Universidad de Valladolid

Facultad de Ciencias

TRABAJO FIN DE GRADO

Grado en Estadística

Desarrollo de un paquete en *R* para el análisis de datos electorales

Alumno:
Ignacio Domínguez Calvo

Tutores:
Jesús Alberto Tapia García
Jesús María Rodríguez Rodríguez

A mi madre por apoyarme siempre.

Agradecimientos

Quiero empezar agradeciendo a mis tutores D. Jesús Alberto Tapia García y D. Jesús María Rodriguez Rodríguez por su ayuda con este trabajo y por haber podido aprender mucho de ellos.

A mis profesores del grado en Estadística, por transmitir su pasión por la Estadística a sus alumnos. Esta pasión, transmitida en cada clase, laboratorio y sesión de tutoría, ha sido una fuente de inspiración constante que ha enriquecido mi experiencia educativa.

A mis compañeros de promoción, por ayudarme siempre que lo he necesitado.

Por último, a mi madre por apoyarme con mis estudios y animarme siempre que he querido estudiar más, a mi hermano por enseñarme matemáticas desde que tengo memoria y al resto de mi familia por todo su apoyo.

Resumen

Resumen

En el entorno actual, donde los organismos especializados producen estadísticas con gran rigor, el desafío no radica tanto en acceder a los datos, sino en la facilidad de su lectura e interpretación. Aunque se avanza hacia la estandarización de formatos para facilitar la comprensión de los datos, aún existen numerosas fuentes que utilizan formatos inusuales, lo que demanda un análisis exhaustivo para su correcta lectura. Este Trabajo Fin de Grado aborda específicamente el desafío de interpretar datos de elecciones presentados en formatos de difícil acceso y la creación de funciones que faciliten su análisis.

Abstract

In the current environment, where specialized organizations produce statistics with great rigor, the challenge lies not so much in accessing the data, but in the ease of reading and interpreting it. Although there is progress towards the standardization of formats to facilitate the understanding of data, there are still numerous sources that use unusual formats, which requires thorough analysis for correct interpretation. This Bachelor's Thesis specifically addresses the challenge of interpreting election data presented in hard-to-access formats and creating functions to facilitate its analysis.

Índice general

Agradecimientos **3**

Resumen **5**

Lista de figuras **8**

Lista de tablas **9**

1. Introducción **13**

1.1. Tecnologías utilizadas 13

1.2. Objetivos 14

1.3. Estructura de la memoria 14

2. Descripción de los datos **17**

2.1. Tipos de elecciones 17

2.2. Tipo de datos 18

3. R como lenguaje de programación **21**

3.1. Paquetes 22

3.1.1. Estructura 22

3.2. Funciones 23

3.3. Orientación a objetos 23

3.4. Documentación 24

3.5. Estructuras de datos 26

4. Desarrollo del paquete **29**

4.1. Lectura de datos 29

4.2. Mapas 32

4.3. Tablas Resumen 36

5. Conclusiones y líneas de trabajo futuras **41**

5.1. Lineas de trabajo futuras 42

A. Código Fuente **43**

A.1. `LecturaDatos.R` 43

A.2. `Mapas.R` 50

A.3. Tablas.R 54

Lista de Figuras

4.1. Diagrama de flujo del proceso de descarga y procesamiento de datos electorales 31

4.2. Ejemplo de representación de los votos en blanco por provincias 34

4.3. Ejemplo de representación de los votos en primera ronda por Comunidades Autónomas 35

4.4. Ejemplo de representación de la proporción de votos en blanco por municipios en la provincia de Valladolid. 36

4.5. Diagrama de flujo del proceso de creación de tabla participación 38

Lista de Tablas

2.1. Tabla de Tipos de Elecciones 18

2.2. Fechas de primer y último dato para cada tipo de elección. 18

2.3. Tabla de ejemplo obtenida de FICHEROS.docx 19

3.1. Etiquetas comunes de roxygen2 para la documentación en R 25

4.1. Tiempos de ejecución para la lectura de los datos 30

4.2. Resumen de Funciones en el fichero `LeerDatos` 32

4.3. Descripción de Funciones para la Creación de Mapas 37

4.4. Descripción de Funciones para la obtención de tablas 39

Capítulo 1

Introducción

En el entorno actual, donde los organismos especializados producen estadísticas con gran rigor, el desafío no radica tanto en acceder a los datos, sino en la facilidad de su lectura e interpretación. Aunque se avanza hacia la estandarización de formatos para facilitar la comprensión de los datos, aún existen numerosas fuentes que utilizan formatos inusuales, lo que demanda un análisis exhaustivo para su correcta lectura.

Esto ocurre en la fuente de datos para elecciones [3] del gobierno, en el que los datos aparecen en múltiples tablas, y no se utiliza un estándar de acceso a ellos como pueda ser CSV, o JSON, si no que a las diferentes variables se accede por posiciones fijas en una sucesión de caracteres.

1.1. Tecnologías utilizadas

La siguientes tecnologías se han utilizado para una o varias partes del proyecto

- *R*:lenguaje de programación básico sobre el que se ha creado el paquete de datos electorales. Se ha elegido este lenguaje de programación por su extendido uso en la estadística.
- *RStudio*: herramienta para la producción y depuración del código en *R*. Tiene soporte con otras tecnologías, como Git o roxygen2
- *Github*: se ha elegido este repositorio como control de versiones del paquete. Cada funcionalidad nueva creada se separaba en un *commit* distinto para poder tener un control claro del código.
- *roxygen2*: paquete de *R* que sirve para documentar el código de forma dinámica, utilizando etiquetas específicas embebidas en el código, lo que agiliza y clarifica el proceso de documentación.

- *Python*: lenguaje de programación con mayor soporte para múltiples tareas, que se ha utilizado por su integración con API.
- *Open-AI API*: modelo GPT para el procesamiento de lenguaje natural.
- *OverLeaf*: editor de texto LaTeXen línea para la escritura de la presente memoria.
- *Draw.io*: herramienta para la creación de diagramas UML.

1.2. Objetivos

El principal objetivo del TFG es la creación de un paquete en R que facilite el acceso a la fuente de datos del gobierno de elecciones [3], de tal manera que mediante su utilización el acceso a los datos sea transparente y el usuario no sea consciente de la forma en la que están almacenados esos datos.

- **Desarrollar funciones en R para la lectura eficiente de datos electorales:** Diseñar y programar funciones robustas en R que faciliten la importación, procesamiento y análisis preliminar de los conjuntos de datos electorales proporcionados por el gobierno. Esto incluirá la gestión de diferentes formatos de archivo, la limpieza de datos y la normalización de estructuras de datos para análisis posteriores.
- **Implementar funciones para la visualización geográfica de datos electorales:** Crear funciones en R que permitan la generación de mapas detallados para visualizar diversos aspectos de los datos electorales, como distribución geográfica de votos, tendencias regionales y comparativas históricas por área. Estas funciones deberán integrar librerías como `ggplot2` y `leaflet` para mapas estáticos y dinámicos. Se deben hacer las funciones lo más independientes posibles, para poder aplicarse en múltiples contextos. Se debe además, permitir crear los mapas para distintas agrupaciones territoriales.
- **Desarrollar funciones para la creación de tablas resumen dinámicas:** Programar un conjunto de funciones en R que faciliten la generación de tablas resumen que consoliden la información clave obtenida de los datos electorales. Las tablas deberán ser configurables para mostrar distintos niveles de detalle y permitir comparaciones entre diferentes elecciones o demografías.
- **Creación y mantenimiento de un paquete de R:** Desarrollar un paquete de R que contenga todas las funciones mencionadas, asegurando su compatibilidad y funcionamiento en distintas versiones de R y en diferentes sistemas operativos. Incluir documentación detallada, ejemplos de uso, guías de instalación y actualización, y pruebas para garantizar la calidad y estabilidad del paquete.

1.3. Estructura de la memoria

Esta memoria consta de los siguientes capítulos:

- **Capítulo 1 - Descripción de los datos.** Se explora la fuente de datos de elecciones, explicando las particularidades en la forma de obtención de los datos. Se ofrecen algunas explicaciones sobre los tipos de elecciones que existen en España porque se necesitan conocer para entender los datos.
- **Capítulo 2 - R como lenguaje de programación.** Se explican los conceptos básicos de R vitales para poder construir un paquete. Se abordan conceptos más avanzados como son la orientación a objetos, la estructura de los paquetes y las buenas prácticas para el desarrollo de código R.
- **Capítulo 3 - Desarrollo del paquete.** Se explica la metodología seguida para la creación de las funciones del paquete y el funcionamiento de todas las funciones.
- **Capítulo 4 - Conclusiones y Líneas de trabajo futuras.** Desde la perspectiva de los objetivos propuestos, se comenta cuales han sido posibles de realizar y cuales no y sus razones. Se añaden líneas de trabajo futuras en base a los objetivos no cumplidos y a las posibles mejoras
- **Anexo I: Código Fuente.** Se adjunta el código fuente para los tres archivos con funciones y se adjunta el URL al repositorio con todo el código.

Capítulo 2

Descripción de los datos

Los fuente de datos electorales del gobierno [3] sirve para la obtención del histórico de elecciones guardado en ese servidor. En el documento sobre metodología [4] se establecen los fundamentos y las limitaciones de esta base de datos:

- El órgano encargado de el almacenamiento y custodia del archivo histórico de resultados electorales es el Ministerio del Interior.
- Los resultados provienen del escrutinio general realizado por las Juntas Electorales. INFOELECTORAL proporciona datos hasta el nivel de mesa electoral, aunque las Juntas Electorales no publican datos oficiales por debajo del nivel de circunscripción, excepto en las elecciones europeas, donde también se incluye el nivel provincial. Esto puede generar discrepancias y posibles errores en algún municipio o mesa electoral, que se corrigen en el escrutinio general. Los únicos datos oficiales son los publicados por las Juntas Electorales con el nivel de detalle que estas recogen.
-

2.1. Tipos de elecciones

Existen distintos tipos de elecciones alojadas en ese servidor como se puede ver en la Tabla. 2.1

El referéndum se convoca de forma esporádica, por lo que el número de ocurrencias es menor. Las elecciones al congreso y al senado se celebran como máximo cada 4 años, aunque frecuentemente se celebran más. Estos dos se celebran siempre juntos. Las elecciones municipales son fijas cada cuatro años, mientras que las elecciones europeas son fijas cada cinco. Los cabildos son órganos de representación especiales de canarias que representan a cada una de las islas y que también se convocan cada cuatro años. En la Tabla 2.2 se puede

Nombre	Prefijo	Descripción
Referéndum	01	Votación directa sobre una propuesta específica a nivel nacional o regional.
Congreso	02	Elecciones para elegir representantes en la cámara baja del parlamento.
Senado	03	Elecciones para elegir representantes en la cámara alta del parlamento.
Municipales	04	Elecciones para elegir representantes a nivel municipal.
Cabildos	06	Elecciones para elegir representantes en el cabildo insular.
Parlamento Europeo	07	Elecciones para elegir representantes en el Parlamento Europeo.

Tabla 2.1: Tabla de Tipos de Elecciones

ver la fecha de los primeros datos de cada tipo de elección y la última, a fecha de la redacción de la presente memoria, y el número de datos que existen.

Tabla 2.2: Fechas de primer y último dato para cada tipo de elección.

Nombre	Primer dato	Último dato	Cantidad
Referéndum	Diciembre 1976	Febrero 2005	4
Congreso	Junio 1977	Julio 2023	16
Senado	Junio 1977	Julio 2023	16
Municipales	Abril 1979	Mayo 2023	12
Cabildos	Abril 1979	Mayo 2023	12
Parlamento Europeo	Junio 1987	Mayo 2019	8

Estos datos se pueden recopilar de tres formas: agrupándose por mesas electorales, agrupando por municipios o por un ámbito superior al de los municipios.

2.2. Tipo de datos

Los datos de elecciones tienen una extesión .DAT que no puede ser interpretada directamente por R. A diferencia de otras extensiones de datos como CSV o JSON, DAT guarda los datos de forma no estandarizada, por lo que no se puede acceder directamente a ellos. Un archivo DAT puede contener datos en texto plano, binario o cualquier formato específico de cuando se creó. La principal desventaja de estos archivos es la falta de portabilidad, y

que si no se conoce la estructura con la que fueron escritos, no se puede acceder a ellos. Sin embargo, ofrecen ventajas como la eficiencia en el almacenamiento o su versatilidad para distintos tipos de datos.

La forma de descifrar los datos aparece cuando se descarga el archivo con los datos. Primero se descarga un archivo comprimido, y al descomprimirse se obtiene un archivo llamado `FICHEROS.docx` con unas Tablas 2.3. Estas están formadas por 5 campos: el carácter en el que empieza la variable, el carácter en el que termina, el tipo de dato entre numérico o alfanumérico y por último la descripción.

