

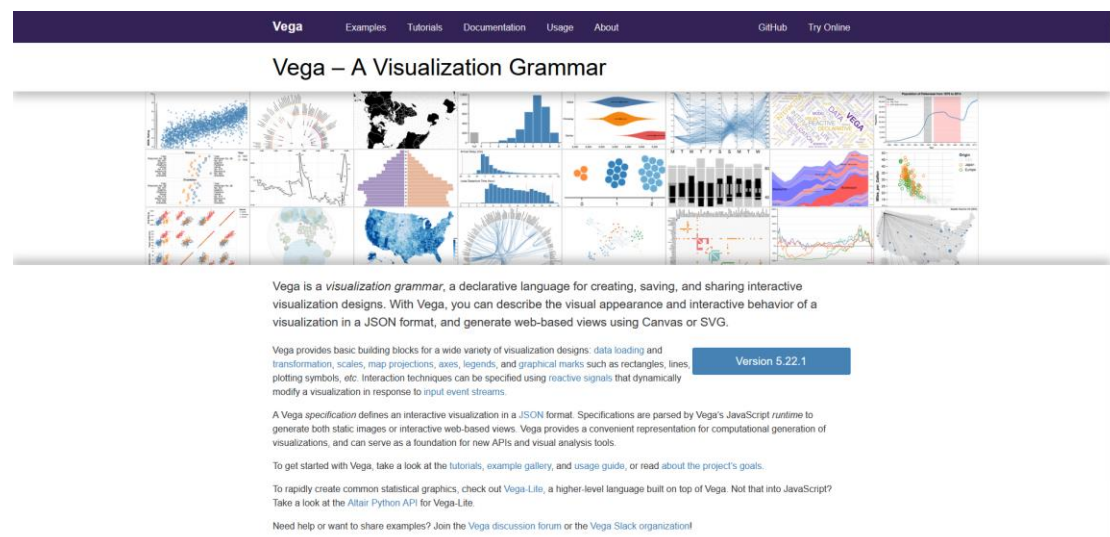
## Descripción del trabajo

En base a lo explicado en clase, se ha comenzado buscando un conjunto de datos abierto en la plataforma del Gobierno de España (<https://datos.gob.es/es/>).

Se ha realizado una búsqueda avanzada filtrando por "Empleo", "Administración del estado", "Estadísticas", "Datos con actualización anual" y "Mercado laboral y salarios". A continuación se presenta el enlace de la búsqueda:

[https://datos.gob.es/es/catalogo?administration\\_level=E&tags\\_es=Estad%C3%ADsticas&theme\\_id=empleo](https://datos.gob.es/es/catalogo?administration_level=E&tags_es=Estad%C3%ADsticas&theme_id=empleo)

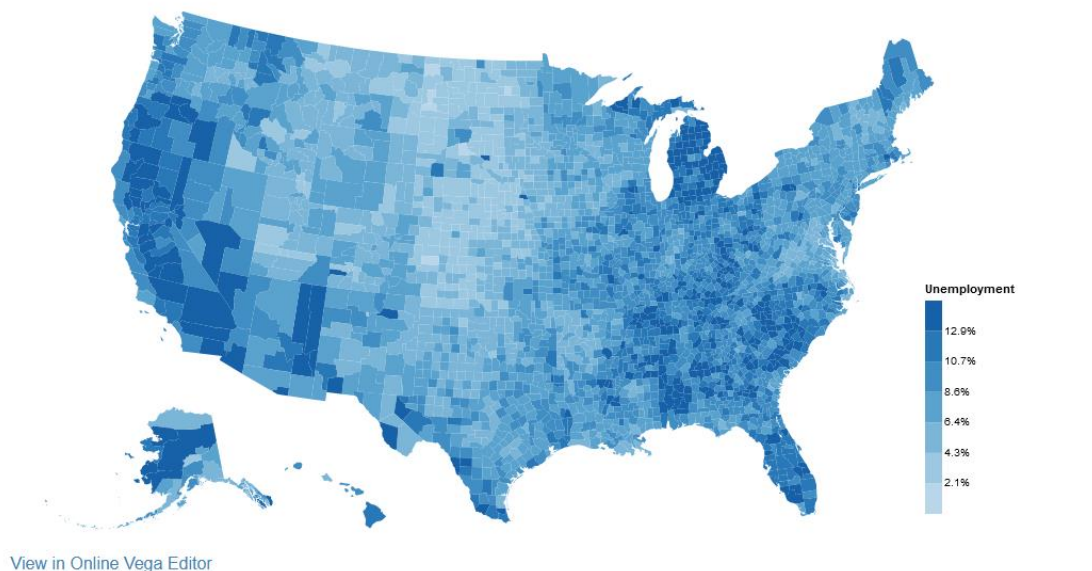
En un primer momento se han mirado los ejemplos que hay disponibles en la página de Vega, como se puede ver a continuación:



Concretamente, el que capturó mi atención fue el siguiente:

## County Unemployment Example

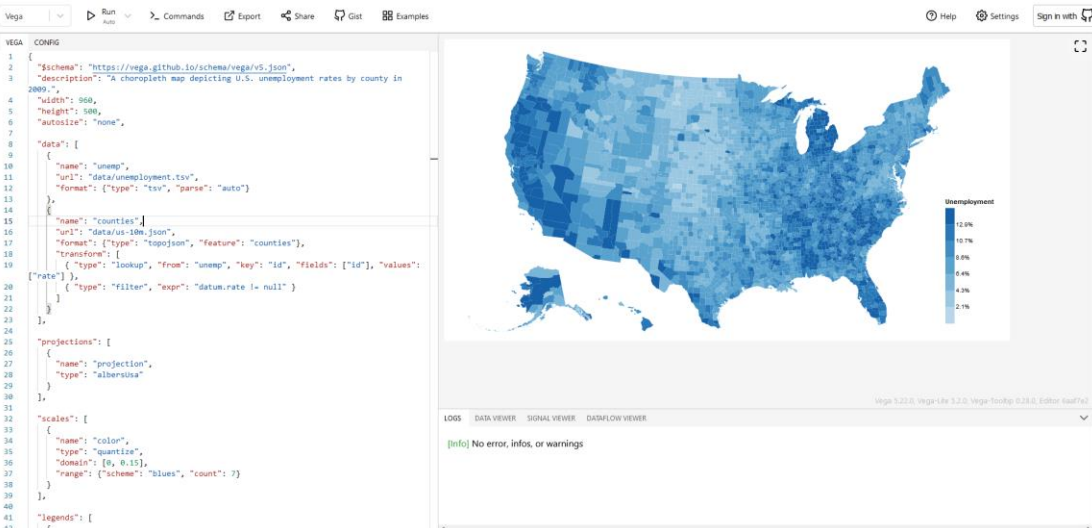
A [choropleth map](#) of 2009 U.S. unemployment rates by county. A [quantize scale](#) is used to divide the color range into seven discrete, uniformly-spaced bins.



