PYTHON, IA

HousePrice App

Jéssica Marrón Carmona

New Technology School

Tokio.

New
Technology
School

JÉSSICA MARRÓN CARMONA

RESPONSABLE DEL DESARROLLO DE LA APLICACIÓN

Fecha: FEBRERO 2026

PYTHON

PROYECTO FINAL. HousePrice App

Jéssica Marrón Carmona

Tokio.

ÍNDICE DE CONTENIDOS

New
Technology
School

SEGUNDA PARTE: INTELIGENCIA ARTIFCIAL, MACHINE LEARNING

1. DEFINICIÓN DEL PROBLEMA Y OBJETIVOS
2. DESCRIPCIÓN DEL CONJUNTO DE DATOS
3. LIMPIEZA Y PREPROCESADO DE DATOS
4. ANÁLISIS EXPLORATORIO DE LOS DATOS (EDA)
5. MODELOS ENTRENADOS Y METODOLOGÍA
6. RESULTADOS Y EVALUACIÓN FINAL
7. CONCLUSIONES FINALES

INTEGRACIÓN DEL MODELO IA EN LA APLICACIÓN PYTHON HOUSEPRICE APP

PRIMERA PARTE: PYTHON

1. Descripción general del proyecto

Este proyecto consiste en una aplicación web desarrollada en Python con Flask cuyo objetivo es estimar el precio de venta de una vivienda a partir de características introducidas por el usuario.

La aplicación integra un modelo de Machine Learning entrenado previamente con un dataset de viviendas (Kaggle – Ames Housing).

La app incluye un sistema de autenticación con control de accesos, distinguiendo entre usuarios normales y administradores.

- Los usuarios pueden iniciar sesión, realizar predicciones y consultar su propio historial.

- Los administradores disponen de un panel adicional desde el que pueden visualizar todas las predicciones realizadas en el sistema.

Además, cada predicción se almacena en una base de datos SQLite para garantizar persistencia y trazabilidad de las consultas.

2. Objetivos del proyecto

El objetivo principal de este proyecto es desarrollar una aplicación web funcional en Python que permita estimar el precio de una vivienda a partir de sus características principales, integrando un modelo de inteligencia artificial previamente entrenado.

De forma más específica, los objetivos del proyecto son:

- Diseñar e implementar una aplicación web utilizando el framework Flask.
- Integrar un modelo de Machine Learning capaz de realizar predicciones de precios de vivienda.
- Implementar un sistema de autenticación de usuarios con diferentes roles (usuario normal y administrador).
- Permitir a los usuarios registrados realizar predicciones y consultar su historial personal.

- Almacenar las predicciones realizadas en una base de datos para garantizar persistencia.
- Desarrollar un panel de administración desde el cual se puedan consultar todas las predicciones realizadas en la aplicación.
- Aplicar buenas prácticas de programación en Python, como modularidad y separación de responsabilidades.

3. Alcance del proyecto

El alcance del proyecto incluye:

- Desarrollo de una aplicación web con Flask.
- Uso de una base de datos SQLite para almacenar usuarios y predicciones.
- Integración de un modelo de Machine Learning entrenado previamente.
- Implementación de formularios web para la introducción de datos.
- Gestión de usuarios con roles diferenciados.
- Visualización del historial de predicciones para usuarios y administradores.

Quedan fuera del alcance del proyecto:

- El despliegue de la aplicación en un entorno de producción.
- La optimización avanzada del rendimiento del modelo.
- La incorporación de pagos, servicios externos o APIs de terceros.
- La gestión avanzada de permisos más allá de los roles definidos.

4. Stack tecnológico y justificación

Stack tecnológico utilizado

Para el desarrollo del proyecto se ha utilizado el siguiente conjunto de tecnologías:

- Python como lenguaje principal de programación.
- Flask como framework web para el desarrollo de la aplicación.

