Universidad de Valladolid

TRABAJO FIN DE GRADO

Grado en Estadística

Facultad de Ciencias

Diseño y utilización de una herramienta de visualización de datos a escala municipal

Tania Martín Fernández

Tutores:

Alfonso Gordaliza Ramos

Francisco Rodríguez Redondo

Agradecimientos

A mis tutores, Alfonso Gordaliza y Francisco Rodríguez, por toda la ayuda que me han proporcionado durante estos últimos meses. Agradezco enormemente toda vuestra dedicación e implicación en este trabajo. Doy las gracias también a Julio Pastor por su colaboración en este proyecto. Sin su ayuda no habría sido posible llevarlo a cabo.

A mi grupo de amigos, la Banda de la UVa, gracias por estos increíbles cinco años.

A mis padres y mi hermano, por todo el apoyo que me dan día a día. Gracias por estar siempre a mi lado y confiar en mí en todo momento. No ha sido un camino fácil, pero gracias a vosotros he logrado llegar hasta aquí.

Índice

Resumen

La tecnología digital actual permite tener acceso a una gran cantidad de información. Esto impulsa el desarrollo de numerosas herramientas que permiten acercar los datos a la ciudadanía. Con la motivación de despertar su interés por la demografía, es posible implementar recursos que les permitan conocer la situación demográfica del área geográfica en la que residen y les ayuden a comprender los fenómenos demográficos y los factores que afectan a los cambios en la distribución de la población por todo el territorio de España. Dada la importancia de la componente geográfica asociada a esta información, la combinación de los datos demográficos con los geográficos es clave para visualizar los fenómenos demográficos.

El propósito de este Trabajo Fin de Grado consiste en desarrollar una aplicación Shiny en R para ofrecer una herramienta dinámica con la que dar acceso a información de la última década sobre la distribución y evolución de la población española a diferentes niveles territoriales, con especial interes en la visualización de datos a escala municipal, por medio de mapas interactivos y de evolución.

Palabras clave: aplicación, mapas, R, Shiny, indicadores de población, visualización

Abstract

Current digital technologies allow access to a large amount of information. As a consequence, the development of more sophisticated tools allow a more accessible approach to data. With the intention of awakening the interest of citizens in demography, it is possible to implement resources which help them know demographic information about the geographic area in which they live, as well as understand the factors which cause changes in the distribution of the population throughout the territory of Spain. Given the importance of the geographic component associated with this information, the combination of demographic and geographic data is key to visualizing demographic phenomena.

The aim of this project is to develop a Shiny app using R programming language, in order to offer a dynamic tool which allows access to information about the distribution of the Spanish population at different territorial levels over the last ten years, putting special attention on the visualization of data at the municipal level, through interactive maps and graphs showing evolution.

Key words: app, maps, R, Shiny, population indicator, visualization

Introducción

El acceso a la información en la era digital está al alcance de la mayor parte de la ciudadanía. Los avances en las tecnologías de la información facilitan el desarrollo de nuevos y mejorados recursos que impulsan el avance del conocimiento en múltiples ámbitos. Gracias a estos progresos, la estadística cobra cada vez más importancia, debido a que la enorme cantidad de información disponible no tiene valor si no se le da una correcta interpretación y un buen procesamiento y uso. La combinación de la estadística y la computación asociada a esta rama de estudio permiten poner a disposición de los ciudadanos herramientas de visualización e interpretación de datos con las que fomentar su interés por la demografía. Es importante que los ciudadanos conozca la estructura demográfica de la población de la que forma parte, así como su evolución temporal, y estos recursos ponen a su disposición información estadística que describe dicha estructura para cada territorio y ayuda a comprender los cambios demográficos que sufren estas regiones.

Con el objetivo de proporcionar acceso de forma dinámica a información sobre la evolución y distribución de la población, el presente Trabajo Fin de Grado se centra en la implementación de una herramienta de visualización de datos a escala municipal, denominada **munimapp**.

El desarrollo de esta aplicación se ha llevado a cabo por medio del software estadístico R haciendo uso de Shiny, un paquete que combina el diseño web con el lenguaje de programación R para la creación de aplicaciones interactivas.

La desagregación de la información por municipios resulta interesante debido a que los cambios de ciertos fenómenos demográficos a niveles superiores pueden verse compensados y no ser observados pese a que pudieran existir variaciones internas importantes en ámbitos inferiores. Así, es posible visualizar fenómenos como la despoblación del medio rural, que además lleva consigo la desigualdad de género de estas zonas, debido al movimiento de la población a las grandes ciudades, que no sería apreciable si se observasen los cambios a nivel provincial o autonómico.

La aplicación trabaja con datos poblacionales desde 2011 hasta 2020. Dentro de ella, esta información se articula en torno a un conjunto de mapas interactivos y gráficos contenidos en ellos. Con estas representaciones se pretende ofrecer información útil acerca de la estructura y la composición de la población del país a diferentes niveles territoriales: por comunidades autónomas, provincias y municipios. Para poder apreciar la evolución

de las cifras y los indicadores que describen la distribución de la población, **munimapp** también cuenta con mapas de evolución a nivel municipal.

La memoria de este Trabajo Fin de Grado se ha dividido en cuatro capítulos:

- En el **primer capítulo** se explica el proceso de generación de la **base de datos** de la aplicación, que implica reunir, depurar, adecuar y organizar toda la información que posteriormente se representará. La datos deben ser independientes del proceso de creación de los mapas para que, en caso de tener que aplicar cambios sobre las representaciones, no sea necesario modificarlos.

- El **segundo capítulo** se centra en describir cómo se sintetiza el contenido de la base de datos para obtener los **mapas**. La información geográfica, que puede tener asociada atributos que describan la ubicación a la que referencia, se conoce como datos espaciales. Esta estructura de datos permite unir la información cartográfica con las cifras o los indicadores de población a representar. Además, en esta sección se hará un inventario de los mapas que se visualizan en la aplicación.

- Para comprender la herramienta con la que se ha creado **munimapp**, en el **tercer capítulo** se ofrece una guía de **Shiny**, en la que se describen las partes que componen una aplicación y el paradigma de programación que rige su comportamiento. Además, incluye una explicación de los recursos que ofrece Shiny para modificar el diseño de cualquier aplicación.

- Por último, el **cuarto capítulo** presenta la aplicación. En esta sección se explica la estructura de **munimapp** y el contenido de cada uno de los cuatro apartados que la componen. Esto permitirá situar los mapas descritos en el segundo capítulo dentro de su contexto. También se describen las características del servidor Shiny que aloja la aplicación, gracias al cual es posible poder compartirla.

El estilo que se ha buscado en la redacción pretende que este proyecto pueda servir a modo de guía para otros estudiantes que quieran iniciarse tanto en la visualización de datos espaciales como en el manejo de Shiny.

Para acceder a **munimapp**, visitar el enlace que se muestra a continuación:
http://shiny1.eio.uva.es:3838/users/tania/munimapp/

1. Base de Datos

Los datos son el pilar fundamental de la aplicación que se desarrolla en este proyecto porque ésta se centra en proporcionar el entorno en el que se visualizarán. No es suficiente con recopilar la información con la que se va a trabajar, también es necesario organizarla y almacenarla sistemáticamente para su posterior uso. Con una base de datos correctamente estructurada se reduce el espacio de almacenamiento de los datos y se consigue mayor flexibilidad y rapidez a la hora de acceder a ellos. Además, reunir y almacenar la información adecuadamente no solo da coherencia a los resultados, sino que mejora el rendimiento y permite evitar la repetición de multitud de operaciones en diferentes procedimientos.

En este capítulo se describen las tres fases que se han desarrollado para a obtener la base de datos utilizada en este proyecto. La primera consiste en reunir la información necesaria, analizar su contenido y estudiar su estructura. El hecho de que los datos procedan de varias fuentes implica la necesidad de someterlos a un proceso de acondicionamiento, armonización y depuración, como parte de la segunda fase, para así asegurar que la información esté estandarizada, no contenga errores y no sea redundante. En esta etapa se inicia el proceso de manipulación de los datos y, por ello, se incluye en este apartado la descripción de los paquetes de R con los que se trabaja. La tercera fase se centra en estructurar y almacenar la información para poder hacer uso de ella de la forma más eficiente posible.

1.1. Fuentes de Información y Datos

Los datos recopilados para este proyecto se dividen en dos grandes bloques. Por un lado, los **datos cartográficos**, que contienen la digitalización georreferenciada de los territorios con la que se generan los mapas, y, por otra parte, la información a representar sobre ellos, compuesta por los datos de la **Estadística del Padrón Continuo** y una serie de **indicadores de población** obtenidos a partir de estas cifras poblacionales.

A continuación, se detallan los diferentes conjuntos de datos de los que se ha extraído la información necesaria para generar la base de datos y se explican los pasos a seguir para su descarga u obtención, incluyendo la metodología para el cálculo de los indicadores de población.

1.1.1. Datos Cartográficos

Los **datos cartográficos** contienen la información geográfica necesaria para representar el territorio y sus divisiones en función del nivel geográfico. Las geometrías asociadas a los recintos municipales y los límites entre provincias y comunidades autónomas proceden de dos fuentes distintas. Los primeros se han tomado de la **Cartografía digitalizada de secciones censales** del Instituto Nacional de Estadística [1.1], mientras que los segundos pertenecen a la colección de datos del Centro de Descargas del Centro Nacional de Información Geográfica, en adelante CNIG, concretamente de los **Límites municipales, provinciales y autonómicos** [1.2].

Pese a que el segundo conjunto de datos contiene la geometría municipal, la estructura territorial a este nivel es variable, ya que se crean y suprimen una pequeña cantidad de municipios casi todos los años. Por este motivo es necesario disponer de la información cartográfica de cada municipio para cada año. Además, la cartografía digitalizada del INE proporciona la relación entre territorios y sus códigos, de gran utilidad para trabajar con estos datos.

Por otro lado, la división por secciones censales no cubre todo el territorio nacional debido a que hay zonas que no pertenecen a municipios concretos. Como se indica en el Estudio sobre el Registro de Entidades Locales [1.3], existen más de 100 territorios, la mayoría de ellos en las provincias de Burgos y de Navarra, que no pertenecen a ningún término municipal y que son administrados proindiviso por dos o más ayuntamientos. Por ello, no es posible obtener los contornos completos de las comunidades autónomas y las provincias a partir de la cartografía del INE, siendo necesario utilizar los datos del CNIG para hacer las representaciones a estos niveles geográficos.

Cartografía digitalizada de secciones censales

El Instituto Nacional de Estadística facilita los contornos georreferenciados de todas las secciones censales a nivel nacional, según las coordenadas UTM[1], huso 28, 29, 30 y 31. Los ficheros están disponibles de forma anual desde 2011 en formato **shapefile**.

[1] El sistema **UTM** (*Universal Transverse Mercator*) se define según [1.4] como un sistema de proyección cartográfico basado en cuadrículas con el cual se pueden referenciar puntos sobre la superficie terrestre. Está basado en un modelo elipsoidal de la Tierra, siendo la proyección UTM un sistema cilíndrico que es tangente al elipsoide en un meridiano origen. Para evitar que las deformaciones producidas en la proyección sean demasiado grandes, se divide el elipsoide terrestre en 60 husos de 6^{o} de amplitud. El trazado de las cuadrículas se realiza en base a estos husos y a zonas UTM.

En la Figura 1.1 se indican los pasos a seguir para descargar los ficheros de cartografía digitalizada de secciones censales del INE y en la Figura 1.2 se describe el contenido de uno de ellos como ejemplo explicativo sobre un archivo con formato **shapefile**.

Figura 1.1. Pasos a seguir para descargar los datos de **cartografía digitalizados de secciones censales** del INE.

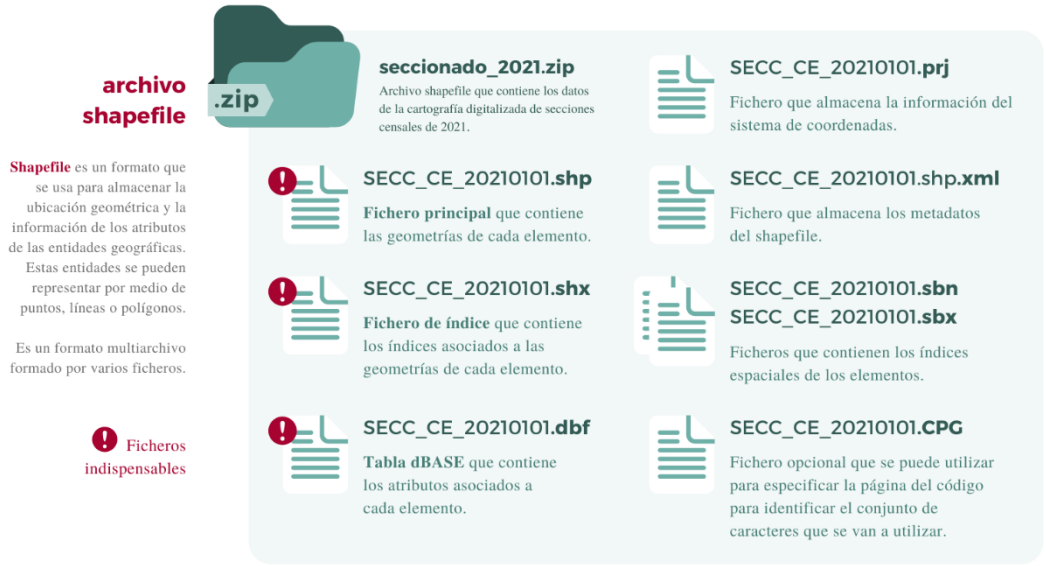


Figura 1.2. Descripción de los ficheros que componen un archivo con formato **shapefile** mediante un ejemplo tomando los datos de la cartografía digitalizada de secciones censales del INE del año 2021. Fuente de información de las definiciones: [1.5].

El formato **shapefile** [1.6] es un formato para almacenar elementos geográficos y atributos asociados a ellos desarrollado por la compañía ESRI, *Environmental Systems Research Institute*, que se ha convertido en un formato estándar de facto en cuanto a información georreferenciada. Un **shapefile** está formado por al menos tres archivos indispensables. El *fichero principal* (con extensión **.shp**) contiene las geometrías. Los índices asociados a cada una se encuentran en el *archivo de índice* (**.shx**), mientras que los atributos de cada elemento se encuentran en la *tabla dBASE* (**.dbf**).

Los conjuntos de datos de la cartografía digitalizada del INE están formados entonces por las **geometrías**, que en este caso son los contornos georreferenciados de las secciones censales, y los **argumentos** asociados a ellas, los cuales ofrecen una tabla de referencia entre los códigos y los nombres asociados a cada región a distintos niveles geográficos. Los argumentos, o variables de clasificación, más relevantes son los siguientes:

- **CCA**: Código de dos dígitos que referencia a cada comunidad/ciudad autónoma.
- **NCA**: Nombre de cada comunidad/ciudad autónoma.
- **CPRO**: Código de dos dígitos que referencia a cada provincia.
- **NPRO**: Nombre de cada provincia.
- **CUMUN**: Código que designa a cada municipio, compuesto por cinco dígitos: los dos primeros hacen referencia al código de la provincia (CPRO) y los otros tres restantes al del municipio dentro de ésta (CMUN).
- **NMUN**: Nombre de cada municipio.
- **CUDIS**: Código de distinción de siete dígitos, compuesto por el código del municipio (CUMUN) y el código de control (CDIS), asignado por medio de una regla de cálculo, que permite la detección de errores de grabación.
- **CUSEC**: Código de sección censal compuesto por el código de municipio (CUMUN), el código de control (CDIS) y el código de identificación de cada sección (CSEC).

En el informe metodológico estandarizado **Relación de Municipios y sus Códigos por Provincias** [1.7] se define a este conjunto de variables, dadas por los argumentos del conjunto de datos, como las denominaciones de todas las regiones que figuran en el Registro de Entidades Locales (REL) del Ministerio de Hacienda y Administraciones Públicas. En dicho registro, creado mediante el Real Decreto 382/1986, de 10 de febrero,

siguiendo las disposiciones establecidas en la Ley 7/1985, de 2 de abril, Reguladora de las Bases del Régimen Local, deben inscribirse todas las entidades locales, entendiendo por tales a los municipios, provincias, islas, entidades de ámbito territorial inferior al municipio y mancomunidades de municipios. Con esta información se relacionan con sus códigos todas las comunidades autónomas, provincias, islas y los municipios del territorio nacional registrados en el REL, incluidas las Ciudades con estatuto de autonomía de Ceuta y Melilla.

Además, los datos de la cartografía digitalizada del INE incluyen la superficie de cada sección censal en metros cuadrados[2], información necesaria para el cálculo de la *densidad de población* (ver 1.1.4).

Límites municipales, provinciales y autonómicos

El Centro Nacional de Información Geográfica, **CNIG**, ofrece los datos topográficos de los contornos provinciales y autonómicos. En la Figura 1.3 se indican los pasos a seguir para descargar el archivo que contiene los ficheros **shapefile**.

Figura 1.3. Pasos a seguir para descargar los datos de **Límites municipales, provinciales y autonómicos** del Centro de Descargas del Centro Nacional de Información Geográfica.

[2] La **superficie** de las secciones censales de la cartografía digitalizada del INE, variable que en los conjuntos de datos se denomina ***Shape__Area*** o ***Shape__area***, se calcula mediante la herramienta *Calculate Geometry* de **ArcMap**, componente principal del conjunto de programas de procesamiento geoespacial **ArcGIS** de **ESRI**. Esta **superficie** se corresponde con la superficie del polígono de cada geometría y podría no coincidir con exactitud con las superficies reales.

La propia descripción de los datos dada por el Centro de Descargas del CNIG [1.2] indica que las geometrías de este archivo se corresponden con la interpretación de los títulos jurídicos inscritos en el Registro Central de Cartografía, RCC, y que se incluye exclusivamente la información cartográfica de las delimitaciones territoriales que constan en él (Real Decreto 1545/2007l).

En la Figura 1.4 se describe el contenido de la carpeta obtenida de la descarga según la Figura 1.3, pero solo se hace referencia a los archivos asociada los recintos provinciales y autonómicos debido a que de este conjunto de datos solo es relevante dicha información. Tener en cuenta que los contornos están separados en varios ficheros según el ámbito geográfico y además se dividen entre Islas Canarias y Península con Baleares.

Figura 1.4. Descripción de la carpeta que contiene los recintos provinciales y autonómicos, tomada del Centro de Descarga del CNIG. Se han omitido los ficheros irrelevantes.

A diferencia de la cartografía digitalizada del INE, en este caso no se hace mención de la información dada por los argumentos de los contornos de estos conjunto de datos porque solo son relevantes las geometrías que proporcionan estos ficheros.

1.1.2. **Estadística del Padrón Continuo**

El **Padrón Municipal** [1.8] es un registro administrativo donde constan las personas empadronadas en cada municipio y una serie de datos sociodemográficos asociados a ellas. Los Ayuntamientos son los encargados de su gestión y mantenimiento, y también son los responsables de realizar las actualizaciones y operaciones oportunas para que las cifras concuerden en todo momento con la realidad, de manera prácticamente continua. Los Ayuntamientos deben remitir al INE, responsable de la coordinación de los padrones municipales, las variaciones mensuales que se producen en los datos.

Se conoce como **Padrón Continuo del INE** a la base de datos formada por todos los ficheros padronales municipales. Las cifras poblacionales, provenientes de la Revisión del Padrón a 1 de enero de cada año, contienen la información básica de la población clasificada según las variables: edad, sexo, nacionalidad, lugar de nacimiento y lugar de residencia. Los datos de la última categoría se facilitan para distintos ámbitos geográficos: nacional, comunidades/ciudades autónomas, provincias, municipios y secciones censales.

La **Estadística del Padrón Continuo**, concretamente las cifras de **Población por sexo, municipios y edad (grupos quinquenales)**, obtenidas de los *resultados detallados por* ***municipios*** [1.9], es el conjunto de datos del Padrón Continuo con el que se ha trabajado en este proyecto. La Figura 1.5. ilustra los pasos a seguir para su descarga. Para este proyecto se ha seleccionado la información correspondiente a la Serie 2013-2020 y los años 2011 y 2012, coincidiendo con los periodos disponibles para la cartografía digitalizada de secciones censales del INE.

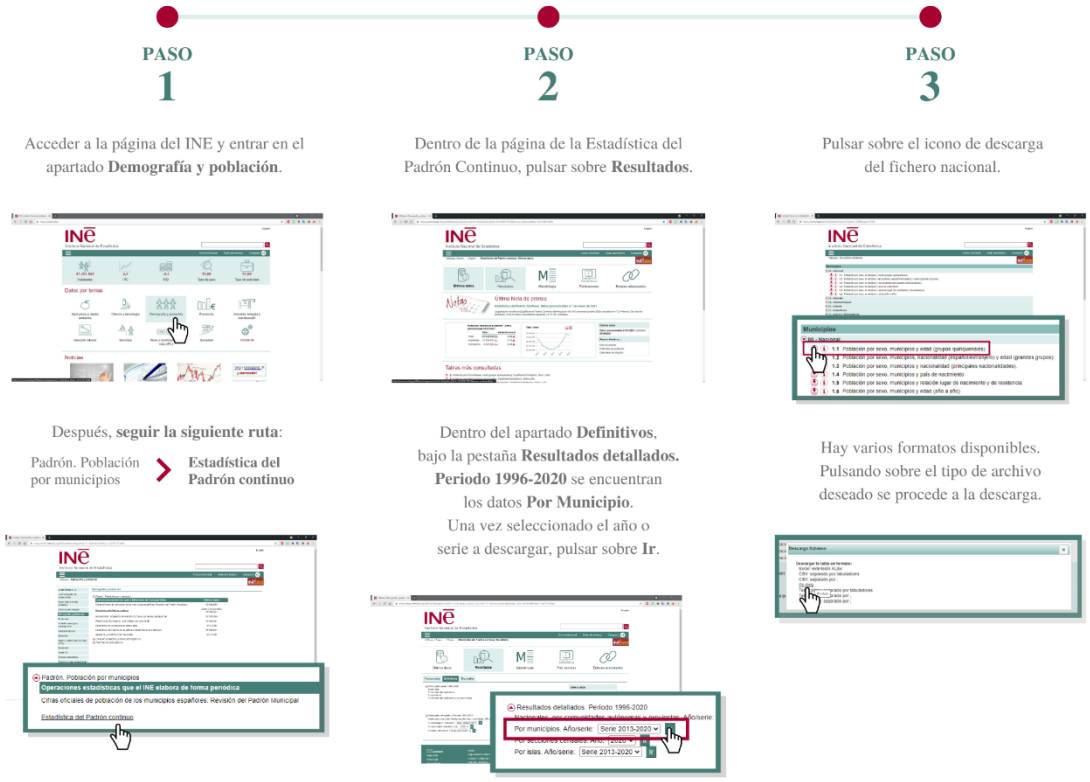


Figura 1.5. Pasos a seguir para descargar los datos de la **Estadística del Padrón Continuo** del INE: **Población por sexo, municipios y edad (grupos quinquenales)**.

Las variables de clasificación de los datos de la Serie 2013-2020 aparecen detalladas a continuación. De los ficheros asociados a 2011 y 2012 se hablará posteriormente (ver 1.2).

- **Edad**: Las cifras de población están agrupadas en 21 grupos de edad quinquenales, es decir, intervalos de cinco años, con los extremos en los múltiplos de 5: (0 a 5), (5 a 10), etc. También se incluyen la cifras totales como *Todas las edades*.
- **Periodo**: Indica el año al que pertenecen los datos (*1 de enero de* _).
- **Municipios**: Esta variable está compuesta por el código del municipio y su nombre. El código de municipio se compone de cinco dígitos: los dos primeros hacen referencia al código de la provincia y los tres restantes al del municipio dentro de ésta. También se incluyen las cifras del *Total Nacional*.
- **Sexo**: Se refiere al sexo biológico de la persona: *Hombres* y *Mujeres*. También se incluyen las cifras de ambos como *Total*.

1.1.3. Extensión superficial de las comunidades y provincias

La cartografía digitalizada de secciones censales del INE incluye las superficies de cada territorio. Como se indicó en el subapartado asociado a ese conjunto de datos, esta información es necesaria para el cálculo de la *densidad de población* (ver 1.1.4). Aunque es posible usar esa información para calcular la superficie a nivel provincial y autonómico, se ha creído más conveniente emplear las cifras oficiales registradas por el Instituto Nacional de Estadística en el Anuario Estadístico de España de 2008, en el que consta la **Extensión Superficial de Comunidades Autónomas y Provincias** [1.10].

1.1.4. Indicadores Demográficos de Población

Los **indicadores demográficos** permiten describir las características estructurales de la población, así como analizar los procesos que influyen en su formación, evolución y desaparición. Los **indicadores de población**, concretamente, proporcionan información sobre el tamaño, la distribución y la estructura de la población, ofreciendo una visión de la situación y evolución de diferentes factores que influyen de una forma significativa en muchos de los fenómenos sociales con los que convivimos.

La información metodológica para el cálculo de los indicadores de población se ha extraído del informe **Indicadores demográficos** del Instituto Gallego de Estadística [1.11]. A continuación se detalla lo que mide cada uno y sus fórmulas de cálculo.

Porcentaje de Población según Ámbito Geográfico

Indica el porcentaje de población de un territorio sobre el total de la población del ámbito geográfico superior.

$$Pob^t_{Nac-CA} = \frac{Pob^t_{CA}}{Pob^t_{Esp}} \times 100 \tag{1.1}$$

$$Pob^t_{CA-Prov} = \frac{Pob^t_{Prov}}{Pob^t_{CA}} \times 100 \tag{1.2}$$

$$Pob^t_{Prov-Mun} = \frac{Pob^t_{Mun}}{Pob^t_{Prov}} \times 100 \tag{1.3}$$

Pob^t_{Amb} es el total de población a 1 de enero del año t en cada ámbito geográfico, Amb, de nivel superior a inferior: Nac (nacional, España), CA (comunidad autónoma), $Prov$ (provincia) y Mun (municipio).

Porcentaje de Población por Grandes Grupos de Edad

Indica el porcentaje de población de un grupo de edad con respecto al total de la población del territorio. Los grupos de edad son: menores de 20 años (< 20), de 20 a 64 ($20 - 64$) y mayores de 64 años (> 64).

$$I^t_{<20} = \frac{Pob^t_{<20}}{Pob^t} \times 100 \tag{1.4}$$

$$I^t_{20-64} = \frac{Pob^t_{20-64}}{Pob^t} \times 100 \tag{1.5}$$

$$I^t_{>64} = \frac{Pob^t_{>64}}{Pob^t} \times 100 \tag{1.6}$$

Pob^t_x es el total de población en el grupo de edad x a 1 de enero del año t, y Pob^t es el total de población a 1 de enero del año t.

Densidad de Población

Número de habitantes por cada kilómetro cuadrado de superficie del territorio.

$$D^t = \frac{Pob^t}{S} \tag{1.7}$$

Pob^t es el total de población a 1 de enero del año t y S es la superficie del territorio considerado, medida en kilómetros cuadrados.

