essential for making their voices heard and learning from others. In this metric we analyze the linguistic origin of the contributions to a certain language community and the contribution of each Wikipedia language community to the global project.
This last dashboard has two different charts to satisfy our two different analysis.
The first bar chart visualization display the percentage or absolute distribution of all the contributions coming from every available language, towards the chosen language community.
For the second analysis we use a peculiar type of graph: a tree-map. As said before, it sums up

the individual contribution of each different language communities to the global community. Every language is represented as a colorful rectangle whose dimension is proportional to the percentage value of that language contribution to the Meta-Wiki project [5]. The rectangles also shows some additional details like the percentage value and the absolute value. By clicking on one of them there is a little nice animation that expands the whole rectangle, allowing to analyze the small communities too.

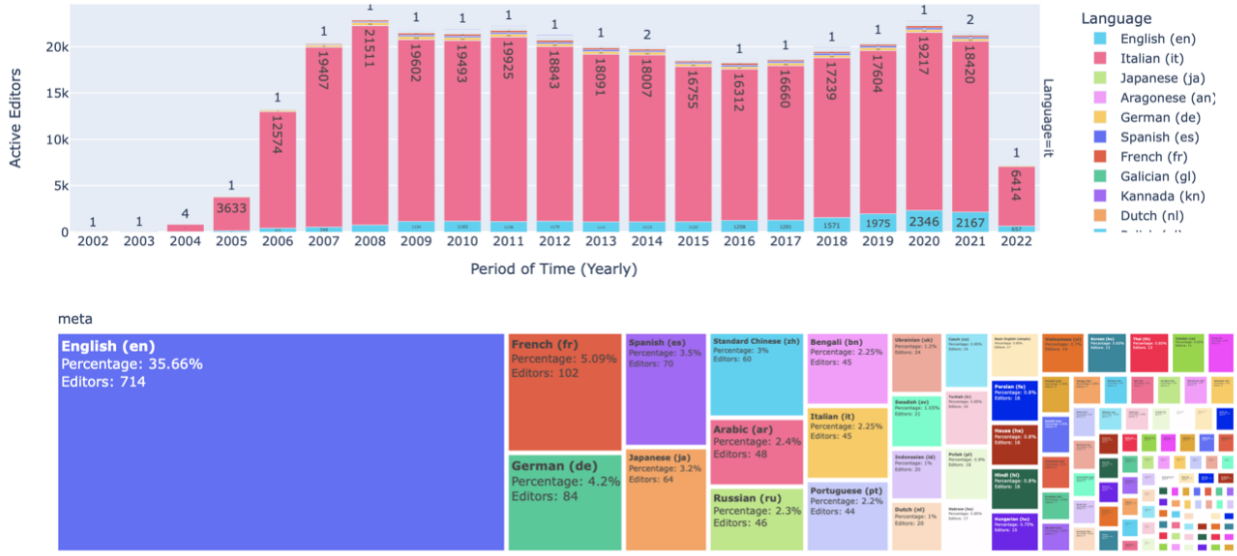

Figure 3.10: Global participation to Italian Wikipedia and Meta-wiki global participation

3.8.1 Global community implementation

These last dashboards are essential for the project goal since without them we would not have a full picture on the health status of the different communities in Wikipedia. These charts have been realized with a similar code to the others but this time the colors play an important role, since we cannot distinguish different languages in the charts if not with the colors. Unfortunately, there has not been found a way to describe a mapping for every possible language to a unique color since there are simply too many of them. Nonetheless, the results have been considered clearer than what was expected, in particular on the first graph.
About the second chart, the tree-map, we used the plotly sub-module `go`, in a similar way to what was done for the retention visualization. It has been preferred to plotly express for its low-level control on the graph figure itself, at the price of a slightly more complex syntax.

Listing 3.5: Creating a stacked bar chart

```
fig2 = go.Figure()

fig2.add_trace(go.Treemap(
    parents = df2["langcode"],
    labels = df2['language_name'],
    values = df2['m2_count'],
    customdata = df2['perc'],
    text=df2['m2_value'],
    texttemplate = '''<b>%{label} </b><br>Percentage:
                    %{customdata}%<br>Editors: %{value}<br>''',
    hovertemplate='''<b>%{label} </b><br>Percentage:
                    %{customdata}%<br>Editors: %{value}<br>
                    %{text}<br><extra></extra>''',
    ))
```

We created a `go.Figure` and then we added the tree-map trace, by specifying the common parent for every node (the rectangles) that is the Meta-Wiki topic. Then we set the data column to be textually displayed with the keyword `texttemplate` and to be displayed when we pass over with

the mouse pointer with `hovertemplate`. In this way we obtain the visualization in Figure 3.10 that correctly show us the quantitative distribution of each language communities' contributions to the Meta-Wiki.