Inicio	**Fin**	**Tipo**	**Long.**	**Descripción**
1	2	Num.	2	Tipo de municipio:
3	6	Num.	4	Año del proceso electoral.
7	8	Num.	2	Mes del proceso electoral.
9	9	Num.	1	Número de vuelta (en procesos a una sola vuelta = 1).
10	11	Num.	2	Código I.N.E. de la provincia.
12	14	Num.	3	Código I.N.E. del municipio.
15	20	Num.	6	Código de la candidatura.
21	23	Num.	3	Votos obtenidos por la candidatura.
24	25	Num.	2	Número de candidatos obtenidos por la candidatura.
26	50	Alf.	25	Nombre del candidato.
51	75	Alf.	25	Primer apellido del candidato.
76	100	Alf.	25	Segundo apellido del candidato.
101	101	Alf.	1	Sexo del candidato (Masculino/Femenino).
102	103	Num.	2	Fecha de nacimiento del candidato (DIA).
104	105	Num.	2	Fecha de nacimiento del candidato (MES).
106	109	Num.	4	Fecha de nacimiento del candidato (AÑO).
110	119	Alf.	10	D.N.I. del candidato
120	122	Num.	3	Votos obtenidos por el candidato.
123	123	Alf.	1	Candidato elegido (Si/No).

Tabla 2.3: Tabla de ejemplo obtenida de FICHEROS.docx

Dado que la lectura de los datos se tiene que hacer haciendo uso de esas tablas se ha procedido de la siguiente manera. Primero se han copiado todas las tablas en un archivo excel separándose por páginas. Después de esto, cuando se quiera guardar esta información en un dataframe, se necesitará generar el nombre de las columnas en base a la descripción. Dado el número tan alto de variables a leer en todas las tablas (209), se ha recurrido a una llamada a la API de OpenAI.

Una API (*Application Programming Interface*) es un conjunto de reglas y protocolos que permiten la comunicación entre dos componentes de software. Estos componentes pueden ser muy diferentes entre sí, pero al aceptar una API, la comunicación entre ellos se estandariza

y permite que se entiendan mutuamente. En la comunicación web, una API se comporta mediante los protocolos estándar HTTP o HTTPS.

Un modelo LLM (*Large Language Model*) es un modelo con miles de millones de parámetros entrenables que aprende a predecir y generar texto en función de las secuencias de palabras que ha observado durante el entrenamiento. Estos modelos utilizan técnicas de aprendizaje automático, en particular el aprendizaje profundo, para comprender y generar texto de manera más precisa y coherente. Un modelo LLM como GPT (*Generative Pre-trained Transformer*) es capaz de aprender patrones complejos en el lenguaje natural y capturar la estructura gramatical y semántica de un texto. Esto se logra mediante el entrenamiento en grandes cantidades de datos textuales, donde el modelo intenta predecir la siguiente palabra en una secuencia dada, dadas las palabras anteriores. A medida que se expone a más datos y se entrena durante más tiempo, el modelo mejora su capacidad para generar texto que se asemeje al lenguaje humano.

Dada la alta precisión que se requiere para nombrar las variables, se ha optado por utilizar el modelo **gpt-4-turbo** de mayor coste para obtener mejores resultados.

Las llamadas a un modelo de lenguaje se basan en dos componentes, un *template* que es la base sobre la que se sustentan las *query* que son las consultas al modelo. En la plantilla se establece la forma en la que se quiere obtener la respuesta. La plantilla utilizada ha sido la siguiente:

> En base a esta descripcion {descripcion} genera el nombre de una variable. Usa solo una palabra o dos o tres separadas por _. Responde solo con el nombre de la variable"

Aunque estas llamadas simplifican mucho la tarea de generar los nombres de las variables, antes de llamarse a las funciones se han tenido que depurar los nombres de algunas variables y suprimir variables sin utilidad real.

Capítulo 3

R como lenguaje de programación

R es un lenguaje de programación que se desarrolló en el Departamento de Estadística de la Universidad de Auckland en 1993 por Robert Gentleman y Ross Ihaka. Es una extensión del lenguaje de programación *S*, desarrollado por la empresa estadounidense AT&T, en referencia a la "S"de *statistics* y al popular lenguaje de programación *C*. R se caracteriza por su tratamiento nativo de múltiples aplicaciones estadísticas. R es un lenguaje multiparadigma, lo que permite obtener mucha flexibilidad en muchas tareas. Los paradigmas en los que se basa son:

- **Orientación a objetos**: como todos los lenguajes modernos, la estructura de datos básica en R es el objeto. Un objeto es una instancia de una clase en específico. Esto permite dar gran flexibilidad a las operaciones, por ejemplo, al llamar a la función **lm()**, te devuelve un objeto que contiene todos los atributos necesarios, a diferencia de otro software, como SAS que devuelve toda la información por pantalla.[8]

  Otra característica muy importante es el uso de **polimorfismo**. Varias funciones con distintos parámetros y distintos tipos de datos pueden llamarse de igual forma y el interprete decide que llamada es la correcta. Por ejemplo, la función **plot()** se puede usar en un vector para representar un diagrama de dispersión, o se puede usar con la salida de una recta de regresión para obtener los gráficos de la regresión.

- **Programación funcional**: es un paradigma de programación que trata los cálculos computacionales como la evaluación de funciones matemáticas y evita los cambios de estado y la mutabilidad de los datos. Se basa en el uso de funciones puras, que son funciones que no tienen efectos secundarios y siempre producen el mismo resultado dado el mismo conjunto de entradas. La programación funcional otorga modularidad, al no depender de estados externos, lo que además añade facilidad de lectura. La ausencia de datos compartidos evita condiciones de carrera en situaciones del paralelismo. En lugar de utilizar iteración explícita con bucles, R fomenta el aprovechamiento de sus características funcionales.

- **Programación iterativa**: como todos los lenguajes de programación clásicos, se permite la opción de programar de forma iterativa, para lo que se implementan bucles como el **for** o el **while**, en aquellas situaciones en las que el control de flujo sea muy importante.
- Otros paradigmas menos importantes que R soporta son la programación reflexiva o procedimental.

3.1. Paquetes

El sistema de paquetes en *R* es una gran ventaja y por lo que goza de tanta popularidad. Un paquete es una colección de estructuras de datos, código y documentación que extiende las capacidades del lenguaje base. Se diferencia de las librerías de otros lenguajes en su rigidez. Esto que a priori parece una desventaja, y lo que provoca que escribir paquetes resulte más difícil, hace que su uso sea mucho más estandarizado y sencillo.

El método centralizado de descarga de paquetes se denomina CRAN (Comprehensive R Archive Network) [6]. Este repositorio mantenido por la *R foundation* permite descargar paquetes de forma muy sencilla sin necesidad de recurrir a métodos más complicados. Sin embargo, dado que la comprobación de los nuevos paquetes tiene que ser aceptado por un reducido grupo de personas, se generan dudas sobre la continuidad de este repositorio en el futuro.

3.1.1. Estructura

Los paquetes no necesitan una estructura específica, pero existen muchas convenciones para organizar el paquete.

- **R/**: contiene el código fuente en archivos .R. Se suele separa en archivos distintos si existen funciones que hagan cosas distintas.
- **man/**: contiene los archivos de coumentación en formato .Rd. Estos documentos se llaman cuando se llama a la ayuda de R para entender el funcionamiento de las funciones.
- **inst/**: incluye otros archivos que deban ser instalados con el paquete pero que no corresponden con el código fuente principal.
- **test/**: incluye archivos con pruebas del paquete. Esto puede incluir scripts de *testthat* que es el principal paquete en R para realizar pruebas unitarias.
- **vignettes/**: una *vignette* es una guía mucho más larga y completa que la documentación. Se trata normalmente de archivos RMarkdown, que simulan el flujo completo de uso de un paquete, no solo un uso simple.
- **DESCRIPTION**: es el archivo principal del paquete que contiene todos los metadatos para su uso. Incluye el nombre del paquete, la versión el autor o la licencia.

- **NAMESPACE**: define los objetos que se exportan y los paquetes que se importan y controla la visibilidad de las funciones y objetos del paquete.
- **README**: proporciona una visión general del paquete, como instalarlo y ejemplos básicos de uso. Es una práctica común en los repositorios de cualquier lenguaje de programación.

3.2. Funciones

Las funciones son las operaciones elementales que realizará nuestro paquete en R. En esta sección estableceremos las buenas prácticas y la forma de documentarlas correctamente mediante el paquete **roxygen2**. Las buenas prácticas incluyen las siguientes [7]:

- **Funciones simples y tareas concretas**: una función debe tener un único propósito, de tal manera que una sola función no se encargue de toda una tarea que pueda dividirse en tareas mas simples
- **Uso de parámetros**: toda las variables a las que vaya acceder una función deben ser pasadas como parámetros. Las funciones nunca deberían tener que acceder a variables globales y mucho menos deberían modificarlas, ya que esto provoca problemas de concurrencia. Una función con las mismas entradas siempre debería dar las mismas salidas, cosa que no ocurre si se accede a variables de fuera del ámbito de la función.
- **Uso de return**: otra buena práctica al escribir funciones en R es el uso del return aunque la función no devuelva nada. Se puede devolver **True** o **False**, dependiendo de las necesidades, y esto hace que se pueda comprobar que se sigue correctamente el flujo de la ejecución y facilita la legibilidad.
- **Consistencia en los tipos de datos devueltos**: una función siempre debe devolver datos del mismo tipo. Aunque parece simple para los tipos elementales esto también implica que no se devuelva algunas veces un dataframe de distintos tamaños o una lista con distinto número de elementos.
- **Manejo de excepciones**: el uso de excepciones permite que, ante un error en el código, no se corrompa todo el programa, sino que se maneje el error de manera controlada, permitiendo que el programa continúe ejecutándose o finalice de manera ordenada.
- **Uso de argumentos por defecto**: proporcionar valores por defecto sensatos provoca una mayor facilidad en el uso de las funciones.

3.3. Orientación a objetos

La orientación a objetos es una característica de los lenguajes de programación que permite estructura el código de manera modular y reutilizable. La orientación a objetos en *R* tiene ciertas particularidades respecto a otros lenguaje de programación ya que

R soporta varios sistemas de programación orientada a objetos [5], cada uno con sus propias características y usos. A continuación, se presentan los tres principales sistemas de orientación a objetos en R:

- **S3**
  - *Simplicidad*: S3 es el sistema de OO más simple y más antiguo en R. Es flexible y fácil de usar.
  - *Implementación basada en convención*: No requiere declaraciones formales de clases o métodos; en su lugar, se basa en convenciones de nombres.
  - *Métodos genéricos y despachadores*: Utiliza funciones genéricas y métodos definidos por convención. Por ejemplo, una función genérica como `print()` llamará al método `print.myclass` si el objeto pertenece a la clase `myclass`.
- **S4**
  - *Formalismo y rigor*: S4 es más estricto y formal que S3. Requiere declaraciones explícitas de clases y métodos.
  - *Definición de clases y métodos*: Utiliza las funciones `setClass` y `setMethod` para definir clases y métodos respectivamente.
  - *Validación*: Incluye validación automática de objetos para asegurar que los datos cumplan con las especificaciones de la clase.
- **R6**
  - *Encapsulamiento y herencia*: R6 proporciona un verdadero soporte para la encapsulación y la herencia, más parecido a otros lenguajes de OO como Python o Java.
  - *Referencias*: Los objetos R6 son modificados por referencia, lo cual es diferente de los sistemas S3 y S4 que son basados en copia.
  - *Métodos públicos y privados*: Permite definir métodos y campos públicos y privados, proporcionando un mejor control sobre la encapsulación.

En resumen, S3 es menos estructurado y mas flexible por lo que se utiliza para casos de uso sencillos. Si se necesita un método más formal se puede utilizar S4 y si se quiere una estructura de objetos más similar a la de otros lenguajes de programación se puede usar R6.

3.4. Documentación

La documentación en R se puede realizar de dos maneras principales. La primera es mediante el uso de archivos .Rd, que son archivos de documentación en formato LaTeX. Estos archivos permiten una documentación detallada y estructurada de las funciones, datos y paquetes. Cada archivo .Rd sigue una estructura específica y debe ser escrito manualmente por el usuario, asegurando que se cubran todos los aspectos relevantes como la descripción,

los argumentos, los valores de retorno, los ejemplos de uso y cualquier referencia adicional pertinente.

Esta forma de documentación es altamente detallada y permite un control exhaustivo sobre el contenido y la presentación de la información. Además, al ser basada en LaTeX, facilita la inclusión de fórmulas matemáticas y otros elementos complejos que pueden ser necesarios en la documentación técnica. La creación manual de estos archivos requiere familiaridad con la sintaxis y estructura de LaTeX, así como con las convenciones específicas de los archivos .Rd en R.

La segunda manera, y la más recomendada, es utilizando roxygen2 [2]. `Roxygen2` es un paquete que permite escribir la documentación directamente en los comentarios del código fuente, justo encima de las funciones que se están documentando, como ocurre en otros lenguajes de programación. Esto hace que la documentación sea más fácil de mantener y actualizar, ya que está directamente asociada con el código al que pertenece. El uso de `roxygen2` facilita la creación de archivos de documentación .Rd, ya que genera estos archivos automáticamente a partir de los comentarios estructurados en el código.