- SQLite como sistema gestor de base de datos.
- SQLAlchemy como ORM para la gestión de la base de datos.
- Flask-Login para la gestión de sesiones y autenticación de usuarios.
- scikit-learn para el entrenamiento y uso del modelo de Machine Learning.
- joblib para la serialización y carga del modelo entrenado.
- HTML y CSS para el desarrollo de la interfaz de usuario.
- Jinja2 como motor de plantillas integrado en Flask.

Justificación de las tecnologías

- Python se ha elegido por su versatilidad, claridad sintáctica y amplio ecosistema de librerías, especialmente en el ámbito del desarrollo web y la inteligencia artificial.
- Flask es un framework web ligero que permite desarrollar aplicaciones de forma modular y sencilla, facilitando la integración de modelos de Machine Learning.
- SQLite resulta adecuado para proyectos académicos por su simplicidad y por no requerir un servidor de base de datos externo.
- SQLAlchemy permite trabajar con la base de datos de forma orientada a objetos, facilitando el mantenimiento del código y evitando el uso directo de consultas SQL complejas.
- Flask-Login se utiliza para gestionar la autenticación y el control de acceso de los usuarios de forma segura.
- scikit-learn proporciona las herramientas necesarias para el preprocesado de datos, el entrenamiento y la evaluación de modelos de Machine Learning.
- joblib permite guardar y cargar el modelo entrenado de forma eficiente.
- HTML y CSS se emplean para crear una interfaz web sencilla, clara y usable.
- Jinja2 facilita la reutilización de plantillas y la generación dinámica de contenido en la aplicación web.

5. Modelo de datos

5.1 Descripción del modelo de datos

La aplicación utiliza una base de datos SQLite formada por dos tablas principales:

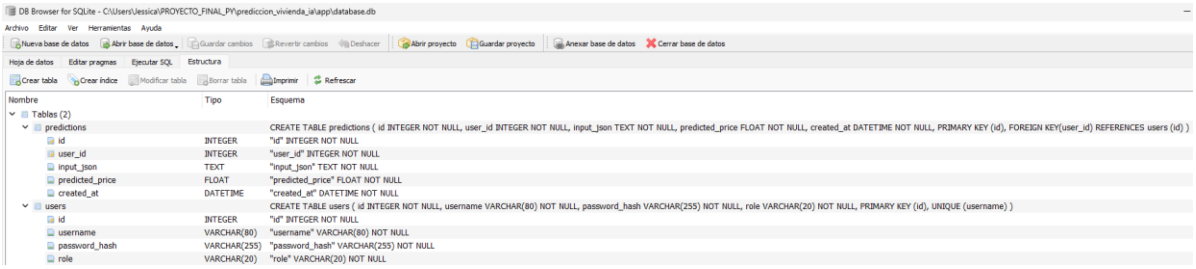
- users: almacena la información de los usuarios que acceden a la aplicación (credenciales y rol).
- predictions: almacena el historial de predicciones realizadas, asociadas a un usuario.

La relación entre ambas tablas es de tipo 1:N, es decir: un usuario puede tener muchas predicciones, pero cada predicción pertenece a un único usuario.

5.2 Esquema de la base de datos y relaciones entre entidades

Estructura de la base de datos SQLite utilizada en la aplicación, donde se observa la relación entre la tabla de usuarios y la tabla de predicciones.

La base de datos utilizada es SQLite y está compuesta por dos tablas principales: users y predictions. La relación entre ambas tablas es de uno a muchos, ya que un usuario puede realizar múltiples predicciones.



Nombre	Tipo	Esquema
Tablas (2)		
predictions		CREATE TABLE predictions (id INTEGER NOT NULL, user_id INTEGER NOT NULL, in
id	INTEGER	"id" INTEGER NOT NULL
user_id	INTEGER	"user_id" INTEGER NOT NULL
input_json	TEXT	"input_json" TEXT NOT NULL
predicted_price	FLOAT	"predicted_price" FLOAT NOT NULL
created_at	DATETIME	"created_at" DATETIME NOT NULL
users		CREATE TABLE users (id INTEGER NOT NULL, username VARCHAR(80) NOT NULL,
id	INTEGER	"id" INTEGER NOT NULL
username	VARCHAR(80)	"username" VARCHAR(80) NOT NULL
password_hash	VARCHAR(255)	"password_hash" VARCHAR(255) NOT NULL
role	VARCHAR(20)	"role" VARCHAR(20) NOT NULL

La tabla *users* almacena la información de autenticación y el rol del usuario, mientras que la tabla *predictions* almacena los datos de entrada utilizados en la predicción, el valor predicho y la fecha de creación.