3.9 Highlights

Disclaimer: In the earlier chapter it was said that the health metrics visualization were paired with automated descriptive labels. In this section we are going to discuss their code and their role. The choice of not talk about them in every single graph was taken due to the fact that the mechanism is the same for every highlight, so we dedicate an entire section to this topic.

To enhance graphs' readability we use the Highlights: automated descriptive labels that help us reading the graph and its highest or lowest values. In general, they briefly contextualize the graph and describe what we are looking at.
These automatic caption have been coded in a very similar way to the one we used to create the charts: we generate a Pandas data frame with the same technique described before, then we use its data column values to extract the values that will be embedded in some prepared textual statement. A peculiarity of this mechanism is that the values needed some sort of pre-processing phase: the values had to be casted into a string to be concatenated in the textual statements or the date had to be converted in a user-friendly style like the common "Month Year" format.

In the end, we have at least one function for each graph about the highlights. But to actually place it in the layout we have a dash callback that reads which vital sign has been selected and return the result of the respective highlights function, preserving the dynamicity of the entire layout. In this way, we also ensure the highlights to be up to date with the data selected with the HTML components like the language selected or the time aggregation.

4 Results

In this chapter we are going to briefly discuss about the results and the trends emerged from the created dashboards. We will show how intuitive the comparison between different language is. For this purpose it will be used a language sample taken from the major European languages, namely Italian, German, French and Spanish. They will be compared under the same vital sign to understand the explanatory power of these interactive visualizations and their potential role in the future steps of the Wiki Community Health Metrics project.

4.1 Editors

We begin with Active editors. We remind they are the editors who edit Wikipedia pages at least five times per month. They are the majority of the editors, reaching up to 88% of the total editors (considering the major language community). The dashboards show how the number of active editors evolved since the birth of Wikipedia until present days.

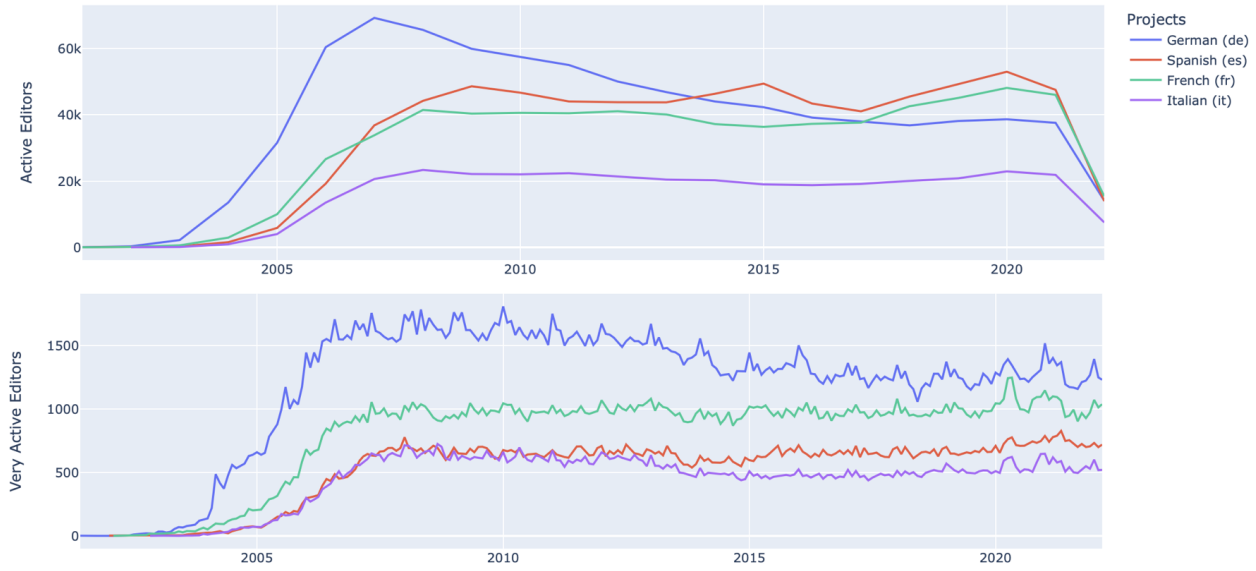

Figure 4.1: Trend of the major European languages' active and very active editors

