Herramientas de visualización interactivas para la toma de decisiones en una startup tecnológica.

Juan Ramos Hernández, 48409815 J

Grado en Administración y dirección de Empresas

Universidad Politécnica de Valencia

Tutor: Fortunato Crespo Abril

Fecha de depósito: 6/7/2020

Tabla de contenido

Índice de figuras

1. Introducción

Durante mi etapa como estudiante de Administración y Dirección de Empresas en la Universitat Politècnica de València (UPV) he sentido particular interés por las asignaturas más analíticas y tecnológicas de la carrera. Más específicamente, he intentado aprovechar las oportunidades que nos ha otorgado la universidad de aprender a utilizar herramientas que extienden las capacidades del ser humano y que, una vez habiendo superado la curva inicial de aprendizaje, dotan al usuario de un abanico extenso de recursos para poder validar y contrastar hipótesis e ideas de manera autónoma.

Debido a mi interés por la programación y el análisis de datos, he estudiado los lenguajes R y Python al mismo tiempo que realizaba la carrera. Han resultado ser herramientas increíblemente útiles que han tenido un retorno superlativo a la inversión de recursos que he asignado para su aprendizaje.

Por ello, me siento afortunado de poder aplicar mis intereses a la hora de desarrollar mi Trabajo de Fin de Grado (TFG). Creo que este proyecto transmitirá lo que he aprendido durante mi tiempo como estudiante, tanto dentro como fuera de la universidad.

El objetivo principal de mi TFG es **el desarrollo y aplicación de herramientas tecnológicas para la ayuda a la toma de decisiones.** Considero que una de las mayores atribuciones de la tecnología ha sido su capacidad para poder resolver problemas que anteriormente carecían de solución. Espero que el mensaje que transmita mi TFG sea ese, que una herramienta es tan útil como el problema que resuelve.

El problema que pretendo resolver nace en la empresa en la que trabajo desde octubre de 2019, **homyspace**. Allí, como analista de datos, me encuentro con no pocos problemas a resolver, los cuales siempre afronto con la mentalidad de desarrollar una solución **sencilla, barata y sostenible.**

Para este proyecto, me voy a centrar en tratar de otorgar una mayor inteligibilidad al ecosistema de proveedores de **homyspace**, para que el equipo sea capaz de tomar decisiones basadas en datos. **La calidad de una decisión no se debería de medir en función de los resultados que obtiene, sino en función de la información que había disponible en el momento que se hizo. Por ende, facilitar el acceso a la mayor cantidad de información posible debería de facilitar la toma de buenas decisiones.**

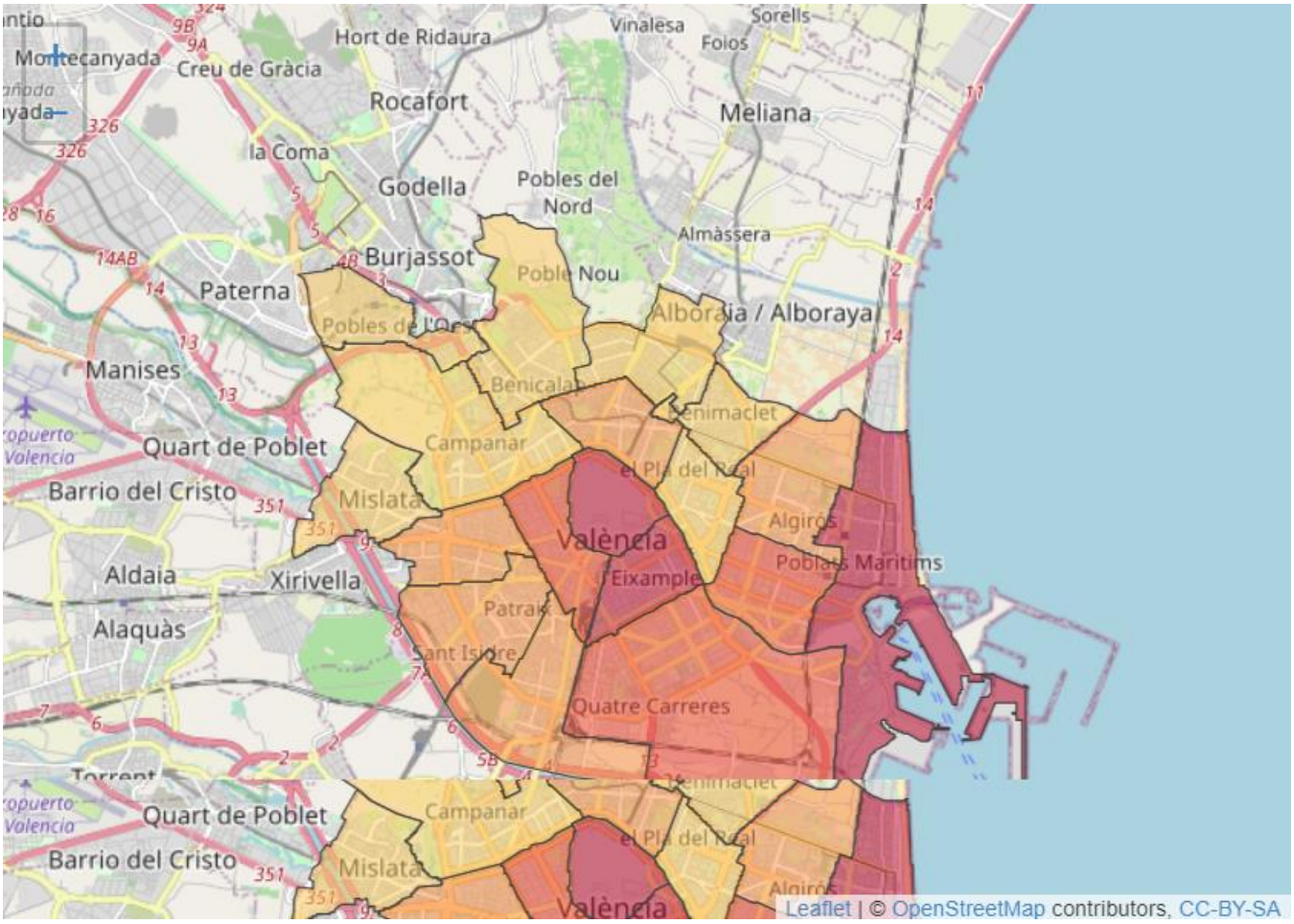

Figura 1: Mapa de coropletas de los distritos censales de Valencia. Fuente: Elaboración propia.

1.1. ¿Qué es homyspace?

Homyspace es una plataforma del tipo *marketplace* que se especializa en el alquiler temporal de media estancia para empresas.

Cuando una empresa va a trasladar a empleados temporalmente, las opciones por las que opta son, por lo general, poco económicas. Un hotel no está pensado para estancias de treinta días o más. Además, la inversión necesaria para encontrar un alojamiento que encaje con las necesidades de la empresa tiende, por lo general, a ser mayor de la esperada.

Homyspace soluciona este problema poniendo en contacto a empresas y arrendatarios, ahorrando tiempo y dinero a la empresa y ofreciendo mayor rentabilidad al arrendador en comparación con el alquiler a largo plazo. Además, aporta mayor seguridad y facilidades en comparación con el alquiler a corto plazo. **Homyspace** fue fundada en el año 2016 por Ángel Mayoral, Alberto García y Rubén Ballesteros.

Una plataforma *marketplace* facilita un espacio donde la oferta y la demanda para un servicio o producto se puedan conectar para hacer negocios de manera centralizada. Por lo tanto y siguiendo esta definición, **homyspace** es una plataforma a la cual acuden empresas que tienen la necesidad de desplazar trabajadores, así como dueños de inmuebles que quieren ofrecerlos para servicios de alquiler.

Homyspace carece de oferta (inmuebles) propia. Más bien, se encarga de seleccionar y ofrecer a los clientes potenciales inmuebles que encajen con sus requisitos y necesidades de entre una vasta lista de propiedades previamente registradas por sus dueños o gestores.

Casi cuatro años después, cuenta con **más de cuarenta empleados, decenas de miles de inmuebles registrados y miles de estancias gestionadas en cientos de localizaciones diferentes para clientes de la talla de H&M, Amazon o Bertolín**.

1.2. Definiciones

Antes de adentrarnos en las diferentes partes que componen este proyecto, nos es necesario comprender los conceptos que van a ser mencionados a lo largo del mismo.

1.2.1. Conceptos relacionados con el mundo start-up

- *Marketplace*: un *marketplace* es un término ampliamente utilizado en el mundo de las *start-up*. Se emplea para referirse a **aquellas empresas cuyo modelo de negocio consiste en crear una plataforma donde unificar la oferta y la demanda de un servicio o producto, ejerciendo en ocasiones de intermediario o regulador.**
- *Lead*: un *lead* es un **usuario que ha entregado sus datos a una empresa**, explícita o implícitamente (en este segundo caso, al aceptar la política de privacidad de la empresa), debido a su interés por el servicio o producto que crea la empresa.
- Embudo o *funnel*: un *funnel* o embudo de una empresa define el **proceso que se genera de manera interna en la misma desde que el lead deja sus datos, hasta que adquiere el servicio o producto ofrecido por parte de la empresa**. Se denomina *embudo* por la forma que presenta este proceso. Al principio del *funnel* el número de leads alcanza su máximo, y éste se va reduciendo conforme se aproxima la compra del servicio o producto.

- Marketing *inbound*: conjunto de técnicas que se encargan de crear contenido para atraer clientes potenciales, como campañas de marketing o desarrollo de un blog, con el propósito de que sea **el cliente el que acuda a la empresa**.
- Marketing *outbound*: se diferencia del marketing *inbound* en que, en este caso, **es la empresa la que analiza e identifica a los clientes potenciales, y acude a ellos**. Está asociado con técnicas de marketing más tradicionales como el marketing de llamadas en frío o telemarketing.

1.2.2. Conceptos relacionados con homyspace

- Solicitudes: en **homyspace**, consideramos solicitudes a todas las **peticiones de alojamiento por parte de los leads con intención de desplazarse.** Se encuentran en la primera etapa del embudo de la demanda de **homyspace**.
- Leads cualificados: los leads cualificados son aquellos **leads que encajan con la tipología de cliente de homyspace.** Empresas que desplazan a sus trabajadores más de 15 días, responsabilizándose del pago de la estancia.
- Oportunidades: las oportunidades son aquellas solicitudes provenientes de **leads cualificados**. Las oportunidades son gestionadas por los agentes personales de **homyspace** Forman parte de la segunda etapa del embudo de **homyspace** para la demanda.
- Propuestas: las propuestas son aquellos **inmuebles que el agente personal de homyspace considera que encajan mejor con las características y necesidades de la oportunidad en concreto**, y que son enviados a la empresa como opciones de alojamiento. Del mismo modo que las oportunidades, éstas también forman parte de la segunda etapa del embudo de **homyspace** para la demanda.
- Calendario electrónico o *iCal*: el calendario electrónico es un programa que permite comprobar de manera automática si el inmueble al que está asociado está o no disponible.
- Curación de proveedores: la curación de proveedores consiste en la **mejora de la base de datos existente de proveedores**. Abarca todo lo relacionado con aumentar la calidad de la información relacionada a un inmueble. Por un lado, asegurar que toda la información de los inmuebles es correcta y que las fotos están actualizadas, por el otro, garantizar que el inmueble tiene un calendario electrónico asociado para poder saber instantáneamente si el inmueble está disponible para ser alquilado o no.

1.3. Flujo de trabajo de homyspace

A continuación, presentamos una explicación simplificada del flujo de trabajo de homyspace, específicamente desde que un *lead* (cliente o proveedor) contacta con nosotros hasta que adquiere el servicio que ofrece.

El flujo estándar comienza con la entrada del *lead*, **como consecuencia de las acciones de *marketing*.**

Podemos diferenciar dos tipos diferentes de *leads*:

- *Leads* de Proveedores: Este *lead* es el dueño de un inmueble o el gestor de una serie de inmuebles, que está interesado en añadir sus propiedades a la oferta disponible de **homyspace**.

- *Leads* de Empresas: Este *lead* es un individuo que gestiona el desplazamiento de un grupo de trabaja- dores o de sí mismo, siempre bajo el amparo de su empresa.

1.3.1. Flujo de Proveedores

El flujo de proveedores comienza cuando un potencial proveedor contacta con nuestra empresa. El *lead* de proveedores va acompañado de información sobre el inmueble, o inmuebles, que ofrece a **homyspace**, así como detalles para facilitar el contacto en caso de que su inmueble encaje con una oportunidad.

El equipo de proveedores se encarga de analizar la información de cada inmueble, así como de valorar la calidad de las fotos aportadas. Si todo alcanza los requisitos de calidad de **homyspace**, el inmueble es añadido a la base de datos. Si no es así, se contacta con el proveedor para que mejore las fotos o amplíe la información de los inmuebles ofrecidos.

1.3.2. Flujo de Empresas

Del mismo modo, consideramos que **el flujo de empresas empieza cuando el empleado de una empresa nos proporciona información para contactar con él.**

Como hemos indicado en *1.2 Definiciones*, esta información se transforma en una solicitud. El equipo de ventas contacta con la empresa que ha realizado la solicitud y, tras obtener la información relevante, la solicitud o bien avanza en el embudo -convirtiéndose en una oportunidad- o es descartada.

Las oportunidades resultantes de la etapa anterior son asignadas a los gestores personales de Homyspace. Estos analizan las características de la oportunidad y buscan en la base de datos aquellos inmuebles que encajan mejor con las necesidades del cliente. Cuando encuentran un inmueble que cumple los requisitos crean una *propuesta*.

Normalmente se envía al cliente de 2 a 3 propuestas. La empresa o cliente tiene 24 horas para aceptar las propuestas o rechazarlas. Si se rechazan, el gestor buscará otros inmuebles en la base de datos que encajen mejor con las necesidades del cliente. Si se aceptan, el embudo pasará a la tercera y última etapa, gestionada por el equipo de reservas.

Figura 2: Flujo de trabajo de **homyspace**.

2. Objetivos

El problema que queremos resolver está centrado **en la adquisición y curación de proveedores.** *¿Cómo podemos saber en qué zonas necesitamos adquirir inmuebles?* Una forma de hacerlo sería comparar la oferta en esa zona con la demanda existente. Pero *¿qué ocurre cuando sí hay oferta disponible en esa zona, pero los inmuebles no están siendo reservados?* Quizá el tipo de inmueble en esa zona no encaja con la demanda, o tal vez el problema yace en que los precios no han sido lo suficientemente negociados, o incluso en que hemos estado captando demasiados inmuebles en una zona donde hay oferta de sobra y estamos desperdiciando recursos. Del mismo modo, *¿Cómo negociamos los precios del inmueble con el proveedor adquirido? ¿En qué nos basamos para decirle si tiene un precio "alto" o "bajo"?*

En referencia a la curación, **necesitamos saber de manera ágil qué inmuebles tienen la información asociada actualizada, así como un calendario electrónico asignado.**

Sin embargo, resulta difícil saber en qué zonas tenemos inmuebles cuya información ha sido recientemente actualizada, y dónde tenemos un mayor número relativo de inmuebles pendientes de ser curados.

En un marketplace como Homyspace, el flujo de datos semanal alcanza un tamaño considerable. Cientos de empresas acuden semana a semana para que Homyspace les ofrezca alojamientos que puedan utilizar para sus estancias a medio plazo. Durante el mismo rango de tiempo, son también cientos los inmuebles que se añaden a la base de datos.

Los inmuebles son adquiridos de dos formas: a través de técnicas *inbound* o de técnicas *outbound*.

Antes de la ejecución de estas estrategias de captación, es necesario saber en qué zonas la oferta de Homyspace no está satisfaciendo la demanda.

Conforme el volumen de clientes y empresas aumenta, el coste de oportunidad de decidir adquirir en una zona determinada también lo hace. Por lo tanto, **tomar la decisión correcta en función de la información disponible en ese momento se vuelve más importante que nunca.**

2.1. Objetivo General

Como objetivo general, este proyecto pretende **desarrollar una herramienta de visualización de datos. Un *dashboard* que implementará gráficos interactivos para ayudar a tomar decisiones inteligentes**. Más concretamente, mediante el uso del software libre *python* se transformarán los datos brutos, actualmente disponibles en formato geoespacial, proporcionando información relacionada con la localización de los inmuebles que forman parte de la base de datos de Homyspace, y se segmentarán por distritos censales definidos por el INE. [1]

Al mismo tiempo emplearemos el software libre *R* y diferentes librerías de su ecosistema para visualizar los datos preparados, de forma que podremos analizar una miríada de métricas relacionadas con cada distrito, lo cual ayudará a comprender mejor el estado de ese distrito y si es necesario que sea un foco de captación o curación de inmuebles.

Previo al desarrollo de esta herramienta, el análisis de zonas de captación se realizaba de forma mucho más rudimentaria, a través de hojas de cálculo de *Microsoft Excel*. Este método, aunque definitivamente útil, es lento, necesita de una descarga y tratamiento de datos previo, y sin lugar a dudas, **es inapropiado cuando se está lidiando con datos que requieren de un análisis geoespacial**. Y no sólo eso, sino que la dificultad del análisis aumenta de manera exponencial en *Microsoft Excel* conforme el volumen de datos de **homyspace** aumenta. Nos es necesario desarrollar una solución a la altura del problema, y para ello *R* es ideal.

2.2. Objetivos Específicos

Con el propósito de alcanzar el objetivo general previamente mencionado, hemos definido una serie de objetivos específicos asociados a acciones concretas. Estos objetivos son:

- **Introducir** los gráficos interactivos: qué son y para qué sirven.
- **Defender** su uso con respecto a los gráficos tradicionales.
- **Exponer** cómo son aplicados en **homyspace**: qué beneficios aportan y qué inconvenientes presentan. **Transmitir** la importancia de la creación de una fuente (o fuentes) de datos, así como de su mantenimiento.
- **Presentar** el desarrollo de una aplicación de visualización de datos.
- **Mostrar** una versión del resultado final.
- **Concluir** resumiendo todo lo expuesto, añadiendo posibles vías de mejora de la aplicación.

3. Aplicación de gráficos interactivos para la toma de decisiones

3.1. ¿Qué es un gráfico interactivo?

Se ha avanzado mucho en el campo de la visualización de datos. El primer gráfico registrado en la historia data de *1628*, creado por Michael Florent van Langren con el propósito de transmitir la magnitud del problema que quería abordar, que consistía en determinar la longitud geográfica. [2] Por entonces, se carecía del conocimiento y de las herramientas que facilitan la creación de gráficos de forma ágil. Aun así, se recurría a ellos por la misma razón por la que los utilizamos a día de hoy: **facilitan la comprensión de la información**.

Con la llegada de los ordenadores, nació la posibilidad de desarrollar gráficos cada vez más complejos, tanto por su diseño y características, como por la cantidad de información que utilizaban como fuente. Gracias al mayor poder computacional, se exploró la viabilidad de que ~~estos gráficos dejaran de ser una "fo~~to" de la información de la que se alimentaban, y se convirtieran en aplicaciones en sí mismas. Así, **un gráfico interactivo es un gráfico que permite al usuario interactuar con la información que se utiliza como fuente de datos.**

[1]Indicadores para secciones censales y cartografía digitalizada, Instituto Nacional de Estadística

[2]Friendly, Michael et al.~The First (Known) Statistical Graph: Michael Florent van Langren and the "Secret" of Longitude. (2010).

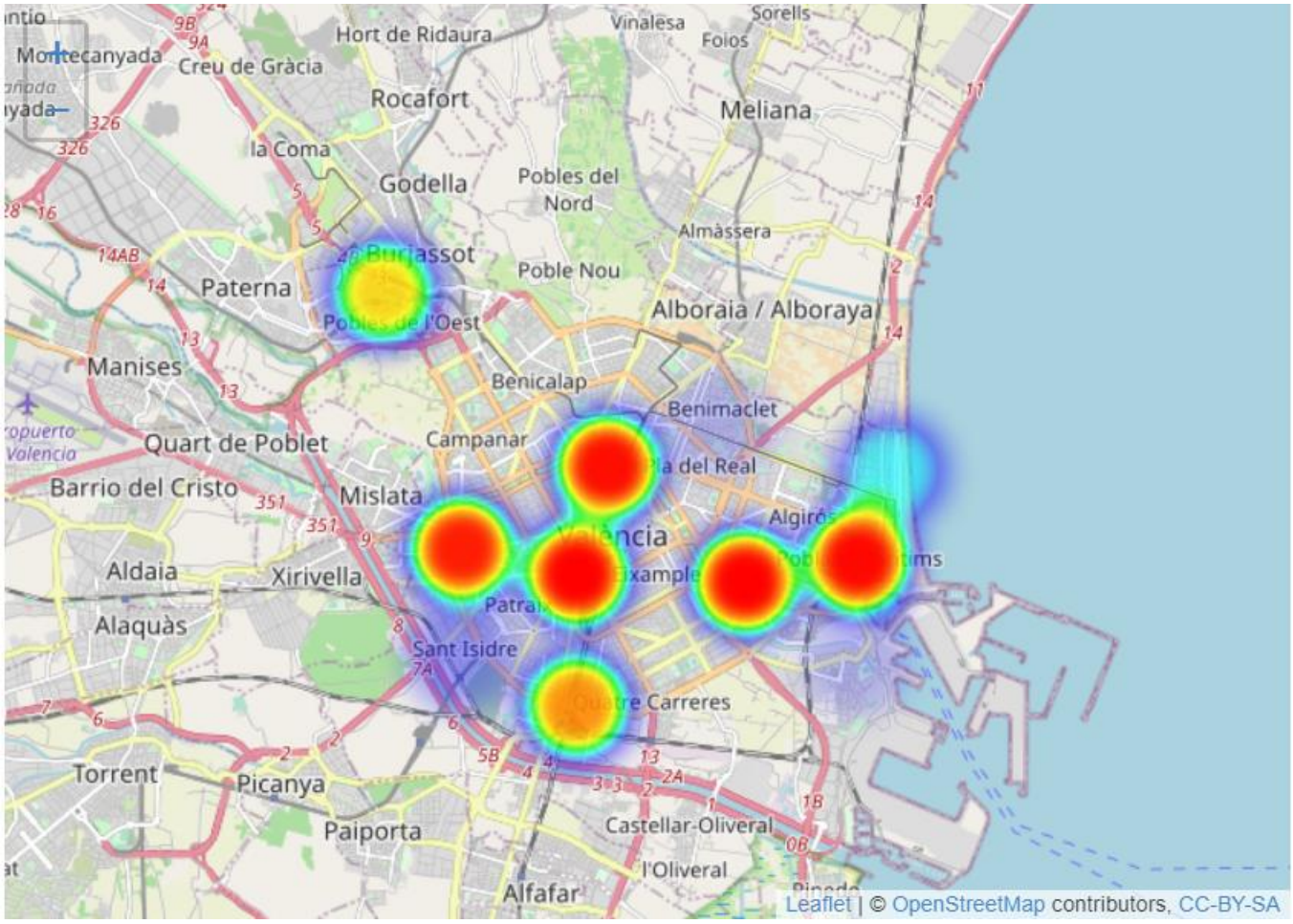

Figura 3: Mapa de calor interactivo que muestra la densidad de inmuebles en Valencia. Fuente: Elaboración propia para exponer las diferencias entre los gráficos estáticos e interactivos.

3.2. Ventajas de los gráficos interactivos frente a los estáticos

A continuación, haremos una comparación de los gráficos interactivos frente a los estáticos. Expondremos las ventajas que ofrece escoger esta tecnología en vez de opciones más tradicionales, así como sus inconvenientes.

Para ello, comenzaremos con un ejemplo sencillo, aumentando la complejidad con un segundo ejemplo. La razón de seguir este procedimiento es que las ventajas de emplear gráficos interactivos aumentan, por lo general, con la complejidad de la información expuesta.

Hemos creado una fuente de datos que indica, en función de cada país, cuál es la pieza de ropa favorita. Por ejemplo, si la popularidad de los pantalones es del 25 % en España, significa que es la prenda favorita del 25 % de la población encuestada.

A continuación, se presentan un gráfico estático tradicional y un gráfico dinámico para describir esta información.

¿Qué gráfico permite un mejor análisis de los datos?

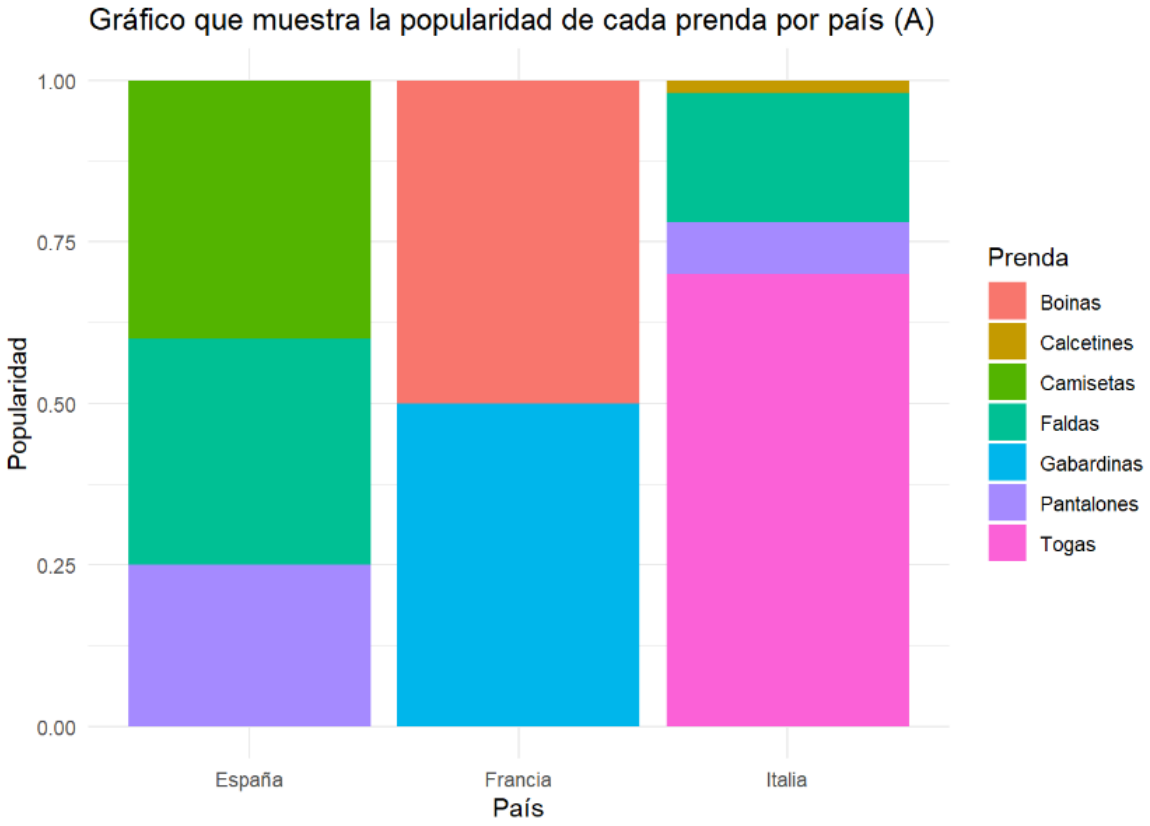

Figura 4: Gráfico estático que muestra la popularidad de cada prenda por país. Fuente: Elaboración propia para exponer las diferencias entre los gráficos estáticos e interactivos.