5.3 Definición de tablas (SQL)

```
CREATE TABLE users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  username TEXT NOT NULL UNIQUE,
  password_hash TEXT NOT NULL,
  role TEXT NOT NULL DEFAULT 'user'
);

CREATE TABLE predictions (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  user_id INTEGER NOT NULL,
  input_json TEXT NOT NULL,
  predicted_price REAL NOT NULL,
  created_at TEXT NOT NULL,
  FOREIGN KEY (user_id) REFERENCES users(id)
);
```

6. Requisitos funcionales de la aplicación

6.1 Requisitos funcionales generales

La aplicación permite a los usuarios realizar predicciones del precio de una vivienda a partir de un formulario web, integrando un modelo de Machine Learning. La aplicación incluye autenticación y control de acceso por roles, y almacena todas las predicciones en una base de datos SQLite.

6.2 Requisitos funcionales para usuarios normales (rol: user)

Un usuario normal puede:

- Registrarse en la aplicación mediante un formulario de alta.
- Iniciar sesión con sus credenciales para acceder a la funcionalidad principal.
- Acceder al formulario de predicción (modo Express y modo Avanzado opcional).
- Introducir los datos de la vivienda y solicitar una predicción.
- Visualizar el resultado de la predicción en pantalla.
- Consultar su historial personal, donde se muestran sus predicciones anteriores con fecha, precio estimado e inputs utilizados.
- Cerrar sesión de forma segura.

Restricción:

- Un usuario normal no puede acceder al historial de otros usuarios ni al panel de administración.

