Universidades de Burgos, León y Valladolid

Máster universitario

Inteligencia de Negocio y Big Data en Entornos Seguros

TFM del Máster Inteligencia de Negocio y Big Data en Entornos Seguros

Flujo de Datos para Descubrimiento de Oportunidades Inmobiliarias

Presentado por David Puerta Martos
en Universidad de Burgos a
22 de febrero de 2024
Tutor: Álvar Arnaiz González
Co-tutor: José Antonio Barbero Aparicio

Resumen

En este proyecto se ha desarrollado un proceso de extracción, transformación y carga de datos inmobiliarios, obtenidos desde uno de los portales más destacados de España, pisos.com. Este proceso se complementa con el análisis de los datos mediante técnicas de Aprendizaje Automático, con el objetivo de descubrir potenciales oportunidades de compra de inmuebles.

Como resultado, se ha creado una interfaz web accesible en www.buscahogar.es, que permite a los usuarios filtrar, buscar y ordenar inmuebles disponibles según el valor asignado por los modelos, además de otras características, ofreciendo así una herramienta poderosa para potenciales compradores. También, se ofrece la posibilidad de visualizar datos agregados de todo el mercado inmobiliario, filtrados y segmentados según distintos intereses, con objetivo de mejorar la toma de decisiones.

Finalmente, el proyecto orquesta los procesos de extracción, transformación, carga, entrenamiento de modelos y análisis de datos, con el fin de mantener la información actualizada a diario y relevante para los usuarios.

Descriptores

scraping, python, mongodb, ETL, SQL, airflow, docker, aprendizaje automático, scikit learn, react, node.js

Abstract

In this project, a process of extraction, transformation, and loading of real estate data has been developed, sourced from one of Spain's most prominent portals, pisos.com. This process is complemented by data analysis using Machine Learning techniques, with the aim of uncovering potential real estate buying opportunities.

As a result, a web interface accessible at www.buscahogar.es has been created. Our portal allows users to filter, search, and sort available properties based on the value assigned by the models, among other characteristics, thus offering a powerful tool for potential buyers. Additionally, it provides the possibility to visualize aggregated data from the entire real estate market, filtered and segmented according to different interests, with the goal of improving decision-making.

Finally, the project orchestrates the processes of extraction, transformation, loading, model training, and data analysis, with the aim of keeping the information updated daily and relevant for users.

Keywords

scrapping, python, mongodb, ETL, SQL, airflow, machine learning, regression, scikit learn, react, node.js

Índice general

Índice de figuras

Índice de tablas

Memoria

Introducción

En la era digital actual, el mercado de inversiones inmobiliarias en España está experimentando una transformación sin precedentes, principalmente por una combinación de factores económicos, sociales y tecnológicos. La búsqueda de inmuebles para comprar se ha vuelto cada vez más desafiante en un contexto donde los precios han visto un encarecimiento notable en los últimos años, lo que ha generado preocupaciones tanto para inversores como para aquellos que simplemente buscan un lugar para vivir.

Según Idealista, el portal inmobiliario líder en España, el año 2023 terminará con un balance aparentemente contradictorio: mientras que las hipotecas disminuyen y el volumen de operaciones se resiente tras un año relevante como fue 2022, los precios de las viviendas continúan al alza, impulsados por una demanda que aún supera a una oferta cada vez más limitada [18].

Este proyecto busca abordar precisamente estos desafíos, aplicando procesos de Ingeniería de Datos y técnicas de Aprendizaje Automático para optimizar la toma de decisiones en el ámbito de las inversiones inmobiliarias. Se pretende ofrecer una herramienta que no solo sea capaz de identificar oportunidades de inversión o búsqueda de nuevo hogar, sino también estudiar las fluctuaciones del mercado.

El proyecto se estructura en varias fases críticas, comenzando con la adquisición y preparación de datos a través de técnicas de *scraping* y procesos ETL, seguido de la implementación de modelos de Aprendizaje Automático para evaluar el valor de las propiedades inmobiliarias en venta. La integración de estos datos enriquecidos en una aplicación web interactiva permite a los usuarios identificar inversiones inmobiliarias atractivas, además de estudiar la evolución del mercado inmobiliario.

Esta memoria también aborda los desafíos técnicos y metodológicos encontrados durante el desarrollo del proyecto, desde la selección de la fuente de datos hasta la optimización de los modelos de Aprendizaje Automático y la implementación de la solución en un entorno de producción. Además, se explora la relevancia de las tecnologías empleadas, remarcando por qué se han considerado para el proyecto.

Al finalizar, se presentan las conclusiones derivadas del trabajo, así como posibles líneas de investigación y desarrollo futuro para expandir y enriquecer aún más la solución propuesta. Por último, se incluyen Apéndices como el plan de proyecto, la guía de diseño de datos y arquitectura, el manual de programador y manual de usuario del sitio web resultante, disponible en buscahogar.es.

Objetivos del proyecto

Este proyecto tiene como objetivo principal desarrollar una solución de ingeniería de datos e inteligencia de negocio para descubrimiento de oportunidades inmobiliarias, para ello se establecen las siguientes metas:

1. Desarrollo de un *scraper* para la sección de compra de inmuebles de `www.pisos.com` :

   Este objetivo implica el desarrollo de un software que acceda periódicamente a la sección de compra-venta de inmuebles del portal inmobiliario y recopile información actualizada de dichos inmuebles a la venta. Se pretende acceder a datos de todas las provincias de España en orden de publicación (más reciente a menos).

2. Almacenamiento de los datos obtenidos por el *scraper* en una base de datos no relacional:

   El software de *scraping*, tras la recopilación de datos de cada inmueble los cargará en diversas colecciones (una por provincia) de una base de datos no relacional. Se guardarán para cada inmueble la mayor cantidad posible de atributos.

3. Transformación y agregación de los datos para su posterior uso:

   Se pretende desarrollar un flujo de datos que extraiga los datos de la base de datos no relacional, los transforme, limpie y además los agregue según distintas dimensiones: territoriales, temporales y según si el anuncio sigue activo.

4. Carga de los datos transformados en una base de datos relacional:

   Los datos transformados y agregados se cargarán en formato tabular en una base de datos relacional que admita consultas SQL.

5. Entrenamiento de modelos de Aprendizaje Automático utilizando los datos almacenados. Enriquecimiento de los datos de la base de datos relacional mediante el uso de dichos modelos:

   Un objetivo principal es utilizar los datos transformados y limpiados para entrenar modelos de Aprendizaje Automático que permitan asignar una puntuación a cada inmueble que permita aproximar cómo de interesante resulta ese inmueble.

6. Desarrollo de una web donde se puedan acceder y visualizar los datos transformados, agregados y enriquecidos:

   Se pretende mostrar los datos recopilados por el flujo en una interfaz web, en esta web también se mostraran las puntuaciones aportadas por los modelos de Aprendizaje Automático. Además permitirá la visualización de los datos agregados.

7. Orquestar el proceso de manera que los datos se extraigan, carguen, transformen y analicen de manera prolongada en el tiempo:

   Se pretende que el flujo de datos se orqueste de manera que funcione de forma automatizada y continúa durante todo el proyecto, ofreciendo datos actualizados a diario.

Conceptos teóricos

En este capítulo se explicarán algunos conceptos teóricos que merece la pena poner en contexto para entender mejor el proyecto y la presente memoria.

3.1. *Scraping*

El *scraping* es una técnica utilizada para extraer información de páginas web de manera automática. El proceso implica hacer solicitudes HTTP a los sitios de interés, recibir las páginas web como respuesta y luego analizar estas páginas para extraer los datos deseados [21].

Este proceso, debe realizarse de manera ética y responsable, evitando sobrecargar los servidores del sitio web con solicitudes y respetando los términos de servicio y las leyes de propiedad intelectual y privacidad. Las técnicas de *scraping* son poderosas pero deben utilizarse con precaución y respeto por las fuentes de datos y los derechos de los demás.

En el contexto de este trabajo, será la principal y única forma de obtención de datos, por lo que representa un pilar fundamental en el flujo. Esto se debe a que el portal `www.pisos.com` no presenta una API funcional que nos permita extraer información, siendo el *scraping* la única herramienta a nuestra disposición.

3.2. Bases de datos no relacionales y relacionales

Existen dos categorías principales de sistemas de gestión de bases de datos: no relacionales y relacionales. La elección depende de los requisitos y objetivos específicos del proyecto.

Las relacionales son ideales para datos estructurados y relaciones complejas, las NoSQL ofrecen mayor flexibilidad y escalabilidad, siendo más adecuadas para manejar grandes volúmenes de información no estructurada o semi-estructurada [22]. Es vital entender las necesidades del proyecto y las características de cada tipo de sistema para tomar una decisión informada.

Bases de datos relacionales

Las bases de datos relacionales están estructuradas de manera que es posible identificar y acceder a los datos en relación con otro dato dentro de la base de datos. Estos sistemas siguen el modelo relacional propuesto por Codd [11].

Características

- **Estructura Tabular:** Los datos se organizan en tablas compuestas por filas y columnas.
- **Integridad Referencial:** Se mantienen las relaciones entre las tablas a través de claves primarias y foráneas.
- **Normalización:** Los datos se organizan para reducir la redundancia y mejorar la integridad.
- **SQL:** Utilizan el lenguaje de consulta estructurado (SQL) para definir, manipular y acceder a los datos.

Ejemplos

Algunos ejemplos son MySQL, PostgreSQL, Microsoft SQL Server, SQLite, Oracle Database.

Bases de datos no relacionales (NoSQL)

Las bases de datos no relacionales, o NoSQL, están diseñadas para permitir operaciones de lectura y escritura de datos más flexibles y escalables que las bases de datos relacionales [15].

Características

- **Esquema Dinámico:** No requieren un esquema predefinido, lo que permite insertar datos de manera más flexible.
- **Escalabilidad Horizontal:** Pueden manejar grandes volúmenes de datos y tráfico a través de sistemas distribuidos.
- **Modelos de datos Variados:** Soportan diversos modelos de datos, incluyendo documentos, clave-valor, columnares y grafos.

Tipos

- **Documentales:** Almacenan los datos en documentos, comúnmente en formato JSON. Ejemplo: MongoDB [27].
- **Clave-Valor:** Los datos se almacenan como pares de clave-valor. Ejemplo: Redis.
- **Columnares:** Organizan los datos por columnas en lugar de filas. Ejemplo: Cassandra.
- **Grafos:** Diseñadas para representar relaciones complejas entre los datos. Ejemplo: Neo4j.

3.3. ETL

ETL es un acrónimo que representa el proceso de Extracción, Transformación y Carga (Extract, Transform, Load en inglés). Este procedimiento es fundamental en el ámbito de la gestión de datos y las bases de datos, especialmente cuando se trata de la manipulación y análisis de grandes volúmenes de información [5].

Extracción

La fase de Extracción consiste en recolectar o extraer datos de diversas fuentes. Estos datos pueden originarse en distintos formatos y estructuras, provenir de distintas bases de datos, archivos de texto, archivos Excel, entre otros. El objetivo de esta etapa es reunir la información necesaria para posterior análisis y procesamiento, considerando los requisitos y necesidades del proyecto en cuestión.

Transformación

Una vez extraídos, los datos pasan por la fase de Transformación. Durante este paso, la información reunida se procesa para convertirla en un formato más adecuado y coherente para el análisis posterior. Esto puede implicar la limpieza de datos, la resolución de inconsistencias, la conversión de tipos de datos, la creación de nuevas variables o características, y otros procesos destinados a mejorar la calidad y utilidad de los datos.

Carga

La última fase, Carga, implica transferir y depositar los datos transformados en un sistema de destino, que puede ser una base de datos, un *data warehouse* [19] o cualquier otro tipo de repositorio de datos. Este paso es crucial para que los datos estén disponibles y accesibles para análisis y consultas posteriores por parte de los usuarios o sistemas que los necesiten.

En conjunto, el proceso ETL facilita la gestión y manipulación de datos, permitiendo que los usuarios y organizaciones accedan a información limpia, consistente y valiosa para la toma de decisiones y la generación de *insights*. Este proceso es fundamental en proyectos de inteligencia de negocios, análisis de datos y ciencia de datos, entre otros campos relacionados con la gestión y análisis de información [5].

3.4. Agregación de datos

La agregación de datos es un proceso fundamental en el tratamiento de grandes volúmenes de información. Este proceso implica combinar datos de múltiples fuentes y resumirlos en un formato más manejable y útil para el análisis [8]. En el contexto de este proyecto, la agregación de datos es crucial para sintetizar la información inmobiliaria extraída y transformada, facilitando su análisis posterior mediante la web y evitando que el *backend* de la web tenga que computar estas agregaciones en cada petición.

3.5. Aprendizaje Automático

El Aprendizaje Automático (AA), o *Machine Learning* (ML) en inglés, es un subcampo de la Inteligencia Artificial (IA) que se centra en construir sistemas que aprenden de los datos. A través del análisis y reconocimiento de patrones complejos en los datos, estos sistemas pueden mejorar su rendimiento y toma de decisiones con el tiempo sin ser explícitamente programados para hacerlo [23].

Tipos de Aprendizaje Automático

Hay varios tipos de Aprendizaje Automático, cada uno con sus propias aplicaciones y uso de datos.

Aprendizaje Supervisado

El aprendizaje supervisado se realiza utilizando un conjunto de datos etiquetado. Una etiqueta es, esencialmente, una respuesta o resultado conocido que se asocia con un dato específico. Por ejemplo el precio, que intentamos predecir de un inmueble, sería su etiqueta. En este enfoque, el algoritmo hace predicciones o clasificaciones basadas en la entrada y se ajusta con el *feedback* proporcionado por las etiquetas [23].

Es importante diferenciar entre dos tipos principales de tareas en el aprendizaje supervisado: clasificación y regresión. Mientras que la clasificación se enfoca en asignar categorías discretas a los ejemplos, la regresión se centra en predecir una cantidad continua, como es el caso del precio de un inmueble mencionado anteriormente. En este proyecto, nos centramos en la regresión, ya que ajustamos un modelo para predecir un valor continúo (precio) basándonos en variables de entrada.

Aprendizaje No Supervisado

El aprendizaje no supervisado trabaja con conjuntos de datos que no están etiquetados. Estos algoritmos identifican patrones y relaciones inherentes en los datos, como la segmentación de clientes en grupos con preferencias similares [23].

Aprendizaje Por Refuerzo

En el aprendizaje por refuerzo, los algoritmos aprenden interactuando con su entorno y recibiendo *feedback* en forma de recompensas o penalizaciones [23].

Proceso de Aprendizaje

A continuación se esquematizará un proceso de aprendizaje/entrenamiento [35], aunque cabe remarcar que esto puede variar:

Análisis y Preparación de datos

Antes del entrenamiento, los datos deben ser preprocesados y divididos en conjuntos de entrenamiento y prueba. Esto facilita la evaluación del rendimiento del modelo.

Entrenamiento del Modelo

El modelo se entrena utilizando el conjunto de datos de entrenamiento, ajustando sus parámetros para minimizar el error en sus predicciones.

Evaluación y Validación

Una vez entrenado, el modelo se evalúa utilizando el conjunto de datos de prueba para estimar su rendimiento y precisión en datos no vistos previamente.

Implementación y Monitorización

Los modelos de Aprendizaje Automático deben ser implementados y monitoreados continuamente para asegurar que sigan siendo efectivos y relevantes con el paso del tiempo y los cambios en los datos.

Algoritmos y Herramientas

Existen numerosos algoritmos de Aprendizaje Automático, desde regresión lineal y logística hasta redes neuronales y algoritmos multiclasificadores (de *ensamble*). Herramientas como TensorFlow, PyTorch y Scikit-Learn [35] permiten implementar y trabajar con diversos algoritmos de AA.

Aplicaciones del Aprendizaje Automático

El Aprendizaje Automático se utiliza en una amplia gama de aplicaciones, incluido el reconocimiento de voz e imagen, recomendación de productos, predicción de ventas, diagnóstico médico y análisis financiero, entre otros. Su versatilidad lo convierte en una herramienta valiosa en muchos campos e industrias. Respecto al análisis de mercados, como puede ser la bolsa o el mercado inmobiliario, aunque sin duda ha habido intentos, se ha logrado menos éxito en la aplicación de estos modelos, en gran parte por la complejidad e impredecibilidad de dichos mercados [40].

3.6. Contenedores de software

Los contenedores de software se han convertido en una herramienta esencial en el desarrollo y despliegue de aplicaciones. Estos contenedores proporcionan un entorno ligero y portátil para ejecutar aplicaciones, garantizando que funcionen de manera uniforme y eficiente en diferentes entornos de computación [44]. Cabe señalar Docker [28], como la herramienta más extendida de contenedores de software, que además coincide con la utilizada en este proyecto.

Principalmente vamos a tratar tres conceptos básicos: Imágenes, Contendedores y Volúmenes [28].

Imagen

Es un paquete ligero, ejecutable e independiente que incluye todo lo necesario para ejecutar una pieza de software, incluyendo el código, el tiempo de ejecución, las herramientas del sistema, las librerías y las configuraciones.

Contenedor

Es una unidad de software que se puede ejecutar y que supone una instancia de una imagen. Encapsula el código y todas sus dependencias para

que la aplicación se ejecute de manera rápida y confiable desde un entorno de computación a otro.

Volumen

Un volumen, en el contexto de los contenedores de software, es un mecanismo para persistir y administrar los datos generados y utilizados por los contenedores. A diferencia de los contenedores, que son efímeros por naturaleza, los volúmenes están diseñados para ser duraderos y existir independientemente del ciclo de vida de los contenedores individuales. Esto los hace esenciales para aplicaciones y servicios que necesitan almacenar datos de manera persistente o compartir datos entre varios contenedores.

3.7. Orquestación

En el ámbito de proyectos de datos, especialmente en tareas de ingestión de datos, ETL y análisis, la orquestación juega un papel crucial. Implica coordinar y automatizar una serie de procesos y tareas interdependientes para garantizar que los flujos de trabajo de datos sean eficientes, escalables y fiables. Un ejemplo popular de herramienta de Orquestación es Apache Airflow [3], utilizada en este proyecto, pero además existen otras muchas como Dagster [12], Luigi de Spotify [42], o AWS Step Functions (AWS: (*Amazon Web Services*) [1].

Características Principales de la Orquestación

- **Planificación y Secuenciación de Tareas:** Define el orden y las dependencias entre las diferentes tareas de ingestión de datos, transformación y análisis.
- **Gestión de Dependencias y Errores:** Maneja las dependencias entre tareas y asegura una gestión adecuada de errores y reintentos.
- **Escalabilidad y Eficiencia:** Optimiza los recursos y escala los procesos según las necesidades de carga de trabajo.

Técnicas y herramientas

En este capítulo se detallarán las técnicas y herramientas utilizadas para el desarrollo e implementación del proyecto, se justificará su elección y como se adecuan a la escala de tiempo y recursos disponibles para el proyecto.

4.1. Tecnología de *scraping*: Scrapy

En el ámbito del web *scraping* existen múltiples herramientas y librerías que facilitan la extracción de datos de sitios web para su posterior análisis o almacenamiento. La Tabla 4.1 compara algunas de las tecnologías de web *scraping* más conocidas, evaluándolas en base a varios criterios como el rendimiento, facilidad de uso y flexibilidad.

Para el proyecto actual, que tiene como objetivo extraer información relevante sobre la compraventa de inmuebles en `pisos.com`, se ha seleccionado Scrapy [38] como la tecnología de *scraping* a utilizar. A continuación, se describen las razones de esta elección:

Rendimiento y Concurrencia: Scrapy es conocido por su alta velocidad y eficiencia, permitiendo la extracción de grandes cantidades de datos en un tiempo reducido. Esto es especialmente útil para nuestro proyecto, donde la actualización frecuente de los anuncios y las ofertas es una constante. Uno de los factores a destacar es su capacidad de concurrencia al utilizar la librería Twisted [45]. El desarrollo de Scrapy basado en dicha librería permite hacer peticiones de forma asíncrona, habilitando gran cantidad de peticiones concurrentes que no se bloquean entre sí [37].