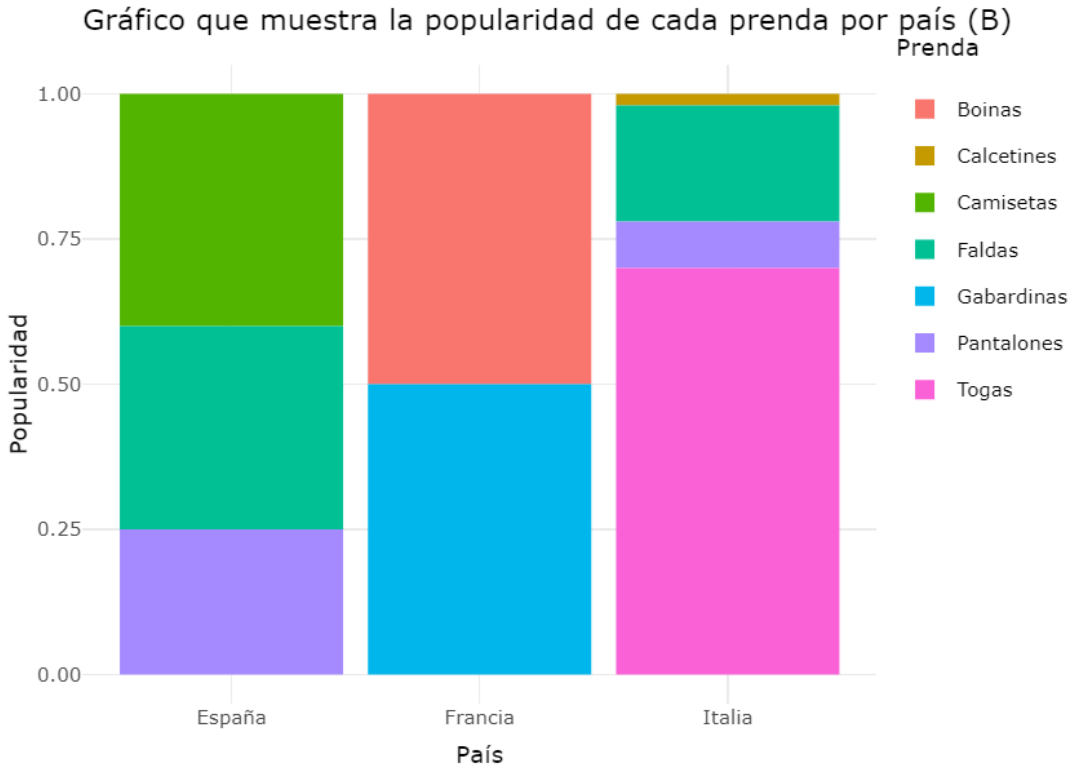

Figura 3.3: Gráfico dinámico que muestra la popularidad de cada prenda por país.

Fuente: Elaboración propia para exponer las diferencias entre los gráficos estáticos e interactivos.

Figura 5: Gráfico dinámico que muestra la popularidad de cada prenda por país. Fuente: Elaboración propia para exponer las diferencias entre los gráficos estáticos e interactivos.

Aunque los datos son limitados y, por lo tanto, la información que se puede extraer de los mismos también lo es, hay una diferencia notable entre ambos gráficos a la hora de digerir la información.

1. Podemos saber el porcentaje exacto de popularidad de cada prenda en cada país, simplemente pasando el ratón por encima del gráfico interactivo (Figura 4). Así como hacer zoom en los lugares que más nos interesan. De esta forma, aumentan la **precisión** y la **profundidad** del análisis. Esto resulta especialmente valioso cuando el volumen de los datos es alto, ya que podemos familiarizarnos con los mismos rápidamente.

2. Podemos filtrar prendas clicando sobre ellas en la leyenda. La productividad del usuario aumenta porque puede realizar el filtrado de manera instantánea, sin necesidad de editar y ejecutar el gráfico cada vez. Esto concede **autonomía** al usuario, ya que no tiene que depender de un tercero para responder las preguntas que considera convenientes. Del mismo modo, al poder decidir qué información desea visualizar en cada instante, el analista puede tratar datos multidimensionales eficientemente.
3. Concede al usuario un **rol activo** en la exploración de los datos. Aunque la información sea la misma en el gráfico estático y en el interactivo, el hecho de poder investigar y jugar con los datos ayuda a analizar con más profundidad la información disponible.

Un segundo ejemplo: hemos creado una serie de pseudo-inmuebles en Valencia que vamos a trasladar a un mapa. Haremos uso de un mapa de calor para ver en qué zonas de Valencia hay más densidad de inmuebles.

Comparemos una visualización estática del mapa frente a una que permita hacer zoom sobre el mapa.

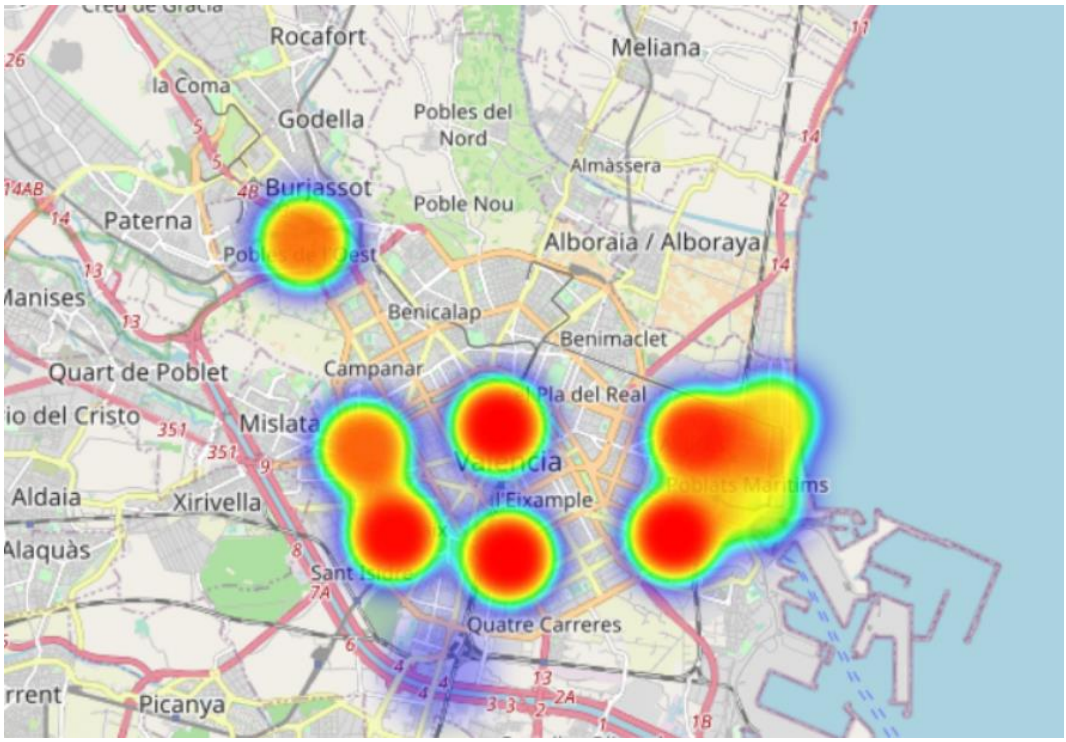

Figura 6: Mapa de calor estático que muestra la densidad de inmuebles en Valencia. Fuente: Elaboración propia para exponer las diferencias entre los gráficos estáticos e interactivos.

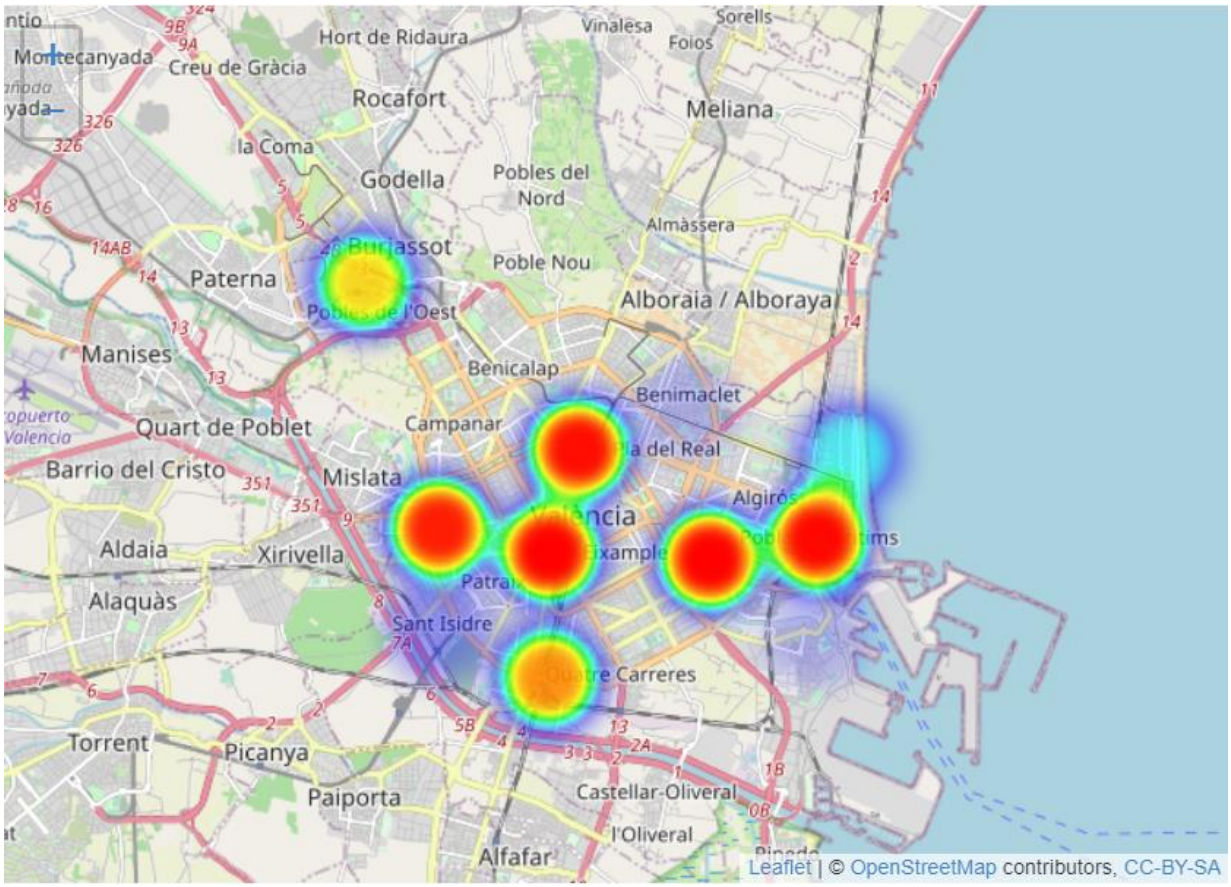

Figura 7: Mapa de calor interactivo que muestra la densidad de inmuebles en Valencia. Fuente: Elaboración propia para exponer las diferencias entre los gráficos estáticos e interactivos.

En este caso, es claramente observable que el gráfico interactivo (Figura 7) aporta más valor que el estático (Figura 6). Debido a la distancia entre los focos, si quisiéramos hacer uso del mapa estático tendríamos que obtener mapas de las diferentes zonas (Ciutat Vella, Poblat Marítims, etc.) y hacer una comparación a posteriori. Con el mapa interactivo, **podemos navegar a través de las diferentes localidades y averiguar de manera sencilla en qué zonas hay una mayor densidad de inmuebles**. Y no sólo eso, sino que podemos hacer zoom para obtener una precisión máxima de la localización de cada foco. Las cualidades mencionadas previamente en el punto 1, **precisión** y **profundidad**, destacan para el análisis de información geoespacial.

Otra ventaja de los gráficos interactivos **es que su implementación puede llegar a ser realmente sencilla**. Volviendo al ejemplo de las prendas favoritas por país, el código para crear el gráfico estático (Figura 3) es el siguiente:

```r
library(ggplot2) # Cargamos la librería ggplot2, la cual utilizaremos para crear el gráfico
filepath <- file.path("data", "fashionData.csv")
data<- read.csv2(filepath, encoding="UTF-8")
data <- data %>%
  rename(
    Pais = "X.U.FEFF.Pais"
  )

# Código para crear el gráfico de barras, donde en el eje x tenemos cada país,
# en el eje y el nivel de popularidad de cada prenda y donde en cada barra
# aparecerá diferenciadas las prendas de ropa por colores

fashionPlot <- ggplot(data=data, aes(x=Country, fill = Clothing, y = Popularity)) +
  geom_bar(stat="identity") +
  labs(title = "Gráfico que muestra prenda favorita por país (A)",
       fill = "Prenda") +
  scale_x_discrete(name = "País") +
  theme_minimal()

fashionPlot # aquí mostramos el gráfico
```

Para convertir este gráfico estático en un gráfico interactivo, todo lo que tenemos que hacer es **seguir dos pasos**:

1. Instalar la librería *plotly*, la cual nos permite convertir gráficos estáticos en interactivos.
2. Utilizar la función *plotly::ggplotly*, pasándole como *input* el gráfico estático previamente creado, *fashion- Plot*.

```r
# install.packages("plotly") # (Únicamente ejecutaríamos esta línea si no tuviéramos la librería # ya instalada en el ordenador)
plotly::ggplotly(fashionPlot)
```

Esta es una forma sencilla de convertir un gráfico estático en un gráfico interactivo, utilizando **R**.

3.2.1. Inconvenientes de los gráficos interactivos

Por desgracia, la implementación de gráficos interactivos también puede suponer inconvenientes y, dependiendo de las necesidades del analista, optar por el uso de gráficos estáticos puede ser acertado en algunos casos. Estos inconvenientes son:

- **Es necesario instalar librerías nuevas**. Por lo general, esto no supone un problema si el análisis se realiza en el ordenador del analista. Sin embargo, hay situaciones en las que la aplicación con la que se va a realizar el análisis se almacena en la nube. Esto puede ser un gran inconveniente, ya que la memoria de la localización donde se alberga la aplicación es limitada y, frecuentemente, es inviable aumentar la capacidad de almacenamiento.
- Siguiendo la lógica del punto anterior, un gráfico interactivo pesa más que uno estático.

- La implementación de gráficos interactivos puede ser sencilla. Especialmente cuando el gráfico estático original es relativamente simple y ha sido creado utilizado alguna librería de uso generalizado, como puede ser *ggplot*. Sin embargo, cuando la complejidad del gráfico aumenta o se trata de un gráfico nicho, **es muy probable que haya que desarrollar el gráfico utilizando librerías completamente nuevas**, incluso obtener tipos de datos diferentes a los utilizados en un gráfico estático. Por ende, es necesario que el analista valore el *trade-off* que esto supone, y decida si le es rentable aprender a utilizar estas librerías o es preferible decantarse por un gráfico más tradicional. De hecho, esto es lo que ocurre cuando analizamos información geoespacial: las librerías que utilizamos para cada caso son totalmente diferentes y no son combinables (como sí lo son *plotly* y *ggplot2*), lo cual exige un desarrollo íntegro tanto del gráfico estático como del interactivo.

En resumen, los principales inconvenientes de los gráficos interactivos son sus **mayores requerimientos de memoria** y la **posible inversión de tiempo para aprender a desarrollarlos e implementarlos**.

3.3. Aplicación de gráficos interactivos para la toma de decisiones: aspecto y componentes

Siguiendo una metodología *top-down*: empezamos utilizando un nivel de abstracción muy alto, donde mostraremos cómo se emplea la herramienta una vez ha sido creada. Veremos el uso actual por parte del equipo de proveedores de la misma, así como su aplicación como herramienta de inversión inmobiliaria.

Una vez sepamos **para qué se utiliza**, reduciremos esta abstracción mostrando **cómo ha sido desarrollada**, dando así respuesta a una serie de cuestiones: ¿por qué se ha optado por estos gráficos?, ¿por qué mostrar una información determinada y no otra?, etc.

Primero, definimos los gráficos interactivos y expondremos sus ventajas con respecto a los gráficos tradicionales, así como sus inconvenientes.

A continuación, abordaremos el uso del conjunto de gráficos interactivos por parte del equipo de proveedores y veremos en qué situaciones y cómo los utilizan.

Por último, expondremos la posible aplicación y comercialización de la herramienta que hemos desarrollado en usos relacionados con la inversión inmobiliaria.

A lo largo de esta sección, encontraremos diferentes figuras en las que la información correspondiente a los ejes x o y, así como a la leyenda, han sido cubiertos con un rectángulo negro. Esto lo hacemos para proteger la información de homyspace.

Como recordatorio, la razón de existencia de esta aplicación es **ayudar a tomar decisiones basadas en datos con respecto a los inmuebles en la base de datos de homyspace, y su interacción con los clientes.** La aplicación está compuesta de 6 filtros distintos y de 4 gráficos interactivos diferentes que se ajustan dinámicamente a los filtros indicados.

El primer filtro con el que nos encontramos, denominado: **"Selecciona en función de qué campo colorear el mapa de calor"**, afecta exclusivamente al mapa de calor, y es el único de todos los filtros que es directamente visible para el usuario de la aplicación. En función del campo seleccionado, estaremos dando más importancia a la oferta, a la demanda o a la intersección de ambas.

- Si elegimos “Número de Oportunidades”, el mapa pintará más claro los distritos con menos demanda, y más oscuros aquellos donde nos hacen más solicitudes de alojamiento.
- Si elegimos “Número de Inmuebles”, el mapa dará mayor o menor importancia a aquellos distritos con más o menos inmuebles respectivamente.
- Por último, si elegimos “Número de Propuestas Aceptadas”, los distritos más oscuros serán aquellos donde se han aceptado más propuestas. Los más claros aquellos donde menos.

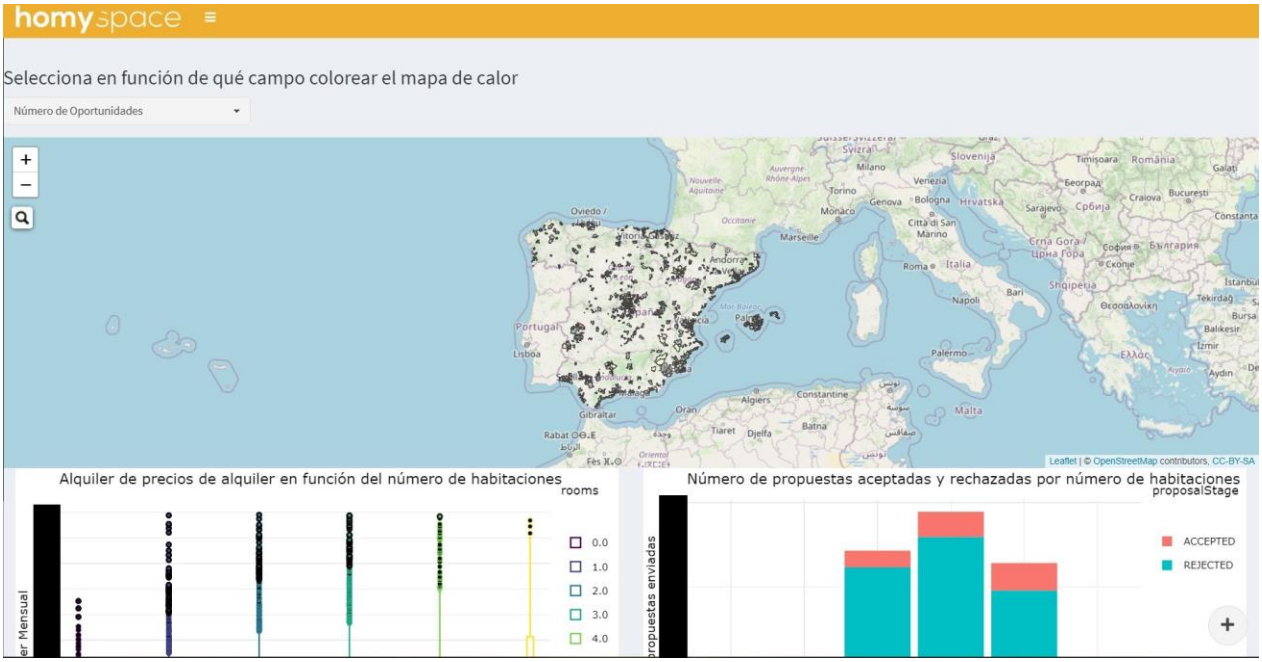

Figura 8: Pantalla principal de lo que vemos al abrir la aplicación.

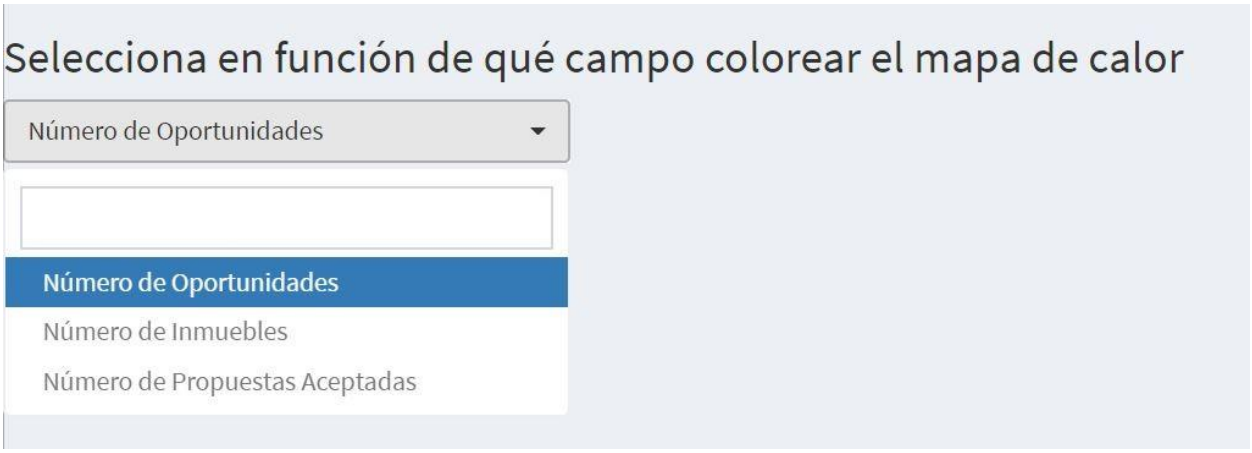

Figura 9: Pantalla principal de las opciones de coloreado del mapa de calor.

Para acceder al resto de filtros, es necesario clicar en el borde izquierdo superior de la pantalla, a la derecha del icono de **homyspace** que vemos en la figura 6. Hecho esto, nos aparecerá una lista vertical de opciones no excluyentes, por lo que podremos combinar diferentes filtros para visualizar la información deseada, como podemos ver en la figura 8. Todos los gráficos interactivos exceptuando el mapa se actualizan dinámicamente en función de las opciones escogidas en cada filtro. La razón por la que se ha excluido la actualización del mapa tiene un trasfondo técnico que se explicará en la sección *4.4 Gráficos Interactivos,* pero, en esencia, el motivo es que si quisiéramos actualizarlo, la velocidad de la aplicación se vería seriamente comprometida.

En cualquier caso, estos filtros son:

- "Filtra por ciudad": aquí podemos decidir sobre qué ciudades nos muestra información la
- aplicación. "Filtra por municipio": aquí podemos ser más concretos y especificar el municipio que queremos visualizar.
- "Filtra por distrito": una vez más, podemos aumentar la precisión de los datos y escoger qué distrito censal nos interesa más.
- "Filtra por número de camas": en este caso, podemos escoger el número de camas de los inmuebles y propuestas a analizar.
- "Filtra por número de habitaciones": por último, podemos elegir el número de habitaciones de los inmuebles y propuestas sobre los que realizar el análisis.

Como hemos comentado antes, estos filtros son **no excluyentes**, por lo que podríamos analizar el estado de los inmuebles de una habitación con dos camas en un distrito de A Coruña, por poner un ejemplo.

A continuación, pasaremos a explicar los gráficos empleados en la aplicación.

El primer gráfico que vemos al acceder a la aplicación es **el mapa por distritos.**

Como podemos ver, el zoom del mapa es a priori muy reducido como para poder diferenciar nada. Aumentamos el zoom para ser capaces de reconocer algunos distritos destacados.

Ahora ya somos capaces de separar visualmente ciertos distritos, en zonas como por ejemplo Toledo y Cáceres. Como podemos apreciar en la figura 10, hay distritos rojos, amarillos y, si nos fijamos en Madrid, también naranjas. Siguiendo la lógica que hemos comentado antes con respecto al filtro **"Selecciona en función de qué campo colorear el mapa de calor"** en la figura 7, los distritos rojos tienen un mayor número de oportunidades. Después, les siguen los distritos naranjas y, por último, los amarillos.

Cabe señalar que no se ha utilizado un número fijo de colores, sino que se ha usado un gradiente de color que varía desde el rojo al amarillo. En los extremos, el color rojo representa a las zonas con mayor número de oportunidades y el amarillo a aquellas con un menor número.

Por otro lado, aquellos distritos que vemos de color gris tienen este color porque no tienen oportunidades. Pero si es así: *¿Por qué aparecen distritos sin oportunidades?* Si aparecen ciertos distritos sin oportunidades:
¿Por qué no aparecen todos los distritos de España? El motivo es que esos distritos no tienen oportunidades, pero sí inmuebles. Por lo tanto, los distritos en gris son **zonas donde tenemos oferta que a día de hoy no ha encajado con las necesidades de nuestros clientes.**

Habiendo visualizado el mapa, podemos seguir bajando por la aplicación. A continuación, nos

encontramos con dos gráficos interactivos. Estos se modifican de dos maneras diferentes:

1. **Cuando se clica en un distrito en el mapa, este se actualiza automáticamente para mostrar la información correspondiente al distrito elegido.**
2. **Cuando se aplican filtros, la información que muestran estos gráficos se filtrará simultáneamente.** Por ejemplo: si filtramos por la ciudad de Madrid, la información que utilizarán será exclusivamente la de esta ciudad.

A la izquierda de la figura 11, se muestra un **diagrama de caja múltiple** (Ampliado en la figura 12). A la derecha un **gráfico de barras** (Ampliado en la figura 13). Estos dos gráficos se complementan: el diagrama de caja resume la distribución de precios de los inmuebles de cada distrito en función del número de habitaciones, y el gráfico de barras muestra el número de propuestas aceptadas y rechazadas para inmuebles con diferente número de habitaciones.

Figura 10: Pantalla principal de los filtros disponibles para decidir qué información mostrar en la aplicación.