Etiqueta	**Descripción**
`@param`	Describe un parámetro de la función. Se debe incluir el nombre del parámetro seguido de una breve descripción.
`@return`	Explica lo que la función devuelve. Debe proporcionar una descripción clara del valor de retorno.
`@examples`	Proporciona ejemplos de cómo utilizar la función. Los ejemplos deben ser en formato de código R y mostrar casos de uso comunes.
`@details`	Proporciona detalles adicionales sobre la función que no se incluyen en las descripciones breves de `@param` y `@return`.
`@export`	Indica que la función debe ser exportada y, por lo tanto, estar disponible para los usuarios del paquete.
`@import`	Especifica los paquetes que deben ser importados para que la función funcione correctamente.
`@seealso`	Sugerencias de otras funciones o documentación relacionada que puede ser útil para el usuario.
`@author`	Indica el autor o los autores de la función.
`@references`	Proporciona referencias adicionales que pueden ser útiles para entender la función o su contexto.
`@note`	Añade notas adicionales que pueden ser importantes para el uso de la función.

Tabla 3.1: Etiquetas comunes de roxygen2 para la documentación en R

3.5. Estructuras de datos

Existen distintas estructuras de datos implementadas en *R* que se relacionan con la eficiencia. Para las distintas estructuras de datos es importante estudiar su notación *big-O*, para las principales operaciones. La notación O grande se refiere en ciencias de la computación a la manera de describir la complejidad de un algoritmo, ya sea en términos de memoria o de tiempo. Se trata de una estimación asintótica que proporciona una idea de cuanto va a crecer un algoritmo según aumente su tamaño de entrada. Esta medida nos comparar estructuras de datos distintas en relación a determinadas operaciones, tales como la inserción o la búsqueda.

- **Vectores**
  - **Características**:
    - Estructura de datos homogénea, se asigna una posición fija de memoria en la que se guardan los datos.
    - Almacena elementos del mismo tipo.
  - **Complejidad**:
    - *Acceso (Lectura)*: O(1)
    - *Escritura*: O(1)
    - *Búsqueda*: O(n)
    - *Inserción*: O(n) (excepto al final)
    - *Eliminación*: O(n)
- **Listas**
  - **Características**:
    - Estructura de datos heterogénea.
    - Puede contener elementos de diferentes tipos.
  - **Complejidad**:
    - *Acceso (Lectura)*: O(1)
    - *Escritura*: O(1)
    - *Búsqueda*: O(n)
    - *Inserción*: O(1) (al final), O(n) (en posiciones específicas)
    - *Eliminación*: O(n)
- **Matrices**
  - **Características**:
    - Estructura de datos bidimensional y homogénea.
    - Almacena elementos del mismo tipo.
  - **Complejidad**:
    - *Acceso (Lectura)*: O(1)
    - *Escritura*: O(1)

    - *Búsqueda*: O(n)
    - *Inserción*: O(n) (excepto al final)
    - *Eliminación*: O(n)

- **Data Frames**

  - **Características**:
    - Estructura compuesta por tantos vectores como columnas haya, unidos mediante referencias.
    - Estructura de datos bidimensional y heterogénea.
    - Cada columna puede contener datos de diferentes tipos.
  - **Complejidad**:
    - *Acceso a Columna (Lectura)*: O(1)
    - *Acceso a Fila (Lectura)*: O(n)
    - *Escritura en Columna*: O(1)
    - *Escritura en Fila*: O(n)
    - *Búsqueda*: O(n)
    - *Inserción de Columna*: O(n). Se deben modificar todas las filas y alocar nuevas posiciones en memoria.
    - *Inserción de Fila*: O(n)
    - *Eliminación de Columna*: O(n)
    - *Eliminación de Fila*: O(n)

Si se tienen datos muy fijos, en los que todos son del mismo tipo y no se necesitan insertar nuevos valores, es mucho más eficiente el uso de vectores y matrices, ya que los datos se colocan en memoria de forma contigua y su acceso es inmediato por sus direcciones de memoria. Sin embargo, si la naturaleza de los datos es más cambiante y se suelen insertar y mover los datos, es más útil recurrir a listas o a data frames. Una lista en *R* se guarda como una lista de punteros en *C* que apuntan a los lugares de memoria, a diferencia de las listas en otros lenguajes.

Todas las estructuras de datos ocupan memoria mientras estén referenciadas. El sistema encargado de liberar la memoria cuando una variable deja de estar referenciada se llama recolector de basura (*Garbage Collector*). En R, el recolector de basura se encarga de gestionar la memoria automáticamente, liberando espacio cuando las variables y objetos ya no son accesibles o referenciados en el código. Este proceso ayuda a prevenir fugas de memoria y a optimizar el uso de los recursos del sistema, permitiendo que los programas de R manejen eficientemente grandes volúmenes de datos y operaciones complejas sin intervención manual.

Capítulo 4

Desarrollo del paquete

En este capitulo se expone la forma en la que se ha ido desarrollando el paquete, su estructura y sus funcionalidades clave. Según la estructura explicada en 3.1.1, las funciones que se desarrollen se guardan en la carpeta **R/** en archivos .R . Para una mejor comprensión de estos archivos, se han dividido por casos de uso: lectura de los datos, mapas temáticos y resúmenes estadísticos.

4.1. Lectura de datos

La lectura de los datos se organiza dentro del fichero `lectura_datos.R`. Este fichero se compone de las siguientes funciones principales.

La primera función, `descargar_Archivo`, toma como parámetros un tipo de elección, un ámbito, un año y un mes, y opcionalmente una carpeta en la cual se depositarán los archivos descargados. El tipo de elección se establece mediante un *String*. En caso de que este no se corresponda con una de las posibilidades, se informa del error y se detiene la función. La descarga de los archivos se realiza concatenando al URL del área de descargas los parámetros establecidos en la función. Si no existe el directorio especificado como parámetro, se crea. Se comprueba que el archivo no exista ya, y si no existe, se descarga el archivo ZIP con los datos y se descomprime el archivo para poder acceder a los datos. Esta función dev uelve un *String* que contiene la ruta de los archivos descargados.

Por otro lado, se usan las funciones `comprueba_tipo_eleccion`, `comprueba_ambito` y `comprueba_mes`, las cuales hacen dos tareas al mismo tiempo. Por un lado, comprueban que la entrada introducida es válida. Es decir, verifican que un mes exista o que el tipo de elección esté entre aquellos que se pueden mostrar. Además, se encargan de traducir las entradas de un formato de texto más entendible a sus correspondencias en número para que puedan ser descargadas.

La última función, `leer_datos` es la función de alto nivel que se encarga de realizar las

llamadas a las otras funciones. A partir de un año, un mes, un tipo de elección, un ámbito y una tabla, devuelve un *data frame* en el que se introducen los datos solicitados. Esta función hace las llamadas a las otras funciones, A partir de un archivo fijo que contiene tablas con las descripciones de los datos, se leen los datos descargados y se almacenan en las variables obtenidas según los nombres y las posiciones de esas variables. Esta operación es crítica en el sentido del tiempo de ejecución, ya que se trata de leer un gran número de filas. Por ello se ha optimizado el algoritmo, varias veces hasta que se han obtenido tiempos de ejecución razonables. El tiempo de ejecución para la lectura de datos de un año 2019, para las elecciones del Congreso, por municipio se puede ver en la tabla 4.1. Siendo

	user	**system**	**elapsed**
Tiempo (s)	6.06	0.03	6.17

Tabla 4.1: Tiempos de ejecución para la lectura de los datos

1. **user**: El tiempo de CPU en segundos que el proceso de R ha utilizado directamente .
2. **system**: El tiempo de CPU en segundos que el sistema operativo ha utilizado en nombre del proceso de R. Esto incluye tareas como la gestión de archivos, la asignación de memoria y otras operaciones del sistema.
3. **elapsed**: El tiempo total real transcurrido en segundos desde el inicio hasta la finalización de la ejecución. Este es el tiempo que se percibe desde el punto de vista del usuario (*wall-clock time*).

El diagrama de flujo de la lectura completa de datos se puede ver en la figura 4.1.

Además de estas funciones principales, se han creado otras funciones auxiliares para facilitar la lectura de los datos. Por ejemplo, se ha implementado la función `leer_varias_tablas`, que encapsula `leer_datos`. Esta función permite pasar un vector con los nombres de las tablas deseadas y devuelve una lista de *data frames* con los datos correspondientes.

La función `get_elecciones`, a partir de un archivo CSV que contiene las elecciones celebradas hasta la redacción de este TFG, devuelve un *data frame* con las fechas, tipos y URLs de las elecciones. Dispone de dos parámetros opcionales: tipo y año. Si no se proporciona ninguno, devuelve todas las elecciones registradas hasta el momento. Si se especifica uno de los dos filtros, se devuelve la información correspondiente a ese filtro específico.

La obtención de los URL correctos se ha realizado mediante solicitudes `HEAD` de HTTP, una metodología que permite verificar la validez de un enlace sin necesidad de descargar todo el contenido asociado. Estas solicitudes son particularmente eficientes porque solo recuperan los encabezados HTTP del recurso solicitado, lo cual consume menos ancho de banda que una solicitud `GET` completa. Los enlaces que devuelven un código **200** se consideran existentes, y sus datos se almacenan en el archivo CSV junto con el tipo de elección.

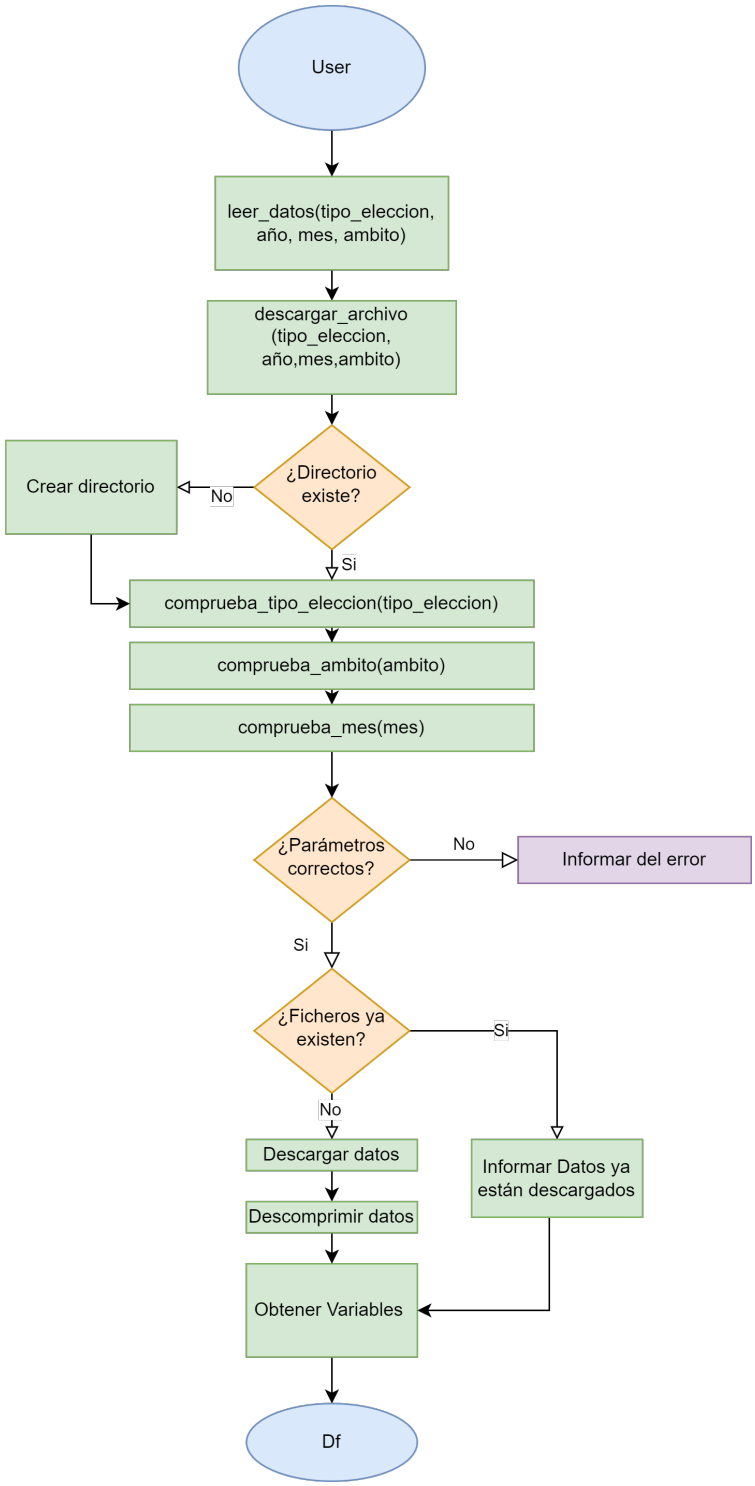

Figura 4.1: Diagrama de flujo del proceso de descarga y procesamiento de datos electorales

Dado que las tablas disponibles dependen del tipo de elección se ha creado una función bastante trivial `get_tablas_disponibles`, a la que se le pasa el tipo de elección y se obtiene las tablas disponibles.