Índice de Envejecimiento

Indica la relación entre la población mayor de 64 años y la de menos de 20 años.

$$IE^t = \frac{Pob^t_{>64}}{Pob^t_{<20}} \times 100 \tag{1.8}$$

Pob^t_x es el total de población en el grupo de edad x a 1 de enero del año t.

Índice de Masculinidad

Indica el número de hombres por cada 100 mujeres en un territorio.

$$IM^t = \frac{Pob^t_{hom}}{Pob^t_{muj}} \times 100 \tag{1.9}$$

Pob^t_{hom} es el total de hombres de la población a 1 de enero del año t y Pob^t_{muj} es el total de mujeres de la población a 1 de enero del año t.

1.2. Manejo, Depuración y Adecuación de los Datos

La Figura 1.6 muestra un esquema del conjunto de ficheros descargados: por un lado, los **datos cartográficos** procedentes de dos fuentes, CNIG e INE, y por el otro los datos del **Padrón Continuo**, obtenidos de la Estadística del Padrón Continuo del INE.

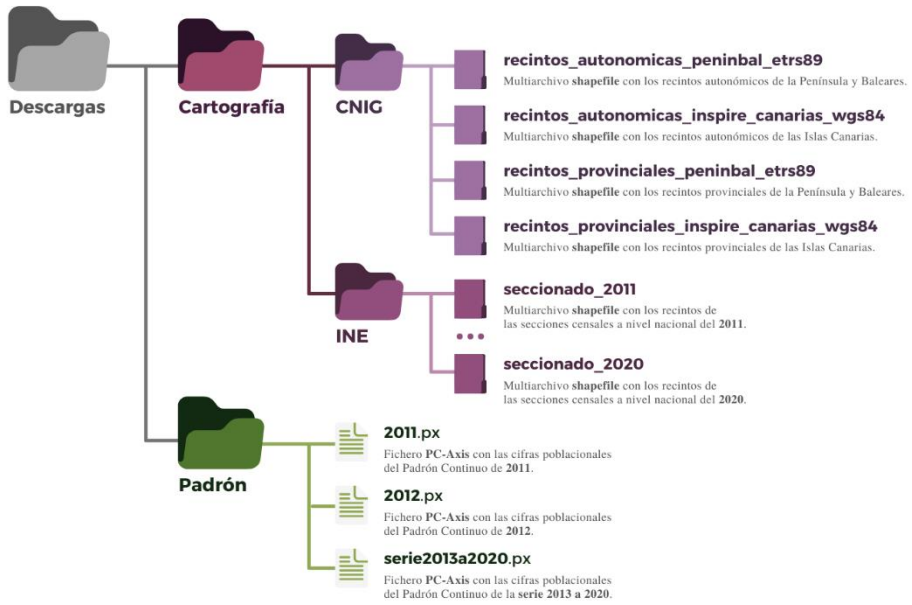


Figura 1.6. Esquema de la carpeta **Descargas** que contiene los archivos relevantes para generar la base de datos.

Respecto al primer bloque de datos, de la carpeta de la Figura 1.4, obtenida del Centro de Descargas del CNIG (ver 1.1.1), se seleccionan solo los ficheros asociados a los recintos provinciales y autonómicos, divididos en Península con Baleares e Islas Canarias. La información topográfica de las secciones censales del INE (ver 1.1.1) está compuesta por los archivos desde 2011 a 2020, cuya estructura sigue el patrón descrito previamente en la Figura 1.2. En cuanto a las cifras poblacionales del Padrón Continuo (ver 1.1.2), se encuentran en tres ficheros con formato PC-Axis.

Se ha podido constatar en este trabajo que existen muchas diferencias entre los datos cartográficos procedentes de cada fuente. La información asociada a los recintos provinciales y autonómicos del CNIG, además de presentar el inconveniente de estar dividida en dos carpetas por cada nivel geográfico, no se asemeja en absoluto en cuanto a estructura a los datos cartográficos de secciones censales del INE. Además, la semántica entre las denominaciones de los territorios de ambos conjuntos de datos no concuerda. Por otro lado, pese a que los tres ficheros que contienen las cifras de población se han extraído del INE, los niveles de los factores de las variables de clasificación son diferentes entre los datos agrupados y la información de 2011 y 2012. Para reducir el número de operaciones al acceder a los datos de cada año es conveniente dividir las cifras de la serie 2013-2020 según la variable adicional que indica el periodo de referencia, registro que no contemplan los datos de los años 2011 y 2012.

Pero antes de adecuar la información, para posteriormente organizarla y generar la base de datos, es necesario depurarla, con el propósito de detectar y corregir posibles errores en las observaciones y así asegurar su calidad y fiabilidad. Esto implica manipular los datos, lo que supone conocer las herramientas necesarias para operar sobre ellos.

1.2.1. **Paquetes para el Manejo de los Datos**

Paquete **sf**

Previamente se explicó la estructura del el tipo de archivo **shapefile** mediante la Figura 1.2, pero no se mencionó el formato de datos que contenían estos ficheros. Aunque la descripción detallada se ofrece en el siguiente capítulo (ver 2.1), a continuación se hace una introducción a los **datos espaciales** para hablar del paquete **sf** [1.12] y los objetos **Simple Features** [1.13].

Los **datos espaciales o georreferenciados** son diferentes al resto de estructuras de almacenamiento de datos de R porque tienen una característica especial que hace que

sean especiales: las **geometrías**. Como define Jean-François Mas [1.14], ***Simple Features*** es un formato estándar desarrollado por el Open Geospatial Consortium, OGC, para la representación de información geográfica. Es un modelo de datos jerárquico que simplifica los datos topográficos al condensar un amplio rango de formas geográficas en una única clase de geometría. Por el momento, para simplificar la descripción de este tipo de datos, se toma la definición de la estructura dada por este autor [1.14], quien explica que los objetos *simple features* se almacenan en un *dataframe*, en el cual los datos geográficos ocupan una columna especial que contiene una lista, generalmente denominada "geom" o "geometry".

Para descargar la versión del de CRAN basta con ejecutar la siguiente línea de código:

```
install.packages("sf")
```

Para descargar la versión de GitHub, usar en su lugar el siguiente fragmento de código:

```
library("devtools")
install_github("r-spatial/sf")
```

Una vez instalada, ejecutar la siguiente línea de código para cargar la librería:

```
library(sf)
```

Este paquete permite importar y exportar archivos de tipo **shapefile**. Para cargar, por ejemplo, la cartografía digitalizada del INE del año 2020, se ejecutaría la línea de código asociada a la función de lectura **st_read()**.

```
cartografia <- st_read("descargas/cartografía/INE/
                        seccionado_2020/SECC_CE_20200101.shp")
```

```
Reading layer 'SECC_CE_20200101' from data source 'descargas\
cartografia\INE\seccionado_2020\SECC_CE_20200101.shp'
using driver 'ESRI Shapefile'
Simple feature collection with 36309 features and 20 fields
geometry type:  MULTIPOLYGON
dimension:      XY
bbox:           xmin: -1004502 ymin: 3132130 xmax: 1126932
                ymax: 4859240
projected CRS:  ETRS89 / UTM zone 30N
```

Para guardar el objeto *sf* obtenido se usaría la función de escritura **sf_write()**:

```
st_write(cartografia,
         "base_de_datos/datos/SECC_CE_20200101.shp",
         delete_layer = TRUE)
```

```
Deleting layer 'cartografia2020' using driver 'ESRI Shapefile'
Writing layer 'cartografia2020' to data source 'base_de_datos/
datos/cartografia2020.shp' using driver `ESRI Shapefile'
Writing 36309 features with 20 fields and geometry type
Multi Polygon.
```

Por otro lado, es de gran relevancia la función **st_transform()**, con la que se pueden convertir las coordenadas de las geometrías de los objetos *sf*. En el fragmento de código que aparece a continuación se indica el tipo de transformación que se aplicará en los diferentes conjuntos cartográficos. En el siguiente capítulo se indicará el motivo de este cambio y el significado de la gramática con la que se especifica el sistema de coordenadas a aplicar (ver 2.1).

```
cartografia <- st_transform(cartografia,
                            "+proj=longlat +datum=WGS84")
```

Además, el paquete **sf** facilita la representación de los objetos *simple features* usando la función **plot()**. Un ejemplo de ello es la Figura 1.7, que se obtiene como resultado del ejemplo que se verá a continuación en la descripción del paquete **dplyr**.

Paquete **dplyr**

El paquete **dplyr** [1.15] ofrece una gramática especial para la manipulación de datos.

Se puede instalar desde CRAN o GitHub:

```
install.packages("dplyr")
```

```
library("devtools")
install_github("hadley/dplyr")
```

Posteriormente, para cargar la librería, se debe ejecutar la siguiente línea de código:

```
library(dplyr)
```

Los principales verbos (funciones) de este paquete son:

- **select()**: Selección de columnas en función de sus nombres.
- **filter()**: Selección de filas en función del valor dado por una condición.
- **arrange()**: Cambiar el orden de las filas.
- **rename()**: Cambiar el nombre de las variables.
- **mutate()**: Añadir nuevas variables o transforma variables existentes.
- **summarise()**: Resumen estadístico de las variables.
- **group_by()**: Agrupa las observaciones por grupos.

Para conectar las operaciones se usa el **operador pipe**, %>%. Como describe Jean-François Mas [1.16], este operador permite encadenar funciones tomando la salida de una y pasándola como entrada de la siguiente. Ayuda enormemente a mejorar la legibilidad del código ya que anidar las funciones puede resultar confuso.

Mediante el siguiente ejemplo se explica cómo manipular los datos usando la gramática del paquete **dplyr** junto con el encadenamiento de operaciones con el operador **pipe**. Se pretende obtener el mapa provincial por municipios de León de la Figura 1.7 usando la cartografía digitalizada de secciones censales del INE de 2020. Primero se plantea el método de resolución y después se muestra el código con el que se obtiene el resultado.

Primero es necesario extraer las observaciones asociadas a la provincia de León con **filter()**, usando como filtro el código de provincia correspondiente esta región. Con **group_by()** se agrupan las filas por código de municipio y luego se aplica **n()** dentro de **summarise()**. Estas funciones trabajan conjuntamente de forma que el resultado de la operación implica crear una nueva variable, denominada **n**, que contiene el número de filas que forman parte de los grupos dados por group_by(). Así, **n** es el número de secciones censales de cada municipio, y el resultado es la agrupación de las geometrías de cada uno. Por último, se aplica **select()** para simplificar el objeto *sf* obtenido y que el conjunto de datos resultante contenga solo los códigos de los municipios y sus geometrías.

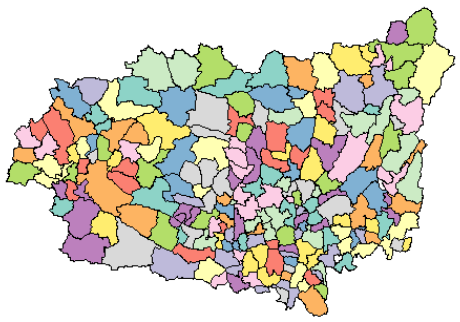


Figura 1.7. Mapa provincial por municipios de León, obtenido para el ejemplo de manipulación de datos con el paquete **dplyr**.

Ejecutando el fragmento de código que se muestra a continuación se obtiene el mapa provincial por municipios de León de la Figura 1.7.

```
contornos <- cartografia %>% filter(CPRO == 24) %>%
             group_by(CUMUN) %>% summarise(n = n()) %>%
             select(CUMUN)
```

El conjunto de datos sobre el que se aplican las operaciones debe situarse en la primera posición, seguido de las funciones. Cada argumento debe estar separado por %>%.

Paquete **pxR**

El formato escogido para la descarga de los ficheros que contienen las cifras poblacionales de la Estadística del Padrón Continuo es **PC-Axis**. Manuel F. García Barona [1.17] describe PC-Axis como un programa desarrollado por el Instituto Estadístico Sueco para la presentación de datos numéricos que es empleado por varias oficinas europeas de estadística, como es el caso de Instituto Nacional de Estadística. Con el paquete **pxR** [1.18] se pueden importar los datos de este tipo de forma sencilla.

Para instalar y cargar el paquete se deben ejecutar las siguientes líneas de código:

```
install.packages("pxR")
library(pxR)
```

Para cargar los datos se usa la función **read.px()**. Con el siguiente fragmento de código se muestra cómo transformar las cifras poblacionales del Padrón Continuo del año 2011 a un *dataframe* tras leer los datos:

```
padron <- as.data.frame(read.px("descargas/padron/2011.px"))
```

1.2.2. Depuración y Adecuación de los Datos

A continuación se enumeran las operaciones aplicadas sobre los diferentes conjuntos de datos para depurarlos y adecuarlos al resto de información de su bloque.

Cartografía del INE

- Localización y corrección de errores en los nombres de algunas provincias en los periodos 2013, 2014 y 2015.
- Transformación de las coordenadas.

Cartografía del CNIG

- Modificación de los nombres de las provincias y comunidades autónomas por sus correspondientes códigos para que la semántica concuerde con la de los datos cartográficos del INE.
- Transformación de las coordenadas.

Cifras poblacionales del Padrón Continuo del INE

- Modificación de todos los campos de los conjuntos de datos de los años 2011 y 2012 para adecuarlos a la semántica de las variables de la serie 2013-2020.
- Modificación del contenido de la variable Municipios, que contiene el código de cada municipio seguido de su nombre en los ficheros originales, para que solo consten los códigos.

1.3. Generación de la Base de Datos

La fase anterior es clave para conocer el contenido de los diferentes conjuntos de datos y aprender a manipularlos mediante su depuración. Ahora, tras estudiar las variables que forman parte de cada uno y planear el tipo de representaciones que posteriormente se obtendrán con la información recabada, se plantea la estructura de los conjuntos de datos y la organización de la base de datos.

La Figura 1.8 muestra la distribución de los ficheros que contienen la información de la base de datos según cuatro bloques. Los datos cartográficos proporcionan la base sobre la que se representan las cifras e indicadores de población. La combinación de la información de estos dos bloques, junto con las tablas de referencia, dará como resultado los mapas, que se describen en el próximo capítulo (ver 2.4).

A continuación se describe la estructura de cada tipo de conjunto de datos dentro de cada uno de los tres primeros bloques.

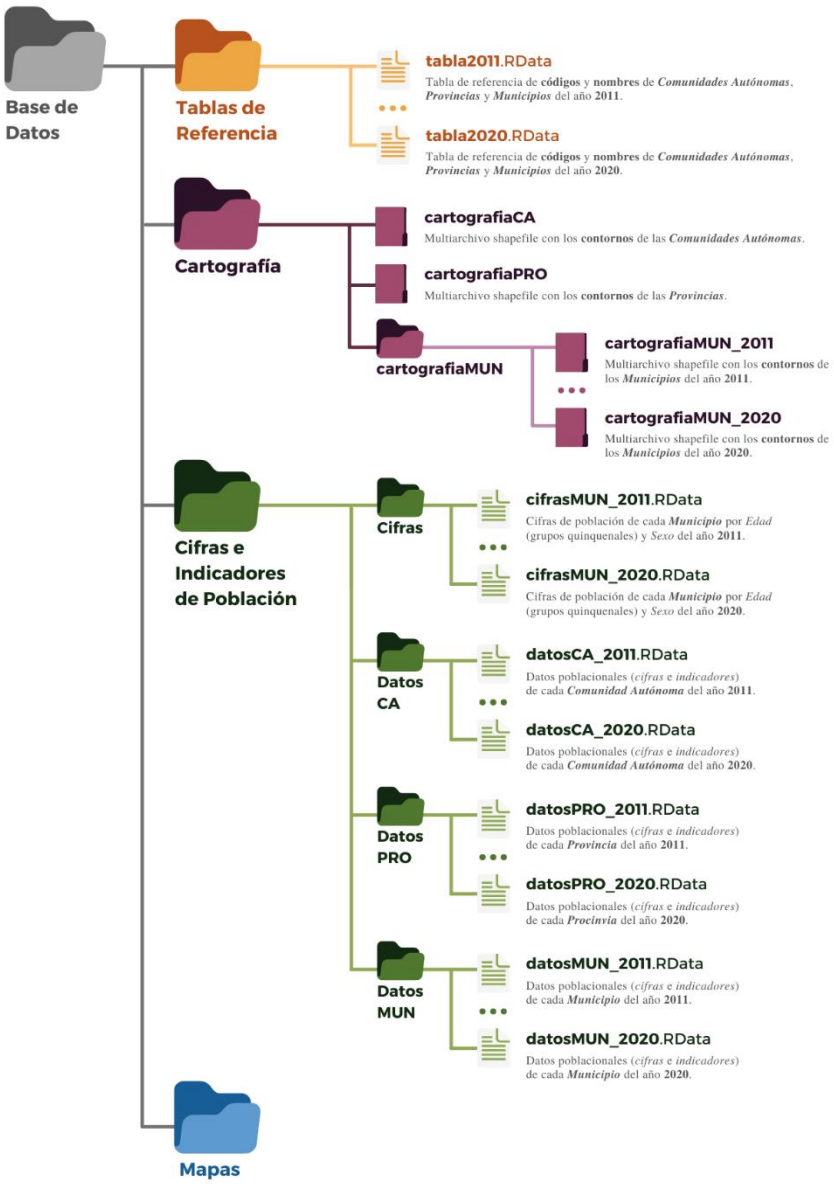


Figura 1.8. Esquema del almacenamiento de la base de datos.

1.3.1. Tablas de Referencia

Las **tablas de referencia** contienen la relación de todos los municipios por provincia y, a su vez, por comunidad autónoma, incluyendo los códigos que referencia a cada territorio en los diferentes ámbitos geográficos. Son una herramienta indispensable para generar el resto de datos y obtener las representaciones.

Estas tablas se obtienen mediante los datos de la cartografía digitalizada de secciones censales del INE (ver 1.1.1) a partir de los códigos (**CCA**, **CPRO** y **CUMUN**) y los

nombres de los territorios asociados a cada uno (**NCA**, **NPRO** y **NMUN**). En total se dispone de 10 tablas para los 10 años de los que se ha tomado la información, de 2011 a 2020. Al dividir los datos por años en varias tablas se solventa el problema de acceso a los datos que ocasionan los cambios en la distribución del territorio por los municipios que desaparecen y se crean cada año.

Cada tabla da la relación entre cada municipio y los ámbitos geográficos superiores, lo que sirve como herramienta para el cálculo de las cifras e indicadores de población a nivel autonómico y provincial, facilitando la agrupación de la información del conjunto de municipios que componen cada territorio en función de los códigos. Por otro lado, como se trabaja únicamente con códigos, los nombres de los territorios asociados a cada uno se pierden. Esta información es necesaria a la hora de representar los datos puesto que los mapas deben contener los nombres de cada municipio, provincia y/o comunidad autónoma, por lo que las tablas de referencia ayudarán a identificarlos según los códigos.

1.3.2. **Cartografía**

Los **datos cartográficos** son estructuras de tipo *simple features* que contienen las geometrías de los contornos de los territorios y los códigos asociados a cada uno. Cada conjunto de datos, según el ámbito geográfico, se almacena en un fichero shapefile, como se indica en la Figura 1.8. Como se menciona al inicio del capítulo, en el apartado en el que se describen estos datos (ver 1.1.1), la información topográfica de comunidades autónomas y provincias es la misma para todos los periodos a representar, ya que no varía, mientras que la distribución municipal del territorio se modifica, aunque ligeramente, prácticamente de forma anual, lo que lleva a tener un conjunto de datos distinto para cada año.

En la Figura 1.9 se esquematiza el contenido de cada tipo de *sf* de este bloque según la información geográfica que contiene.

Figura 1.9. Esquema del contenido de los datos del bloque **Cartografía**.

Los datos cartográficos de las comunidades autónomas y las provincias se obtienen combinando los códigos de cada región con su geometría, extrayendo la información de cada contorno de los ficheros del CNIG. La cartografía municipal requiere agrupar las secciones censales por cada municipio y eliminar toda la información irrelevante.

1.3.3. **Cifras Poblacionales e Indicadores de Población**

Este bloque está compuesto por dos tipos de datos: por un lado, las cifras de población por municipios dadas por los archivos del Padrón Continuo, cuya estructura no se ha modificado respecto a la original (ver 1.1.2), y por otro lado, las tablas a diferentes ámbitos geográficos que contienen las cifras de población totales y desagregadas por grandes grupos de edad (población menor de 20 años, GE1, de entre 20 y 64 años, GE2, y mayor de 64 años, GE3) y sexo (hombres y mujeres) junto con los indicadores de población calculados a partir de las fórmulas dadas según la metodología previamente explicada (ver 1.1.4). En la Figura 1.10 se puede ver un esquema del contenido de cada tipo de dato de este bloque.

Figura 1.10. Esquema del contenido de los datos del bloque **Cifras e Indicadores de Población**.

Las **cifras de población** son el resultado del proceso de depuración y adecuación de los datos de la etapa anterior. Las **tablas con los datos de población** se obtienen agrupando la información anterior según el ámbito geográfico y desagregándola según grandes grupos de edad y sexo. Con los datos obtenidos de esta operación, junto con las superficies (ver 1.1.1 y 1.1.3), se calculan los indicadores de población.

2. Mapas

La información recopilada, depurada y almacenada en los tres primeros bloques de la base de datos permitirá generar las representaciones que se visualizarán en la aplicación.

Crear y visualizar mapas interactivos en el momento conlleva demasiadas operaciones, lo que hace que el tiempo de respuesta de la aplicación sea muy elevado. Es necesario generarlos previamente y guardarlos como archivos HTML para después visualizarlos en función de las especificaciones dadas por el usuario. Por ello, los mapas componen el último bloque de la base de datos, como se puede ver en la Figura 1.8. La Figura 2.14 muestra el esquema de almacenamiento de los ficheros que contienen estas representaciones cartográficas, obtenidas como se describe en este capítulo.

Pero antes es necesario profundizar en el término **datos espaciales**, mencionados en el capítulo anterior para hacer una breve introducción del paquete **sf** (ver 1.2.1), y continuar con la descripción de los objetos de tipo *simple features*.

2.1. Datos Espaciales con Simple Features

Un **dato espacial** [2.1] es la información geográfica asociada a una ubicación concreta y que además puede contener atributos que describen esa localización. Existen dos tipos de datos espaciales: los datos **raster**, que representa la información topográfica mediante una cuadrícula de píxeles, y los datos **vectoriales**, que lo hacen mediante geometrías basadas en puntos, líneas y polígonos.

Simple features [1.13][2.2] es un formato estándar (ISO 19125-1:2004) para almacenar **datos espaciales vectoriales**, incluyendo tanto información espacial (geometrías) como no georreferenciada (atributos). El paquete **sf** [1.12] implementa los objetos de tipo *simple features*, ***sf***, en R [0.1] y proporciona las funciones necesarias para crear y manipular este tipo de datos.

Un objeto ***sf***, *simple feature*, está formado por un *dataframe* que contiene una columna ***sfc***, *simple feature geometry list-column*, con objetos ***sfg***, *simple feature geometry*, que contienen las geometrías. En la Figura 2.1 se puede ver cada parte de un ***sf***, mientras que la Figura 2.2 ilustra el esquema de esta descomposición.

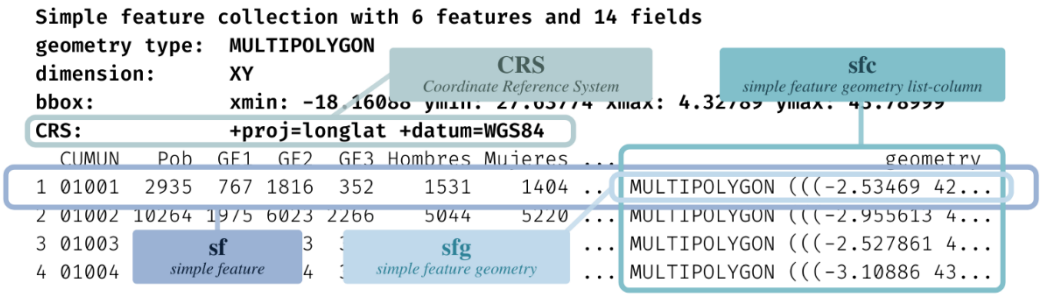


Figura 2.1. Partes de un ***sf***, *simple feature*.

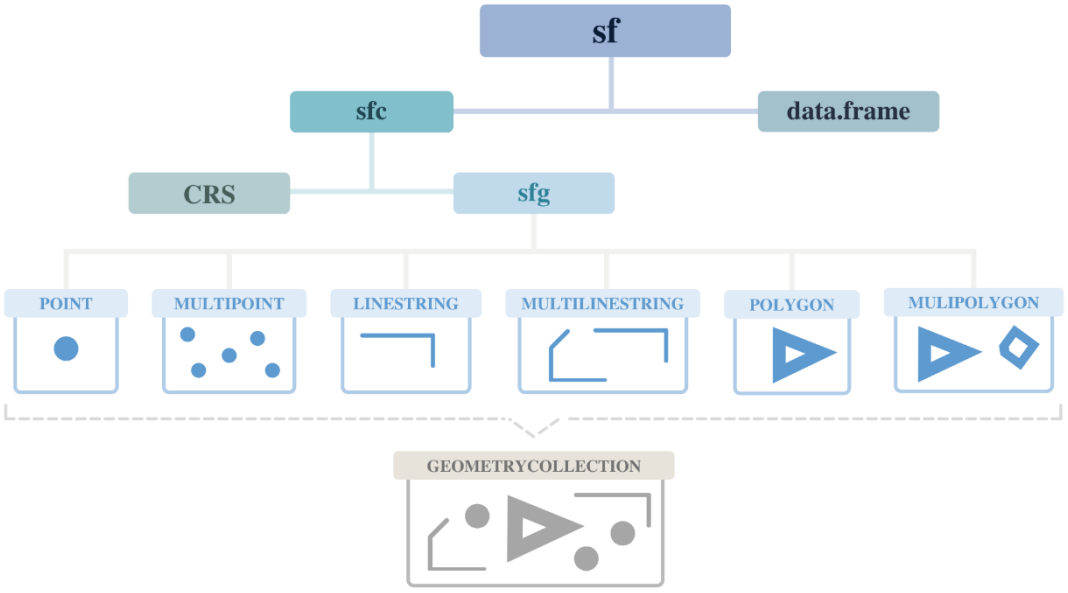


Figura 2.2. Esquema de la descomposición del objeto ***sf***.

La clase ***sfg***, *simple feature geometry*, contiene las geometrías de cada *simple feature*. Hay 7 tipos básicos, ilustrados en la Figura 2.2, y detallados a continuación. El paquete **sf** incluye funciones específicas para crear cada uno.

- **POINT**: Punto simple, vector (x, y). Función: **st_point()**.
- **MULTIPOINT**: Múltiples puntos, matriz con un punto en cada fila. Función: **st_multipoint()**.
- **LINESTRING**: Líneas, secuencia de puntos conectados por líneas rectas, lista de matrices. Función: **st_linestring()**.
- **MULTILINESTRING**: Múltiples líneas, lista de listas de marices. Función: **st_multilinestring()**.

- **POLYGON**: Polígono cerrado que puede huecos en su interior, lista matrices. Función: **st_polygon()**.
- **MULTIPOLYGON**: Múltiples polígonos. Función: **st_multipolygon()**.