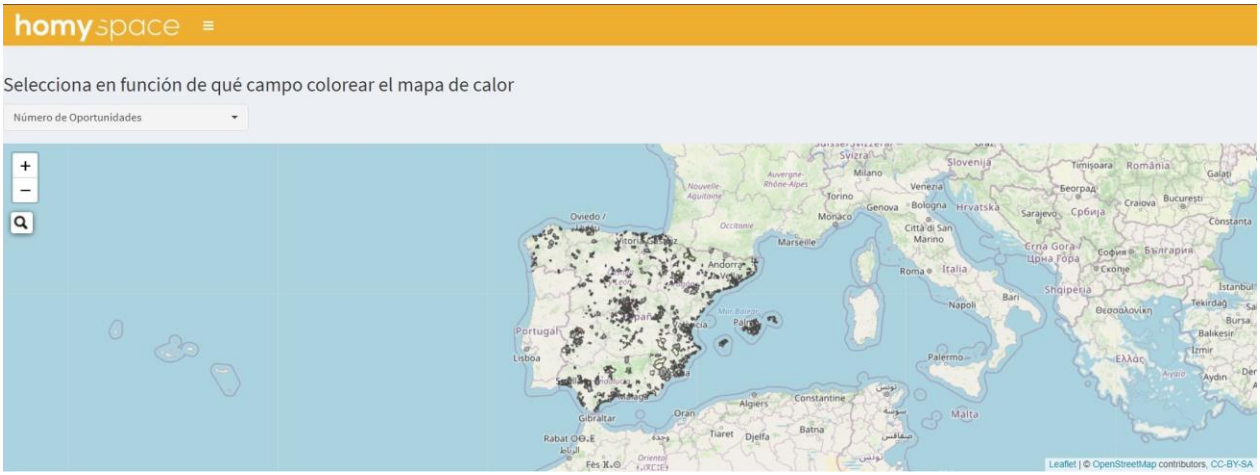

Figura 11: Pantalla principal del mapa de calor al abrir la aplicación.

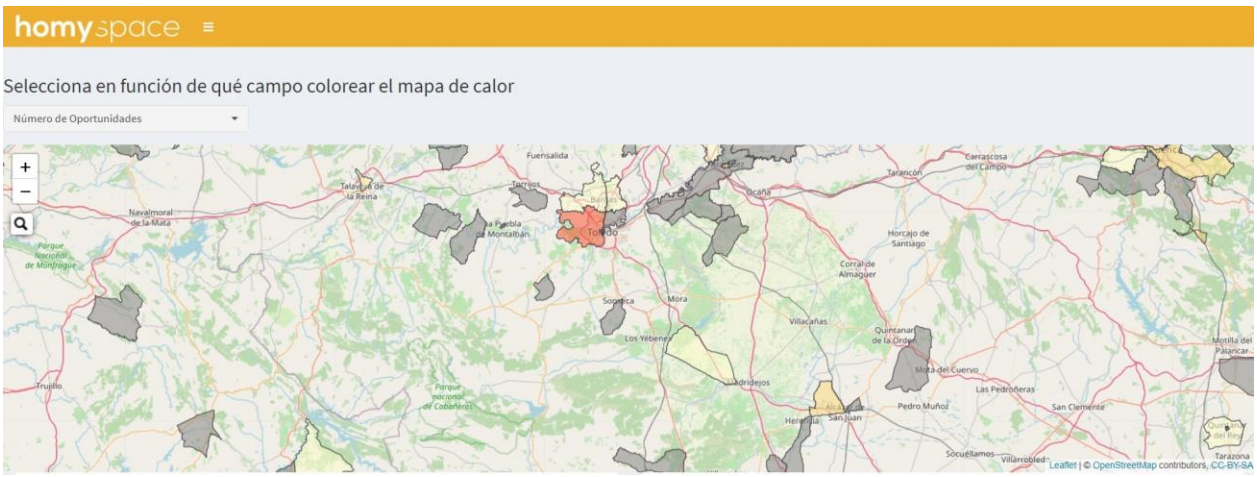

Figura 12: Pantalla principal del mapa de calor zoomeado en Toledo.

Figura 13: Pantalla principal mostrando los gráficos que aparecen inmediatamente después del mapa de calor.

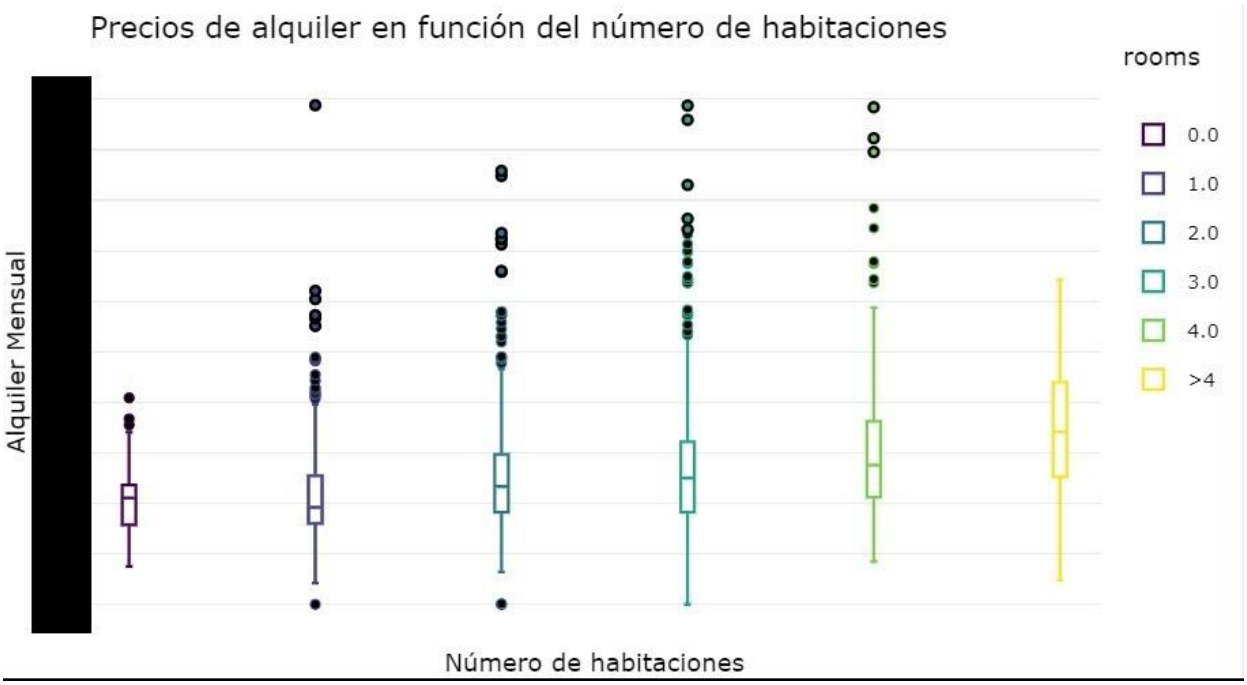

Figura 14: Pantalla principal del diagrama de caja que indica el alquiler de precios en función del número de habitaciones.

A continuación, en la figura 14 se muestran otras dos visualizaciones interactivas que siguen la misma lógica que los gráficos explicados anteriormente.

A la izquierda tenemos un gráfico de barras que nos indica el número de inmuebles que hay en el distrito en función del número de habitaciones (ampliado en la figura 15). A la derecha, un diagrama de barras que nos indica el porcentaje con respecto al total de inmuebles en el distrito que tienen (o no) calendario electrónico, o *iCal* (ampliado en la figura 16).

Finalmente hemos añadido una tabla resumen en la que podemos visualizar **toda la información relacionada con cada distrito**. Podemos visualizar la estructura de esta tabla en la figura 17.

Esta tabla contiene filtros propios, muchos más rápidos que los previamente mencionados ya que sólo afectan a la información de la tabla. A diferencia de los filtros previamente expuestos en la figura 8, cuya modificación afectaba a todos los componentes de la aplicación (más concretamente los gráficos expuestos en las figuras 11 y 14), los filtros presentes en la tabla resumen de la figura 17 **únicamente** afectarán a la misma tabla resumen, acelerando mucho la velocidad de carga en comparación con el resto de gráficos, que no se verán afectados con cada alteración de los filtros de la tabla.

Esto nos permite abordar el análisis de una manera diferente: en vez de tratar de dar visibilidad al estado de cada distrito, con esta tabla se pretende poder **obtener aquellos distritos que cumplan con las condiciones que nos interesen a nosotros**. Por ejemplo, podemos filtrar únicamente los distritos cuyo número de propuestas sea mayor de 100 y tengan por lo menos 5 inmuebles reservados.

Esta estrategia es útil porque podemos identificar qué distritos están funcionando peor para hacer un análisis posterior de por qué no están funcionando. Del mismo modo, podemos averiguar cuáles sí están funcionando y analizar las posibles razones de su éxito, con el propósito de aplicar una estrategia similar en otros distritos.

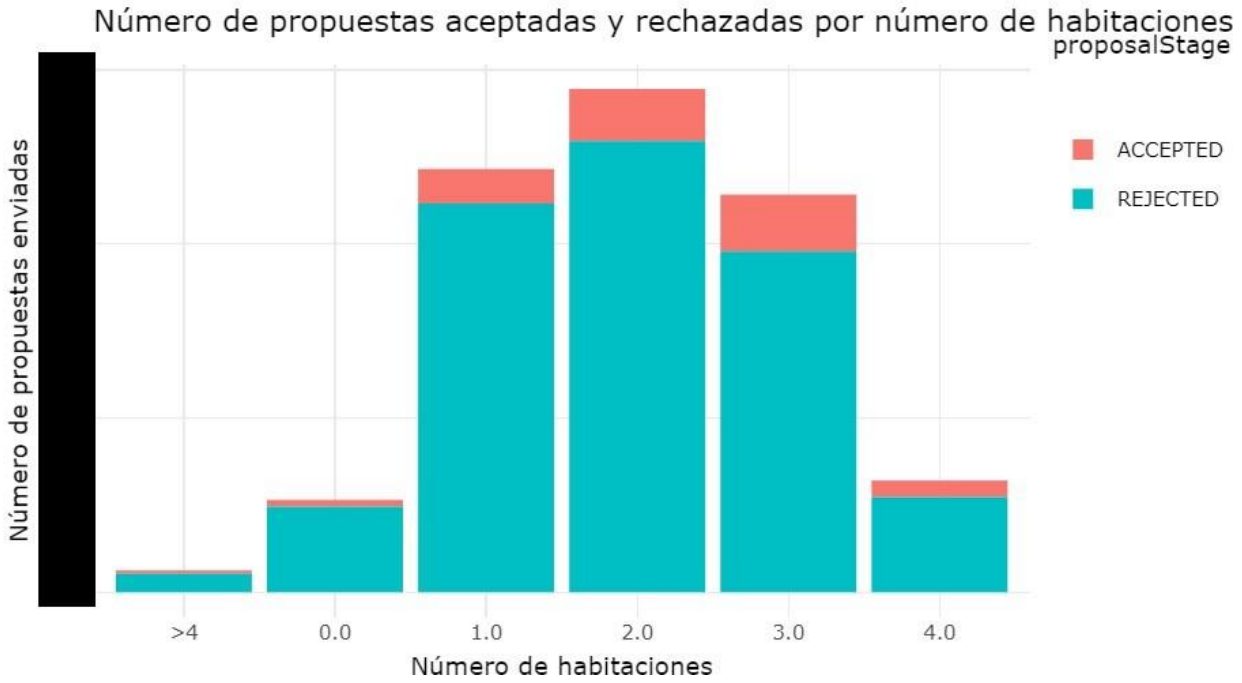

Figura 15: Pantalla principal del gráfico de barras que muestra el número de propuestas aceptadas y rechazadas por número de habitaciones.

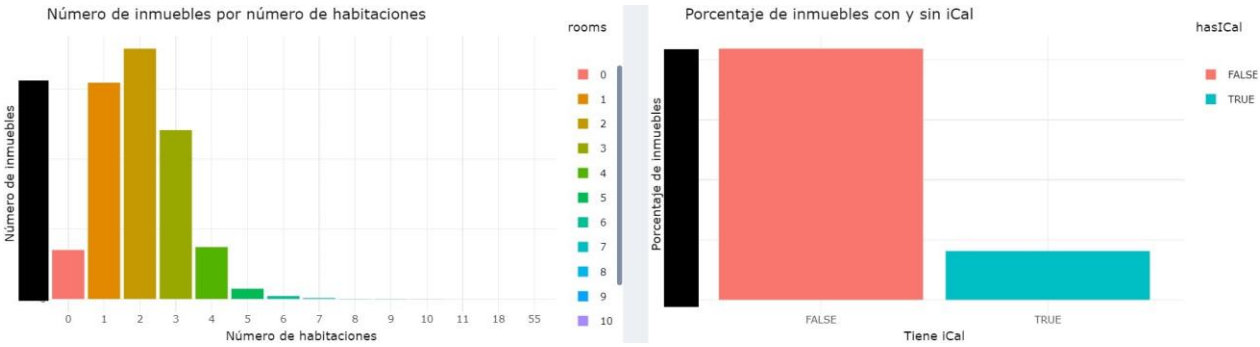

Figura 16: Pantalla principal mostrando la segunda fila de gráficos que aparece tras el mapa de calor.

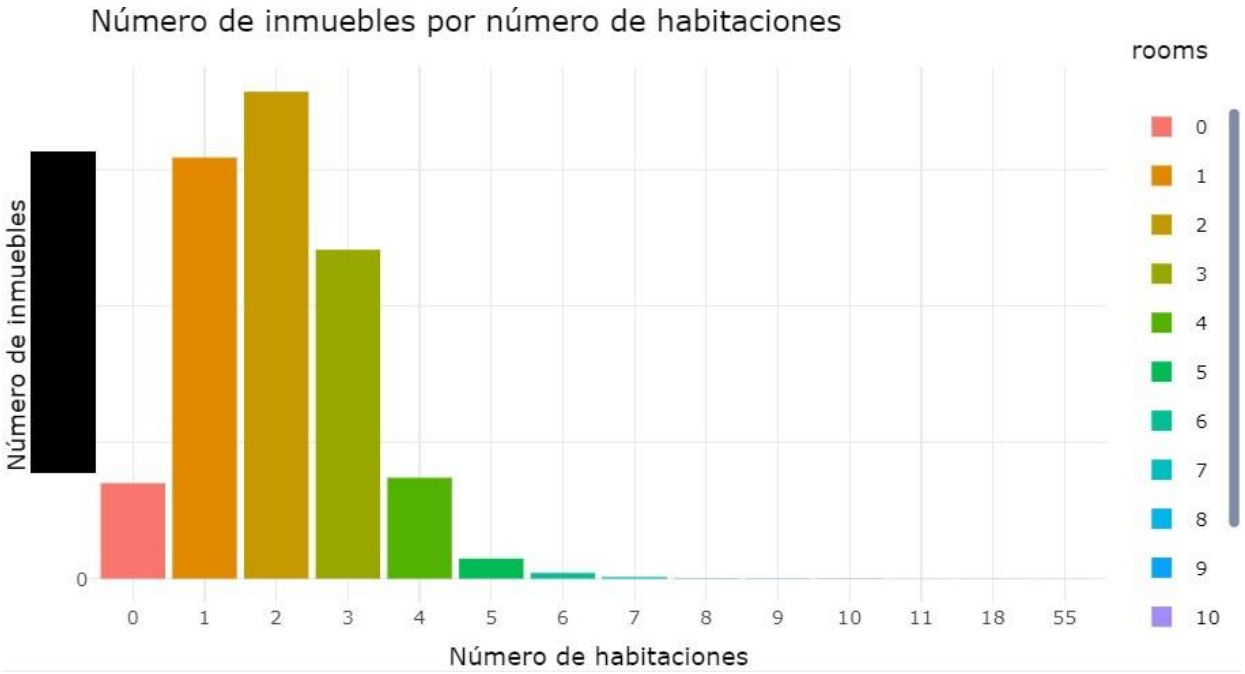

Figura 17: Pantalla principal del gráfico de barras que muestra el número de inmuebles por número de habitaciones.

Figura 18: Pantalla principal del gráfico de barras que indica el porcentaje de inmuebles con y sin iCal.

Figura 19: Pantalla principal de la tabla que resume la información por distrito de todas las propuestas.

3.4. Aplicación en el equipo de proveedores

Una vez hemos mostrado la aplicación, nos disponemos a desarrollar **cómo está siendo utilizada por el equipo de proveedores.** En este punto se detalla la metodología que hemos seguido para difundir el uso de la aplicación por parte del equipo de proveedores. Después, visibilizaremos cómo el equipo la utiliza para tareas de captación y negociación, así como de curación.

3.4.1. Métodos para la difusión del uso de la aplicación

Tras la creación de la aplicación, es esencial obtener *feedback* del uso de la misma. Para ello, hemos creado un vídeo en el que hacemos un *tour* explicando el funcionamiento de la herramienta, para qué casos de uso puede ser de utilidad, cuáles son sus límites y cómo prevemos que será mejorada. Este vídeo se aloja en una plataforma que utilizamos en **homyspace** para transmitir información entre diferentes miembros del equipo. **El vídeo va acompañado de un resumen de las funcionalidades de la aplicación, en caso de que el usuario potencial no quiera invertir tiempo en la visualización del vídeo.**

Del mismo modo, una vez la aplicación ha sido creada y publicada en un servicio gestionado, realizamos una *masterclass* con el equipo de proveedores con el propósito de explicar y extender su uso. Asimismo, **se realizan reuniones periódicas en las que cada uno de los miembros del equipo expone qué les está gustando, dónde se podría mejorar y a qué no le encuentran utilidad.**

La difusión de contenido que ayude a emplear la herramienta es tan importante como la creación de la misma, ya que si la aplicación no es utilizada de nada habrá servido desarrollarla. Por ende, **si queremos maximizar el ROI (Return On Investment, retorno a la inversión en español) de un proyecto tecnológico, es necesario invertir en la adopción de aquello que es creado.**

Como recapitulación, la metodología ha consistido en:

1. Formación pasiva, a través de la creación de contenido a la cual puede acudir el usuario (el vídeo tutorial, el resumen).
2. Formación activa, mediante la realización de una *masterclass*, así como el acuerdo de hacer reuniones periódicas en las que recogemos *feedback* sobre la aplicación.

3.4.2. Captación

A continuación, trataremos los diferentes usos de la aplicación por parte del equipo de proveedores.

Un caso de uso para el que se recurre a la aplicación es **para ayudar a decidir en qué zonas se van a focalizar los esfuerzos de captación. Es decir, seleccionar en qué focos se deben obtener más inmuebles.**

Para ello, es necesario responder las siguientes preguntas:

- **¿Dónde tenemos más producto?**
- **¿Dónde tenemos mayor demanda?**
- **¿En qué lugares está encajando bien la oferta disponible con la demanda?**

Como hemos indicado antes, estas cuestiones son analizadas a través de la navegación por los distintos filtros disponibles al abrir el mapa, ampliados aquí y en la figura 7.

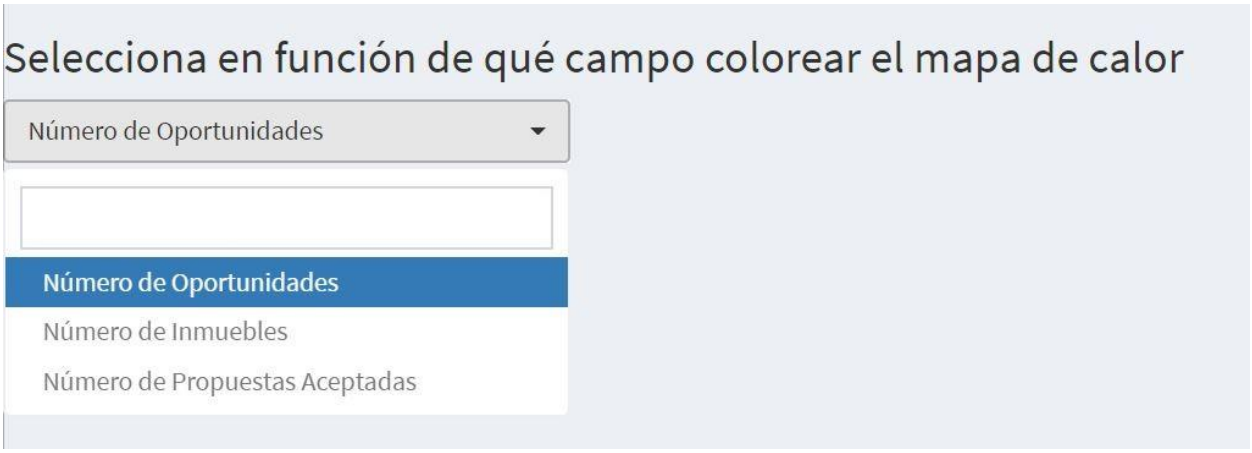

Figura 20: Pantalla principal de las opciones de coloreado del mapa de calor.

En cualquier caso, las preguntas previamente mencionadas son una simplificación del análisis real que realiza el equipo, ya que facilita la asociación de cada pregunta con los filtros creados.

Tras haber conversado con un miembro del equipo, exponemos un supuesto proceso en el que se analiza en qué zonas debería de captar **homyspace**:

Primero, el gestor entra en *Kibana*, la aplicación para visualización de datos de *Elasticsearch* [3]. Esta aplicación permite el análisis de datos en tiempo real, aunque sacrifica la posibilidad de realizar gráficos y visualizaciones más complejas a favor de la actualización instantánea y de la facilidad para crear gráficos. Dentro de *Kibana* accede al *dashboard* mostrado en la figura 19.

Elasticsearch es un motor de búsqueda que facilita el acceso a información almacenada en diferentes índices [4]. Un índice es una base de datos que, normalmente, almacena todo el contenido relacionado con un concepto. Por ejemplo, en **homyspace** podemos tener diferentes índices: uno para inmuebles, otro para solicitudes, otro para propuestas, etc. Y *Elasticsearch* nos permite acceder a todos a través de *Kibana*.

Kibana es la interfaz de usuario que te permite visualizar los datos a los que *Elasticsearch* tiene

acceso. Por ende, *Elasticsearch* te permite interactuar con toda la información que se va registrando en las bases de datos de **homyspace** a tiempo real a través de *Kibana*.

En este dashboard podemos ver en qué localizaciones estamos recibiendo más demanda, así como dónde están convirtiendo más. Del mismo modo, el gestor puede aplicar filtros de localización, por el número de personas que se van a desplazar, etc. Así, se puede hacer una idea inmediata de ~~en qué zonas debería de centrar el análi~~sis.

[3]Kibana

[4]Elasticsearch

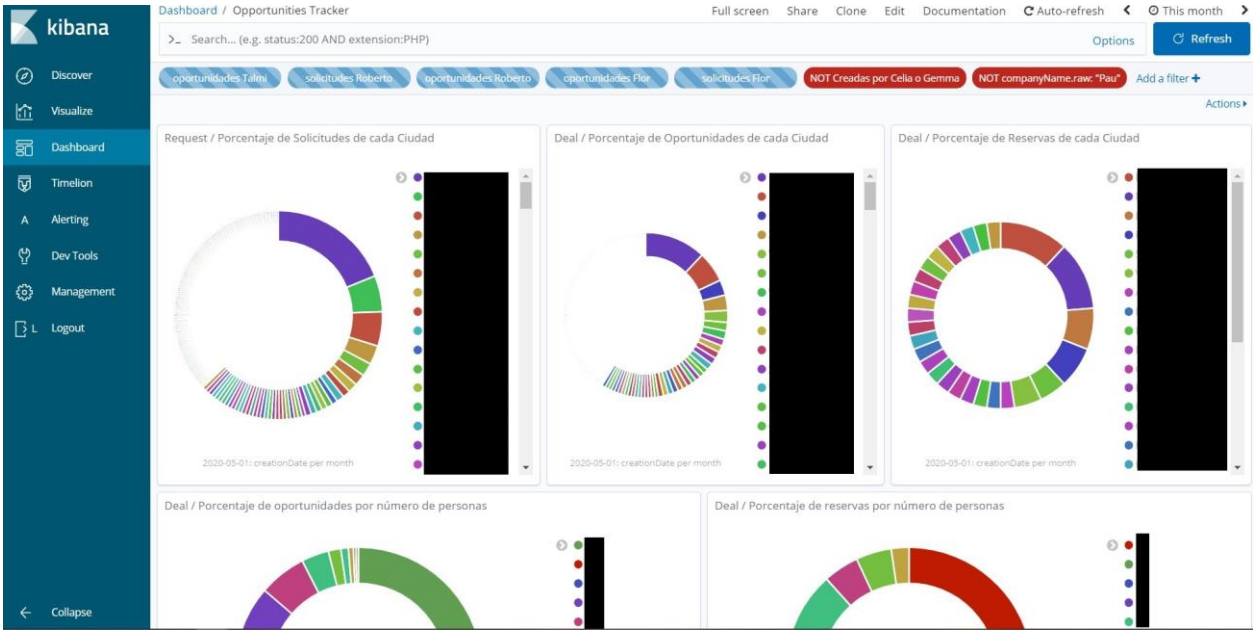

Figura 21: Pantalla principal del dashboard de Kibana utilizado para hacer un análisis superficial del estado de la demanda.

Los rectángulos negros están ocultando la leyenda que nos indica qué ciudad representa cada color. Esto lo hacemos debido a la confidencialidad de la información de **homyspace**.

Arriba de los gráficos observamos rectángulos azules y rojos, estos rectángulos son filtros que permiten al usuario seleccionar rápidamente el tipo de información que quiere visualizar. Cuando el rectángulo es de color azul es porque está aplicado de forma estándar. Si es azul con líneas blancas, como es el caso en este ejemplo, es porque está desactivado. Por último, si está en color rojo es porque está aplicado de forma inversa, es decir, queremos que nos muestre todo aquello diferente a lo que el filtro indica.

¿Por qué no realizar todo el análisis en Kibana?

A pesar de las ventajas previamente mencionadas de Kibana: su capacidad para estar actualizado constantemente y su facilidad para crear gráficos por lo general interactivos-, **no ofrece la posibilidad de adaptación de los gráficos a la información a analizar** (es decir, no podemos desarrollar un mapa como el creado en Shiny), su catálogo de gráficos es muy limitado, y la inserción de información que no se encuentre ya en la base de datos de la que se alimenta (un índice de *Elasticsearch*) es muy compleja, a menudo imposible.

Después, habiendo decidido en qué ciudades quiere profundizar, acude a la aplicación de Shiny. Aquí, podrá ir más allá del análisis por ciudad y podrá diferenciar el estado de cada distrito de esa ciudad. Por ejemplo, digamos que escoge Valencia. En ese caso, hará zoom en la ciudad de Valencia y podrá visualizar los diferentes distritos en función del filtro escogido para colorearlos.

En el ejemplo, la información mostrada ha sido falsificada para no revelar información de **homyspace**. No obstante, los campos que aparecen en el *pop-up* tras clicar en cada distrito son los mismos campos que se utilizan en la aplicación real.