Tecnologías	Rendimiento	Facilidad	Flexibilidad	Escogida
Scrapy	**XX**	X	**XX**	**X**
Beautiful Soup	X	**XX**	X	
Selenium	X	X	X	
Requests	X	**XX**	X	
Mechanize	X	X	X	

Tabla 4.1: Comparación de Tecnologías de Web *scraping*

Facilidad de Uso: Aunque Scrapy tiene una curva de aprendizaje inicial más pronunciada en comparación con, por ejemplo, Beautiful Soup [7], ofrece una gran cantidad de funcionalidades *out-of-the-box* que aceleran el proceso de desarrollo una vez se comprende su funcionamiento básico.

Flexibilidad: Scrapy es altamente configurable y extensible, lo cual permite adaptar el scraper a necesidades específicas. Esto resulta particularmente útil cuando se trata de sitios web con estructuras más complejas o cuando se requieren funcionalidades avanzadas como el manejo de sesiones, cookies o cabeceras HTTP.

Madurez y Comunidad: Scrapy es una tecnología madura con una comunidad activa, lo que asegura un buen soporte y una amplia disponibilidad de documentación y recursos de aprendizaje.

Preferencia del autor para aprender la herramienta: Algunas de las herramientas como Beautiful Soup [7] y Selenium [39] ya habían sido usadas por el autor, por lo tanto uno de los motivos para decidir por Scrapy fue el hecho de aprender una nueva herramienta ampliamente usada en la comunidad.

Por estos motivos, Scrapy se presenta como la opción más robusta y versátil para el web *scraping* de `www.pisos.com`, proporcionando las herramientas necesarias para llevar a cabo un proyecto exitoso.

4.2. Base de datos primaria: MongoDB

Los datos recopilados a través de *scraping* deben almacenarse de forma permanente para su posterior uso. Esto requiere inevitablemente usar una base de datos.

Una de las primeras decisiones fue usar una base de datos no relacional, por la flexibilidad que podía aportar al tratarse de datos de *scraping*. Es frecuente que estos datos no tengan una estructura muy definida o que varíen considerablemente en formato, campos disponibles o tipos de datos. En tales casos, las bases de datos relacionales pueden resultar restrictivas, ya que exigen un esquema fijo y predefinido que todos los registros deben seguir. Por el contrario, MongoDB [27], al tratarse de una base de datos NoSQL documental [10], permite una estructura más flexible, lo que hace que sea más adecuada para manejar datos heterogéneos.

Dentro de las bases de datos NoSQL, otra decisión rápida fue la de elegir una que fuera Open Source y con licencia gratuita. Esto no solo ayuda a mantener bajos los costos del proyecto, sino que también abre la puerta a una amplia comunidad de desarrolladores y una gran cantidad de recursos y documentación en línea. De las bases de datos que cumplían con estos requisitos, MongoDB destacó por varias razones:

- **Prevalencia en la Industria**: MongoDB es una de las bases de datos NoSQL más populares y ampliamente utilizadas. Esto no solo la hace una tecnología atractiva para aprender desde una perspectiva de desarrollo profesional, sino que también asegura una amplia gama de soporte comunitario y empresarial.
- **Experiencia Previa**: Contar con experiencia previa en MongoDB reduce significativamente la curva de aprendizaje, permitiendo un desarrollo más rápido y eficiente del proyecto.
- **Escalabilidad**: MongoDB ofrece una excelente escalabilidad horizontal, lo que permite manejar grandes volúmenes de datos y tráfico sin degradar el rendimiento. Esto es especialmente útil para proyectos de *scraping* que comienzan con un tamaño pequeño pero crecen rápidamente.
- **Consultas Flexibles**: MongoDB ofrece un sistema de consultas flexible y potente que permite realizar búsquedas complejas, algo que es especialmente útil cuando se trata de analizar y utilizar los datos recopilados.

- **Integración con otras Tecnologías**: MongoDB se integra fácilmente con numerosas plataformas y lenguajes de programación, lo que la convierte en una opción versátil para cualquier *stack* tecnológico. La integración con Scrapy, la librería de Python para web *scraping* utilizada fue trivial gracias a la librería Pymongo, la librería de mongodb para Python.

Por estas razones, MongoDB se convirtió en la opción más atractiva para este proyecto, ofreciendo la combinación ideal de flexibilidad, escalabilidad y soporte comunitario.

4.3. Herramientas de ETL: Pandas

Para el proceso de Extracción, Transformación y Carga (ETL) de los datos, se decidió utilizar Python con la ayuda de la librería Pandas [32]. A continuación, se describen las razones que llevaron a esta elección:

- **Volumen de datos**: Uno de los principales factores fue el tamaño moderado del conjunto de datos con el que se está trabajando. Con aproximadamente 500 MB de datos recopilados cada 2-3 meses, se ha considerado que dicho volumen no justifica el uso de una herramienta diseñada para el procesamiento de datos masivos, como Apache Spark.
- **Eficiencia y Velocidad**: Con Pandas y Python, la transformación de toda la base de datos se puede realizar en cuestión de segundos. Esto significa que no hay una necesidad inmediata de una infraestructura más compleja y potente. Usar Spark en este contexto sería una forma de sobreingeniería que añadiría complejidad innecesaria al proyecto.
- **Recursos de Máquina**: Spark exige una asignación de recursos considerablemente mayor que Pandas [4], especialmente si se configura en un modo distribuido con múltiples nodos reales. Estos recursos podrían ser más eficientemente dedicados a otras tecnologías o aspectos del proyecto en la única máquina virtual que tenemos disponible.
- **Costo y Licencia**: Al ser una librería de código abierto, Pandas no incurre en costos adicionales, lo cual es beneficioso desde el punto de vista económico del proyecto.

Por todas estas razones, se optó por utilizar Python con Pandas para las operaciones de ETL en este proyecto. Esta elección se alinea con los

requisitos y limitaciones del proyecto, ofreciendo una solución eficiente y efectiva, sin incurrir en la complejidad y los costos adicionales que implicaría la implementación de una herramienta como Spark.

4.4. Base de datos secundaria: SQLite

Los datos originados de la base de datos primaria y transformados mediante el proceso ETL (Extracción, Transformación y Carga) son nuevamente almacenados en una base de datos. Para este paso del flujo de datos, se optó por una base de datos relacional como SQLite [43] por diversas razones:

- **Búsquedas y Ordenaciones Rápidas**: Las bases de datos relacionales son particularmente eficientes en la realización de consultas que involucran ordenamientos y *joins* [15], lo que resulta útil para las interfaces web donde se necesita acceder a los datos de forma rápida y eficiente.
- **Estructura Clara**: Este tipo de base de datos proporciona un esquema bien definido, lo cual facilita el análisis de datos y las operaciones de Aprendizaje Automático, que generalmente aceptan sin transformaciones adicionales los datos tabulares.
- **Economía de Recursos**: Dado que el volumen de datos a manejar es relativamente pequeño (inferior a 500 MB cada 2-3 meses de recolección), una base de datos ligera y eficiente como SQLite es más que suficiente para satisfacer las necesidades del proyecto.
- **Integración con Python**: SQLite permite una integración nativa, sin librerías adicionales y extremadamente sencilla con Python, lo que simplifica significativamente el flujo de trabajo y evita la necesidad de implementar y mantener un servidor de base de datos separado, la base de datos con tecnología SQLite se almacena simplemente como un archivo ".db".
- **Facilidad de Migración**: En caso de que el proyecto escale y requiera una solución más robusta, la migración a una base de datos relacional más potente, como PostgreSQL o MySQL, sería un proceso relativamente sencillo. Esto es debido a que el esquema y las consultas SQL podrían reutilizarse con pocos ajustes.

Por lo tanto, SQLite se convierte en una excelente opción para este escenario específico. Ofrece la ventaja de ser ligero y eficiente en el uso de

recursos, mientras proporciona todas las funcionalidades necesarias para realizar análisis de datos y Aprendizaje Automático de forma eficaz. Además, su fácil integración con Python y la flexibilidad para una futura migración hacen de SQLite una elección pragmática y efectiva para este proyecto. El hecho de que la base de datos entera se almacene en un único archivo `.db` ha facilitado enormemente el esfuerzo iterativo de desarrollo.

Cabe señalar que, si este proyecto se fuese a desplegar en un entorno más ambicioso, la migración a alguna tecnología de base de datos relacional como Postgres o MySQL es extremadamente recomendable.

4.5. Orquestador: Apache Airflow

Originalmente, la ingestión y transformación de datos se manejaban con una simple programación `cron`, característica de sistemas Unix. Sin embargo, con la evolución del proyecto y la incorporación de nuevos flujos de trabajo, como el entrenamiento periódico de modelos de Aprendizaje Automático y la aplicación de dichos modelos a nuevos datos, se hizo evidente la necesidad de un sistema de orquestación más robusto y flexible. Se optó por utilizar Apache Airflow [3] por las siguientes razones:

- **Gestión de Dependencias**: Apache Airflow permite definir de manera clara y estructurada las dependencias entre las diferentes tareas del flujo de trabajo. Esto resulta especialmente útil cuando los flujos de trabajo se vuelven más complejos y dependen de múltiples etapas y condiciones para su ejecución exitosa.
- **Interfaz de Usuario**: Apache Airflow viene con una interfaz de usuario intuitiva que facilita el monitorización del estado de los flujos de trabajo, la revisión de logs y la identificación de cuellos de botella o fallos en el sistema. Ver Figura C.7.
- **Programación Flexible**: A diferencia de `cron`, que tiene limitaciones en cuanto a la programación de tareas, Airflow permite una gran flexibilidad en la definición de horarios y desencadenantes para la ejecución de tareas.
- **Integración con Otras Herramientas**: Airflow se integra fácilmente con una amplia variedad de tecnologías y plataformas, desde bases de datos hasta servicios de almacenamiento en la nube, lo que facilita la implementación de flujos de trabajo complejos.
- **Gestión de Errores y Reintentos**: Con Airflow, es posible configurar políticas de reintentos y alertas, lo que mejora la robustez del sistema al manejar fallos y excepciones de manera más efectiva.
- **Código Como Configuración**: Airflow permite definir flujos de trabajo como código en las denominadas DAGs o *Directed Acyclic Graphs* [41]. Esto facilita la versión, el mantenimiento y la colaboración en el desarrollo.
- **Comunidad y Soporte**: Apache Airflow cuenta con una comunidad de desarrolladores activa y una gran cantidad de documentación en

línea [3], lo que facilita el proceso de adaptación y resolución de problemas.

Por lo aquí expuesto, Apache Airflow fue elegido como la solución de orquestación más adecuada para este proyecto. Su flexibilidad, escalabilidad y robustez hacen que sea una herramienta altamente eficaz para coordinar múltiples tareas, algo que `cron` simplemente no podría manejar de manera tan efectiva.

4.6. Machine Learning: `Scikit-learn`

Para las tareas de Aprendizaje Automático en este proyecto, se ha seleccionado `Scikit-learn` [33], una librería de Aprendizaje Automático de código abierto para el lenguaje de programación Python.

Características y Razones para la elección:

A continuación, se describen las razones de esta elección y las características destacadas de la librería:

- **Amplia Gama de Algoritmos**: Ofrece una extensa colección de algoritmos de Aprendizaje Automático para tareas de clasificación, regresión, clustering y reducción de dimensionalidad, lo cual es esencial para experimentar con diferentes enfoques y técnicas en el análisis de datos inmobiliarios.
- **Facilidad de Uso y Flexibilidad**: La API de `Scikit-learn` es coherente y fácil de usar, lo que permite una rápida implementación y experimentación con modelos de Aprendizaje Automático. Su diseño intuitivo y la integración con otras librerías de Python, como NumPy y Pandas, facilitan el trabajo con conjuntos de datos y la transformación de datos.
- **Documentación y Comunidad**: Cuenta con una amplia documentación y tutoriales, así como una comunidad activa, lo que facilita el aprendizaje y la resolución de problemas durante el desarrollo del proyecto.
- **Rendimiento y Eficiencia**: A pesar de ser una librería de alto nivel, está optimizada para un alto rendimiento, lo que es crucial para el procesamiento eficiente de conjuntos de datos de tamaño moderado.

- **Interoperabilidad con Herramientas de ETL y Orquestación**: Se integra perfectamente con las herramientas de ETL como Pandas y el orquestador Apache Airflow, lo que permite una gestión eficaz del flujo de trabajo de datos desde la extracción y transformación hasta el análisis y modelado de Aprendizaje Automático.
- **Soporte para Evaluación y Validación de Modelos**: La librería incluye herramientas para la validación cruzada y la evaluación de modelos, lo cual es fundamental para garantizar la precisión y robustez de los modelos de Aprendizaje Automático desarrollados.

Aplicaciones de `Scikit-learn` en el Proyecto

La librería se utilizó para abordar diversas tareas clave en el proyecto:

1. **Predicción de Precios de Inmuebles**: Para la inferencia del precio de los inmuebles, se emplearon diversos algoritmos de Aprendizaje Automático, incluyendo RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor, ExtraTreesRegressor, DecisionTreeRegressor, LinearRegression, Ridge Regression, Lasso Regression, Elastic Net, SVR (Support Vector Regression) y KNeighborsRegressor, todos ellos disponibles en `Scikit-learn`.

   Estos algoritmos se entrenan para predecir el valor de los inmuebles según características como ubicación, tamaño, número de habitaciones, entre otras. Para cada provincia, se utilizan todos los algoritmos mencionados y se selecciona aquel que presentaba el menor valor de RMSE (*Root Mean Squared Error*). El RMSE es una medida de precisión que indica la magnitud promedio de los errores en las predicciones [9]; un RMSE menor implica un modelo de predicción más preciso y fiable para la estimación de precios de inmuebles.

   Como detalle a mencionar, para cada provincia se entrenan dos modelos, uno para los inmuebles con precio inferior a 350 000€ y otro para los inmuebles con precio superior a 350 000€. Esta división se realizó porque se observó que el RMSE relativo (Considerado como el RMSE dividido por la media de precios del conjunto de datos) era inferior tanto en los modelos con precio inferior y superior a 350 000€ comparado con modelos que consideraran todo el rango de precio agrupado. La explicación se amplia en la Sección 5.5.

2. **Asignación de una puntuación al precio real del inmueble**: Una vez entrenados los modelos de Aprendizaje Automático para cada provincia y rango de precios. Se utilizaban dichos modelos para inferir y asignar un precio al inmueble según sus características. Este precio "justo" asignado se comparaba con su precio real. Con esta comparación se le asignó una puntuación al inmueble, si su precio real estaba por debajo del predicho, la puntuación era positiva y si su precio real estaba por encima del predicho, la puntuación era negativa.

La fórmula utilizada para asignar la puntuación fue la siguiente:

$$\text{Puntuación} = \frac{\text{Precio Asignado por Modelo} - \text{Precio Real}}{\text{Precio Real}} \tag{4.1}$$

A continuación se muestran dos ejemplos, aplicando la fórmula. Un ejemplo de inmueble considerado "no interesante" para inversión:

$$\text{Puntuación (-0.5)} = \frac{\text{Precio Asignado (100000€)} - \text{Precio Real (200000€)}}{\text{Precio Real (200000€)}} \tag{4.2}$$

Un ejemplo de inmueble considerado "interesante" para inversión:

$$\text{Puntuación (0.3)} = \frac{\text{Precio Asignado (260000€)} - \text{Precio Real (200000€)}}{\text{Precio Real (200000€)}} \tag{4.3}$$

4.7. Aplicación Web: *Frontend* - React

La interfaz de usuario de la aplicación web, que sirve como el punto de interacción principal para los usuarios, ha sido desarrollada utilizando React. Se trata de una librería de JavaScript para construir interfaces de usuario, conocida por su eficiencia, flexibilidad y el rico ecosistema de componentes y herramientas que ofrece [26].

Razones para Elegir React

A continuación, se detallan los aspectos clave de la elección de la librería y su implementación en el proyecto:

- **Componentes Reutilizables:** Se basa en un enfoque de componentes reutilizables [25], lo que facilita el desarrollo y mantenimiento de la interfaz de usuario. Esto permite una construcción modular del *frontend*, donde cada componente puede ser desarrollado y probado de manera independiente.
- **Estado y Gestión de datos:** Proporciona un sistema robusto para el manejo del estado de la aplicación, lo que resulta crucial para una aplicación web que necesita responder dinámicamente a los cambios en los datos y a las interacciones del usuario, por ejemplo resulta ideal para crear un cuadro de mandos con visualizaciones.
- **Rendimiento:** Gracias al Virtual DOM de React, la actualización de la interfaz de usuario es eficiente y rápida, lo cual es esencial para proporcionar una experiencia de usuario fluida, especialmente en aplicaciones web que manejan una gran cantidad de datos en tiempo real.

Implementación en el Proyecto

La aplicación web desarrollada con React juega un papel crucial en la visualización y el acceso a los datos recopilados y procesados en el proyecto:

1. **Visualización de datos:** Se implementaron componentes para mostrar datos inmobiliarios de manera interactiva, incluyendo listas de propiedades, mapas y gráficos estadísticos.
2. **Interactividad y Filtrado de datos:** La aplicación permite a los usuarios filtrar y buscar propiedades basándose en varios criterios,

como ubicación, precio y características de la propiedad, lo que mejora significativamente la experiencia del usuario.

3. **Integración con *Backend*:** React se ha integrado de manera efectiva con el *backend* del proyecto, asegurando que los datos se actualicen y se muestren en tiempo real.

4. **Responsividad - *Responsive*:** Se intentó prestar atención a la responsividad del diseño, asegurando que la aplicación web funcione sin problemas en una variedad de dispositivos y tamaños de pantalla.

4.8. Aplicación Web: *Backend* - Node.js, Express

El *backend* de la aplicación web, encargado de manejar la lógica de negocio, la interacción con la base de datos y la comunicación con el *frontend*, ha sido desarrollado utilizando Node.js [30] junto con la librería Express [14].

Razones para Elegir Node.js y Express

Esta combinación se ha elegido por su eficiencia, escalabilidad y la facilidad con la que se puede construir aplicaciones web robustas y de alto rendimiento. Node.js, basado en el motor V8 de JavaScript, permite desarrollar servidores de aplicaciones ligeros y eficientes, mientras que Express aporta una capa de abstracción para manejar rutas, solicitudes y respuestas HTTP de una manera más conveniente. A continuación, se listan algunas características adicionales que se consideran importantes para la elección:

- **Desarrollo Unificado en JavaScript:** Utilizar Node.js permite un desarrollo cohesivo entre el *frontend* y el *backend*, ya que ambos pueden ser escritos en JavaScript. Esto facilita la integración y reduce la curva de aprendizaje para los desarrolladores familiarizados con este lenguaje.

- **Rendimiento y Escalabilidad:** Node.js es conocido por su alto rendimiento en aplicaciones basadas en eventos y operaciones no bloqueantes. Esto es ideal para manejar múltiples solicitudes simultáneas, lo cual es común en aplicaciones web modernas.

- **Ecosistema Rico:** Esta combinación tiene un ecosistema muy amplio, con una gran cantidad de módulos y librerías disponibles a través del gestor de paquetes npm, lo que facilita la implementación de funcionalidades adicionales y la integración con otras herramientas y servicios.

- **Flexibilidad y Minimalismo de Express:** Express, siendo una librería minimalista, otorga una gran flexibilidad en la construcción del servidor, permitiendo que la aplicación sea estructurada de acuerdo a las necesidades específicas del proyecto.

Implementación en el Proyecto

En el contexto de este proyecto, el *backend* desarrollado con Node.js y Express desempeña varias funciones clave:

1. **API RESTful:** Se ha implementado una API RESTful para facilitar la comunicación entre el *frontend* y el *backend*, permitiendo realizar operaciones de CRUD (Crear, Leer, Actualizar y Eliminar) en los datos inmobiliarios almacenados en la base de datos SQLite. Aunque a día de hoy todas las peticiones implementadas son de lectura de la base de datos, en el futuro esto puede ser ampliar.