Cualquier combinación de los anteriores forma una **GEOMETRYCOLLECTION**, el séptimo tipo de geometría básico. Este tipo de geometría puede crearse con la función es **st_geometrycollection()**.

Los ***sfc***, *simple feature geometry list-column*, son objetos ***sfg*** a los que se les asocia un **CRS,** *Coordinate Reference System*. No basta con crear las geometrías de los datos espaciales, es esencial asignarles la información geográfica necesaria para ubicarlos en el mundo real. Utilizando la función **st_sfc(sfg, crs)** se asocia un CRS a uno o más ***sfg***, que puede definirse de dos formas: mediante un código **EPSG** o bien como texto con el formato **proj4string**.

Un **Sistema de Coordenadas de Referencia** o **CRS** [2.3] proporciona una forma estandarizada de describir la posición de una localización geográfica. Los tres principales atributos de un **CRS** son el **Elipsoide de Referencia**, el **Datum** y la **proyección**. La Figura 2.3 describe estos componentes.

Figura 2.3. Descripción de los tres componentes principales de un **CRS**.

Debido a que la Tierra no es esférica, se emplea un **Elipsoide de Referencia** como aproximación a la forma del geoide. Para que un CRS se ajuste de la mejor forma posible a la superficie terrestre, se emplea el **Datum**, un sistema de referencia que especifica el

elipsoide a usar y define el origen y la orientación de los ejes de coordenadas. La **proyección** convierte los Datums en un plano de dos dimensiones.

Los dos sistemas de referencia de coordenadas más usados son, por un lado, el **sistema de referencia de coordenadas geográfico**, que emplea las coordenadas angulares **latitud** (norte y sur) y **longitud** (este oeste), y, por otro lado, **sistema de referencia de coordenadas proyectadas**, que se define sobre una superficie plana. El sistema **UTM**, del que se habló al inicio del primer capítulo (ver 1.1.1) es un ejemplo de sistema de referencia de coordenadas proyectadas.

El **ETRS89** [2.4], *European Terrestrial Reference System 1989*, es el sistema de referencia geodésico oficial en España adoptado según el Real Decreto 1071/2007. Su código EPSG es **4258**, con el que se especifica tanto el **Datum** como las **coordenadas geográficas**. Para **UTM**, el código EPSG es **258xx**, donde los tres primeros dígitos especifican el sistema ETRS89 y los dos últimos se refieren al huso (28, 29, 30 o 31). Para transformar las coordenadas de un ***sf*** con los códigos **EPSG** mencionados, según la proyección a aplicar, se usaría **st_transform(sf, 4258)** o **st_transform(sf, 258xx)**. Sin embargo, para definir el sistema de coordenadas y el Datum con **proj4string**, se usa una cadena de caracteres. El equivalente al código EPSG 4258 sería **st_transform(sf, "+proj=longlat +datum= ETRS89")** y para UTM se aplicaría **st_transform(sf, "+proj=utm +zone=xx +datum= ETRS89")**.

Para poder representar los datos cartográficos con la librería **Leaflet**, que se describe en el siguiente apartado, es necesario transformar los datos para usar el Datum **WGS84** y el sistema de coordenadas geográficas. El **WGS84** [2.4], *World Geodetic System 1984*, es el sistema geodésico de coordenadas global que permite localizar cualquier ubicación de la Tierra. Su código EPSG es **4326** y previamente se indicó cómo transformar las coordenadas con proj4string (ver 1.2.1).

2.2. Mapas Interactivos

Los **mapas interactivos** permiten visualizar datos sobre una representación cartográfica que ofrece la posibilidad de que el usuario pueda desplazarse o hacer zum para consultar la información que se muestra. Además, es posible incluir **pop-ups**, ventanas emergentes que muestran contenido complementario.

Cada tipo de mapa requiere un código distinto debido a ciertas diferencias entre unos y otros. Más adelante se describirá cada mapa con ejemplos. Sin embargo, la estructura es similar entre los códigos y sigue el esquema que se muestra en la Figura 2.4.

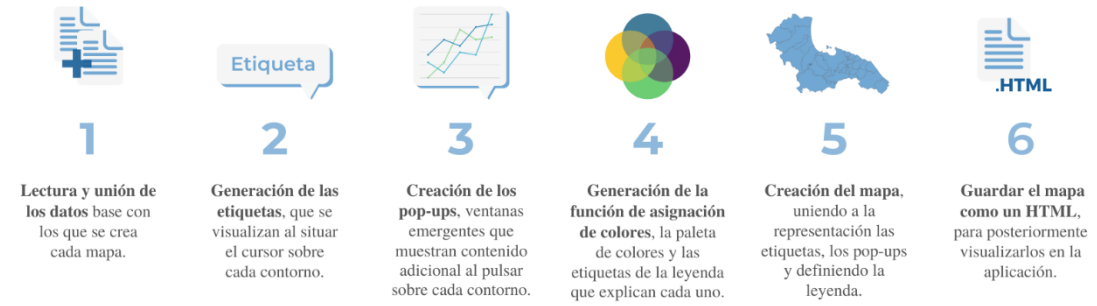


Figura 2.4. Esquema de las partes del código para la generación de los mapas interactivos.

Leaflet es una librería de JavaScript para crear mapas interactivos. El paquete de R con el mismo nombre, **leaflet** [2.5][2.6], permite crear e integrar estos mapas en R. Para crear el contenido de los **pop-ups** se utiliza **ggplot2** [2.7][2.8], un paquete creado por Hadley Wickham basado en *The Grammar of Graphics* [2.9] de Leland Wilkinson. El paquete **leafpop** [2.10] permite integrar los pop-ups en los mapas.

Por otro lado, es necesario guardar los mapas generados como archivos HTML, por el motivo que se indicó al inicio de este capítulo. Para ello se usa la función **mapshot** del paquete **mapview** [2.11]. De nuevo, para instalar y cargar esta librería se deben usar las funciones **install.packages()** y **library()**, respectivamente.

La paleta de colores escogida para representar la información en los mapas es **viridis**, del paquete con el mismo nombre, **viridis** [2.12].

2.2.1. Generación de los Mapas Interactivos

Para ilustrar el proceso de generación de mapas se hará uso de la cartografía de las provincias (ver Figura 1.9) junto con el conjunto de datos que contiene las cifras e indicadores de población asociados del año 2020 (ver Figura 1.10).

Utilizando la función **merge**, del paquete **data.table** [2.13], una extensión de data.frame, se une la información cartográfica y los datos de población por medio de la variable **CPRO**, que contiene los códigos de cada provincia. La Figura 2.5 muestra el resumen del conjunto de datos resultante.

```
base <- merge(cartografia, datos, by.x = "CPRO", by.y = "CPRO")
```

```
Simple feature collection with 52 features and 14 fields
geometry type:  MULTIPOLYGON
dimension:      XY
bbox:           xmin: -18.16118 ymin: 27.63772 xmax: 4.327785 ymax: 43.79238
CRS:            +proj=longlat +datum=WGS84
First 10 features:
```

	CPRO	Población	GE1	GE2	GE3	Hombres	Mujeres	PjeAG	PjeGE1	PjeGE2	PjeGE3	Densidad	IndEnv	IndMas	geometry
1	01	333940	64486	198978	70476	164524	169416	0.70	19.31	59.58	21.10	109.92	109.29	97.11	MULTIPOLYGON (((-3.131297 4...
2	02	388270	74873	238018	75379	194081	194189	0.82	19.28	61.30	19.41	26.01	100.68	99.94	MULTIPOLYGON (((-2.762077 3...
3	03	1879888	367512	1139374	373002	931856	948032	3.96	19.55	60.61	19.84	323.17	101.49	98.29	MULTIPOLYGON (((-0.7535028 ...
4	04	727945	161308	457956	108681	371558	356387	1.53	22.16	62.91	14.93	82.96	67.37	104.26	MULTIPOLYGON (((-3.035286 3...
5	05	157664	25694	90838	41132	79109	78555	0.33	16.30	57.61	26.09	19.59	160.08	100.71	MULTIPOLYGON (((-4.461321 4...
6	06	672137	129558	410587	131992	332204	339933	1.42	19.28	61.09	19.64	30.88	101.88	97.73	MULTIPOLYGON (((-6.924769 3...
7	07	1171543	232156	756163	183224	584298	587245	2.47	19.82	64.54	15.64	234.68	78.92	99.50	MULTIPOLYGON (((1.479072 38...
8	08	5743402	1144373	3511690	1087339	2804316	2939086	12.10	19.93	61.14	18.93	743.19	95.02	95.41	MULTIPOLYGON (((2.639748 41...
9	09	357650	61681	210180	85789	178578	179072	0.75	17.25	58.77	23.99	25.03	139.08	99.72	MULTIPOLYGON (((-2.86886 42...
10	10	391850	65742	234854	91254	194084	197766	0.83	16.78	59.93	23.29	19.72	138.81	98.14	MULTIPOLYGON (((-5.810367 3...

Código CPRO

Cifras e Indicadores de Población de las Provincias

Geometrías de los Datos Cartográficos

Figura 2.5. Resumen del conjunto de datos **base** obtenido al unir la cartografía con los datos de población de las provincias del año 2020.

Cada contorno deberá tener asociado una **etiqueta** que muestre información de la ubicación sobre la que se sitúe el cursor, por lo que se debe generar una lista que contenga el texto a visualizar. Para esta operación se utilizan las **tablas de referencia** (ver Tablas de Referencia), que darán los nombres de cada región. En el ejemplo que se está siguiendo, para crear las etiquetas que se muestran en la Figura 2.6, se utiliza el siguiente fragmento de código. En la Figura 2.7 y la Figura 2.8 se puede ver como unirlas con los mapas.

```
etiquetas <- list()
for(i in 1:52) {
  ind   <- which(base$CPRO[i] == tabla$CPRO)[1]
  NPRO  <- codificacion(as.character(tabla[ind,'NPRO']))
  NCA   <- codificacion(as.character(tabla[ind,'NCA']))
  texto <- paste("<strong>", NPRO, ", ", NCA,
                 "</strong>", sep = "")
  etiquetas[[i]] <- htmltools::HTML(texto)
}
```

Tener en cuenta que se emplean etiquetas propias del lenguaje HTML, como para generar el texto en negrita, para crear el texto que compone cada etiqueta. Por ello, es necesario procesar la cadena de caracteres para darle el formato adecuado con las características propias de HTML. De este modo, al generar el mapa, se aplicará el formato especificado al texto contenido en las etiquetas.

La función **codificación** cambia los caracteres especiales (caracteres acentuados, eñes o cedillas, entre otros) por sus correspondientes códigos en HTML[3]. Pese a que en todo momento se trabaja con codificación UTF-8, algunos símbolos producen errores en la conversión del mapa, por ello es muy importante transformar esos caracteres.

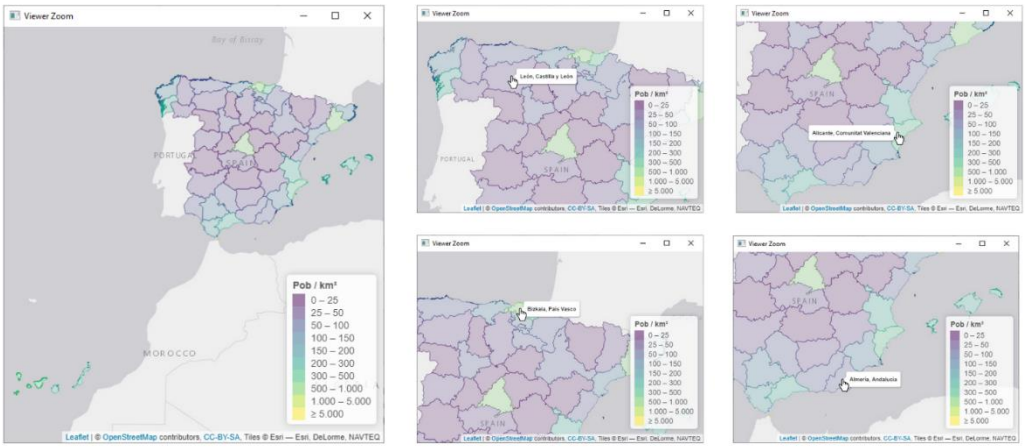


Figura 2.6. Ejemplo de la vista de varias etiquetas sobre el mapa que representa la densidad de población por provincia según los datos de 2020.

Para asignar los colores a cada contorno se puede emplear dos funciones distintas, dependiendo de la división que se desee hacer. Por un lado, **colorQuantile** proporciona una partición equiespaciada entre un determinado rango de valores. Esta opción es útil, por ejemplo, para generar la asignación de colores de los mapas que muestran las cifras de población. Como el rango de valores es muy amplio, especialmente a nivel municipal, es necesario aplicar una transformación logarítmico decimal[4] y con **colorQuantile** se generará una gama de colores adecuada, como se puede ver en el ejemplo ilustrado en la Figura 2.7. Por otro lado, la función **colorBin** permite crear una división especificando los valores concretos en los que realizar los costes. Por ejemplo, en el caso de los mapas que muestran la densidad de población, es preferible crear más grupos entre un cierto rango de valores para apreciar diferencias más pequeñas. La Figura 2.8 muestra un ejemplo aplicando esta función.

[3] Ver https://www.freeformatter.com/html-entities.html.

[4] El rango de valores de las cifras de población es muy amplio, lo que impide ver diferencias entre territorios con menor población al quedar completamente opacados por el tamaño de regiones con mayor número de habitantes. Aplicando la **transformación logarítmica decimal** se logra distribuir magnitudes muy distintas en una escala de valores que viene a representar el número de cifras de cada valor, facilitando así la interpretación de los datos.

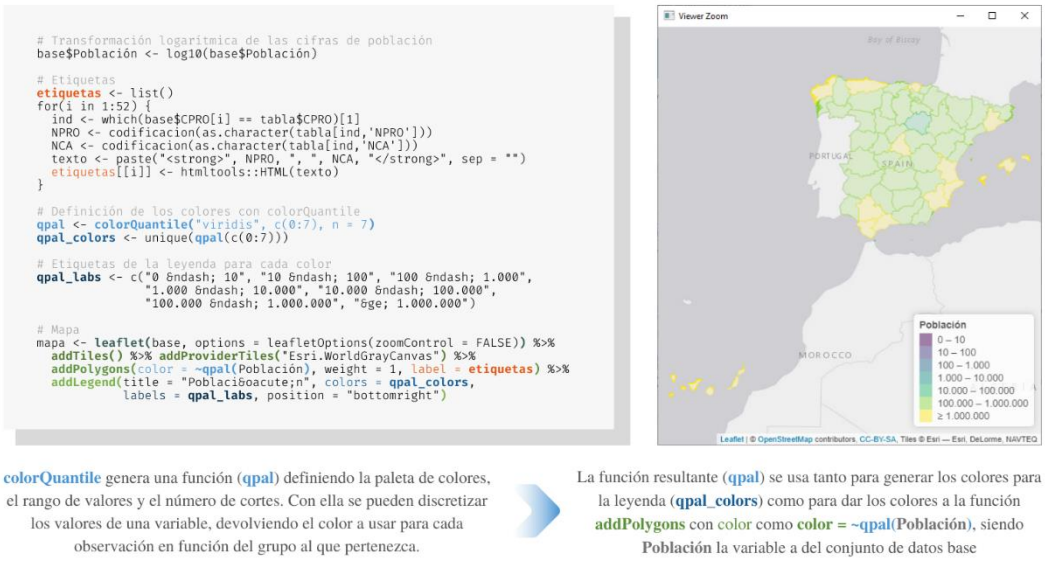


Figura 2.7. Código para la generación del mapa interactivo que representa las cifras de población de cada provincia según los datos del 2020. Ejemplo de uso de **colorQuantile**.

Figura 2.8. Código para la generación del mapa interactivo que representa la densidad de población de cada provincia según los datos del 2020. Ejemplo de uso de **colorBin**.

Como se ha podido comprobar, para generar los mapas se usan diferentes funciones anidadas con el **operador pipe**, %>%, cuya descripción se dio previamente (ver 1.2.1). La Figura 2.9 describe cada una de las funciones utilizadas para generar los mapa de la Figura 2.7 y la Figura 2.8.

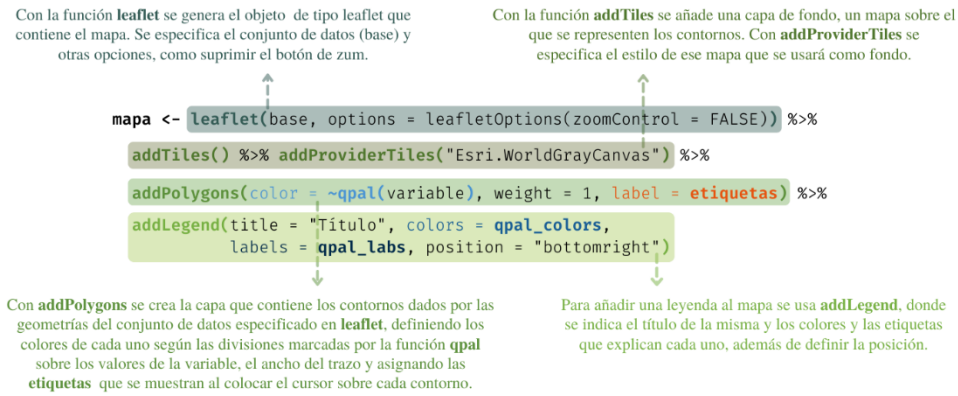


Figura 2.9. Descripción de las funciones **leaflet**, **addTiles**, **addProviderTiles**, **addPolygons** y **addLegend**, empleadas para generar los mapas.

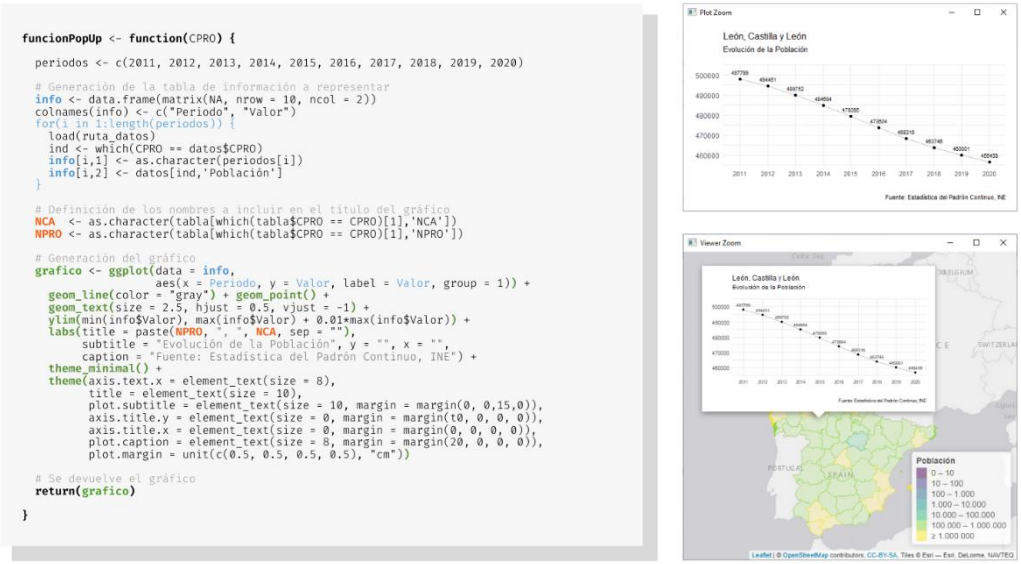
2.2.2. Pop-Ups

Los **pop-ups**, al igual que las etiquetas, se almacenan en una lista que posteriormente se le pasará a uno de los comandos de **leaflet** para unirlos con los mapas haciendo uso de una función dada por el paquete **leafpop**. Siguiendo con el ejemplo propuesto para representar la información asociada a las provincias, con el siguiente fragmento de código se muestra el bucle con el que se generan los gráficos que se visualizan en las ventanas emergentes. Se ha supuesto una función genérica denominada **funciónPopUp**.

```
popups <- list()
popups <- lapply(1:52, function(i){
  popups[[i]] <- funcionPopUp(base$CPRO[i])
})
```

Esta función devuelve el gráfico que se visualizará en el pop-up asociado al contorno según el código de provincia, CPRO. La Figura 2.10 describe la estructura que comparten el conjunto de funciones que crean estas representaciones, que muestran la evolución de las cifras o los indicadores de población entre 2011 y 2020, tomando como ejemplo la función que da los pop-ups que se conectan al mapa de la Figura 2.7.

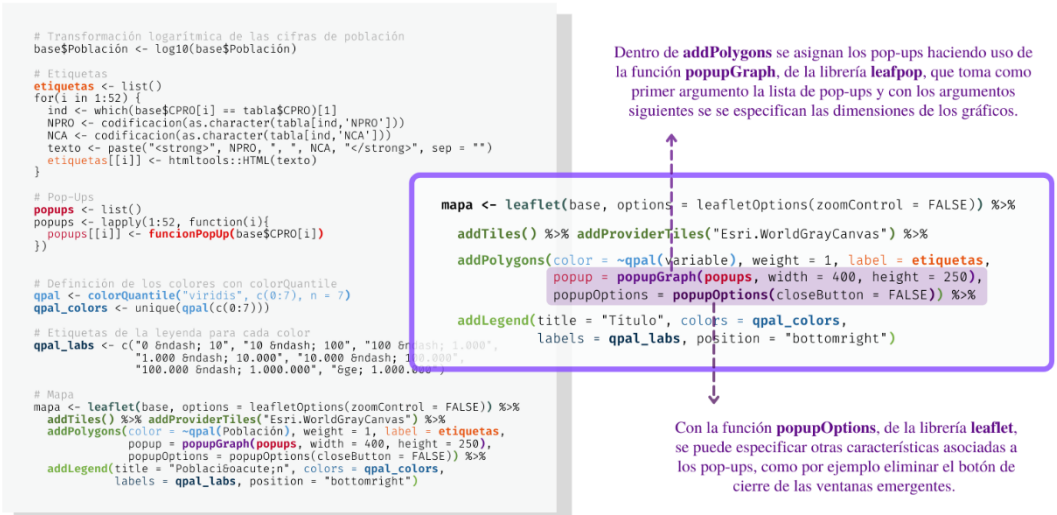
Primero se genera la tabla de información, tomando los datos de los diferentes periodos de la variable a representar. Después, se crea el gráfico haciendo uso de las funciones del paquete **ggplot2**. El título de la representación viene dado por el nombre del territorio según el código que recibe como argumento **funciónPopUp**, que se obtiene utilizando las **tablas de referencia** (ver 1.3.1). Por último, se devuelve el gráfico, para almacenarlo en la lista que contiene cada uno de los pop-ups.



```r
funcionPopUp <- function(CPRO) {

  periodos <- c(2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020)

  # Generación de la tabla de información a representar
  info <- data.frame(matrix(NA, nrow = 10, ncol = 2))
  colnames(info) <- c("Periodo", "Valor")
  for(i in 1:length(periodos)) {
    load(ruta_datos)
    ind <- which(CPRO == datos$CPRO)
    info[i,1] <- as.character(periodos[i])
    info[i,2] <- datos[ind,'Población']
  }

  # Definición de los nombres a incluir en el título del gráfico
  NCA  <- as.character(tabla[which(tabla$CPRO == CPRO)[1],'NCA'])
  NPRO <- as.character(tabla[which(tabla$CPRO == CPRO)[1],'NPRO'])

  # Generación del gráfico
  grafico <- ggplot(data = info,
                    aes(x = Periodo, y = Valor, label = Valor, group = 1)) +
    geom_line(color = "gray") + geom_point() +
    geom_text(size = 2.5, hjust = 0.5, vjust = -1) +
    ylim(min(info$Valor), max(info$Valor) + 0.01*max(info$Valor)) +
    labs(title = paste(NPRO, ", ", NCA, sep = ""),
         subtitle = "Evolución de la Población", y = "", x = "",
         caption = "Fuente: Estadística del Padrón Continuo, INE") +
    theme_minimal() +
    theme(axis.text.x = element_text(size = 8),
          title = element_text(size = 10),
          plot.subtitle = element_text(size = 10, margin = margin(0, 0,15,0)),
          axis.title.y = element_text(size = 0, margin = margin(t0, 0, 0, 0)),
          axis.title.x = element_text(size = 0, margin = margin(0, 0, 0, 0)),
          plot.caption = element_text(size = 8, margin = margin(20, 0, 0, 0)),
          plot.margin = unit(c(0.5, 0.5, 0.5, 0.5), "cm"))

  # Se devuelve el gráfico
  return(grafico)

}
```


Figura 2.10. Código de la función para generar los gráficos de la evolución de las cifras de población entre 2011 y 2020 para los mapas nacionales por provincias. La imagen superior derecha muestra un ejemplo de la representación que devuelve la función y la inferior el resultado en el mapa interactivo.

En la Figura 2.11 se puede ver cómo unir la lista de gráficos con el mapa para que cada uno se asocie a su correspondiente contorno y se visualicen al pulsar sobre cada región en el mapa interactivo resultante.



```r
# Transformación logarítmica de las cifras de población
base$Población <- log10(base$Población)

# Etiquetas
etiquetas <- list()
for(i in 1:52) {
  ind <- which(base$CPRO[i] == tabla$CPRO)[1]
  NPRO <- codificacion(as.character(tabla[ind,'NPRO']))
  NCA <- codificacion(as.character(tabla[ind,'NCA']))
  texto <- paste("<strong>", NPRO, ", ", NCA, "</strong>", sep = "")
  etiquetas[[i]] <- htmltools::HTML(texto)
}

# Pop-Ups
popups <- list()
popups <- lapply(1:52, function(i){
  popups[[i]] <- funcionPopUp(base$CPRO[i])
})

# Definición de los colores con colorQuantile
qpal <- colorQuantile("viridis", c(0:7), n = 7)
qpal_colors <- unique(qpal(c(0:7)))

# Etiquetas de la leyenda para cada color
qpal_labs <- c("0 &ndash; 10", "10 &ndash; 100", "100 &ndash; 1.000",
               "1.000 &ndash; 10.000", "10.000 &ndash; 100.000",
               "100.000 &ndash; 1.000.000", "&ge; 1.000.000")

# Mapa
mapa <- leaflet(base, options = leafletOptions(zoomControl = FALSE)) %>%
  addTiles() %>% addProviderTiles("Esri.WorldGrayCanvas") %>%
  addPolygons(color = ~qpal(Población), weight = 1, label = etiquetas,
              popup = popupGraph(popups, width = 400, height = 250),
              popupOptions = popupOptions(closeButton = FALSE)) %>%
  addLegend(title = "Población",
            labels = qpal_labs, position = "bottomright")
```

Dentro de **addPolygons** se asignan los pop-ups haciendo uso de la función **popupGraph**, de la librería **leafpop**, que toma como primer argumento la lista de pop-ups y con los argumentos siguientes se se especifican las dimensiones de los gráficos.

```r
mapa <- leaflet(base, options = leafletOptions(zoomControl = FALSE)) %>%
  addTiles() %>% addProviderTiles("Esri.WorldGrayCanvas") %>%
  addPolygons(color = ~qpal(variable), weight = 1, label = etiquetas,
              popup = popupGraph(popups, width = 400, height = 250),
              popupOptions = popupOptions(closeButton = FALSE)) %>%
  addLegend(title = "Título", colors = qpal_colors,
            labels = qpal_labs, position = "bottomright")
```

Con la función **popupOptions**, de la librería **leaflet**, se puede especificar otras características asociadas a los pop-ups, como por ejemplo eliminar el botón de cierre de las ventanas emergentes.