Con esta información, **el gestor puede analizar el estado de salud de cada distrito y plantearse preguntas que ahora *sí* puede responder**, tales como:

- ¿Qué distrito de la ciudad tiene mayor ratio de conversión?
- Si el distrito con más demanda tiene baja conversión, ¿es por falta de producto? ¿por precios? Si es por precios, ¿cuál de los distritos más cercanos tiene mejores precios? ¿Tenemos suficiente oferta en ese distrito? Podemos captar ahí y ofrecer el producto a clientes potenciales que tengan coche o con presupuesto más ajustado
- Este distrito tiene muchos inmuebles, pero muy pocos de ellos han sido propuestos. ¿Por qué? ¿No encajan con lo que necesitan los clientes? Del mismo modo, este distrito tiene pocos inmuebles, pero se han reservado todos. ¿Necesitamos captar más? ¿Qué características tienen esos inmuebles? ¿Podemos replicar esa tipología de inmueble en otras zonas?

Campo	*Definición*
Distrito	El código de identificación del distrito en cuestión.
Oportunidades	Número de oportunidades en el distrito.
Reservas	Número de reservas en el distrito.
Ratio de Conversión	Reservas entre oportunidades.
Inmuebles	Número de inmuebles en el distrito.
Inmuebles propuestos	Número de inmuebles propuestos en el distrito.
Inmuebles reservados	Número de inmuebles reservados en el distrito.
Propuestas Hechas	Número de propuestas hechas de inmuebles del distrito.
Propuestas Aceptadas	Número de propuestas aceptadas de inmuebles del distrito.
Benchmark	Propuestas aceptadas entre propuestas hechas.
Precio Medio (1 y 2 habitaciones)	Precio medio de los inmuebles de 1 y 2 habitaciones del distrito.
Estancia Media	Estancia media de las oportunidades en el distrito.

Figura 22: Definiciones de los campos mostrados en el pop-up del mapa de calor.

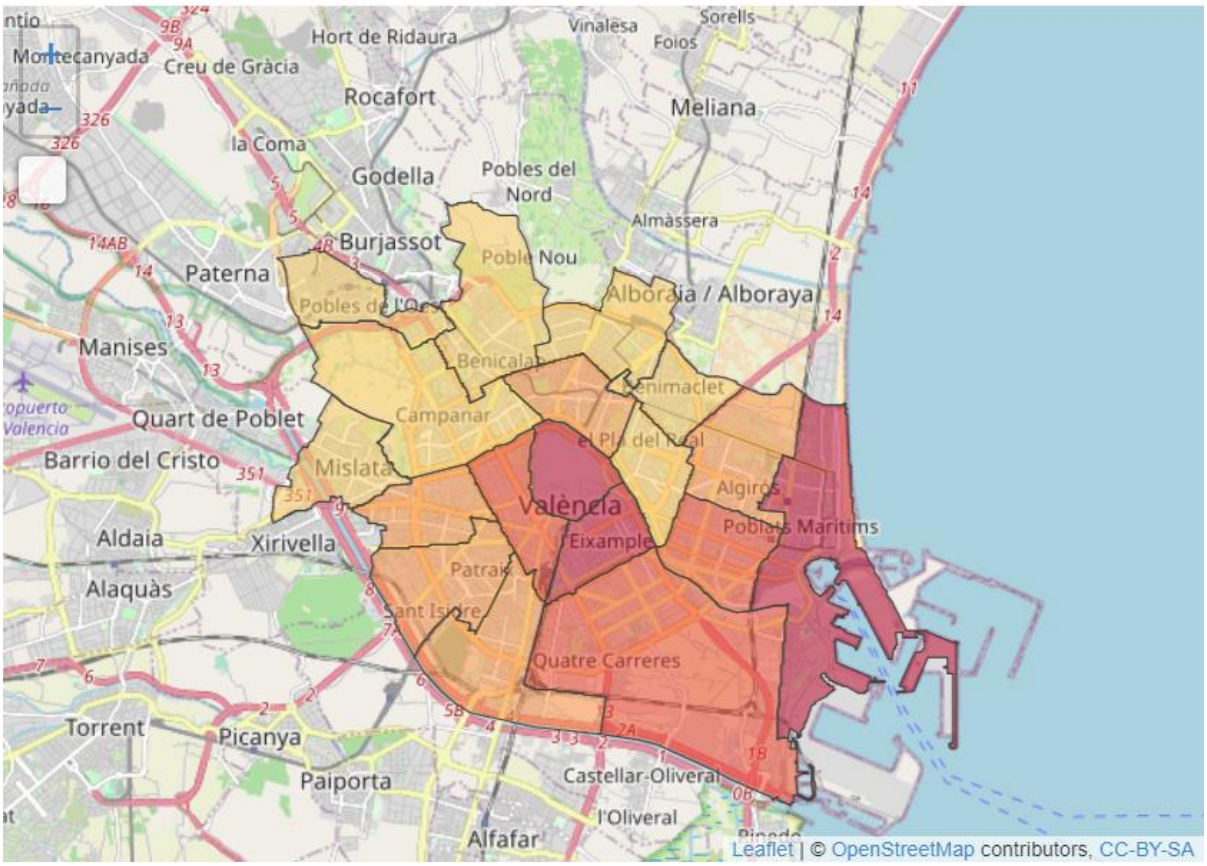

Figura 23: Mapa de coropletas de los distritos censales de Valencia. Fuente: Elaboración propia para exponer el uso del mapa por parte del equipo de proveedores.

En conclusión, aunque la aplicación no es la solución única y definitiva para la toma de decisiones con respecto a la captación de nuevos inmuebles, sí permite hacerse preguntas más concretas, así como profundizar más en cada zona y por ende, concretar acciones más precisas.

Todo esto sumado a la facilidad de uso de la aplicación, conseguida a través de, por un lado, un proceso iterativo que ha permitido perfeccionar la aplicación hasta alcanzar el punto en el que se encuentra ahora y, por el otro, a la flexibilidad y capacidad de *Shiny*, que permiten adaptar la aplicación rápidamente a los problemas que surgen de forma dinámica en el departamento, evitando la obsolescencia de la herramienta.

3.4.3. Negociación

Por otro lado, para la negociación con potenciales nuevos proveedores, **el equipo se informa de la localización del inmueble, inserta la localización en el mapa y analiza si nos interesa añadir ese inmueble, y a qué precio.**

Digamos, para mantener el ejemplo consistente con el anterior, que la calle de este inmueble es "Carrer d'Àngel Guimerà". Entonces, el gestor buscará esta ubicación en el mapa mostrado en la figura 21 (debajo de los iconos para zoomear hay un cuadrado que, al clicarlo, muestra un buscador). Si el buscador encuentra la ubicación, indicará en qué distrito se encuentra. En ese caso, el gestor clicará en el distrito y todos los gráficos bajo el mapa se actualizarán para mostrar la información correspondiente a ese distrito.

Tras haber hecho esto, todos los gráficos interactivos expuestos en las figuras 11 y 14 se adaptarán para mostrar la información correspondiente al distrito.

De este modo el gestor podrá:

1. Analizar si nos interesa ese inmueble: **¿tenemos suficientes inmuebles con esas características? ¿Tiene esa tipología de inmueble un historial de encajar con las necesidades del cliente?**
2. Si es así, el gestor podrá negociar con el proveedor basándose en datos. Observando el diagrama de caja, el gestor será capaz de reconocer de forma inmediata si el precio que el proveedor sugiere es demasiado caro y dar argumentos sólidos para convencer de una reducción de precios. Argumento estándar: *"El precio que me indicas está por encima del precio del X % de inmuebles con las mismas características que tenemos en esta zona".*

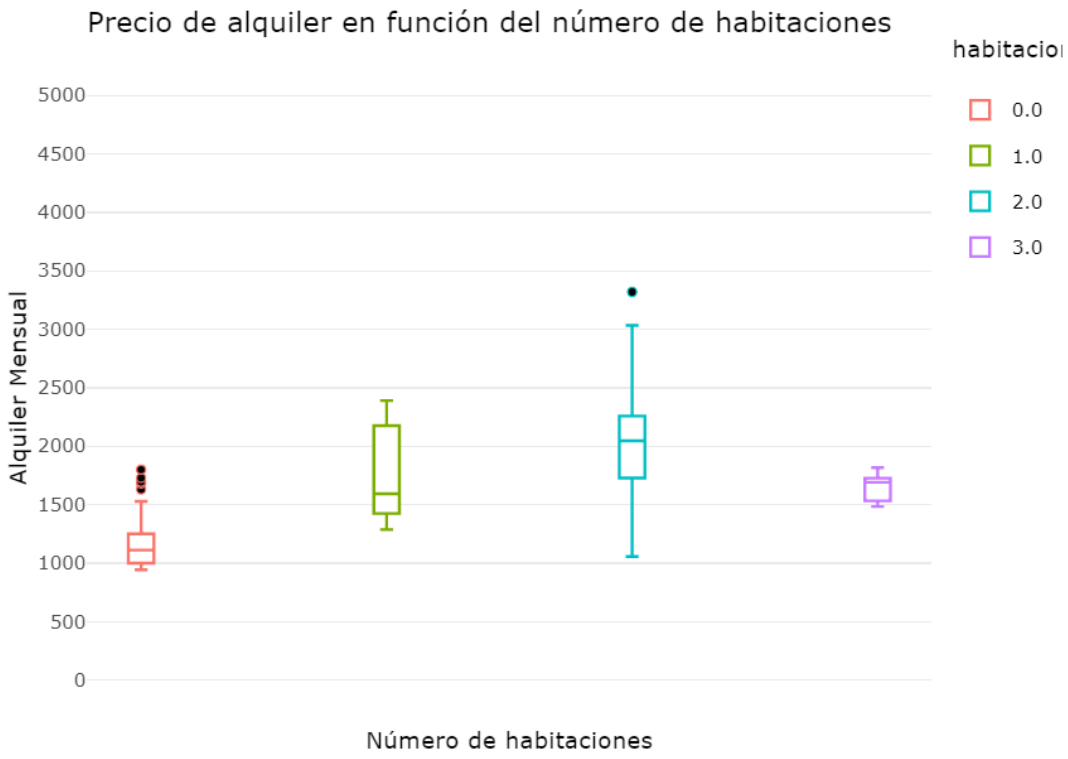

Figura 24: Gráfico interactivo que muestra la distribución del alquiler de precios en función del número de habitaciones. Fuente: Elaboración propia para exponer el uso del diagrama de caja por parte del equipo de proveedores.

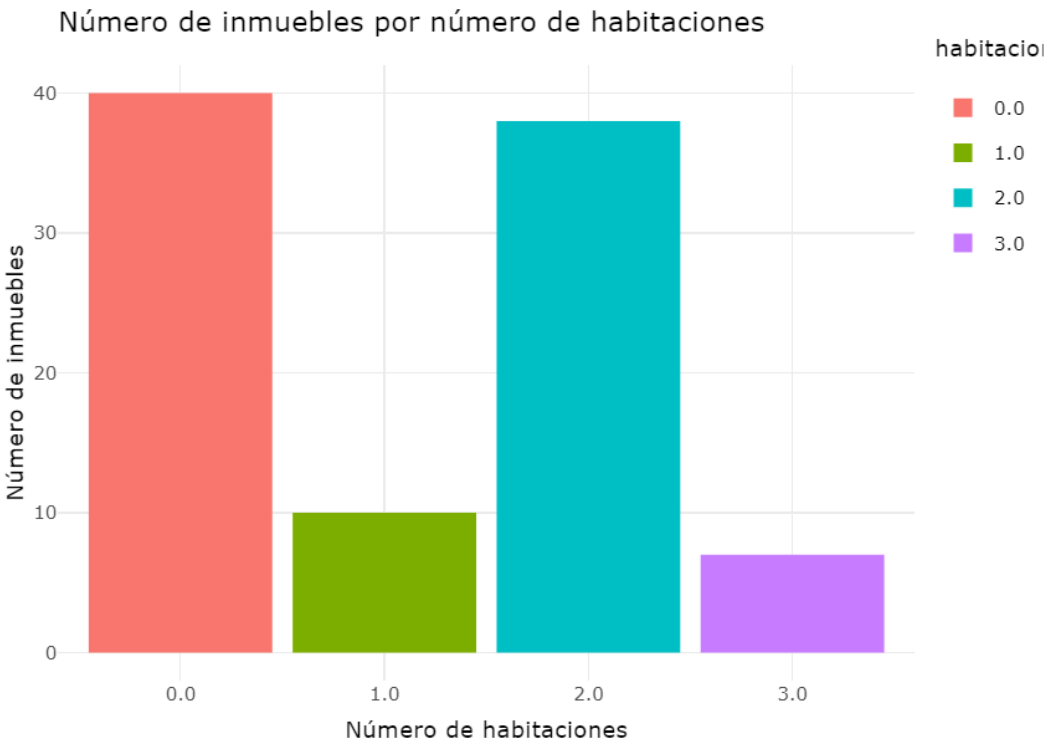

Figura 25: Gráfico interactivo que muestra el número de inmuebles por número de habitaciones en el distrito. Fuente: Elaboración propia para exponer el uso del gráfico de barras por parte del equipo de proveedores.

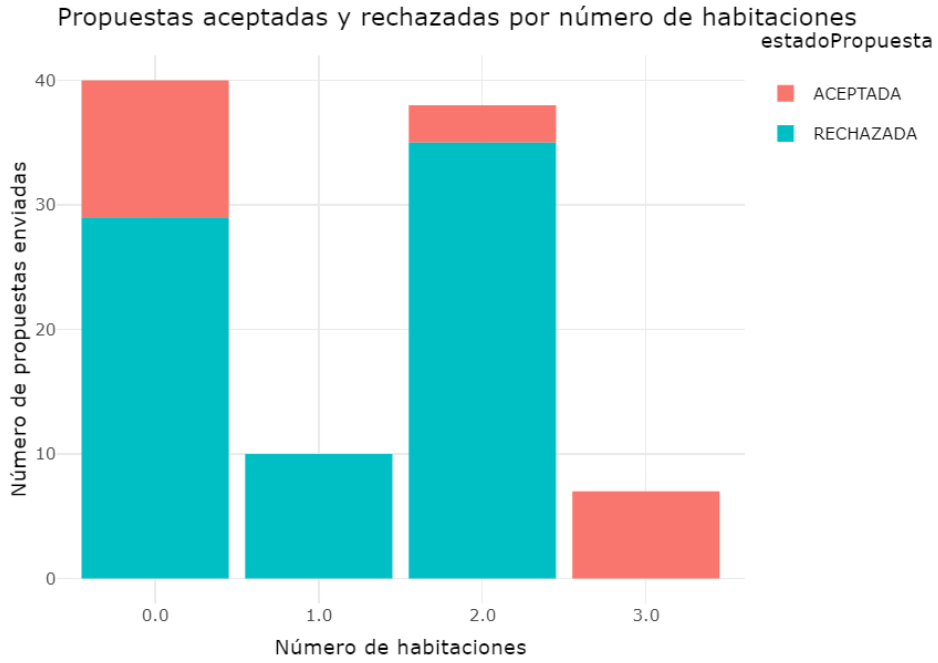

Figura 26: Gráfico interactivo que muestra el número de propuestas aceptadas y rechazadas en función del número de habitaciones. Fuente: Elaboración propia para exponer el uso del gráfico de barras por parte del equipo de proveedores.

3.4.4. Curación

Otra tarea para la que el equipo de proveedores recurre a la herramienta es para tareas de curación. Como hemos comentado en la sección *1.2.2 Conceptos relacionados con homyspace*, llamamos curación de pro- veedores a **la mejora de la base de datos existente de proveedores.** Frecuentemente, esto implica maximizar el número de enlaces para acceder a los calendarios electrónicos asociados a los inmuebles.

Esto facilita la creación automática de propuestas, ya que no hace falta comprobar la disponibilidad de un inmueble manualmente. Nos lo indica *iCal*.

Por lo tanto, a la hora de realizar curación de la base de datos, el equipo de proveedores sigue una estrategia similar a la empleada en captación:

Primero, el gestor entra en *Kibana* y analiza qué zonas tienen más demanda con el dashboard previamente mostrado en la figura 19. Después, entra en la aplicación y analiza cada uno de los distritos de esa zona, dándole particular importancia al gráfico mostrado en la figura 16.

Tras haber visualizado el porcentaje de inmuebles con *iCal*, así como el número de inmuebles y el ratio de conversión de ese distrito, el gestor establece un porcentaje objetivo de inmuebles con iCal a alcanzar.

Hecho esto, el gestor decide **qué distritos priorizar y cuántos de ellos tratar.**

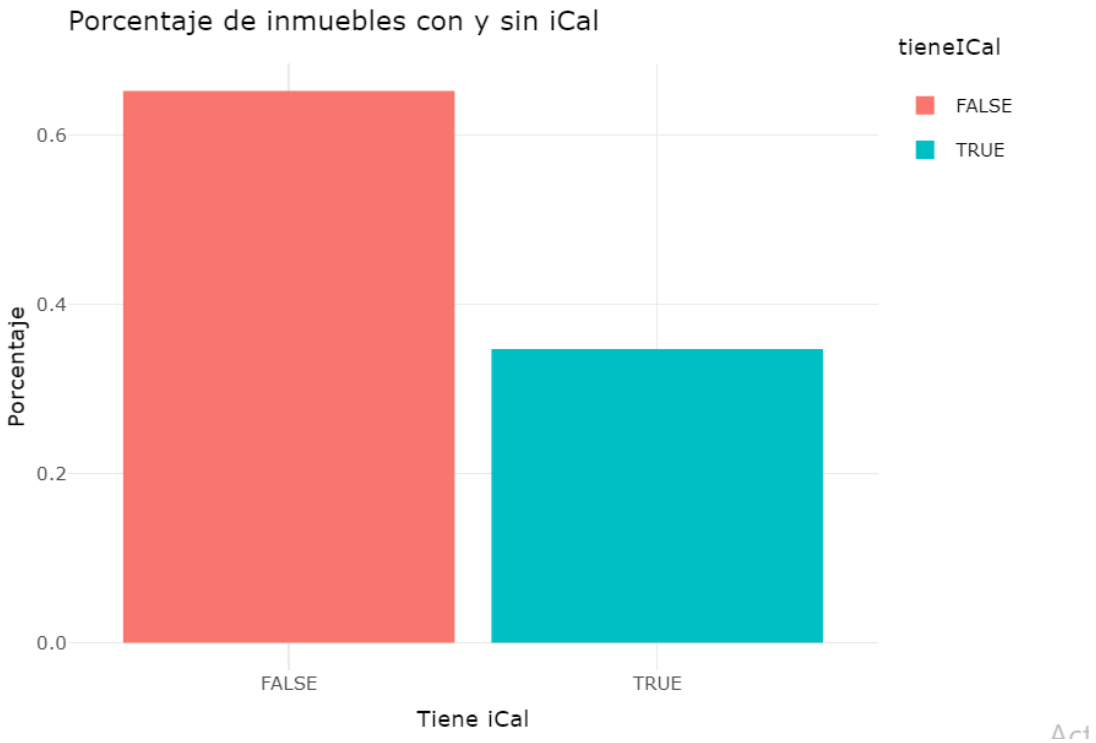

Figura 27: Gráfico interactivo que muestra el porcentaje de inmuebles con y sin iCal en el distrito. Fuente: Elaboración propia para exponer el uso del gráfico de barras por parte del equipo de proveedores.

3.5. Aplicación como herramienta de inversión

Por último, la aplicación se ha utilizado como **asistente para la toma de decisiones en el sector de inversión inmobiliaria.**

Para ello, el inversor comparte las direcciones y características de los inmuebles que está valorando para realizar la inversión.

Tras esto, enviamos las direcciones como input a un algoritmo que devuelve como output la geolocalización de estos edificios, así como a qué distrito pertenecen. Después, añadimos estos datos a la información inicial, constituyendo el fichero que vamos a pasar al mapa de calor.

Entonces, añadimos la información del fichero como una capa adicional al mapa de calor inicial, indicando con marcadores la localización de estos edificios. También otorgamos interactividad a estos marcadores, pues al clicarlos obtendremos información con respecto a su número de inmuebles en función del número de habitaciones, al mismo tiempo que el resto de gráficos interactivos se adaptarán para mostrarnos la información relativa al distrito que contiene el marcador en cuestión.

Aunque el ejemplo mostrado a continuación sólo contiene 4 ejemplos de edificios, en los casos de uso en los que se ha sido empleado las opciones superaban los 200 edificios por toda España, por lo que poder reflejar todas estas ubicaciones en un mapa interactivo facilita la toma de decisiones, agilizando la comparación de opciones, así como amplificando la información relacionada con cada una de ellas.

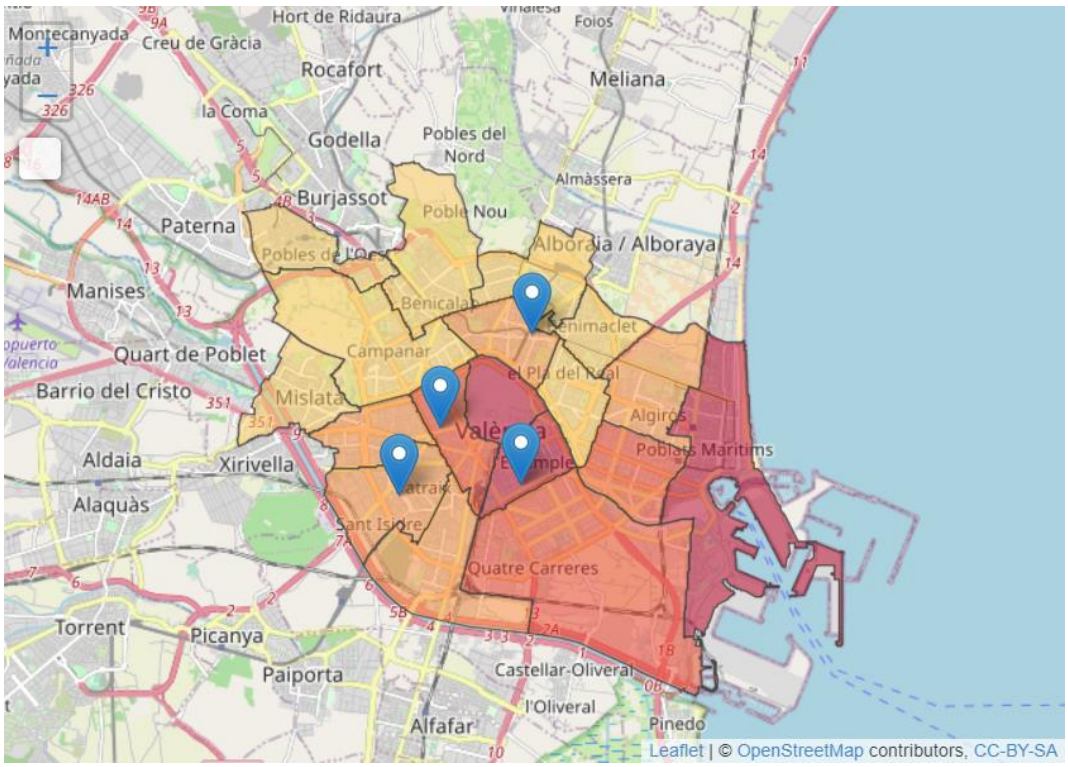

Figura 28: Mapa de coropletas de los distritos censales de Valencia con una capa añadida con las localizaciones de los edificios disponibles en los que invertir. Fuente: Elaboración propia para exponer el uso del mapa como herramienta de inversión.

4. Desarrollo de la aplicación

En esta sección, abordaremos la parte más técnica de este proyecto: Desde la creación de *ETL pipelines*, cuyo significado explicaremos en la sección *4.1.2 ETL Pipeline: Definición*, hasta la selección y desarrollo de los diferentes gráficos que componen la aplicación, así como la configuración de la interacción dinámica entre los diferentes componentes de la misma.

Para empezar, documentaremos la creación de las fuentes de datos: De dónde obtenemos la información y el proceso que seguimos para unificarla en una sola plataforma y facilitar su uso. Nos centraremos en transmitir la importancia de la creación de una fuente (o fuentes) de datos, así como de su mantenimiento.

Después, presentaremos el desarrollo de la aplicación de gráficos interactivos, tanto de cada uno de sus componentes como de la interacción entre los mismos, donde hacemos uso de la funcionalidad de *reactividad* de *Shiny*, concepto que definiremos en la sección *4.5.5 Desarrollo de los gráficos y de la reactividad*. También justificaremos por qué hemos decidido emplear cada gráfico en particular y no otro.

4.1. Fuente de datos

4.1.1. Introducción

Aunque no sea una tarea visible, **la preparación de la información para que pueda ser explotada es la base de cualquier proyecto relacionado con la visualización de datos.**

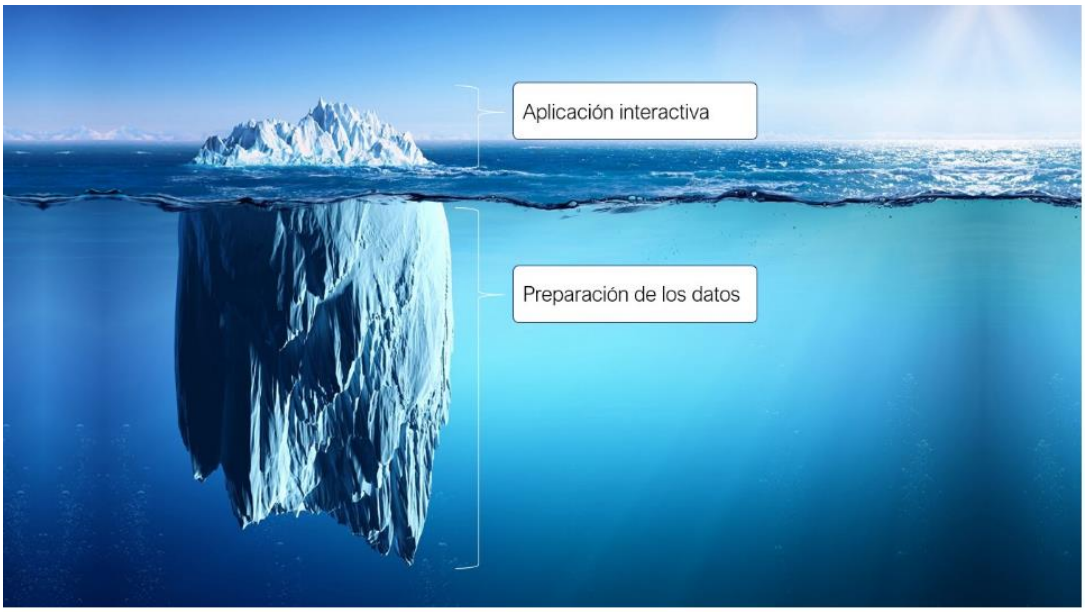

Figura 29: Imagen que refleja la importancia de la preparación de los datos en un proyecto que gira en torno a los mismos.