Por último, se ha creado la función `descripcion_tabla` que devuelve la descripción de la tabla especificada como parámetro.

En resumen, el fichero `LeerDatos` tiene las funciones especificadas en la tabla 4.2.

Nombre de la Función	**Parámetros**	**Devuelve**
`descargar_Archivo`	tipo_eleccion, año, mes, ambito, directorio (opcional)	Ruta del directorio donde se ha descargado el archivo
`comprueba_tipo_eleccion`	tipo_eleccion	Código correspondiente al tipo de elección
`comprueba_ambito`	ambito	Código correspondiente al ámbito
`comprueba_mes`	mes	Mes formateado a dos dígitos
`leer_tabla`	tipo_eleccion, año, mes, ambito, directorio, tabla (opcional)	Data frame con los datos procesados de las elecciones
`get_elecciones`	tipo (opcional), año (opcional)	Data frame de elecciones filtradas por tipo y/o año
`get_tablas_disponibles`	tipo_eleccion	Vector de caracteres con los prefijos de ficheros para el tipo especificado
`descripcion_tabla`	tabla	Descripción del fichero asociado al prefijo dado
`leer_varias_tablas`	tipo_eleccion, año, mes, ambito, directorio, tablas	Lista de data frames, cada uno conteniendo los datos procesados de una tabla específica

Tabla 4.2: Resumen de Funciones en el fichero `LeerDatos`

4.2. Mapas

La representación mediante mapas es una herramienta muy poderosa para el análisis de elecciones, ya que proporciona una forma más intuitiva de visualizar los resultados por unidades territoriales en comparación con, por ejemplo, una tabla equivalente. Sin embargo, para interpretar correctamente estos gráficos, se presupone que el lector tiene conocimiento de la información territorial. En el caso de los mapas desarrollados, dado que representan las provincias españolas, se asume que estas son conocidas por los lectores.

Para el análisis de los datos de elecciones mediante mapa se han creado dos mapas que analizan todo el territorio nacional, dependiendo de si se quiere representar mediante provincias o agrupar por comunidades autónomas. Estas dos funciones están en el archivo `Mapas.R`.

La función `crear_mapa_provincias` tiene como objetivo crear un mapa de las provincias de España coloreado según una variable específica de un *data frame*. A continuación, se describen los pasos que sigue la función.

Primero, se convierte el argumento de la variable a una cadena de caracteres y se limpia para obtener el nombre real de la variable. Luego, se crea un *data frame* con los códigos INE y los nombres de las provincias españolas.

Se obtienen los datos geográficos de las provincias utilizando la función `gisco_get_nuts`. A continuación, se seleccionan y renombran las columnas relevantes del *data frame* de provincias españolas. Se agrupan los datos por código INE de provincia y se calcula la suma total de la variable especificada.

La función `gisco_get_nuts` se utiliza para obtener datos geoespaciales de las unidades territoriales de la Nomenclatura de Unidades Territoriales Estadísticas (NUTS) de Eurostat. Proporciona polígonos geográficos detallados de varias subdivisiones administrativas, como países, regiones y provincias. En este caso, se obtienen los límites geográficos de las provincias españolas (NUTS nivel 3) con una resolución específica, lo que permite visualizar correctamente las áreas geográficas en los mapas creados.

Posteriormente, se combinan los *data frames* de provincias seleccionadas, códigos INE y datos de población. Se realiza un tratamiento especial para las Islas Canarias, ya que la posición real de las Islas Canarias no es óptima para la visualización. Por ello, se ha decidido mover 18 grados al norte y 7 grados al este, de forma que quede representada en el sureste de la península. De manera similar, se necesita juntar los datos obtenidos para cada una de las islas en las provincias correspondientes. En el caso de Canarias se agrupan en Santa Cruz de Tenerife y Las Palmas, mientras que las Islas Baleares se agrupan juntas. Finalmente, se combinan todos los datos y se crea el mapa utilizando `ggplot2`. La figura 4.2 es un ejemplo de salida de esta función.

La segunda función que se implementa es análoga a la primera con las comunidades autónomas. Sin embargo, hay que hacer varias modificaciones para poder agrupar las provincias en la CCAA. Para ello se crea un *data frame* que mapea para cada código del INE las provincias y las Comunidades Autónomas.

Para visualizar el mapa, se utiliza `geom_sf` de `ggplot2`, que es adecuado para trabajar con datos espaciales. La escala de colores se define mediante `scale_fill_viridis_c`, usando la opción "D"que proporciona una paleta de colores con alto contraste, lo que permite una buena visualización de los datos. Además, los números en la leyenda de colores se formatean con `label_number` de la biblioteca `scales` para mejorar la legibilidad, eliminando la notación científica y añadiendo comas como separadores de miles. El título del mapa se genera dinámicamente utilizando la función `paste` para incluir el nombre de la variable visualizada.

Un ejemplo del uso de la función `crear_mapa_CCAA` se puede ver en la figura 4.3.

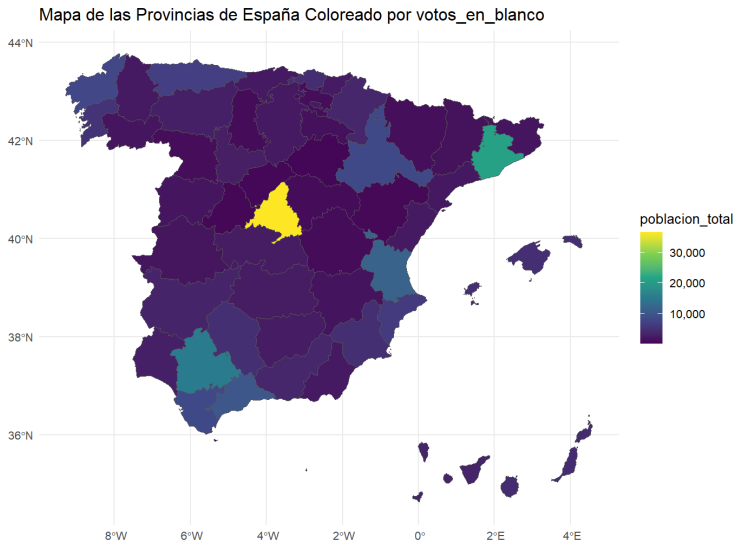

Figura 4.2: Ejemplo de representación de los votos en blanco por provincias

Dado que los datos que se obtienen vienen agregados por unidades más pequeñas que las provincias se ha creado la función `mapa_provincia_secciones` que representa los datos en función de los municipios de una provincia. Esta función toma como parámetros un dataframe, la columna del dataframe que se quiere representar y el nombre de la provincia que se quieren tomar los datos. Como medida para evitar los fallos al escribir la provincia se ha implementado la distancia de Levenshtein [9]. La distancia de Levenshtein es una métrica para medir la diferencia entre dos secuencias. Es especialmente útil en corrección ortográfica. La distancia se define como el número mínimo de ediciones (inserciones, eliminaciones o sustituciones de un solo carácter) necesarias para cambiar una cadena de texto en otra.

Dados dos cadenas, a y b, su distancia de Levenshtein se calcula de la siguiente manera:

1. Si una de las cadenas está vacía, la distancia es la longitud de la otra cadena.
2. Si las cadenas son iguales, la distancia es 0.
3. Si las cadenas son diferentes, se considera el costo mínimo entre:
   - Insertar un carácter en una cadena.
   - Eliminar un carácter de una cadena.
   - Sustituir un carácter por otro.

La distancia se calcula utilizando un algoritmo que construye una matriz de tamaño $m+1 \times n+1$, donde m y n son las longitudes de las cadenas a y b respectivamente. La celda

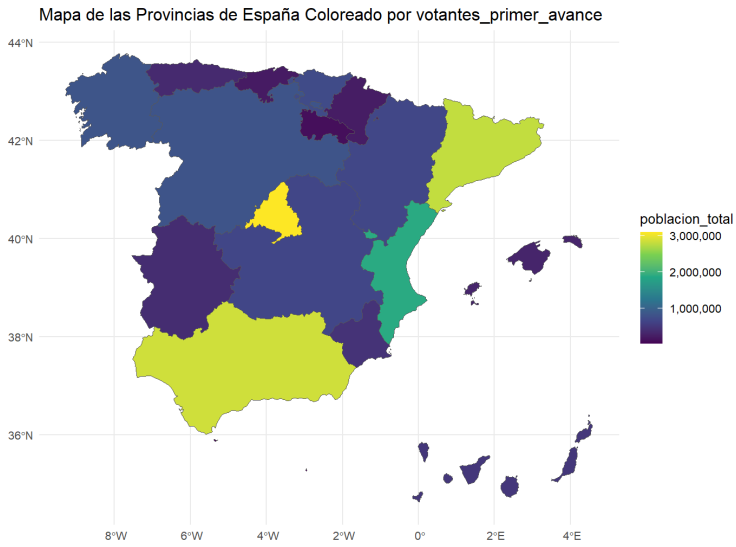

Figura 4.3: Ejemplo de representación de los votos en primera ronda por Comunidades Autónomas

en la posición (i, j) de la matriz representa la distancia de Levenshtein entre los primeros i caracteres de a y los primeros j caracteres de b.

Formalmente, la distancia se calcula con la siguiente función recursiva:

$$\text{lev}_{a,b}(i,j) = \begin{cases} \text{máx}(i,j) & \text{si } \text{mín}(i,j) = 0, \\ \text{mín} \begin{cases} \text{lev}_{a,b}(i-1,j) + 1, \\ \text{lev}_{a,b}(i,j-1) + 1, \\ \text{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j} \end{cases} & \text{si } i,j > 0. \end{cases}$$

donde $\text{cost}(a_i, b_j)$ es 0 si $a_i = b_j$ y 1 si $a_i \neq b_j$, representando las operaciones de sustitución, inserción, y eliminación respectivamente.

Dadas las palabras “Barcelona” y “Barceclona”, analizando la transformación requerida:

- Barcelona → Barceclona (inserción de ’c’ después de la segunda ’c’, costo = 1)

Por lo tanto, la distancia de Levenshtein entre “Barcelona” y “Barceclona” es 1, ya que solo se requiere una inserción para transformar una palabra en la otra.

La distancia de Levenshtein se ha implementado en la función `get_min_distancia_Levenshtein` que tiene como parámetros la cadena a evaluar y un conjunto finito de posibles valores, y devuelve aquel valor con el que la distancia sea mínima. Esto es muy útil como método de control de errores para no tener que informar al usuario cada vez que comete un error menor, y parar el flujo de ejecución de la función.

El funcionamiento de la función `mapa_provincia _secciones` es el siguiente: se obtiene la información geográfica mediante la función `est_get_numeric()` del paquete `mapSpain`. Para hacer el *join* entre el *dataframe* con los datos y el observado hay primero que filtrar ambos para la provincia que se necesita y después se comparan los identificadores de municipio y se crea el mapa. La Figura 4.4 representa la proporción del voto total que representa el voto en blanco por municipios de la provincia de Valladolid.

```
1    mapa_provincia_secciones(df,df$prop_votos_blanco,"valladolid")
```

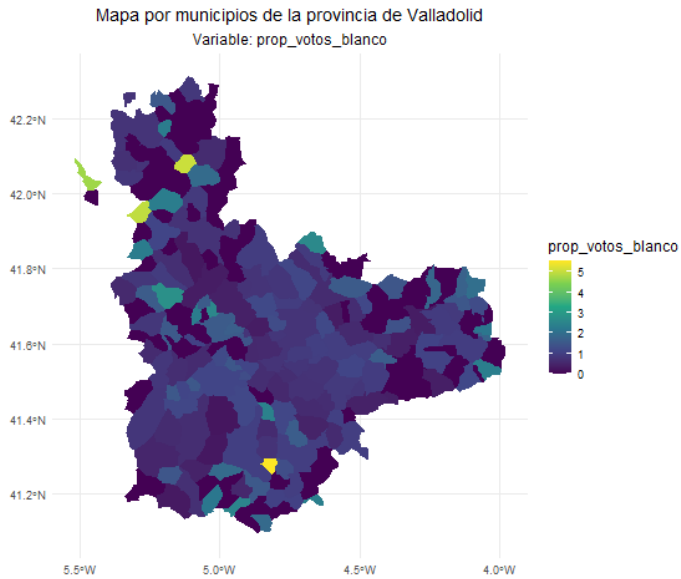

Figura 4.4: Ejemplo de representación de la proporción de votos en blanco por municipios en la provincia de Valladolid.

La Tabla 4.3 resume las funciones relativas a los mapas creadas.

4.3. Tablas Resumen

El último fichero, el fichero `Tablas.R` implementa una tabla de resumen de una elección específica. Para ello se utiliza la función `tabla_participacion`, que en función de la elección especificada imprime por pantalla un resumen de la participación y devuelve un *dataframe* opcional con esas medidas.