Figura 2.11. Código para la generación del mapa interactivo que representa las cifras de población de cada provincia según los datos del 2020 incluyendo **Pop-Ups**. Descripción de las funciones **popupGraph** y **popupOptions**, de los paquetes leafpop y leaflet, respectivamente, con las que se integran los gráficos a visualizar en las ventanas emergentes.

2.2.3. Guardar los Mapas como HTML

Para guardar los mapas como archivos HTML se usa la función **mapshot** del paquete **mapview**, como se mencionó al inicio de este apartado. El siguiente fragmento de código muestra cómo se genera un fichero que contendrá un mapa interactivo.

```
mapshot(mapa, url = "ruta_destino")
```

Como resultado se obtiene un documento HTML, desde el que se puede visualizar directamente el mapa, y una carpeta que contiene todas las especificaciones e información asociadas a él.

2.3. Mapas Animados

Los **mapas animados**, como su propio nombre indica, son animaciones que permiten visualizar la evolución de cierta variable sobre el mapa. Son estáticos en el sentido de que no ofrecen ningún tipo de interacción con el usuario, a diferencia de los interactivos, por lo que su creación no implica generar ni etiquetas ni pop-ups.

La función **saveGIF** del paquete **animation** [2.14] permite convertir una serie de imágenes en una animación en formato **gif**. La Figura 2.12 muestra cómo se genera un mapa animado empleando esta función.

```
saveGIF({
  for(periodo in c(2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020)) {

    # Generación de los datos base
    ruta_cartografia <- paste(".../cartografiaMUN_", periodo, ".shp", sep = "")
    cartografia <- st_read(ruta_cartografia)
    ruta_datos <- paste(".../datosMUN_", periodo, ".RData", sep = "")
    load(ruta_datos)

    # Datos Base
    base <- merge(cartografia, datos, by.x = 'CUMUN', by.y = 'CUMUN')
    base$Población <- log10(base$Población)

    # Mapa
    grafico <- ggplot() + geom_sf(base, aes(fill = Población), lwd = NA) +
      scale_fill_gradientn("Población", colours = viridis(6),
                           breaks = c(1, 2, 3, 4, 5, 6), limits = c(0, 7),
                           labels = etiquetas.leyena) + theme_void() +
      labs(title = paste("Evolución de la Población, ", periodo, sep = "")) +
      theme(plot.title = element_text(size = 20, margin = margin(0, 0, 6, 0)),
            plot.subtitle = element_text(size = 18, margin = margin(0, 0, 10, 0)),
            legend.title = element_text(size = 15, margin = margin(0, 0, 5, 0)),
            legend.key.height = unit(1, 'cm'), legend.key.width = unit(0.7, 'cm'),
            legend.text = element_text(size = 11),
            plot.margin = unit(c(0.5, 2, 0, 1), "cm"),
            legend.box.margin = margin(0, 0, 0, 5))

    print(grafico)

  }
}, movie.name = ruta_destino, interval = 2, ani.width = 1000, ani.height = 700)
```

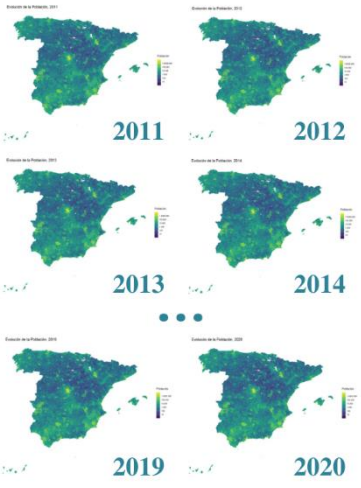


Figura 2.12. Función para generar el mapa nacional de la evolución de las cifras de población por municipios entre 2011 y 2020 con la función **saveGIF** utilizando el paquete **ggplot2** para crear los mapas. A la derecha se ve la sucesión de imágenes que se generan dentro del bucle.

El primer argumento de la función **saveGIF** es un bucle que crea cada mapa según el periodo, entre 2011 y 2020, y lo imprime. El resto de argumentos especifican el nombre (movie.name), el intervalo de tiempo en segundos entre cada gráfico (interval) y el ancho y largo de la imagen final (ani.width y ani.height, respectivamente).

La primera parte del bucle no es más que la lectura y unión de los datos, como se explicó en la primera parte de la generación de los mapas interactivos (ver 2.2.1), pero en este caso utilizando la información asociada a los municipios. Puesto que los mapas son estáticos, se utiliza el paquete **ggplot2**. La Figura 2.13 describe el fragmento de código de la Figura 2.12 asociado a su generación. Por último, se imprime el mapa creado, de modo que **saveGIF** generará una imagen animada como resultado de la impresión de cada figura dentro del bucle.

Figura 2.13. Descripción de las funciones **ggplot**, **geom_sf** y **scale_fill_gradientn**, del paquete **ggplot2** con las que se crean las representaciones que componen los mapas animados.

2.4. Mapas de la Aplicación

En este apartado se detallan los diferentes mapas que se visualizarán en la aplicación. La Figura 2.14 muestra un esquema de la organización de los ficheros HTML e imágenes en formato gif que componen el último bloque de la base de datos (ver Figura 1.8). Para resumir el contenido, se muestra el inicio del nombre que tienen en común varios archivos y después se lista la extensión que caracteriza a cada uno. Las carpetas con nombre CCA o CPRO no son un directorio, sino que indican que en el conjunto de carpetas cada una toma su correspondiente código como nombre.

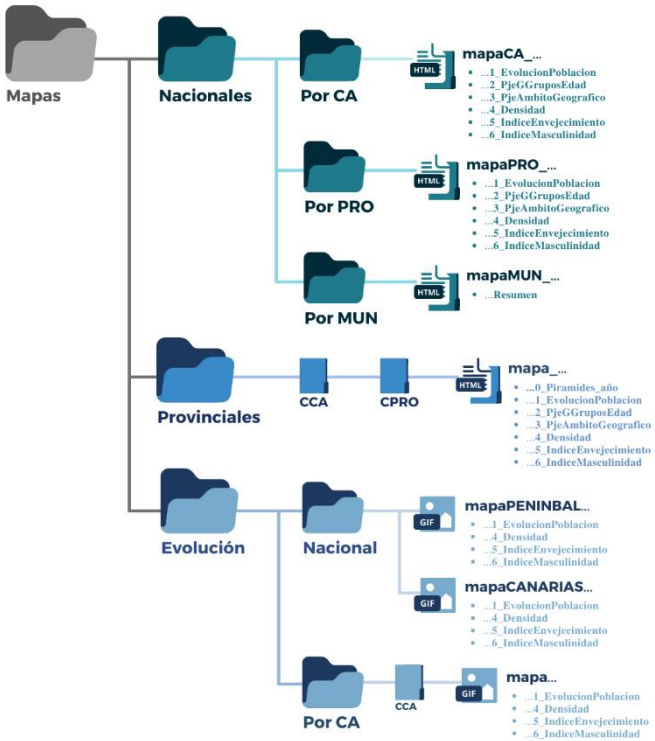


Figura 2.14. Esquema del almacenamiento de los mapas en la base de datos.

2.4.1. Mapas Nacionales

Mapas Nacionales por Comunidades Autónomas

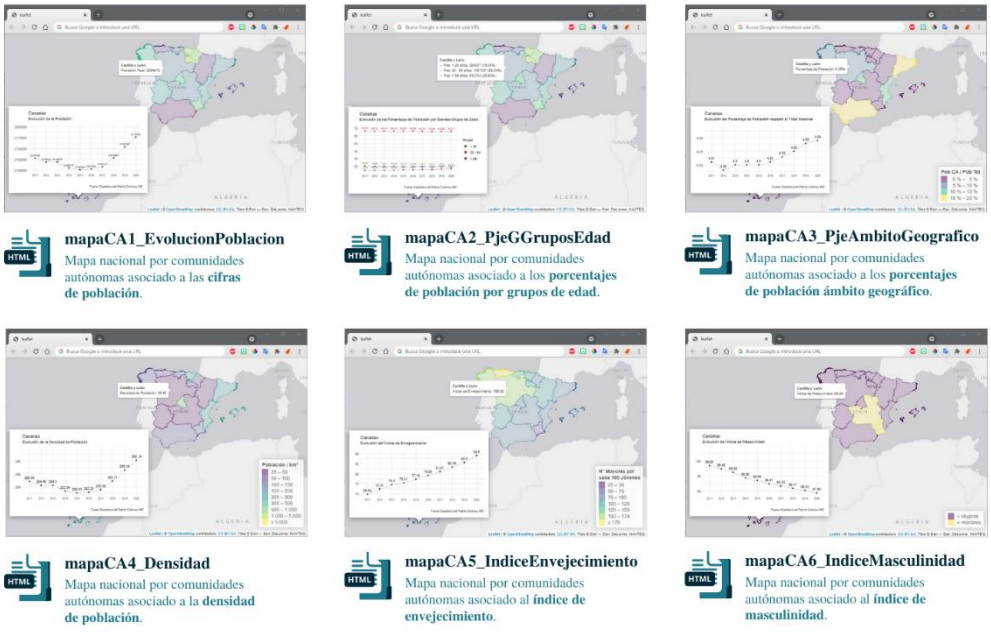


Figura 2.15. Aspecto de los mapas nacionales por comunidades autónomas.

Estos mapas interactivos muestran la evolución de las cifras e indicadores de población de cada comunidad autónoma entre 2011 y 2020 mediante los pop-ups. Sobre el mapa se representan los datos del último año asociados a cada uno, constando en las etiquetas la información exacta de 2020 según el tipo de información que se muestra, a excepción de los dos primeros mapas. Como las diferencias de las cifras de población de entre los diferentes territorios no son notables, se ha optado por considerar colores aleatorios para cada comunidad, por lo que no se especifica una leyenda.

Para hacer más ligero el tiempo de espera e informar de que la renderización de las representaciones puede alargarse durante unos segundos, se ha añadido una imagen en formato gif que se muestra antes de visualizar cada indica que se está cargando el mapa.

Mapas Nacionales por Provincias

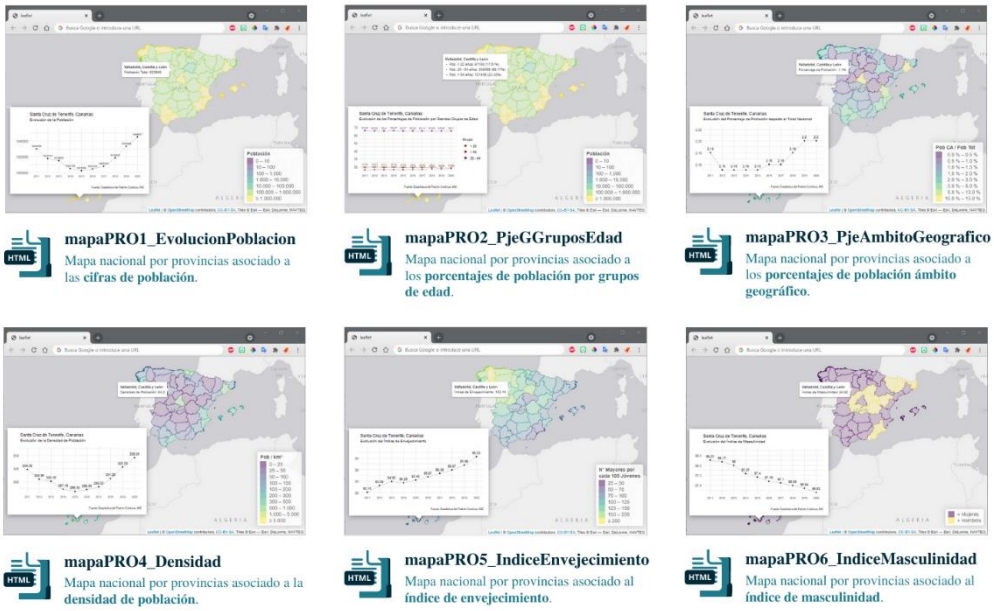


Figura 2.16. Aspecto de los mapas nacionales por provincias.

Los mapas nacionales por provincias son representaciones interactivas del territorio español sobre las que se muestran las cifras e indicadores de población de cada territorio del año 2020. Las etiquetas especifican los valores exactos de dicho periodo, mientras que los pop-ups amplían la información reflejada sobre el mapa mediante gráficos que muestran la evolución de los indicadores desde 2011 hasta 2020.

Al igual que los mapas nacionales por comunidades autónomas, se incluye una imagen que informa al usuario de que el mapa puede tardar en visualizarse.

Mapas Nacionales por Municipios

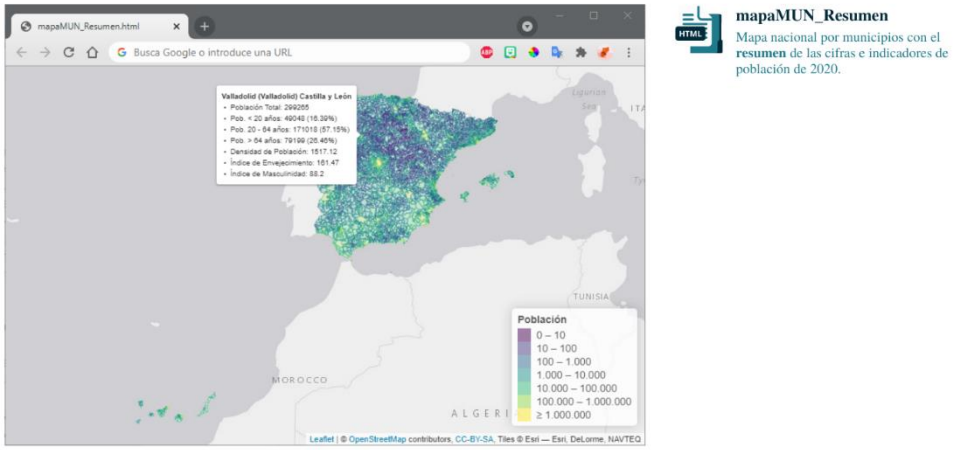


Figura 2.17. Aspecto del mapa nacional por municipios.

A diferencia de los mapas a nivel nacional explicados previamente, la cantidad de información que contendrían las representaciones por municipios si se incluyeran pop-ups excedería los límites de memoria que no permitirían realizar la conversión de los objetos leaflet a formato HTML. Por ello, se ha optado por mostrar en cada etiqueta un resumen de las cifras e indicadores de población del año 2020. Sobre el mapa se representa el número de habitantes dé cada municipio según los datos de ese mismo año.

De nuevo, como en los dos casos anteriores, se incluye una imagen que avisa de que el mapa puede tardar en mostrarse, para que el tiempo de espera hasta la visualización de la aplicación no de la sensación de que se ha podido producir algún tipo de error que haya podido bloquear la aplicación.

2.4.2. Mapas Provinciales

Cifras e Indicadores Municipales

Para mostrar la información detallada de la evolución de las cifras e indicadores a nivel municipal se ha optado por generar mapas de cada provincia. De esta forma, los datos no están concentrados en una única representación, lo que hará que la aplicación sea más dinámica y su contenido esté distribuido en diferentes apartados.

Estas representaciones, expuestas en la Figura 2.18 con el ejemplo de Asturias, muestran la información asociada a 2020 sobre el mapa, indicando con las etiquetas los datos de dicho año, y por medio de los pop-ups se puede ver de la evolución de cada indicador.

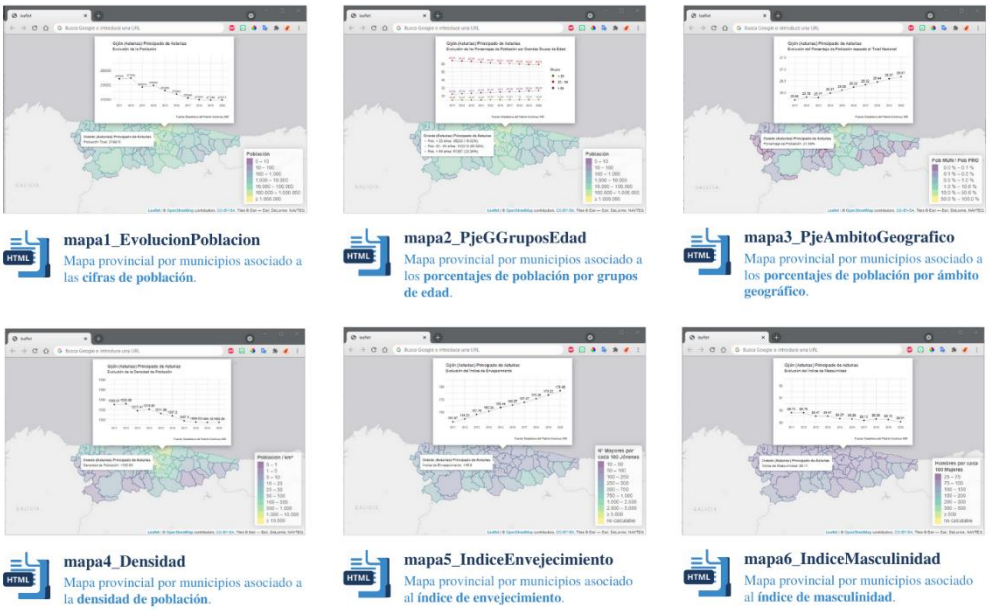


Figura 2.18. Aspecto de los mapas provinciales por municipio de las cifras e indicadores de población.

Pirámides de Población Municipales

Puesto que se dispone de las cifras por edad en grupos quinquenales y sexo anuales, se utilizará esa información para generar pirámides de población con las que se ofrece la posibilidad de conocer su distribución en cada municipio.

A diferencia de los mapas interactivos vistos hasta el momento, en este caso no se muestra sobre el mapa la información asociada al último año del que se disponen datos, sino que cada periodo está representado sobre un mapa distinto, en el que las etiquetas ofrecen un resumen de las cifras e indicadores de dicho año y los pop-ups muestran las pirámides de población de cada uno. La Figura 2.19 muestra el aspecto de los mapas de 2018 a 2020 con el ejemplo de Alicante.

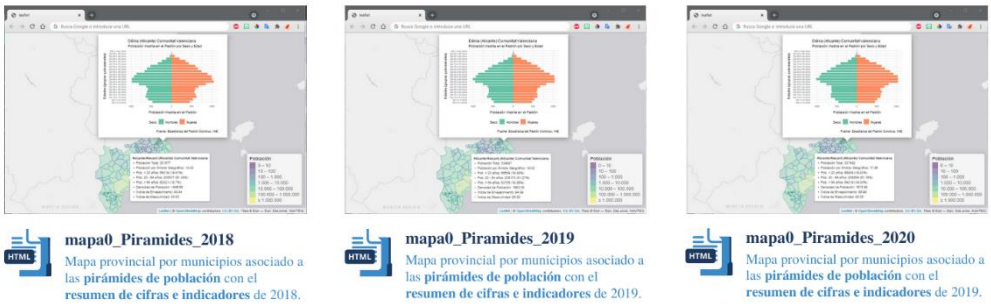


Figura 2.19. Aspecto de los mapas provinciales por municipios de las pirámides de población con los resúmenes anuales de cifras e indicadores.

2.4.3. **Mapas Animados de Evolución**

Mapa Nacional

Los mapas animados nacionales permiten ver la evolución entre 2011 y 2020 de las cifras y los indicadores, concretamente de la densidad de población e índices de envejecimiento y masculinidad, a nivel municipal.

Debido a que la distancia sobre el mapa entre las Islas Canarias y el resto del país es muy grande, las representaciones resultantes al considerar todo el territorio en una misma figura hacen que las imágenes sean demasiado pequeñas. Por este motivo es necesario dividir el territorio en dos mapas que posteriormente se colocaran juntos en la aplicación. Por ello, la imagen del mapa de la península con las Islas Baleares contendrá el título, mientras que el de Canarias solo mostrará los contornos de ese territorio. La Figura 2.20 muestra el aspecto de los mapas combinados en la aplicación.

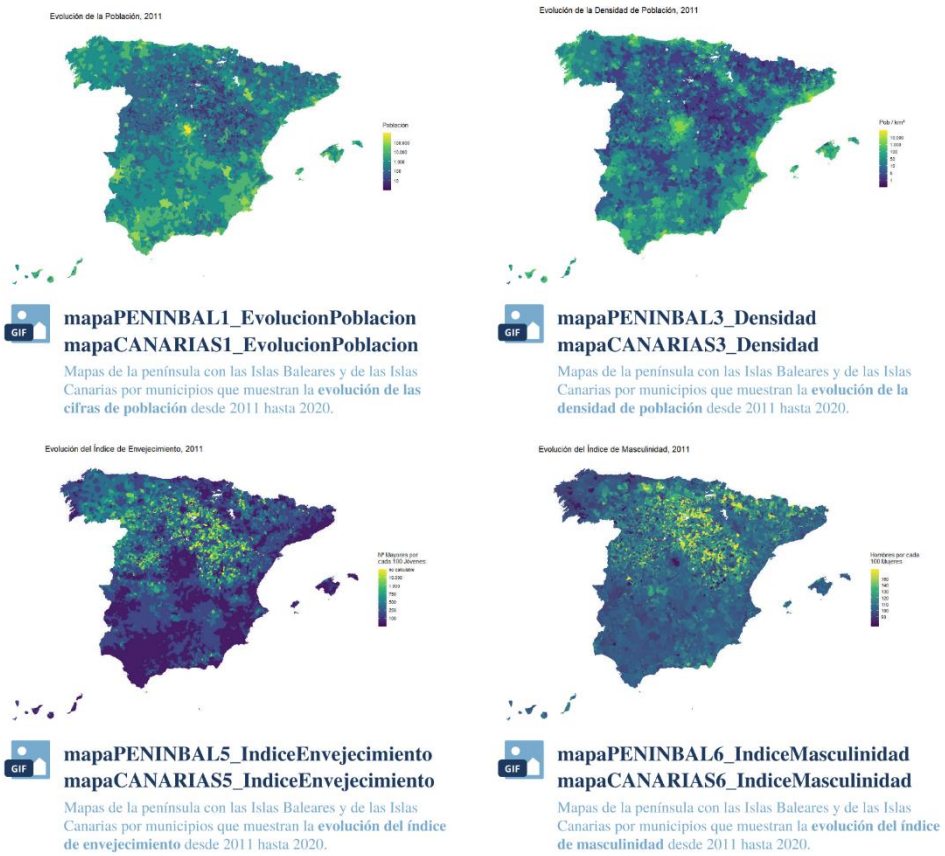


Figura 2.20. Aspecto de los mapas nacionales animados por municipios.

Mapas Autonómicos

Los mapas autonómicos animados ofrecen la posibilidad de ver la evolución a nivel municipal de las cifras e indicadores por cada comunidad autónoma y así poder apreciar mejor los cambios que a nivel nacional se hacen difíciles de ver. La Figura 2.21 muestra el aspecto de este tipo de imágenes tomando como ejemplo Castilla y León.

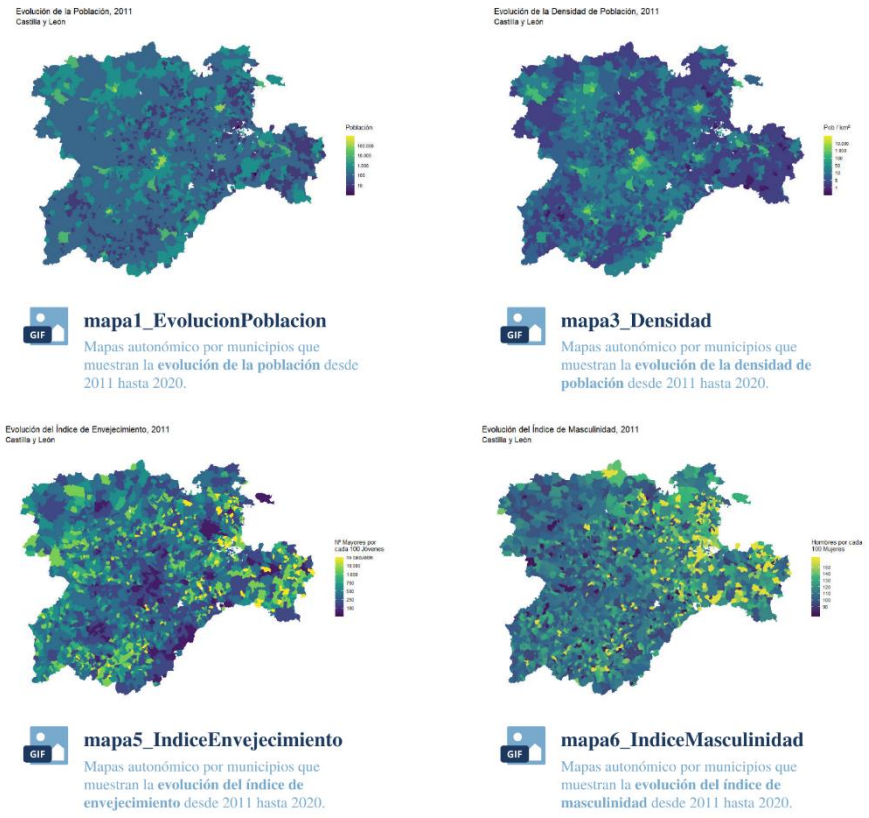


Figura 2.21. Aspecto de los mapas autonómicos animados por municipios.

3. Shiny

Shiny [3.1] es un paquete del software estadístico R [0.1] que ofrece un marco para crear aplicaciones web interactivas. Es una herramienta orientada a científicos de datos que permite desarrollar una interfaz de usuario utilizando código R, sin necesidad de aplicar conocimientos sobre desarrollo web, aunque no se limita al uso de este lenguaje de programación, ya que también es posible usar HTML, CSS o JavaScript para crear dicha interfaz. Shiny utiliza un modelo de programación reactiva que conecta los parámetros de entrada con los de salida y permite controlar el código para prevenir el manejo desordenado de eventos. Además, ofrece una amplia variedad de temas y herramientas para el diseño de la aplicación.

Este capítulo se divide en tres partes, esquematizadas en la Figura 3.1. Para crear una aplicación Shiny es necesario conocer la estructura que debe tener y comprender la división entre el código que genera la interfaz de usuario (*front end*) y el código que genera el comportamiento de la aplicación (*back end*). Por ello, la primera parte se centra en los conceptos básicos para crear aplicaciones y se explica cómo compartirlas. En la segunda, se describe en qué consiste la programación reactiva de Shiny y cuáles son las estructuras reactivas más comunes. La reactividad se refiere a la conexión entre los objetos que recogen la entrada del usuario, *inputs*, y los que visualizan los parámetros de salida, *outputs*, y es un concepto clave para entender el flujo de eventos de una aplicación Shiny. Por último, se ofrece una guía para aprender a manipular el aspecto básico para convertirlo en un diseño atractivo. Esta última parte servirá de introducción al siguiente capítulo, en el que se describe la estructura de la aplicación en la que se centra este trabajo.