Considering a yearly visualization , the first graph in figure 4.1 show us a general pattern for the major European language communities: after a rapid spike in the early days of Wikipedia, the number of active editors declined until, around the year 2014, they generally reached a stable value.
Considering the very active editors and monthly visualization, we get richer charts. The trend in Figure 4.1 stays the same but we can also see the number of very active editors fluctuating even during the years themselves: at the beginning of the new year the editors are more that the rest of the year.

4.2 Retention

We now consider the retention dashboards considering the same language sample as before. We remind that the retention rate of a certain community is calculated as the percentage of new editors that survives after a certain threshold from their first edit. We consider the threshold value of 60 days.

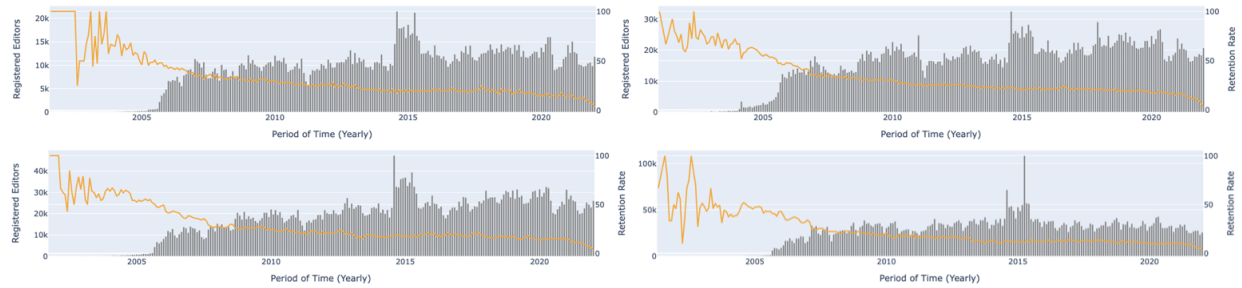

Figure 4.2: Trend of the major European languages' retention rates

The retention rates of our language sample shows a dramatic declining trend. In every language we observe a common pattern: in the early days we can see a lot of instability, then around the year 2004 the retention rate steadily decreases, especially in the last years. Ideally, the various community should adopt behavior to revert this trend.

4.3 Stability

Before analyzing the stability trends we remind that the stability of the community is given by the persistence of active and very active editors, as well as by the succession of the various generations of editors over time.

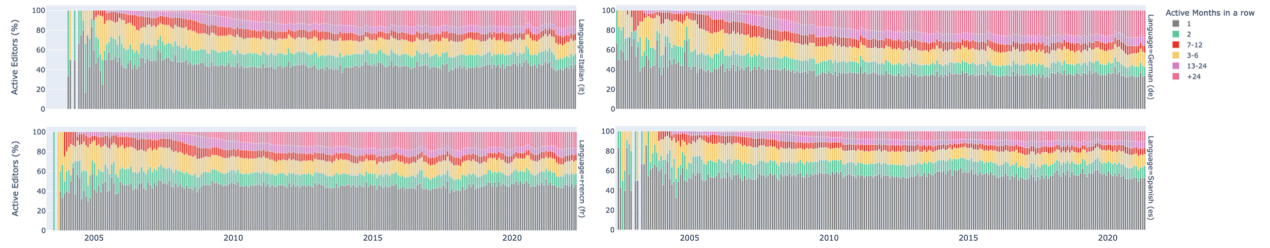

Figure 4.3: Trend of the major European languages' stability among active editors

Among active editors, the greater percentage of editors keeps editing for one month. Thus we could deduce that active editors tend to be volatile and less persistent. This trend is consistent in the whole language sample.