2. **Integración con la Base de datos:** El *backend* gestiona todas las interacciones con la base de datos SQLite, especial mención a las consultas, que a día de hoy son la mayor parte de las interacciones.

3. **Procesamiento de datos:** Además de servir datos al *frontend*, el *backend* también lleva a cabo procesamientos complejos, como filtrados y búsquedas avanzadas, necesarios para las visualizaciones en la aplicación web.

4.9. Contenedores de Software - Docker

En este proyecto, se ha optado por utilizar Docker [24], una plataforma de contenedores de software, para asegurar la consistencia del entorno de desarrollo y producción, así como para facilitar la implementación y escalabilidad del sistema. Docker proporciona una forma de empaquetar y distribuir aplicaciones en contenedores ligeros y portátiles, lo que asegura que la aplicación funcione de manera uniforme y eficiente en cualquier entorno.

Características Principales de Docker

Docker se ha convertido en una herramienta esencial en el desarrollo de aplicaciones modernas debido a sus siguientes características [28]:

- **Independencia de Entorno:** Los contenedores Docker encapsulan todo lo necesario para ejecutar una aplicación, incluyendo el código, las librerías, las dependencias del sistema y los archivos de configuración. Esto elimina el problema de "funciona en mi máquina" al asegurar la coherencia entre los entornos de desarrollo, prueba y producción.

- **Eficiencia en Recursos:** A diferencia de las máquinas virtuales, los contenedores Docker comparten el núcleo del sistema operativo del host y no requieren un sistema operativo completo para cada contenedor, lo que los hace más ligeros y rápidos.

- **Portabilidad:** Los contenedores pueden ejecutarse en cualquier máquina que tenga Docker instalado, independientemente del sistema operativo y de la infraestructura subyacente, lo que facilita la migración y el despliegue en diferentes entornos.

- **Escalabilidad y Aislamiento:** Docker facilita la escalabilidad y el balanceo de carga de las aplicaciones. Cada contenedor funciona de forma aislada, lo que mejora la seguridad y permite escalar o replicar contenedores de manera individual.

Uso de Docker en el Proyecto

En el marco de este proyecto, Docker ha sido utilizado para varios propósitos clave:

1. **Desarrollo Consistente:** Se utilizan contenedores Docker para mantener un entorno de desarrollo consistente con el entorno de producción. Esto reduce los problemas de incompatibilidades.

2. **Despliegue y Distribución:** Docker simplifica el proceso de despliegue al empaquetar la aplicación y sus dependencias en un contenedor, que puede ser fácilmente distribuido y ejecutado en diferentes sistemas y plataformas. Todos los servicios de este proyecto están desplegados a través de contenedores Docker, y orquestados utilizando Docker Compose [13].

4.10. Despliegue - Oracle Cloud - *Compute Instance*

El despliegue efectivo de la aplicación web y sus componentes asociados, como las bases de datos, el proceso de *scraping*, ETL y *Machine Learning*, se ha realizado en una *Compute Instance* de Oracle Cloud [31]. Oracle Cloud Infrastructure (OCI) ofrece servicios de computación en la nube de alto rendimiento y escalables, que son ideales para alojar todo tipo de aplicaciones. La elección de Oracle Cloud para el despliegue se basa en que ofrecen una máquina de altas prestaciones de forma gratuita. Además, la integración con contenedores Docker desplegados en una *Compute Instance* facilita un proceso de despliegue y operación eficiente y automatizado.

Características Clave de Oracle Cloud y *Compute Instance*

La infraestructura de Oracle Cloud y su servicio *Compute Instance* ofrecen varias ventajas y características importantes, aunque muchas de ellas son comunes a otras compañías que ofrecen servicios cloud, merece la pena mencionarlas:

- **Alto Rendimiento y Disponibilidad:** Las *Compute Instance*s de Oracle Cloud están diseñadas para ofrecer un alto rendimiento y una alta disponibilidad, lo que es esencial para mantener la aplicación web accesible y eficiente en todo momento.
- **Precio gratuito:** Oracle Cloud proporciona una máquina de 4 núcleos y 24GB de RAM con 200GB de disco duro de forma gratuita.
- **Seguridad Mejorada:** Con herramientas avanzadas de seguridad y una infraestructura robusta, Oracle Cloud garantiza la protección de los datos y las aplicaciones alojadas.
- **Gestión Simplificada:** La interfaz de usuario y las herramientas de gestión de OCI simplifican la configuración, el monitorización y la administración de las *Compute Instance*s.

Implementación del Proyecto en Oracle Cloud

El despliegue del proyecto en Oracle Cloud incluyó los siguientes pasos y consideraciones:

1. **Configuración de la *Compute Instance*:** Se estableció una instancia en Oracle Cloud con sistema operativo Ubuntu 20.04.
2. **Despliegue de Contenedores Docker:** Todos los componentes de la aplicación, incluyendo el servidor *backend* y la base de datos, se empaquetaron en contenedores Docker y se desplegaron en la *Compute Instance*. Esto asegura una integración y funcionamiento fluidos de todos los componentes del sistema.
3. **Configuración de Red y Seguridad:** Se realizaron ajustes en la configuración de la red y la seguridad para proteger la instancia y garantizar un acceso controlado a la aplicación y a sus servicios.

Beneficios del Despliegue en Oracle Cloud

El uso de Oracle Cloud para el despliegue del proyecto ha aportado beneficios significativos, tales como una mayor eficiencia en la gestión de recursos, una mejora en la disponibilidad ya que se requiere la ejecución de procesos varias veces al día. Se ha conseguido sincronizar todos los procesos para un buen rendimiento de la aplicación, con flexibilidad para adaptarse a las cambiantes necesidades del proyecto a medida que se avanzaba en el desarrollo. Además, en un futuro sería trivial migrar los servicios de la *Compute Instance* de Oracle Cloud a cualquier servidor, sea alojado en la nube o no, gracias al uso de contenedores de software (Docker).

Aspectos relevantes del desarrollo del proyecto

Este capítulo aborda los aspectos más críticos en el desarrollo del proyecto. Se han explicado aquellos aspectos del proyecto que han supuesto un gran impacto en diseño, han alterado el curso del mismo o han supuesto una gran inversión de tiempo o recursos.

5.1. Diseño inicial

Desde el comienzo del proyecto, se esbozó una visión general de su desarrollo. Esta visión implicaba la ingestión de datos inmobiliarios para posteriormente transformarlos, limpiarlos y aplicar modelos de Aprendizaje Automático, con el objetivo de brindar valor al usuario final, quien sería el consumidor de estos datos.

Inicialmente, la ingestión de datos se contemplaba a través de dos vías posibles: mediante una API o mediante técnicas de *scraping*. Se consideró utilizar Apache Spark como herramienta principal para las tareas de Extracción, Transformación y Carga (ETL) de datos. Sin embargo, tras un análisis cuidadoso, esta opción fue desestimada por considerarse innecesaria. Dentro del diseño, se identificó la necesidad de contar con una base de datos primaria (datos sin procesar, menos estructurados) y otra secundaria (datos procesados y tabulares). Asimismo, se exploraron diversas librerías de Aprendizaje Automático, como ML/MLlib de Spark y Scikit-Learn, evaluando su potencial y adecuación al proyecto.

En cuanto a la visualización de datos y resultados derivados del análisis, se contempló inicialmente la creación de una aplicación móvil, la cual permitiría

a los usuarios acceder fácilmente a los hallazgos y *insights* generados tras la implementación de los modelos de Aprendizaje Automático seleccionados. En partes posteriores del proyecto, se descartó la aplicación móvil y se optó por una web que también estuviese adaptada a dispositivos móviles.

5.2. Elección de la fuente de datos

En una fase inicial, se consideró `www.idealista.com` como la fuente principal de datos, dado que este portal es el más grande en el sector inmobiliario de España.

Idealista ofrece una sección donde los usuarios pueden solicitar acceso a su API. Aunque se recibieron credenciales y autorización para el uso de la API por parte de idealista, el acceso estaba notablemente limitado a 100 llamadas API mensuales. Dicha restricción resultaba claramente insuficiente para un proyecto como este. Pese a que se intentó negociar un incremento en el límite de llamadas, no se recibió respuesta alguna por parte de Idealista. Adicionalmente, la plataforma de Idealista posee robustas medidas anti-*scraping*, y está claramente limitado como se indica en su archivo "robots.txt". Aún así y tras diversas pruebas, observamos que las técnicas de *scraping* fueron efectivamente identificadas y bloqueadas por su sistema.

En consecuencia, se hizo necesario explorar alternativas. El portal `pisos.com` se identificó como la fuente ideal de datos para nuestro proyecto, y fue desde este portal de donde, finalmente, se extrajeron los datos necesarios.

5.3. Selección de Tecnología ETL

En la sección de diseño inicial, se discutió la posibilidad de emplear Apache Spark como tecnología principal para el proceso de Extracción, Transformación y Carga (ETL) de datos. No obstante, tras un análisis más cuidadoso, se determinó que no había necesidad de recurrir a una herramienta de la envergadura de Spark. A pesar de que los datos incorporaban una considerable cantidad de propiedades en venta en España, estos no alcanzaban un volumen masivo, sumando aproximadamente 500 MB generados cada varios meses. En este contexto, la computación en paralelo no solo resultaba innecesaria, sino que podía llegar a ser contraproducente. Vale la pena mencionar que, utilizando la librería Pandas de Python, el proceso ETL completo de la base de datos apenas tardaba segundos.

Adicionalmente, otro factor crucial en la decisión de descartar Spark fue el consumo de recursos. El proyecto opera en una Máquina Virtual gratuita proporcionada por Oracle de 4 núcleos y 24GB de RAM, la cual sirve como servidor para múltiples servicios. Dada esta infraestructura, se concluyó que no resultaba eficiente mantener procesos de Spark activos: Se recomiendan al menos 8GB (por nodo/máquina del cluster) para ejecutar Spark [4], nosotros contábamos solo con un nodo/máquina de 24GB, en la que había que incluir: servicio de *scraping*, base de datos primaria (mongodb), servicio de ETL, base de datos secundaria (SQLite), servicio de entrenamiento y predicción utilizando Aprendizaje Automático y servidor web con posibilidad de múltiples usuarios concurrentes. Teniendo en cuenta que nos encontrábamos ante volúmenes de datos fácilmente almacenables en memoria sin uso de computación distribuida, no se observó necesidad de usar Spark.

Por estos motivos, se optó por realizar todo el proceso ETL utilizando Python, con el soporte de librerías eficientes y confiables como Pandas, sin usar computación distribuida.

5.4. Optimización del Sistema de Orquestación

Inicialmente, los procesos de ingestión (*scraping*) y ETL estaban orquestados de manera rudimentaria utilizando `cron` en la máquina virtual. Sin embargo, con el inicio del análisis de datos y la exploración preliminar de modelos de *Machine Learning*, se hizo evidente la necesidad de una coordinación más sofisticada entre diversos procesos. Estos incluían el entrenamiento de modelos, la aplicación de dichos modelos a los datos para su utilización por parte del usuario final, y la ejecución de *backups* de las bases de datos, entre otros.

Este reconocimiento de necesidades más complejas y específicas llevó a la implementación de un sistema de orquestación más avanzado y apto, en este caso, Apache Airflow. Este no solo ofrece una interfaz gráfica web intuitiva y fácil de usar, sino que también proporciona una plataforma robusta que facilita la definición, configuración y programación de flujos de trabajo complejos y dependientes. Con la adopción de Airflow, se logró una mejora significativa en la gestión y coordinación de los distintos procesos involucrados, lo que a su vez ha contribuido a un funcionamiento más eficiente y fiable del sistema en su conjunto. Además, se brinda flexibilidad para escalar y expandir el sistema en el futuro, facilitando así la incorporación y orquestación de nuevos procesos y servicios conforme evolucionan las necesidades y objetivos del proyecto.

5.5. Implementación de modelos de Aprendizaje Automático en el flujo de datos. División de los modelos

La integración de modelos de Aprendizaje Automático en el flujo de datos del proyecto representó una etapa crítica, que comenzó con una fase exploratoria exhaustiva de los datos. Utilizando Jupyter Notebooks [20], se realizó un análisis preliminar para comprender las características y la distribución de los datos inmobiliarios. Este análisis permitió identificar patrones, tendencias que podrían influir en el rendimiento de los modelos predictivos.

En esta fase exploratoria, se experimentó con diversos algoritmos de Aprendizaje Automático, incluyendo regresión lineal, *Random Forest*... Una de las decisiones estratégicas más importantes fue la separación de los modelos según el territorio. Se observó que tener un único modelo para todos los inmuebles empeoraba notablemente el rendimiento, respecto a tener distintos modelos entrenados con los datos de un territorio (posteriormente reorganizados en provincias, explicado en Sección 5.8).

Además, dentro de cada territorio, se diferenció entre inmuebles con un valor de 350 000€ o más ("caros") y aquellos con un valor inferior (a partir de ahora, "baratos"), creando así, dos modelos por territorio. Esta estrategia de segmentación resultó en un notable incremento en el rendimiento de los modelos, ya que predecían de forma más precisa, algo bastante lógico ya que cada territorio, región o ciudad puede tener un mercado inmobiliario muy diferente y además como afectaban las características de inmuebles "caros" a su precio en el mercado, era muy distinto a como afectaban a los inmuebles "baratos". Los inmuebles "baratos" solían tener un precio regido por características más lineales y predecibles, como tamaño, número de habitaciones, planta, si tiene terraza... en cambio los "caros" eran muy variables y dependientes de la zona, algo que cobra sentido ya que el "lujo" puede ser un mercado ciertamente subjetivo.

Sin embargo, trasladar los modelos de una fase exploratoria a un entorno de producción presentó diversos desafíos. Uno de los principales obstáculos fue la automatización del proceso de entrenamiento y la integración de los modelos en el flujo de datos existente, de manera que las predicciones pudieran generarse y actualizarse de forma continua. La infraestructura de datos tuvo que ser cuidadosamente diseñada para soportar la recopilación,

limpieza y preparación de datos en tiempo real, asegurando que los modelos siempre tuvieran acceso a los datos más recientes y relevantes.

Otro aspecto crucial fue la gestión del rendimiento y los recursos. Dado el volumen y la complejidad de los datos, así como la necesidad de actualizar los modelos regularmente, se estableció un ciclo de entrenamiento de 30 días. Este intervalo se eligió para equilibrar la necesidad de mantener los modelos actualizados con las tendencias actuales del mercado, sin imponer una carga excesiva en los recursos computacionales disponibles, ya que el propio entrenamiento se hacía en la única máquina virtual del proyecto. De forma aproximada, el entrenamiento llevaba aproximadamente una hora.

5.6. Implementación de contenedores de software

Desde el inicio, el proyecto tuvo servicios modularizados y diferenciados. Además estos servicios estuvieron desplegados y en funcionamiento en una máquina Ubuntu. Por ejemplo, en un inicio la base de datos MongoDB y Apache Airflow estaban instalados directamente en el sistema operativo de dicha máquina. Al igual que el servicio de *scraping* o el de ETL con Python/Pandas estaban instalados en entornos virtuales de Python montados directamente en el sistema operativo de la máquina.

A medida que el número de servicios crecía y la interacción entre estos se hacía más compleja, se complicaba el paso del entorno desarrollo a la máquina de producción. Para solventar este desafío se optó por crear contenedores de software para todos los servicios, utilizando Docker [24].

Para algunos servicios como los de MongoDB y Apache Airflow, directamente se utilizó una imagen oficial, teniendo que hacer pocos cambios sobre ellos. En el caso de MongoDB, los datos de la base de datos se montan en un volumen. Los volúmenes son unidades de espacio en disco que se pueden montar en distintos contenedores y quedan almacenados de forma permanente [24]. Lo que se hizo fue hacer un volcado de los datos de MongoDB en este volumen, restaurando así la base de datos hasta la fecha en una base de datos MongoDB funcionando ya, plenamente en un contenedor.

Para otros servicios como el de scraping (Ver Sección 4.1), ETL (Ver Sección 4.3) o el de Aprendizaje Automático (Ver Sección 4.6) se utilizó como base una imagen `Python:3.11-slim` sobre la que se instalaban librerías adicionales según las que necesitara el servicio.

Para el servicio de SQLite (Ver Sección 4.3), al tratarse de un único archivo `.db` se almacenaba en un volumen que se compartía con los distintos contenedores que necesitaban acceso a dicha base de datos.

Posteriormente, para el servicio web, se desplegó un contenedor para el *frontend* (Ver Sección 4.7), otro para el *backend* (Ver Sección 4.8) y otro para el servidor Nginx de *proxy-reverso* [36].

Finalmente, se crearon unos volúmenes adicionales para el almacenamiento permanente de los *logs* de los distintos servicios y uno para almacenar los modelos de Aprendizaje Automático.

5.7. Rediseño de módulo de *scraping* debido a cambios en la web pisos.com

El rediseño del módulo de *scraping* debido a los cambios en la página web de pisos.com fue un hito crítico en el desarrollo del proyecto. Este cambio supuso un desafío significativo, ya que supuso un parón total en la recogida de los datos y por tanto en todo el flujo de datos. La página individual para un inmueble, donde se especifican todos los detalles, experimentó una actualización que afectó directamente a la estrategia de extracción de datos previamente implementada. Esto ocurrió en torno al 17-18 de enero de 2024, y se detectó el 19 de enero.

Inicialmente, el módulo de *scraping* estaba diseñado para interactuar con una estructura web específica, extrayendo información como la ubicación, el precio, las características del inmueble, entre otros detalles, todo ello a partir de la estructura html de la página. Para arreglar el problema, simplemente se adaptó el código del *scraper*. Sin embargo, con el código actualizado, el *scraper* se observó que aleatoriamente era capaz de recoger los datos y otras veces no.

Después de un análisis detallado y diversas pruebas, se identificó la causa raíz del problema: la presencia de un balanceador de cargas o algún mecanismo similar en el servidor de pisos.com. Este sistema alternaba aleatoriamente entre servir la versión antigua y la nueva del sitio web a los usuarios, incluido el *scraper*. Como resultado, el módulo de *scraping* debía ser capaz de reconocer qué versión de la página estaba procesando para aplicar el conjunto correcto de reglas de extracción de datos.

Para superar este desafío, se implementó una solución dinámica en el módulo de *scraping*. Primero, el *scraper* fue equipado con un mecanismo de detección que identificaba la versión de la página web con la que estaba interactuando. Dependiendo de esta identificación, el *scraper* seleccionaba dinámicamente el conjunto de reglas de *scraping* adecuado para esa versión específica de la página. Esta estrategia permitió una adaptabilidad esencial, asegurando una recolección de datos consistente y fiable.

La implementación de esta solución no solo restableció la eficacia del proceso de *scraping* sino que también destacó la importancia de la flexibilidad y la adaptabilidad en los sistemas de extracción de datos. Este aprendizaje fue crucial para el proyecto, ya que subrayó la necesidad de diseñar sistemas capaces de ajustarse a cambios inesperados en las fuentes de datos, garantizando así la continuidad y la fiabilidad del análisis inmobiliario a largo plazo.

Este suceso reafirmó la importancia de una monitorización continua y una rápida capacidad de respuesta ante los cambios en las fuentes de datos externas. Además, demostró que, en el dinámico entorno de la web, los proyectos de análisis de datos deben estar preparados para adaptarse rápidamente a nuevas circunstancias para mantener la integridad y la relevancia de su análisis.

En este caso concreto, el día 21 de enero de 2024 se restableció el flujo de datos. Además el módulo de *scraping* se diseñó para tener flexibilidad en como de frecuente se ejecutaba, no suponiendo ningún problema que no hubiese funcionado durante unos días, ya que los inmuebles actualizados durante esos días en `pisos.com`, se extrajeron posteriormente.