Es, por lo general, la parte del proyecto más laboriosa, extensa y, a menudo, técnica. Antes de explicar qué motivos nos llevan a decir esto, nos es necesario definir terminología básica que se utiliza en el campo de la adaptación de datos para su uso:

- **Limpieza de datos**: *data cleaning* en inglés, consiste en detectar y tratar información

incompleta, incorrecta, imprecisa o irrelevante procedente de un conjunto de datos, ya sea modificándola, reemplazándola o eliminándola del mismo conjunto. Forma parte del proceso de transformación de los datos, que explicaremos en la sección *4.1.2 ETL Pipeline: Definición*.

- **Datos crudos**: *raw data* en inglés, nos referimos al conjunto de datos antes de pasar por el proceso de limpieza.
- **Datos limpios**: *clean data* en inglés, es el conjunto de datos tras haber pasado por el proceso de limpieza.

Aclarados estos términos, ya podemos adentrarnos en la explicación de por qué la preparación de la información puede ser tan compleja.

Para empezar, los datos crudos rara vez están adaptados para facilitar su explotación. Esto se puede deber a diferentes razones:

- Los datos no han sido almacenados siguiendo una estructura que facilite su almacenamiento. El proceso a seguir para optimizar el almacenamiento de la información no tiene por qué ser el mismo que el empleado para optimizar su explotación.
- Hay errores en el proceso de almacenamiento: Por ejemplo, los casos vacíos se gestionan de manera diferente en cada columna o en función del tipo de información almacenada. Un ejemplo claro de esto es rellenar los casos vacíos de una columna compuesta por elementos del tipo *string* con "" y los casos vacíos de una columna compuesta por elementos del tipo *float* con NA.
- El conjunto de datos no va acompañado de un documento que explique el significado de cada
- columna. Cada conjunto de datos está almacenado en una plataforma diferente.

Por otro lado, durante el proceso de limpieza de datos nos encontramos con problemas nuevos frecuentemente, por lo que la automatización de la limpieza de cada conjunto de datos que obtengamos es inviable. Para expresarnos de manera clara y concisa, reciclaremos la frase de Tolstói: *"Todas las familias felices se parecen unas a otras, pero cada familia infeliz lo es a su manera"*.

Para el proceso de adaptación de los datos crudos, diríamos algo así: ***"Los conjuntos de datos adaptados para su explotación se parecen unos a otros, pero cada conjunto de datos crudos es diferente a su manera"***. Cada vez que tratemos un conjunto de datos por primera vez, tendremos que ser conscientes de esto. Todavía no existe ningún software capaz de recibir como *input* cualquier conjunto de datos y devolver como *output* los datos adaptados para su explotación.

4.1.2. ETL Pipeline: Definición

Las siglas representan *Extract* (extraer), *Transform* (transformar) y *Load* (cargar). Del mismo modo, una *pipeline* en ingeniería informática se refiere a un conjunto de elementos que se ejecutan secuencialmente, donde el *output* de cada elemento es el *input* del siguiente.

Por ende, Una *ETL Pipeline* es un conjunto de procesos que engloba extraer datos de diferentes fuentes de datos, transformarlos para permitir su uso y cargarlos en una base de datos o un sistema de alojamiento de archivos.

Para este proyecto, la *ETL Pipeline* que desarrollaremos tendrá como fin albergar en el sistema de almacenamiento a elegir, en este caso *Dropbox* debido a su facilidad de conexión a través de su *API*, un conjunto de datos adaptados para poder analizar el estado de cada distrito desde el punto de vista de la oferta (inmuebles), la demanda (oportunidades) y su intersección (propuestas).

4.1.3. Orígenes de datos

Utilizamos diferentes fuentes de datos para componer los distintos conjuntos de datos de los que hace uso la aplicación. Obtenemos los datos crudos de **dos fuentes diferentes**:

Del Instituto Nacional de Estadística obtenemos el archivo del tipo *Shapefile* que vamos a utilizar para definir los distritos censales [5]. Para ello, tendremos que realizar una **transformación de datos**.

Este archivo crea multipolígonos (conjunto de puntos geográficos, cada uno con sus propias coordenadas, que al unirse crean un espacio cerrado) que delimitan cada sección censal en el mapa de España. No obstante, las secciones censales *son demasiado pequeñas como para agrupar un conjunto de inmuebles lo suficientemente grande como para que las conclusiones extraídas sean significativas*. Por ello, las *disolvemos* en los distritos censales que utilizaremos.

Cada distrito censal está compuesto por un conjunto de secciones censales, y disolverlas implica agrupar todos los multipolígonos de cada distrito censal y convertirlo en uno sólo.

De hecho, esta es la primera *ETL Pipeline* de la que hacemos uso. ¿Por qué? Porque estamos extrayendo los datos del INE *(extract)*, transformándolos de su unidad original como secciones censales a distritos censales *(transform)* y, por último, subiendo la información ya transformada a *Dropbox* para poder cargarla y utilizarla cuando nos sea conveniente *(load)*. Por ende, podemos decir que **para construir nuestra *ETL Pipeline* principal es necesario crear una previa, menos compleja.**

Por otro lado, los datos de **homyspace** los obtenemos de *Elasticsearch*: recurrimos a diferentes índices para obtener la información. Como hemos indicado en la sección *3.4.2 Captación*, un índice es una base de datos que, por lo general, almacena todo el contenido relacionado con un concepto. **Ya que estamos interesa- dos en las oportunidades, propuestas e inmuebles de homyspace, necesitamos conectarnos a *Elasticsearch* para acceder a sus índices correspondientes.**

Para extraer estos datos, volvemos a recurrir a *python* y creamos una función a la que llamamos getData, a la cual pasaremos como *input* el índice en cuestión, las columnas que deseamos extraer de ese índice, y filtros opcionales.

Habiendo utilizado la primera pieza de código para preparar la información geoespacial del *INE* para su uso, y teniendo la capacidad de extraer datos de los diferentes índices de *Elasticsearch* a través de la segunda pieza de código, ya estamos preparados para poder **diseñar y desarrollar la pipeline que vamos a utilizar para adaptar los datos para el uso por parte de la aplicación.**

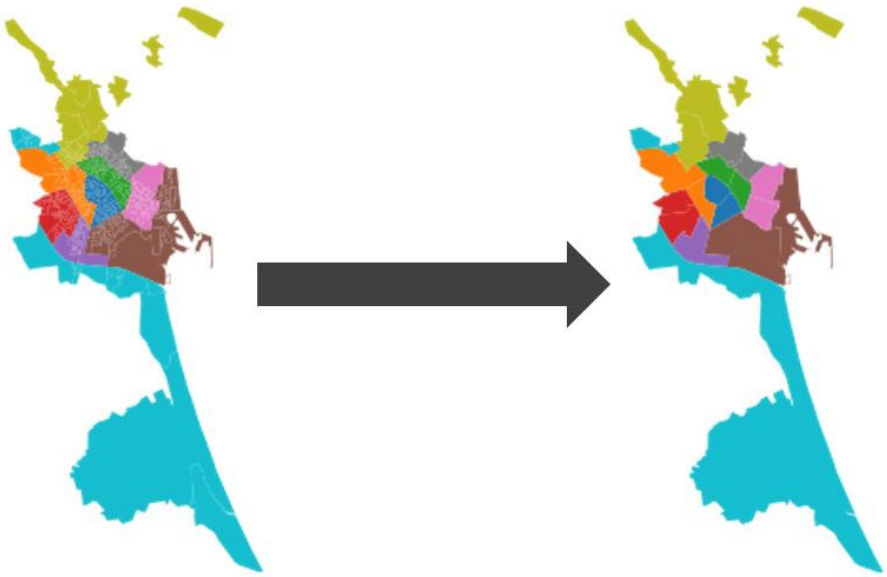

Figura 30: Imagen que refleja la disolución de las secciones censales del municipio de Valencia a distritos censales.

[5]Cartografía digitalizada: Contorno de las secciones censales a 1 de noviembre de 2011 en formato SHP (comprimido ZIP)

4.2. Diseño de la Pipeline

La *ETL Pipeline* se diferencia en las etapas de extracción, de transformación y de carga de los datos.

Para la etapa de **extracción**, haremos uso de lo mencionado en el punto *4.1.3 Orígenes de los datos*: por un lado, descargaremos el fichero de distritos censales que tenemos alojado en *Dropbox* y, por el otro, a través de la función *getData*, extraeremos de los diferentes índices de *Elasticsearch* la información correspondiente a las oportunidades, las propuestas y los inmuebles.

Hecho esto, también crearemos funciones para la segunda etapa, **transformación**. Primero desarrollaremos aquella función que, dados como *inputs* unas coordenadas y los distritos censales del *INE*, devuelva el distrito censal al que pertenecen estas coordenadas. Esta función la utilizaremos tanto con los inmuebles como con las oportunidades.

Después, y ya con los inmuebles asociados a sus respectivos distritos, podremos obtener los distritos de las propuestas identificando los inmuebles de cada propuesta. De este modo, ya habremos localizado las oportunidades, inmuebles y propuestas de cada distrito.

Por último, agruparemos toda esta información por distrito, para tener en un único fichero la información relativa a la demanda, oferta y el encaje de estas de cada distrito.

En lo que a la fase de **carga** se refiere, nos conectamos con *Dropbox*, donde guardaremos cuatro ficheros diferentes.

Los dos primeros serán del tipo *Shapefile*, e indicarán la geolocalización de los inmuebles y de las oportunidades. La razón por la que creamos estos documentos y los almacenamos es porque el volumen de inmuebles y oportunidades es lo suficientemente grande como para ralentizar la ejecución de la *ETL Pipeline* si lo hiciéramos desde cero cada vez. Almacenando estos datos, podemos ejecutar la función de la localización de las coordenadas en los distritos **únicamente** en los inmuebles y oportunidades que no han sido previamente localizados.

Asimismo, convertimos en *CSV* las propuestas con los distritos indicados. Este fichero lo utilizaremos para la mayoría de gráficos interactivos que no son el mapa. Todo lo que necesitamos para convertir el documento a *CSV* es eliminar la columna que indica el multipolígono del distrito en el que se encuentra cada propuesta.

Además, exportaremos a *CSV* el documento en el que hemos agrupado y resumido la información por distritos, siguiendo el mismo proceso de eliminación de la columna con el multipolígono. Utilizaremos este fichero para la tabla resumen que se encuentra al final de la aplicación.

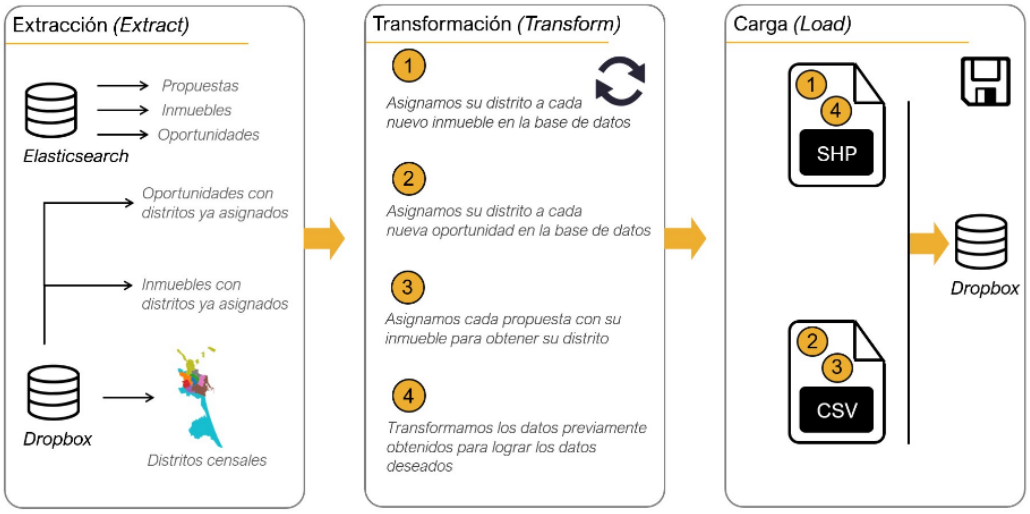

Figura 31: Estructura de la ETL Pipeline desarrollada para la aplicación.

4.3. Desarrollo de la Pipeline

Para empezar, cargaremos las librerías necesarias para acceder a las diferentes fuentes de datos.

Utilizaremos *elasticsearch* y *elasticsearch_dsl* para conectarnos con *Elasticsearch*. Como en *homyspace* utilizamos la versión de *Amazon Web Services (AWS)*, también cargaremos *requests_aws4auth*. Esta librería nos permite autentificarnos con *AWS* utilizando las credenciales adecuadas.

A través de la librería de *dropbox*, seremos capaces de acceder a *Dropbox*, tanto para descargar documentos como para subirlos.

Con *requests* y *zipfile* extraeremos los archivos *ZIP* en los que tenemos comprimidos los documentos *Shapefile* de los distritos y de los inmuebles, cuyos distritos ya hemos identificado previamente.

Utilizaremos *pandas* y *numpy* para gestionar los datos. Con estas dos librerías realizaremos toda la limpieza de datos que sea necesaria.

Por último, *geopandas* y *shapely* nos permitirán tratar la información geoespacial, dándonos la posibilidad de crear y alterar documentos *Shapefile*.

Siguiendo los pasos necesarios para poder realizar la etapa de extracción, nos conectamos con *Elasticsearch*.

Habiendo instalado las librerías necesarias, así como habiendo desarrollado las funciones y la conexión a *Drop- box* y *Elasticsearch* respectivamente, **nos disponemos a diferenciar las funciones creadas y utilizadas en función de la etapa de la *ETL Pipeline* a la que pertenecen**.

Después, en la sección *4.3.4 Arquitectura final y gestión de la actualización* indicaremos cómo unimos estas funciones para crear la *ETL Pipeline* que deseamos.

4.3.1. Extracción (extract)

Comenzaremos por la etapa de **extracción**. Como hemos mencionado en la sección *4.1.2 ETL Pipeline: Definición*, el propósito de esta etapa es habilitar el acceso a la información para ser

capaces de hacer uso de ella más adelante.

Dividiremos las funciones que hacemos en dos:

- Por un lado, tenemos ***getPropertiesData***, ***getOpportunitiesData*** y ***getProposalsData***. Estas tres funciones nos conectan con *Elasticsearch*, y nos permiten extraer los datos de los índices relacionados con inmuebles, oportunidades y propuestas, respectivamente. En la figura **31**, representaría la parte relacionada con *Elasticsearch* de la etapa de **extracción**.
- Por el otro, hemos creado las funciones ***getDistrictsFromDBX***, ***getPropertiesFromDBX*** y ***ge- tOpportunitiesFromDBX***. Con estas funciones podemos acceder a los distritos censales y a los inmuebles y oportunidades cuyos distritos ya han sido asignados en previos lanzamientos de la *ETL Pipeline*.

Con este conjunto de funciones ya hemos satisfecho todos los requerimientos para acceder a la información que nos aporta valor. No obstante, todavía no ha sido tratada para que nos aporte ese valor. Para ello, necesitamos del proceso de **transformación**, en el cual ahondaremos a continuación.

4.3.2. Transformación (transform)

El proceso de **transformación** de los datos consiste en, dado un conjunto de información recibida como input, realizar una serie de alteraciones de ese input (ya sean adiciones, substracciones o modificaciones) para obtener un output que nos sea útil para un fin específico.

Al igual que en la etapa de **extracción**, podemos dividir las funciones para la **transformación** en dos:

Primero, tenemos las funciones que utilizaremos para, una vez recibidos los datos de los inmuebles y oportunidades, asignar a cada uno de ellos el distrito en el que se encuentran. En la figura **31**, estas funciones equivalen a la **1** (***assignDistrictsToProperties***) y **2** (***assignDistrictsToOpportunities***) de la etapa de **transformación**.

El paso **3** (***assignPropertiesToProposals***) también pertenece a este subconjunto, aunque se diferencia en el hecho de que, como ya hemos asignado los inmuebles a sus distritos, todo lo que tendremos que hacer para localizar las propuestas será relacionarlas a sus inmuebles asociados, cuyos distritos ya han sido aclarados.

La última de las funciones que necesitamos para completar el proceso de **transformación** es la que equivale al paso **4** en la figura **31**. Con esta función agruparemos por distrito todo la información relativa a las oportunidades, los inmuebles y las propuestas.

De esta transformación obtendremos los datos que necesitamos para crear el mapa en la aplicación de gráficos interactivos. No obstante, para ello necesitamos almacenar esta información y la que hemos transformado en una plataforma de almacenamiento de datos. Nosotros hemos elegido *Dropbox* debido a la facilidad de conexión con su *API* y a que, además, podemos guardar hasta 2GB gratuitamente, lo cual es más que de sobra para nuestro caso de uso.

En la sección *7.1 Desarrollo de la aplicación* mostramos las funciones que hemos utilizado para almacenar la información correspondiente en *Dropbox*.

4.3.3. Carga (load)

En la última etapa de la *ETL Pipeline* nos centramos en almacenar toda la información que hemos obtenido como *output* de la etapa de **transformación**.

Siguiendo la misma dinámica de las dos etapas anteriores, en la etapa de **carga** podemos diferenciar en dos las funciones creadas:

- Por un lado, creamos las funciones ***backup***, ***checkFileDetails*** y ***uploadDropbox***. A través de las mismas subiremos los ficheros correspondientes a la platafroma de *Dropbox*.
- Por el otro, tenemos las funciones ***savePropertiesToDBX***, ***saveOpportunitiesToDBX***, ***saveHeatmapToDBX*** y ***saveProposalsToDBX***. Con estas funciones estamos transformando los datos en ficheros *CSV* o *SHP* y, para el último, comprimiéndolo en un *ZIP* antes de subirlos a *Dropbox* con ayuda de la función ***uploadDropbox***.

Habiendo creado todas estas funciones, **ya somos capaces de hacer uso de nuestra propia *ETL Pipeline***.

En la próxima sección describiremos cómo unimos todas estas funciones para que la *ETL Pipeline* funcione como debería, así como el proceso que seguimos para que la información se mantenga actualizada.

4.3.4. Arquitectura final y gestión de su actualización

A continuación, definiremos la estructura que sigue la *ETL Pipeline* para aportarnos los datos de los que la aplicación de gráficos interactivos se alimenta.

```
# Extraemos los inmuebles, los asociamos a sus distritos y los almacenamos en Dropbox
propertiesData = getPropertiesData()
properties = assignDistrictsToProperties(propertiesData)
savePropertiesToDBX(properties)

# Extraemos las oportunidades, las asociamos a sus distritos y las almacenamos en Dropbox
opportunitiesData =  getOpportunitiesData()
opportunities = assignDistrictsToOpportunities(opportunitiesData)
saveOpportunitiesToDBX(opportunities)

# Extraemos las propuestas, las asociamos a sus inmuebles y las almacenamos en Dropbox
proposalsData = getProposalsData()
proposalsData = assignDistrictsToProposals(proposalsData)
saveProposalsToDBX(proposalsData)

# Agrupamos los datos de las propuestas por distritos y los almacenamos en Dropbox
heatmap_data = groupByDistricts(proposalsData)
saveHeatmapToDBX(heatmap_data)
```

En lo relativo a la actualización, la realizamos ejecutando la *ETL Pipeline* los viernes por la tarde, una vez la jornada laboral ha finalizado con el *Planificador de tareas* de *Windows*. Esto nos permite ejecutar de manera automática el código semanalmente.

¿Por qué semanalmente? Lo ejecutamos únicamente una vez por semana ya que la información no aumenta de forma notable de un día para otro, y el coste de oportunidad de correr el código es que no se debería utilizar la aplicación de gráficos interactivos mientras la *ETL Pipeline* está en proceso, y el tiempo de ejecución es de unos cuantos minutos. En el futuro automatizaremos este proceso para poder ejecutarlo desde la nube, permitiéndonos realizar la actualización durante horas no lectivas.

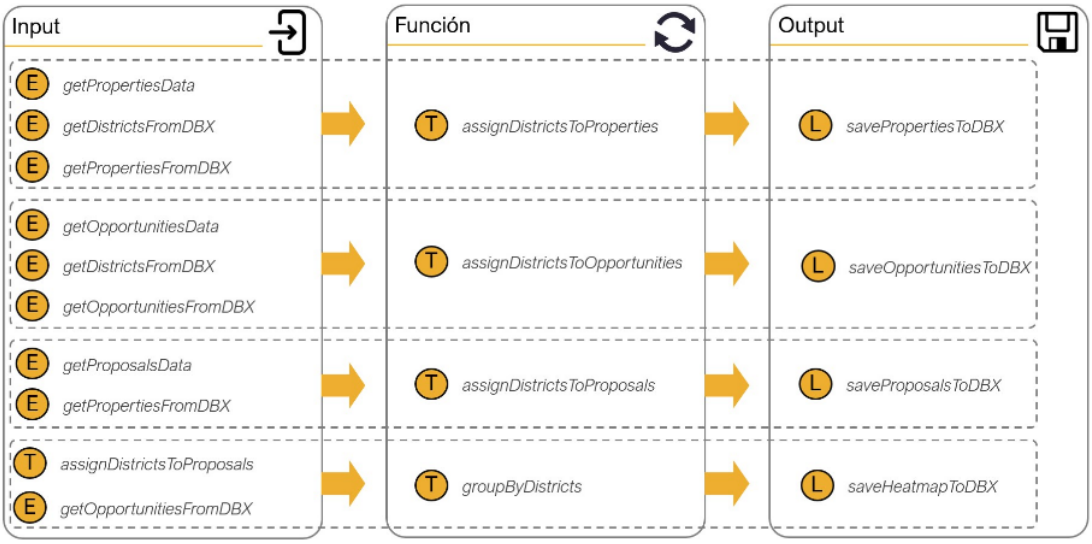

Figura 32: Funciones utilizadas como inputs, transformadores y outputs respectivamente.

4.4. Gráficos Interactivos

4.4.1. Introducción

Teniendo los datos listos para su explotación, a continuación abordaremos la creación de la aplicación para el aprovechamiento de los mismos.

Esta sección, por lo tanto, girará en torno al desarrollo de aplicaciones interactivas: explicaremos los diferentes componentes de nuestra aplicación, cómo queremos que interactúen entre sí, cómo deseamos organizar cada uno de los gráficos en la aplicación y, por último, cómo podemos hacer todo esto con ayuda de diferentes librerías de **R**, principalmente *shiny*.

En la sección *4.1 Fuente de datos*, resaltamos la importancia de crear un conjunto de datos para que pueda ser explotado. Ahora es necesario destacar la necesidad de transformar los datos en información comprensible para el usuario a través de un conjunto de gráficos que aporten valor adicional.

¿Cómo podemos aportar valor al usuario a través de una aplicación de gráficos interactivos?

- Creamos gráficos que respondan a las preguntas más importantes y frecuentes que se pueden hacer los usuarios con respecto a la información dada. Sabiendo qué problema queremos resolver o qué decisión queremos ayudar a tomar, podemos plantearnos qué gráficos mostrar y en torno a qué tipo de información.

En nuestro caso, si queremos ayudar a negociar precios, por ejemplo, podremos crear un diagrama de caja que muestre la distribución de los precios en función del número de habitaciones de los inmuebles de una zona concreta. En la próxima sección desarrollaremos este punto.

- Interactividad entre los componentes de la aplicación para permitir una navegación fácil a través de los datos. Aprovechamos la capacidad de *shiny* de facilitar la interactividad entre diferentes gráficos, para filtrar de manera automática en el resto de gráficos de la aplicación la información que seleccionamos en uno de ellos. Hablaremos más de este concepto en la sección *4.4.3 Reactividad en Shiny: qué es y cómo explotarla*.
- Filtros para que el usuario pueda acotar tanto como quiera la información obtenible. Como complemento al punto anterior, facilitamos en la aplicación un conjunto de filtros que permiten al usuario la mayor precisión posible a la hora de obtener la información que desea.
- Experiencia agradable para el usuario. A través de un diseño atractivo y una usabilidad sencilla e intuitiva logramos que el usuario disfrute su experiencia analizando la información mostrada. Esto resulta esencial, ya que buscamos la recurrencia de uso.
- Acciones no relacionadas directamente con el desarrollo, pero igual de importantes, tales como la recogida de feedback y la corroboración de la fiabilidad de los datos. Pretenden dotar de credibilidad a la aplicación.

Si seguimos estos puntos lograremos construir una herramienta que aporte valor al usuario final y añadirá calidad a su toma de decisiones, teniendo un impacto final en la empresa.

4.4.2. Componentes

A continuación, expondremos los diferentes gráficos que componen la aplicación interactiva, siguiendo la dinámica de la sección *3.3 Aplicación de gráficos interactivos para la toma de decisiones: aspecto y componentes*. No obstante, en esta sección explicaremos **por qué hemos elegido cada tipo de gráfico entre algunas de las alternativas disponibles.**

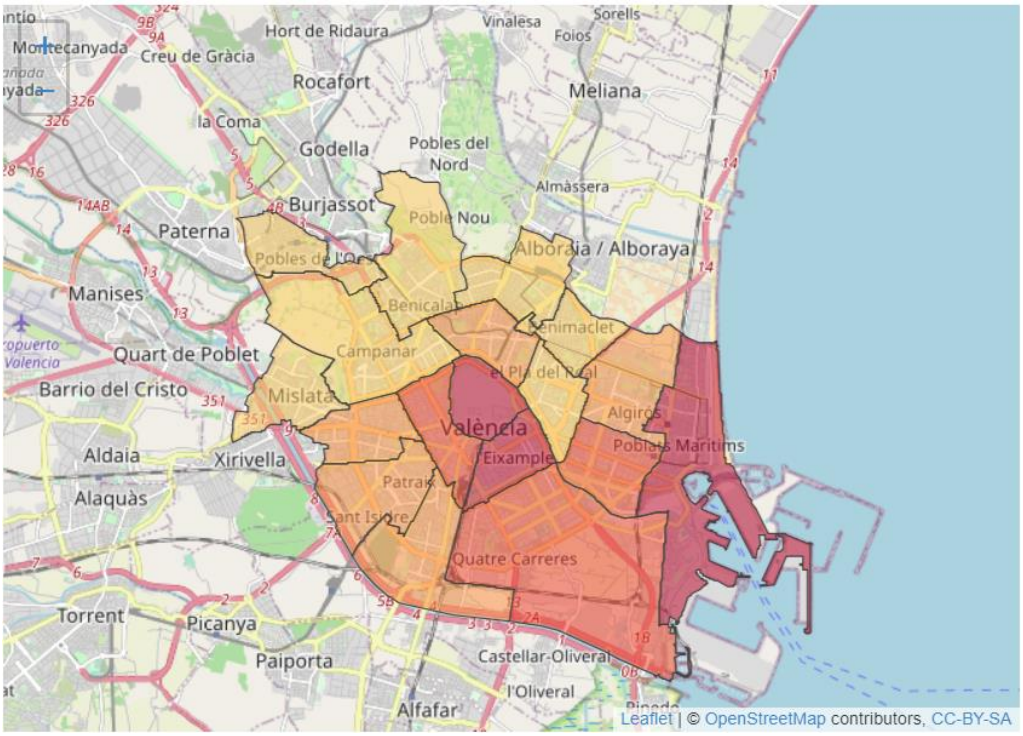

Figura 33: Mapa de coropletas de los distritos censales de Valencia. Fuente: Elaboración propia para exponer el uso del mapa.