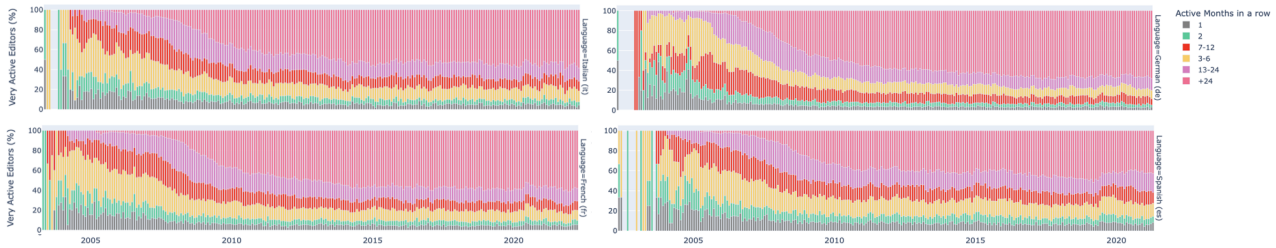

Figure 4.4: Trend of the major European languages' stability among very active editors

While among the very active editors, the main part is represented by the editors which keep editing for more than 24 months. Since being a very active editor require more effort than being simply active, we could argue that this very same effort pushes the editors towards being more and more dedicated to their role.

4.4 Balance

We now consider the balance visualizations. The Community balance metric measures the ability to maintain an equal proportion of old and new contributors.

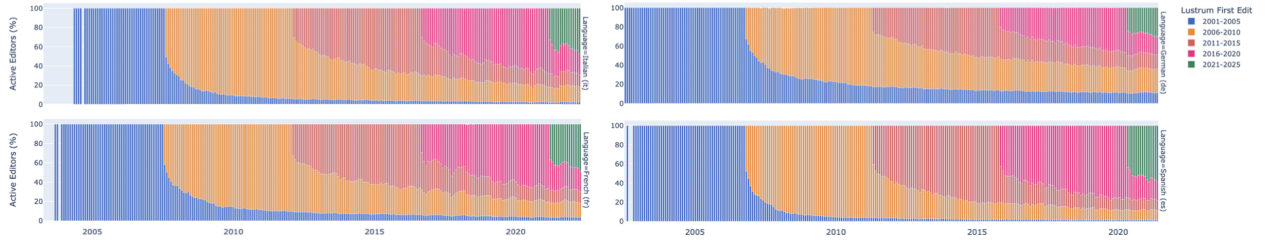

Figure 4.5: Trend of the major European languages' balance among active editors

There is a recurrent pattern: through the years, the new generations overtake the old ones, but not entirely. A generation is considered to be 5 years.
The last generation (2021-2025) should occupy from 10 to 40% depending on the years which have passed since its beginning, while the oldest generation (2001-2005) share should remain around 5 to 15%, not greater, to guarantee the generational change.

4.5 Special functions

Special functions are about editors which hold a certain role that allows them to perform special actions in Wikipedia. In particular, we have two distinct roles: Technical functions and Community coordination functions. We will now analyze them both.

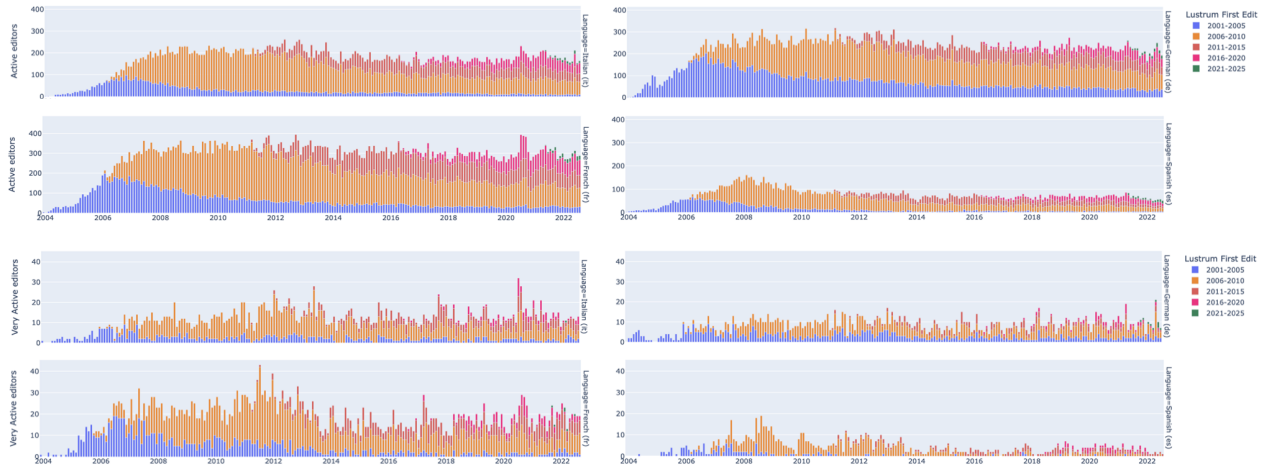

Figure 4.6: Trend of the major European languages' technical active and very active editors

First we have technical editors. The majority of the them belongs to the 2006-2010 generation. The last generation 2021-2025 struggles at finding his space even if the older generation are now retreating. From the Figure 4.6 we can see that, regardless of the editors being active or very active, this trend is consistent. It it also common among the different languages.