5.8. Reorganización de los datos en provincias y capitales

La reorganización de los datos en provincias representó una etapa crucial en el proceso de mejora y refinamiento del proyecto de análisis de datos inmobiliarios. Esta decisión fue motivada por la necesidad de estandarizar y optimizar la estructura de datos para facilitar el análisis y la aplicación de modelos de Aprendizaje Automático de manera más efectiva. La estructura inicial de recogida de datos presentaba una mezcla heterogénea de ciudades, provincias e islas, lo que complicaba el análisis comparativo y la precisión de los modelos predictivos. Además en el proceso de ETL se incorporó la creación de un valor booleano según si el inmueble pertenecía a la capital de provincia o no.

Redefinición de la Estructura de datos

El primer paso en este proceso de reorganización fue la redefinición de la estructura de datos para agrupar la información inmobiliaria por provincias. Esta tarea implicó un trabajo de clasificación y asignación de cada inmueble a su provincia correspondiente, eliminando las inconsistencias previas donde elementos como Bilbao (ciudad) y Gijón (ciudad) se mezclaban con provincias e islas sin una distinción clara. La adopción de este enfoque provincial permitió un análisis más coherente y comparativo entre diferentes áreas geográficas, facilitando la identificación de tendencias y patrones específicos por provincia.

Actualización de las Colecciones en MongoDB

Para implementar esta nueva estructura de datos, fue necesario realizar una actualización exhaustiva de las colecciones en MongoDB. Este proceso implicó la reorganización de los registros existentes y la actualización de los esquemas de datos para reflejar la agrupación por provincias. Además, se desarrollaron scripts específicos para automatizar la migración de datos a la nueva estructura, asegurando la integridad de los mismos durante el proceso de transición.

Modificación de los Puntos de Entrada para el *Scraper*

La estrategia de *scraping* también se vio afectada por esta reorganización. Los puntos de entrada para el *scraper* fueron modificados para alinearse con la nueva estructura de provincias. Esto implicó la actualización de las URL objetivo y la adaptación de las reglas de *scraping* para capturar datos de manera más eficiente y organizada según la provincia.

Ajuste en el Proceso de ETL y agregación

El proceso de Extracción, Transformación y Carga (ETL) también se adaptó para soportar la nueva estructura de datos. Los flujos de trabajo de ETL se reconfiguraron para procesar y organizar los datos inmobiliarios recogidos en función de su provincia correspondiente.

De forma relevante para el producto final, ahora el proceso de ETL utilizaba la ubicación del inmueble (en texto) para ver si correspondía con la Capital de Provincia o no.

Además el proceso de Agregación de datos ahora también agrupaba según la provincia y según si el inmueble pertenecía a la Capital de Provincia o no.

Entrenamiento de Modelos de Aprendizaje Automático y Detección de Capitales

La reorganización de los datos en provincias trajo consigo la oportunidad de mejorar los modelos de Aprendizaje Automático. Los modelos fueron entrenados específicamente con datos organizados por provincias, lo que permitió una mayor consistencia, al no mezclar ciudades, provincias e islas.

Además, se incorporó el valor booleano de si un inmueble pertenecía a la capital de la provincia o no, un factor relevante en la valoración inmobiliaria. Este enfoque diferenciado mejoró significativamente la capacidad del sistema para predecir precios de manera más precisa, considerando la influencia que tiene la ubicación en la capital o no en el valor de un inmueble.

Trabajos relacionados

En este capítulo revisamos el trabajo previo en el campo de estudio de análisis inmobiliario mediante Aprendizaje Automático y como ha afectado al desarrollo del proyecto.

De forma significativa, encontramos un artículo de Baldominos et al. [6] que destaca la utilización de métodos variados de *Machine Learning*, incluyendo conjuntos de árboles de regresión, regresión de vectores de soporte, *k-nearest neighbors* y redes neuronales. Pese a la diversidad de enfoques, en el artículo se resalta una notable escasez de investigaciones que apliquen directamente el *Machine Learning* en la valoración de propiedades inmobiliarias. El documento propone un innovador enfoque de regresión utilizando características extraídas de listados en línea para estimar precios de mercado, marcando un punto de partida significativo para futuros estudios y además, el presente proyecto.

El estudio de Niu et al. [29] introduce un sistema de valoración inmobiliaria avanzado que aprovecha técnicas de *Machine Learning* no solo para evaluar los inmuebles sino también para abordar desafíos específicos como la identificación de registros duplicados y la extracción/identificación efectiva de características a usar en los modelos. La identificación de registros duplicados resulta esencial si se extraen datos de distintos portales inmobiliarios, ya que los anuncios habitualmente están repetidos. Además, el enfoque del artículo, resalta la importancia de aplicar un conjunto de múltiples modelos a la vez, como *Gradient Boosting Decission Trees*, *Random Forest* y redes neuronales de tipo *Back Propagation* (BP), para mejorar la precisión en la valoración de propiedades. Los resultados experimentales obtenidos con datos de propiedades en Hangzhou, China, demuestran la superioridad de

su sistema combinado, suponiendo un interesante punto de partida para la posible futura expansión del actual proyecto.

Otro punto interesante, volviendo al artículo de Baldominos et al. [6], es que en este se estudia un solo barrio, el barrio de Salamanca de Madrid. Se muestra cómo este enfoque, barrio por barrio, es más potente, ya que en cada zona de una ciudad se crea un mercado diferenciado y se valoran distintas características de los inmuebles. Dentro de la misma ciudad, Madrid, es muy distinto el barrio de Salamanca al de Carabanchel. También debemos señalar, que este modelo barrio por barrio hace mucho más difícil escalar el proyecto a nivel de todo un país, sobre todo por la falta de suficientes datos. En ciudades pequeñas o medianas, es habitual que solo haya algunos inmuebles a la venta en un determinado barrio en un determinado período de tiempo, siendo insuficientes para entrenar un modelo. También es cierto, que en ciudades pequeñas o medianas suele haber menos heterogeneidad entre barrios. Sería interesante, para futuros estudios, la intersección entre un mecanismo híbrido, donde en provincias o ciudades con suficientes datos, se aplican modelos a nivel de barrio y en zonas con insuficientes datos, se aplique a nivel provincial o municipal.

Finalmente, en el artículo se observa que los modelos que mejores resultados obtienen son los árboles de regresión conjuntos, o *Ensembles of Regression Trees* (Figura 6.1). Esto resulta consistente con lo observado en el presente proyecto, donde en la mayoría de las provincias el modelo con mejor resultado suele ser un árbol de regresión conjunto.

En el presente proyecto se han separado los modelos de *Machine Learning* a nivel de provincia, construyendo así dos modelos por provincia, uno para inmuebles "caros" y otro para inmuebles "baratos" (Para detalles ver Sección 5.8). Los modelos no van a ser tan precisos como los utilizados en el enfoque que se hizo en [6], sin embargo, permiten escalar el proyecto a nivel de estudio inmobiliario de todo un país, habiendo pocos precedentes similares en la literatura.

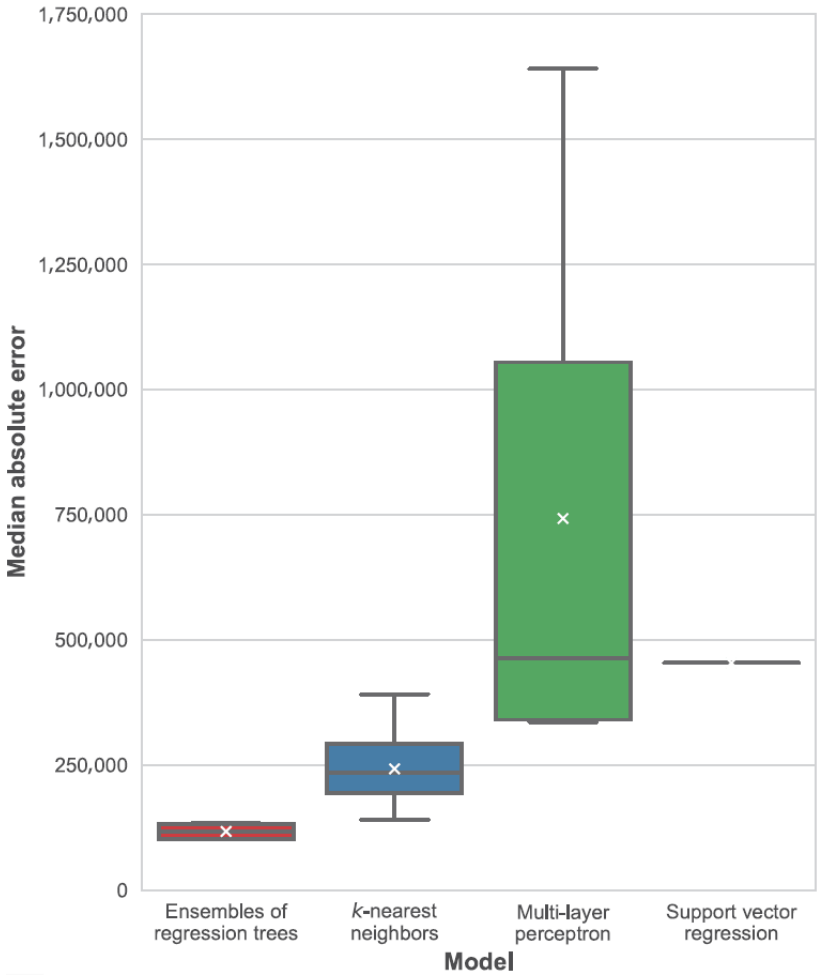

Figura 6.1: Resultados de MAE (*Median Absolute Error*) de los distintos tipos de modelos para regresión de precios en Barrio de Salamanca, Madrid [6]. En el eje Y, vemos MAE en € y en el eje X los distintos tipos de modelos analizados.

Conclusiones y Líneas de trabajo futuras

Este proyecto ha demostrado la viabilidad y el potencial de aplicar procesos ETL y técnicas de Aprendizaje Automático para guiar la toma de decisiones en inversiones inmobiliarias. A través de un enfoque sistemático y la utilización de tecnologías de código abierto, se ha logrado desarrollar una solución que facilita el análisis de las tendencias del mercado inmobiliario y además ofrece una guía de predicción del valor de las propiedades inmobiliarias en forma de una web interactiva y accesible para cualquier persona.

Hallazgos Clave:

Los principales hallazgos del proyecto se pueden resumir en los siguientes puntos:

- La simple extracción, transformación y agregación de los datos, permite crear visualizaciones que aportan conclusiones interesantes sobre el mercado inmobiliario.
- La implementación de modelos de Aprendizaje Automático específicos para diferentes rangos de precios y provincias ha permitido obtener predicciones más precisas y adaptadas a las particularidades del mercado inmobiliario español.
- La utilización de Docker y Oracle Cloud ha proporcionado una infraestructura robusta, escalable y eficiente, esencial para el manejo de los numerosos servicios que intervienen en todo el flujo de datos.

- Aunque los modelos presentan errores significativos en ciertos casos, es posible aplicar un método de filtrado efectivo mediante el uso de un criterio simple: descartar aquellos inmuebles a los cuales el algoritmo asigna un valor aproximadamente un 33-40 % superior al precio de venta. Esta sobrevaloración generalmente se debe a la falta de información en el anuncio, errores en los datos proporcionados, o a características del inmueble (como aspectos negativos visibles en fotos o desventajas de la ubicación) que el modelo no logra interpretar adecuadamente.

Limitaciones y Desafíos:

A continuación se muestran las principales limitaciones encontradas hasta la fecha:

- La recopilación de datos encontró obstáculos significativos debido a las limitaciones de acceso y las medidas anti-*scraping* implementadas por algunos portales inmobiliarios.
- El constante cambio del portal inmobiliario de origen obliga a continuamente adaptar la solución software de *scraping* a dichos cambios.
- La escalabilidad del proyecto se ve limitada por la capacidad de la infraestructura actual: Una sola máquina virtual.
- Los modelos utilizados son totalmente incapaces de interaccionar elementos muy importantes para categorizar el valor: Las fotografías y la descripción. En ellas se pueden ver muchas sutilezas y datos que los seres humanos utilizamos para dar valor a un inmueble.

Líneas de Trabajo Futuras:

Las principales líneas de trabajo futuras que se consideran interesantes son las siguientes:

1. **Expansión de la Fuente de datos:** Explorar nuevos portales inmobiliarios y negociar accesos a APIs para enriquecer el conjunto de datos, mejorar la precisión de los modelos predictivos y eliminar las limitaciones provenientes del uso de *scraping*.

2. **Optimización de Modelos:** Actualmente no se ajustan los hiperparámetros de los modelos, simplemente se prueban distintos algoritmos con los parámetros base y se escoge el que menor error relativo ofrece. Un enfoque inmediato es dedicar una máquina exclusivamente al entrenamiento de los modelos, lo que permitirá dedicar muchas más horas de computación a ajustar los parámetros, sin afectar al resto de servicios.

3. **Mejora de la Infraestructura:** Esta mejora abriría un gran abanico de posibilidades:

   Permitiría un **procesamiento de datos más frecuente** y "casi en tiempo real". Se podría hacer una actualización del flujo de datos cada hora, dando una sensación casi en tiempo real para el usuario, para ello sería esencial tener un servidor dedicado exclusivamente a ETL.

   Se podría expandir notoriamente la **capacidad de usuarios concurrentes** usando la web: Se dedicaría un servidor exclusivamente a la web y otro exclusivamente a la base de datos relacional, que debería migrarse a un sistema más apto para producción que SQLite como por ejemplo Postgres o MySQL.

4. **Aumentar la interacción del usuario con los modelos:** Una expansión interesante de las capacidades del producto final, es permitir a los usuarios interactuar directamente con los modelos. Los usuarios podrían introducir las características que desean para su "inmueble" y el modelo les devolvería el valor medio de mercado esperado para un inmueble de esas características, esto sería útil tanto para saber por cuanto vender un inmueble de forma ajustada con el mercado, como para saber por cuanto podrían esperar comprar en el mercado un inmueble de las características señaladas.

5. **Expandir el uso de Inteligencia Artificial:** Integrar tecnologías avanzadas de Inteligencia Artificial a los modelos de regresión actuales

podría enriquecer significativamente el análisis. La implementación de algoritmos capaces de analizar fotografías para detectar características relevantes, identificar posibles errores en los anuncios, y mejorar la clasificación de inmuebles mediante el procesamiento del lenguaje natural (por ejemplo, analizando las descripciones de los anuncios) podría incrementar notablemente la precisión y profundidad del análisis de propiedades. Por ejemplo, en el artículo de Niu et al. [29], se utiliza un enfoque combinado de identificación de inmuebles repetidos (provenientes de distintos portales), selección de características a usar para entrenar en los modelos y evaluación simultánea con distintos modelos. Esta aproximación supone un interesante punto de partida para la mejora del algoritmo de evaluación de inmuebles del actual proyecto.

Apéndices

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este apartado se pretende exponer la planificación y viabilidad del proyecto. Se mostrará el avance temporal del proyecto, los avances que tuvieron lugar entre cada reunión y el estudio de viabilidad económica y legal.

A.2. Planificación temporal

Se ha intentado seguir la metodología Agile, utilizando *sprints* o bloques de trabajo de unas dos semanas. En ocasiones por circunstancias de trabajo, vacaciones o imposibilidad los *sprints* se han prolongado durante 1 mes. Además hubo una interrupción del proyecto entre el *sprint* 7 y 8 por cuestiones laborales y personales del autor/alumno.

El trabajo no se ha realizado de forma homogénea en todos los *sprints*. Durante los períodos vacacionales (verano y navidad) se ha realizado la mayor parte del desarrollo, y en los meses anteriores al depósito de la memoria se avanzó la mayor parte de la escritura y documentación del proyecto.

Tras cada *sprint* se mantenía una reunión entre los tutores y alumno, en la cual se debatían los avances y se planificaba en qué se iba a trabajar en el siguiente sprint.

Sprint 1 - 27/06/23 a 5/07/23

En este *sprint* se hizo un primer estudio de la viabilidad del proyecto y se decidieron las tecnologías a utilizar según la idea propuesta originalmente. Además se hizo un prototipo del *scraper*.

- Se comprobó que la página web `www.idealista.com` estaba totalmente protegida contra el *scraping*. Tras solicitar una clave para *API* que permitiera el desarrollo del proyecto no se obtuvo respuesta por parte de idealista.

  Por tanto, se decidió que la fuente de los datos iba a ser `www.pisos.com`, otro portal inmobiliario de los más grandes de España y que no bloquea el uso de *scraping*.

- Se hizo un diagrama de las tecnologías que se pretendían usar para la fase de extracción, transformación, carga de los datos y su análisis mediante *Machine Learning*, además de la orquestación de los distintos procesos.

- Se probaron diversas tecnologías de *scraping*, para finalmente decidir usar scrapy.

- Se desarrolló un prototipo de *scraper* para inmuebles en venta en `www.pisos.com`.

Sprint 2 - 05/07/23 a 29/08/23

En este sprint, algo más largo por las fechas vacacionales, se logró un gran avance en la parte de desarrollo.

- Se amplió el software de *scraping*, pasando a recopilar más de 50 datos por inmueble.

- Se diseñó y desplegó la base de datos primaria que almacenaría la ingestión de datos proveniente de *scraping*: MongoDB.

- Se desplegó una máquina virtual Ubuntu alojada en Oracle *cloud*, en la cual se empezaron a recoger datos diariamente.

- Se estableció un mecanismo de *backup* de la base de datos primaria en caso de catástrofe en la máquina virtual del proyecto.

- Se comenzó a esbozar el procedimiento de ETL de datos *raw* almacenados en la MongoDB.

Sprint 3 - 29/08/23 - 13/09/23

En este sprint, se desplegó el proceso de ETL.

- Finalización del diseño y decisión de las tecnologías aplicadas en el proceso de ETL.
- Se desplegó el proceso de ETL, en Python, que recoge los datos en crudo de la MongoDB, los transforma y los carga, en una base de datos relacional: SQLlite.
- Se hicieron algunas exploraciones previas de los datos limpios.

Sprint 4 - 13/09/23 - 26/09/23

Este *sprint* se dedicó a la puesta al día de la documentación del proyecto. Además se desplegó Apache Airflow como orquestador, debido al previsible aumento de complejidad de los procesos.

- Despliegue de Apache Airflow como orquestador.
- Puesta al día de la documentación y memoria de los *sprints* 1, 2, 3 y 4.

Sprint 5 - 26/09/23 - 10/10/23

Este *sprint* se dedicó a finalizar la puesta al día de la documentación del proyecto.

- Puesta al día de la documentación y memoria de los *sprints* 1, 2, 3 y 4.

Sprint 6 - 10/10/23 - 24/10/23

En este *sprint* se comenzó el análisis de los datos provenientes de la ingestión. Se realizó un análisis exploratorio en Jupyter Notebook

- Análisis exploratorio de los datos en la base de datos relacional utilizando Jupyter Notebooks.
- Esbozo del problema a resolver mediante ciencia de datos.

Sprint 7 - 24/10/23 - 07/11/23

En este *sprint* se exploraron los modelos de *Machine Learning* a utilizar y el proceso de entrenamiento de dichos modelos.

- Iteraciones de entrenamiento y predicción con modelos de la libreria Scikit-Learn para intentar resolver el problema de asignar una puntuación a inmuebles.
- Pruebas de entrenamiento y predicción con todo el conjunto de datos en el entorno de Jupyter Notebooks.

Proyecto parado - 07/11/23 - 07/12/23

Por motivos laborales y personales en este período de tiempo no se pudo avanzar en el proyecto.

Sprint 8 - 07/12/23 - 19/12/23

En este *sprint* se creó el esqueleto de la web de visualización de los datos.

- Creación de esqueleto *frontend* funcional con React.
- Creación de servidor *backend* funcional con Node.js y ExpresS que conecta con la base de datos de producción SQLite.

Sprint 9 - 19/12/23 - 02/01/24

En este *sprint* se arreglaron defectos en el proceso de ingestión de datos. Además, se añadieron nuevas funcionalidades a dicho proceso de ingestión de datos. Por otra parte, se desplegó el servicio de *Machine Learning*. Finalmente, se continuaron añadiendo funcionalidades a la web.