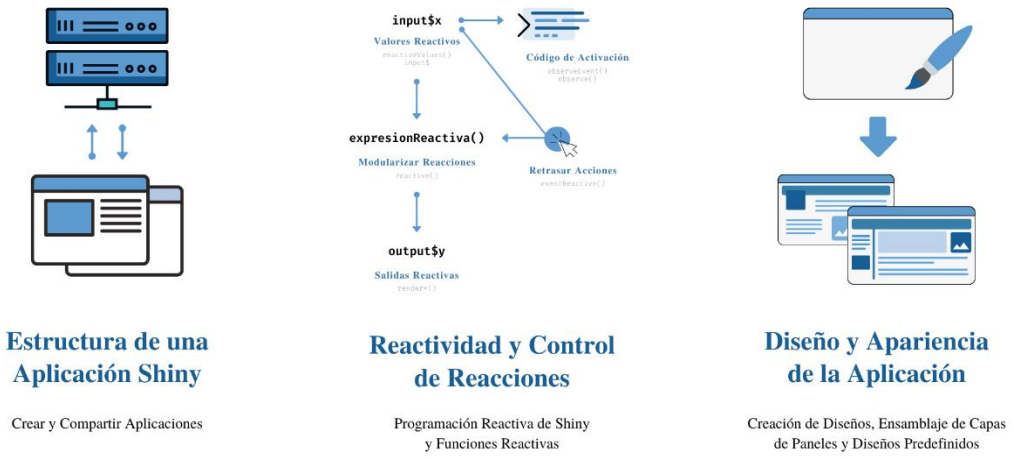


Figura 3.1. Esquema del tercer capítulo: **Shiny**.

Para descargar la versión del paquete Shiny de CRAN, se ejecuta la siguiente línea de código desde la consola de R:

```
install.packages("shiny")
```

Para descargar la versión de GitHub, se usa siguiente fragmento de código:

```
if (!require("devtools"))
  install.packages("devtools")
devtools::install_github("shiny", "rstudio")
```

Una vez instalada, se ejecuta la siguiente línea de código para cargar la librería:

```
library(shiny)
```

El paquete Shiny incluye once ejemplos distintos, así como una función para visualizarlos, de tal modo que se facilite que los usuarios puedan comprender la estructura de una aplicación y aprendan el funcionamiento de Shiny. Para ver el primer ejemplo, bastaría con ejecutar la orden siguiente, indicando el nombre de la aplicación (en caso de no especificar un nombre, se mostrará el listado de los ejemplos).

```
runExample("01_hello")
```

Este primer ejemplo será útil para exponer varias de las ideas que se verán a lo largo del capítulo sobre la estructura y el modelo de programación de Shiny.

3.1. Estructura de una Aplicación Shiny

La primera aplicación de la colección de muestras del paquete Shiny, denominada "*Hello Shiny*" [3.2], es un ejemplo sencillo en el que se representa el conjunto de datos **faithful**[5] mediante un histograma en función del número de barras que el usuario escoja.

[5] **Old Faithful Geyser Data** (faithful) [3.3], es un conjunto de datos integrado en R que recoge los datos de tiempo de duración y espera entre erupciones de géiser Old Faithful, ubicado en el Parque Nacional de Yellowstone, en Wyoming, Estados Unidos.

La parte visual de la aplicación, Figura 3.2, es solo la punta del iceberg, ya que por debajo se encuentra una sesión R abierta que espera a que el usuario interactúe para generar una representación en vista a su elección de parámetros. En un principio, el soporte es el ordenador, ya que es el que ejecuta el código que permite visualizar la aplicación, a pesar de que tenga aspecto de página web. Mediante un servidor, la aplicación sería independiente del ordenador y cualquiera podría tener acceso a ella porque el mantenimiento sería independiente de una máquina específica.

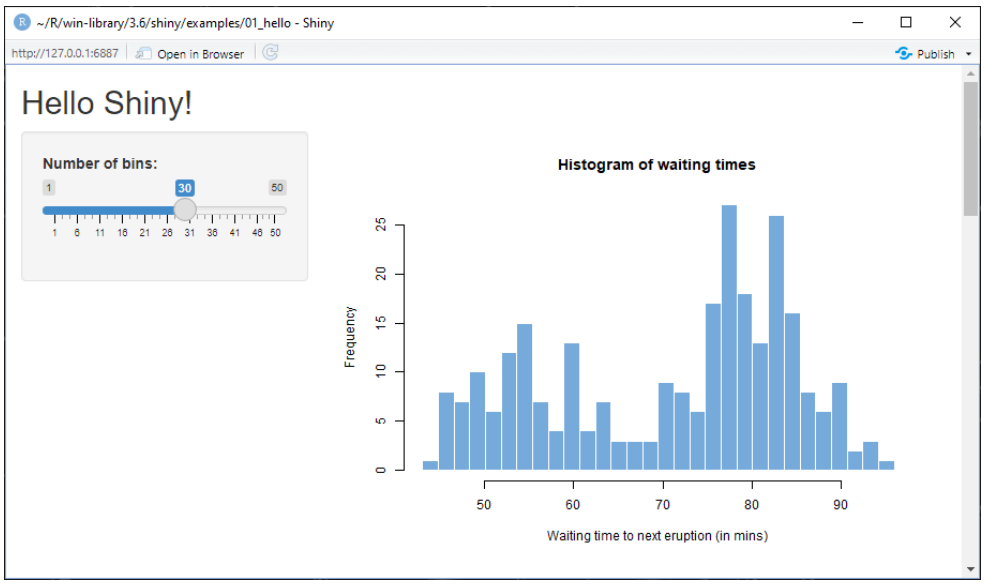


Figura 3.2. Apariencia de la aplicación "*Hello Shiny*", integrada en el paquete Shiny.

Toda aplicación Shiny se divide en dos partes:

- **UI**, *user interface*. Se trata del código que genera la aplicación web que los usuarios podrán visualizar. Esta parte realmente es un documento HTML, aunque no se programe en este lenguaje. De hecho, si se ejecuta directamente desde la consola, lo que se genera es el código HTML.

- **Server**, *server instructions*. Se trata del conjunto de instrucciones escritas en R que el servidor seguirá y serán las que especificarán qué hacer cuando se modifiquen los parámetros de entrada. Estas instrucciones contemplarán cualquier tipo de cambio que el usuario aplique sin necesidad de supervisar el funcionamiento de la aplicación.

La Figura 3.3 es una ayuda visual para facilitar la compresión de la división que hace Shiny entre la **interfaz de usuario** o **UI** (*front end*), donde se define la apariencia de la aplicación, y el **servidor** o **server** (*back end*), parte en la que se especifican las instrucciones que rigen su comportamiento.

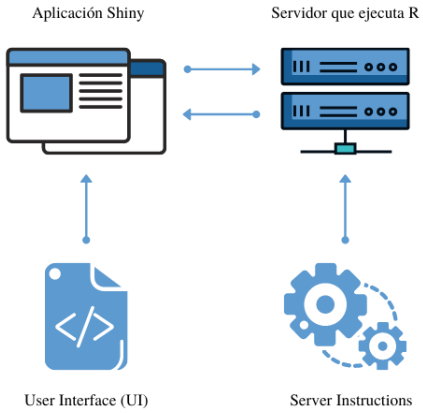


Figura 3.3. Esquema del comportamiento de una aplicación Shiny.

Trasladando esto al ejemplo "*Hello Shiny*", la Figura 3.2 se corresponde entonces con la aplicación Shiny del esquema de la Figura 3.3, mientras que el servidor que ejecuta R es el ordenador. En la Figura 3.4 se muestra el código asociado a este ejemplo, donde se pueden identificar el **UI**, que define el diseño y la distribución de los elementos que componen la aplicación, y el **server**, función que contiene las instrucciones que se necesita ejecutar para generar los histogramas en función del número de barras que el usuario seleccione.

```
library(shiny)

ui <- fluidPage(
  titlePanel("Hello Shiny!"),
  sidebarLayout(
    sidebarPanel(
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)),
    mainPanel(
      plotOutput(outputId = "distPlot")
    )
  )
)

server <- function(input, output) {
  output$distPlot <- renderPlot({
    x    <- faithful$waiting
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, col = "#75AADB", border = "white",
         xlab = "Waiting time to next eruption (in mins)",
         main = "Histogram of waiting times")
  })
}

shinyApp(ui, server)
```

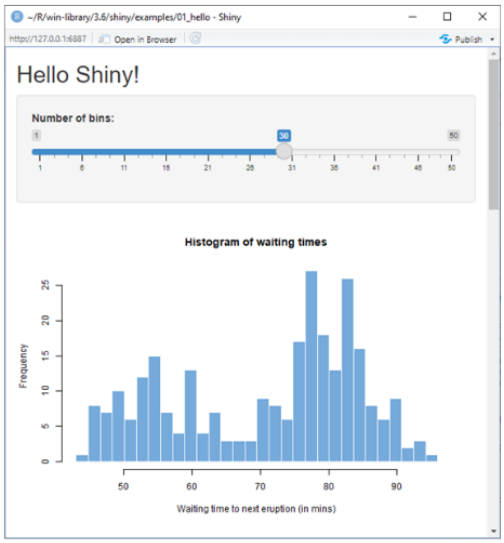


Figura 3.4. Código y apariencia de la aplicación "*Hello Shiny*".

La parte resaltada en azul del código de la Figura 3.4 es la plantilla de cualquier aplicación básica de Shiny. Prescindiendo del código contenido en ella, como se muestra a continuación, se crearía una aplicación vacía, ya que no se estaría especificando qué elementos deben mostrarse ni qué instrucciones se deben seguir. La última línea es la orden que une los dos componentes, UI y server.

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui, server)
```

3.1.1. **UI**. Parámetros de Entrada y de Salida

Las aplicaciones Shiny se construyen en torno a los objetos que recogen los parámetros de entrada, ***inputs***, y los que visualizan la salida, ***outputs***. Estos elementos se definen en la interfaz de usuario, UI, puesto que forman parte del entorno visual.

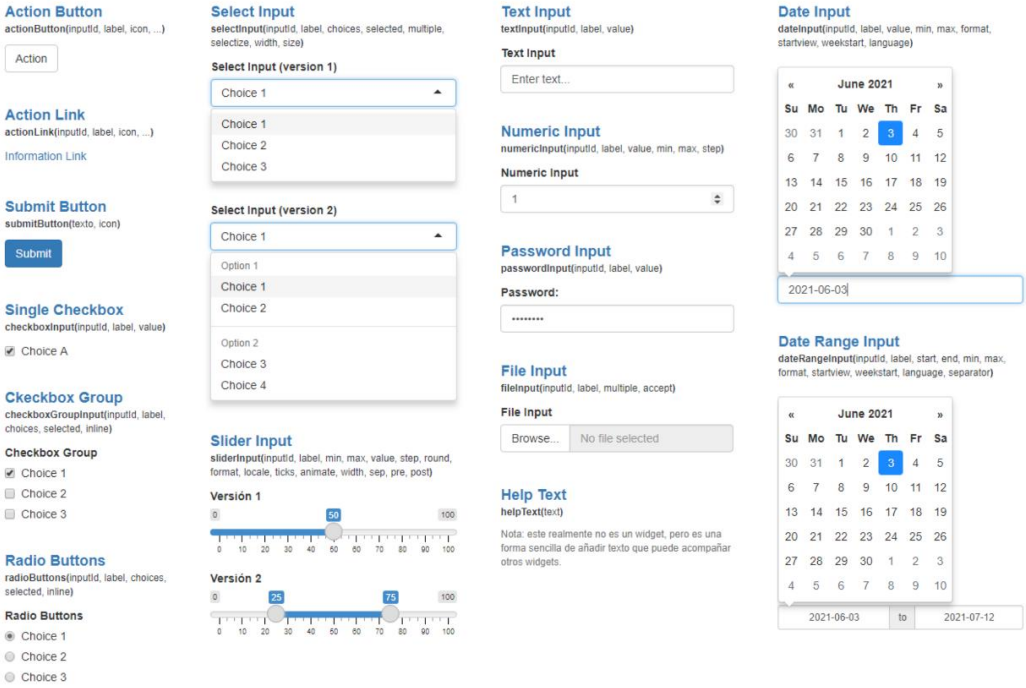


Figura 3.5. Listado de ***inputs***, objetos que recogen los parámetros de entrada.

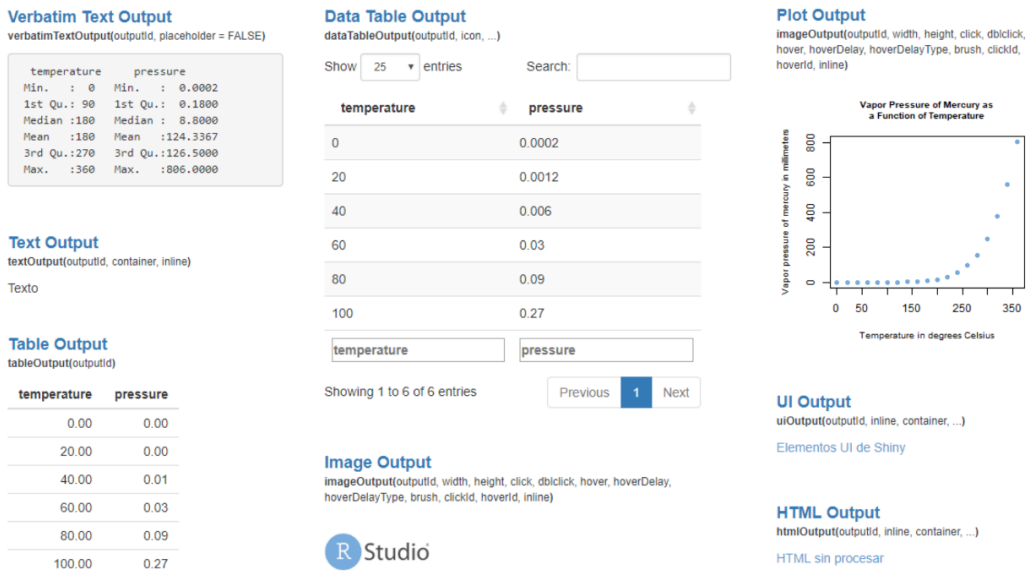


Figura 3.6. Listado de los principales ***outputs***, objetos que visualizan los parámetros de salida.

Inputs

Las funciones que recogen la entrada tienen en común los dos primeros argumentos: **inputId**, que da nombre al parámetro a nivel interno, y **label**, con el que se define el título del elemento en el entorno gráfico, de gran utilidad para indicar el tipo de valor que debe introducirse. También pueden tener otros argumentos, como es el caso de **sliderInput**, función que recoge el parámetro de entrada de "*Hello Shiny*" y que se detalla en la Figura 3.7.

Outputs

Al igual que los inputs, las funciones que muestran los parámetros de salida coinciden en el primer argumento, **outputId**, que indica el nombre del objeto a mostrar. Su sintaxis aclara el tipo de salida que pueden representar. Por ejemplo, **plotOutput**, detallada en la Figura 3.8, sirve para visualizar gráficos, como su propio nombre especifica. Sin embargo, en vista del código, como aún no se ha definido el histograma, solo se está reservando espacio para el objeto en el entorno visual.

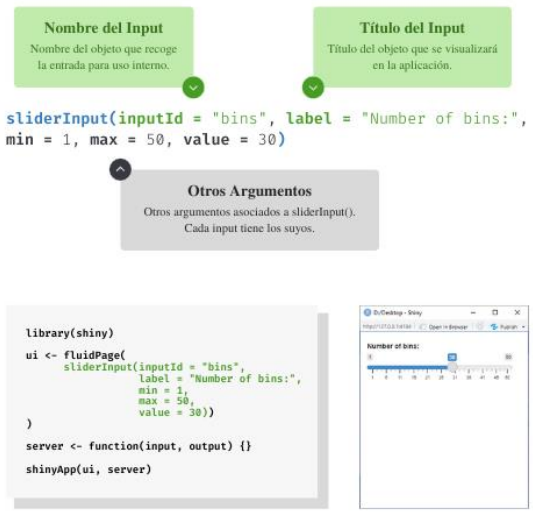


Figura 3.7. Ejemplo de un ***input*** con **sliderInput()**.

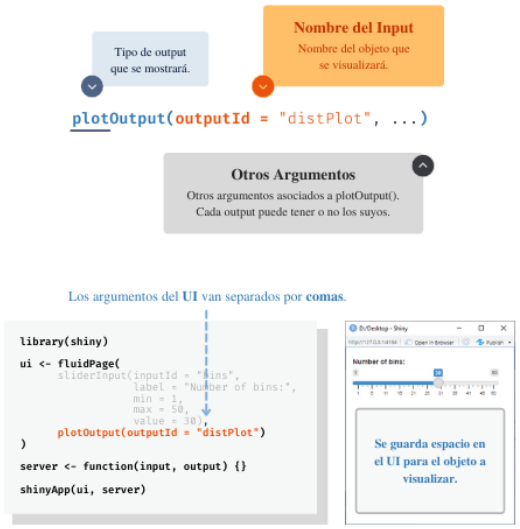


Figura 3.8. Ejemplo de un ***output*** con **plotOutput()**.

3.1.2. **Server**. Crear la conexión entre los *inputs* y los *outputs*

Los outputs se definen en el **server** y en el **UI** se configura su visualización. Para crear los parámetros de salida se deben usar 3 reglas:

Regla 1: Guardar los objetos en **output$**

Figura 3.9. Esquema de la **primera regla** para la creación de un objeto.

La primera regla consiste en que el parámetro de salida debe guardarse en **output$**, acompañado del nombre del objeto, el cual servirá de referencia en el UI a la hora de visualizarlo. En la Figura 3.9 se crea el gráfico para el que previamente se había reservado espacio en la Figura 3.8.

Regla 2: Construir objetos para renderizarlos con una función **render*()**

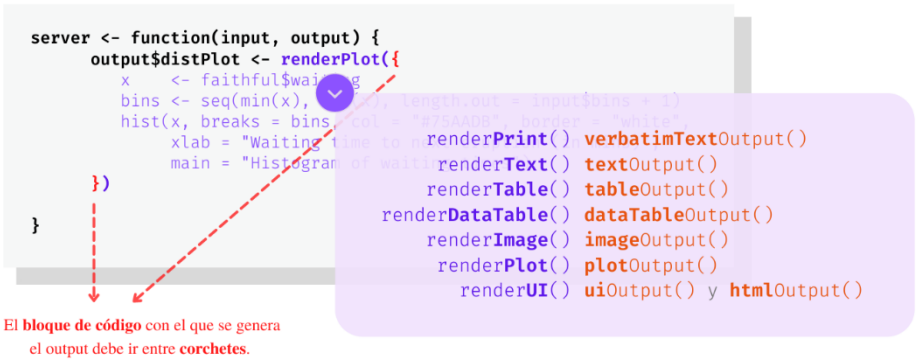


Figura 3.10. Esquema de la **segunda regla** para la creación de un objeto.

Los objetos se renderizan con **funciones de renderización** para poder visualizarlos en la aplicación. Hay una relación directa entre las funciones que visualizan los parámetros de salida, ***Output()**, y las que los renderizan, **render*()**. La sintaxis de ambas ayuda a ver la conexión entre ellas, como se muestra en la Figura 3.10.

En el caso del ejemplo que se está siguiendo, a partir de la aplicación "*Hello Shiny*", la función **renderPlot()**, con la que se renderiza el gráfico, está unida a **plotOutput()**, con la que se representa el histograma. En la Figura 3.10, además de mostrar el listado de funciones render*() y sus respectivas funciones *Output(), se ejemplifica la generación del histograma. Es muy importante tener en cuenta que el bloque de código dentro de la función de renderización debe ir entre corchetes.

Regla 3: Acceso al valor de los inputs mediante **input$**

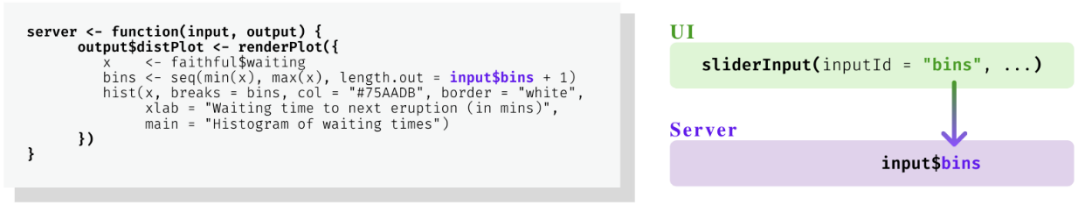


Figura 3.11. Esquema de la **tercera regla** para la creación de un objeto.

Por último, para conectar los parámetros de entrada con los de salida y que los segundos dependan del valor de los primeros, es necesario acceder a los inputs mediante **input$**, como se muestra en la Figura 3.11.

De esta forma, input y output están conectados, ya que cuando cambia la entrada todo lo que está unido a ese valor, en este caso renderPlot(), se actualizará, y el gráfico cambiará instantáneamente para cumplir las especificaciones del nuevo parámetro. A esto se le conoce como **reactividad**.

3.1.3. Guardar y compartir la Aplicación

Guardar Aplicaciones

Hay dos formas de guardar una aplicación Shiny, como se indica en el esquema de la Figura 3.12. El primer método, *un script en un directorio*, se describió anteriormente mediante la Figura 3.4, donde la parte resalada del código en azul daba la plantilla para crear una aplicación en un único archivo. La segunda versión consiste en dividir UI y Server en dos scripts dentro de un mismo directorio.

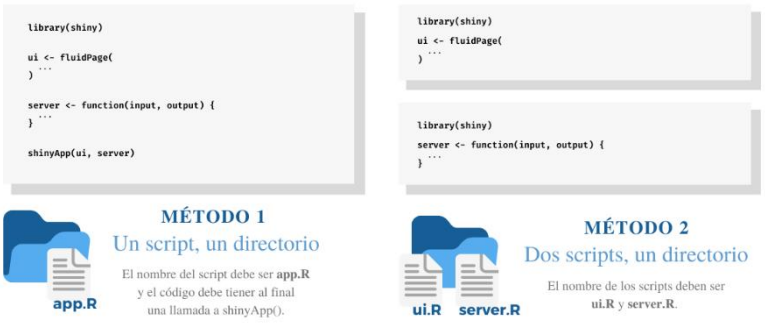


Figura 3.12. Esquema los dos métodos para guardar una aplicación Shiny.

Adicionalmente, la carpeta que contiene el código de la aplicación puede incluir otras dentro de ella que contengan datos o imágenes, como se verá posteriormente.

Compartir Aplicaciones con Shinyapps.io

Shinyapps.io [3.4] es una plataforma que permite alojar aplicaciones Shiny de forma gratuita aunque con ciertas limitaciones. Gracias a esta herramienta es posible compartir aplicaciones, debido a que dejan de ser dependientes del ordenador. Sin embargo, no es posible hacer uso de este servidor con versiones de R posteriores a la 4.0.0.

A continuación, se explican los pasos a seguir para crear una cuenta en **shinyapps.io** y subir aplicaciones a la plataforma:

En primer lugar, es necesario instalar y cargar el paquete **rsconnect** [3.5]:

```
install.packages("rsconnect")
library(rsconnect)
```

El segundo paso es crear una cuenta en **shinyapp.io**. Para nuevos usuarios, se muestra una guía inicial en la que se indica que, tras haber instalado y cargado la librería rsconnect, se debe copiar y pegar el fragmento de código que autoriza la instalación del paquete **shinyapps**.

Por último, para publicar una aplicación basta con pulsar el botón **Publish** que aparece en la esquina derecha superior de la ventana en la que se muestra la aplicación al ejecutarla, como se puede ver en la Figura 3.13. De esta forma, quedará almacenada en la nube y otras personas podrán acceder a ella.

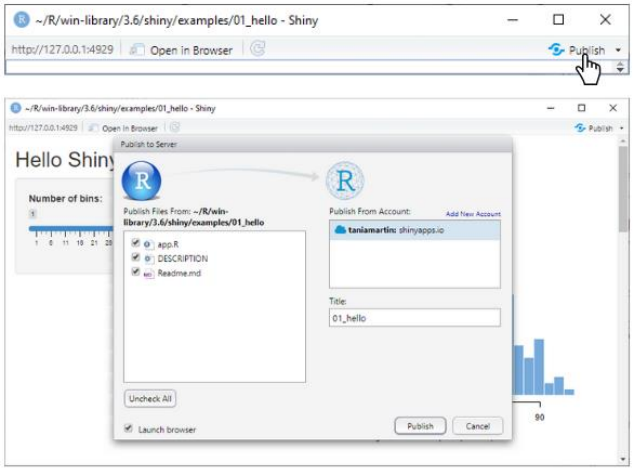


Figura 3.13. Publicación de una aplicación Shiny con **shinyapps.io**.

Construir un Server

Es posible crear un servidor mediante la herramienta gratuita **Shiny Server**, aunque también tiene sus limitaciones. Como se describe en la página web del producto [3.6], Shiny Server permite a los usuarios alojar y administrar aplicaciones Shiny en internet con su propia dirección web. En el documento Shiny Server Professional Administrator's Guide [3.7] se detalla toda la información necesaria para la instalación y gestión de aplicaciones con Shiny Server, incluyendo las especificaciones para la versión profesional.

En el caso de la aplicación en la que se centra este trabajo, se usa un servidor cuyas características se detallan en el siguiente capítulo (ver 4.1).

3.2. Reactividad y Control de Reacciones

En la primera parte de este capítulo (ver 3.1) se ha explicado cómo crear, guardar y compartir aplicaciones Shiny. Se describieron las partes que las forman, UI y Server, y los elementos que les dan sentido, inputs y outputs, así como las funciones que recogen los parámetros de entrada y las que renderizan y visualizan los de salida. Además, se hizo mención al concepto **reactividad**, del que trata íntegramente esta parte del capítulo, dada su importancia, ya que es el paradigma con el que se programa el comportamiento de las aplicaciones.

La **reactividad** se refiere a la conexión entre los input y los outputs, por la que al cambiar el parámetro de entrada se modifica instantáneamente el de salida en función del valor introducido. Sin embargo, la reactividad en Shiny es mucho más sofisticada que simplemente inputs y outputs.

Hay diferentes estructuras reactivas en función del tipo de comportamiento que se quiera aplicar. La Figura 3.14 muestra un esquema de las diferentes posibilidades. Por ejemplo, dependiendo del parámetro de entrada puede ejecutarse un fragmento de código específico por medio de *código de activación*. También se puede controlar la reactividad mediante el uso de botones para *retrasar acciones*, de tal forma que al pulsarlo se esté indicando que se debe modificar el output porque se ha modificado el input. Incluso se puede retrasar el proceso de reactividad *modularizando reacciones* para que así los cambios en la entrada lleven a crear objetos secundarios y que estos apliquen ciertas modificaciones sobre la salida.

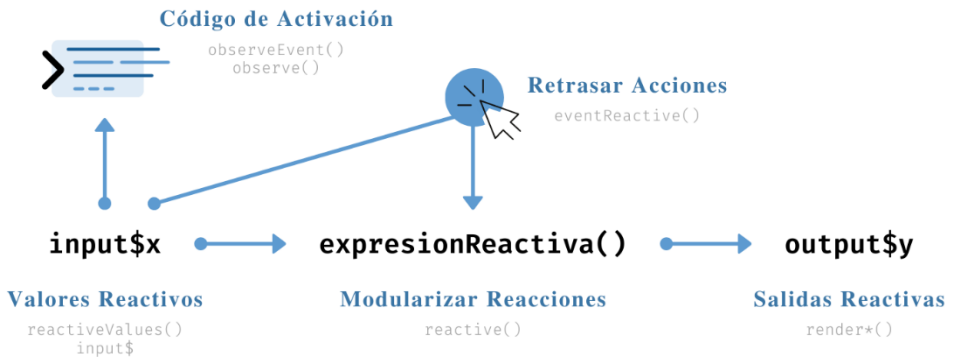


Figura 3.14. Esquema de las funciones reactivas.

3.2.1. Valores Reactivos

Como se describió mediante el ejemplo de la construcción de la aplicación "*Hello Shiny*" en la primera parte, la tercera regla para crear outputs (ver 3.1.2) especificaba que para poder acceder a los parámetros de entrada es necesario acceder a su valor mediante **input$**. De este modo, **input$bins**, en el esquema de la Figura 3.11, es el **valor reactivo** de ese ejemplo.

Un **valor reactivo** es un objeto que cambia/reacciona a un tipo de input, a diferencia del resto de parámetros que forman parte de la aplicación, los cuales son estáticos y solo varían cuando un fragmento de código lo especifica. Los valores reactivos no puede ser utilizados fuera de las **funciones reactivas**. Estas dos ideas, **valores** y **funciones reactivas**, trabajan de forma conjunta para crear los parámetros de salida que se muestran en la aplicación, como se muestra en la Figura 3.15.

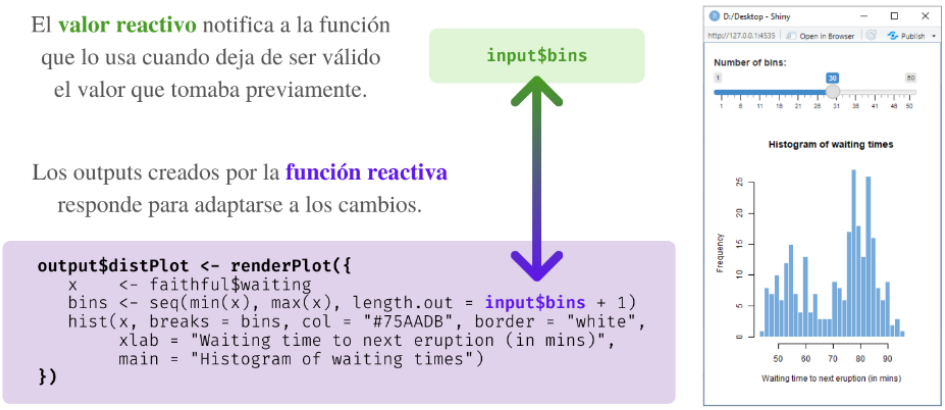


Figura 3.15. Reactividad como un proceso de dos pasos. Conexión entre **valores** y **funciones reactivas**.

3.2.2. Funciones Reactivas

Dentro de las **funciones reactivas** se utiliza un fragmento de código para construir (y reconstruir) las salidas, el cual responderá a los cambios de los valores reactivos. Las **estructuras reactivas** que se verán a continuación condicionan el comportamiento de la aplicación, adecuándolo al tipo de respuesta que se espera de ella.

Mostrar una salida con render*()

Las funciones **render*()** renderizan los objetos a visualizar en la aplicación (ver su esquema en la Figura 3.16). En la Figura 3.10 se puede ver el listado de funciones, así como un ejemplo de construcción de una **salida reactiva**. El código dentro de la función reactiva se ejecuta como una única unidad, por lo que cada modificación sobre el parámetro de entrada implica que se ejecute todo el bloque de código.

Figura 3.16. Esquema de la reactividad simple que implica un valor y una salida reactivos.

Esta estructura reactiva es útil en casos como el ejemplo "*Hello Shiny*", debido a la sencillez de esta aplicación y el conjunto de datos con el que se trabaja. Sin embargo, resulta ser un inconveniente en casos como el que se muestra en la Figura 3.17.

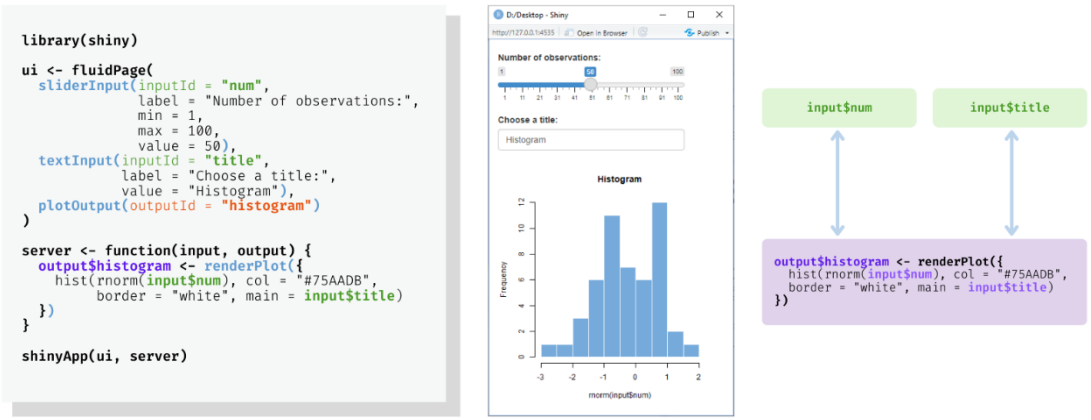


Figura 3.17. Código, apariencia y esquema de una aplicación con dos valores reactivos dentro de una misma función reactiva.

En este ejemplo, en vez de usar un conjunto de datos concreto, se genera una muestra aleatoria en función del número de observaciones que el usuario escoja para visualizar un histograma, cuyo título también podrá definir. Al modificar el número de observaciones se ejecuta todo el bloque de código contenido en la función reactiva, creándose un nuevo histograma, ocurriendo lo mismo al cambiar el título, con lo que se genera una nueva muestra igualmente, cuando lo que se busca es simplemente modificar un parámetro del gráfico, no del conjunto de datos con el que se trabaja. Por ello, es necesario **modularizar el código** para prevenir que la modificación en uno de los dos parámetros de entrada afecte a la salida.

Modularizar el Código con reactive()

Para solventar el problema que se planeta en la aplicación de la Figura 3.17 se hace uso de la función **reactive()** (ver su esquema en la Figura 3.18), con la que se crea un objeto reactivo denominado **expresión reactiva**.

Figura 3.18. Esquema la estructura reactiva para *modularizar código* con reactive().

En el ejemplo de la Figura 3.19 se hace uso de reactive() para almacenar el conjunto de datos en una expresión reactiva. Para acceder al contenido de los datos se hace una llamada a al objeto reactivo como si fuera una función: **data()**. De esta forma, la muestra que depende del número de observaciones que el usuario escoja no se verá afectada por los cambios que se puedan aplicar sobre el título del gráfico.

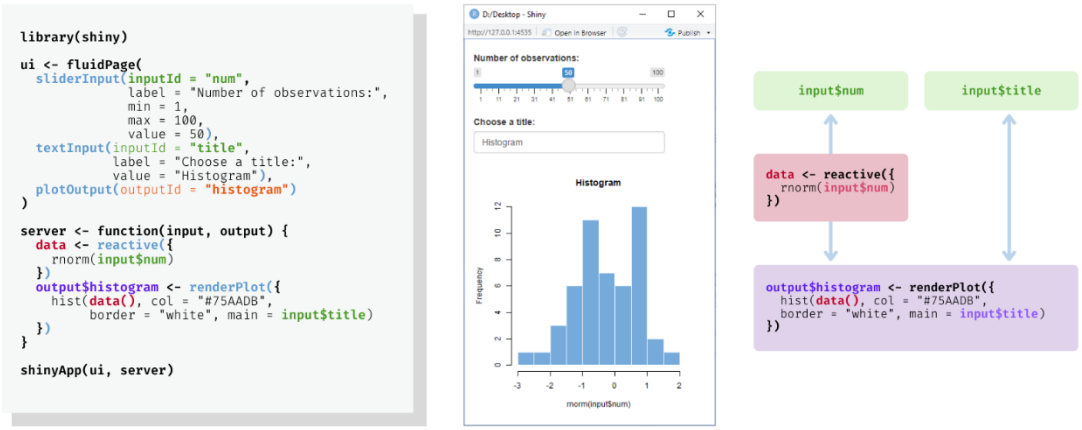


Figura 3.19. Código, apariencia y esquema de una aplicación con modularización de código.

Prevenir Reacciones con isolate()

Existe otra solución al problema planteado en la Figura 3.17 que consiste en aislar la reacción asociada al cambio del título del gráfico. Con **isolate()** se controla la reactividad convirtiendo un objeto reactivo en estático para que los cambios de dicha entrada no afecten a otras partes de la aplicación, como se muestra en la Figura 3.20.

Figura 3.20. Código, apariencia y esquema de una aplicación con control de reactividad.

De esta forma, cuando se cambia el título no se observará ningún cambio. A nivel interno, el nuevo valor se almacenará en su correspondiente input y cuando se apliquen cambios sobre otro parámetro de entrada, en este caso modificando el tamaño de la muestra, el nuevo título se visualizará. Su valor se ha aislado, por lo que no cambia el histograma instantáneamente como antes, sino que espera a que otro input obligue a ejecutar de nuevo el bloque de código contenido en la función reactiva que renderiza el gráfico para aplicar las modificaciones en el título.

Código de Activación con observeEvent() y observe()

Es posible activar cierto código en base a un parámetro de entrada utilizando las funciones **observeEvent()** y **observe()** (ver su esquema en la Figura 3.21). La sintaxis de la segunda función es igual a la de las funciones vistas hasta este punto, mientras que la de la primera incorpora un nuevo argumento, como se muestra en la Figura 3.22. Este parámetro es el valor reactivo que actúa como un interruptor para activar la función reactiva. El código contenido en ella puede depender de otros valores reactivos distintos al que da la orden de ejecución, pero los cambios aplicados sobre ellos no harán que la función reaccione. En definitiva, la función trata al código como si estuviera aislado por isolate() y solo el input que la activa permite acceder al él.

El valor reactivo al que reacciona observeEvent() es la acción que se realiza al pulsar el *botón de acción* de los inputs **actionButton()** o **submitButton()**, incluidos en el listado de funciones de la Figura 3.5.

Figura 3.21. Esquema la estructura reactiva para usar *código de activación* mediante el uso de observe() y observeEvent().

El ejemplo de la Figura 3.22 consiste en una aplicación que únicamente muestra un botón. El input que se recoge con actionButton() se va actualizando a medida que el usuario pulsa sobre el objeto y cada uno de los cambios sobre el parámetro de entrada activa el código contenido en observeEvent(), el cual indica que debe imprimirse el número de clics que se almacenan en el valor reactivo. Sin embargo, no se está especificando que se muestre el número de clics en la aplicación porque no se utiliza ninguna función de renderización, sino que se está indicando que se debe imprimir por consola el número de veces que se pulsa sobre el botón.

Con esto se pretende hacer entender que la función **observeEvent()** no está pensado para trabajar con parámetros de entrada cuyo valor sea significativo, sino que provoquen cambios a los que responder. Las acciones no deben depender de un valor específico.

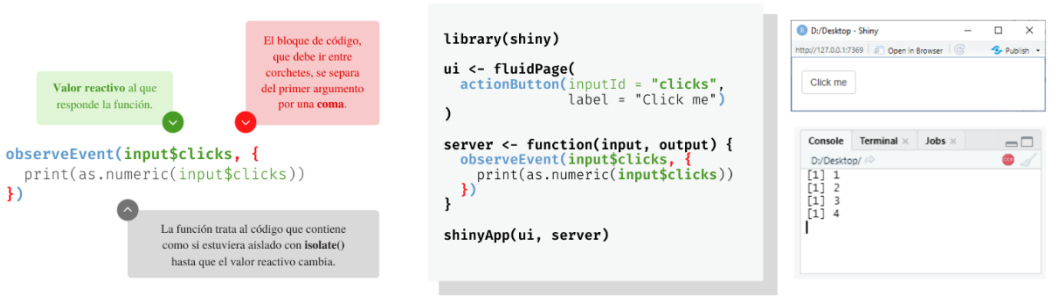


Figura 3.22. Esquema de observeEvent() acompañado del código y la apariencia de una aplicación con código de activación.

Retrasar Reacciones con eventReactive()

Volviendo al problema del ejemplo de la Figura 3.17, otra forma de prevenir cambios indeseados al actualizar parámetros de entrada es combina la *modularización de código*, vista con la aplicación de la Figura 3.19, con el *código de activación*, explicado en el punto anterior (ver su esquema en la Figura 3.23).

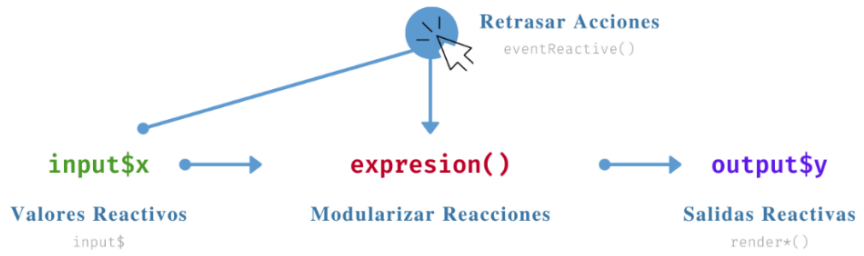


Figura 3.23. Esquema de la estructura reactiva para *retrasar acciones* con eventReactive().

Para ello, se utiliza la función **eventReactive()**, con la que se retrasa una reacción haciendo uso de *botones de acción* con **actionButton()** o **submitButton()**, evitando que el código no se ejecute hasta que el usuario no lo pida explícitamente. Esto marca la diferencia entre la aplicación de la Figura 3.19, en la que se usa reactive(), y la de la Figura 3.24. Mientras que en la primera se visualiza directamente el histograma al iniciar la aplicación, en la segunda es necesario pulsar el botón para que se muestre el gráfico, debido a que el código contenido en la función eventReactive() no se activará hasta que el valor reactivo no lo indique.

Figura 3.24. Esquema de eventReactive() acompañado del código y la apariencia de una aplicación con retraso de acciones.

Gestionar Estados con reactiveValues()

Shiny no permite definir los valores reactivos en el código, son parámetros que dependen de la elección del usuario. Sin embargo, con la función **reactiveValues()** se pueden crear objetos para almacenar esos valores, cuya estructura es similar a la de una lista, pero con capacidades especiales para la programación reactiva.

En la Figura 3.25 se muestra un nuevo ejemplo que consiste en una aplicación que visualiza un histograma de una muestra aleatoria generada en función de la distribución elegida por el usuario. El objeto, **obj**, generado e inicializado con reactiveEvent(),

contiene un elemento denominado **data**, donde se almacenará el conjunto de datos, al que se accede con **obj$data**. Con observeEvent(), que reacciona a la acción de pulsar sobre uno de los dos botones, creados con actionButton(), se define cuál de las dos muestras debe tomar el objeto reactivo para generar el histograma.

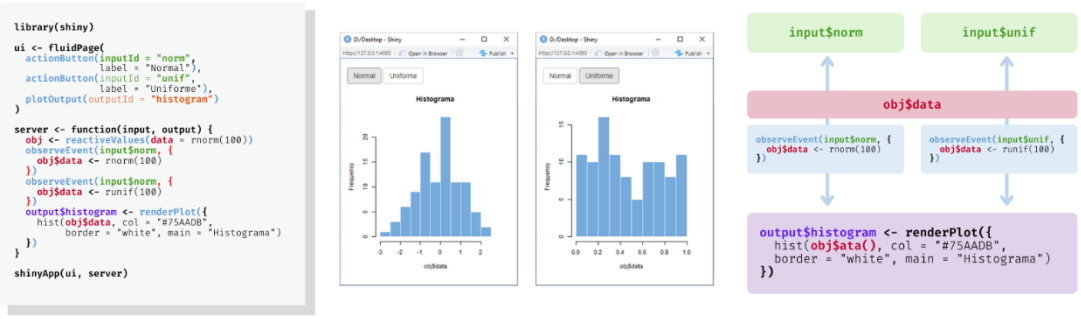


Figura 3.25. Código, apariencia y esquema de una aplicación con gestión de estados

3.3. Diseño y Apariencia de la Aplicación

Para convertir cualquiera de los ejemplos que se han visto hasta el momento en aplicaciones visualmente atractivas, es necesario conocer las herramientas que ofrece Shiny para manipular su aspecto básico.

3.3.1. Añadir Contenido Estático

Hasta ahora se ha descrito cómo trabajar con el contenido reactivo de la aplicación, pero la apariencia depende además de otros elementos. Como ya se mencionó en la primera parte de este capítulo, el UI es un documento HTML, por lo que haciendo uso de este lenguaje se puede incorporar contenido estático al entorno gráfico. Shiny incluye una función especial, denominada **tags** [3.8], que sirve para convertir el código R a HTML. Esta función, a su vez, contiene un lista de funciones para cada tag que puede crearse. En la Figura 3.26 se muestra el listado de tags. Aquellos que aparecen resaltados son los tags más comunes, que tienen su propia función para evitar hacer la llamada **tag$** para acceder a ellos.

En la Figura 3.27 se resumen varios de los tags más comunes, los cuales sirven para *crear secciones* (**h1**, …, **h6**), *agrupar contenido* (**p**, **hr**), editar la *semántica del texto* (**em**, **strong**, **br**, **code**, **a**) y añadir *contenido ilustrado* (**img**).

```
tags$a            tags$data         tags$h6           tags$nav          tags$span
tags$abbr         tags$datalist     tags$head         tags$noscript     tags$strong
tags$address      tags$dd           tags$header       tags$object       tags$style
tags$area         tags$del          tags$hgroup       tags$ol           tags$sub
tags$article      tags$details      tags$hr           tags$optgroup     tags$summary
tags$aside        tags$dfn          tags$HTML         tags$option       tags$sup
tags$audio        tags$div          tags$i            tags$output       tags$table
tags$b            tags$dl           tags$iframe       tags$p            tags$tbody
tags$base         tags$dt           tags$img          tags$param        tags$td
tags$bdi          tags$em           tags$input        tags$pre          tags$textarea
tags$bdo          tags$embed        tags$ins          tags$progress     tags$tfoot
tags$blockquote   tags$eventsource  tags$kbd          tags$q            tags$th
tags$body         tags$fieldset     tags$keygen       tags$ruby         tags$thead
tags$br           tags$figcaption   tags$label        tags$rp           tags$time
tags$button       tags$figure       tags$legend       tags$rt           tags$title
tags$canvas       tags$footer       tags$li           tags$s            tags$tr
tags$caption      tags$form         tags$link         tags$samp         tags$track
tags$cite         tags$h1           tags$mark         tags$script       tags$u
tags$code         tags$h2           tags$map          tags$section      tags$ul
tags$col          tags$h3           tags$menu         tags$select       tags$var
tags$colgroup     tags$h4           tags$meta         tags$small        tags$video
tags$command      tags$h5           tags$meter        tags$source       tags$wbr
```

Figura 3.26. Listado de **tags**.

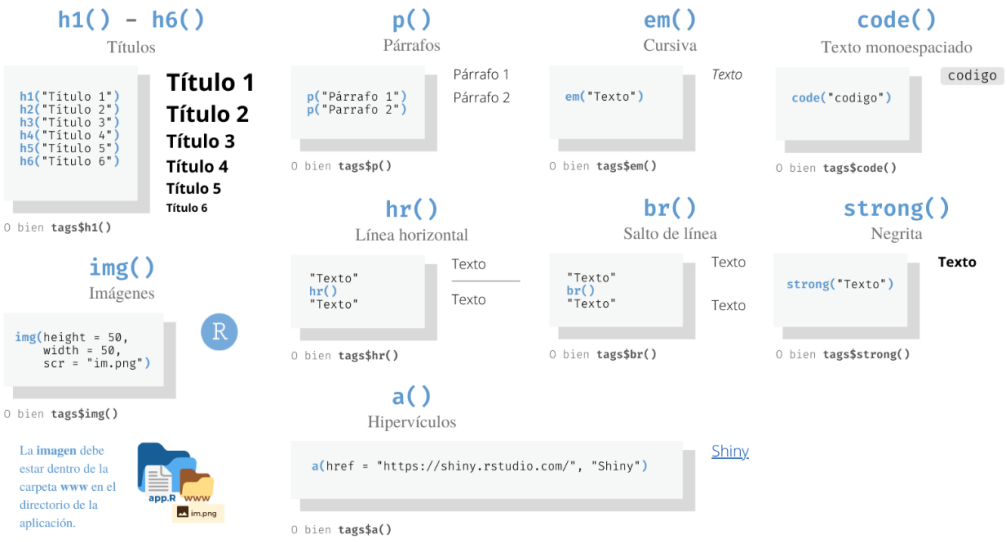


Figura 3.27. Esquema de los principales tags más comunes.

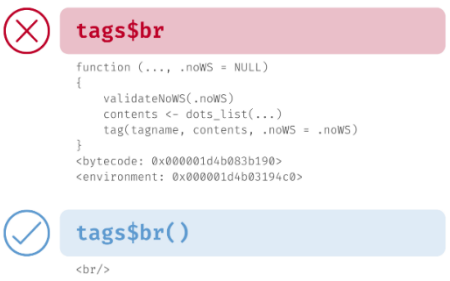


Figura 3.28. Uso de los tags.

Es importante tener en cuenta que cada tag es una función, por lo que es necesario añadir paréntesis al final. De lo contrario, se mostrará el contenido de la función, como se puede ver en la Figura 3.28. Parece obvio cuando se usan tags con argumentos, pero en el caso de **hr** y **br**, descritos en la Figura 3.27, puede llevar a errores de ejecución.

3.3.2. **Diseño del Entorno Visual**

El diseño puede verse como un marco en dos dimensiones, de tal forma que colocar un elemento en la aplicación se reduce a escoger las coordenadas en las que situarlo. También puede añadirse una tercera dimensión incorporando capas, como se verá posteriormente.

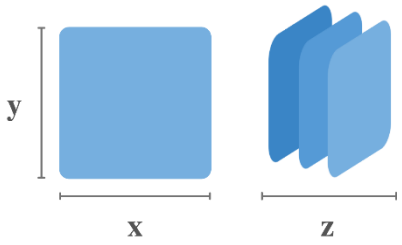


Figura 3.29. Perspectiva del entorno gráfico en dos y tres dimensiones.

Desde la perspectiva 2D, el entorno visual de la aplicación se convierte en una cuadrícula de 12 columnas y número indeterminado de filas, definida por el **sistema de retículas** [3.9], herramienta de diseño gráfico que permite estructurar y jerarquizar los contenidos de una página web. Este sistema está diseñado de tal manera que el diseño de la aplicación sea capaz de reaccionar a los cambios en las dimensiones de la pantalla para distribuir adecuadamente el contenido en función del tamaño del entorno o del dispositivo desde el que se visualice la página.

Con **fluidPage()**, función con la que se genera el UI, los elementos se redistribuyen para llenar el ancho disponible de la página de navegación, pero con **fixedPage()** se crea un entorno fijo con tres tamaños por defecto: 724, 940 o 1170 pixeles de ancho, dependiendo del espacio en el que se visualice la aplicación.

Para hacer particiones de la pantalla se usan **fluidRow()** y **column()**, dos funciones que devuelven **divs**, divisiones de HTML, que sirven para crear divisiones y agrupar contenido. En la Figura 3.30 se muestra un esquema de su uso y comportamiento. Con fluidRow() se divide el entorno en filas y se trabaja junto con column() para organizar cada una. Para colocar un elemento dentro de cada módulo simplemente se debe escribir el código asociado al objeto dentro del espacio en el que se quiere situar.

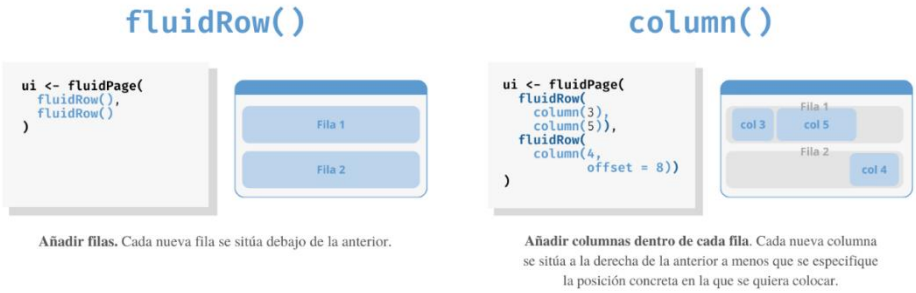


Figura 3.30. Esquema de fluidRow() y column().

Para generar filas con fluidPage() se usa fluidRow() y con fixedPage() es más adecuado utilizar fixedRow().

3.3.3. Paneles y Ensamblaje de Capas

Los **paneles** [3.9] son los elementos básicos de agrupación en Shiny. Los objetos situados en un panel actúan como un conjunto, de tal forma que esa unidad tendrá propiedades por sí misma que no afectarán a otras partes de la aplicación.

Hay 12 funciones distintas, descritas en la Figura 3.31, con las que se puede crear diferentes tipos de paneles. Tres de ellas, concretamente **tabPanel()**, **tabsetPanel()** y **navlistPanel()**, son destacables porque involucran capas que se solapan, transformando el entorno visual en un marco tridimensional.

Figura 3.31. Listado de paneles y descripción de cada uno.

La función **tabPanel()** sirve para aplicar capas de elementos. Cada panel es independiente del resto, de modo que se puede estructurar el diseño de cada uno sin que afecte a los demás. Deben tener un título que sirva para diferenciarlos e identificarlos, dado que este componente es esencial en la navegación entre capas.

Los paneles está diseñados para poder trabajar con **tabsetPanel()** y **navlistPanel()**. En la Figura 3.32 muestra el esquema de ensamblaje de paneles con estas dos funciones.

Con **tabsetPanel()** se combinan **tabs**, capas creadas con tabPanel(), sobre un panel que hace de fondo, como se muestra en la ilustración de la izquierda de la Figura 3.32, de modo que se navega entre capas pulsando sobre las etiquetas de la barra superior de la aplicación. En cuanto a **navlistPanel()**, utiliza la navegación por barra lateral, por lo que el menú de tabs se encuentra en el lado izquierdo de la aplicación.

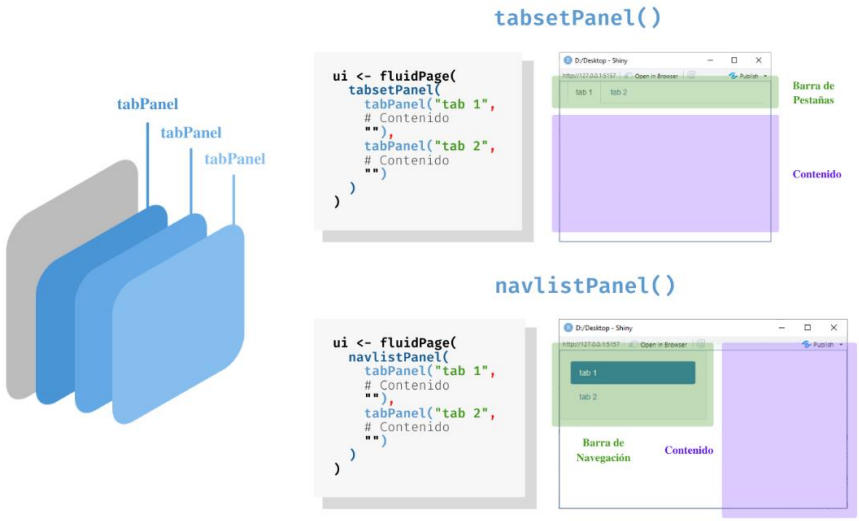


Figura 3.32. Esquema de ensamblaje de paneles con **tabsetPanel()** y **navlistPanel()**.

3.3.4. Shiny Dashboard

Shiny Dashboard [3.10] es una herramienta que facilita la creación de un diseño muy atractivo y organizado. No está incluido en el propio paquete de Shiny, sino que cuenta con su propia librería, que además incluye elementos visuales especiales.

Para instalar y cargar el paquete **shinydashboard** basta con ejecutar el siguiente fragmento de código:

```
install.packages("shinydashboard")
library(shinydashboard)
```

La estructura del diseño Shiny Dashboard [3.11] consta de tres partes, como se puede ver en la Figura 3.33. El entorno de UI se genera mediante **dashboradPage()** y se usan tres funciones distintas para crear cada uno de los componentes de la aplicación.

Figura 3.33. Código y apariencia del diseño predeterminado de Shiny Dashboard.

Notar que la estructura del diseño de shiny dashboard es similar a la que se obtiene mediante el ensamblaje de capas con navlistPanel() (ver Figura 3.32).

Encabezado

El **encabezado** se genera con la función **dashboardHeader()**. Éste puede contener el *título* de la aplicación, que se sitúa en la parte derecha sobre la barra de navegación, como se puede ver en la Figura 3.34, y *menús desplegables*, en la parte izquierda.

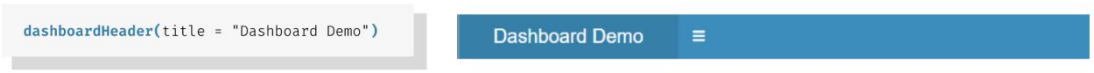


Figura 3.34. Código y apariencia de un encabezado sencillo con Shiny Dashboard.

Barra de Navegación

La **barra de navegación** se genera con **dashboardSidebar()**. Contiene un *menú* con los títulos de los paneles por los que estará formada la aplicación, como se muestra en la Figura 3.35. También puede incluir funciones para recoger inputs.

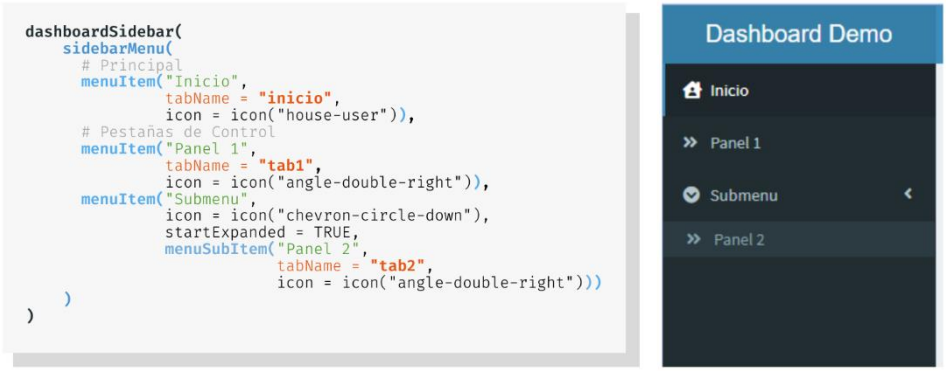


Figura 3.35. Código y apariencia de una barra de navegación con Shiny Dashboard.

Se usa **sidebarMenu()** para crear el menú dentro de dashboardSidebar(). Los paneles se referencian con **menuItem()**, función que también puede utilizarse para crear un submenú desplegable, en cuyo caso no estaría asociado a una capa concreta, sino que contendría referencias a paneles mediante **menuSubItem()**.

El primer argumento de menuItem() y menuSubItem() es el título que se muestra en el menú y el segundo, **tabName**, es el nombre del panel, que debe coincidir con el que lo identifica.

Contenido

El **contenido** de la aplicación se incluye dentro de **dashboardBody()**. Con un único panel, éste será el lienzo sobre el que se distribuirán los diferentes componentes de la aplicación. Usando diferentes paneles, será la zona del entorno en la que se visualizará cada uno de ellos, teniendo entonces tantos lienzos como paneles tenga la aplicación. En la Figura 3.36 se puede ver el esqueleto del contenido de una aplicación con tres paneles.

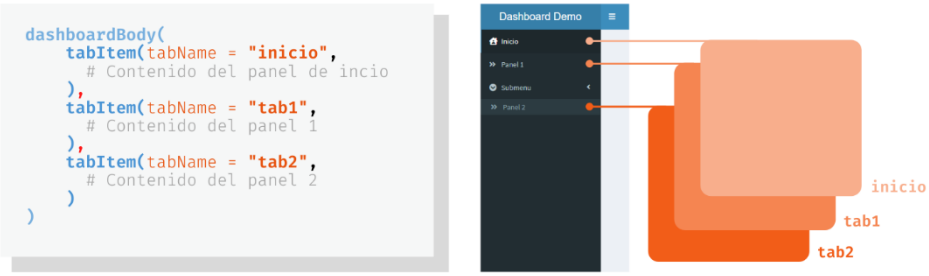


Figura 3.36. Código y esquema del cuerpo de una aplicación con Shiny Dashboard.

Los paneles se crean con la función **tabItem()**. El nombre de cada uno debe coincidir con el **tabName** definido en el menú para que se cree la conexión entre la referencia de la barra de navegación y el tab.

4. munimapp

En los capítulos anteriores se describió la base de datos, compuesta por los mapas y toda la información que se representa en ellos, y se dio a conocer Shiny, la herramienta con la que se ha creado la aplicación en la que se centra este proyecto. Con el propósito de explicar cómo se han aplicado los recursos que ofrece Shiny para desarrollar el entorno en el que se integran los mapas, en este capítulo se detalla la estructura de la aplicación y sus componentes.

La aplicación, denominada **munimapp**, nombre creado a partir de la combinación de las palabras clave municipio, mapa y app, ofrece una plataforma que facilita el acceso a información demográfica, así como a diversos indicadores derivados de estos datos, permitiendo que los usuarios conozcan detalles de la distribución de la población de cada territorio según cada ámbito geográfico. Como ya se hizo mención en el punto 3.1.3, la aplicación está alojada en un servidor Shiny, de cuyas características se hablará en el primer apartado de este capítulo (ver 4.1). Este servidor también da soporte a los mapas interactivos que se almacenan como archivos HTML, lo que los convierte en páginas web que se visualizarán por medio de la aplicación. En la descripción de la estructura de **munimapp** (ver 4.2) se indicará a qué sección pertenece cada uno de esos mapas y se comentará la distribución del contenido de la aplicación. Por último, se profundizará en la implementación de la **interfaz de usuario** y el **server** (ver 4.3 y 4.4) para explicar cómo se han definido la apariencia y el conjunto de instrucciones que rigen el comportamiento de la aplicación.

Se puede visitar **munimapp** mediante el siguiente enlace:
http://shiny1.eio.uva.es:3838/users/tania/munimapp/

4.1. Características del Servidor Shiny

La instalación y configuración del servidor de la aplicación ha sido llevada a cabo por el personal informático del departamento de Estadística e Investigación Operativa, quien además ha facilitado la información que se incluye en este apartado. [6]

[6] Además del servidor Shiny, se incluyen las características de herramientas adicionales que han sido instaladas para poder dar servicio y soporte al presente trabajo.

Características del Servidor Shiny

- Sistema Operativo Ubuntu Server 18.04.5
- Servidor Web Apache 2.4.29 [7]
- Base de Datos MySql 5.7.34
- PHP 7.2.24
- R 3.6
- RStudio Server 1.4.1103
- Shiny Server 1.5.15.953

Tras la instalación, fue necesario configurar el servidor para permitir que los usuarios pudieran publicar sus trabajos y también se modificaron los permisos del servidor Apache para poder alojar las páginas web que usa la aplicación.

4.2. **Estructura de la Aplicación**

La aplicación está compuesta por cuatro apartados. El panel de la **página principal**, cuya apariencia puede verse en la Figura 4.1, muestra un índice del contenido, en el que se incluye una breve descripción del tipo de representaciones que el usuario se encontrará en cada categoría. A continuación, se detalla cada apartado y sus subcategorías.

Figura 4.1. Apariencia del **menú principal** de la aplicación.

[7] Este servidor se utiliza como repositorio de las páginas web de tipo HTML que almacenan los mapas interactivos, creadas para dar servicio a la aplicación alojada en el servidor Shiny.

4.2.1. **Definiciones e Información**

Para facilitar a los usuarios la comprensión e interpretación de la información que se visualiza en los mapas, en este apartado se ofrece una breve descripción de los indicadores de población, que coincide con la definición dada en el apartado 1.1.4 de este documento, incluyendo ejemplos para ayudar a entender mejor cada uno de los conceptos. El aspecto de este panel puede verse en la Figura 4.2.

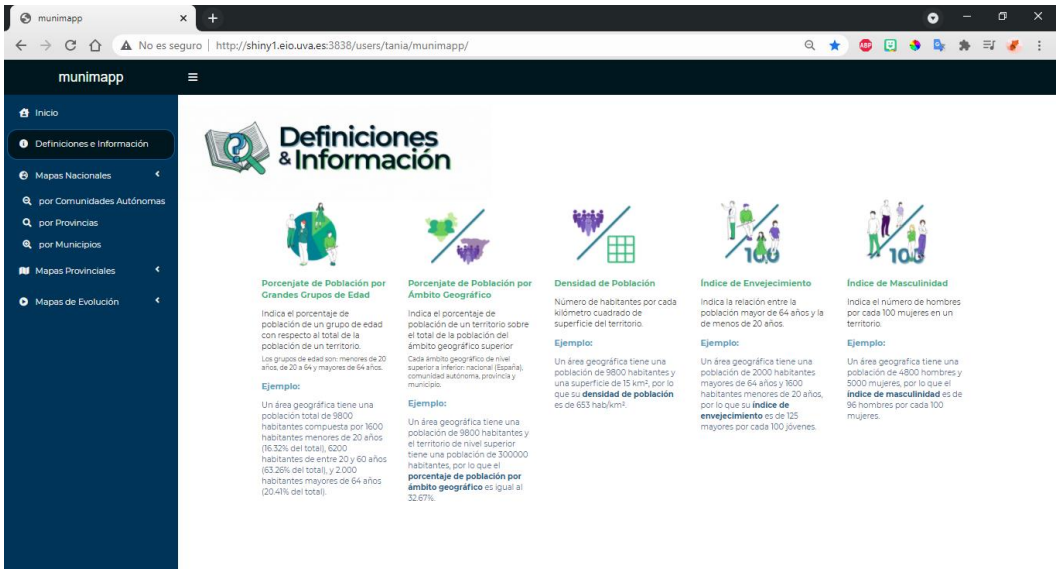


Figura 4.2. Apariencia del apartado **Definiciones e Información**.

4.2.2. **Mapas Nacionales**

En esta categoría se engloban los mapas nacionales divididos en tres subapartados según el ámbito geográfico. En los mapas se representa la información asociada a 2020 y los gráficos de los pop-ups permiten ver los cambios producidos en los últimos 10 años.

Mapas Nacionales por Comunidades Autónomas

En este subapartado se muestran los mapas interactivos descritos en el punto 2.4.1. La Figura 4.3 muestra el aspecto de este panel que está formado por un menú de selección al lado derecho y el mapa de fondo.

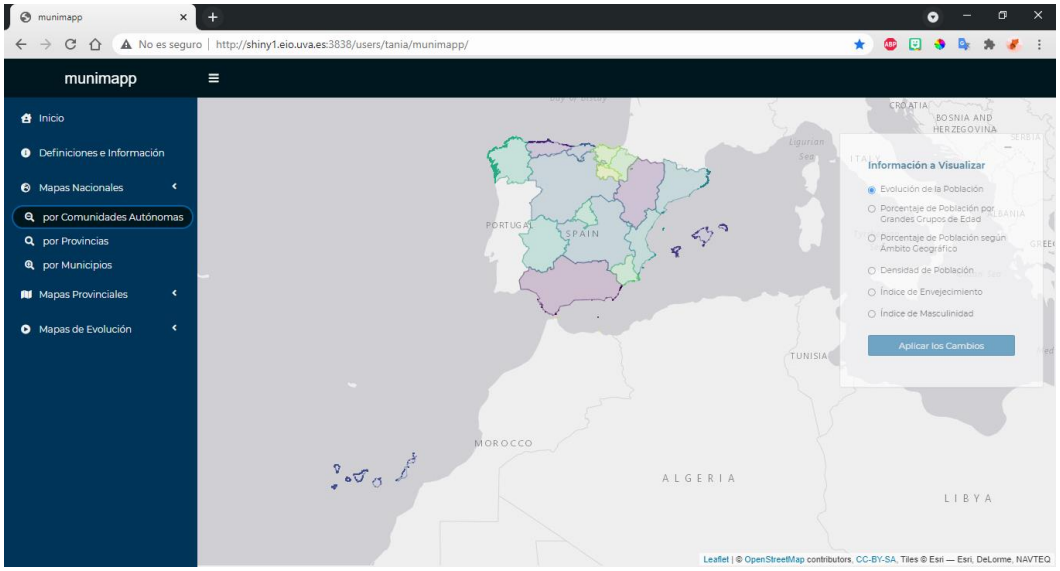


Figura 4.3. Apariencia de **Mapas Nacionales: por Comunidades Autónomas**.

El menú de selección permite elegir la información a representar, que cambiará al seleccionar otras opciones y pulsar el botón que indica la acción de aplicar los cambios. Para que el usuario pueda disfrutar de una visión amplia del mapa, es posible minimizar este panel como se muestra en la Figura 4.4. El cuadro que contiene el menú presentará una ligera transparencia cuando no esté seleccionado, volviéndose más nítido al realizar un clic sobre él.

Figura 4.4. Como **minimizar** y **maximizar** el menú de selección.

Mapas Nacionales por Provincias

En la Figura 4.5 se puede ver el aspecto de este panel, idéntico al de la subcategoría anterior, con la diferencia de que en este se visualiza los mapas interactivos descritos en el punto 2.4.1. El menú de selección de información a visualizar puede cubrir parcialmente la leyenda, por lo que se recomienda minimizar el panel como se indica en la Figura 4.4.

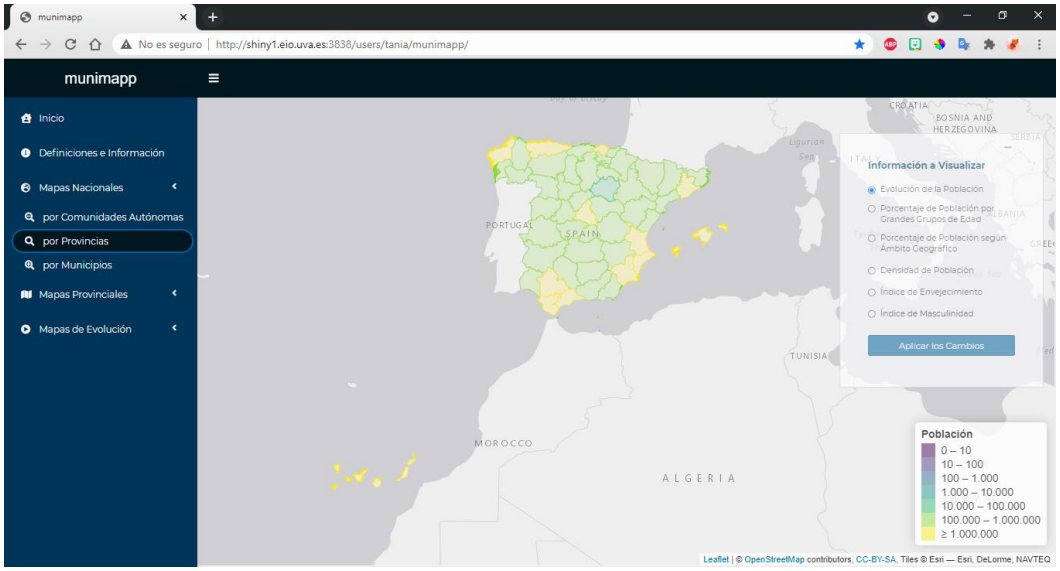


Figura 4.5. Apariencia de **Mapas Nacionales: por Provincias**.

Mapas Nacionales por Municipios

En esta subcategoría se puede ver el mapa interactivo que muestra todo el territorio español por municipios, el cual se describió en el punto 2.4.1. El tiempo que tarda en visualizarse es superior al de las representaciones de las otras dos subcategorías dentro de Mapas Nacionales debido a que la cantidad de información contenida en este archivo es mucho mayor, como consecuencia de que el número de municipios españoles es 8131. También se diferencia de los otros mapas en que no es posible modificar los datos que se representan sobre él, como se puede ver en la Figura 4.6, sino que el menú de selección pasa a ser un cuadro de texto que hace referencia al subapartado de la categoría siguiente donde se puede visualizar más información.

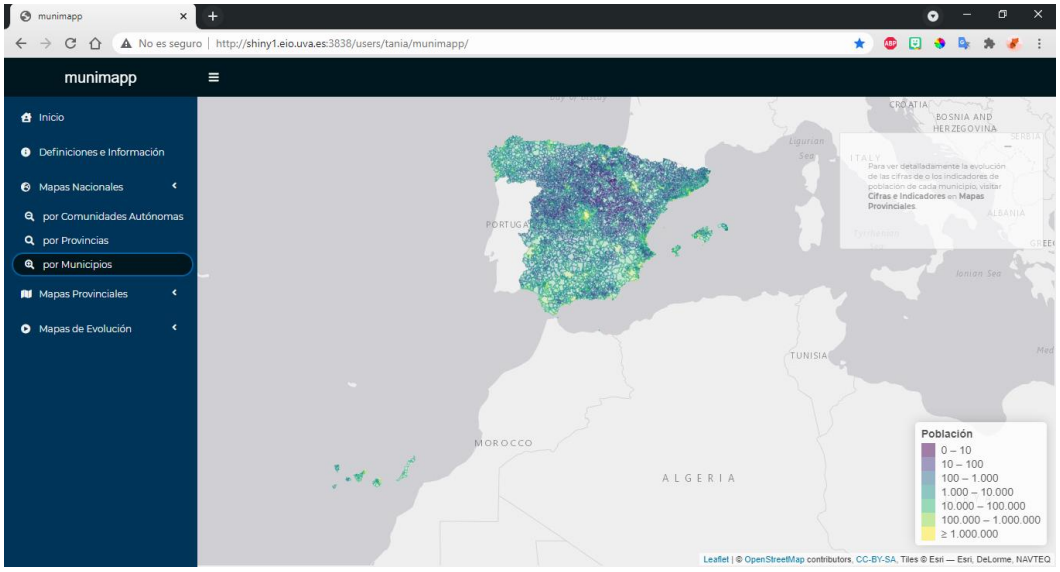


Figura 4.6. Apariencia de **Mapas Nacionales: por Municipios**.

4.2.3. Mapas Provinciales

En esta categoría se engloban los mapas provinciales divididos en dos subapartados que se describen a continuación.

Cifras e Indicadores

A este subapartado se ha trasladado la información de la representación de **Mapas Nacionales: por Municipios** para dividirla por provincias. El patrón que siguen las representaciones de la categoría anterior de la aplicación (ver 4.2.2) es mostrar por medio de los pop-ups la evolución de las cifras e indicadores de población sobre un mapa que contiene los datos asociados al año 2020. En esta subcategoría se ofrece la posibilidad de visualizar la información municipal del mismo modo, mediante los mapas descritos en el punto 2.4.2 de este documento.

La Figura 4.7 muestra la apariencia de este panel, compuesto por el mapa de fondo y el menú de selección al lado derecho de la pantalla. En este caso se puede escoger tanto la información a visualizar como la provincia a representar. Es posible minimizar el menú para ver la leyenda como se indica en la Figura 4.4.

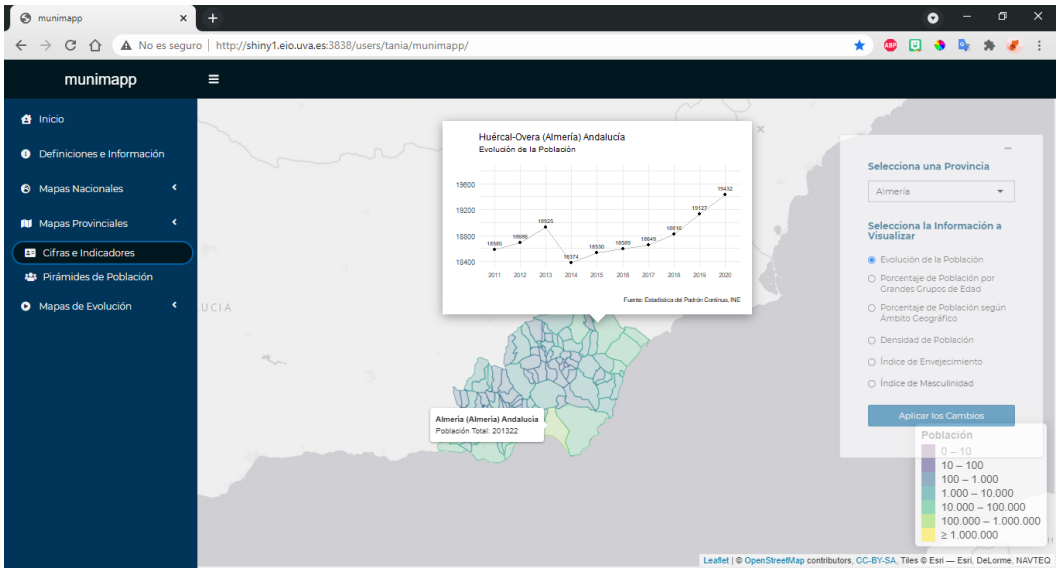


Figura 4.7. Apariencia de **Cifras e Indicadores** en **Mapas Provinciales**.

Pirámides de Población

Para ver más información a nivel municipal, en este panel, cuyo aspecto puede verse en la Figura 4.8, se ofrece la posibilidad de visualizar los mapas provinciales descritos en el punto 2.4.2.

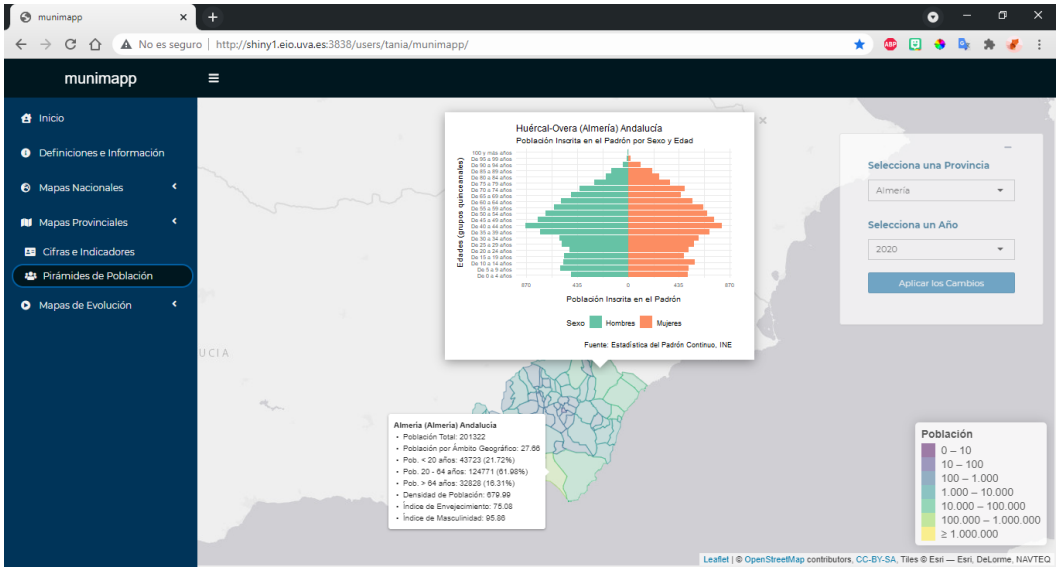


Figura 4.8. Apariencia de **Pirámides de Población** en **Mapas Provinciales**.

Las etiquetas contienen resúmenes de las características de cada territorio por año y los pop-ups muestran pirámides de población que indican la distribución de la población inscrita en el Padrón por sexo y edad por grupos quinquenales. Por ello, el menú de selección de este subapartado permite escoger la provincia y el año del mapa a visualizar.

4.2.4. Mapas de Evolución

En esta categoría se engloban los mapas animados que muestran la evolución de las cifras e indicadores como la densidad de población y los índices de envejecimiento y masculinidad. La capacidad de la herramienta para visualizar estos fenómenos es limitada, dado el corto periodo de tiempo en estudio, aspecto éste que ofrece una oportunidad de mejora para futuros proyectos.

Mapa Nacional

A excepción de los mapas provinciales, que incluyen pirámides de población en las ventanas emergentes de cada contorno, el resto solo muestran los datos asociados a 2020 sobre el mapa. Para poder ver la evolución de las cifras e indicadores sobre el territorio, en esta subcategoría se incluyen las representaciones descritas en el punto 2.4.3.

La Figura 4.9 muestra la apariencia de este panel, distinta a la de los anteriores. Al lado izquierdo de encuentra el menú de selección y al derecho la imagen en formato gif que muestra la evolución de la información escogida.

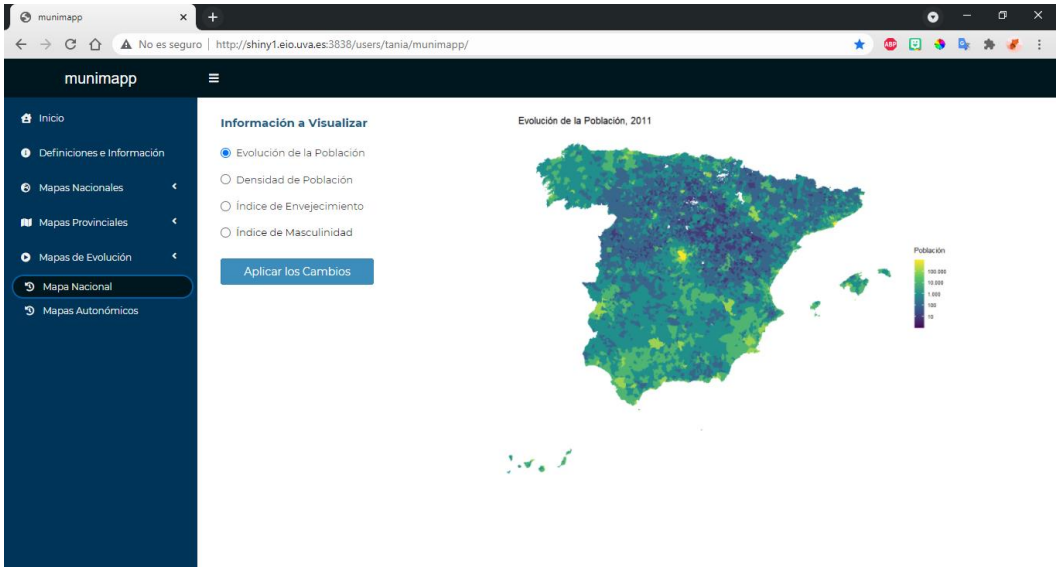


Figura 4.9. Apariencia de **Mapa Nacional** en **Mapas de Evolución**.

Mapas Autonómicos

Hay cambios imperceptibles en la evolución de las cifras e indicadores de población de los mapas de la subcategoría anterior que se pueden ver mejor dividiendo el territorio por comunidades autónomas. Los mapas de esta categoría están descritos en el punto 2.4.3.

La apariencia de este panel es similar a la del anterior, con la diferencia de que en este caso se incluye la opción de seleccionar la comunidad autónoma a visualizar, como se ve en la Figura 4.10.

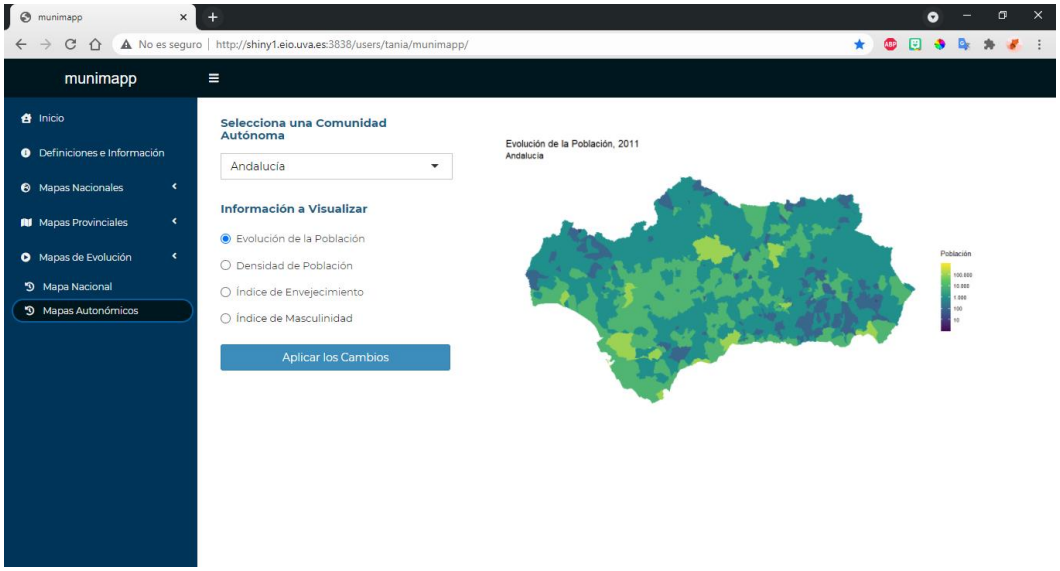


Figura 4.10. Apariencia del subapartado **Mapas Autonómicos** en **Mapas de Evolución**.

4.3. **Interfaz de Usuario**

La **interfaz de usuario**, **UI**, está implementada haciendo uso de la herramienta **shinydashboard** (ver 3.3.4). La Figura 3.33 muestra el aspecto predeterminado de este tipo de diseño, que ha sido modificado con la función **shinyDashboardThemeDIY**, del paquete **dashboardthemes** [4.1], para cambiar toda la apariencia visual del entorno, sin cambiar la estructura, y conseguir el resultado que se puede ver en la Figura 4.1.

El **encabezado** y la **barra de navegación** se han creado siguiendo el código de los ejemplos de la Figura 3.34 y la Figura 3.35, respectivamente, mientras que el **contenido** de la interfaz se generado siguiendo el esquema planteado en la Figura 3.36, donde se definió el esqueleto del cuerpo de una aplicación. Cada uno de los paneles, creados con la función **tabItem**, contienen el código que genera los elementos que se visualizan en cada categoría o subclase de un apartado.

El panel de **Inicio**, cuya apariencia se puede ver en la Figura 4.1, está compuesto únicamente por contenido estático, mezclando imágenes con texto, al igual que el primer apartado, **Definiciones e Información** (ver 4.2.1), visible en la Figura 4.2. Haciendo uso de las funciones **fluidRow()** y **column()**, esquematizadas en la Figura 3.30, junto con ilustraciones animadas y estáticas que acompañan al texto, se ha logrado crear ambos entornos. Se utilizan también tags, descritos en la Figura 3.27, para aplicar estilo sobre el texto y así generar los títulos y las definiciones.

Por otro lado, los paneles de los subapartados de **Mapas de Evolución** (ver 4.2.4), se han diseñado dividiendo el entorno en dos columnas con **column()**. Tanto en **Mapa Nacional** como en **Mapas Autonómicos**, en la primera columna se sitúan un input de tipo **radioButtons**, para la elección de la información a visualizar, y un botón de acción, para aplicar los cambios, creado con **submitButton**, mientras que es necesario incluir un **selectInput** en el caso del panel del segundo subapartado, **Mapas Autonómicos**, para recoger el parámetro de entrada que indica la comunidad autónoma que se debe mostrar en pantalla. La descripción de estos tres inputs se puede ver en la Figura 3.5. En la segunda columna se visualiza el mapa con la función **outputImage**, mencionada en la Figura 3.6.

El resto de paneles, asociados a las subcategorías de los apartados **Mapas Nacionales** (4.2.2) y **Mapas Provinciales** (4.2.3), siguen la misma estructura, diferenciándose unos de otros por los inputs que contienen los menús de selección. Su diseño está basado en el

ejemplo **SuperZip**[8], de la galería de temas de Shiny. La Figura 4.11 muestra el esquema del código de estos paneles. El contenido del **tabItem** se incluye dentro un una sección definida con **div**, una etiqueta HTML que permite agrupar componentes en un bloque, al que se le asigna la apariencia especificada en un archivo **.css** haciendo uso de **tags$head** (ver 3.3.1) y la función **includeCSS**. El código de ese documento de estilo define la apariencia de los paneles, indicando la posición del mapa como fondo y el aspecto del panel que contiene el menú de selección y sus características. El mapa se renderiza con la función **htmlOutput**, y los inputs que contiene cada panel se sitúan en un cuadro definido con **absolutePanel**, función descrita en la Figura 3.31, y a su vez dentro de una caja definida por **box**, un componente integrado en el paquete **shinydashboard** que sirve para agrupar contenido. Estas cajas dan la opción de contraerse, recurso que se aprovecha para minimizar el menú de selección como se indicó en la Figura 4.4.

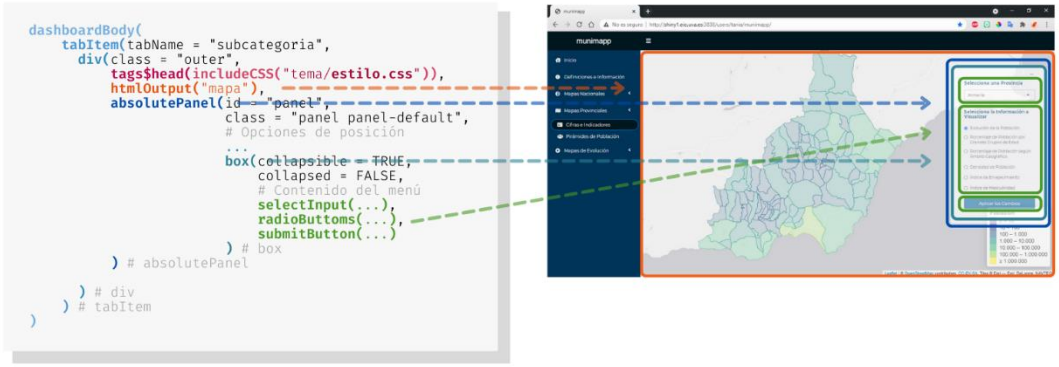


Figura 4.11. Código genérico del aspecto de los paneles que contienen mapas interactivos y asociación con los componentes visuales en la aplicación.

Es importante recalcar la ventaja que supone incluir el menú dentro de un **box** con la opción de minimizar activada. Esto no solo ofrece la posibilidad de ampliar el rango de visión de los mapas en la pantalla del ordenador, sino que hace que la aplicación sea compatible con dispositivos móviles al dar la opción de minimizar el menú, que puede ocupar toda la pantalla, como se muestra en la Figura 4.12.

[8] Para ver el ejemplo **SuperZip**, visitar https://github.com/rstudio/shiny-examples/tree/master/063-superzip-example

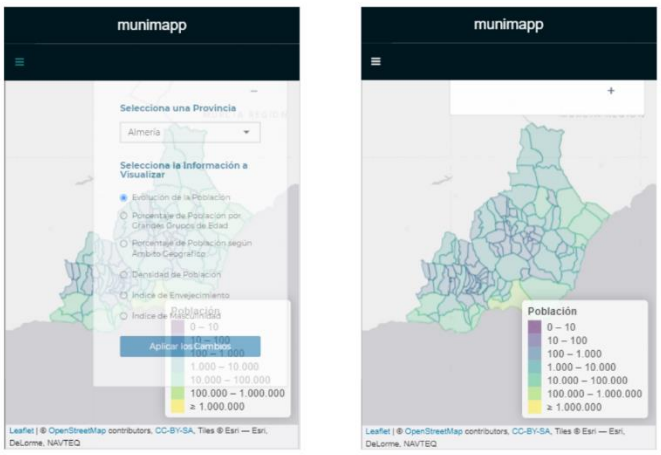


Figura 4.12. Apariencia de la aplicación en la versión para dispositivos móviles.

4.4. **Server**

Por último, se describe el **server** de la aplicación. Tiene una estructura muy sencilla porque al haber generado todos los mapas previamente, el server se limita a acceder a los archivos porque, en función de los valores reactivos que especifican la representación que el usuario quiere visualizar, y renderiza los mapas.

Se ha optado por controlar la reactividad de la aplicación retrasando las acciones con **eventReactive** (ver 3.3.2). Cuando el usuario indique que se debe aplicar un cambio, pulsará el botón de acción, provocando que la expresión reactiva que contiene el vínculo que lleva a cada mapa se actualice y así la función de renderización asociada a ella actuará según sus especificaciones. La Figura 4.13 muestra un ejemplo del fragmento de código con el que se generan los output. Dependiendo de los inputs de cada panel, **eventReactive** contendrá más o menos parámetros para crear la dirección web de cada representación, pero en esencia seguirá la estructura de la ilustración mencionada. Los mapas nacionales, por ejemplo, solo necesitan que se especifique el tipo de información que debe visualizarse, mientras que para los mapas provinciales se debe tener en cuenta el territorio escogido para acceder a su carpeta. La dirección web resultante se renderiza mediante **renderUI**, haciendo uso de la función **iframe** del paquete **shinyLP** [4.2].

En el caso de la categoría **mapas de evolución** (ver 4.2.4), como las representaciones no son archivos HTML, sino imágenes en formato gif, para renderizar los mapas se usa **renderImage** en vez de usar **renderIU**, pero manteniendo la estructura descrita en la Figura 4.13.