### Vega JSON Specification <>

```
{
  "$schema": "https://vega.github.io/schema/vega/v5.json",
  "description": "A choropleth map depicting U.S. unemployment rates by county in 2009.",
  "width": 900,
  "height": 500,
  "autosize": "none",
  "data": [
    {
      "name": "unemp",
      "url": "data/unemployment.tsv",
      "format": {
        "type": "tsv",
        "parse": "auto"
      }
    },
    {
      "name": "counties",
      "url": "data/us-10m.json",
      "format": {
        "type": "topojson",
        "feature": "counties"
      },
      "transform": [
        {
          "type": "lookup",
          "from": "unemp",
          "key": "id",
          "fields": ["id"],
          "values": ["rate"]
        },
        {
          "type": "filter",
          "expr": "datum.rate != null"
        }
      ]
    }
  ],
  "projections": [
    {
      "name": "projection",
      "type": "albersusa"
    }
  ],
  "scales": [
    {
      "name": "color",
      "type": "quantize",
      "domain": [0, 0.15],
      "range": {
        "scheme": "blues",
        "count": 7
      }
    }
  ],
  "legends": [
  ]
}
```

Aquí vemos el código JSON del gráfico de coropletas:




De modo que mi idea inicial fue buscar un mapa de coropletas de España (ya que el conjunto de datos es sobre las comunidades autónomas de nuestro país), y realizar algún tipo de animación que mostrara la evolución a lo largo del tiempo, cambiando los colores de cada zona. De modo que me puse a buscar mapas en la web:

geojson map of spain

Todo Imágenes Vídeos Noticias Mapas Ajustes

Spain (es) Búsqueda segura: moderado En cualquier momento

Imágenes para geojson map of spain



Más Imágenes →

<https://geojson-maps.ash.ms>

### GeoJSON Maps of the globe

To use your map with Leaflet: Rename your downloaded geojson file and publish it to your webserver. Create a HTML page using the template below. Replace myGeoJSONPath with the path to your own file. That's it! The Code

→ [https://github.com/codeforgermany/click\\_that\\_hood/blob/main/public/data/spain-provinces.geojson](https://github.com/codeforgermany/click_that_hood/blob/main/public/data/spain-provinces.geojson) at main ...

click\_that\_hood / public / data / spain-provinces.geojson Go to file Go to file T; Go to line L; Copy path Copy permalink; This commit does not belong to any branch on this repository, and may belong to a fork outside of the repository. Cannot retrieve contributors at this time.

Y encontré el siguiente:


codeforgermany/click\_that\_hood

main click\_that\_hood / public / data / spain-provinces.geojson

Latest commit 272e3c on 25 Dec 2014 History

1 contributor

1 lines (1 blob) 669 KB

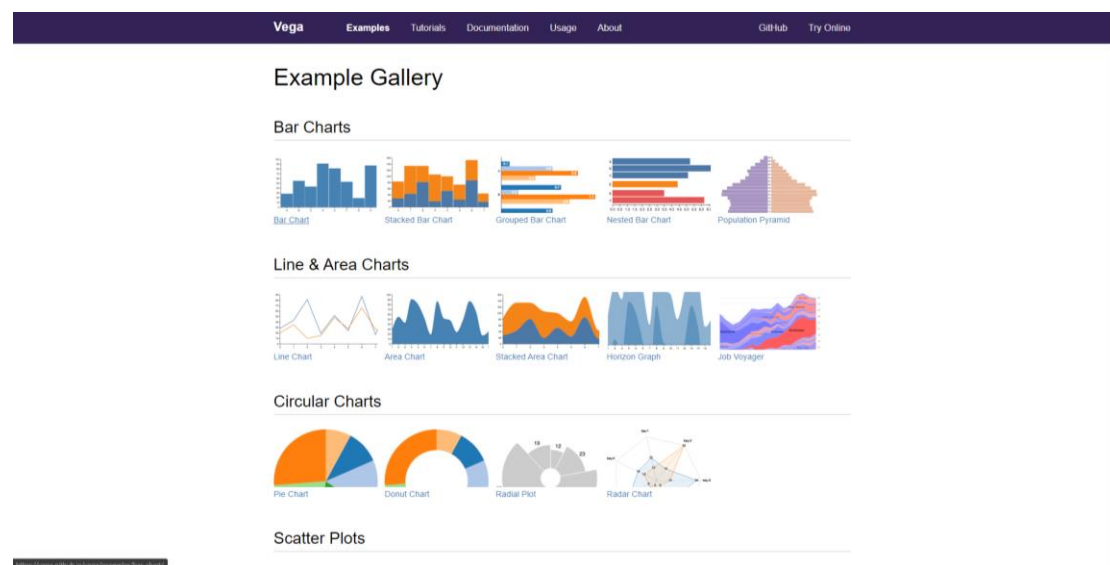


Y una vez con el mapa, me imaginé cómo sería el resultado, haciendo que las comunidades cambiaran de color en base al % de paro. Hize el siguiente boceto:



La verdad es que no me pareció tan buena idea como al principio, de modo que decidí desecharla.

Navegué hasta la sección de ejemplos, y decidí probar suerte con el primero que aparecía, el gráfico de barras:



Una vez se entra en la página, se ve el código del gráfico:

[Vega](#)[Examples](#)[Tutorials](#)[Documentation](#)[Usage](#)[About](#)[GitHub](#)[Try Online](#)

## Bar Chart Example

A bar chart encodes quantitative values as the extent of rectangular bars. This example includes basic highlighting and tooltips on mouse hover. For a step-by-step guide to building this visualization, see the [bar chart tutorial](#).

Category	Amount
A	20
B	55
C	43
D	91
E	83
F	53
G	19
H	87

[View in Online Vega Editor](#)

### Vega JSON Specification <>

```
{
  "schema": "https://vega.github.io/schema/vega/v5.json",
  "description": "A basic bar chart example, with value labels shown upon mouse hover.",
  "width": 400,
  "height": 200,
  "padding": 5,
  "data": [
    {
      "name": "table",
      "values": [
        {"category": "A", "amount": 20},
        {"category": "B", "amount": 55},
        {"category": "C", "amount": 43},
        {"category": "D", "amount": 91},
        {"category": "E", "amount": 83},
        {"category": "F", "amount": 53},
        {"category": "G", "amount": 19},
        {"category": "H", "amount": 87}
      ]
    }
  ]
}
```

Podemos hacer click en el botón que aparece justo debajo del código para editarlo en una nueva pestaña:

Vega

Run

Commands

Export

Share

Gist

88 Examples

Help

Settings

Sign in with

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

```
1 {
2   "schema": "https://vega.github.io/schema/vega/v5.json",
3   "description": "A basic bar chart example, with value labels shown upon mouse hover.",
4   "width": 400,
5   "height": 200,
6   "padding": 5,
7
8   "data": [
9     {
10      "name": "table",
11      "values": [
12        {"category": "A", "amount": 20},
13        {"category": "B", "amount": 55},
14        {"category": "C", "amount": 43},
15        {"category": "D", "amount": 91},
16        {"category": "E", "amount": 83},
17        {"category": "F", "amount": 53},
18        {"category": "G", "amount": 19},
19        {"category": "H", "amount": 87}
20      ]
21    }
22  ]
23
24  "signals": [
25    {
26      "name": "tooltip",
27      "value": {},
28      "on": [
29        {"events": "rect:mouseover", "update": "datum"},
30        {"events": "rect:mouseout", "update": "{}"}
31      ]
32    }
33  ],
34
35  "scales": [
36    {
37      "name": "xscale",
38      "type": "band",
39      "domain": {"data": "table", "field": "category"},
40      "range": "width",
41      "padding": 0.45,
42      "round": true
43    },
44    {
45      "name": "yscale",
46      "domain": {"data": "table", "field": "amount"},
```