- Se arregló un defecto que hacía que un único inmueble (y por tanto una única clave primaria) estuviera duplicada en la base de datos secundaria. El error se producía cuando un inmueble había estado listado en distintas ciudades (y por tanto se ingestaba en distintas colecciones de la base de datos MongoDB).
- Se implementó en el servicio de *scraping* una comprobación de si el inmueble sigue activo. Tanto dentro de la base de datos primaria (MongoDB) como secundaria (SQLite) el inmueble se marca si está activo o no.

- Se desplegó el proceso de entrenamiento de los modelos de *Machine Learning*.
- Se desplegó el proceso que asignaba una puntuación a los inmuebles ya existentes y a todos los nuevos que se fuesen introduciendo, utilizando los modelos previamente mencionados.
- Se continuó el desarrollo de funcionalidades del sitio web donde se muestran los datos y puntuaciones de los inmuebles.
- Se dio por concluida la parte del proyecto de ingeniería y ciencia de datos, salvo por retoques finales.

Sprint 10 - 02/01/24 - 16/01/24

Durante este *sprint* se hizo un esfuerzo por mejorar el entorno DevOps del proyecto mediante el uso de contenedores de software con la tecnología Docker. El proyecto quedó desplegado en una máquina virtual de Oracle Cloud con Sistema Operativo Ubuntu en la que se desplegaron dichos contenedores. El proyecto se consideró operativo a falta de finalizar el desarrollo de la web de análisis y visualización de datos y finalizar la documentación.

- Creación de un contenedor Docker para el servicio de Scrapy.
- Creación de un contenedor Docker para los servicios de ETL.
- Creación de un contenedor Docker para los servicios de *Machine Learning* (Entrenamiento y Predicción).
- Creación de un contenedor Docker para la MongoDB.
- Creación de un contenedor Docker para Airflow Webserver y otro para Airflow Scheduler.
- Creación de un contenedor Docker para el *frontend* y otro para el *backend* de los servicios web.
- Creación de volúmenes docker para almacenamiento permanente de logs, base de datos secundaria (SQLite), base de datos primaria (mongodb) y *logs* y datos de Apache Airflow.
- Orquestación y despliegue de los contenedores utilizando Docker Compose.

- Puesta al día de la documentación y memoria de los *sprints* 6, 7, 8, 9 y 10.

Sprint 11 - 16/01/24 - 30/01/24

En este *sprint* se detectaron cambios en la web www.pisos.com que afectaron significativamente al flujo de datos ya que el *scraping* dejó de funcionar. Supuso un rediseño del módulo de *scraping*.

Además se rediseñó la agrupación geográfica de pisos para que fuera más consistentes. Por último se añadió un nuevo parámetro a todo el flujo de datos: Si in inmueble pertenecía a la capital o no.

Se avanzó significativamente en el *frontend* de la web, tanto en la parte de funcionalidades como en la estética.

- El módulo de *scraping* se adaptó a la nueva web de *www.pisos.com*.
- El script que comprobaba si un inmueble seguía activo se adaptó a la nueva web de *www.pisos.com*.
- Se reorganizaron las colecciones de la base de datos MongoDB, para que coincidieran con las provincias de españa.
- El proceso de ETL crea un nuevo parámetro que indica si el inmueble pertenece a la capital de provincia o no.
- El proceso de agregación de datos ahora tiene en cuenta las provincias y si es capital de provincia o no.
- El proceso de entrenamiento y predicción de Aprendizaje Automático ahora tiene en cuenta las provincias como agrupación territorial y tiene en cuenta si el inmueble es capital o no.
- Se adaptó el *backend* de la web para filtrar por provincias y si el inmueble es capital o no.
- Añadidas todas las funcionalidades principales a las vistas de la web: “INICIO”, “EXPLORA”, “VISUALIZA” e “INFO”.
- Se mejoró la estética de la web.

Sprint 12 - 30/01/24 - 13/02/24

En este *sprint* se continuó con la escritura de la memoria del proyecto y la revisión de cambios propuestos por tutor y co-tutor. Además se corrigieron algunos *bugs* detectados en la web y se mejoraron algunos aspectos de la interfaz de usuario señalados por los revisores.

- Arreglado bug en pestaña "VISUALIZA". No se estaban haciendo la media de las comunidades autónomas, se estaba asignado el valor de la primera provincia de esa comunidad autónoma (por orden alfabético).
- Mejorado comportamiento de filtros en pestaña "EXPLORA". Más intuitivos y fácil de seleccionar rango, solo se actualiza el estado cuando el usuario deja de arrastrar el selector. Añadido comportamiento de *scroll* infinito en el listado de inmuebles.
- Agregada pestaña de información en puntuaciones y selector de modo. "Inteligente" en pestaña "INICIO"
- Añadidas leyendas, títulos y unidades a gráficos de líneas.
- Añadida página de *landing* en pestaña "INICIO"
- Mejorado comportamiento de la web en dispositivos móviles
- Creado dominio buscahogar.es y creado auto renovado de certificado https

Sprint 13 - 13/02/24 - 22/02/24

En este *sprint* se finalizó la escritura de la memoria del proyecto y la revisión por parte del tutor y co-tutor. Además, se retocaron algunos aspectos visuales menores de la web. Se reestructuró código para hacerlo más mantenible.

A.3. Estudio de viabilidad

En esta sección se va a desarrollar cómo de viable ha resultado el desarrollo del proyecto a nivel económico y legal.

Viabilidad económica

Se ha decidido desglosar la viabilidad económica en apartados de personal, infraestructura en la nube y hardware. Finalmente se mostrará el sumatorio.

Coste Personal

El proyecto se ha llevado a cabo por un desarrollador autónomo contratado durante 225 horas. Se considera un coste por hora de 40€ antes de IVA.

Concepto	**Coste/h**	**Horas**	**Coste Acumulado**
Salario bruto	40€	225	9000€
IVA (21 %)	8.4€		1890€
Total Proyecto			10 890€

Tabla A.1: Costes de personal desarrollador

Coste Infraestructura y hardware - Prueba de concepto actual

Uno de los mayores hitos de este proyecto es que se ha enmarcado todo en una sola máquina virtual de tipo `VM.Standard.A1.Flex`, con 24GB de RAM y 100GB de almacenamiento. Esta máquina es proporcionada indefinidamente de forma gratuita por *Oracle Cloud*, por lo que el coste real de infraestructura la prueba de concepto ha sido de **0€** a lo largo de 8 meses.

En el caso de que no se tuviera acceso a esta promoción y como ejercicio para estudiar la viabilidad del proyecto podemos estimar el cálculo de una máquina similar en el proveedor de *Amazon Web Services o AWS*. Se ha estimado el coste de una máquina similar a la usada, *EC2 m6g.xlarge* (Ver Figura A.1). A este coste habría que sumar gastos de almacenamiento [2]. La suma de costes se puede ver en Tabla A.2

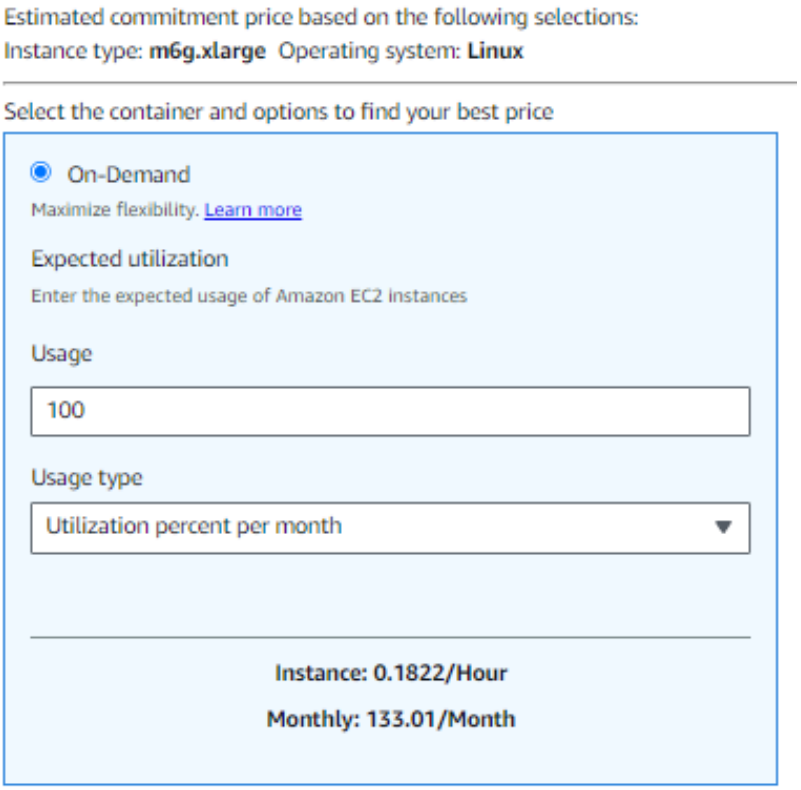

Figura A.1: Coste del servicio de computación en la nube (EC2) de AWS para una máquina *m6g.xlarge* (4 núcleos y 16GB de ram). Coste a día 8 febrero de 2024 para región EU (Spain).

Concepto	**Coste/mes**	**Meses**	**Coste Acumulado**
EC2 m6g.xlarge	133$	8	1064$
EB2 storage (100gb)	8$	8	64$
IVA (21 %)			236$
Total Proyecto en $			1364$
Total Proyecto en € Ratio $ a € de 0.93			1 466€

Tabla A.2: Costes de infraestructura en *cloud* - Estimados para AWS

Coste total del proyecto

En la tabla A.4 se muestra la suma de los costes de personal, infraestructura en la nuble y hardware, mostrados en subsecciones anteriores.

Concepto	Coste	Amortizado/mes	Meses	Total
Portátil	1500€	31.25€	8	250€
Periféricos	300€	6.25€	8	50€
Monitores	300€	6.25€	8	50€
Silla y escritorio	500€	10.42€	8	83.36€
Total Proyecto				433.36€

Tabla A.3: Costes de hardware. Para el cálculo de coste mensual, se ha considerado un período de amortización de 48 meses para el coste total del producto.

Concepto	Coste
Personal	10 890€
Infraestructura Cloud	1 466€
Hardware y accesorios	433€
Total Proyecto	12 789€

Tabla A.4: Coste total del proyecto (8 meses).

Viabilidad legal

En este subapartado se muestran las licencias que tienen las herramientas y librerías usadas (Ver tabla A.5). Siguiendo las recomendaciones GNU [17], se ha escogido la licencia GPL-3.0 [16] para el presente proyecto. Esta licencia permite la modificación, uso y distribución del software, siempre que se realice bajo la misma licencia, se indiquen los cambios y se mencione al autor original.

Por último, durante este proyecto, se ha comprobado que el uso de *scraping* en el portal `www.pisos.com` se trata de un proceso legal y ético, ya que se cumple en su totalidad las recomendaciones especificadas por la página web en su archivo "robots.txt" [34]. Además se han seguido prácticas de limitación de número de peticiones diarias, para no suponer una sobrecarga a los servidores de dicho portal.

Dependencia	**Licencia**
Librerías Python	
Scrapy	BSD
Pandas	BSD-3-Clause
Numpy	BSD
Pymongo	Apache License 2.0
Scikit-learn	BSD
Scipy	BSD
Matplotlib	Python Software Foundation License
Librerías Web	
React	MIT
Node.js	MIT
Express	MIT
Cors	MIT
Dotenv	MIT
Pg	MIT
Sqlite3	MIT
MUI	MIT
Axios	MIT
Bootstrap	MIT
Leaflet	BSD
Recharts	MIT
Herramientas	
MongoDB	Server Side Public License (SSPL)
Apache Airflow	Apache License 2.0
Nginx	BSD de 2 cláusulas

Tabla A.5: Tabla de dependencias del proyecto

Apéndice B

Especificación de diseño

B.1. Introducción

En este apéndice se mostrarán la estructura de los datos, a nivel de base de datos y a nivel de tabla/colección (tanto en MongoDB como en SQLite). Además se detallará la arquitectura del proyecto.

B.2. Diseño de datos

Datos en MongoDB

La base de datos MongoDB recibe el nombre de "pisos" y de divide en las siguientes colecciones:

- `amount_parsed`

  Esta colección sirve para monitorizar de cuantos inmuebles nuevos se introducen en la base de datos en cada ejecución del *scraper*. Se crea un nuevo documento en cada ejecución con la fecha de finalización y el número de inmuebles almacenados en dicha ejecución.

- `last_updated_dates`

  Colección operacional donde se almacenan la fecha del inmueble más reciente de cada colección. Se usa para que cuando el *scraper* llegue a inmuebles actualizados menos recientemente, para de operar. Por ejemplo, si para la colección de Jaén, la última fecha de actualización es el 20 de Enero de 2024, si el *scraper* encuentra un inmueble del 19

de Enero de 2024, se detecta que no es necesario seguir extrayendo datos para esa provincia, ya que ya estarán insertados en la base de datos. Esto funciona porque se acceden a los inmuebles en orden de última actualización decreciente en www.pisos.com.

- `{nombre_de_provincia}`

  En estas es donde se alojan los documentos de inmuebles para cada provincia. Las colecciones que existen son las siguientes:

  a_coruna, alava_araba, albacete, alicante, almeria, andorra, asturias, avila, badajoz, barcelona, burgos, caceres, cadiz, cantabria, castellon_castello, ceuta, ciudad_real, cordoba, cuenca, girona, granada, guadalajara, guipuzcoa_gipuzkoa, huelva, huesca, islas_baleares_illes_balears, jaen, la_rioja, las_palmas, leon, lleida, lugo, madrid, malaga, melilla, murcia, navarra_nafarroa, ourense, palencia, pontevedra, salamanca, santa_cruz_de_tenerife, segovia, sevilla, soria, tarragona, teruel, toledo, valencia, valladolid, vizcaya_bizkaia, zamora, zaragoza

Se debe señalar que los campos de datos de *scraping* cambian según el documento (Ver Figura B.1 para ejemplo), según si hay más detalles listados en el anuncio o menos, aunque todos siguen un patrón similar. Además merece la pena señalar que todos los campos provenientes de *scraping* se almacenan como cadenas de texto, para posteriormente darle tipado en la ETL con carga en la base de datos SQLite. Los campos operacionales esenciales presentes en todos los documentos son los siguientes:

- `id`: Id extraida del portal pisos.com que sirve como identificador único de principio a fin del flujo de datos
- `updated_date`: Fecha de actualización del anuncio extraida del portal pisos.com
- `createdAt`: Momento en el que se almacenó por primera vez el inmueble en la base de datos
- `updatedAt`: Momento en el que se actualizó por última vez el documento en la base de datos
- `version`: Indica cuantas veces se ha actualizado el documento, comienza en 0 con incrementos +1
- `active`: Booleano que indica si el anuncio sigue activo o no

```
_id: ObjectId('64d15b8639613e5fd5bcbf13')
id: "36696441401.106900"
title: "Piso en A Piroja"
location: "Cabreira (Camariñas)"
price: "A consultar"
description: "Piso de 106 m2 construidos a la venta en Camariñas (A Coruña). Se trat…"
link: "https://www.pisos.com/comprar/piso-cabreira_camarinas-36696441401_1069…"
updated_date: "Actualizado el 05/08/2023"
superficie_construida: "106 m²"
habitaciones: "2"
baños: "1"
planta: "1ª"
antigüedad: "Entre 20 y 30 años"
conservación: "En buen estado"
gastos_de_comunidad: "300.00 euros."
referencia: "PA86-I000358-P000028"
amueblado: "Amueblado"
carpintería_interior: "Sin especificar"
tipo_suelo: "Plaqueta"
calefacción: "Eléctrica"
cocina: "Cocina amueblada."
comedor: "Comedor"
balcón: "Tiene 1 balcon(es)"
▸ photos: Array (17)
createdAt: 2023-08-18T14:40:51.316+00:00
version: 0
updatedAt: 2023-08-18T14:40:51.316+00:00
active: true
province: "a_coruna"
```


Figura B.1: Ejemplo de documento almacenado en base de datos MongoDB que representa los datos de un inmueble en la colección “a_coruna”

Datos en SQLite

Tras un proceso de transformación los datos de la base de datos MongoDB se cargan en esta base de datos que consta principalmente de tres tablas:

Tabla `last_updated_dates`

Tabla operacional simple que almacena una columna con el nombre de la colección (de MongoDB) y otra con el tiempo de actualización (`updated_at`) más reciente de dicha colección. Sirve para continuar el proceso de ETL en la siguiente iteración.

Tabla `pisos`

Consta de los datos de inmuebles ya limpiados y transformados. Sirve tanto para el entrenamiento y predicción de los modelos de Aprendizaje

Automático, como punto de partida de los datos agregados y para listar los inmuebles en el sitio web.

Algunas columnas que merece la pena mencionar por su importancia operacional son:

- `id` : Identificador único proveniente de pisos.com
- `province` : La provincia del inmueble y por tanto su colección de MongoDB.
- `capital` : Tiene valor 1 si pertenece a la capital de provincia, 0 si no.
- `type` : Tipo de inmueble, por ejemplo Piso, Casa, Apartamento...
- `price_euro` : El precio de venta anunciado, incluye descuentos.
- `link` : Enlace al inmueble en pisos.com.
- `updated_date` : Fecha actualización del anuncio reflejada en pisos.com.
- `active` : Tiene valor 1 si el inmueble está considerado activo, 0 si no. Resultado del flujo de Scrapy que comprueba si el anuncio sigue activo.
- `createdat` : Fecha de primera carga en MongoDB.
- `updatedat` : Fecha de última actualización en MongoDB.
- `version` : Indicador de cuantas veces se ha actualizado el inmueble en MongoDB.
- `prediction` : Valor en € asignado por los modelos de Aprendizaje Automático.
- `predictionupdatedat` : Tiempo en el que se actualizó el campo `prediction`.
- `rating` : Valor calculado a través de `prediction` y `price_euro`, explicado en 4.6). Sirve para ordenar los inmuebles de mayor a menor oportunidad.

En esta tabla, encontramos otros muchos campos que se pueden consultar en detalle en Tabla B.1. Cabe mencionar que no todos los inmuebles presentan

todos los campos, dependerá de cuales estén listados en el anuncio original, así encontramos muchos de ellos con valor `null`.

Finalmente, merece la pena mencionar que observamos algunos valores calculados que llevan el sufijo `_summary` (marcado como `_summ` en la tabla de referencia B.1) o `_cleaned`, estos datos son reagrupaciones de las mismas columnas sin sufijo que se pueden ver en la tabla, generalmente se trata de campos consolidados para su uso en los modelos. Se ha considerado que merece la pena mantener en la misma tabla los campos sin consolidar (más fieles al anuncio original) y consolidados.