Como hemos visto antes, el primer gráfico con el que nos encontramos al abrir la aplicación es el mapa de calor. Lo hemos llamado *mapa de calor* debido al rango de colores del que hace uso que, en función del campo que seleccionemos, será más claro u oscuro en cada distrito. No obstante, el nombre técnico correcto para este tipo de mapa es *mapa de coropletas,* o *choropleth map*, pero utilizamos el término "calor" debido a que es más comprensible y por lo tanto más fácil de utilizar en conversaciones dentro de **homyspace**.

Un *mapa de coropletas* es un tipo de mapa que está compuesto por regiones delimitadas en el espacio geográfico, en nuestro caso distritos (aunque podrían ser municipios, provincias, etc.), que se colorean de diferentes colores, en general dentro de una misma gama cromática, en función de una variable.

La variable (a escoger) para nuestro mapa es número de oportunidades, de inmuebles o de propuestas aceptadas.

Hemos elegido este tipo de mapa principalmente por tres razones:

1. Necesidad de crear agrupaciones más pequeñas que las que ya existían en la base de datos de **homyspace** (la unidad más pequeña es la ciudad), con el propósito de tener información más precisa en función de la geolocalización de la oportunidad, propuesta o inmueble.

2. Necesidad de crear agrupaciones establecidas bajo un consenso común y que sigan una lógica por ciudad, provincia y comunidad autónoma. Esto facilita el entendimiento por parte del usuario. Previamente utilizábamos un algoritmo de agrupación *K-means*, que no seguía ninguna lógica y agrupaba inmuebles de diferentes ciudades, provincias e incluso comunidades autónomas. Esto dificultaba mucho la comprensión de los datos: *si tenemos una agrupación con inmuebles de Getafe y Madrid, y suben los precios de la agrupación, ¿es posible que sólo hayan subido los precios en Madrid? ¿o en Getafe?*

3. Siguiendo la pista del punto anterior, buscábamos agrupaciones que facilitaran la comunicación interdepartamental: *"han subido los precios en el distrito 211"* **vs** *"han subido los precios en el barrio Salamanca"*. Aunque parezca un detalle nimio, sucede que es fundamental. La capacidad de tomar decisiones de manera ágil y rápida depende directamente de la facilidad de aquel que va a tomar la decisión de transmitir la información que desea claramente, sin invertir tiempo explicando conceptos que no impactan directamente en la decisión (como pueden ser qué representa cada número, qué localizaciones engloba cada agrupación, qué lógica se ha seguido para agrupar los inmuebles, etc.)

Estas razones hacen del *mapa de coropletas* un gráfico fácil de entender incluso por usuarios sin conocimientos estadísticos. Esta propiedad elimina cualquier barrera de entrada a su uso, y maximiza el retorno a la inversión de los recursos que exige su desarrollo.

Tras el mapa de calor, nos encontramos con dos gráficos en la misma fila. A la izquierda (el más importante) hemos puesto el diagrama de caja segmentado por número de habitaciones.

Un diagrama de caja es un gráfico que presenta algunos estadísticos relevantes de la distribución de una variable numérica, a veces segmentada por una segunda variable. En nuestro caso, indicamos la distribución de los *precios mensuales* segmentada por *número de habitaciones*.

A raíz de un análisis exhaustivo de los precios de la base de datos de **homyspace**, nos hemos dado cuenta de que la existencia de casos anómalos es relativamente abundante en función del distrito y del número de habitaciones. Esto, sumado a la necesidad por parte del usuario no sólo de saber si un

inmueble es caro en comparación con la mediana (o la media) de precios del distrito al que pertenece, sino también dónde se sitúa el precio del inmueble en el rango de percentiles de precios del distrito, nos hace requerir de este tipo de gráfica.

¿Por qué necesita el gestor saber en qué rango del percentil se encuentra el precio del inmueble? La razón principal es que facilita la negociación de precios. Es mucho más sencillo justificar una solicitud de reducción del alquiler mensual de un inmueble si va acompañado de datos que lo apoyen: *"el precio de tu inmueble es superior al 70 % de los inmuebles de las mismas características en la zona"* **vs** *"el precio de tu inmueble es caro"*.

Aportar datos ayuda al propietario a entender por qué debe bajar los precios y, por lo tanto, facilita la cooperación.

La introducción de este tipo de gráfico requiere una inversión relativamente pequeña en formación estadística, aunque mayor que para otros tipos de gráficos, ya que varios de los miembros del equipo desconocen (o no dominan) el concepto de percentiles. No obstante, las ventajas superan con creces los inconvenientes, ya que una vez abordada esta barrera de entrada es un tipo de gráfico que aporta muchísima información de forma rápida y comprensible.

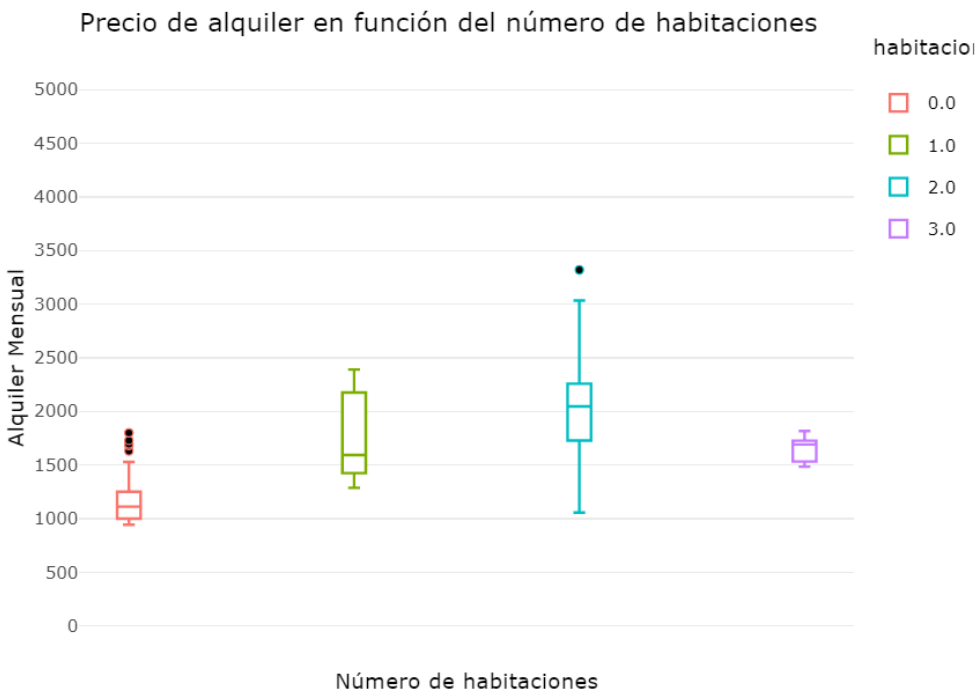

Figura 34: Gráfico interactivo que muestra la distribución del alquiler de precios en función del número de habitaciones. Fuente: Elaboración propia para exponer el uso del diagrama de caja.

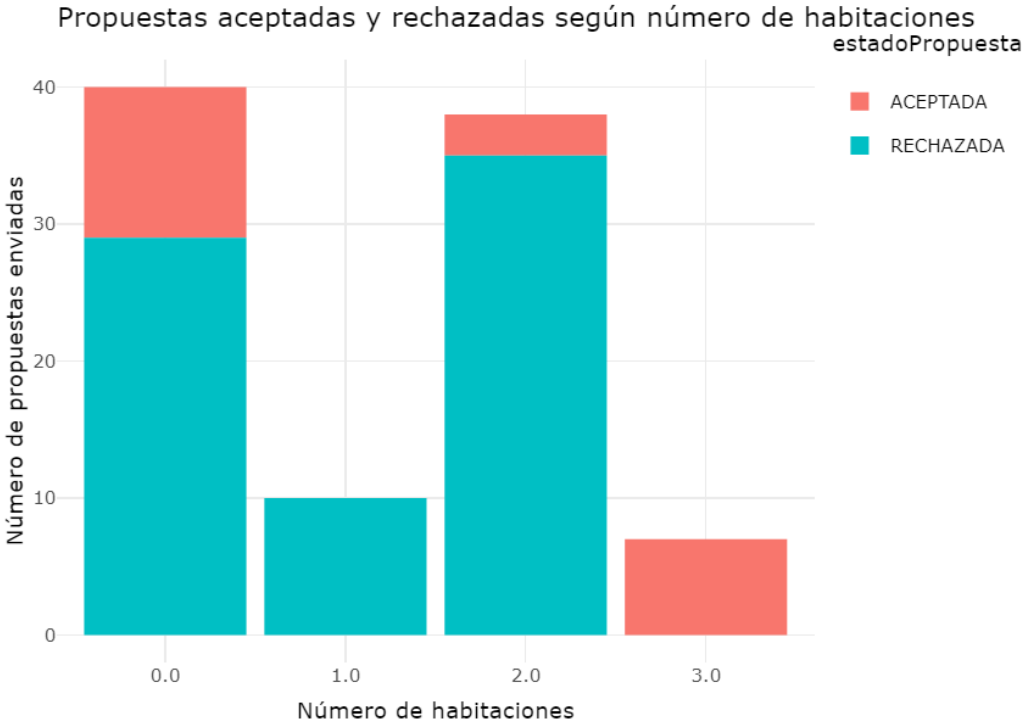

Figura 35: Gráfico interactivo que muestra el número de propuestas aceptadas y rechazadas en función del número de habitaciones. Fuente: Elaboración propia para exponer el uso del gráfico de barras.

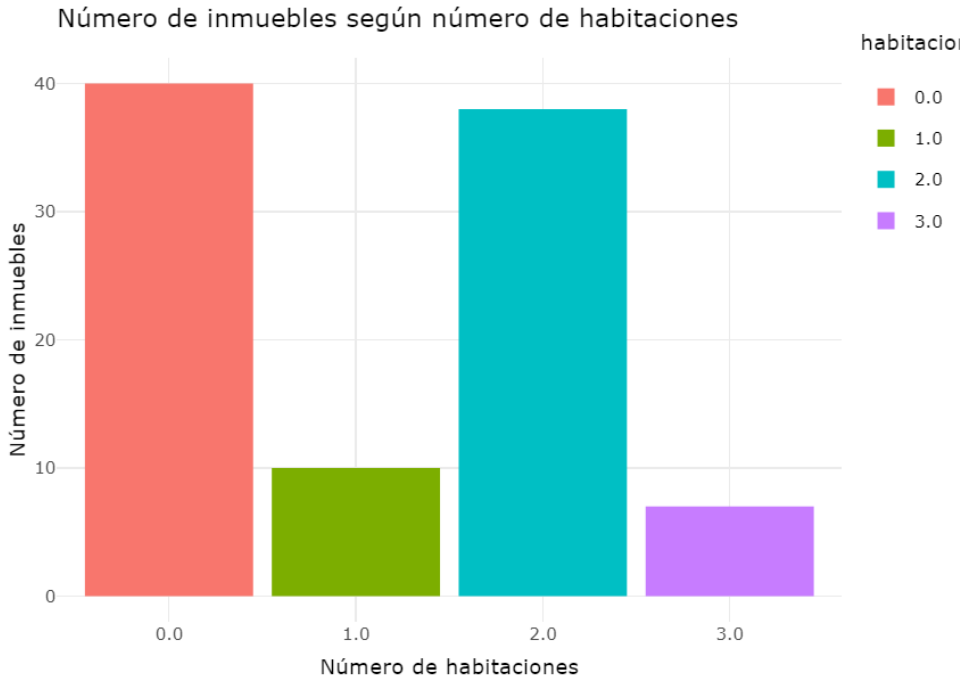

Figura 36: Gráfico interactivo que muestra el número de inmuebles según el número de habitaciones en el distrito. Fuente: Elaboración propia para exponer el uso del gráfico de barras.

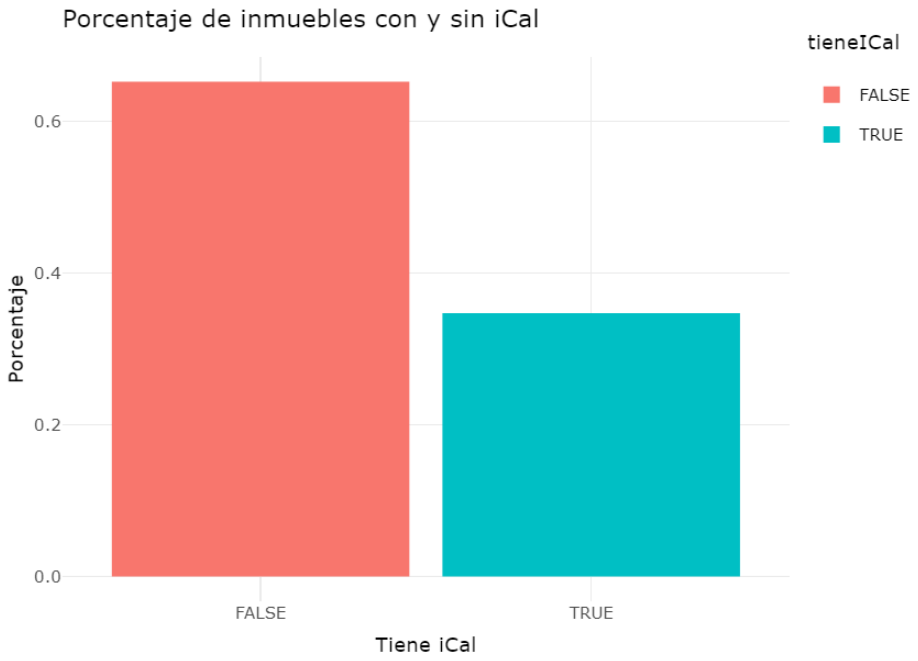

Figura 37: Gráfico interactivo que muestra el porcentaje de inmuebles con y sin iCal en el distrito. Fuente: Elaboración propia para exponer el uso del gráfico de barras.

A veces podemos segmentar esta variable en función de una segunda variable, como hacemos en la figura 35 con la variable X *Número de habitaciones* y la variable *estadoPropuesta*.

Es un gráfico fácil de entender y que no tiene prácticamente curva de aprendizaje. De ahí que recurramos tanto a él.

En la aplicación, sirven tanto un propósito en sí mismos, como pueden ser el segundo y tercer diagrama de barras, que aportan información valiosa sobre el distrito, como un propósito de asistencia o de validación de los datos, como ocurre con el primer gráfico que, aunque aporta información relevante acerca de las propuestas y la conversión según el número de habitaciones del distrito, su función principal es reforzar la validez de las distribuciones de precios del diagrama de caja. Es decir, **permite al usuario comprobar si hay el suficiente número de propuestas para los inmuebles con un número de habitaciones concreto como para que la distribución de precios tenga validez estadística.**

Para la selección y creación de los gráficos, hemos recurrido reiteradamente al recurso *The R Graph Gallery* [6] , que facilita no sólo decidir qué gráfico encaja más con los tipos de datos que queremos visualizar, sino que también ayuda en su desarrollo mostrándonos el código en *R*.

4.4.3. Reactividad en Shiny: qué es y cómo aprovecharla

La **reactividad** es una propiedad muy poderosa de *Shiny* que, si se aprende a utilizar, multiplica drásticamente el valor de las aplicaciones construidas con esta librería [7]. Esta propiedad nos permite interactuar con la aplicación, almacenar momentáneamente esas interacciones y actualizar la aplicación dinámicamente como consecuencia de dichas interacciones. **Logramos maximizar la interactividad y el dinamismo de una aplicación *Shiny* aprovechando sus propiedades de reactividad.**

Figura 38: Imagen que representa el concepto de reactividad.

Esta característica de *Shiny* se puede introducir principalmente de dos maneras diferentes:

- A través de *inputs*, lo que en nuestra aplicación son los filtros. Cada vez que modificamos uno de los filtros, todos los gráficos mostrados en la figura **38** se actualizarán de manera acorde para sólo mostrar la información filtrada.
- A través de interacciones del usuario con la aplicación. En nuestro caso, almacenamos los *clicks* del usuario en el mapa de calor y, cada vez que pincha en un distrito, el filtro del distrito se actualiza dejando únicamente el distrito clicado. De esta forma, se activa la reacción indicada en el punto anterior, actualizando los gráficos para mostrar exclusivamente información relacionada con el distrito seleccionado.

No obstante, debido a la capacidad de la **reactividad** de alterar de manera sustancial la experiencia de usuario, debemos planear en qué gráficos la aplicamos y cómo lo hacemos. Si nos excedemos, estaremos afectando la navegación del usuario por la aplicación, ya que la actualización de datos implica un tiempo de carga de los mismos cada vez que la **reactividad** responde a una interacción. Nos arriesgamos a que el tiempo de carga sea mayor al de exploración de la información. Del mismo modo, desencadenantes de **reactividad** demasiado complejos pueden dificultar la comprensión del funcionamiento de la aplicación al usuario, lo cual será contraproducente, debido a que reducirá el interés del usuario por recurrir a la misma.

Por eso, hemos insertado un nivel de **reactividad** en la aplicación lo suficientemente alto como para agilizar la exploración de datos por parte del usuario, al mismo tiempo que permitimos que profundice en la información disponible, pero sin aumentar la complejidad hasta el punto de que convierta los desencadenantes de la **reactividad** en una caja negra.

[6] Holtz, Yan. "The R Graph Gallery". The R Graph Gallery.
[7] Concepto de reactividad explicado por los desarrolladores de Shiny

Como hemos dicho antes, tenemos dos tipos de desencadenantes:

- Los filtros.
- El *click* en los distritos del mapa de calor.

Después de mucha prueba y error, hemos llegado a la conclusión de que **demasiada complejidad entorpece la experiencia de usuario. Nos es más rentable maximizar el nivel de sencillez de la aplicación al mismo tiempo que intentamos optimizar su usabilidad.**

Aquí tenemos un ejemplo de reactividad aplicada a un conjunto de datos *datosOriginales*.

Con la función *reactive*, convertimos este conjunto de datos estándar en datos que se actualizarán automáticamente cada vez que el filtro se modifique, manteniendo únicamente los datos cuya propiedad pertenezca al filtro seleccionado.

Después, el gráfico en cuestión llamará a los datos reactivos (que ahora son una función), y será modificado cada vez que los datos reactivos se actualicen.

```
library(shiny)
library(plotly)
library(ggplot2)

datosReactivos <- reactive({
  datosOriginales %>%
    filter(propiedadDeLosDatos %in% input$filtro)
})

output$grafico <- renderPlotly({
  datosParaElGrafico <- datosReactivos()
  p <- ggplot(datosParaElGrafico)

  ggplotly(p)
})
```

4.5. Desarrollo de la aplicación en Shiny

A continuación, ilustraremos, sin entrar en un gran nivel de detalle, el proceso de desarrollo de la aplicación en *Shiny*. Para los que deseen una mayor comprensión del proyecto, el código completo está disponible en la sección *7.2 Desarrollo de la aplicación en Shiny*.

El desarrollo de una aplicación de *Shiny*, al igual que el desarrollo de una *ETL Pipeline*, se puede dividir en diferentes partes. Por lo general, **estas secciones son:**

1. Carga de librerías.
2. Creación de funciones.
3. Carga y transformación de los datos.

4. Desarrollo de la interfaz de usuario.
5. Desarrollo de los gráficos y de la reactividad (servidor).

Siendo todavía más concisos, lo esencial de una aplicación *Shiny* es la interfaz de usuario y el servidor, *ui* y *server*. Con estos dos componentes somos capaces de desarrollar una aplicación de *Shiny*. **El código que se muestra a continuación es todo lo que necesitamos.**

```
library(shiny)

ui <- fluidPage(
      titlePanel("¡Hola Mundo!"),
      mainPanel(
          plotOutput("distPlot")
      )
)

server <- function(input, output) {

      output$distPlot <- renderPlot({
            x    <- faithful[, 2]
            bins <- seq(min(x), max(x), length.out = 10)
            hist(x, breaks = bins)
      })
}

shinyApp(ui = ui, server = server)
```

Como podemos ver de forma intuitiva mediante el código, a través de *ui* indicamos dónde vamos a colocar los componentes de nuestra aplicación. En el ejemplo, estamos señalando cómo queremos añadir el gráfico **"distPlot"** al panel principal, **mainPanel**.

Será en *server* donde desarrollemos este gráfico, siempre añadiendo el prefijo *output$* al nombre del componente, ya que queremos que su resultado se muestre en la *ui* al usuario.

4.5.1. Carga de librerías

Para empezar, cargamos en memoria las librerías que vamos a utilizar en la aplicación. Podemos agrupar estas librerías en tres categorías:

- **Relacionadas con la visualización de datos.** Estas serían *plotly* (gráficos interactivos), *leaflet* (mapas interactivos), *leaflet.extras* (añade funcionalidad a *leaflet*) y *DT* (tablas

interactivas). Hemos decidido incluir aquí *tidyverse*, que es en sí misma una agrupación de las librerías más utilizadas, ya que contiene *ggplot2*, la librería para gráficos externa de ***R*** más conocida, aunque también contiene librerías de transformación de datos como *dplyr*.

- **Relacionadas con *Shiny*.** Aquí tendríamos *shiny*, *shinymanager* (permite añadir usuario y contraseña a la aplicación), *shinyWidgets* (extiende la variedad de *widgets* que contiene *shiny* por defecto), *shiny- dashboard* (añade una capa de abstracción que permite crear aplicaciones organizadas y estéticamente agradables de forma sencilla) y *shinycssloaders* (añade animaciones de carga a los gráficos).
- **Relacionadas con la carga (y transformación) de datos.** Para esta agrupación tenemos *rdrop2* (extraemos datos de *Dropbox*), *sf* (convertimos el *Shapefile* de los distritos en información gestionable) y *lubridate* (para tratar información relacionada con fechas).

4.5.2. Creación de funciones

En lo que a funciones se refiere, crearemos 4 (y una variable) que también pueden ser subdivididas:

- Utilizaremos *toMoney* y *toPercentage* para transformar variables numéricas en variables asociadas a dinero (de 5.0 a 5.0€*) y porcentaje (*de 0.5 a 50 %), respectivamente. Su propósito será estético, ya que únicamente cambiará cómo el usuario ve esa información.

Figura 39: Pantalla de autentificación para acceder a la aplicación.

- *loadHeatmap*, función que utilizaremos para cargar el mapa de calor, y actualizarlo cada vez que seleccionamos una variable diferente sobre la que basarnos a la hora de colorear los distritos.
- *lang* y *credentials* son funciones asociadas con la pantalla de autentificación del usuario. En *lang* modificaremos el texto que verá el usuario al abrir la aplicación, *credentials* establece el usuario y contraseña necesarios para poder acceder a la aplicación.

4.5.3. Carga y transformación de los datos

En esta etapa de la aplicación nos encargamos de cargar todos los datos de los que vamos a hacer uso, así como transformarlos para poder explotarlos de la manera que deseamos. Cabe señalar que la transformación aquí es mínima, y normalmente consiste en añadir variables que necesitan de agrupaciones dinámicas, etc. La parte más extensa de transformación de datos es delegada, como ya bien sabemos, a la *ETL Pipeline*.

Una vez los datos ya han sido cargados y transformados, estamos listos para empezar a desarrollar los componentes principales de la aplicación.

4.5.4. Desarrollo de la interfaz de usuario

El desarrollo de la interfaz de usuario consiste en diseñar cómo el usuario va a comunicarse con el producto que quieres desarrollar, en nuestro caso la aplicación.

Hemos optado por desarrollar una interfaz sencilla, con los filtros disponibles en una columna desplegable a la izquierda, dejando en la pantalla principal únicamente los gráficos interactivos. Para ello, hacemos uso de la librería *shinydashboard* y sus funciones *dashboardPage* (aplicación en sí), *dashboardHeader* (aquí introduciremos el logo de **homyspace**), *dashboardBody* (en esta función introduciremos los gráficos) y *dashboardSidebar* (aquí insertaremos los filtros).

```
title <- tags$a(href="homyspace.com",
                tags$img(src="http://homybrain.com/img/logo.png", width = "200"))

ui <- secure_app(
  theme = shinythemes::shinytheme("flatly"),
  dashboardPage(
    title = "homyspace",
    dashboardHeader(title = title),
    dashboardSidebar(
    ),
    dashboardBody(
      tags$head(tags$style(HTML('
        .skin-blue .main-header .logo {
          font-family: "Nexa Bold";
          font-size: 24px;
          color: #000000;
          background-color: #eeaf30;
        }

        /* navbar (rest of the header) */
        .skin-blue .main-header .navbar {
        background-color: #eeaf30;
        color: #000000;
        }
      '))),
    )

  )
)
```

Con una estructura de la *ui* o *User Interface* (interfaz de usuario en inglés) como esta, logramos esta plantilla:

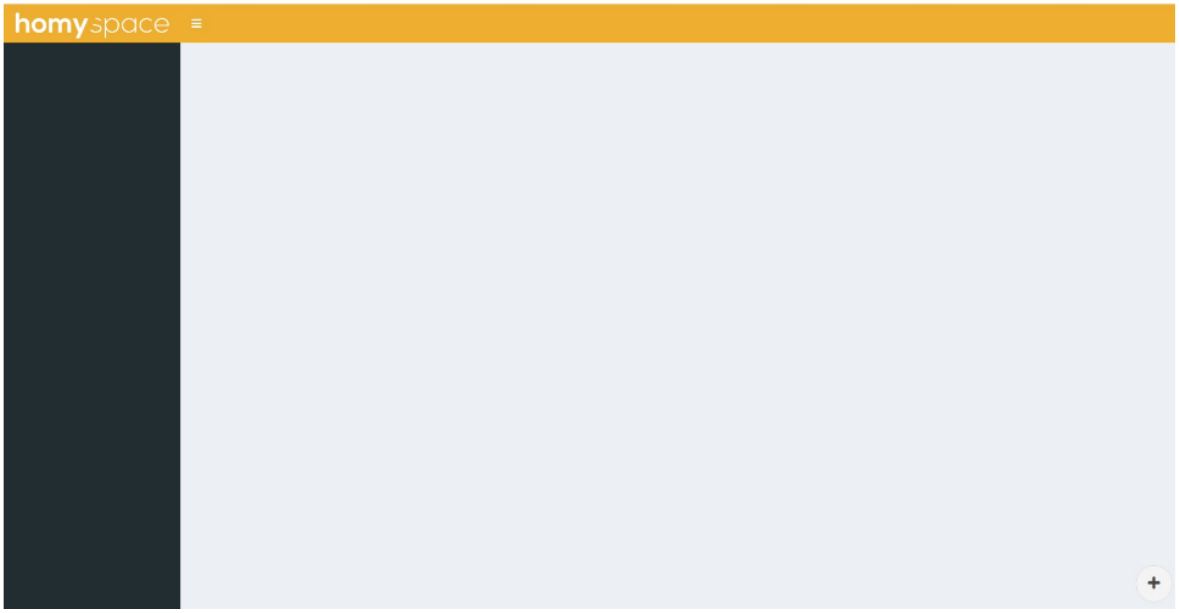

Figura 40: Plantilla de las aplicaciones shiny utilizadas en homyspace.