Vega 5.22.0, Vega-Lite 5.2.0, Vega-Tooltip 0.20.0, Editor 0.0.1

LOGS DATA VIEWER SIGNAL VIEWER DATAFLOW VIEWER

Ahora, convertí el fichero CSV con el conjunto de datos a formato JSON, para poder usarlo más fácilmente en el editor web:







Como se puede apreciar en la siguiente imagen, en la documentación se describe que las transformaciones sirven para “filtrar” el conjunto de datos y de esa forma mostrar solamente los datos que se precisen:

## Data

The `data` property is an array of data definitions. Each entry in the data array must be an object with a unique `name` for the data set. As shown here, data can be directly defined inline using the `values` property. In this example, we have an array of data objects with fields named `category` (a string label) and `amount` (a number).

```
"data": [
  {
    "name": "table",
    "values": [
      {"category": "A", "amount": 28},
      {"category": "B", "amount": 55},
      {"category": "C", "amount": 43},
      {"category": "D", "amount": 91},
      {"category": "E", "amount": 81},
      {"category": "F", "amount": 53},
      {"category": "G", "amount": 19},
      {"category": "H", "amount": 87}
    ]
  }
],
```

In Vega specifications, data can be:

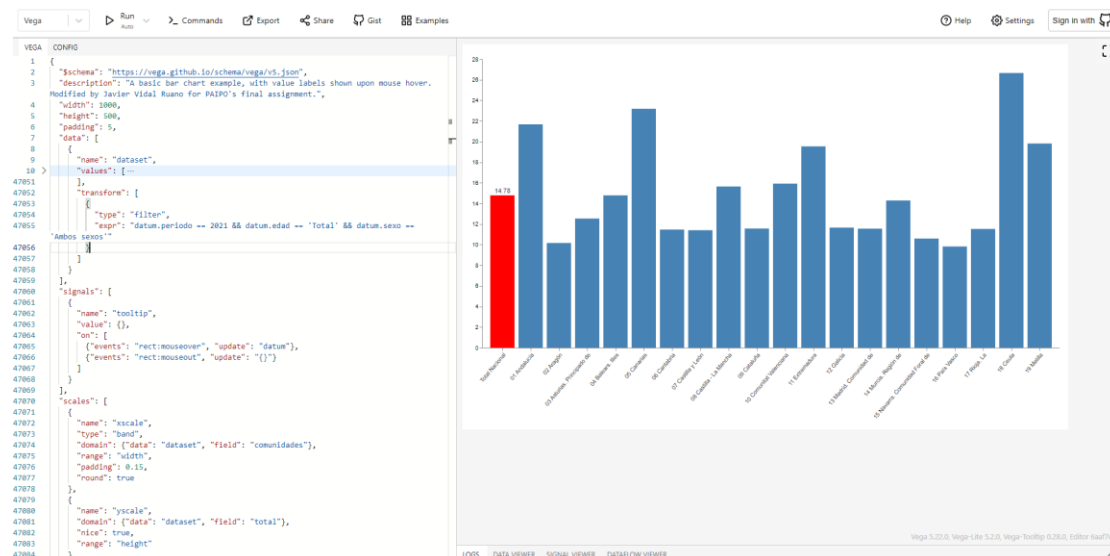
- loaded from the web by using the `url` property (including JSON and CSV files),
- derived from a previously defined data set using the `source` property, or
- left undefined and dynamically set when the visualization is constructed.

Only one of the `values`, `url` or `source` properties may be defined.

Data sets in Vega can be modified using a collection of transforms such as filtering, aggregation and layout operations. Transformations are specified using the `transform` property, which takes an array of transform definitions.

For more details, see the [data](#) and [transform](#) documentation.

De modo que se ha usado una transformación de prueba para comprobar el funcionamiento, y el resultado es satisfactorio:



Una vez tenemos la posibilidad de realizar filtros sobre los datos, lo que falta es permitir al usuario elegir dichos filtros, para que la gráfica sea interactiva. Para ello, se puede usar las señales, como describe la documentación:



In addition to standard graphical marks (rectangles, arcs, plotting symbols, etc), Vega also supports nested marks through the special `group` mark type. Groups are marks that can contain other marks, for example to create [small multiple displays](#). Groups can also include custom `scales` and `axes` definitions that are specific to a group instance.

For more details, see the [marks](#) documentation.

## Signals

Signals act as dynamic variables: expressions that are automatically reevaluated when other signal values change, or when input events occur. Each signal must have a unique `name` and an initial `value`; other properties define how the signal value can change.

Here we use a signal to define a tooltip interaction. In this example, the value of the `tooltip` signal changes in response to `mouseover` and `mouseout` events on `rect` marks. Every time these events occur, the corresponding expression is evaluated and set as the `tooltip` value. Thus, when the mouse pointer is moved over a rectangle mark, `tooltip` is equal to the mark's backing datum; when the pointer is moved off the rectangle, `tooltip` is an empty object.

```
"signals": [{
  {
    "name": "tooltip",
    "value": {},
    "on": [
      {"events": "rect:mouseover", "update": "datum"},
      {"events": "rect:mouseout", "update": "{}"}
    ]
  }
],
```

Our `tooltip` signal tracks the datum for the currently highlighted bar. We now use this signal to dynamically adjust the visual encoding rules of a text label:

```
{
  "marks": [
    ...,
    {
      "type": "text",
      "encode": {
        "enter": {
          "align": {"value": "center"},
          "baseline": {"value": "bottom"},
          "fill": {"value": "#333"}
        },
        "update": {
          "x": {"scale": "xscale", "signal": "tooltip.category", "band": 0.5},
          "y": {"scale": "yscale", "signal": "tooltip.amount", "offset": -2},
          "text": {"signal": "tooltip.amount"},
          "fillOpacity": [
```

```

    ...,
    {
      "type": "text",
      "encode": {
        "enter": {
          "align": {"value": "center"},
          "baseline": {"value": "bottom"},
          "fill": {"value": "#333"}
        },
        "update": {
          "x": {"scale": "xscale", "signal": "tooltip.category", "band": 0.5},
          "y": {"scale": "yscale", "signal": "tooltip.amount", "offset": -2},
          "text": {"signal": "tooltip.amount"},
          "fillOpacity": [
            {"test": "isNaN(tooltip.amount)", "value": 0},
            {"value": 1}
          ]
        }
      }
    }
  ]
}

```

A single text mark instance serves as our tooltip text (note that the `from` property is omitted). The position and text values are drawn directly from the `tooltip` signal. To only show the tooltip text when the mouse pointer is over a rectangle, we set the `fillOpacity` using *production rules*: a chain of if-then-else rules for visual encoding. If `tooltip` is an empty object, the tooltip text is fully transparent since `isNaN(tooltip.amount)` is `true`, otherwise it is opaque.