Nombre de la Función	Parámetros	Devuelve
`crear_mapa_provincias`	df,df$variable	Mapa de las provincias de España coloreado por la variable especificada.
`crear_mapa_CCAA`	df,df$variable	Mapa de las comunidades autónomas de España coloreado por la variable especificada.
`mapa_provincia_secciones`	df,df$variable, provincia	Mapa de la provincia por municipios coloreado por la variable especificada.
`get_min_distancia_Levenshtein`	cadena, conjunto	Cadena del conjunto cuya distancia a la cadena original sea mínima

Tabla 4.3: Descripción de Funciones para la Creación de Mapas

La función `tabla_participación` acepta como parámetros el tipo de elección, el año, el mes y opcionalmente la provincia. Si no se especifica una provincia se devuelven los resultados agregados para toda España. Lo primero que hace esta función es llamar a la función *leer_tabla*, del archivo de lectura de datos para obtener los datos electorales de esa elección. Dado que para esta tabla resumen no se necesita la información por mesas, se establece el parámetro ámbito a municipio y se selecciona la tabla 5 como aquella que se quiere obtener. Una vez se tiene el *dataframe*, en función del parámetro de provincia, se realiza una selección de los datos de esa provincia o no. Dado que existe la circunstancia, de que el nombre de la provincia introducida no corresponda con el que se implementa en el método, en especial por la diferente nomenclatura de las provincias en distintas lenguas cooficiales, se ha establecido que ante un nombre de provincia erroneo, se devuelva un mensaje de error seguido por el nombre de las provincias posibles. La figura 4.5 es el diagrama de flujo de como funciona este caso de uso.

Para aportar modularidad al código, se ha separado las operaciones de cálculo de las variables y la impresión por pantalla en la función `resultados_tabla_participacion`. Esta tabla toma como parámetros el df, del que ya se ha seleccionado la provincia, y el tipo de elección. Esta información se necesita porque en el caso de los referéndum, es útil conocer también el porcentaje de votos afirmativos y el de negativos. Se calculan para todas las variables estudiadas la magnitud absoluta y la relativa. Para todas las elecciones se obtienen los votos en primer y segundo avance, y el porcentaje acumulado de cada uno de ellos. Además se muestra, los votos en blancos y los votos nulos y su proporción respecto al total, que es la suma de estos dos con los votos a candidaturas. En el caso de que no sea un referéndum se muestra la cantidad y porcentaje de votos a candidaturas. En el caso del referéndum, se suman los votos afirmativos con los negativos y se comparan con los votos nulos y en blanco y se muestran los votos afirmativos y negativos separados. A continuación se muestran las tablas obtenidas para el referéndum de 2005 en la provincia de Valladolid.

```
1 tabla_participacion('referendum',2005,2,'valladolid')
```

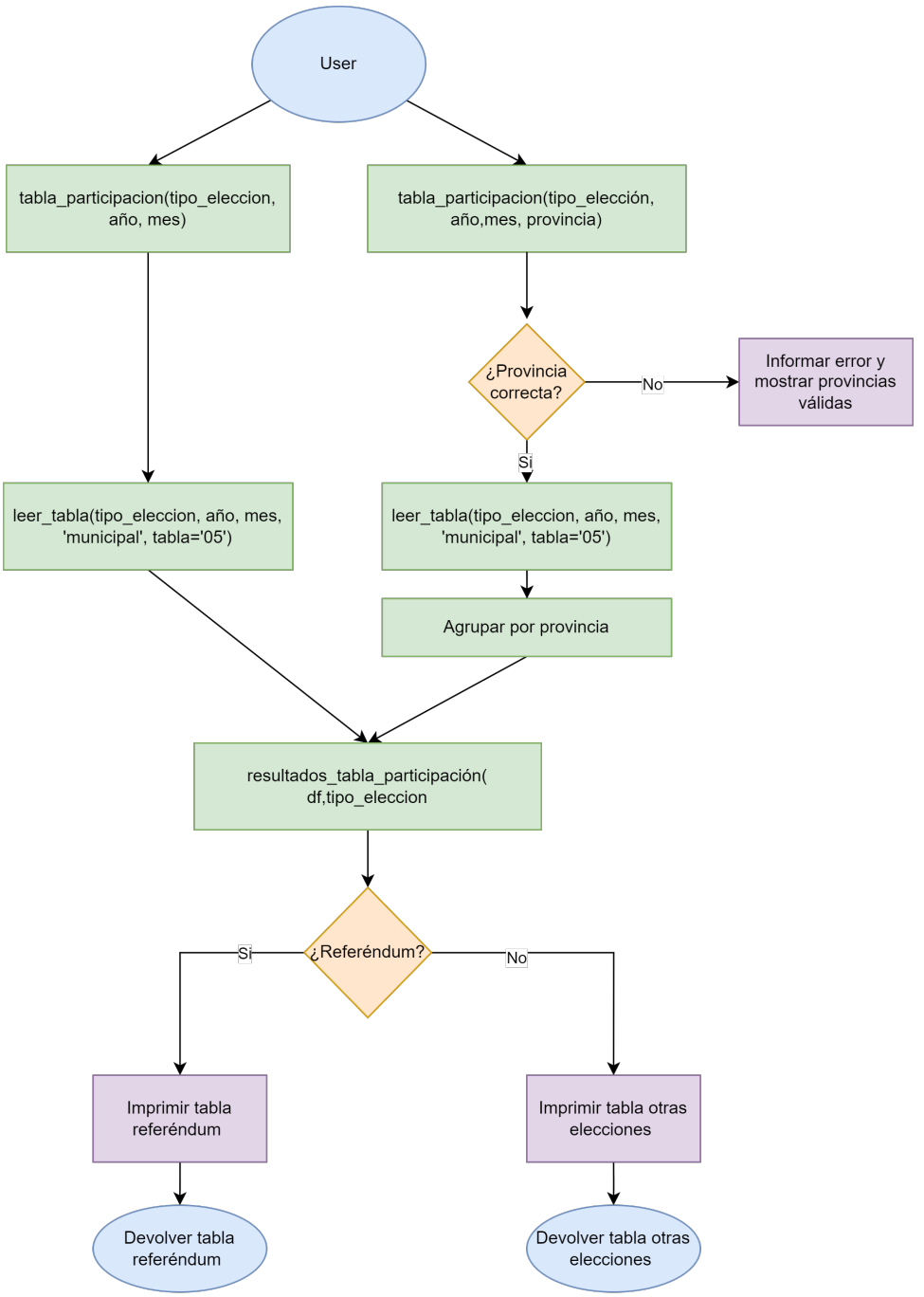

Figura 4.5: Diagrama de flujo del proceso de creación de tabla participación

```
Datos Referéndum     |         Totales |   Porcentaje
-------------------------------------------------------------
Primer avance        |          107333 |        50.84%
Segundo avance       |          162734 |        77.08%
-------------------------------------------------------------
Votos en blanco      |           14599 |         6.91%
Votos nulos          |            2306 |         1.09%
Votos referéndum     |          194218 |        91.99%
-------------------------------------------------------------
Votos afirmativos    |          159555 |        82.15%
Votos negativos      |           34663 |        17.85%
-------------------------------------------------------------
```

Este es el resultado de la llamada que muestra la elección al congreso de abril de 2019.

```
1 tabla_participacion('congreso',2019,4)
```

```
Datos                |         Totales |   Porcentaje
-------------------------------------------------------------
Primer avance        |        16817259 |        54.81%
Segundo avance       |        24650054 |        80.34%
-------------------------------------------------------------
Votos en blanco      |          227958 |         0.74%
Votos nulos          |          306306 |         1.00%
Votos candidaturas   |        30148642 |        98.26%
-------------------------------------------------------------
```

La función `tabla_participacion` devuelve un *dataframe* invisible, es decir, con la llamada a esta función no se muestra en la consola el *dataframe*, pero si se asigna la función a una variable si que se obtiene. Este *dataframe* contiene la misma información que se ha obtenido en la tabla.

La Tabla 4.4 es un resumen de estas dos funciones.

Nombre de la Función	**Parámetros**	**Devuelve**
`tabla_participacion`	tipo_eleccion, año, mes, provincia (opcional)	Un dataframe invisible con los resultados de participación y una impresión en consola de los resultados detallados.
`resultados_tabla_participacion`	df, tipo_eleccion	Un dataframe con las categorías de votación y sus totales y porcentajes.

Tabla 4.4: Descripción de Funciones para la obtención de tablas

Capítulo 5

Conclusiones y líneas de trabajo futuras

En este capitulo se estudian las conclusiones, estudiadas sobre todo en base a los objetivos fijados en la introducción, y posibles líneas de trabajo futuras.

Se ha conseguido desarrollar un paquete en R que extiende bastante las funcionalidades de R para adaptarse a los datos de elecciones del gobierno. Respecto a la consecución de los objetivos podemos establecer lo siguiente:

- Se han creado funciones que implementan diversas funciones para la lectura de los datos. Se han creado funciones que descargan los datos, los descomprimen y obtienen de ellos la tabla que se quiera analizar. Las funciones creadas han sido implementadas teniendo en cuenta todos los posibles flujos de ejecución teniendo un correcto manejo de los posibles errores. Los procesos de lectura de datos al ser críticos en uso de recursos han sido optimizados para obtener unos tiempos de ejecución razonables.

  Aparte de las funciones principales de lectura de datos, se han creado numerosas funciones auxiliares para facilitar al usuario información sobre a que datos se puede acceder. Se ha creado una función para la obtención de las elecciones en función de su tipo o de su año, otra para obtener las tablas disponibles para cada tipo de elección, una para obtener una descripción de la tabla y por último una función que facilite el proceso de lectura de varias tablas.

- Se ha creado un fichero que contiene tres funciones para la obtención de los mapas dependiendo de si se quiere visualizar por provincias o por comunidades autónomas, o por municipios de una provincia. Dado que la función de estas funciones es el de graficar los mapas, y es independiente del contexto de los datos, siempre que se pase como parámetros un dataframe con una columna que sea `codigos_provincias_ine` y la columna que se quiere visualizar, los mapas serán correctos y se podrá visualizar correctamente. Además se ha implementado un sistema para evitar errores en la introducción de datos por parte del usuario basado en la distancia de Levenshtein.

- Por último, se han creado tablas resumen para algunos datos. Se ha creado una tabla que establece unos parámetros de participación. Esta tabla se puede obtener para una provincia o para todo el país. Sin embargo, para esta función solo se han utilizado datos de la tabla número 5 de las que se obtienen al descargar los datos. Se podrían realizar más tablas similares que faciliten la comprensión de los datos.

5.1. Lineas de trabajo futuras

Se pueden marcar dos líneas sobre las que ampliar el presente trabajo: mejoras en la eficiencia de las funciones creadas y la creación de nuevas funciones. Respecto a la eficiencia, dado que se ha intentado optimizar los algoritmos implantados sobre todo en aquellas partes críticas en recursos, como la lectura de los datos, no hay muchas opciones de ganancia en términos de eficiencia. Se puede comprobar además que los tiempos de ejecución han sido razonables. Un matiz que surge al terminar el desarrollo de *Sotware* es la aparición de errores que no han sido controlados durante la fase de test. Algunos de estos errores son inevitables, por lo que la depuración del código creado también entra como línea de trabajo futura.

Respecto a la creación de nuevas funciones, se pueden ampliar más para tener un paquete más completo, y más visual para la tarea de visualización de los datos. Se podría usar Shiny [1] para obtener visualizaciones interactivas. El uso de Shiny en la visualización de datos electorales permite a los usuarios interactuar con los datos de formas que los métodos estáticos no pueden igualar. Por ejemplo, para los mapas obtenidos se podrían establecer botones para controlar las provincias en las que hay algún suceso.

Por último, respecto al nivel de abstracción de los datos, se han obtenido para comunidades autónomas, provincias y municipios. La base de datos de elecciones del gobierno da niveles de abstracción a nivel de mesa y el INE proporciona mapas de secciones censales, por lo que se podrían crear funciones que analizaran los datos a estos niveles de abstracción.