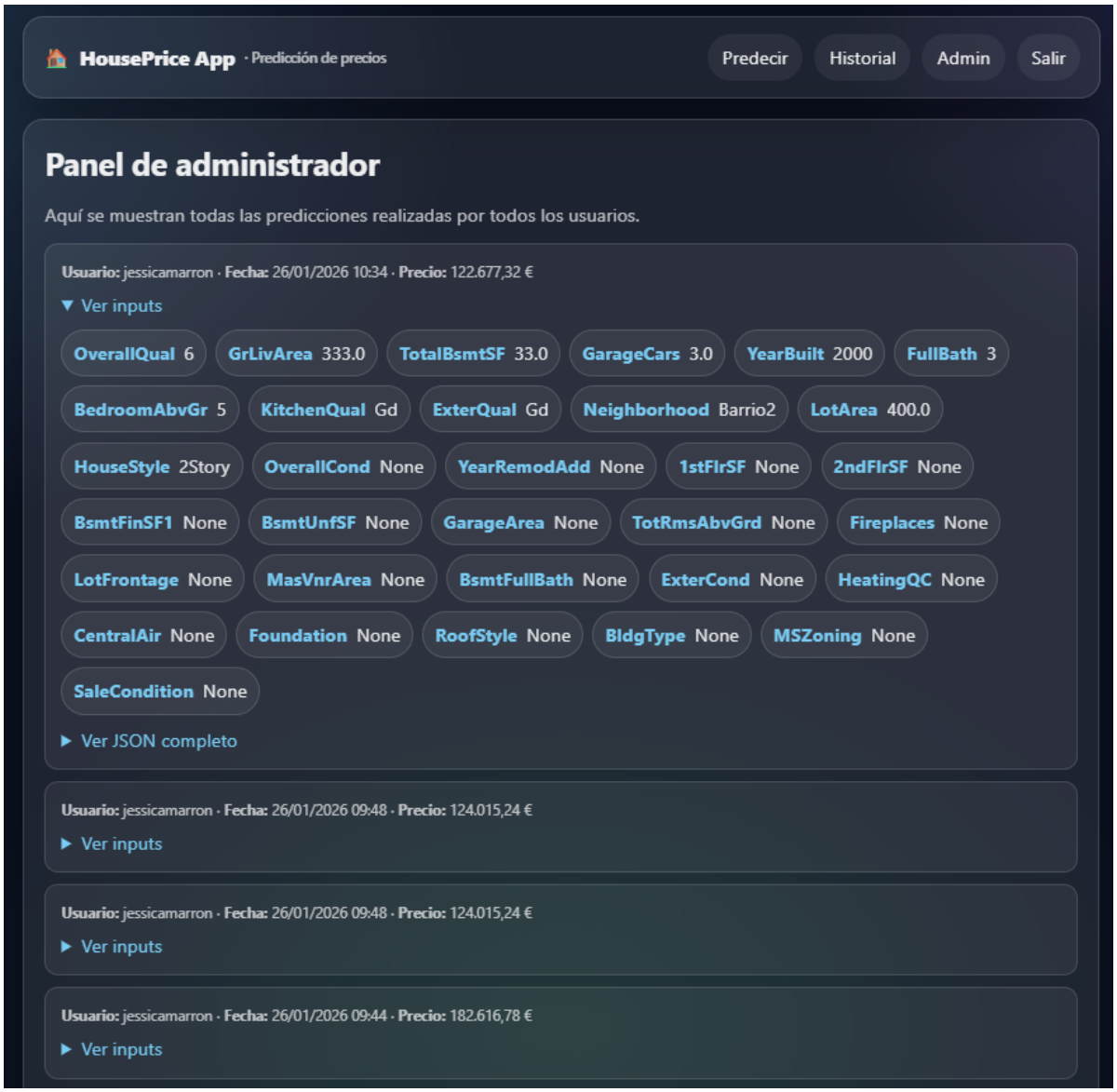
6.3 Requisitos funcionales para administradores (rol: admin)

Además de todas las funcionalidades de un usuario normal, un administrador puede:

- Acceder a un panel de administración donde se muestran todas las predicciones almacenadas en el sistema.
- Consultar predicciones de cualquier usuario, incluyendo fecha, precio e inputs introducidos.
- Aplicar filtros de consulta (por ejemplo, filtrar por usuario) para facilitar la revisión de registros.

Restricción:

- El panel de administración solo es accesible para usuarios con rol admin.