```r
function(input, output) {

  # url del HTML del mapa elegido
  url <- eventReactive(c(input$territorio, input$eleccion), {
    tipo <- switch(input$eleccion,
                   EvolucionPoblacion = "1_EvolucionPoblacion",
                   PjeGGruposEdad = "2_PjeGGruposEdad",
                   PjeAmbitoGeografico = "3_PjeAmbitoGeografico",
                   Densidad = "4_Densidad",
                   IndiceEnvejecimiento = "5_IndiceEnvejecimiento",
                   IndiceMasculinidad = "6_IndiceMasculinidad")
    CCA  <- tabla[tabla$NPRO == input$territorio, "CCA"][1]
    CPRO <- tabla[tabla$NPRO == input$territorio, "CPRO"][1]
    paste("...ruta/", CCA, "/", CPRO, "/mapa", tipo, ".html", sep = "")
  })

  # Función reactiva para generar el output:
  output$mapa <- renderUI({
    iframe(width = "100%", height = "100%", url_link = url())
  })

}
```

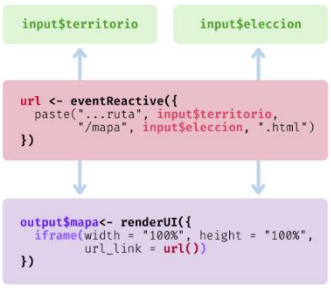


Figura 4.13. Esquema simplificado del **server** de la aplicación.

Conclusiones y Trabajos Futuros

El desarrollo de **munimapp** ha supuesto combinar conocimiento acerca de gestión de bases de datos, estadística descriptiva, diseño técnico del software con R, diseño gráfico y diversas herramientas relacionadas con el diseño web, como lenguaje HTML, hojas de estilo en cascada, etc., por lo que, por lo que se trata de un proyecto claramente multidisciplinar. La combinación de estas disciplinas ha dado como resultado una herramienta para la visualización de datos demográficos que reúne una gran cantidad de información para exponerla por medio de mapas interactivos. Este enfoque plantea un método de divulgación de información que pretende dar acceso a los datos de forma más llamativa y atractiva para la ciudadanía, sintetizándolos en una misma plataforma que hace que sea más cómodo consultarlos. Además, esta aplicación puede resultar de utilidad como fuente de información para el estudio de distintos fenómenos demográficos (despoblación y masculinización del medio rural, envejecimiento de la población, etc.) y para el análisis de los factores que contribuyen a la actual distribución de la población en los distintos territorios.

La gestión de la base de datos interna, compuesta por la información que se utiliza para crear los mapas de la aplicación, implica generar y almacenar por adelantado los datos, para así darles coherencia y asegurar que serán siempre los mismos. Esto lleva a minimizar la redundancia, evitando que se realicen las mismas operaciones sobre los datos cada vez que se genera una representación con ellos, y permite agilizar la creación de los mapas al no tener que manipular la información en el acto. De este modo, el acceso a los conjuntos de datos resulta más rápido y eficiente.

La descripción de la información recabada y almacenada se realiza utilizando técnicas de estadística descriptiva por medio de representaciones gráficas creadas con R. Al unirlos con los mapas interactivos, en vez de simplemente mostrar los datos con tablas y gráficos, se ofrece un marco innovador para visualizar la información, que permite asociar las características de un territorio con su posición geográfica.

La integración de los mapas interactivos en el entorno visual generado con Shiny ha supuesto un gran reto dada la dificultad que implica visualizar información sobre este tipo de representaciones procurando que el tiempo de respuesta sea mínimo. Éste es un elemento fundamental en una aplicación.

La herramienta implementada en este trabajo no solo se basa en su contenido, también es importante la apariencia del entorno en el que se muestran los mapas. En el desarrollo de aplicaciones, el diseño web es un elemento muy importante, ya que una apariencia visual adecuada y de calidad mejora la experiencia del usuario. Es vital hacer una buena elección de los colores que sea acorde a la que se utiliza para representar la información en los mapas, de modo que haya armonía entre todos los elementos de la aplicación. También es imprescindible elegir adecuadamente la tipografía para que ésta sea legible y acorde a la página. Además, dotar a la aplicación de elementos visuales, como iconos, texto y textura, es importante para captar la atención del usuario, despertar su interés por explorar la interactividad de la aplicación y mejorar su comprensión de la información que se ofrece.

El resultado final de este Trabajo Fin de Grado supone una herramienta cómoda y original que brinda acceso de forma dinámica a información sobre la distribución, composición y evolución de la población a diferentes niveles territoriales, centrándose en la visualización de datos a escala municipal.

Trabajos Futuros

Durante el desarrollo de este trabajo han surgido ideas futuras que podrían llevarse a cabo para ampliar el contenido de la aplicación.

Se seleccionaron los datos entre 2011 y 2020 para que los datos del Padrón Continuo pudieran adecuarse a la cartografía digitalizada de secciones censales del INE, como se mencionó en el apartado 1.1.2. Sin embargo, disponiendo del listado de municipios que se han creado y suprimido a lo largo de los años, sería interesante estudiar el modo de hacer frente a la falta de información cartográfica a nivel municipal anterior a 2011 para poder representar más periodos. De este modo se lograría observar muchos más cambios en la evolución de las cifras e indicadores de población, ya que ciertos fenómenos son imperceptibles considerando un lapso de tiempo de una década.

Considerar indicadores demográficos que proporcionen información acerca de otras características en aspectos como natalidad, fecundidad y mortalidad, sería enriquecedor para la aplicación porque permitiría visualizar cómo afectan estos fenómenos tanto en la evolución como en la distribución de la población. También sería posible estudiar cualquier otro tipo de indicadores que describan la economía y su desarrollo, la situación sanitaria (de especial interés en esta época de pandemia), la situación educativa, social, etc.

En definitiva, **munimapp** puede ser considerada como un primer paso que podría tener una continuidad a través de sucesivos proyectos que completen y enriquezcan el contenido de la misma.

Referencias

[0.1] R Core Team (2019). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

[1.1] Instituto Nacional de Estadística. *Cartografía digitalizada de secciones censales* [Digitalización de los contornos georreferenciados de todas las secciones censales, según coordenadas UTM, huso 28, 29, 30 y 31]. INEbase. https://www.ine.es/uc/TGWSB6oe

[1.2] Instituto Geográfico Nacional. *Límites municipales, provinciales y autonómicos* [Recintos municipales y líneas límite (municipales, provinciales y autonómicos)]. Centro de Descargas del Centro Nacional de Información Geográfica. https://doi.org/10.7419/162.09.2020

[1.3] Secretaría General Técnica del Ministerio de Política Territorial. (2009, agosto). *Estudio sobre el Registro de Entidades Locales*. Ministerio de Política Territorial, p.8. https://www.hacienda.gob.es/Documentacion/Publico/SGT/CATALOGO_SEFP/219_Estudio-Registro-Entidades-Locales-INTERNET.pdf

[1.4] Ibáñez Asensio, S., Gisbert Blanquer, J. M., & Moreno Ramón, H. (2020, 9 noviembre). *El sistema de coordenadas UTM*. RiuNet, Repositorio Institucional UPV, p. 2-3. Recuperado el 20 de junio de 2021, de https://riunet.upv.es/handle/10251/10772

[1.5] Environmental Systems Research Institute. (s. f.). *Qué es un shapefile*. ESRI. Recuperado el 20 de junio de 2021, de https://desktop.arcgis.com/es/arcmap/10.3/manage-data/shapefiles/what-is-a-shapefile.htm

[1.6] Environmental Systems Research Institute. (1998, marzo). *ESRI Shapefile Technical Description*. ESRI. https://support.esri.com/es/white-paper/279

[1.7] Instituto Nacional de Estadística. (2020, 7 febrero). *Relación de Municipios y sus Códigos por Provincias*. INEbase. Recuperado el 20 de junio de 2021 de, https://www.ine.es/dynt3/metadatos/es/RespuestaDatos.htm?oe=30247

[1.8] Instituto Nacional de Estadística. (2021, 20 abril). *Estadística del Padrón Continuo*. INEbase. Recuperado el 21 de junio de 2021, de https://www.ine.es/uc/TE1eJxUT

[1.9] Instituto Nacional de Estadística. (2021, 21 enero). *Población por sexo, municipios y edad (grupos quinquenales)*. INEbase. Recuperado el 20 de junio de 2021, de https://www.ine.es/uc/8IGtXb2t

[1.10] Instituto Nacional de Estadística. (2008, marzo). Entorno físico y medio ambiente, Territorio: Extensión superficial de las comunidades autónomas y provincias. En *Anuario Estadístico de España* (p. 10). INEbase. https://www.ine.es/prodyser/pubweb/anuario08/anu08_01entor.pdf

[1.11] Instituto Gallego de Estadística. (s.f.). *Indicadores Demográficos. Metodología*. IGE, p. 2-5. Recuperado el 22 de junio de 2021, de https://www.ige.eu/estatico/pdfs/s3/metodoloxias/met_indicadores_demograficos_es.pdf

[1.12] Pebesma, E., 2018. *Simple Features for R: Standardized Support for Spatial Vector Data*. The R Journal 10(1), 439-446, https://doi.org/10.32614/RJ-2018-009

[1.13] Pebesma, E. (s. f.). *Simple Features for R. Simple Features for R*. Recuperado el 23 de junio de 2021, de https://r-spatial.github.io/sf/articles/sf1.html

[1.14] Mas, J. F. (2018). Organización de los objetos espaciales en R. En *Análisis Espacial con R: Usa R como un Sistema de Información Geográfica* (p. 33). European Scientific Institute. http://eujournal.org/files/journals/1/books/JeanFrancoisMas.pdf

[1.15] Wickham, H., François, R., Henry, L., & Müller, K. (2021*). dplyr: A Grammar of Data Manipulation*. R package version 1.0.4. https://CRAN.R-project.org/package=dplyr

[1.16] Mas, J. F. (2018). Operaciones básicas en R. En *Análisis Espacial con R: Usa R como un Sistema de Información Geográfica* (p. 31-32). European Scientific Institute. http://eujournal.org/files/journals/1/books/JeanFrancoisMas.pdf

[1.17] García Barona, M. F. (2014, 4 diciembre). *Importación de archivos Pc-Axis a R* [Entrada de blog]. El meu compte de dades i estadístiques. https://garmafer.blogs.uv.es/archivos/34

[1.18] Gil Bellosta, C. J., Viciana, F. J., & Perpinan Lamigueiro, O. (2020). *pxR: PC-Axis with R*. R package version 0.42.4. https://CRAN.R-project.org/package=pxR

[2.1] Pebesma, E., & Bivand, R. (2021). *Spatial Data Science*. Recuperado el 2 de julio de 2021, de https://keen-swartz-3146c4.netlify.app/

[2.2] Villalba Bergado, F. (2019, 26 febrero). *Manual de sf para SIG* [Entrada de blog]. enrdados. http://enrdados.net/post/manual-de-sf-para-sig/

[2.3] Frazier, M. (s. f.). *Overview of Coordinate Reference Systems (CRS) in R*. NCEAS. Recuperado el 3 de julio de 2021, de https://www.nceas.ucsb.edu/sites/default/files/2020-04/OverviewCoordinateReferenceSystems.pdf en https://www.nceas.ucsb.edu/r-spatial-guides

[2.4] Ministerio de Agricultura, Pesca y Alimentación. (s. f.). *Características de los servicios de mapas (Web Map Service – WMS)*. La Moncloa. Recuperado el 11 de julio de 2021, de https://www.mapa.gob.es/es/cartografia-y-sig/ide/directorio_datos_servicios/caracteristicas_wms.aspx

[2.5] Cheng, J., Karambelkar, B., & Xie, Y. (2021). *leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library*. R package version 2.0.4.1. https://CRAN.R-project.org/package=leaflet

[2.6] RStudio. *R Interface to Leaflet Maps*. Leaflet for R. Recuperado el 4 de julio de 2021, de https://rstudio.github.io/leaflet/

[2.7] Wickham., H. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.3.5. https://CRAN.R-project.org/package=ggplot2

[2.8] Wickham, H., Navarro, D., & Pedersen, T. L. (s. f.). *ggplot2: Elegant Graphics for Data Analysis* (2.a ed.). https://ggplot2-book.org/

[2.9] Wilkinson, L. (2005). *The Grammar of Graphics* (2.a ed.). Springer. https://doi.org/10.1007/0-387-28695-0

[2.10] Appelhans, T. & Detsch, F. (2020). *leafpop: Include Tables, Images and Graphs in Leaflet Pop-Ups*. R package version 0.0.6. https://CRAN.R-project.org/package=leafpop

[2.11] Appelhans, T., Detsch, F., Reudenbach, C., & Woellauer, S. (2020). *mapview: Interactive Viewing of Spatial Data in R*. R package version 2.9.0. https://CRAN.R-project.org/package=mapview

[2.12] Garnier, S. (2018). *viridis: Default Color Maps from 'matplotlib'*. R package version 0.5.1. https://CRAN.R-project.org/package=viridis

[2.13] Dowle, M., & Srinivasan, A. (2021). *data.table: Extension of 'data.frame'*. R package version 1.14.0. https://CRAN.R-project.org/package=data.table

[2.14] Xie, Y. (2013, 21 abril). *animation: An R Package for Creating Animations and Demonstrating Statistical Methods*. Journal of Statistical Software, 53(1), 1-27. http://www.jstatsoft.org/v53/i01/

[3.1] Chang, W., Cheng, J., Allaire, J.F., Sievert, C., Schloerke, B., Xie, Y., Allen, J., McPherson, J., Dipert, A., & Borges, B. (2021). *shiny: Web Application Framework for R*. R package version 1.6.0. https://CRAN.R-project.org/package=shiny

[3.2] *The basic parts of a Shiny app*. (2017, 28 junio). Shiny. https://shiny.rstudio.com/articles/basics.html

[3.3] R Core Team. (2019). *Old Faithful Geyser Data* (version 3.6.1) [Waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA]. Azzalini, A. and Bowman, A. W. (1990). A look at some data on the Old Faithful geyser. Applied Statistics, 39, 357–365. https://doi.org/10.2307/2347385

[3.4] Kipp, A. (2017, 30 mayo). *Getting started with shinyapps.io*. Shiny. https://shiny.rstudio.com/articles/shinyapps.html

[3.5] Allaire, J.J. (2019). *rsconnect: Deployment Interface for R Markdown Documents and Shiny Applications*. R package version 0.8.16. https://CRAN.R-project.org/package=rsconnect

[3.6] *Shiny Server*. [Open Source Shiny Server provides a platform on which you can host multiple Shiny applications on a single server, each with their own URL or port. It enables you to support non-websocket-enabled browsers like Internet Explorer 10, and is available under an AGPLv3 license]. RStudio. https://www.rstudio.com/products/shiny/download-server/

[3.7] RStudio. (s. f.*). Shiny Server Professional Administrator's Guide*. Recuperado el 27 de junio de 2021, de https://docs.rstudio.com/shiny-server/#getting-started

[3.8] Grolemund, G. (2017, 8 agosto). *Shiny HTML Tags Glossary*. Shiny. https://shiny.rstudio.com/articles/tag-glossary.html

[3.9] Allaire, J. J. (2021, 1 febrero). *Application layout guide. Shiny*. https://shiny.rstudio.com/articles/layout-guide.html

[3.10] Winston Chang and Barbara Borges Ribeiro (2018). *shinydashboard: Create Dashboards with 'Shiny'*. R package version 0.7.1. https://CRAN.R-project.org/package=shinydashboard

[3.11] RStudio. (s. f.). *Shiny Dashboard Structure*. shinydashboard. Recuperado 11 de julio de 2021, de https://rstudio.github.io/shinydashboard/structure.html

[4.1] Lilovski, N. (2020*). dashboardthemes: Customise the Appearance of 'shinydashboard' Applications using Themes*. R package version 1.1.3. https://CRAN.R-project.org/package=dashboardthemes

[4.2] Dumas, J. (2018). *shinyLP: Bootstrap Landing Home Pages for Shiny Applications*. R package version 1.1.2. https://CRAN.R-project.org/package=shinyLP