Apéndice A

Código Fuente

Todo el código fuente desarrollado se puede encontrar en el repositorio `https://github.com/ignacio-dominguez-calvo/paquete.elecciones`

A.1. LecturaDatos.R

```
1
2  #' Descargar Archivo
3  #'
4  #' Descarga un archivo desde la fuente de datos electorales del gobierno
5  #'
6  #' @param tipo_eleccion Tipo de elecci n. Puede ser 'referendum', 'congreso',
       'senado', 'municipales', 'cabildos' o 'europeas'.
7  #' @param a o A o de la elecci n.
8  #' @param mes Mes de la elecci n.
9  #' @param ambito mbito de la elecci n. Puede ser 'mesa', 'municipio' o '
       superior'.
10 #' @param directorio Directorio donde se guardar el archivo descargado. Por
       defecto es "./descargas/".
11 #'
12 #' @return La ruta del directorio donde se ha descargado el archivo.
13 #'
14 #' @details Esta funci n descarga un archivo en base a unos parametros que
       define cada elecci n, de los que se genera una URL de la que obtener los
       datos.
15 #'
16 #'
17 #' @examples
18 #' # Descargar archivo de elecciones municipales del a o 2023, mes 05, en el
       mbito mesa
19 #' descargar_Archivo(tipo_eleccion = "municipales", a o = 2023, mes = '05',
       ambito = "mesa")
20 #'
21 #' @importFrom utils download.file unzip
22 #' @export
```

```
#' @seealso https://infoelectoral.interior.gob.es/es/elecciones-celebradas/area-de-descargas/ URL base de descarga de archivos
#' @author Ignacio Dom nguez Calvo
#' @keywords archivo descarga elecci n elecciones
#'
descargar_Archivo <- function(tipo_eleccion, a o , mes, ambito, directorio = "./descargas/") {
  # Diccionarios con los par metros admitidos para tipo eleccion y ambito y sus correspondencias


  tipo_eleccion<-comprueba_tipo_eleccion(tipo_eleccion)
  ambito<-comprueba_ambito(ambito)
  mes<-comprueba_mes(mes)

  #Si la carpeta especificada no existe, se crea
  if (!file.exists(directorio)) {
    dir.create(directorio)
  }
  url_base <- "https://infoelectoral.interior.gob.es/estaticos/docxl/apliextr/"
  nombre_archivo <- paste(tipo_eleccion, a o , mes, '_', ambito, ".zip", sep = "")
  nombre_archivo <- trimws(nombre_archivo)
  url_completo <- paste(url_base, nombre_archivo, sep = "")
  directorio <- trimws(directorio)  # Eliminar espacio en blanco al final de directorio
  ruta <- file.path(directorio, nombre_archivo)  # Usar file.path para construir la ruta
  if (!file.exists(ruta)) {
    download.file(url_completo, destfile = ruta, mode = "wb")
    unzip(ruta, exdir = directorio, overwrite = FALSE, junkpaths = FALSE)  # Usar solo el directorio como exdir
  } else {
    message("El archivo ya existe en el directorio especificado.")
  }
  return(directorio)
}



#' Comprueba el tipo de elecci n y devuelve su c digo correspondiente.
#'
#' Esta funci n toma un tipo de elecci n y devuelve su c digo correspondiente
#' seg n un diccionario predefinido.
#'
#' @param tipo_eleccion El tipo de elecci n a comprobar.
#' @return El c digo correspondiente al tipo de elecci n.
#' @examples
#' comprueba_tipo_eleccion("congreso")
comprueba_tipo_eleccion <- function(tipo_eleccion){
  tipo_eleccion<-tolower(tipo_eleccion)
  dict_elec <- list(
    'referendum' = '01',
    'refer ndum' = '01',
    'congreso' = '02',
    'senado' = '03',
    'municipales' = '04',
```

```
    'cabildos' = '06',
    'europeas' = '07'
  )
  if (!tipo_eleccion %in% names(dict_elec)) {
    stop("El tipo de elecci n proporcionado no es v lido. Tipos v lidos:
    referendum, congreso,senado,municipales,cabildos,europeas")
  }

  return(dict_elec[[tipo_eleccion]])
}

#' Comprueba el  mbito  y devuelve su c digo correspondiente.
#'
#' Esta funci n toma un  mbito  y devuelve su c digo correspondiente seg n
    un diccionario predefinido.
#'
#' @param ambito El  mbito  a comprobar.
#' @return El c digo correspondiente al  mbito .
#' @examples
#' comprueba_ambito("municipio")
comprueba_ambito <- function(ambito){
  ambito <- tolower(ambito)
  dict_ambito <-list(
    'mesa' = 'MESA',
    'municipio' = 'MUNI',
    'superior' = 'TOTA'
  )

  if (!ambito %in% names(dict_ambito)) {
    stop("El  mbito  proporcionado no es v lido.  mbitos  v lidos: mesa,
    municipio o superior")
  }
  return(dict_ambito[[ambito]])
}

#' Comprueba el mes y lo formatea si es un n mero.
#'
#' Esta funci n comprueba si el mes es un n mero y lo formatea a dos
    d gitos si es necesario.
#'
#' @param mes El mes a comprobar y formatear.
#' @return El mes formateado.
#' @examples
#' comprueba_mes(5)
comprueba_mes<-function(mes){
  if (is.numeric(mes)) {
    # Si el mes es un n mero, lo formateamos a dos d gitos
    mes <- sprintf('%02d', mes)
  }
  return(mes)
}



#' Leer Datos de Elecciones
#'
#' Esta funci n descarga y procesa archivos de datos electorales seg n los
    par metros especificados.
#'
```

```
127 #' @param tipo_eleccion Un car cter que indica el tipo de elecci n de entre
      referendum,congreso,senado,municipales,cabildos o europeas.
128 #' @param a o Un n mero que indica el a o de la elecci n.
129 #' @param mes Un n mero que indica el mes de la elecci n.
130 #' @param ambito Un car cter que indica el  mbito  de la elecci n que puede
      ser mesa,municipio o superios.
131 #' @param directorio Un car cter que indica el directorio donde se guardar n
       los archivos descargados. Por defecto es "./descargas/".
132 #' @param tabla Un car cter que indica el c digo de la tabla de datos a
      procesar. Por defecto es '05'.
133 #' @return Un data.frame con los datos procesados de las elecciones.
134 #' @importFrom readxl read_excel dplyr
135 #' @export
136 leer_tabla <- function(tipo_eleccion, a o , mes, ambito, directorio = "./
      descargas/", tabla = '05') {
137   suppressWarnings({
138   directorio <- descargar_Archivo(tipo_eleccion, a o , mes, ambito, directorio
        )
139   tablas_path <- "C:\\Users\\nacho\\OneDrive\\Escritorio\\TFG ESTADISTICA\\
        paquete.elecciones\\inst\\extdata\\tablas_finales_2.xlsx"
140
141   if (tabla == '11' || tabla == '12') {
142     sheet_name = paste(tabla, "04aamm.DAT", sep = "")
143   } else {
144     sheet_name = paste(tabla, "xxaamm.DAT", sep = "")
145   }
146
147   tabla_variables <- read_excel(tablas_path, sheet = sheet_name)
148   tabla_variables <- tabla_variables[-c(1, 2), , drop = FALSE]
149   ultimos_dos_digitos <- substr(a o , nchar(a  o) - 1, nchar(a  o))
150
151
152   mes<-comprueba_mes(mes)
153   tipo_eleccion<-comprueba_tipo_eleccion(tipo_eleccion)
154   nombres_columnas <- unique(tabla_variables$Variable)
155   nuevo_df <- data.frame(matrix(ncol = length(nombres_columnas)))
156   names(nuevo_df) <- nombres_columnas
157
158   ruta<- paste(directorio,tabla,tipo_eleccion,ultimos_dos_digitos,mes,".DAT",
        sep="")
159   file_content <- readLines(ruta, encoding = 'UTF-8')
160   file_content <- iconv(file_content, "latin1", "UTF-8")
161
162   lista_valores <- vector("list", length = length(file_content))
163
164   for (index in seq_along(file_content)) {
165     municipio <- file_content[index]
166     # Extraer los valores para la l nea actual
167     valores <- mapply(function(inicio, fin, tipo) {
168       valor <- substr(municipio, as.integer(inicio), as.integer(fin))
169       if (grepl("Num.", tipo)) {
170         return(as.integer(valor))
171       }
172       return(valor)
173     }, tabla_variables$Inicio, tabla_variables$Fin, tabla_variables$Tipo)
174
175     # Almacenar los valores en la lista
176     lista_valores[[index]] <- valores
177   }
```

```
178
179   # Convertir la lista en un DataFrame
180   nuevo_df <- do.call(rbind, lista_valores)
181   if ("nombre_municipio" %in% names(nuevo_df)) {
182     nuevo_df[, nombre_municipio <- trimws(nombre_municipio, which = "right")]
183   }
184
185   # Transformar columnas que se puedan convertir a n mero
186   nuevo_df <- as.data.frame(nuevo_df)  # Aseg rate de que es un dataframe
187   nuevo_df[] <- lapply(nuevo_df, function(x) {
188     if(is.character(x)) {
189       # Intenta convertir a num rico
190       y <- as.numeric(x)
191       # Si la conversi n falla completamente (solo NA's), devuelve x original
      , de lo contrario y
192       if(all(is.na(y))) x else y
193     } else {
194       x  # Devuelve la columna sin cambios si no es de tipo caracter
195     }
196   })
197   names(nuevo_df) <- nombres_columnas
198
199   # Mostrar el nuevo DataFrame
200   return(nuevo_df)
201
202 }
203
204 #' Obtener Datos de Elecciones Filtrados
205 #'
206 #' Esta funci n lee un archivo CSV que contiene el hist rico de elecciones y
       a partir de ella devuelve
207 #' todas las elecciones que se han celebrado hasta el momento.
208 #'
209 #' @param tipo El tipo de elecci n a filtrar, como 'referendum', 'congreso',
     etc.
210 #'        Si se especifica, la funci n filtrar  los datos para incluir solo
     las entradas
211 #'        que correspondan a este tipo. Debe ser uno de los siguientes:
212 #'        'referendum', 'congreso', 'senado', 'municipales', 'cabildos', '
     europeas'.
213 #'        Si es `NULL`, no se aplica ning n filtro por tipo.
214 #' @param a o El a o de las elecciones a filtrar.
215 #'        Si se especifica, la funci n filtrar  los datos para incluir solo
     las entradas
216 #'        de ese a o espec fico. Si es `NULL`, no se aplica ning n filtro
     por a o.
217 #'
218 #' @return Un data.frame que contiene las columnas del archivo CSV original,
219 #'         pero filtrado seg n los criterios especificados y con los c digos
       de tipo
220 #'         de elecci n traducidos a texto.
221 #' @examples
222 #' get_elecciones(tipo = "congreso", a o = 2021)
223 #' get_elecciones(a o = 2020)
224 #' get_elecciones(tipo = "municipales")
225 #' @import dplyr
226 #' @export
227 #'
228 #' @importFrom readr read_csv
```

```
229 get_elecciones <- function(tipo = NULL, a o = NULL) {
230   # Diccionario de tipo a c digo
231   if(!is.null(tipo)){
232     tipo<-tolower(tipo)
233   }
234   dict_elec_text_to_num <- list(
235     'referendum' = 1,
236     'congreso' = 2,
237     'senado' = 3,
238     'municipales' = 4,
239     'cabildos' = 6,
240     'europeas' = 7
241   )
242
243   # Diccionario de c digo a tipo (los c digos son enteros sin ceros a la
        izquierda)
244   dict_elec_num_to_text <- list(
245     '1' = 'referendum',
246     '2' = 'congreso',
247     '3' = 'senado',
248     '4' = 'municipales',
249     '6' = 'cabildos',
250     '7' = 'europeas'
251   )
252   datos <- read.csv("C:/Users/nacho/OneDrive/Escritorio/TFG ESTADISTICA/
        paquete.elecciones/inst/extdata/entradas_validas.csv", stringsAsFactors =
        FALSE)
253   if (!is.null(tipo) && tipo %in% names(dict_elec_text_to_num)) {
254     codigo_tipo <- dict_elec_text_to_num[[tipo]]
255     datos <- datos %>% filter(Tipo == codigo_tipo)
256   }
257
258   if (!is.null(a o)) {
259     datos <- datos %>% filter(A o == a o)
260   }
261
262   # Traducir los c digos de tipo a texto
263   datos$Tipo <- sapply(datos$Tipo, function(x) dict_elec_num_to_text[[as.
        character(x)]])
264
265   return(datos)
266 }
267 #' Obtener los prefijos de tablas disponibles para un tipo de elecci n
        espec fico
268 #'
269 #' Esta funci n devuelve los prefijos de los ficheros asociados con un tipo
        espec fico
270 #' de elecci n , como refer ndum , congreso , senado , etc.
271 #'
272 #' @param tipo_eleccion Una palabra que indica el tipo de elecc n.
273 #' @return Un vector de caracteres con los prefijos de ficheros para el tipo
        de elecci n especificado.
274 #' @examples
275 #' get_tablas_disponibles('02')
276 #' @export
277 get_tablas_disponibles<-function(tipo_eleccion){
278   tipo_eleccion<-comprueba_tipo_eleccion(tipo_eleccion)
279   prefijos_por_tipo <- list(
280     '01' = c("05", "07", "09"),   # Refer ndum
```

```
    '02' = c("03", "04", "05", "06", "07", "08", "09", "10"),    # Congreso
    '03' = c("03", "04", "05", "06", "07", "08", "09", "10"),    # Senado
    '04' = c("03", "04", "05", "06", "07", "08", "09", "10", "11", "12"),  # Municipales
    '06' = c("03", "04", "05", "06", "07", "08", "09", "10"),    # Cabildos
    '07' = c("03", "04", "05", "06", "07", "08", "09", "10")     # Europeas
  )

  return(prefijos_por_tipo[[tipo_eleccion]])
}#' Descripci n de la tabla basada en el prefijo num rico
#'
#' Esta funci n devuelve la descripci n asociada a un prefijo de fichero espec fico,
#' como los ficheros de control, identificaci n del proceso electoral, etc.
#'
#' @param tabla Un car cter que indica el prefijo de la tabla.
#'        Debe ser un n mero de dos d gitos como '01', '02', etc.
#' @return Una cadena con la descripci n del fichero asociado al prefijo dado.
#' @examples
#' descripcion_tabla('09')
#' @export
descripcion_tabla <- function(tabla) {
  # Asegurarse de que el n mero est  en formato de dos d gitos
  numero_formateado <- sprintf("%02d", as.integer(tabla))

  # Diccionario que mapea n meros de fichero a descripciones
  descripciones <- list(
    '01' = 'Fichero de CONTROL de los ficheros que componen el proceso electoral.',
    '02' = 'Fichero de IDENTIFICACION del proceso electoral.',
    '03' = 'Fichero de CANDIDATURAS.',
    '04' = 'Fichero de RELACION DE CANDIDATOS.',
    '05' = 'Fichero de DATOS COMUNES DE MUNICIPIOS.',
    '06' = 'Fichero de DATOS DE CANDIDATURAS DE MUNICIPIOS.',
    '07' = 'Fichero de DATOS COMUNES DE AMBITO SUPERIOR AL MUNICIPIO.',
    '08' = 'Fichero de DATOS DE CANDIDATURAS DE AMBITO SUPERIOR AL MUNICIPIO.',
    '09' = 'Fichero de DATOS COMUNES DE MESAS y del C.E.R.A.',
    '10' = 'Fichero de DATOS DE CANDIDATURAS DE MESAS y del C.E.R.A.',
    '11' = 'Fichero de DATOS COMUNES DE MUNICIPIOS menores de 250 habitantes. (Solo en Elecciones Municipales)',
    '12' = 'Fichero de DATOS DE CANDIDATURAS DE MUNICIPIOS menores de 250 hab. (Solo en Elecciones Municipales)'
  )

  # Devolver la descripci n correspondiente al n mero de fichero formateado
  if (numero_formateado %in% names(descripciones)) {
    return(descripciones[[numero_formateado]])
  } else {
    return("N mero de fichero no reconocido.")
  }
}
#' Leer Varias Tablas de Datos Electorales
#'
#' Esta funci n descarga y procesa m ltiples archivos de datos electorales basados en los par metros especificados.
#' Cada archivo representa un conjunto de datos asociado a un tipo de elecci n, a o y mes espec ficos,
```

```
#' y es procesado de acuerdo con su  mbito  particular.
#'
#' @param tipo_eleccion Un car cter que indica el tipo de elecci n de entre: referendum, congreso, senado, municipales, cabildos o europeas.
#' @param a o Un n mero que indica el a o de la elecci n.
#' @param mes Un n mero que indica el mes de la elecci n.
#' @param ambito Un car cter que indica el  mbito  de la elecci n que puede ser mesa, municipio o superior.
#' @param directorio Un car cter que indica el directorio donde se guardar n los archivos descargados.
#'        Por defecto es "./descargas/".
#' @param tablas Un vector de caracteres que contiene los c digos de las tablas de datos a procesar.
#' @return Una lista de data.frames, cada uno conteniendo los datos procesados de una tabla espec fica.
#' @examples
#' tablas <- c("05", "07", "09")
#' datos_electorales <- leer_varias_tablas("congreso", 2021, 6, "nacional", "./descargas/", tablas)
#' @export
#' @importFrom readr read_csv
#' @importFrom dplyr bind_rows
leer_varias_tablas <- function(tipo_eleccion, a o , mes, ambito, directorio = "./descargas/", tablas) {
  # Usar lapply para aplicar la funci n leer_datos a cada elemento del vector tablas
  dfs <- lapply(tablas, function(tabla) {
    leer_tablas(tipo_eleccion, a o , mes, ambito, directorio, tabla)
  })
  return(dfs)
}
```