Tabla B.1: Esquema completo de la tabla `pisos`

Columna	**Tipo**	**Desc.**
id	TEXT	ID único
title	TEXT	Título
location	TEXT	Ubicación
province	TEXT	Provincia
capital	INTEGER	0 No, 1 Yes
price_euro	REAL	Precio €
description	TEXT	Descripción
link	TEXT	Enlace
updated_date	TEXT	Fecha act.
sup._constr._m2	REAL	Sup. constr. m^2
sup._util_m2	REAL	Sup. útil m^2
habitaciones	REAL	Habitaciones
banos	REAL	Baños
planta	TEXT	Planta
exterior	TEXT	Exterior
antiguedad	TEXT	Antigüedad
conservacion	TEXT	Conservación
referencia	TEXT	Ref.
terraza	TEXT	Terraza
photos	TEXT	Fotos
createdat	INTEGER	Creado
version	INTEGER	Versión
updatedat	INTEGER	Actualizado
active	INTEGER	Activo
amueblado	TEXT	Amueblado
armarios_empotrados	TEXT	Armarios

Continúa en la siguiente página

Tabla B.1 – continuación de la página anterior

Columna	**Tipo**	**Desc.**
tipo_suelo	TEXT	Suelo
vidrios_dobles	TEXT	Vidrios dobles
carpinteria_exterior	TEXT	Carp. exterior
aire_acondicionado	TEXT	Aire acond.
cocina	TEXT	Cocina
comedor	TEXT	Comedor
garaje	TEXT	Garaje
trastero	TEXT	Trastero
puerta_blindada	TEXT	Puerta blindada
ascensor	TEXT	Ascensor
orientacion	TEXT	Orientación
soleado	TEXT	Soleado
chimenea	TEXT	Chimenea
piscina	TEXT	Piscina
gastos_de_comunidad	TEXT	Gastos com.
agua	TEXT	Agua
calefaccion	TEXT	Calefacción
old_price_euro	REAL	Precio antiguo €
sistema_de_seguridad	TEXT	Seguridad
balcon	TEXT	Balcón
superficie_solar_m2	REAL	Sup. solar m^2
tipo_de_casa	TEXT	Tipo de casa
urbanizado	TEXT	Urbanizado
calle_alumbrada	TEXT	Calle alumbrada
calle_asfaltada	TEXT	Calle asfaltada
portero_automatico	TEXT	Portero autom.
adaptado_movilidad_reducida	TEXT	Adapt. movilidad
jardin	TEXT	Jardín
lavadero	TEXT	Lavadero
se_aceptan_mascotas	TEXT	Mascotas
telefono	TEXT	Teléfono
luz	TEXT	Luz
cocina_equipada	TEXT	Cocina equip.
carpinteria_interior	TEXT	Carp. interior
interior	TEXT	Interior
gas	TEXT	Gas
no_se_aceptan_mascotas	TEXT	No mascotas

Continúa en la siguiente página

Tabla B.1 – continuación de la página anterior

Columna	Tipo	Desc.
esquina	TEXT	Esquina
exterior_summ	TEXT	Resumen exterior
vidrios_dobles_summ	TEXT	Resumen vidrios
adapt._mov._redu._summ	TEXT	Resumen adapt. movilidad
puerta_blindada_summ	TEXT	Resumen puerta blindada
ascensor_summ	TEXT	Resumen ascensor
balcon_summ	TEXT	Resumen balcón
portero_automatico_summ	TEXT	Resumen portero
garaje_summ	TEXT	Resumen garaje
comedor_summ	TEXT	Resumen comedor
terraza_summ	TEXT	Resumen terraza
jardin_summ	TEXT	Resumen jardín
armarios_empotrados_summ	TEXT	Resumen armarios
aire_acondicionado_summ	TEXT	Resumen aire acond.
trastero_summ	TEXT	Resumen trastero
piscina_summ	TEXT	Resumen piscina
chimenea_summ	TEXT	Resumen chimenea
lavadero_summ	TEXT	Resumen lavadero
urbanizado_summ	TEXT	Resumen urbanizado
calle_alumbrada_summ	TEXT	Resumen calle alumbrada
calle_asfaltada_summ	TEXT	Resumen calle asfaltada
soleado_summ	TEXT	Resumen soleado
gas_summ	TEXT	Resumen gas
sistema_de_seguridad_summ	TEXT	Resumen seguridad
interior_summ	TEXT	Resumen interior
esquina_summ	TEXT	Resumen esquina
amueblado_summ	TEXT	Resumen amueblado
cocina_equipada_summ	TEXT	Resumen cocina equip.
mascotas_summ	TEXT	Resumen mascotas
gastos_de_comunidad_cleaned	REAL	Gastos com. limpios
carpinteria_exterior_cleaned	TEXT	Carp. exterior limpia
tipo_suelo_summ	TEXT	Resumen suelo
calefaccion_summ	TEXT	Resumen calefacción
cocina_summ	TEXT	Resumen cocina
orientacion_summ	TEXT	Resumen orientación
agua_summ	TEXT	Resumen agua
type	TEXT	Tipo

Continúa en la siguiente página

Tabla B.1 – continuación de la página anterior

Columna	Tipo	Desc.
alcantarillado	TEXT	Alcantarillado
alcantarillado_summ	TEXT	Resumen alcantarillado
prediction	REAL	Predicción
predictionupdatedat	TIMESTAMP	Fecha act. predicción
rating	REAL	Valoración

Tabla `pisos_dw`

Esta tabla es el resultado del proceso de agregación de datos de la ETL, por lo tanto, se recalcula entera dos veces al día. Su objetivo principal es precalcular datos agregados para las visualizaciones, evitando la necesidad de computar estos datos en la web cada vez que el usuario haga una petición, como por ejemplo, al cambiar los filtros en la pestaña *"VISUALIZA"*. Su origen único siempre es la tabla `pisos`.

Encontraremos todos los campos agrupados por:

- `province_group`: Provincia. Obtenido de `province` de tabla `pisos`.
- `capital_group`: Si pertenecen a Capital de Provincia o no. Obtenido de `capital` de tabla `pisos`.
- `active_group`: Si están en anuncios activos o no. Obtenido de `active` de tabla `pisos`.
- `updated_month_group`: Mes-Año en el que se insertaron por primera vez en MongoDB (calculado desde campo `createdat` de tabla `pisos`)

Además, para cada agrupación, existe un valor `'all'`, que identifica el cálculo sin tener en cuenta dicha agrupación. Por ejemplo, si todas las agrupaciones tienen valor `'all'` los cálculos agregados pertenecerán a todos los inmuebles de la base de datos, sin tener en cuenta provincia, capital, activos o tiempo de creación.

Para algunas columnas provenientes de tabla `pisos`, denominadas como **numéricas** (Ver en Tabla B.2), se calculan la media y la desviación estándar, descartando los *outliers* para el cálculo.

El cálculo de *outliers* se realiza teniendo en cuenta unos límites superiores e inferiores muy restrictivos, ya que no queremos eliminar muchos datos.

Siendo $q1$ el cuartil 1, $q3$ el cuartil 3, e $iqr = q3 - q1$, los límites se definen como:

$$\text{Límite inferior} = q1 - 5 \times iqr$$
$$\text{Límite superior} = q3 + 5 \times iqr$$

Para el resto de columnas provenientes de tabla `pisos`, denominadas como **categóricas** (Ver en Tabla B.3), lo que se hace es calcular qué porcentaje de los inmuebles adquiere cada valor único posible, también siendo agrupados según los grupos mencionados previamente.

Por ejemplo para `piscina_summary` los valores posibles son `YES` o `NO`. Se calculará que porcentaje de los inmuebles tienen cada uno de los valores, y se almacenarán dichos porcentajes en columnas distintas para cada valor único, que en este caso reciben el nombre de `piscina_summary_yes_pct` y `piscina_summary_no_pct`. Debemos tener en cuenta que estos porcentajes se calculan para cada combinación de agrupaciones, siendo así cada fila de la tabla una combinación distinta de agrupaciones.

Columnas	
`price_euro`	`superficie_construida_m2`
`superficie_util_m2`	`superficie_solar_m2`
`habitaciones`	`banos`
`gastos_de_comunidad_cleaned`	

Tabla B.2: Columnas o campos numéricos de la tabla `pisos` usados para tabla agrupada `pisos_dw`. Se muestran en dos grupos de columnas, para comprimir esta tabla.

Columnas	
exterior_summary	comedor_summary
vidrios_dobles_summary	terraza_summary
adaptado_..._reducida_summary	jardin_summary
puerta_blindada_summary	armarios_empotrados_summary
ascensor_summary	aire_acondicionado_summary
balcon_summary	trastero_summary
portero_automatico_summary	piscina_summary
garaje_summary	chimenea_summary
lavadero_summary	soleado_summary
gas_summary	amueblado_summary
cocina_equipada_summary	calefaccion_summary
conservacion	antiguedad
carpinteria_exterior_cleaned	tipo_suelo_summary
cocina_summary	orientacion_summary

Tabla B.3: Columnas o campos categóricos de la tabla `pisos` usados para tabla agrupada `pisos_dw`. Se muestran en dos grupos de columnas, para comprimir esta tabla.

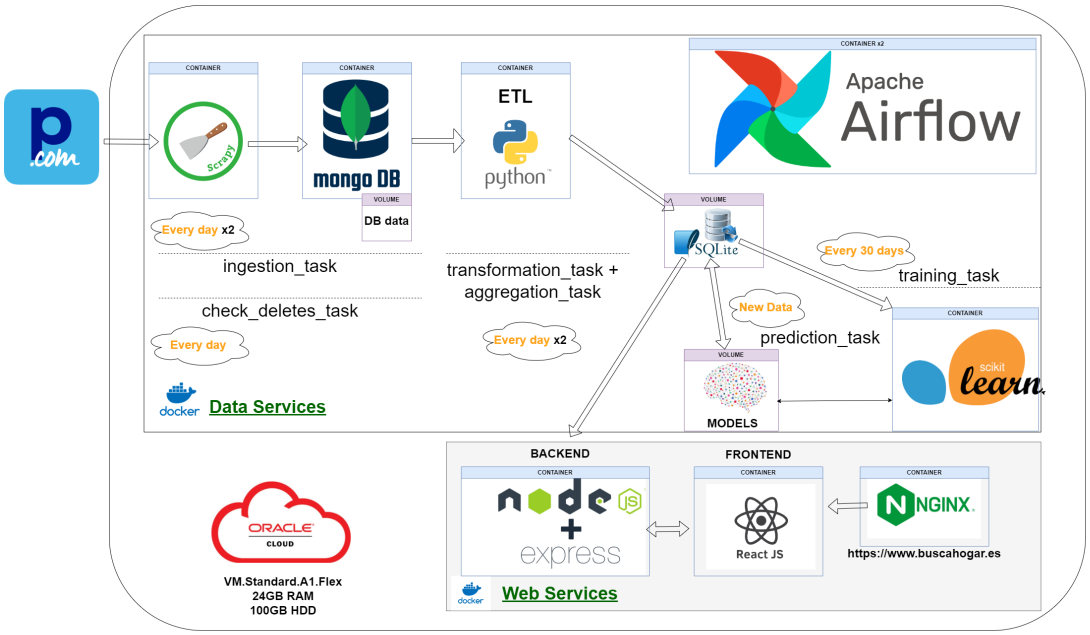

Figura B.2: Arquitectura de software del proyecto

B.3. Diseño de Arquitectura de Software

El diagrama de componentes se muestra en la Figura B.2. Englobando todos los componentes, encontramos un bloque en forma de elipse que representa la máquina virtual del proyecto. Este bloque a su vez contiene dos bloques rectangulares, uno que representa los servicios relacionados con los datos (*Data Services*) y otro, los relacionados con la web (*Web Services*).

Dentro de estos bloques encontramos que cada contenedor de Docker se representa como un bloque con título en azul, cada volumen se representa con un bloque con título en morado. Encontramos las diferentes tareas (con sufijo `_task`) englobando a distintos contenedores, y la frecuencia de la tarea se representa en un bloque con forma de nube.

Apéndice C

Documentación técnica de programación

C.1. Introducción

El objetivo de este Apéndice es documentar a nivel técnico el presente proyecto, con objetivo de permitir la instalación en local del proyecto, facilitar el acceso a un entorno de desarrollo y la extensión del proyecto.

C.2. Estructura de directorios

Todo el proyecto se ha desarrollando utilizando en un único repositorio `git` para control de versiones, que está disponible en el siguiente enlace: https://github.com/dpuertamartos/big_data_tfm.

Dentro del directorio raíz del repositorio, encontramos las carpetas mostradas en la Figura C.1 y además algunos archivos fundamentales para el proyecto.

En la mayoría de las carpetas del repositorio, excepto `Latex`, `airflow` y `utils`, encontraremos un archivo con nombre *Dockerfile*, que es el archivo de configuración del contenedor de Docker (Ver Sección 5.6). En el servicio Airflow no es necesario dicho archivo, pues se usa una imagen oficial. En la carpeta `utils` simplemente se encuentra un *script* que se corre periódicamente para renovar el certificado https, no siendo estrictamente necesario para el funcionamiento del proyecto.

```
/big_data_tfm/
├── Latex (Memoria del Proyecto)
│   ├── img
│   └── tex
├── ETL (Servicio de ETL) 4.3
├── ingestion_scrapper (Servicio de scraping) 4.1
│   └── ingestion_scrapper
│       ├── spiders
│       └── fixes
├── data_analysis (Servicio de Aprendizaje Automático) 4.6
├── airflow (DAGs de Airflow) 4.5
│   └── dags
├── backend (Backend del sitio web) 4.8
│   ├── controllers
│   ├── requests
│   └── util
├── frontend (Frontend del sitio web) 4.7
│   ├── public
│   └── src
├── utils
├── README.md (Documentación resumida)
├── architecture.drawio (Esquema de arquitectura)
└── docker-compose.yaml (Configuración de Docker Compose)
```

Figura C.1: Árbol de directorios General

C.3. Manual del programador

En esta sección se entrará más en detalle en como está estructurado y configurado el código, habrá una subsección para cada servicio (titulada con el nombre de la carpeta en el repositorio que aloja el código de dicho servicio) y una para las bases de datos.

Algunos aspectos generales que se mencionaran previamente para no repetirlos en cada subsección son los siguientes:

- `requirements.txt`: Se tratan de archivos que describen las dependencias del servicio. Generados mediante el comando `pip freeze > requirements.txt`. Posteriormente se usan en el flujo de Docker para instalar dependencias. Usado para servicios que usan Python.
- `package.json`: Archivos de dependencias para los servicios web, gestionados por `npm`. Usado para servicios que usan Javascript (web).
- `Dockerfile`: Archivo de configuración de la imagen de Docker.
- `readme.md`: Archivos con documentación del servicio.
- Scripts shell (`.sh`): Actúan como vínculo entre el orquestador (Apache Airflow) y el propio servicio. Generalmente lanzan archivos `.py` y guardan *logs*. Son ejecutados por el contenedor de Docker cuando este se lanza a través de Airflow.

ingestion_scrapper - Servicio de Scrapy

Este servicio está escrito completamente en Python y sigue el esquema a creado tras ejecutar el comando `scrapy startproject ingestion_scrapper`. En este proyecto los ficheros añadidos, modificados e inalterados respecto a la plantilla generada tras iniciar el proyecto, se muestran en la Figura C.2. Este servicio está pensado para funcionar con ejecuciones individualizadas, así, cada vez se lanza un contenedor de Docker que posteriormente se elimina. Con dicho contenedor, se puede lanzar tanto el servicio de recogida de datos de inmuebles (actualmente, 2 veces al día), como el servicio de comprobación de si los anuncios siguen activos (actualmente, 1 vez al día), según el *Script* de Shell que lance.

Algunos detalles que merece la pena mencionar con los siguientes:

- Carpeta `.../fixes/`: En esta carpeta se han recogido algunos scripts que se fueron utilizando a lo largo de los meses para adaptar el proyecto a nuevas funcionalidades. A día de hoy son prescindibles y no son necesarios para el funcionamiento del proyecto. Por ejemplo, encontramos un script `script_to_add_active_field.py`, que se usó para añadir el campo `active` a todos los inmuebles en la distintas colecciones de la MongoDB.
- Archivo `settings.py`: Desde aquí se modifica el comportamiento general de Scrapy
- Carpeta `.../spiders/`: Esta es la carpeta que realmente personaliza la funcionalidad del servicio de *Scraping*.

  Encontramos dos "arañas" distintas: una llamada `/spiders/pisos_spider.py`, que es la que contiene toda la lógica para recopilar la información de inmuebles y otra, `/spiders/check_up_spider.py`, que es la que contiene toda la lógica de verificar sin un anuncio sigue disponible o no.
- Los scripts de shell: `ingestion_script.sh`, `ad_up_checking_script.sh`. Son el punto único de enlace entre el orquestador y los dos tipos de tarea: La de recopilación/ingestión de inmuebles y la de comprobar si un anuncio sigue activo. Estos scripts generan logs que se almacenan en un volumen de Docker.

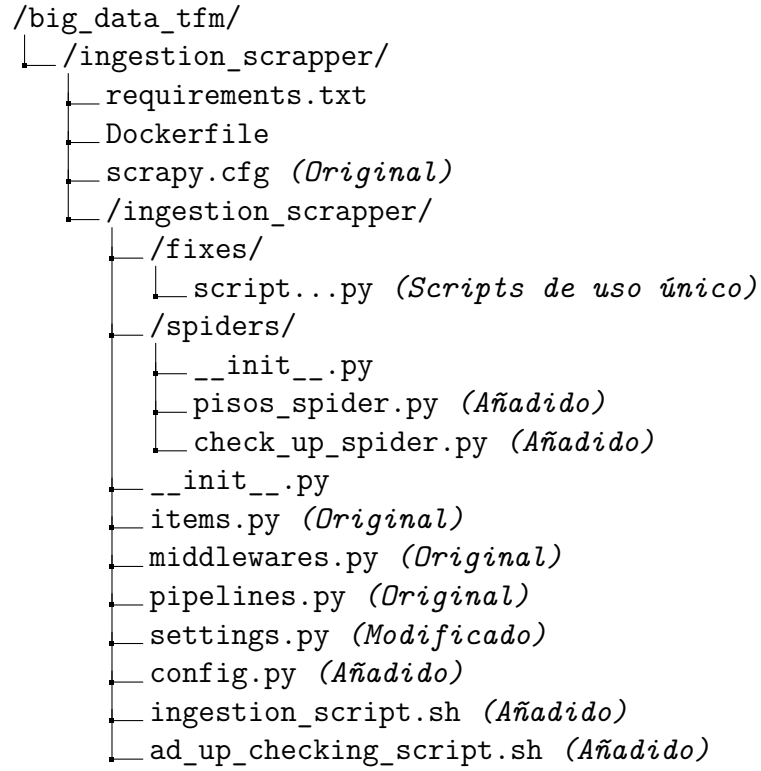

Figura C.2: Árbol de directorios de servicio *Scrapy*

```
/big_data_tfm/
└── /ETL/
    ├── requirements.txt
    ├── Dockerfile
    ├── config.py
    ├── aggregation.py (Principal Agregación)
    ├── extraction.py (ETL)
    ├── transformation.py (ETL)
    ├── transformation_utils.py (ETL)
    ├── loading.py (ETL)
    ├── main.py (Principal ETL)
    ├── transformation_script.sh (ETL)
    └── aggregation_script.sh (Agregación)
```

Figura C.3: Árbol de directorios de servicio de ETL y Agregación

ETL

Este servicio también está escrito completamente en Python, usando principalmente la librería Pandas [32]. Este servicio engloba dos tareas de las orquestadas por Airflow:

- Transformación (ETL en sí): Lee de las colecciones de MongoDB y da lugar a la tabla `pisos` en SQLite (Ver subsección B.2).
- Agregación: Lee de la tabla `pisos` y genera la tabla `pisos_dw` (Ver subsección B.2).

De igual forma que el servicio de *Scrapy*, este servicio está pensado para funcionar con ejecuciones individualizadas, lanzando un contenedor de Docker que realizará una función según el *Script* de Shell que lance. En la actualidad se ejecuta el servicio de transformación dos veces al día, seguido del de agregación. Se detallan los archivos en Figura C.3.

```
/big_data_tfm/
└── /data_analysis/
    ├── requirements.txt
    ├── Dockerfile
    ├── config.py
    ├── convert_db_to_csv.py .3 data_cleaning.py
    ├── model_generation.py
    ├── train.py
    ├── generate_predictions.py
    ├── predict.sh (Regresión de Precio)
    └── train.sh (Entrenamiento de modelos)
```

Figura C.4: Árbol de directorios de servicio de Aprendizaje Automático

data_analysis - Servicio de Aprendizaje Automático

Este servicio se basa en el paquete `scikit learn` [33] y está escrito en Python. Por una parte consta del código que entrena y genera los modelos, cuyo archivo principal es `train.py`, este utiliza funciones de `model_generation.py`, y crea y almacena un "limpiador/transformador" final de los datos utilizando `data_cleaning.py`. Este flujo de entrenamiento acaba con el almacenamiento permanente de los modelos y el "limpiador" en un volumen de Docker.