Signals can be applied throughout a specification. For example, they can be used to specify the properties of transforms, scales and mark encodings. For more details, see the [signals](#) documentation.

## Next Steps

We've now worked through a full Vega visualization! Next, we recommend experimenting with and modifying this example. Copy & paste the full specification above into the online [Vega Editor](#) or fork [our example Block](#).

- Can you adjust the scales and axes?
- Can you change the chart from a vertical bar chart to a horizontal bar chart?
- Can you visualize a new data set with a similar structure?

You should then be ready to understand and modify other examples. Many of the more advanced examples include data transforms that organize data elements and perform layout. As you experiment with different examples, you may find it useful to refer to the documentation for each of the main specification components.

[Edit this Page](#)



La primera línea de la página sobre las señales, describe que se pueden usar (una vez definidas) en toda la especificación de Vega. Esto sería lo mismo que definir una variable global, si estuviéramos programando:

Documentation  
Specification  
Config  
Data  
Transforms  
Triggers  
Projections  
Scales  
Schemes  
Axes  
Legends  
Title  
Marks  
**Signals**  
Event Streams  
Expressions  
Layout  
Types

## Signals

**Signals** are dynamic variables that parameterize a visualization and can drive interactive behaviors. Signals can be used throughout a Vega specification, for example to define a mark property or data transform parameter.

Signal values are *reactive*: they can update in response to input event streams, external API calls, or changes to upstream signals. [Event streams](#) capture and sequence input events, such as `mousedown` or `touchmove`. When an event occurs, signals with associated event handlers are re-evaluated in their specification order. Updated signal values then propagate to the rest of the specification, and the visualization is re-rendered automatically.

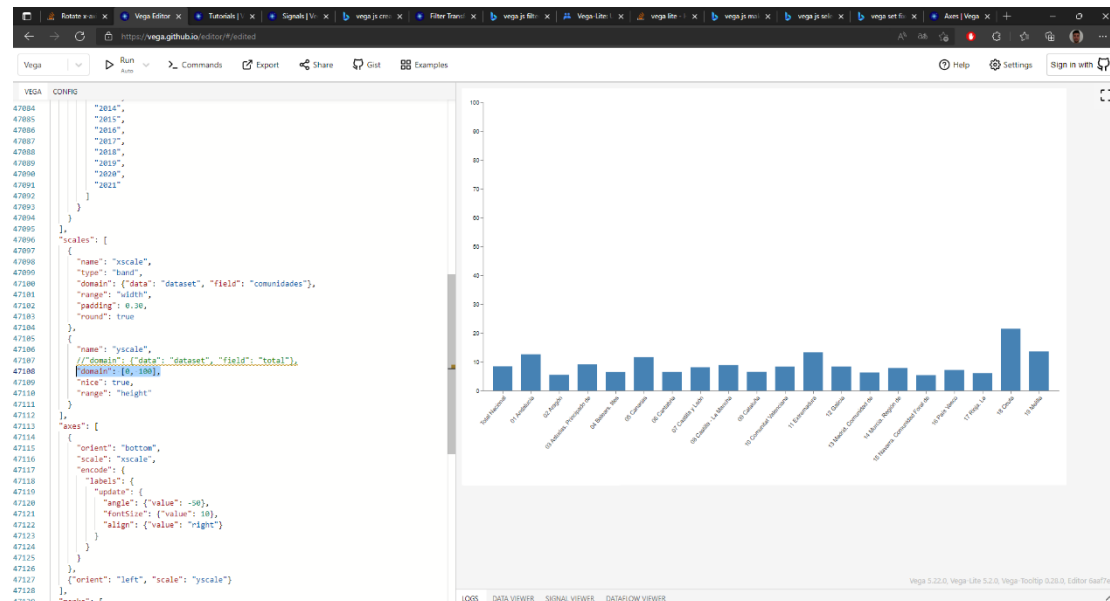
A signal definition, and its use in the rest of a specification, looks something like this:

```

{
  "signals": [
    {
      "name": "indexDate",
      "description": "A date value that updates in response to mousemove.",
      "update": "datetime(2005, 0, 1)",
      "on": [{"events": "mousemove", "update": "invert('xscale', x())"}]
    }
  ],
  "data": [
    {
      "name": "stocks", ...
    }
  ],
  "scales": [
    { "name": "x", "type": "time", ... }
  ],
  "marks": [
    {
      "type": "rule",
      "encode": {
        "update": {
          "x": {"scale": "x", "signal": "indexDate"}
        }
      }
    }
  ]
}

```

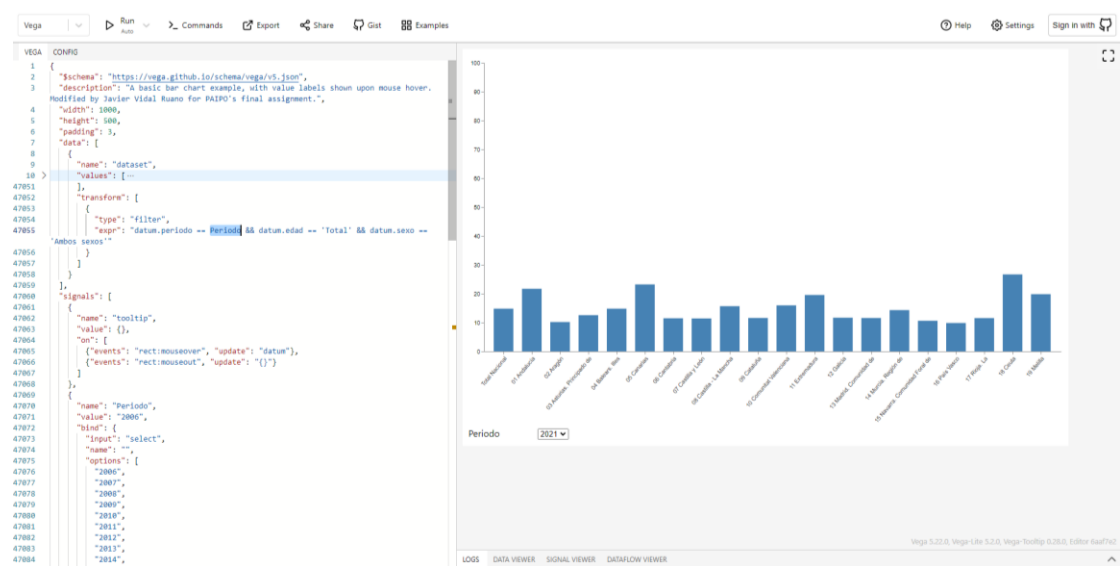
Antes de proceder, se ha establecido una escala fija para el eje de las Y, usando el intervalo [0-100], puesto que la información viene en porcentajes:



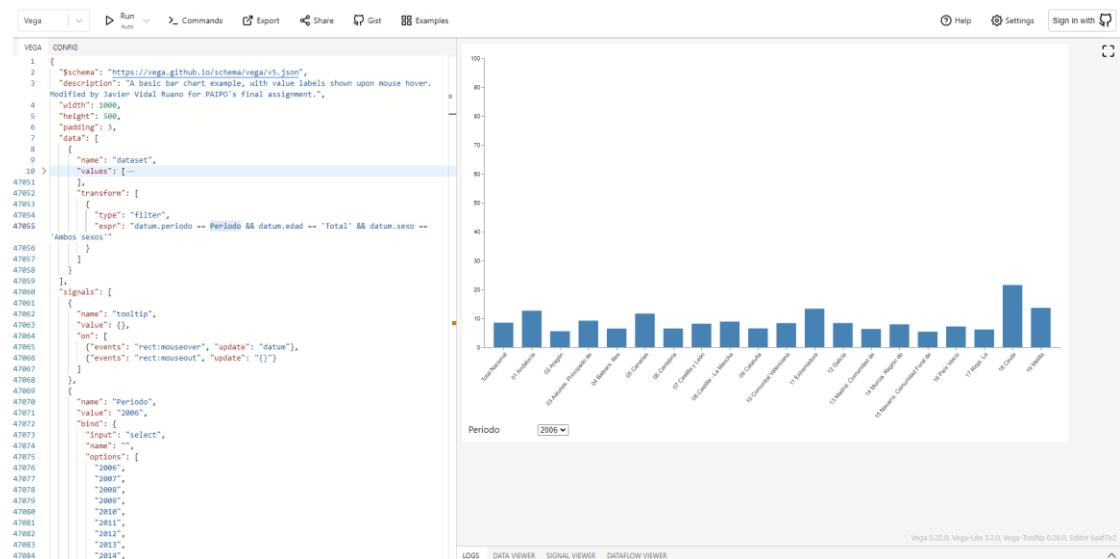
Ahora, se ha definido la señal para que el usuario pueda seleccionar el periodo (año) del cual quiere ver los datos en la gráfica, y como se puede ver, ha aparecido el selector debajo de la gráfica:

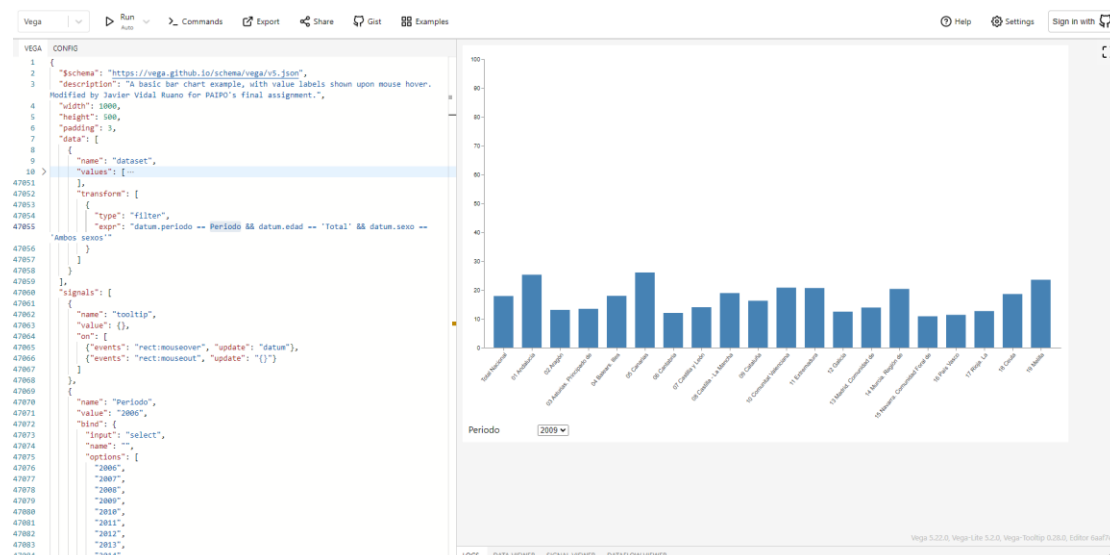
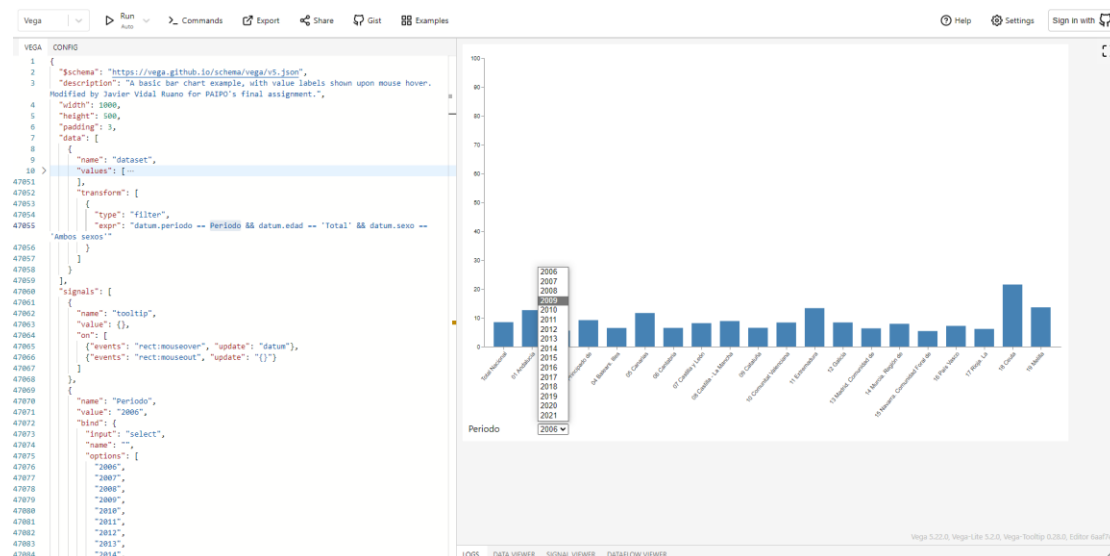


Para hacer que el selector de periodo actualice los datos de la gráfica, hay que cambiar la transformación definida previamente, y cambiar los valores escritos de forma fija por el nombre de la variable (la señal) definida previamente:

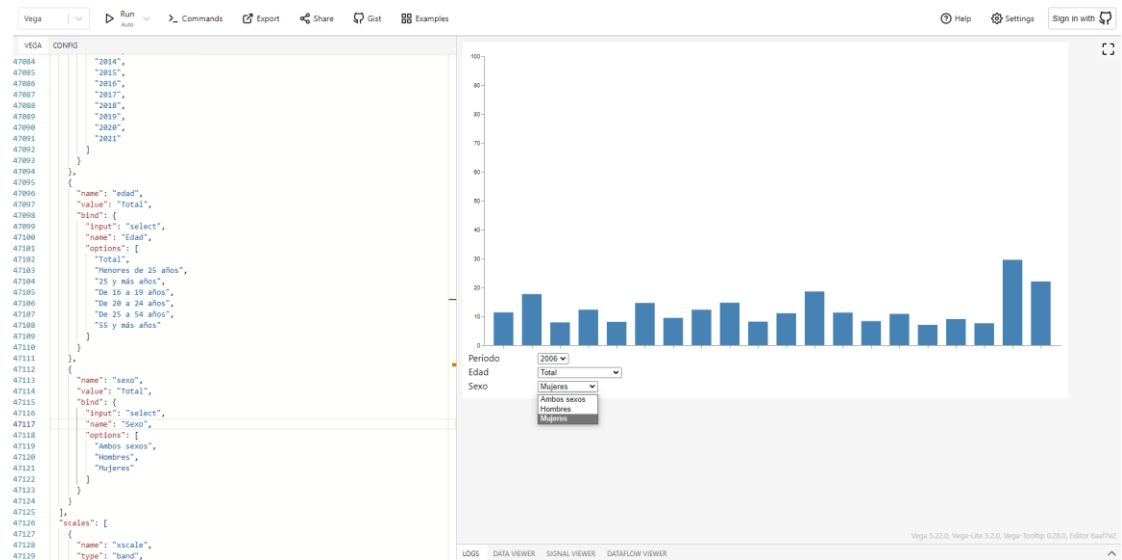
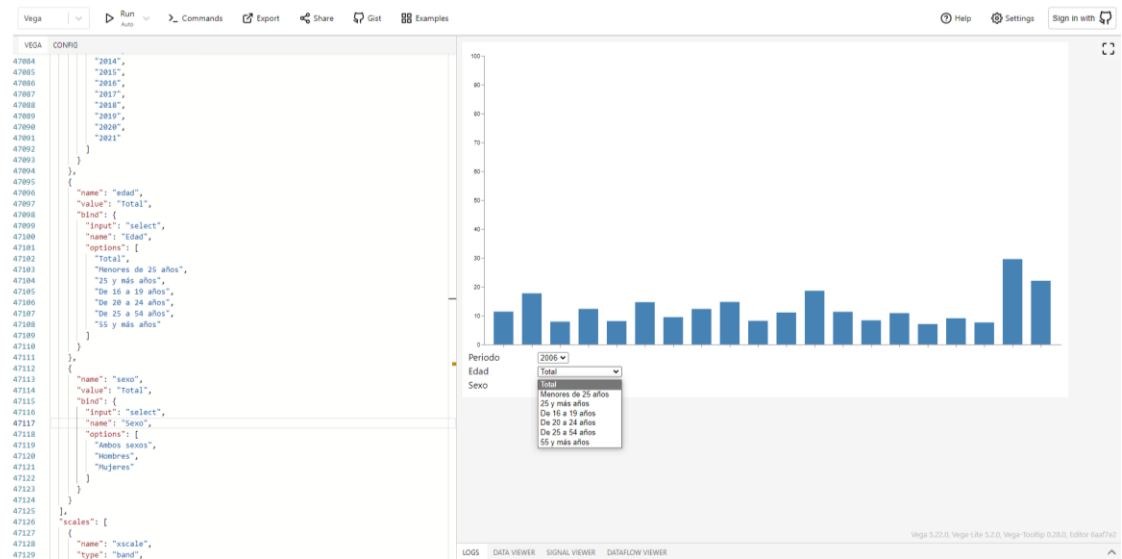
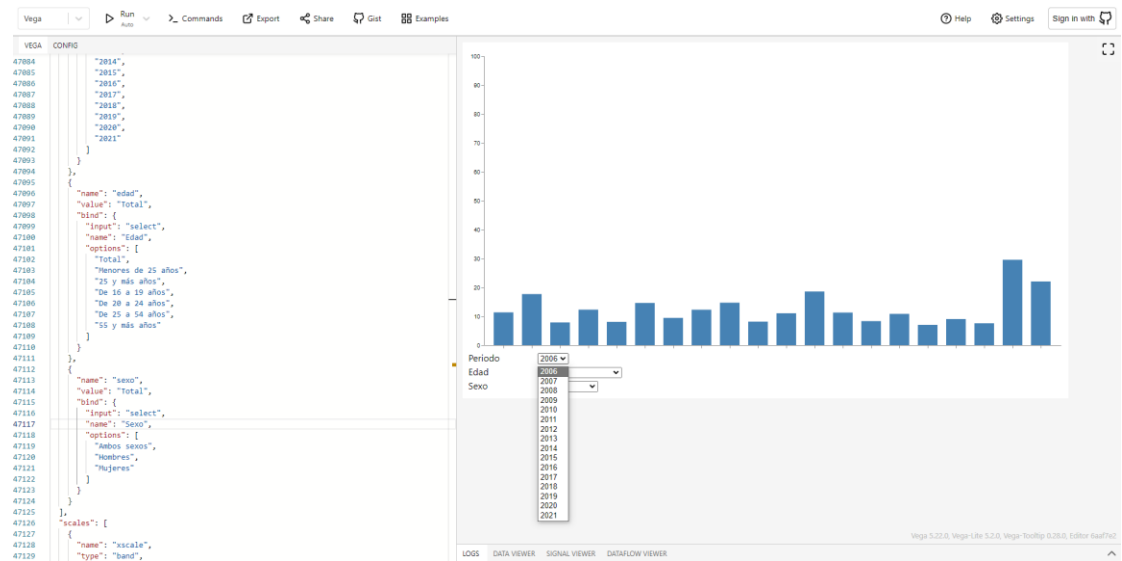


Como se puede ver en las siguientes imágenes, ahora el selector de año funciona correctamente:



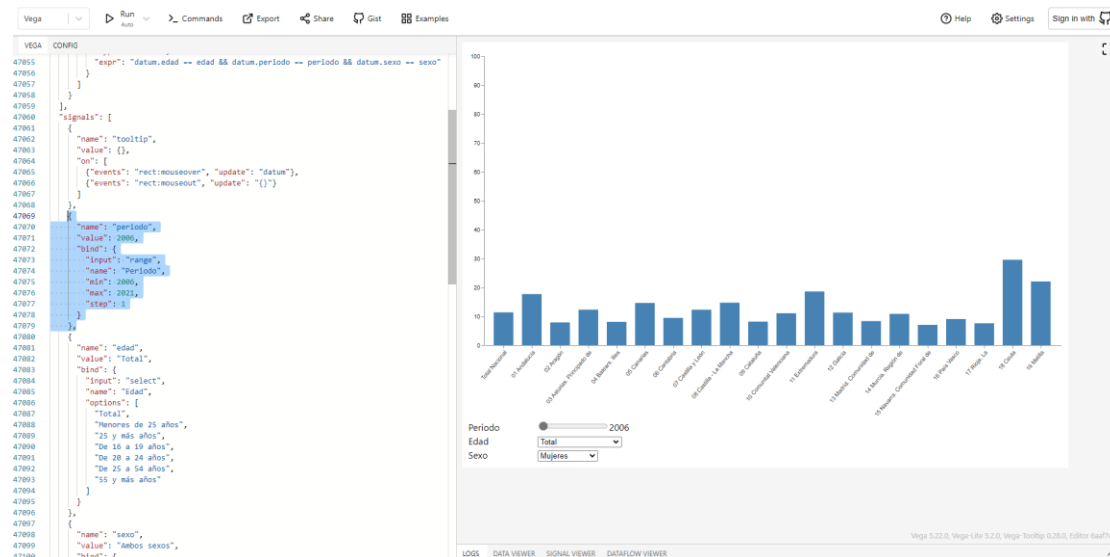


Se ha realizado el mismo procedimiento para el resto de variables (Edad, Sexo). De manera que ahora el usuario ya tiene la posibilidad de filtrar los datos por año, edad y sexo:

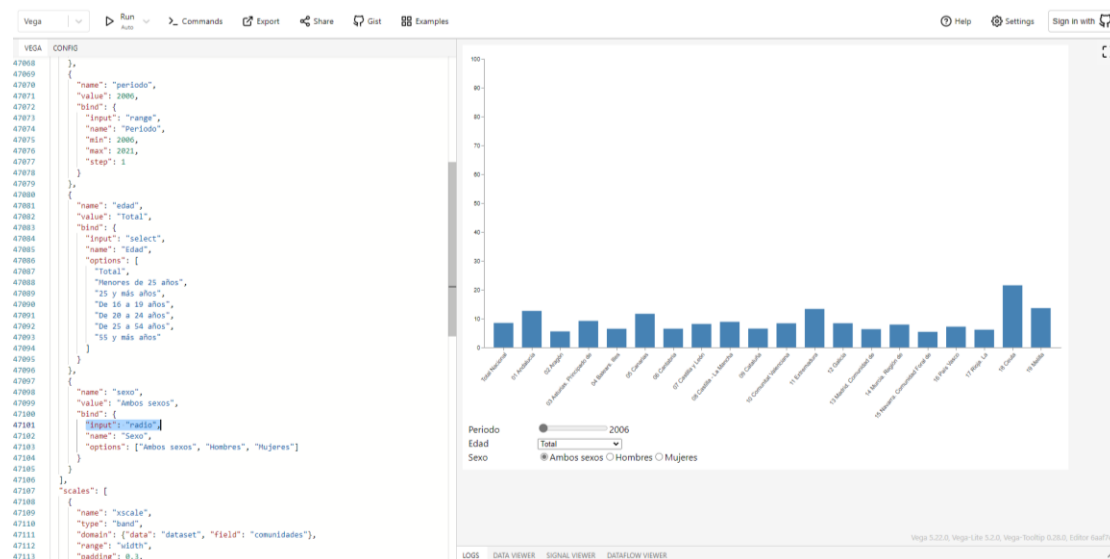




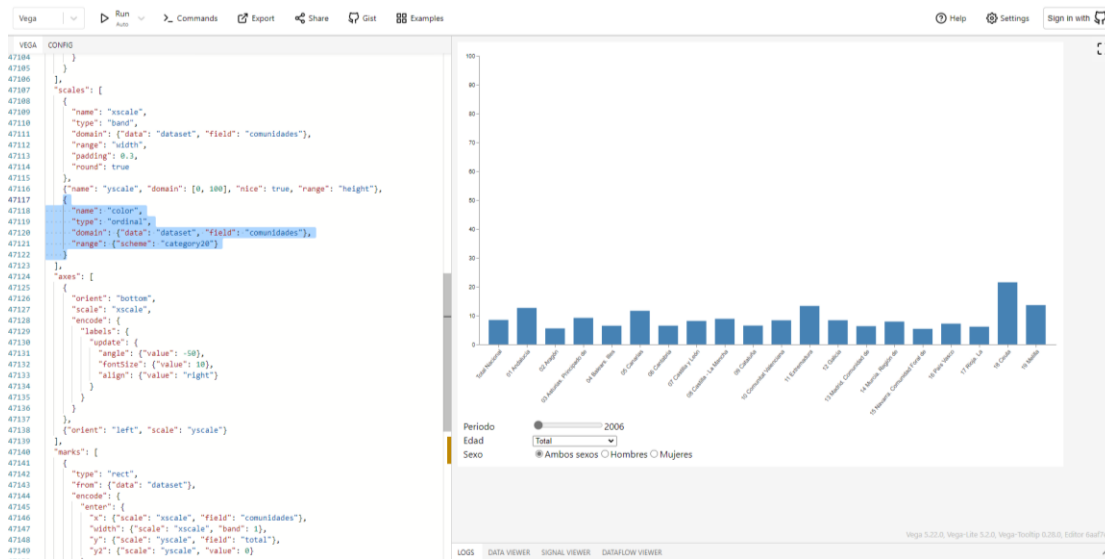
Se ha decidido hacer un pequeño cambio para mejorar la experiencia de usuario. Se ha sustituido el menú estilo dropdown para seleccionar el periodo por un slider (como se había pensado en un principio, en la idea inicial), de manera que ahora es mucho más sencillo interactuar con el gráfico, variando el periodo:



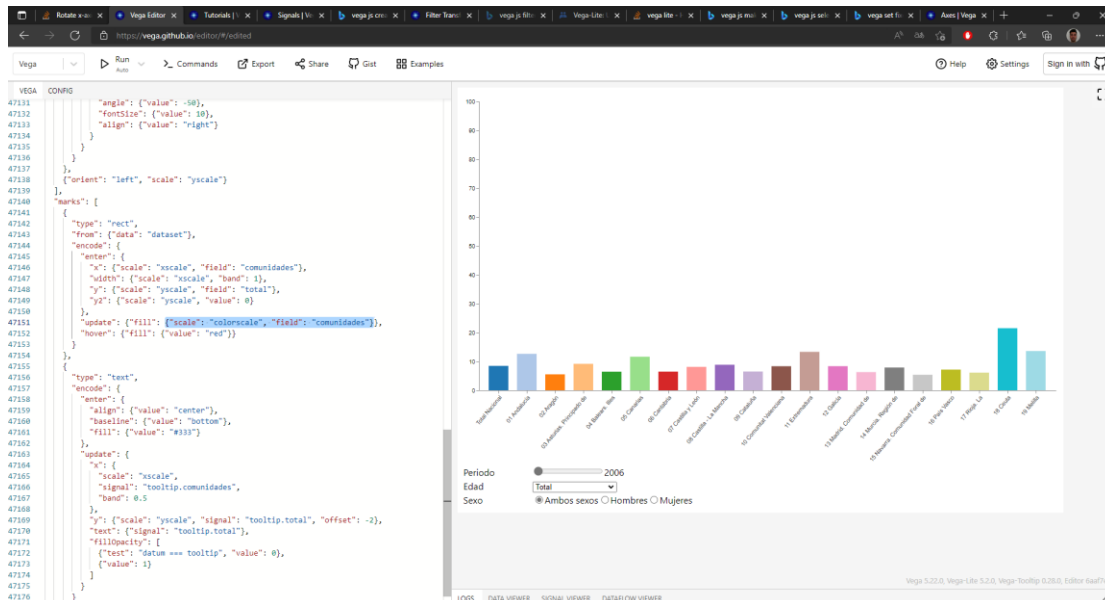
También se ha sustituido el menú para seleccionar el sexo por unos "radio buttons", ya que como tiene pocas opciones, se adapta perfectamente a ese tipo de selector:



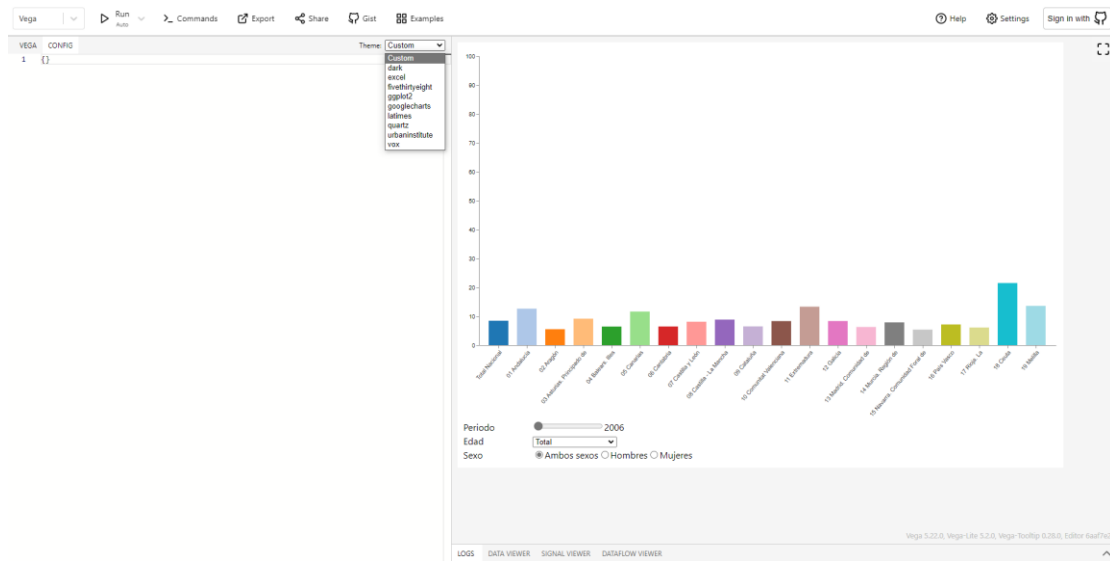
Se ha decidido cambiar los colores del gráfico, para poder distinguir mejor unas barras de las otras. Para ello, primero se crea una escala:



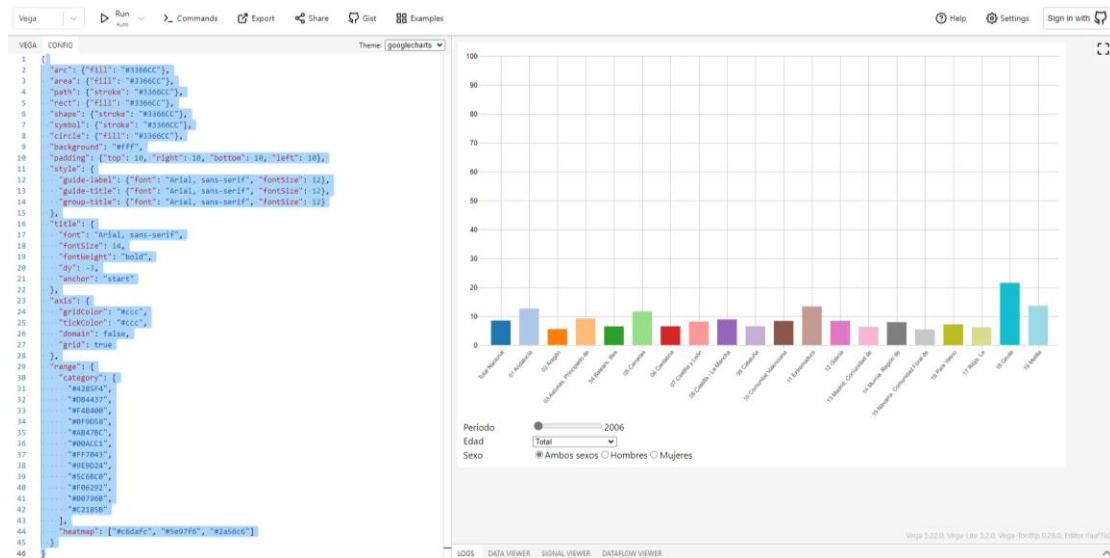
Y después, se asigna dicha escala a las barras del gráfico:



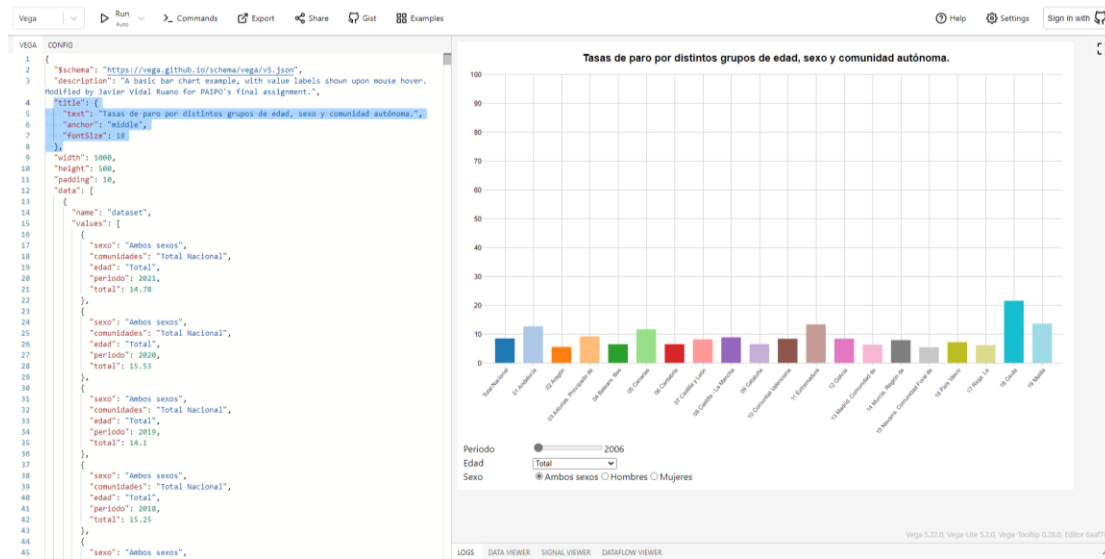
Por último, se ha probado las diferentes configuraciones que Vega ofrece, para cambiar el aspecto de la gráfica:



Se ha decidido usar el tema que copia el estilo de las gráficas de Google:



Se ha hecho una última modificación, y esta es añadir un título a la gráfica, como se puede ver a continuación:



Una vez acabado el desarrollo de la gráfica, se ha exportado como fichero HTML para poder incluirlo en mi página web, hosteada en GitHub Pages, y a la cual se puede acceder mediante el siguiente enlace:

<https://javiervidrua.github.io/infovis/>