Una vez establecida la plantilla, iremos rellenando cada una de sus partes con los componentes que deseemos.

4.5.5. Desarrollo de los gráficos y de la reactividad

Debido al carácter técnico de esta sección nos vamos a centrar no tanto en el desarrollo en sí, sino en diferentes consejos y pautas a seguir a la hora de realizar una aplicación en *Shiny*.

- **Seguimos una metodología basada en prueba y error.** Cada vez que desarrollamos un nuevo componente, lo testeamos inmediatamente.
- Del mismo modo, cuando creamos un sistema de reactividad entre diferentes componentes de la aplicación, también lo probamos antes de proseguir con el desarrollo del resto de componentes. En este caso, **nos planteamos qué escenarios pueden hacer que la aplicación se caiga y los validamos, así como que la reactividad sólo se da entre los componentes que esperamos y no se producen efectos cascada inesperados.**
- **Establecemos nombres claros e inteligibles a cada uno de los componentes de la aplicación**, tanto en la *ui* como en el *server*. Esto nos ayuda a comprender mejor el código, tanto a nosotros mismos como a posibles compañeros que quieran modificarlo o, simplemente, entenderlo mejor.

Para aquellos interesados en profundizar más en el desarrollo de un servidor de *Shiny* para una aplicación utilizada en el mundo real, el código se encuentra en la sección *7.2 Desarrollo de la aplicación en Shiny*.

4.5.6. Publicación de la aplicación

Una vez desarrollada la aplicación, es necesario publicarla en la web para que esté disponible para el equipo. *RStudio* nos proporciona un servicio gestionado de calidad llamado *shinyapps.io* [8] donde alojaremos la aplicación.

Para ello nos registraremos en *shinyapps.io* y nos identificaremos a través de *RStudio*.

Tras haber realizado este paso, ya podemos afirmar que nuestra aplicación está lista para ser lanzada. No obstante, esto es **sólo el principio**. Una vez ha sido lanzada, tendremos que seguir los pasos indicados en la sección *3.4.1 Métodos para la difusión del uso de la aplicación*.

Del mismo modo, **raramente la primera versión de una aplicación es la definitiva.** A través del *feedback* por parte del usuario objetivo iremos mejorando el contenido de la aplicación, para que cada vez esté más cerca del valor máximo que los gráficos interactivos pueden aportar para la mejora de la calidad de la toma de decisiones.

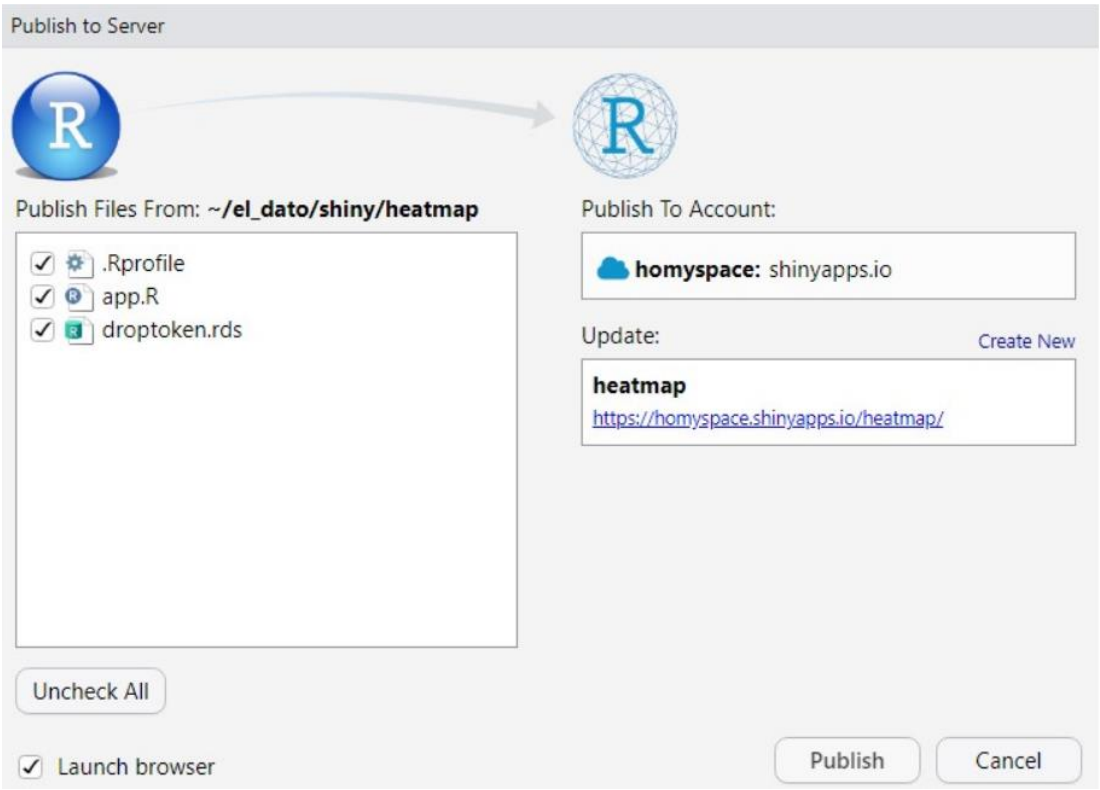

Figura 41: Pantalla que vemos en el proceso de publicación de la aplicación.

4.6. Aspecto Final

Como conclusión, mostramos una demo de la aplicación que hemos desarrollado para este proyecto. Para mantener la privacidad de **homyspace**, hemos eliminado los ejes x e y de todos los gráficos.

[8] shinyapps.io

5. Conclusión

El desarrollo de esta aplicación ha sido una tarea compleja tanto desde el punto de vista técnico, como de diseño y de investigación. No obstante, la inversión de recursos en la misma ha resultado un éxito, ya que hemos mejorado la calidad de la toma de decisiones de **homyspace** y facilitado la implantación de aplicaciones basadas en datos, impactado en la cultura de la empresa de manera positiva.

Por lo tanto, **concluimos este trabajo de fin de grado afirmando que se ha alcanzado tanto el objetivo general como los específicos**. Este proyecto sirve no sólo para convencer al lector de la utilidad de los gráficos interactivos, sino como marco de trabajo para el desarrollo de aplicaciones de gráficos interactivos de manera autónoma, ya sea para sí mismo o para la empresa en que trabaja.

5.1. Aprendizajes

- **Hemos mostrado el uso de gráficos interactivos para la ayuda a la toma de decisiones en homyspace**, informándonos de en qué situaciones la inversión al desarrollo de aplicaciones basadas en torno a ellos tiene un retorno positivo.
- **Hemos aprendido a explotar la librería *shiny* para permitir extraer de los datos la información relevante, filtrando la información y recibiendo respuestas de manera inmediata**.
- **Hemos ganado capacidad y comprensión tecnológica, y hemos facilitado la implementación de herramientas basadas en datos en el futuro.** Al mismo tiempo, hemos abierto las puertas a nuevas áreas de negocio. Algo esencial en una *start-up*.
- **Hemos transmitido la importancia del desarrollo de *ETL Pipelines* que faciliten la explotación de datos**.
- **Hemos aumentado la calidad de la toma de decisiones en homyspace** gracias no sólo a la reducción de las barreras de entrada a la hora de acceder a la información, sino que también hemos implementado el uso de aplicaciones de gráficos interactivos en el proceso de toma de decisiones de otros departamentos de la empresa, **teniendo un impacto positivo en la cultura de la misma.**
- **Hemos obtenido la capacidad de redactar documentos con la librería *bookdown*, con la cual está redactado este proyecto.** Esto nos permite automatizar la redacción de informes para poder optimizar el tiempo realizando el análisis.
- **Hemos aprendido a desenvolvernos en un campo desconocido**. Al principio de este proyecto, no sabíamos siquiera utilizar *Shiny*. A través de una metodología basada en prueba y error, hemos acabado manejando diferentes tecnologías y coordinándolas para construir una herramienta que aporta valor a usuarios en diferentes departamentos de la empresa.

5.2. A mejorar

- Añadir a la aplicación la posibilidad de insertar un fichero con la información relacionada con los inmuebles gestionados y que esta se refleje en el mapa de calor, **para poder localizar los inmuebles instantáneamente por distritos y poder analizar a qué precios deberían ofertarse.**
- Añadir a la aplicación precios de inmuebles de otras plataformas, **para poder saber si estamos siendo competitivos en comparación tanto con nuestros clientes como con nuestros pro- veedores.**
- Añadir leyenda al mapa de calor que nos permita percibir **qué número exacto representa el rango de colores**.
- Mejorar el código de la *ETL Pipeline* para que sea mantenible a largo plazo.
- Aprovechar *Amazon Web Services* para ejecutar la *ETL Pipeline* en la nube de manera automática. Esto nos permitirá actualizar las fuentes de datos durante horario no lectivo sin necesidad de que haya un ordenador encendido que haga de servidor.

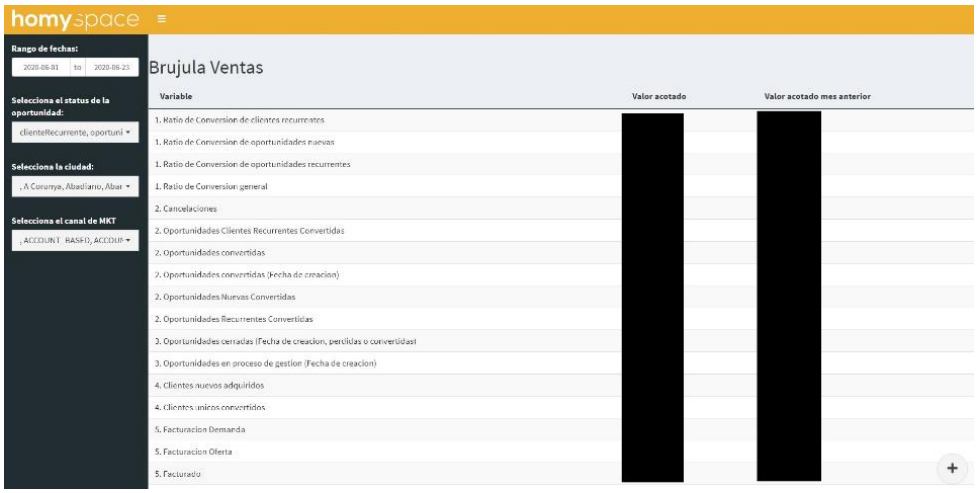

Figura 42: Otras aplicaciones de Shiny: Dashboard que muestra las métricas más importantes de cada departamento.

Figura 43: Otras aplicaciones de Shiny: Dashboard que nos ayuda a analizar el comportamiento de las propuestas recomendadas a través de métodos de aprendizaje automático.

6. Código

Esta sección contiene el código que hemos utilizado tanto para desarrollar la *ETL Pipeline* como la aplicación en *Shiny*.

6.1. Desarrollo de la ETL Pipeline

```
import os
import numpy as np
import geopandas as gpd
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
from matplotlib.colors import ListedColormap
import earthpy as et
distritos = gpd.read_file("distritos.shp", SHAPE_RESTORE_SHX = "YES") # distritos.shp es el
# documento descargado del INE
distritos_df = distritos.dissolve(by="CUDIS") # disolvemos por distrito
distritos_df = distritos_df.reset_index()
#Guardamos el archivo diferenciado por distritos a Shapefile
distritos_df.to_file("coordenadasDistritos.shp")
```

```
from elasticsearch import Elasticsearch, RequestsHttpConnection
from elasticsearch_dsl import Search, Q
from requests_aws4auth import AWS4Auth
import pandas as pd

def getData(index, fields, query=None, endpoint = endpoint):
    # index es el índice al que vamos a llamar.
    # fields, los campos que queremos extraer.
    # query es el filtro que podemos aplicar. Por defecto, no aplicamos filtro.
    # endpoint es la api a través de la cual nos conectamos a Elasticsearch.
    access_key = accessKey
    secret_key = secretKey
    auth=AWS4Auth(access_key, secret_key, region, service)
    # En Homyspace, Elasticsearch se utiliza como servicio gestionado a través
    # de Amazon Web Services, de ahí que necesitemos una autentificación adaptada.
    # Region es la región que vamos a utilizar de AWS, y service el servicio,
    # en este caso Elasticsearch.

    es = Elasticsearch(
        hosts = [{"host": endpoint, "port": port}],
        http_auth=auth,
        use_ssl=True,
        verify_certs=True,
        connection_class=RequestsHttpConnection
    )
```

```
    if not  query:
        indexQuery = Search(using=es, index=index).source(includes=fields)
        data = pd.DataFrame((d.to_dict() for d in indexQuery.scan()))
    else:
        indexQuery  =  Search(using=es,  index=index).source(includes=fields)\
        .query("bool", filter = query)
        data = pd.DataFrame((d.to_dict() for d in indexQuery.scan()))

    # Por último, transforma la respuesta en una matriz de datos que pueda ser utilizada
    # en Python, y la devuelve
    return data
```

```
from elasticsearch import Elasticsearch, RequestsHttpConnection
from elasticsearch_dsl import Search, Q
from requests_aws4auth import AWS4Auth
import dropbox
from dropbox.files import WriteMode
from dropbox.exceptions import ApiError, AuthError
import requests
import zipfile, io
from zipfile import ZipFile
import pandas as pd
import numpy as np
import geopandas as gpd
from shapely.geometry import Point
```

```
endpoint = endpoint
access_key = access_key
secret_key = secret_key
auth=AWS4Auth(access_key, secret_key, region, "es")

es  =  Elasticsearch(
    hosts = [{"host": endpoint, "port": 443}],
    http_auth=auth,
    use_ssl=True,
    verify_certs=True,
    connection_class=RequestsHttpConnection
)
```

```
def  getPropertiesData():
    s1 = Search(using=es, index="property").source(includes=
    ["address.locality",
    "address.administrativeAreaLevel2",
    "address.country",
    "id",
    "address.geoLocation"]
    )

    propertiesData = pd.DataFrame((d.to_dict() for d  in  s1.scan()))
    propertiesData["country"] = propertiesData.address.apply(lambda  x:  x.get("country")\
    if type(x) is not float else  "")
    propertiesData["province"] = propertiesData.address.apply(lambda x: x.get\
    ("administrativeAreaLevel2") if type(x) is not float else "")
```

```
    propertiesData = propertiesData[propertiesData.country == "España"]
    propertiesData(columns={"id": "propertyId"}, inplace = True)
    propertiesData["lon"] = propertiesData.address.apply(lambda x: x.get("geoLocation")\
    .get("lon") if type(x) is not float else "")
    propertiesData["lat"] = propertiesData.address.apply(lambda x: x.get("geoLocation")\
    .get("lat") if type(x) is not float else "")
    propertiesData["propertyCity"] = propertiesData.address.apply(lambda x:
    x.get("locality")\ if type(x) is not float else "")
    propertiesData = propertiesData.replace("Á", "A").replace("á", "a").replace("À", "A")\
    .replace("à", "a").replace("É", "E").replace("é", "e").replace("È", "E").replace("è",
    "e")\
    .replace("Í", "I").replace("í", "i").replace("Ì", "I").replace("ì", "i").replace("Ó",
    "O")\
    .replace("ó", "o").replace("Ò", "O").replace("ò", "o").replace("Ú","U").replace("ú",
    "u")\
    .replace("Ù", "U").replace("ù", "u").replace("Ñ", "NY").replace("ñ",
    "ny").replace("Ü", "U")\
    .replace("ü", "u")

    propertiesData = propertiesData.drop(["address", "country"], axis=1)
    geometry = propertiesData.apply(lambda x: Point(x.lon, x.lat), axis =
    1) propertiesData = gpd.GeoDataFrame(propertiesData, geometry =
    geometry)

    return propertiesData

def getOpportunitiesData():
    s1 = Search(using=es,
    index="deal*").source(includes=
    ["address.locality",
    "address.administrativeAreaLevel2",
    "rentalRequestStage",
    "address.country",
    "id",
    "address.geoLocation"]
    )
    opportunitiesData = pd.DataFrame((d.to_dict() for d in s1.scan()))
    opportunitiesData["country"] = opportunitiesData.address.apply(lambda x:
    x.get("country")\ if type(x) is not float else "")
    opportunitiesData["province"] = opportunitiesData.address.apply(lambda
    x: x.get\ ("administrativeAreaLevel2") if type(x) is not float else "")
    opportunitiesData = opportunitiesData[opportunitiesData.country == "España"]
    opportunitiesData["lon"] = opportunitiesData.address.apply(lambda x:
    x.get("geoLocation")\
    .get("lon") if type(x) is not float else "")
    opportunitiesData["lat"] = opportunitiesData.address.apply(lambda x: x.get("geoLocation")\
    .get("lat") if type(x) is not float else "")
    opportunitiesData = opportunitiesData.replace("Á", "A").replace("á", "a").replace("À",
    "A")\
    .replace("à", "a").replace("É", "E").replace("é", "e").replace("È", "E").replace("è",
    "e")\
    .replace("Í", "I").replace("í", "i").replace("Ì", "I").replace("ì", "i").replace("Ó",
    "O")\
    .replace("ó", "o").replace("Ò", "O").replace("ò", "o").replace("Ú","U").replace("ú",
```

```
"u")\
.replace("Ù", "U").replace("ù", "u").replace("Ñ", "NY").replace("ñ",
"ny").replace("Ü", "U")\
.replace("ü", "u")

opportunitiesData = opportunitiesData.drop(["address", "country"],
axis=1) geometry = opportunitiesData.apply(lambda x: Point(x.lon,
x.lat), axis = 1) opportunitiesData = gpd.GeoDataFrame(opportunitiesData,
geometry = geometry)

return opportunitiesData
```

```
def getProposalsData():
    statusQuery = Q("match", proposalStatus = "OK")
    acceptedQuery = Q("match", proposalStage =
    "ACCEPTED") rejectedQuery = Q("match",
    proposalStage = "REJECTED")
    completeQuery = statusQuery & (acceptedQuery | rejectedQuery)

    s1 = Search(using=es, index="proposal*").source(includes=
    ["address.locality",
    "address.administrativeAreaLevel
    2", "address.postalCode",
    "dateCheckIn",
    "dateCheckOut",
    "monthlyPriceDossier.amou
    nt", "property.beds",
    "proposalStage",
    "proposalStatus",
    "property.id",
    "creationDate",
    "address.geoLocation"]
    ).query("bool", filter = completeQuery)
    proposalsData = pd.DataFrame((d.to_dict() for d in
    s1.scan())) proposalsData =
    proposalsData[(proposalsData.dateCheckIn > 0) &\
    (proposalsData.dateCheckOut > 0)]
    proposalsData["propertyId"] = proposalsData.property.apply(lambda x:
    x.get("id")\ if type(x) is not float else "")
    proposalsData["propertyCity"] = proposalsData.address.apply(lambda x:
    x.get("locality")\ if type(x) is not float else "")
    proposalsData["province"] = proposalsData.address.apply(lambda x: x.get\
    ("administrativeAreaLevel2") if type(x) is not float else "")
    proposalsData["monthlyPrice"] = proposalsData.monthlyPriceDossier.apply(lambda x:
    x.get\ ("amount") if type(x) is not float else "")
    proposalsData["dateCheckIn"] = pd.to_datetime(proposalsData["dateCheckIn"], unit = "ms")
    proposalsData["dateCheckOut"] = pd.to_datetime(proposalsData["dateCheckOut"], unit = "ms")
    proposalsData["beds"] = proposalsData.property.apply(lambda x: x.get("beds")\
    if type(x) is not float else "")
    proposalsData["duration"] = (proposalsData["dateCheckOut"] -
    proposalsData["dateCheckIn"])\
    .dt.days

    proposalsData =
    proposalsData.drop( ["property",
    "proposalStatus",
    "address",
    "monthlyPriceDossier
    ", "creationDate",
    "dateCheckIn",
    "dateCheckOut"],
    axis=1
    )
    proposalsData = proposalsData.replace("Á", "A").replace("á", "a").replace("À", "A")\
    .replace("à", "a").replace("É", "E").replace("é", "e").replace("È", "E").replace("è",
    "e")\
```

```
    .replace("Í", "I").replace("í", "i").replace("Ì", "I").replace("ì", "i").replace("Ó",
    "O")\
    .replace("ó", "o").replace("Ò", "O").replace("ò", "o").replace("Ú","U").replace("ú",
    "u")\
    .replace("Ù", "U").replace("ù", "u").replace("Ñ", "NY").replace("ñ",
    "ny").replace("Ü", "U")\
    .replace("ü", "u")
    return proposalsData

def
    getDistrictsFromDBX(zip_file_url
    ): zip_file_url = zip_file_url
    r = requests.get(zip_file_url)
    z = zipfile.ZipFile(io.BytesIO(r.content))
    z.extractall()
    distritos =
    gpd.GeoDataFrame.from_file("coordenadasDistritos.shp")
    distritos = distritos.to_crs(epsg = 4326)
    # distritos = distritos.replace("Araba/Álava",
    "Álava")\ # .replace("Balears, Illes", "Illes
    Balears")\
    # .replace("Castellón/Castelló",
    "Castellón")\ # .replace("Coruña, A", "A
    Coruña")\
    # .replace("Rioja, La", "La Rioja")\
    # .replace("Palmas, Las", "Las Palmas")\
    # .replace("Valencia/Valéncia", "Valencia")

    return distritos

def
    getPropertiesFromDBX(zip_file_u
    rl): zip_file_url = zip_file_url
    r = requests.get(zip_file_url)
    z = zipfile.ZipFile(io.BytesIO(r.content))
    z.extractall()
    propertiesGeometry = gpd.GeoDataFrame.from_file("properties.shp")

    return propertiesGeometry

def getOpportunitiesFromDBX(url):
    opportunitiesDistritos = pd.read_csv(url)
    opportunitiesDistritos.drop("Unnamed: 0", axis = 1, inplace = True)

    return opportunitiesDistritos
```

```
def assignDistrictsToProperties(propertiesData):

    propertiesGeometry = getPropertiesFromDBX()
    distritos = getDistrictsFromDBX()
    propertiesData = propertiesData[~propertiesData["propertyId"]\
    .isin(propertiesGeometry["propertyId"])].reset_index()

    properties = propertiesData.copy()
    properties["distrito"] = np.nan
    properties["municipio"] = np.nan
    properties["polygon"] = np.nan
    length = properties.shape[0]
    totales = 0
    sumat = 0
    for i, inmueble in properties.iterrows():
        provincia = inmueble.province
        point = inmueble.geometry
        distritosProvincia = distritos[distritos.NPRO == provincia]
        for j, distrito in distritosProvincia.iterrows():


            geom =
            distrito.geometry cudis
            = distrito.CUDIS
            nmun = distrito.NMUN
            if geom.contains(point):
                properties["distrito"].loc[i] =
                cudis
                properties["municipio"].loc[i] =
                nmun properties["polygon"].loc[i]
                = geom break

        totales += 1
        if totales == 100:
            sumat +=
            totales totales =
            0
            print(str(sumat) + " completados de " + str(length))

    properties.drop(columns=["geometry", "index"], inplace = True)
    properties.rename(columns={"polygon": "geometry"}, inplace = True)
    properties = gpd.GeoDataFrame(properties, geometry =
    properties.geometry) properties = properties[["propertyId", "municipio",
    "distrito", "geometry"]]

    properties = pd.concat([propertiesGeometry, properties], sort =
    True) properties = properties.drop_duplicates(subset =
    "propertyId")

    return properties
```

```
def assignDistrictsToOpportunities(opportunitiesData):
    opportunitiesDistritos = getOpportunitiesFromDBX(url)
    opportunitiesData =
    opportunitiesData[~opportunitiesData["id"]\
    .isin(opportunitiesDistritos["id"])].reset_index()
    opportunitiesData.drop(columns=["index"], inplace=True)
    distritos =
    gpd.GeoDataFrame.from_file("coordenadasDistritos.shp")
    distritos = distritos.to_crs(epsg = 4326)
    # distritos = distritos.replace("Araba/Álava",
    "Álava")\ #.replace('Balears, Illes', "Illes
    Balears")\
    #  .replace('Castellón/Castelló',
    "Castellón")\ #.replace('Coruña, A', "A
    Coruña")\
    # .replace('Rioja, La', "La Rioja")\
    #.replace('Palmas, Las', "Las Palmas")\
    #.replace('Valencia/Valéncia', "Valencia")

    opportunities =
    opportunitiesData.copy()
    opportunities["distrito"] = np.nan
    opportunities["municipio"] = np.nan
    length = opportunities.shape[0]
    totales = 0
    sumat = 0
    for i, opportunity in opportunities.iterrows():
        provincia = opportunity.province
        point = opportunity.geometry
        distritosProvincia = distritos[distritos.NPRO == provincia]
        for j, distrito in
            distritosProvincia.iterrows(): geom =
            distrito.geometry


def groupByDistricts(proposalsData):
    opportunities =
    getOpportunitiesFromDBX()

    pricePerRooms = proposalsData.dropna(subset=["monthlyPrice"]).groupby(["distrito",
    "rooms"])\ ["monthlyPrice"].median().reset_index()\
    .pivot(index = "distrito", columns = "rooms", values = "monthlyPrice").reset_index()

    proposalsNumber = proposalsData.groupby("distrito")["proposalStage"].count().reset_index()\
    .sort_values(by="proposalStage", ascending = False)\
    .rename(columns = {"proposalStage": "proposalsNumber"})

    propertiesProposedPerDistrict = proposalsData.groupby("distrito")["propertyId"]\
    .nunique().reset_index().rename(columns = {"propertyId": "numberPropertiesProposed"})

    propertiesAcceptedPerDistrict = proposalsData[proposalsData.proposalStage == "ACCEPTED"]\
    .groupby("distrito")["propertyId"].nunique().reset_index()\
    .rename(columns = {"propertyId": "numberPropertiesAccepted"})

    propertiesPerDistrict = properties[["distrito", "propertyId"]]\
```

```
.groupby("distrito").count().reset_index().rename(columns={"propertyId":
"numberProperties"})

acceptedProposalsNumber = proposalsData[proposalsData.proposalStage == "ACCEPTED"]\
.groupby("distrito")["proposalStage"].count().reset_index()\
.sort_values(by="proposalStage", ascending = False)\
.rename(columns = {"proposalStage": "acceptedProposalsNumber"})

durationPerDistrict = proposalsData.groupby("distrito")["duration"].median().reset_index()

opportunitiesPerDistrict = opportunities.groupby("distrito")["id"].count().reset_index()\
.rename(columns={"id": "numOpportunities"})\
.sort_values(ascending = False, by = "numOpportunities")

bookingsPerDistrict = opportunities\
[(opportunities.rentalRequestStage == "S030_CONTRACT_INVOICE_MANAGMENT")\
| (opportunities.rentalRequestStage =="S040_CHECK_IN") | \
(opportunities.rentalRequestStage ==
"S050_RENTAL_CONFIRMED") | \
(opportunities.rentalRequestStage
=="S060_CONVERSION_HISTORY")]\
.groupby("distrito")["id"].count().reset_index().rename(columns={"id": "numBookings"})\
.sort_values(ascending = False, by = "numBookings")

distritos = proposalsData[["distrito", "geometry", "municipio", "propertyCity"]]\
.drop_duplicates(subset = "distrito")

heatmap_data = proposalsNumber.merge(acceptedProposalsNumber, how = "left", on
="distrito")\
.merge(propertiesPerDistrict, how = "left", on = "distrito")\
.merge(propertiesProposedPerDistrict, how = "left", on = "distrito")\
.merge(propertiesAcceptedPerDistrict, how = "left", on = "distrito")\
.merge(durationPerDistrict, how = "left", on = "distrito")\
.merge(pricePerRooms, how = "left", on = "distrito")\
.merge(opportunitiesPerDistrict, how = "left", on = "distrito")\
.merge(bookingsPerDistrict, how = "left", on = "distrito")\
.merge(distritos, how = "left", on = "distrito")\

heatmap_data = gpd.GeoDataFrame(heatmap_data, geometry = heatmap_data.geometry)
heatmap_data["conversionRatio"] = heatmap_data["numBookings"]\
/heatmap_data["numOpportunities"]
heatmap_data["benchmark"] = heatmap_data["acceptedProposalsNumber"]\
/heatmap_data["proposalsNumber"]
heatmap_data["propertiesUsed"] = heatmap_data["numberPropertiesProposed"]\
/heatmap_data["numberProperties"]
heatmap_data["propertiesConverted"] = heatmap_data["numberPropertiesAccepted"]\
/heatmap_data["numberPropertiesProposed"]

heatmap_data["distrito"] = heatmap_data.distrito.astype(str)

return heatmap_data
```

```
# Access token
TOKEN = TOKEN

# Uploads contents of LOCALFILE to Dropbox
def backup():
    with open(LOCALFILE, 'rb') as f:
        print("Uploading " + LOCALFILE + " to Dropbox as " + BACKUPPATH + "...")
        try:
            dbx = dropbox.Dropbox(TOKEN)
            dbx.files_upload(f.read(), BACKUPPATH, mode=WriteMode('overwrite'))
        except ApiError as err:
            if err.user_message_text:
                print(err.user_message_text)
                sys.exit()
            else:
                print(err)
                sys.exit()


# Adding few functions to check file details
def checkFileDetails():
    print("Checking file details")

    for entry in dbx.files_list_folder('').entries:
        print("File list is : ")
        print(entry.name)


# Run this script independently
def uploadDropbox(TOKEN, LOCALFILE, BACKUPPATH):
  if __name__ == '_main_':
    # Check for an access token
    if (len(TOKEN) == 0):
        sys.exit("ERROR: Looks like you didn't add your access token. \
        Open up backup-and-restore-example.py in a text editor and paste in your token in      ine 14.")

    # Create an instance of a Dropbox class, which can make requests to the API.
    print("Creating a Dropbox object...")
    dbx = dropbox.Dropbox(TOKEN)
    print("Creating  backup...")
    # Create a backup of the current settings file
    backup()
    print("Done!")
```

```
def savePropertiesToDBX(properties):
    properties.to_file("properties.shp")

    filename = "properties.zip"
    with ZipFile(filename, "w") as zip:
        for file in ["properties.shp", "properties.cpg", "properties.dbf", "properties.shx"]:
            zip.write(file)

    LOCALFILE = "properties.zip"
    BACKUPPATH = "/properties.zip"
    uploadDropbox(TOKEN, LOCALFILE, BACKUPPATH)

def saveOpportunitiesToDBX(opportunities):
    opportunities.to_csv("opportunities.csv")
    LOCALFILE  =  "opportunities.csv"
    BACKUPPATH = "/opportunities.csv"
    uploadDropbox(TOKEN, LOCALFILE, BACKUPPATH)

def saveHeatmapToDBX(heatmap_data):
    heatmap_data.to_file("heatmap.shp")

    filename = "heatmap.zip"
    with ZipFile(filename, "w") as zip:
        for file in ["heatmap.shp", "heatmap.cpg", "heatmap.dbf", "heatmap.shx"]:
            zip.write(file)

    LOCALFILE = "heatmap.zip"
    BACKUPPATH = "/heatmap.zip"
    uploadDropbox(TOKEN, LOCALFILE, BACKUPPATH)

def saveProposalsToDBX(proposalsData):
    proposalsData.drop(columns=["geometry"], inplace = True)
    proposalsData["distrito"] = proposalsData.distrito.astype(str)

    proposalsData.to_csv("proposalsData.csv")
    LOCALFILE = "proposalsData.csv"
    BACKUPPATH = "/proposalsData.csv"
    uploadDropbox(TOKEN, LOCALFILE, BACKUPPATH)
```