Figure 4.7: Trend of the major European languages' active and very active coordinator editors

Then we talk about coordinators. There are fewer and fewer new coordinators. Otherwise as before, this trend is more evident in the very active editors visualizations, as we can see that the color representing the second generation is more present than the other.
These trends could be the symptoms of a lesser interest in the specific functions of Wikipedia from the editors.

4.6 Admin flags

We now observe the admin distribution among our language sample. This time the result is independent from the type of editors because it is function-specific. For this example, we only consider the "sysop" flag since it is the more paradigmatic one among the admin flags.

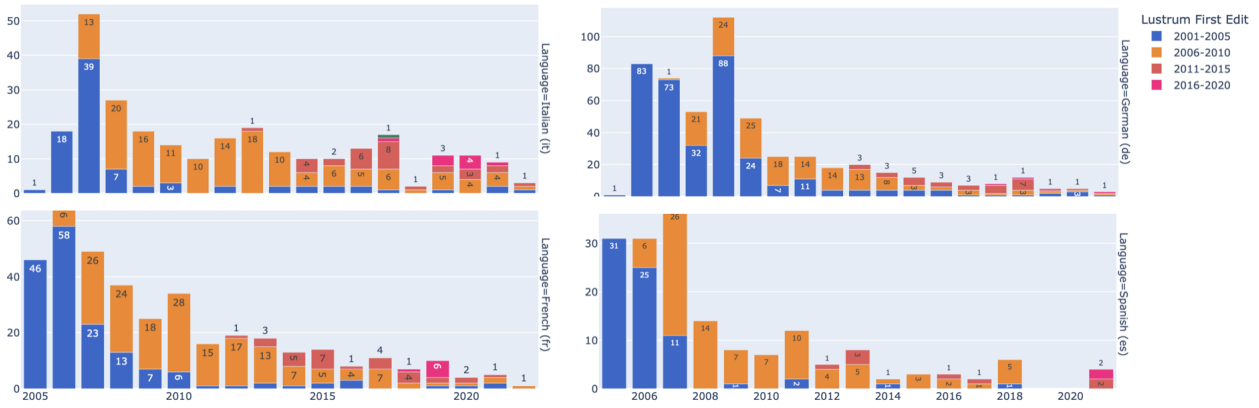

Figure 4.8: Trend of the major European languages' admin distribution

Admin flags were given mainly in the past, especially to the first and second generation's editors. In general, the percentage of a given admin flag, among active and very active editors, is borderline between being below and inline with the desired target of 1-5%.

4.7 Global community participation

Finally we discuss the trend emerging from the global community. These trends are as important as the others since they move the focus on the bigger picture regarding the Wikipedia project as a whole.

Figure 4.9: Trend of the major European languages' admin distribution

From the Figure 4.9, this time with a yearly time aggregation, we can see that the majority of the contributions for a certain community come mainly from their primary language editors. In fact, the higher percentage are of the same color[1] of the language analyzed. However, the English community contributes consistently to every language analyzed.

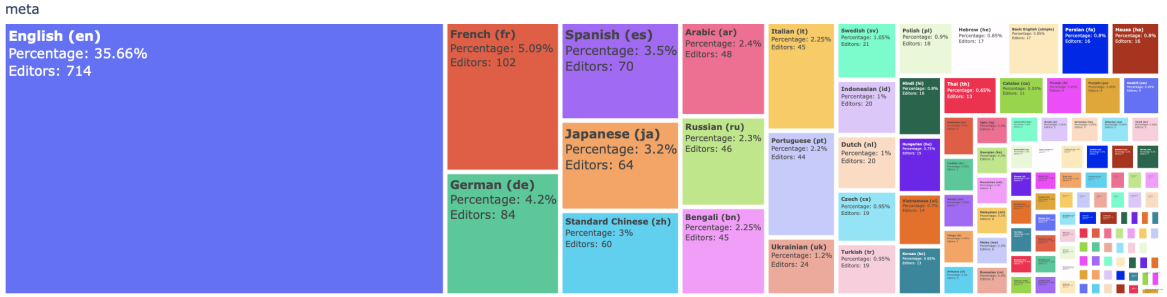

Figure 4.10: Meta-wiki global participation

Now we discuss the contribution of each language to the Meta-wiki. It is clear that the bigger language play a major role than the smaller ones. The English Wikipedia is the bigger contributor to the project, followed by other common languages like German, French but also Japanese and Chinese.