En el flujo de predicción, mucho más simple, encontramos solamente `generate_predictions.py`. Este flujo carga los modelos y el limpiador de datos para aplicarlos las entradas más recientes de la tabla `pisos`.

Los datos se leen directamente de la tabla `pisos` de SQLite tanto para el entrenamiento (en este caso se leen datos de como máximo 6 meses de tiempo), como para el flujo de predicción de valor/puntuación, en este caso se leen los datos que aún no tienen predicción asignada.

Como entrada al código de Python, para cada uno de los dos flujos, encontramos de nuevo dos *Scripts* de Shell: `predict.sh` y `train.sh`. Como pasaba para el servicio de *Scrapy* y ETL, este servicio funciona con la construcción de un contenedor de Docker temporal, que llama a uno de estos dos *Scripts* según la tarea y es orquestado por Airflow.

airflow - Servicio de Airflow

El servicio de Airflow se basa en la imagen `apache/airflow:2.7.1`. Así, se despliegan dos contenedores permanentes (a diferencia de los servicios anteriores), uno para *webserver* (interfaz web de Airflow) y otro para *scheduler*, para ver más detalles se puede ver el archivo `docker-compose.yaml` del proyecto.

Para iniciar correctamente el servicio por primera vez hay que seguir las instrucciones explicadas en Sección C.4.

Una vez en funcionamiento los contenedores y seguidas las instrucciones de instalación, los únicos archivos que encontramos en el proyecto son los relacionados con las DAGs [41], es decir los que indican las tareas a orquestar por Airflow, almacenados en `/big_data_tfm/airflow/dags/`. Estos archivos que configuran las DAGs se montan en los contenedores de Airflow.

Debemos tener en cuenta que todas las tareas de Airflow consisten en desplegar un contenedor de Docker temporal. Las podemos ver en Tabla C.1. Para más detalle se recomienda consultar el código de dichas DAGs.

DAG	**Frecuencia**	**Tareas**

`ingestion_and_ETL_dag.py`	2 por día	`ingestion_task`
		`checking_deletes_task` *(1/día)*
		`transformation_task`
		`aggregation_task`
		`prediction_task`

`train_dag.py`	1 por mes	`training_task` `prediction_task`
`mongo_backup_dag.py`	Cada 3 días	`mongo_backup_task`
`initial_run_dag.py`	Solo manual	Regenera el proyecto. Utiliza una MongoDB restaurada

Tabla C.1: Descripción de las DAGs en Apache Airflow

backend - Servicio *Backend* de la Web

Este servicio se compone de un servidor Node.js utilizando el framework Express, desarrollado completamente en JavaScript. El objetivo principal de este servicio es facilitar la comunicación entre la base de datos SQLite y la interfaz de usuario (*Frontend*) del sitio web.

Arquitectura y Componentes Clave: La arquitectura del servicio de backend está diseñada para ser modular, permitiendo una fácil expansión y mantenimiento (Ver Figura C.5). Los componentes clave incluyen:

- **Controllers**: Carpeta `/controllers` contiene archivos como `flats.js` y `trends.js`, que definen las rutas de la API y los controladores asociados a ellas. Estos controladores procesan las peticiones entrantes, interactúan con la base de datos y devuelven las respuestas adecuadas al cliente.
- **Utilidades**: La carpeta `/util` incluye archivos cruciales como `config.js` para la configuración del entorno, `db.js` que maneja la conexión y operaciones de la base de datos, `logger.js` para el registro de actividades y errores, utilizado por middleware `middleware.js` que contiene middleware para manejo de errores, y *logging*.
- **Configuración y Despliegue**: Archivos como `.dockerignore`, `.eslintrc.js`, y `.gitignore` configuran aspectos del entorno de desarrollo, *linting* y versionado, respectivamente. `Dockerfile` permite la contenerización del servicio para un despliegue eficiente y consistente.
- **Punto de Entrada**: `app.js` configura la aplicación Express, estableciendo middlewares y rutas principales, mientras que `index.js` sirve como el punto de entrada del servicio, iniciando el servidor y realizando configuraciones iniciales.
- **Gestión de Dependencias**: `package.json` y `package-lock.json` gestionan las dependencias del proyecto, asegurando versiones consistentes de las librerías y facilitando la instalación y actualización de paquetes.

Funcionamiento y Flujo de datos: Cuando una solicitud es recibida por el servidor, es procesada por el middleware correspondiente. Después, la solicitud es dirigida al controlador adecuado, donde se realizan operaciones específicas, como consultas o actualizaciones en la base de datos, antes de enviar una respuesta al cliente.

```
/big_data_tfm/
└── /backend/
    ├── /controllers/
    │   ├── flats.js
    │   └── trends.js
    ├── /util/
    │   ├── config.js
    │   ├── db.js
    │   ├── logger.js
    │   └── middleware.js
    ├── .dockerignore
    ├── .eslintrc.js
    ├── .gitignore
    ├── Dockerfile
    ├── app.js
    ├── index.js
    ├── package.json
    └── package-lock.json
```

Figura C.5: Árbol de directorios del *Backend* de la web

frontend - Servicio *Frontend* de la Web

El servicio *Frontend* está desarrollado utilizando React, una librería de JavaScript para construir interfaces web para el usuario. Se comunica con el servicio de *Backend* para obtener y enviar datos, lo que permite a los usuarios interactuar con la información almacenada en la base de datos de forma intuitiva. La estructura del proyecto *Frontend* se organiza según lo mostrado en la Figura C.6.

En la carpeta `/public/` encontraremos recursos estáticos, imágenes e iconos que se usan en la aplicación. Dentro de la carpeta `/src/`, encontraremos las siguientes subcarpetas y archivos importantes:

- `/assets/`: Contiene otros recursos estáticos
- `/components/`: Aquí se almacenan los componentes React, que son las unidades básicas de construcción de la interfaz de usuario. Cada componente (`.jsx`) representa una parte de la interfaz, como la página de inicio, la lista de pisos, las tendencias, el contacto y el pie de página. A su vez estas páginas están divididas en otros componentes

- `/services/`: Incluye servicios para la comunicación con el backend, como la obtención de datos de pisos y tendencias
- `/utils/`: Ofrece utilidades y funciones auxiliares, como opciones de selectores que son usadas a través de la aplicación
- `App.jsx`: El componente principal que engloba toda la aplicación React. En él se define la barra de navegación
- `index.css`: Hoja de estilos principal para la aplicación. No se ha utilizado demasiado, ya que se ha optado por dar la mayoría de los estilos dentro de los componentes.
- `main.jsx`: Punto de entrada de la aplicación React, donde se monta el componente principal `App`

Fuera de las carpetas `/public/` y `/src/`, encontramos varios archivos y configuraciones relevantes para el proyecto:

- `.dockerignore`, `.eslintrc.js`, `.gitignore`: Archivos de configuración para Docker, ESLint y Git, respectivamente, que ayudan a gestionar el entorno de desarrollo y control de versiones.
- `Dockerfile`: Define las instrucciones para crear una imagen Docker del servicio
- `package.json` y `package-lock.json`: Estos archivos gestionan las dependencias del proyecto, especificando las librerías y paquetes necesarios para el desarrollo y ejecución de la aplicación.
- `vite.config.js`: Configuración de Vite [46] específica para este proyecto, optimizando el proceso de construcción y desarrollo. `provinces.json`: Un archivo de datos utilizado por la aplicación, datos estáticos.

Utilizando Vite [46], un paquete para crear rápidamente plantillas para aplicaciones React, este proyecto *Frontend* aprovecha una compilación rápida y un servidor de desarrollo eficiente con autorecarga cuando se cambia algún componente, lo que resulta en una mejora significativa en la velocidad de desarrollo.

```
/big_data_tfm/
└── /frontend/
    ├── /public/
    │   ├── flats.jpg
    │   └── favicon.ico
    ├── /src/
    │   ├── /assets/
    │   │   └── react.svg
    │   ├── /components/
    │   │   ├── Home.jsx
    │   │   ├── Flats.jsx
    │   │   ├── Trends.jsx
    │   │   ├── Contact.jsx
    │   │   ├── Footer.jsx
    │   │   └── ....jsx
    │   ├── /services/
    │   │   ├── flats.js
    │   │   └── trends.js
    │   ├── /utils/
    │   │   └── selector_options.js
    │   ├── App.jsx
    │   ├── index.css
    │   └── main.jsx
    ├── .dockerignore
    ├── .eslintrc.js
    ├── .gitignore
    ├── Dockerfile
    ├── index.html
    ├── index.js
    ├── package.json
    ├── package-lock.json
    ├── vite.config.js
    ├── README.md
    └── provinces.json
```

Figura C.6: Árbol de directorios del *Frontend* de la web

Bases de datos: MongoDB y SQLite

La base de datos MongoDB: Su elección y utilidad se explica en la Sección 4.2, funciona con un contenedor de Docker que utiliza la última imagen disponible `mongo:latest`, y almacena los datos en un volumen (`mongodb-data`) montado en el contenedor. No necesita ningún tipo de instalación más que seguir el procedimiento de la Sección C.4

En el archivo `/big_data_tfm/readme.md`, existe una sección titulada `6. MongoDB` donde se explica como restaurar un *backup* de mongodb. Estos *backups* se generan cada 3 días si se activa el DAG `mongo_backup_dag.py` (Ver Tabla C.1). Para mover dicho *backup* del volumen `{project}_mongodb-backups` al sistema de archivos local se puede seguir la sección titulada `8. DEV-UTILS` del archivo `readme.md`.

Esta base de datos es la clave del proyecto, y utilizando un *backup* de la misma, se puede regenerar la base de datos SQLite, los modelos y continuar con el proceso de *scraping* desde un punto anterior, aunque hayan pasado múltiples días/semanas e incluso meses sin actualización.

La base de datos SQLite: Se trata únicamente de un archivo nombrado `pisos.db`, que se genera por primera vez, si no existe, tras ejecutar la tarea de ETL. Este archivo se almacena en un volumen de Docker que se monta en los distintos contenedores (Ver archivo `docker-compose.yaml` y observar volumen nombrado como `sqlite-db`).

Es extremadamente sustituible, esta base de datos completa se puede regenerar en pocos minutos a partir de una copia de la base de datos MongoDB, por ello no se realizan *backups*. Su mayor utilizad es dar rápido acceso a datos estructurados/tabulares por la web y los tareas de Aprendizaje Automático.

C.4. Compilación, instalación y ejecución del proyecto

En esta sección se abordará como ejecutar el proyecto completo en un entorno local. Aunque el proyecto está desplegado en una máquina virtual de Oracle Cloud, podría ser desplegado en cualquier máquina de forma similar, incluida la web.

Solo se abordará la ejecución mediante Docker, ya que es la única solución unificada. Sin embargo, sería totalmente posible la instalación y ejecución de cada uno de los servicios sin utilizar ningún tipo de contenedor, aunque no está explicada ni soportada en esta memoria por su extensión. Cabe señalar que hay mucha información ampliada en el archivo `/big_data_tfm/readme.md`, disponible para consulta si se quieren más detalles.

Los prerequisitos son los siguientes:

- Docker instalado
- Terminal Unix (Compatible WSL de Windows)
- Dar permisos de escritura y lectura en el socket de Docker para todos los usuarios. Esto es necesario para que Airflow utilice el socket.

  ```
  sudo chmod 666 /var/run/docker.sock
  ```

A continuación necesitamos completar algunos pasos imprescindibles, antes de tener el proyecto en funcionamiento:

- Clonar repositorio

  ```
  git clone https://github.com/dpuertamartos/big_data_tfm.git
  ```

- Iniciar la base de datos de Airflow

  ```
  docker-compose run -rm airflow_webserver airflow db init
  ```

- Crear usuario para Airflow. Por defecto, usuario `admin`, password `admin`.

  ```
  docker-compose run \
  --rm airflow_webserver airflow \
  users create \
  --username admin \
  ```

```
--firstname FIRST_NAME \
--lastname LAST_NAME \
--role Admin \
--email admin@example.org \
--password admin
```

- Construir contenedores para *Scraping*, ETL y Aprendizaje Automático. Estos contenedores son lanzados y destruidos tras cada ejecución, no están continuamente activos.

  `docker-compose build scraper etl data_analysis`

- Lanzar contenedores para Airflow, Mongodb y la web

  `docker-compose up -d mongodb airflow_webserver airflow_scheduler app backend nginx`

  Se debe tener en cuenta que la web en local no mostrará datos hasta que pasen unos días con el servicio de ingestión o bien se regeneren las bases de datos a partir de un *backup* (Explicado en subsección C.5).

- Acceder a interfaz de Airflow en el Navegador `http://0.0.0.0:8080/`:

  Como mínimo se debe activar la DAG `ingestion_transformation_dag`, que iniciará la recogida de datos.

  Se recomienda activar `training_dag`, solo si ya se tienen unas semanas de recogida de datos o si se ha partido de un *backup* de MongoDB y se ha lanzado ya `initial_run_dag`.

  Opcionalmente podemos activar `mongo_backup_dag` si queremos que se realice un backup del volumen que contiene los datos de MongoDB cada tres días.

  En la Figura C.7 se pueden ver las DAGs activadas en una ejecución del producto madurada durante varios meses.

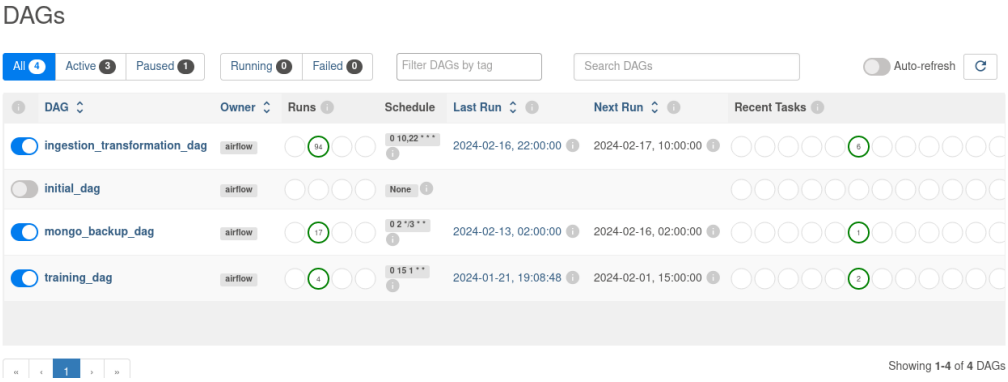

Figura C.7: Interfaz de Apache Airflow Web Server, con DAGs del proyecto activadas en la máquina del entorno de producción. Se activan pulsando el *slider* azul/gris a la izquierda del nombre.

C.5. Cómo regenerar todo el proyecto desde un *backup* de MongoDB

Primero, debemos restaurar los datos en MongoDB, el proceso de recuperación dependerá de como se ha generado el *backup*. Si se trata de un *backup* generada a través de la DAG de este proyecto destinada a tal causa, seguiremos los siguientes pasos después de clonar el repositorio (Se deben cumplir los prerrequisitos de la Sección C.4):

- Si existe un volumen previo con datos de MongoDB, lo borramos

  ```
  docker volume remove big_data_tfm_mongodb-data
  ```

- Iniciamos el contenedor de MongoDB, creando el volumen de datos asociado:

  ```
  docker compose up -d mongo
  ```

- Descomprimiremos el *backup* obteniendo una carpeta denominada *mongo_backup* o similar

- Copiaremos el *backup* dentro de una carpeta temporal del contenedor

  ```
  docker cp /path/to/backup/mongo_backup/ mongodb-container:/tmp/bkup
  ```

- Restauraremos el *backup* con el siguiente comando

  ```
  docker exec -it mongodb-container mongorestore /tmp/bkup
  ```

- Eliminamos la carpeta temporal del contenedor

  ```
  docker exec mongodb-container rm -rf /tmp/bkup
  ```

Posteriormente continuaremos con los pasos de la sección C.4 y al llegar al apartado de iniciar las DAGs, deberemos activar manualmente la DAG `initial_run_dag` una sola vez, esto regenerará la base de datos SQLite y se entrenaran los modelos (Puede tardar un par de horas según la máquina). Finalmente, tras el éxito de regenerar el resto de elementos a partir del *backup* de MongoDB, ya podemos activar las DAGs como aparece en la Figura C.7, tendremos todos los servicios orquestados y la web disponible con los datos hasta el punto que tuviese almacenado el *backup*.

Apéndice D

Documentación de usuario

D.1. Introducción

En este Apéndice se resumirá como interaccionar con la interfaz web que da acceso al conjunto de datos curado.

D.2. Requisitos e Instalación

Al tratarse de una aplicación web al uso, el usuario simplemente debe acceder con cualquier navegador con JavaScript activo (Chrome, Firefox, Edge...) al dominio buscahogar.es.

D.3. Manual del usuario

En esta sección se detallará como interactuar con las distintas vistas del sitio web: `INICIO`, `EXPLORA`, `VISUALIZA` e `INFO`. Tras acceder a la página, lo que vemos se observa en la Figura D.1

Se puede acceder a las distintas vistas del sitio web a través de la barra de navegación, mostrada en la parte superior derecha de la Figura D.1. En dicha barra de navegación, la vista seleccionada se remarca en color azul.

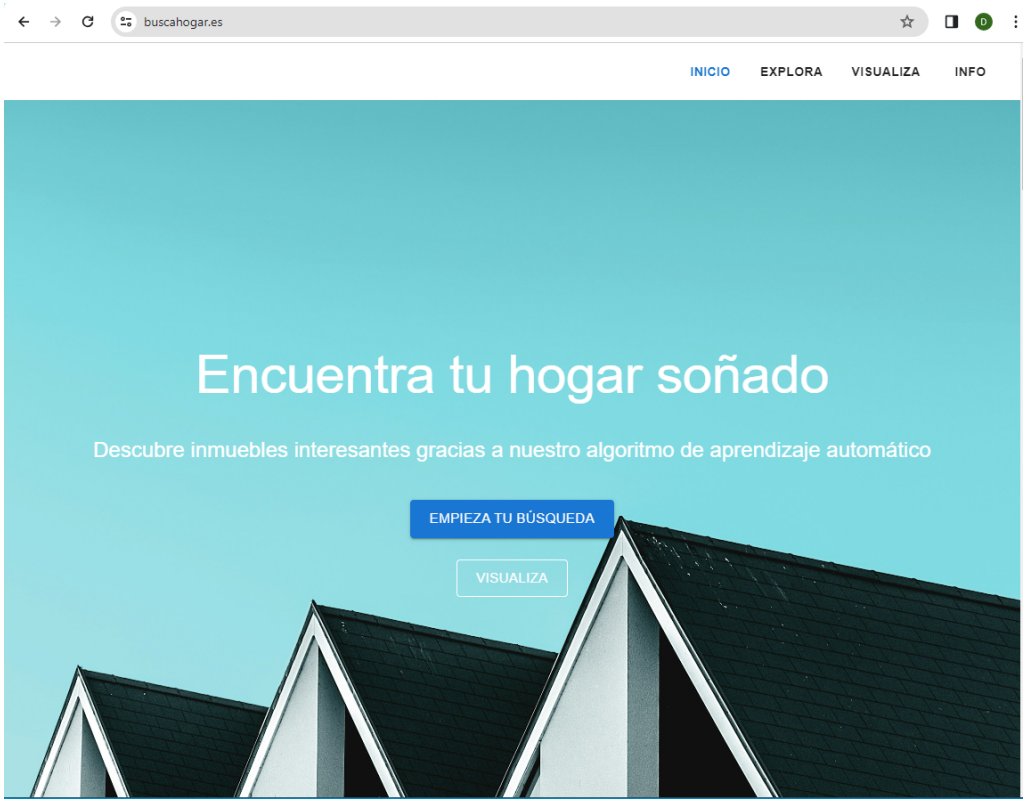

Figura D.1: Página de inicio `www.buscahogar.es`, en la parte superior derecha vemos la barra de navegación

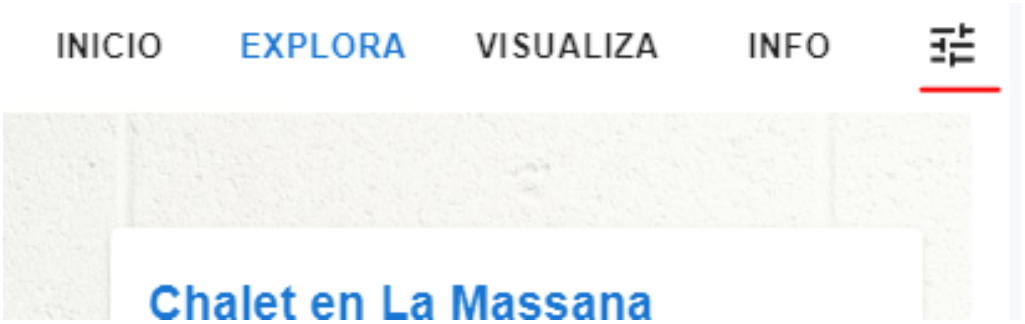

Figura D.2: Barra de navegación de `www.buscahogar.es` en pantallas móviles. Al pulsar el botón marcado en rojo se despliegan los filtros de las pestañas `INICIO`, `EXPLORA` y `VISUALIZA`.