6.2. Desarrollo de la aplicación en Shiny

```
#####
##### Carga de librerías

# Relacionados con la visualización (y transformación) de datos
library(tidyverse)
library(plotly)
library(leaflet)
library(leaflet.extras)
library(DT)

# Relacionados con Shiny
library(shiny)
library(shinymanager)
library(shinyWidgets)
library(shinydashboard)
library(shinycssloaders)

# Relacionados con la carga (y transformación) de datos
library(rdrop2)
library(sf)
library(lubridate)


#####
##### Creación de funciones

# Funciones y variables para la pantalla de acceso
lang <- shinymanager:::language$new()
lang$add(
  "Please authenticate" = "Bienvenid@ al mapa de calor!"
)

credentials <- data.frame(
```

```
  user = "",
  password =
  "",
  stringsAsFactors = FALSE
)


# Funciones para la transformación de datos
toPercentage <- function(value) {
  return(paste0(round(100*value,2),' %'))
}

toMoney <- function(value) {
  return(paste0(round(value,2),' euro'))
}

# Función para crear el mapa de calor, y alterar el rango de colores en función del campo
seleccionado
loadHeatmap <- function(mapData, campoColor) {
  if (campoColor == "Número de Oportunidades")
  {
    colorSeleccionado = "numOpportunities"
    bins <- c(0, 5, 10, 15, 20, 25, 50, 75, 100, Inf)
  }


  else if (campoColor == "Número de Inmuebles")
  {
    colorSeleccionado = "numberProperties"
    bins <- c(0, 5, 10, 15, 20, 25, 50, 75, 100, Inf)
  }


  else if (campoColor == "Número de Propuestas Aceptadas")
  {
    colorSeleccionado = "acceptedProposals"
    bins <- c(0, 5, 10, 15, 20, 25, 50, 75, 100, Inf)
  }
  pal <- colorBin("YlOrRd", domain = mapData[[colorSeleccionado]], bins = bins)

  mapData %>%
    leaflet() %>%
    setView(lng = -3.6, lat = 40.45, zoom =
    5) %>% addTiles()
    %>%
    addPolygons(
      layerId = ~distrito,
      label = ~distrito,
      popup =
      ~relevantInfo,
      fillColor =
      ~pal(mapData[[colorSeleccionado]]), color
      = "#444444",
      weight = 1,
```

```
smoothFactor = 0.5,
opacity = 1.0,
fillOpacity = 0.5,
highlightOptions = highlightOptions(color = "white",
                                    weight = 2,
                                    bringToFront =
                                    TRUE)
```

```
    ) %>%
    #  addTiles()  %>%
    # addAwesomeMarkers(
    #    data = inmuebles,
    #    lng = ~lon,
    #    lat = ~lat,
    #    label = ~BX.ID,
    #    layerId =  ~distrito,
    #    popup =~relevantInfo
    # ) %>%
    addSearchOSM(options = searchOptions(autoCollapse = TRUE, minLength = 2))
}

#####
##### Carga y transformación de los datos

# Carga de la shapefile con los distritos
token <- readRDS("droptoken.rds")
drop_download("/heatmap.zip", dtoken = token, overwrite
= T) unzip("heatmap.zip")
polyData <- read_sf(dsn = path.expand("heatmap.shp"),
layer="heatmap") polyData <- polyData %>% rename(
  "proposalsNumber" = "proposalsN",
  "acceptedProposals" = "acceptedPr",
  "numberProperties" = "numberProp",
  "0_rooms" = "X0.0",
  "1_rooms" = "X1.0",
  "2_rooms" = "X2.0",
  "3_rooms" = "X3.0",
  "4_rooms" = "X4.0",
  "more4_rooms" = "X.4",
  "propertyCity" = "propertyCi",
  "numberPropertiesProposed" =
  "numberPr_1", "numberPropertiesConverted"
  = "numberPr_2",
  "percentagePropertiesProposed" =
  "properties", "conversionRatioProperties" =
  "properti_1", "numOpportunities" =
  "numOpportu", "numBookings" =
  "numBooking", "conversionRatio" =
  "conversion"
)

polyData$propertyCity <-
factor(polyData$propertyCity) polyData$municipio
<- factor(polyData$municipio) polyData$distrito
<- factor(polyData$distrito)
polyData$`12_rooms` <- (polyData$`1_rooms` +
polyData$`2_rooms`)/2 # Creación del texto que aparecerá
al clickar en el distrito polyData$relevantInfo <-
paste("<strong>Distrito:</strong>",
                                  polyData$distrito, "<br>",
                                  "<strong>Oportunidades:</strong>",
                                  polyData$numOpportunities, "<br>",
```

```
"<strong>Reservas:</strong>",
polyData$numBookings, "<br>",
"<strong>Ratio de
Conversión:</strong>",
```

```
                                    toPercentage(polyData$conversionRatio),
                                    "<br>", "<strong>Inmuebles:</strong>",
                                    polyData$numberProperties, "<br>",
                                    "<strong>Inmuebles propuestos:</strong>",
                                    polyData$numberPropertiesProposed,  paste("(",
                                    toPercentage(polyData$ "<strong>Inmuebles reservados:</strong>",
                                    polyData$numberPropertiesConverted,  paste("(",
                                    toPercentage(polyData$ "<strong>Propuestas Hechas:</strong>",
                                    polyData$proposalsNumber, "<br>",
                                    "<strong>Propuestas Aceptadas:</strong>",
                                    polyData$acceptedProposals, "<br>",
                                    "<strong>Benchmark:</strong>",
                                    toPercentage(polyData$benchmark), "<br>",
                                    "<strong>Precio Medio (1 y 2
                                    habitaciones):</strong>",
                                    toMoney(polyData$`12_rooms`),  "<br>",
                                    "<strong>Estancia  Media:</strong>",
                                    polyData$duration)

# Carga de los csv de los que se alimentarán los gráficos
heatmapData  <- drop_read_csv("/heatmapData.csv",  dtoken  =
token) heatmapData$distrito <- factor(heatmapData$distrito)
heatmapData$rooms <- factor(heatmapData$rooms)
heatmapData$beds <-
factor(as.integer(heatmapData$beds))
heatmapData$duration <-
round(heatmapData$duration)

properties  <- drop_read_csv("/properties.csv",  dtoken  =
token) properties$distrito <- factor(properties$distrito)
properties$distrito  <- gsub("x",  "",
properties$distrito) properties$distrito  <- gsub("None",
NA,  properties$distrito) properties$distrito <-
factor(properties$distrito) properties$rooms <-
factor(properties$rooms) properties$hasICal <-
as.logical(properties$hasICal) properties$municipio <-
factor(properties$municipio)
%%

# Carga de información de inversores, si tuviéramos

inmuebles  <-  read.csv("inmuebles.csv",
encoding="UTF-8") inmuebles <- inmuebles  %>%
rename(
  "proposalsNumber" = "proposalsN",
  "acceptedProposals" = "acceptedPr",
  "numberProperties"  =  "numberProp",
  "0_rooms" =  "X0.0",
  "1_rooms"  =  "X1.0",
  "2_rooms"  =  "X2.0",
  "3_rooms"  =  "X3.0",
  "4_rooms" = "X4.0",
  "more4_rooms" = "X.4",
  "propertyCity" = "propertyCi",
  "numberPropertiesProposed"  =  "numberPr_1",
```

```
"numberPropertiesConverted"  =  "numberPr_2",
"percentagePropertiesProposed" = "properties",
"conversionRatioProperties" = "properti_1",
"numOpportunities"  =  "numOpportu",
```

```
  "numBookings"  =  "numBooking",
  "conversionRatio" = "conversion"
)
inmuebles$lon <- as.numeric(inmuebles$lon)
inmuebles$lat <- as.numeric(inmuebles$lat)
inmuebles$`1_rooms` <-
as.numeric(inmuebles$`1_rooms`)
inmuebles$`2_rooms` <-
as.numeric(inmuebles$`2_rooms`)
inmuebles$`12_rooms` <- (inmuebles$`1_rooms` +
inmuebles$`2_rooms`)/2 inmuebles$relevantInfo <- paste(
                                    "<strong>ID:</strong>",
                                    inmuebles$BX.ID,
                                    "<br>",
                                    "<strong>Inmuebles 0
                                    habitaciones:</strong>",
                                    inmuebles$X0D.Units, "<br>",
                                    "<strong>Inmuebles 1
                                    habitaciones:</strong>",
                                    inmuebles$X1D.Units, "<br>",
                                    "<strong>Inmuebles 2
                                    habitaciones:</strong>",
                                    inmuebles$X2D.Units, "<br>",
                                    "<strong>Inmuebles 3
                                    habitaciones:</strong>",
                                    inmuebles$X3D.Units, "<br>",
                                    "<strong>Inmuebles 4
                                    habitaciones:</strong>",
                                    inmuebles$X4D.Units, "<br>",
                                    "<strong>Inmuebles 5
                                    habitaciones:</strong>",
                                    inmuebles$X5D.Units, "<br>",
                                    "<strong>Capex:</strong>",
                                    inmuebles$capex, "<br>", "<hr>",
                                    "<strong>Distrito:</strong>",
                                    inmuebles$distrito, "<br>",
                                    "<strong>Oportunidades:</strong>",
                                    inmuebles$numOpportunities, "<br>",
                                    "<strong>Reservas:</strong>",
                                    inmuebles$numBookings, "<br>",
                                    "<strong>Ratio de
                                    Conversión:</strong>",
                                    toPercentage(inmuebles$conversionRatio),
                                    "<br>", "<strong>Inmuebles:</strong>",
                                    inmuebles$numberProperties, "<br>",
                                    "<strong>Inmuebles propuestos:</strong>",
                                    inmuebles$numberPropertiesProposed,  paste("(",
                                    toPercentage(inmuebles$ "<strong>Inmuebles reservados:</strong>",
                                    inmuebles$numberPropertiesConverted,  paste("(",
                                    toPercentage(inmuebles$ "<strong>Propuestas Hechas:</strong>",
                                    inmuebles$proposalsNumber, "<br>",
                                    "<strong>Propuestas
                                    Aceptadas:</strong>",
                                    inmuebles$acceptedProposals, "<br>",
```

```
                    "<strong>Benchmark:</strong>",
                    toPercentage(inmuebles$benchmark),
                    "<br>",
                    "<strong>Precio Medio (1 y 2
                    habitaciones):</strong>",
                    toMoney(inmuebles$`12_rooms`), "<br>",
                    "<strong>Estancia  Media:</strong>",
                    inmuebles$duration)
```

```
#####
##### Diseño de la interfaz de usuario
```

```
title <- tags$a(href="homyspace.com",
              tags$img(src="http://homybrain.com/img/logo.png", width = "200"))

ui <-
  secure_app(
    theme = shinythemes::shinytheme("flatly"),
    dashboardPage(
      title = "homyspace",
      dashboardHeader(title =
      title), dashboardSidebar(
        collapsed = T,
        pickerInput(inputId = "propertyCity", label = "Filtra por ciudad", choices =
        levels(polyData$
          `actions-box` =
          TRUE, size = 10,
          `live-search` = TRUE
        )),
        pickerInput(inputId = "municipio", label = "Filtra por municipio", choices =
          levels(polyData$mu `actions-box` = TRUE,
          size = 10,
          `live-search` = TRUE
        )),
        pickerInput(inputId = "distrito", label = "Filtra por distrito", choices =
          levels(polyData$dist `actions-box` = TRUE,
          size = 10,
          `live-search` = TRUE
        )),
        pickerInput(inputId = "beds", label = "Filtra por número de camas", choices =
          levels(heatmapDat `actions-box` = TRUE,
          size = 10,
          `live-search` = TRUE
        )),
        pickerInput(inputId = "rooms", label = "Filtra por número de habitaciones", choices =
          levels(he `actions-box` = TRUE,
          size = 10,
          `live-search` = TRUE
        ))
      ),
      dashboardBody(
        tags$head(tags$style(HTM
        L('
      .skin-blue .main-header
        .logo { font-family: "Nexa
        Bold"; font-size: 24px;
        color: #000000;
        background-color: #eeaf30;
      }

      /* navbar (rest of the header) */
      .skin-blue .main-header
      .navbar { background-color:
      #eeaf30;
      color: #000000;
      }
```

’))
),

```
        fluidRow(
          h3("Selecciona en función de qué campo colorear el mapa de calor")
        ),
        fluidRow(
          pickerInput(inputId = "color", choices = c("Número de Oportunidades", "Número de Inmuebles",
            `actions-box` = TRUE,
            size = 10,
            `live-search` = TRUE
          ))
        ),
        fluidRow(
          leafletOutput("heatmap") %>%
            withSpinner(color="#EEAF30")
        ),
        fluidRow(
          column(6, plotlyOutput("pricesLine") %>%
                   withSpinner(color="#EEAF30")),
          column(6, plotlyOutput("roomsBar") %>%
                   withSpinner(color="#EEAF30"))
        ),
        fluidRow(
          column(6, plotlyOutput("roomsPerDistrict") %>%
                   withSpinner(color="#EEAF30")),
          column(6, plotlyOutput("hasIcal") %>%
                   withSpinner(color="#EEAF30"))
        ),
        h3("Resumen por distrito"),
        fluidRow(
          dataTableOutput("dataDistrito") %>%
            withSpinner(color="#EEAF30")
        )
      )
    )
  )

#####
##### Desarrollo de los gráficos y de la reactividad (servidor)

server <- function(input, output, session) {
  result_auth <- secure_server(check_credentials = check_credentials(credentials))

  # observe({
  #   event <- input$propertyCity
  #   data <- subset(polyData, propertyCity %in% event)
  #   updateSelectInput(session = session, inputId =
  "municipio", #         selected =
  levels(data$municipio))
  # })

  # reactiveInputData <- reactive({                                    %>%
  #   subset(polyData, propertyCity                                    in input$propertyCity &
  municipio                          in input$municipio & distrito# })
  observe({
    event <- input$heatmap_shape_click
```

```
  updateSelectInput(session, inputId = "distrito", selected = event$id)
  updateSelectInput(session, inputId = "propertyCity", selected =
  levels(polyData$propertyCity)) updateSelectInput(session, inputId = "municipio",
  selected = levels(polyData$municipio))
})

observe({
  event <- input$heatmap_marker_click
  updateSelectInput(session, inputId = "distrito", selected = event$id)
  updateSelectInput(session, inputId = "propertyCity", selected =
  levels(polyData$propertyCity)) updateSelectInput(session, inputId = "municipio",
  selected = levels(polyData$municipio))
})

output$heatmap <- renderLeaflet({
  loadHeatmap(polyData,
  input$color)
})
reactiveDistrictData <-
  reactive({ heatmapData
    filter(propertyCity in input$propertyCity & municipio
    input$municipio & distrito
    mutate(monthlyPrice = round(monthlyPrice, 0), duration = round(duration, 0))
})

reactivePropertiesData <-
  reactive({ properties
    filter(distrito in input$distrito, !is.na(hasICal))
    # # group_by(rooms)
    # summarise(numberOfProperties = n(), withIcal = sum(hasICal))
})

output$pricesLine <-
  renderPlotly({ data <-
  reactiveDistrictData()
    filter(!is.na(monthlyPrice) & !is.na(rooms) & rooms != "nan")

  data$rooms <- factor(data$rooms,
                       levels = c("0.0", "1.0", "2.0", "3.0", "4.0", ">4"), ordered =
                       TRUE)

  p <- ggplot(data = data, aes(y = monthlyPrice, group = rooms, color = rooms, text =
    toMoney(monthly geom_boxplot() +
    scale_y_continuous(limits = c(0, 5000), breaks = seq(0, 5000, 500), name = "Alquiler
    Mensual") + scale_x_discrete(name = "Número de habitaciones") +
    labs(title = "Alquiler de precios de alquiler en función del número de habitaciones", subtitle
    =
    theme_minimal()

  ggplotly(p, tooltip = "y")
    layout(boxmode='group',
```

```
            xaxis = list(autorange = T))
})

output$roomsBar <-
  renderPlotly({ data  <-
  reactiveDistrictData()            >
    filter(!is.na(rooms) & rooms != "nan")

  p <- ggplot(data = data, aes(x = rooms, fill = proposalStage)) +
```

```
    geom_bar() +
    scale_x_discrete(name = "Número de habitaciones") +
    scale_y_continuous(name = "Número de propuestas enviadas") +
    labs(title = "Número de propuestas aceptadas y rechazadas por número de habitaciones",
    subtitle =
    theme_minimal()

  ggplotly(p, tooltip = "y")
})

output$roomsPerDistrict <-
  renderPlotly({ data <-
  reactivePropertiesData()

  p <- ggplot(data = data, aes(x=rooms, fill=rooms)) +
    geom_bar() +
    scale_x_discrete(name = "Número de habitaciones") +
    scale_y_continuous(name = "Número de inmuebles")
    +
    labs(title = "Número de inmuebles por número de habitaciones", subtitle = "Homyspace") +
    theme_minimal()

  ggplotly(p, tooltip = "y")
})

output$hasIcal <- renderPlotly({
  data  <-  reactivePropertiesData()
              %>% mutate(n=n())
              %>%
    group_by(hasICal)%>%
    summarise(porcentaje  =  n()/mean(n))

  p <- ggplot(data = data, aes(x=hasICal, y=porcentaje, fill=hasICal)) +
    geom_bar(stat="identity") +
    scale_x_discrete(name = "Tiene iCal") +
    scale_y_continuous(name = "Porcentaje de
    inmuebles") +
    labs(title = "Porcentaje de inmuebles con y sin iCal", subtitle = "Homyspace") +
    theme_minimal()

  ggplotly(p, tooltip = "y")
})

output$dataDistrito <- renderDataTable({
  districtData <- reactiveDistrictData()

  datatable(districtData  %>%
              group_by(distrito)  %>%
              summarise(numeroPropuestas  =  n(),  numeroReservas  =  sum(proposalStage  ==
                        "ACCEPTED"),  be duracionEstancia = median(duration, na.rm = T),
                        inmuebles = length(unique(propertyId)),
                        inmueblesConvertidos = n_distinct(propertyId[proposalStage ==
                        "ACCEPTED"])
              )  %>%
```

```
    arrange(desc(numeroPropuestas
    )),
filter = "top",
selection="multipl
e", options = list(
    # autoWidth = TRUE,
```

```
                    sDom = '<"top">lrt<"bottom">ip',
                    ordering =  T,
                    scrollX = T
                    # lengthChange =  FALSE,
                    # info = FALSE,
                    # paging = F
                ),
                rownames = F
    )
  })

}
```

```
#####
##### Cargamos la aplicación con la interfaz de usuario (ui) y el servidor (server)

shinyApp(ui = ui, server = server)
```

Referencias

Boeing, Geoff. “Clustering to Reduce Spatial Data Set Size”. Clustering to Reduce Spatial Data Set Size. (2014).

Bustillo, Rubén F. “¿Cómo realizar mapas de España con archivos shapefile?”. ¿Cómo realizar mapas de España con archivos shapefile?. (2019).

Chambers, John M. et al.“Graphical Methods for Data Analysis”.

(1983). Chang, Winston. “R Graphics Cookbook”. (2012).

Correa, Juan Carlos y Nelfi González. “Grafícos Estadísticos con R”. (2002).

Crane, Nic. “Interactive Maps in R with Shiny and Leaflet”. Interactive Maps in R with Shiny and Leaflet. (2016).

Ellis, Laura. “Map Plots Created With R And Ggmap”. Map Plots Created With R And Ggmap.

(2018). Engel, Claudia A. “Using Spatial Data with R”. Using Spatial Data with R. (2019).

Friendly, Michael et al.The First (Known) Statistical Graph: Michael Florent van Langren and the “Secret” of Longitude. The First (Known) Statistical Graph: Michael Florent van Langren and the “Secret” of Longitude. (2010).

Fry, Ben. “Visualizing Data”. (2008).

Garrido, Manuel. “Making a Beautiful Map of Spain in ggplot2”. Making a Beautiful Map of Spain in ggplot2. (2017).

Gregory, Matt. “Shiny interactive map app development”. Shiny interactive map app

development.(2017). Holtz, Yan. “The R Graph Gallery”. The R Graph Gallery.

Li, Angela. “R Spatial Workshop Notes”. R Spatial Workshop Notes. (2019).

Maindonald, John and John Braun. “Data Analysis and Graphics Using R”.

(2006).

Milnes, Jason. “Effective Visualizations for Credible, Data-Driven Decision Making”. Effective Visualizations for Credible, Data-Driven Decision Making. (2020).

Mittal, Hrishi V. “R Graphs Cookbook”. (2011).

Molnar, Christoph. “Interpretable Machine Learning”. Interpretable Machine Learning. (2020)

Moreno, Mel and Mathieu Basille. “Drawing beautiful maps programmatically with R, sf and ggplot2 - Part 1: Basics”. Drawing beautiful maps programmatically with R, sf and ggplot2 — Part 1: Basics. (2018).

Pork Chop. Answer to “r - How to show Spinning Wheel or Busy Icon while waiting in Shiny”. r - How to show Spinning Wheel or Busy Icon while waiting in Shiny. (2018).

r-tastic, “Exploring London Crime with R heat maps”. Exploring London Crime with R heat

maps. (2018). shinyapps.io team. “Shinyapps.io user guide”. shinyapps.io user guide. (2020).

Sievert, Carson. “Interactive web-based data visualization with R, plotly and shiny”. Interactive web-based data visualization with R, plotly, and shiny. (2019)

Soltoff, Benjamin. “Computing for the Social Sciences”. Computing for the Social

Sciences. (2020). Wickham, Hadley. “R for Data Science”. R for Data Science. (2017).

Wyckoff, Joy P. “How to make interactive maps in R Shiny (brief tutorial)”. How to make interactive maps in R Shiny (brief tutorial). (2018).

Xie, Yihui. “Bookdown: Authoring Books and Technical Documents with R Markdown”. Bookdown: Autho- ring Books and Technical Documents with R Markdown. (2020).

Xie, Yihui et al.“R Markdown: The definitive Guide”. R Markdown: The definitive Guide. (2020)