A.2. Mapas.R

```

#' Crear un mapa de las provincias de Espa a coloreado por una variable espec fica
#'
#' Esta funci n toma un dataframe y una variable de inter s , y crea un mapa de las provincias de Espa a
#' coloreado seg n los valores agregados de la variable especificada.
#'
#' @param df Un dataframe que contiene datos de poblaci n por provincias.
#' @param variable La variable del dataframe cuyo total se desea representar en el mapa.
#' @return Un mapa de las provincias de Espa a coloreado por la variable especificada.
#' @examples
#' \dontrun{crear_mapa_provincias(df,df$votos_en_blanco)
#' }
#' @import dplyr
#' @import ggplot2
#' @import sf
#' @import giscoR
#' @importFrom stats sym
#' @export
```

```r
crear_mapa_provincias <- function(df, variable) {
  variable <- deparse(substitute(variable))
  variable <- sub(".*\\$", "", variable)
  ine_codes <- data.frame(
    INE_CODE = as.numeric(c(2, 3, 4, 1, 33, 5, 6, 7, 8, 48, 9, 10, 11, 39, 12, 13, 14, 15,
                  16, 20, 17, 18, 19, 21, 22, 23, 24, 25, 27, 28, 29, 30, 31, 32, 34, 35,
                  36, 26, 37, 38, 40, 41, 42, 43, 44, 45, 46, 47, 49, 50, 51, 52)),
    PROVINCE = c("Albacete", "Alicante/Alacant", "Almer a", "Araba/ lava ", "Asturias", " vila ", "Badajoz", "Balears, Illes",
                  "Barcelona", "Bizkaia", "Burgos", "C ceres", "C diz", "Cantabria", "Castell n/Castell ", "Ciudad Real",
                  "C rdoba", "A Coru a", "Cuenca", "Gipuzkoa", "Girona", "Granada", "Guadalajara", "Huelva", "Huesca",
                  "Ja n", "Le n", "Lleida", "Lugo", "Madrid", "M laga", "Murcia", "Navarra", "Ourense", "Palencia",
                  "Las Palmas", "Pontevedra", "La Rioja", "Salamanca", "Santa Cruz de Tenerife", "Segovia", "Sevilla",
                  "Soria", "Tarragona", "Teruel", "Toledo", "Valencia/Val ncia", "Valladolid", "Zamora", "Zaragoza",
                  "Ceuta", "Melilla")
  )

  spain_provinces <- gisco_get_nuts(country = "ES", nuts_level = 3, resolution = "03", year = "2021")


  spain_provinces_selected <- spain_provinces %>%
    select(NUTS_ID, NAME_LATN) %>%
    rename(PROVINCE = NAME_LATN)

  df_poblacion_sum <- df %>%
    group_by(codigo_ine_provincia) %>%
    summarise(poblacion_total = sum(as.numeric(!!sym(variable)), na.rm = TRUE)
    )

  combined_df <- left_join(spain_provinces_selected, ine_codes, by = "PROVINCE") %>%
    left_join(df_poblacion_sum, by = c("INE_CODE" = "codigo_ine_provincia"))

  canarias <- combined_df %>%
    filter(PROVINCE %in% c("El Hierro", "Fuerteventura", "La Gomera", "La Palma", "Gran Canaria", "Lanzarote", "Tenerife"))

  canarias_poblacion <- data.frame(
    PROVINCE = c("El Hierro", "Fuerteventura", "Gran Canaria", "La Gomera", "La Palma", "Lanzarote", "Tenerife"),
    codigo_ine_provincia = c(38,35,35,38, 38, 35, 38)
  )
  canarias_poblacion_sum <- canarias_poblacion %>%
    left_join(df_poblacion_sum, by = "codigo_ine_provincia") %>%
    group_by(codigo_ine_provincia, PROVINCE) %>%
    summarise(poblacion_total = sum(poblacion_total, na.rm = TRUE))

  canarias <- canarias %>%
    left_join(canarias_poblacion_sum, by = "PROVINCE") %>%
    mutate(poblacion_total = coalesce(poblacion_total.y, poblacion_total.x))
```

```
    %>%
64    select(-poblacion_total.x, -poblacion_total.y)
65
66  canarias_shifted <- canarias %>%
67    mutate(geometry = st_geometry(geometry) + c(18, 7))
68
69  st_crs(canarias_shifted) <- st_crs(combined_df)
70
71  baleares <- combined_df %>%
72    filter(PROVINCE %in% c("Eivissa y Formentera", "Mallorca", "Menorca"))
73
74  baleares_poblacion <- data.frame(
75    PROVINCE = c("Eivissa y Formentera", "Mallorca", "Menorca"),
76    codigo_ine_provincia = c(07, 07, 07)
77  )
78  baleares_poblacion_sum <- baleares_poblacion %>%
79    left_join(df_poblacion_sum, by = "codigo_ine_provincia") %>%
80    group_by(codigo_ine_provincia, PROVINCE) %>%
81    summarise(poblacion_total = sum(poblacion_total, na.rm = TRUE))
82
83  baleares <- baleares %>%
84    left_join(baleares_poblacion_sum, by = "PROVINCE") %>%
85    mutate(poblacion_total = coalesce(poblacion_total.y, poblacion_total.x))
      %>%
86    select(-poblacion_total.x, -poblacion_total.y)
87
88  # Combinar los datos
89  spain_provinces_shifted <- combined_df %>%
90    filter(!PROVINCE %in% c("El Hierro", "Fuerteventura", "La Gomera", "La
      Palma", "Gran Canaria", "Lanzarote", "Tenerife", "Eivissa y Formentera", "
      Mallorca", "Menorca")) %>%
91    bind_rows(canarias_shifted) %>%
92    bind_rows(baleares)
93
94  # Crear el mapa de las provincias de Espa a coloreado por la variable total
95  ggplot(data = spain_provinces_shifted) +
96    geom_sf(aes(fill = poblacion_total)) +
97    scale_fill_viridis_c(
98      option = "D",
99      labels = label_number(big.mark = ",", accuracy = 1)  # Formatea los
      n meros con comas y una cifra decimal
100   ) +
101   theme_minimal() +
102   ggtitle(paste("Mapa de las Provincias de Espa a Coloreado por", variable)
      )
103 }
104 crear_mapa_CCAA <- function(df, variable) {
105   library(dplyr)
106   library(ggplot2)
107   library(sf)
108   library(giscoR)
109   library(rlang)
110
111   variable <- deparse(substitute(variable))
112   variable <- sub(".*\\$", "", variable)
113   # Mapa de provincias a comunidades aut nomas
114   provincia_CCAA <- data.frame(
115     INE_CODE = as.numeric(c(2, 3, 4, 1, 33, 5, 6, 7, 8, 48, 9, 10, 11, 39, 12,
       13, 14, 15,
```

```
                          16, 20, 17, 18, 19, 21, 22, 23, 24, 25, 27, 28,
  29, 30, 31, 32, 34, 35,
                          36, 26, 37, 38, 40, 41, 42, 43, 44, 45, 46, 47,
  49, 50, 51, 52)),
  PROVINCE = c("Albacete", "Alicante/Alacant", "Almer a", "Araba/ lava ", "
  Asturias", " vila ", "Badajoz", "Balears, Illes",
               "Barcelona", "Bizkaia", "Burgos", "C ceres", "C diz", "
  Cantabria", "Castell n/Castell ", "Ciudad Real",
               "C rdoba", "A Coru a", "Cuenca", "Gipuzkoa", "Girona", "
  Granada", "Guadalajara", "Huelva", "Huesca",
               "Ja n", "Le n", "Lleida", "Lugo", "Madrid", "M laga", "
  Murcia", "Navarra", "Ourense", "Palencia",
               "Las Palmas", "Pontevedra", "La Rioja", "Salamanca", "Santa
  Cruz de Tenerife", "Segovia", "Sevilla",
               "Soria", "Tarragona", "Teruel", "Toledo", "Valencia/Val ncia
  ", "Valladolid", "Zamora", "Zaragoza",
               "Ceuta", "Melilla"),
  CCAA = c("Castilla-La Mancha", "Comunitat Valenciana", "Andaluc a", "
  Pa s Vasco", "Principado de Asturias", "Castilla y Le n", "Extremadura",
   "Illes Balears",
           "Catalu a", "Pa s Vasco", "Castilla y Le n", "Extremadura", "
  Andaluc a", "Cantabria", "Comunitat Valenciana", "Castilla-La Mancha",
           "Andaluc a", "Galicia", "Castilla-La Mancha", "Pa s Vasco", "
  Catalu a", "Andaluc a", "Castilla-La Mancha", "Andaluc a", "Arag n",
           "Andaluc a", "Castilla y Le n", "Catalu a", "Galicia", "
  Comunidad de Madrid", "Andaluc a", "Regi n de Murcia", "Comunidad Foral
  de Navarra", "Galicia", "Castilla y Le n",
           "Canarias", "Galicia", "La Rioja", "Castilla y Le n", "Canarias"
  , "Castilla y Le n", "Andaluc a", "Castilla y Le n", "Catalu a",
           "Arag n", "Castilla-La Mancha", "Comunitat Valenciana", "
  Castilla y Le n", "Castilla y Le n", "Arag n",
           "Ciudad de Ceuta", "Ciudad de Melilla")
)
df <- df %>%
  left_join(provincia_CCAA, by = c("codigo_ine_provincia" = "INE_CODE"))

# Sumarizar los datos por CCAA

df_CCAA_sum <- df %>%
  group_by(CCAA) %>%
  summarise(poblacion_total = sum(as.numeric(.data[[variable]]), na.rm =
  TRUE))


df_CCAA_sum# Obtener datos geogr ficos de las CCAA de Espa a
spain_CCAA <- gisco_get_nuts(country = "ES", nuts_level = 2, resolution = "
  03", year = "2021")

# Unir datos geogr ficos con datos sumarizados
combined_df <- left_join(spain_CCAA, df_CCAA_sum, by = c("NAME_LATN"="CCAA")
  )
df_poblacion_sum <- df %>%
  group_by(codigo_ine_provincia) %>%
  summarise(poblacion_total = sum(as.numeric(!!sym(variable)), na.rm = TRUE)
  )

canarias <- combined_df %>%
  filter(NAME_LATN %in% c("Canarias"))