INICIO

En esta vista se muestra la página de inicio, que hace la función de ventana de *marketing* para el resto del portal. Tras unos breves textos de "llamada a la acción", encontramos una sección con funcionalidad que merece la pena explicar (Ver Figura D.3):

En esta sección se muestran los inmuebles con mayor puntuación asignada por los modelos para las provincias seleccionadas (Ver calculo en Sección 4.6). Por defecto, aparecen los cinco inmuebles de mayor puntuación para todas las provincias y además, el modo inteligente (*Smart Mode*) se encuentra activado. Este modo limita la puntuación máxima a 0.4 (Lo que significa que los modelos le asignan un 40 % más de precio que el de venta), ya que se ha observado que inmuebles con más de esa puntuación suelen ser falsos positivos.

Por lo tanto, encontramos tres selectores: (1) Provincia, (2) Si queremos inmuebles solo en capital de provincia, (3) Modo Inteligente (limitación a 0.4 de puntuación máxima). Además, encontramos botón de interrogación que explica en qué consiste el modo inteligente. Estos selectores se repliegan en pantallas móviles, para desplegarlos hay que pulsar el icono de filtro en la parte superior derecha (ver Figura D.2).

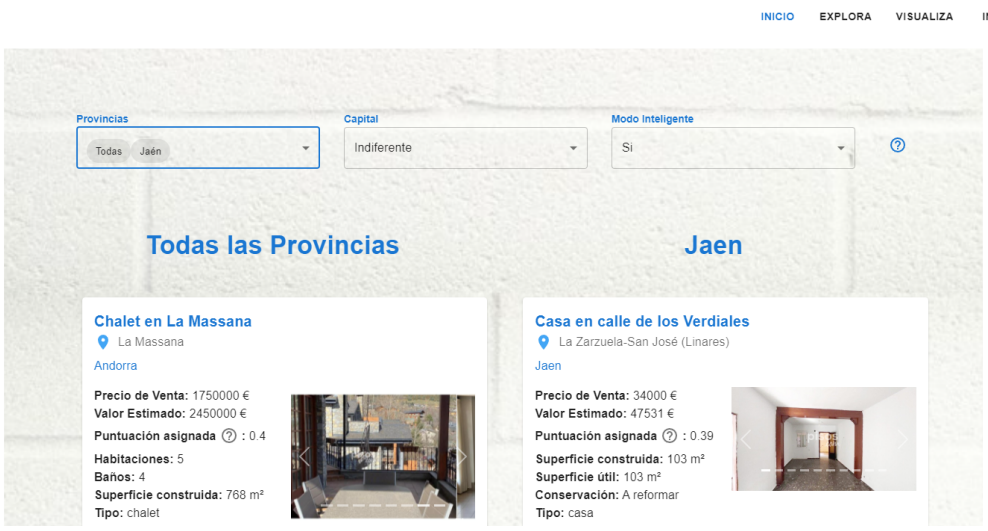

Figura D.3: Parte inferior de la página de INICIO en `www.buscahogar.es` con provincia "Todas" y "Jaen" seleccionadas

EXPLORA

En esta sección, que representa el núcleo de la web, encontramos el acceso al listado de todos los inmuebles en la base de datos. En la parte izquierda encontramos los filtros y en la derecha el listado de inmuebles (Ver Figura D.4. En pantallas móviles o de pequeño tamaño, los filtros de la parte izquierda se repliegan, para desplegarlos hay que pulsar el icono de filtro en la parte superior derecha (Ver Figura D.2).

Por defecto, los inmuebles están organizados en orden de puntuación descendente, con un máximo de 0.4 (equivalente al modo Inteligente, explicado en Sección D.3. Todos los inmuebles listados incluyen los datos originales listados en `www.pisos.com`, y además el valor estimado por los modelos de Aprendizaje Automático, junto con la puntuación (Para ver como se calcula ver 4.6), además de un botón con forma de interrogación que explica la puntuación asignada.

Los filtros, desde la parte superior a la inferior, son los siguientes:

- Provincia: La Provincia de la que quieres ver inmuebles
- Capital: Si quieres ver inmuebles solo en la capital de Provincia, fuera de la capital o indiferente.
- Tipo: El tipo de inmueble, puede ser indiferente. Piso, casa, apartamento...
- Precio (*Slider*): El rango de precios (en €) que queremos ver. Se trata del precio de venta real del inmueble.
- Habitaciones (*Slider*): Los inmuebles que se mostrarán estarán en ese rango de número de habitaciones que queremos ver.
- M2 Útiles (*Slider*): Los inmuebles que se mostrarán estarán en ese rango de metros cuadrados útiles que queremos ver.
- Puntuación (*Slider*): Los inmuebles que se mostrarán estarán en ese rango de puntuaciones (asignadas por Aprendizaje Automático)
- Ordenar por: La forma en la que quieres ordenar el listado, por defecto en puntuación descendente.

De forma interesante, la pestaña explora no incorpora paginación como suele ocurrir en los portales inmobiliarios comunes, se ha implementado un

mecanismo de *scroll* infinito similar al de la mayoría de redes sociales, en el que se van cargando inmuebles que cumplen las especificaciones del filtro a medida que el usuario alcanza el límite inferior de la página.

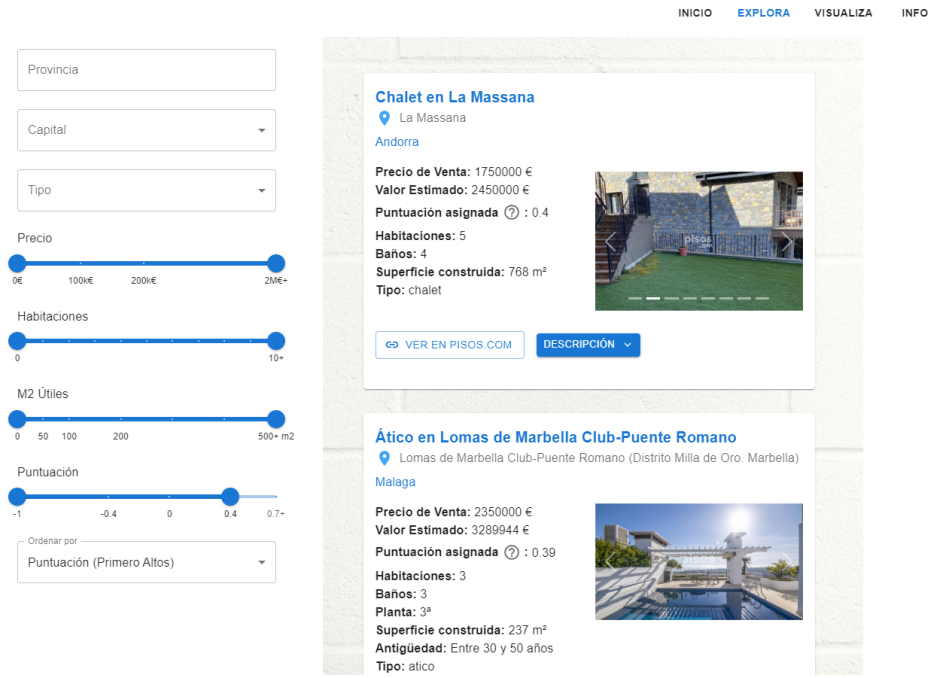

Figura D.4: Vista general de página EXPLORA en www.buscahogar.es

VISUALIZA

Esta pestaña es la zona de visualización de los datos agregados en la tabla explicada en el Apéndice B.2.

Se presentan tres zonas claramente diferenciadas, una superior con filtros para las gráficas, una media con gráfica(s) de líneas y una inferior con un gráfico de barras y un mapa. En pantallas móviles o de pequeño tamaño, los filtros superiores se repliegan, para mostrarlos hay que pulsar el icono de filtro en la parte superior derecha (Ver Figura D.2).

Los filtros son los siguientes:

- Comunidad Autónoma (CA): La CA de la que se quieren ver datos. Al seleccionarla, automáticamente se seleccionan todas las Provincias de dicha CA. Selección múltiple
- Provincia: La Provincia de la que se quieren ver inmuebles. Selección múltiple.
- Capital: Si se quieren ver inmuebles solo en la capital de Provincia, fuera de la capital o indiferente.
- Tipo: El tipo de inmueble, puede ser indiferente. Piso, casa, apartamento...
- Activo: Si se quieren datos de anuncios activos, retirados/vendidos o indiferente.

En la gráficas de líneas muestran datos promediados (eje Y), denominados como `Medidas` en la interfaz. Se observa la evolución de dichas medidas a lo largo de los meses (eje X). Como máximo se pueden seleccionar tres a la vez, mostrándose tres gráficas de líneas, en las que cada línea es una CA/Provincia/Media Total. Dichos datos son los siguientes actualmente:

- Precio (€)
- Superficie útil (m^2)
- Superficie de solar (m^2)
- Número de habitaciones (ud)
- Número de baños (ud)

- Gastos de comunidad (€)
- Cantidad de inmuebles (anuncios) en venta / extraídos (ud)
- Precio por Metro Cuadrado (€)
- Precio por Habitación (€)
- Precio por Baño (€)

En la parte inferior del gráfico (parte denominada como "categórica"), se muestran (en %), los valores que toman las categorías seleccionadas.

Por ejemplo, si seleccionamos categoría `Piscina Sí (%)` se nos muestra el % de inmuebles de cada comunidad/provincia/media total que tienen Piscina (Ver Figura D.5). En el gráfico de barras se muestra una barra con el porcentaje, y en el mapa aparece un círculo localizado en dicha provincia con área proporcional al porcentaje. Si se ha seleccionado una comunidad autónoma, en el gráfico de barras aparece la media de la comunidad autónoma y en el mapa todas las provincias de dicha comunidad autónoma.

INFO

Se trata de una sección de contacto, con el e-mail del autor principal del presente proyecto y algunas preguntas y respuestas frecuentes.

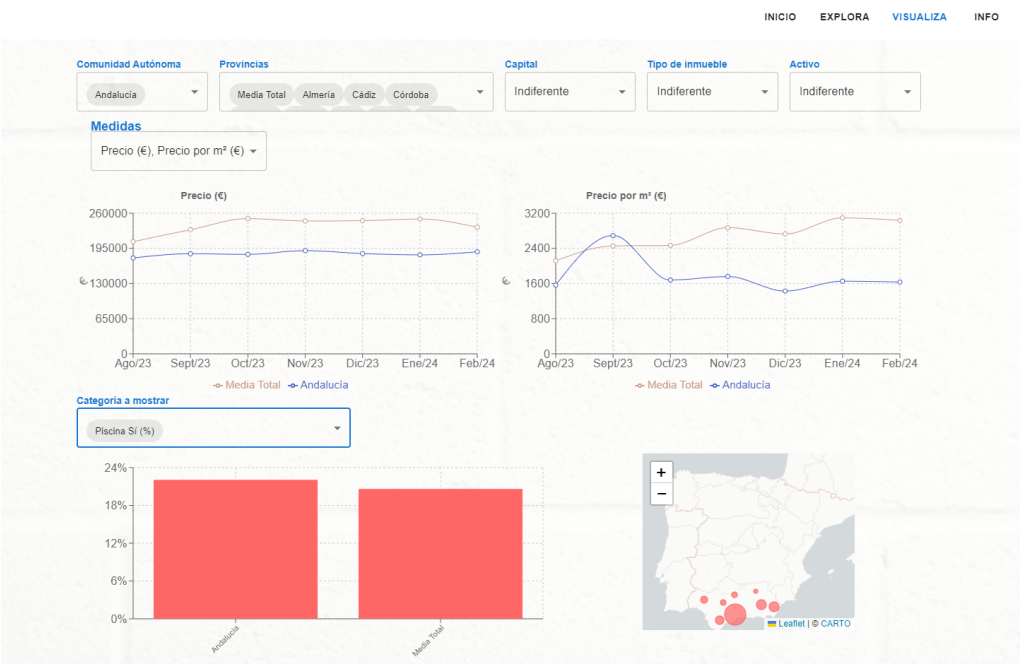

Figura D.5: Vista general de página VISUALIZA en `www.buscahogar.es`. Se ha seleccionado la comunidad autónoma Andalucía, y la categoría a mostrar `Piscina Si (%)`

Bibliografía

[1] Amazon Web Services. Página Web de AWS Step Functions. https://aws.amazon.com/es/step-functions/. [Último Acceso: 31 Enero 2024].

[2] Amazon Web Services. Precios oficiales de EBS Volume Storage. https://aws.amazon.com/ebs/pricing/, 2024. [Último Acceso: 8 Febrero 2024].

[3] Apache. Página Web de Apache Airflow. https://airflow.apache.org/. [Último Acceso: 31 Enero 2024].

[4] Apache. Spark hardware recommendations. https://spark.apache.org/docs/0.9.0/hardware-provisioning.html, 2024. [Último Acceso: 31 Enero 2024].

[5] Faisal Aqlan and Joshua Nwokeji. Big data etl implementation approaches: A systematic literature review. 07 2018.

[6] Alejandro Baldominos, Iván Blanco, Antonio José Moreno, Rubén Iturrarte, Óscar Bernárdez, and Carlos Afonso. Identifying real estate opportunities using machine learning. *Applied sciences*, 8(11):2321, 2018.

[7] Beautiful Soup Labs. Documentación sobre Beautiful Soup. https://www.crummy.com/software/BeautifulSoup/bs4/doc/. [Último Acceso: 22 Febrero 2024].

[8] Simin Cai, Barbara Gallina, Dag Nyström, and Cristina Seceleanu. Data aggregation processes: a survey, a taxonomy, and design guidelines. *Computing*, 101, 10 2019.

[9] Tianfeng Chai and Roland R Draxler. Root mean square error (rmse) or mean absolute error (mae). *Geoscientific model development discussions*, 7(1):1525–1534, 2014.

[10] Anjali Chauhan. A review on various aspects of mongodb databases. *Int. J. Eng. Res. Sci. Technol*, 8(5):90–92, 2019.

[11] E. Codd. A relational model for large shared data banks. *Communications of the ACM*, 13:377–, 06 1970.

[12] Dagster Labs. Página Web de Dagster. https://dagster.io/. [Último Acceso: 31 Enero 2024].

[13] Docker Labs. Documentación sobre Docker Compose documentation. https://docs.docker.com/compose/. [Último Acceso: 1 Febrero 2024].

[14] Express Labs. Página Web de Express WebPage. https://expressjs.com/es/. [Último Acceso: 1 Febrero 2024].

[15] Musa Garba. A comparison of nosql and relational database management systems (rdbms). 12 2020.

[16] GNU. GNU General Public License v3.0. https://www.gnu.org/licenses/gpl-3.0.en.html. [Último Acceso: 8 Febrero 2024].

[17] GNU. How to choose a license for your own work. https://www.gnu.org/licenses/license-recommendations.en.html. [Último Acceso: 8 Febrero 2024].

[18] Idealista. Análisis del mercado inmobiliario de 2023. https://www.idealista.com/news/inmobiliario/vivienda/2023/11/30/809560-el-analisis-del-mercado-inmobiliario-en-2023-y-que-se-preve-para-2024, 2023. [Último Acceso: 17 Febrero 2024].

[19] W.H Inmon. *Building the Data Warehouse*. 01 1992.

[20] Jupyter Labs. Página Web de Jupyter Notebooks. https://jupyter.org/. [Último Acceso: 1 Febrero 2024].

[21] Moaiad Khder. Web scraping or web crawling: State of art, techniques, approaches and application. *International Journal of Advances in Soft Computing and its Applications*, 13:145–168, 12 2021.

[22] Neal Leavitt. Will nosql databases live up to their promise? *Computer*, 43:12 – 14, 03 2010.

[23] Batta Mahesh. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9(1):381–386, 2020.

[24] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[25] Meta. Documentación sobre Componentes de React. https://react.dev/reference/react/Component. [Último Acceso: 1 Febrero 2024].

[26] Meta. Página Web de React. https://es.react.dev/. [Último Acceso: 31 Enero 2024].

[27] MongoDB Labs. Página Web de MongoDB WebPage. https://www.mongodb.com/. [Último Acceso: 31 Enero 2024].

[28] Adrian Mouat. *Using Docker: Developing and deploying software with containers*. .ºReilly Media, Inc.", 2015.

[29] Jiafei Niu and Peiqing Niu. An intelligent automatic valuation system for real estate based on machine learning. In *Proceedings of the international conference on artificial intelligence, information processing and cloud computing*, pages 1–6, 2019.

[30] Node.js Labs. Página Web de Node.js. https://nodejs.org/en, 2024. [Último Acceso: 1 Febrero 2024].

[31] Oracle. Documentación sobre Oracle Cloud Compute. https://www.oracle.com/cloud/compute/, 2024. [Último Acceso: 1 Febrero 2024].

[32] Pandas Labs. Página Web de Pandas. https://pandas.pydata.org/. [Último Acceso: 31 Enero 2024].

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[34] Pisos.com. Archivo robots.txt del Portal pisos.com. https://www.pisos.com/robots.txt. [Último Acceso: 8 Febrero 2024].

[35] Sebastian Raschka and Vahid Mirjalili. *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing Ltd, 2019.

[36] Will Reese. Nginx: the high-performance web server and reverse proxy. *Linux Journal*, 2008(173):2, 2008.

[37] Scrapy Labs. Documentación de la Arquitectura de Scrapy. https://docs.scrapy.org/en/latest/topics/architecture.html. [Último Acceso: 31 Enero 2024].

[38] Scrapy Labs. Página Web de Scrapy WebPage. https://scrapy.org/. [Último Acceso: 31 Enero 2024].

[39] Selenium Labs. Documentación sobre Selenium. https://selenium-python.readthedocs.io/. [Último Acceso: 22 Febrero 2024].

[40] Pramila P Shinde and Seema Shah. A review of machine learning and deep learning applications. In *2018 Fourth international conference on computing communication control and automation (ICCUBEA)*, pages 1–6. IEEE, 2018.

[41] B Shubha and A Prasad. Airflow directed acyclic graph. *J Signal Process*, 5(2).

[42] Spotify. GitHub de Luigi. https://github.com/spotify/luigi. [Último Acceso: 31 Enero 2024].

[43] SQLite Labs. Página Web de SQLite. https://www.sqlite.org/index.html. [Último Acceso: 31 Enero 2024].

[44] Madiha H Syed, Eduardo B Fernandez, et al. The software container pattern. In *Proceedings of the 22nd Conference on Pattern Languages of Programs*, pages 24–26, 2015.

[45] Twisted Labs. Página Web de Twisted WebPage. https://twisted.org/. [Último Acceso: 31 Enero 2024].

[46] Vite Labs. Documentación sobre Vite React. https://vitejs.dev/guide/. [Último Acceso: 17 Febrero 2024].