[1] Unfortunately the Italian and English happened to have the same color

This result is extremely interesting since it shows that the European languages are not alone in the Wikipedia field, but there are the Asian giants too.

5 Conclusions

As described in the previous chapter, the goal of my project was to build interactive online dashboards to allow quantitative and qualitative analysis to be done on the health status of the Wikipedia community, expressed under the sum of each contribution coming from every language community present on the online encyclopedia.

Results The dashboards are meant to be comprehensible by everyone, not only for data scientist and researcher. The WikiCHM project aims at spreading knowldge and awareness among the Wikipedians, in order to inspire them at acting in a way to improve the six community vital signs. For this very goal, the visualizations have been made minimal, easy to interact with and to understand. Particular attention has been made to the user experience to prevent unnecessary page reloading every time a paramter is changed, thus the use of asynchronous calllback and the implementation of an interactive layout.Furthermore, the automatic highlights make the charts even more human-friendly.
The dashboards help us read data that would be otherwise hard to understand in a raw form of a database.
In general, we can conclude the pattern analyzed from these dashboards are the same for every major language. In fact, it is evident that since the birth of Wikipedia the trends are overall negative, since the people are slowly drifting away from the online encyclopedia, risking to make it a static and immutable place.
We hope these dashboards can help solving this problem.

Contribution I have contributed at the project by

- handling and analyzing over 20 GB of data from Wikipedia database
- experimenting and discovering different kind of visualization and graphs for each health metric
- mainly, implementing and coding Python scripts to visualize and plot Wikipedia vital signs data
- studying different approaches for the code structure
- adapting and deploying the scripts from a local architecture to a remote server architecture
- studying how to improve the frontend user experience
- defining and generating automated description to enhance the graph

I'm personally very proud of having learned a new coding language, Python, and having realized a working and interactive website displaying visualizations about such an important topic as Wikipedia health metrics.
I hope that my small contribution to the bigger project may help the community to improve as in its early days.

Future project steps The project does not stop here.
This was only the beginning of sharing knowledge among the Wikipedians in order to empower them. The goal is to make them realize they have the online enclyclopedia's future in their hands.
In the future, the Eurecat research team will provide useful Wikipedia pages about these topics,

embedding the results in a table-like format, in order to make them accessible by everyone, on the Meta-wiki itself. The process will be automated, releasing the latest data each time they have them available.
The project will also publish and raise awareness on the existence of the dashboards, to make them a useful tool among editors and Wikipedians. The visualizations will always offer up to date charts.
The monitoring of Wikipedia health status will continue, by studying data and emerging and trends, and also by studying how the general public respond to the dashboards. Possibly, new visualizations will be realized or the existing ones will be corrected.

Bibliography

[1] Plotly - plotly graph objects. https://plotly.com/python/graph-objects/. [Online; accessed 21-June-2022].

[2] Plotly - plotly express. https://plotly.com/python/plotly-express/. [Online; accessed 21-June-2022].

[3] Dash overview. https://plotly.com/dash/. [Online; accessed 22-June-2022].

[4] Pandas - python data analysis library. https://pandas.pydata.org. [Online; accessed 22-June-2022].

[5] Wikimedia - meta-wiki main page. https://meta.wikimedia.org/wiki/Main_Page. [Online; accessed 23-June-2022].

[6] Consonni Cristian, Ribé Marc-Miquel, and Laniado David. Community health metrics. *Wikimedia project*, page 1, 2022.

[7] Lim Kian Long, Lim Chien Huiand, and Gim Yeong Fook. A study on the effectiveness of tree-maps as tree visualization techniques. *ScienceDirect*, pages 109–110, 2017.

[8] Dijkstra Edsger W. On the role of scientific thought. *Selected writings on Computing: A Personal Perspective*, pages 60–66, 1982.