```

```
155
156   canarias_shifted <- canarias %>%
157     mutate(geometry = st_geometry(geometry) + c(18, 7))
158
159   st_crs(canarias_shifted) <- st_crs(combined_df)
160
161   baleares <- combined_df %>%
162     filter(NAME_LATN %in% c("Illes Balears"))
163
164   baleares_poblacion <- data.frame(
165     CCAA = c("Illes Balears","Illes Balears","Illes Balears"),
166     codigo_ine_provincia = c(07, 07, 07)
167   )
168   baleares_poblacion_sum <- baleares_poblacion %>%
169     left_join(df_CCAA_sum, by = "CCAA") %>%
170     group_by(CCAA) %>%
171     summarise(poblacion_total = sum(poblacion_total, na.rm = TRUE))
172
173
174   # Combinar los datos
175   spain_provinces_shifted <- combined_df %>%
176     filter(!NAME_LATN %in% c("Illes Balears","Canarias")) %>%
177     bind_rows(canarias_shifted) %>% bind_rows(baleares)
178
179   # Crear el mapa de las provincias de Espa a coloreado por la variable total
180   ggplot(data = spain_provinces_shifted) +
181     geom_sf(aes(fill = poblacion_total)) +
182     scale_fill_viridis_c(
183       option = "D",
184       labels = label_number(big.mark = ",", accuracy = 1)  # Formatea los
      n meros con comas y una cifra decimal
185     ) +
186     theme_minimal() +
187     ggtitle(paste("Mapa de las Provincias de Espa a Coloreado por", variable)
      )
188
189 }
```

A.3. Tablas.R

```
1     source("./R/LecturaDatos.R")
2 #' Calcula y muestra tablas de participaci n basadas en el tipo de elecci n
      y provincia
3 #'
4 #' Esta funci n carga datos electorales para un tipo y fecha especificados,
5 #' opcionalmente filtrando por provincia. Luego calcula res menes de
      participaci n
6 #' y muestra los resultados en forma tabular. Si se especifica una provincia,
7 #' filtra los datos para esa provincia usando los c digos INE.
8 #'
9 #' @param tipo_eleccion Tipo de elecci n como texto.
10 #' @param a o  A o de la elecci n.
11 #' @param mes Mes de la elecci n.
12 #' @param provincia Nombre de la provincia (opcional).
13 #' @return Un dataframe invisible con los resultados de participaci n y una
14 #'   impresi n en consola de los resultados detallados
```

```
15 #' @import dplyr
16 #' @export
17 tabla_participacion <- function(tipo_eleccion, a o , mes, provincia=NULL) {
18   ambito='municipio'
19   if(!is.null(provincia)){
20     provincia<-tolower(provincia)
21   }
22
23   if(!is.null(provincia)) {
24     ine_codes <- data.frame(
25       INE_CODE = as.numeric(c(2, 3, 4, 1, 33, 5, 6, 7, 8, 48, 9, 10, 11, 39,
     12, 13, 14, 15,
26                               16, 20, 17, 18, 19, 21, 22, 23, 24, 25, 27, 28,
     29, 30, 31, 32, 34, 35,
27                               36, 26, 37, 38, 40, 41, 42, 43, 44, 45, 46, 47,
     49, 50, 51, 52)),
28       PROVINCE = tolower(c("Albacete", "Alicante", "Almer a", "Araba/ lava ",
      "Asturias", " vila ", "Badajoz", "Balears, Illes",
29                            "Barcelona", "Bizkaia", "Burgos", "C ceres", "C diz", "
     Cantabria", "Castell n", "Ciudad Real",
30                            "C rdoba", "A Coru a", "Cuenca", "Gipuzkoa", "Girona", "
     Granada", "Guadalajara", "Huelva", "Huesca",
31                            "Ja n", "Le n", "Lleida", "Lugo", "Madrid", "M laga", "
     Murcia", "Navarra", "Ourense", "Palencia",
32                            "Las Palmas", "Pontevedra", "La Rioja", "Salamanca", "Santa
      Cruz de Tenerife", "Segovia", "Sevilla",
33                            "Soria", "Tarragona", "Teruel", "Toledo", "Valencia", "
     Valladolid", "Zamora", "Zaragoza",
34                            "Ceuta", "Melilla"))
35     )
36     provincia_code <- ine_codes$INE_CODE[ine_codes$PROVINCE == provincia]
37     if (length(provincia_code) > 0) {
38       df <- leer_tabla(tipo_eleccion, a o , mes, ambito, directorio = "./
     descargas/", tabla = '05')
39       df <- df[df$codigo_ine_provincia == provincia_code, ]
40       df_final <- resultados_tabla_participacion(df,tipo_eleccion)
41     } else {
42       stop(paste("Provincia no v lida. Lista de provincias v lidas:", paste(
     ine_codes$PROVINCE, collapse=", ")))
43
44     }
45   }
46   else{
47
48     df_final <- resultados_tabla_participacion(df,tipo_eleccion)}
49
50   return(invisible(df_final))
51 }
52
53 #' Calcula los resultados de participaci n para un dataframe dado
54 #'
55 #' Dada una tabla de datos, esta funci n suma los votos y calcula los
     porcentajes
56 #' de participaci n para diferentes categor as como avances de votaci n,
     votos en blanco,
57 #' y votos nulos. Si el tipo de elecci n es un refer ndum, tambi n procesa
     votos afirmativos
58 #' y negativos.
59 #'
```

```
60 #' @param df Dataframe con los datos electorales.
61 #' @param tipo_eleccion Tipo de elecci n para determinar si se incluyen
       resultados de refer ndum.
62 #' @return Un dataframe con las categor as de votaci n y sus totales y
       porcentajes.
63 #' @export
64
65 resultados_tabla_participacion<- function(df,tipo_eleccion){
66   primer_avance<-sum(df$votantes_primer_avance)
67   segundo_avance<-sum(df$votantes_segundo_avance)
68   blanco<-sum(df$votos_en_blanco)
69   nulo<-sum(df$votos_nulos)
70   if(tolower(tipo_eleccion) == 'referendum' || tolower(tipo_eleccion) == '
       refer ndum'){
71     total<-blanco+nulo+sum(df$votos_negativos_referendum)+sum(df$votos_
       afirmativos_referendum)
72   }
73   else{
74     total<-blanco+nulo+sum(df$votos_candidaturas)
75   }
76
77   resultados_df <- data.frame(
78     Categoria = c("Primer avance", "Segundo avance", "Votos en blanco", "Votos
        nulos"),
79     Total = c(sum(df$votantes_primer_avance, na.rm = TRUE), sum(df$votantes_
       segundo_avance, na.rm = TRUE),
80               sum(df$votos_en_blanco, na.rm = TRUE), sum(df$votos_nulos, na.rm
        = TRUE)),
81     Porcentaje = c(sum(df$votantes_primer_avance, na.rm = TRUE) / total * 100,
82                    sum(df$votantes_segundo_avance, na.rm = TRUE) / total *
       100,
83                    sum(df$votos_en_blanco, na.rm = TRUE) / total * 100,
84                    sum(df$votos_nulos, na.rm = TRUE) / total * 100)
85   )
86
87   # Funci n auxiliar para a adir filas al dataframe
88
89   if(tolower(tipo_eleccion) == 'referendum' || tolower(tipo_eleccion) == '
       refer ndum'){
90     afirmativos<-sum(df$votos_afirmativos_referendum)
91     negativos<-sum(df$votos_negativos_referendum)
92     votos_referendum <- afirmativos + negativos
93     referendum_df <- data.frame(
94       Categoria = c("Votos refer ndum", "Votos afirmativos", "Votos negativos
       "),
95       Total = c(votos_referendum, afirmativos, negativos),
96       Porcentaje = c(votos_referendum / total * 100, afirmativos / votos_
       referendum * 100, negativos / votos_referendum * 100)
97     )
98     cat(
99       sprintf("%-21s | %15s | %12s", "Datos Refer ndum", "Totales", "
       Porcentaje"), "\n",
100      "-------------------------------------------------------------", "\n",
101      sprintf("%-20s | %15d | %12.2f%%", "Primer avance", primer_avance,
       primer_avance / total * 100), "\n",
102      sprintf("%-20s | %15d | %12.2f%%", "Segundo avance", segundo_avance,
       segundo_avance / total * 100), "\n",
103      "-------------------------------------------------------------", "\n",
104      sprintf("%-20s | %15d | %12.2f%%", "Votos en blanco", blanco, blanco /
```

```
    total * 100), "\n",
105   sprintf("%-20s | %15d | %12.2f%%", "Votos nulos", nulo, nulo / total *
    100), "\n",
106   sprintf("%-21s | %15d | %12.2f%%", "Votos refer ndum", votos_referendum
    , votos_referendum / total * 100), "\n",
107   "_____________________________________________________________", "\n",
108   sprintf("%-20s | %15d | %12.2f%%", "Votos afirmativos", afirmativos,
    afirmativos / votos_referendum * 100), "\n",
109   sprintf("%-20s | %15d | %12.2f%%", "Votos negativos", negativos,
    negativos / votos_referendum * 100), "\n",
110   "_____________________________________________________________", "\n",
111   sep=""
112 )
113 }
114 else{
115   votos_candidaturas<-sum(df$votos_candidaturas)
116   referendum_df <- data.frame(
117     Categoria = c("Votos candidaturas"),
118     Total = c(votos_candidaturas),
119     Porcentaje = c(votos_candidaturas / total * 100)
120   )
121   print(sprintf("%-20s | %15s | %12s", "Datos ", "Totales", "Porcentaje"))
122   print("_____________________________________________________________")
123   print(sprintf("%-20s | %15d | %12.2f%%", "Primer avance", primer_avance,
    primer_avance / total * 100))
124   print(sprintf("%-20s | %15d | %12.2f%%", "Segundo avance", segundo_avance,
     segundo_avance / total * 100))
125   print("_____________________________________________________________")
126   print(sprintf("%-20s | %15d | %12.2f%%", "Votos en blanco", blanco, blanco
     / total * 100))
127   print(sprintf("%-20s | %15d | %12.2f%%", "Votos nulos", nulo, nulo / total
     * 100))
128   print(sprintf("%-20s | %15d | %12.2f%%", "Votos candidaturas", votos_
    candidaturas, votos_candidaturas / total * 100))
129
130 }
131 resultados_df <- rbind(resultados_df, referendum_df)
132 return(resultados_df)
133 }
```

Bibliografía

[1] Shiny: Web application framework for r. https://shiny.posit.co/. Último acceso: 2024-06-10.

[2] CRAN. *Roxygen2: In-Line Documentation for R*, 2022. Último acceso: 2024-06-15.

[3] Ministerio del Interior. Elecciones celebradas - ministerio del interior. https://infoelectoral.interior.gob.es/es/elecciones-celebradas/, 2024. Último acceso: 2024-06-05.

[4] Ministerio del Interior de España. Metodología. procesos electorales en democracia en españa. https://infoelectoral.interior.gob.es/export/sites/default/pdf/metodologia/Metodologia.pdf, 2021. Último acceso: 2024-05-24.

[5] Geeks for Geeks. R - object oriented programming, 2022. Último acceso: 2024-06-15.

[6] Hadley Wickham. R packages, 2022. Último acceso: 2024-06-15.

[7] C. Ismay and A.Y. Kim. Best coding practices for r. functions. https://bookdown.org/content/d1e53ac9-28ce-472f-bc2c-f499f18264a3/func.html, 2024. Último acceso: 2024-5-24.

[8] Norman Matloff. The art of r programming. https://diytranscriptomics.com/Reading/files/The%20Art%20of%20R%20Programming.pdf, 2011. Último acceso: 2024-05-24.

[9] Ethan Nam. Understanding the levenshtein distance equation for beginners. https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0, 2018. último acceso: 2024-06-16.