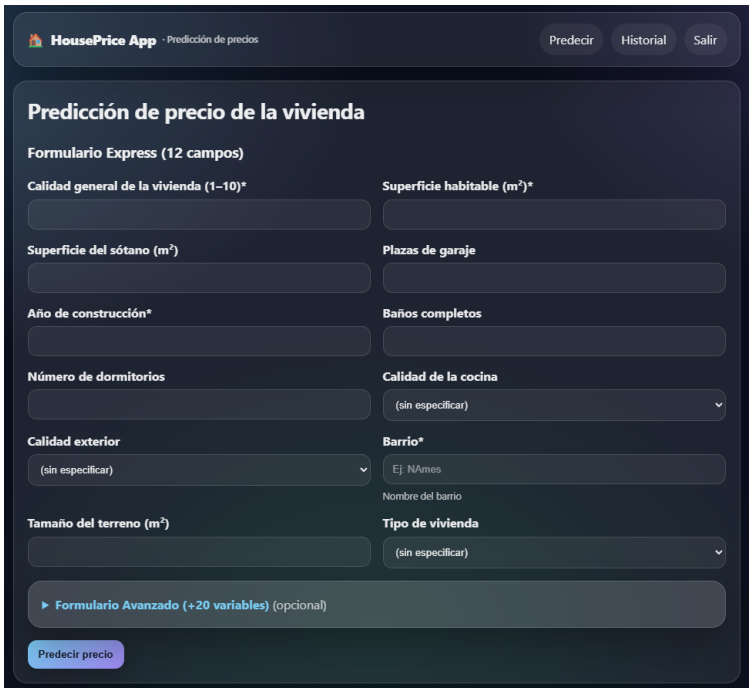
6.4 Persistencia de datos

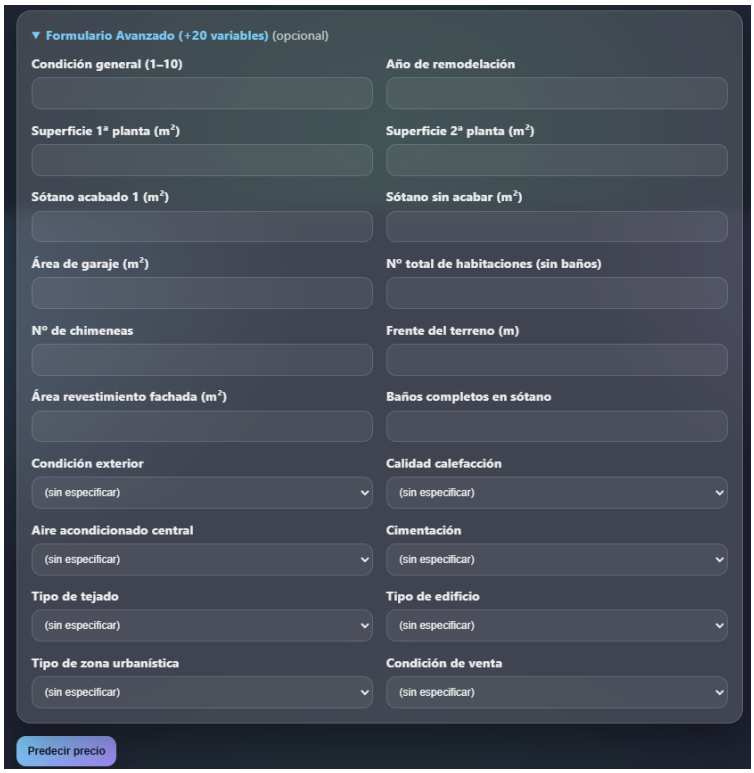
- Cada predicción realizada se guarda en la base de datos SQLite en la tabla predictions.
- Los usuarios se almacenan en la tabla users, incluyendo su rol (user o admin).
- Las predicciones se relacionan con el usuario mediante la clave foránea predictions.user_id → users.id.

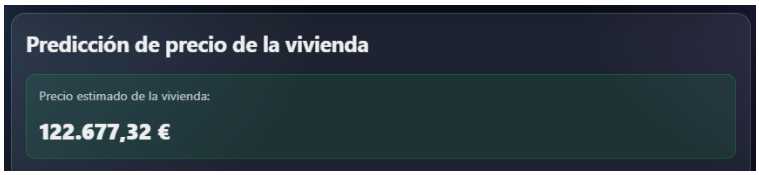
	id	user_id	input_json	predicted_price	created_at
	Filtro	Filtro	Filtro	Filtro	Filtro
1	1	1	{"OverallQual": 10, "GrLivArea": …	235213.463333333	2026-01-26 08:16:11.830954
2	2	1	{"OverallQual": 10, "GrLivArea": …	241462.123333333	2026-01-26 08:16:31.242715
3	3	1	{"OverallQual": 10, "GrLivArea": …	241462.123333333	2026-01-26 08:17:00.827968
4	4	2	{"OverallQual": 5, "GrLivArea": …	102295.46	2026-01-26 09:34:14.224889
5	5	2	{"OverallQual": 8, "GrLivArea": …	182616.783333333	2026-01-26 09:44:37.935719
6	6	2	{"OverallQual": 6, "GrLivArea": …	124015.24	2026-01-26 09:48:08.148300
7	7	2	{"OverallQual": 6, "GrLivArea": …	124015.24	2026-01-26 09:48:41.917091

6.5 Interfaz y experiencia de usuario

- La aplicación incluye un formulario en español para facilitar la introducción de datos.
- Se ofrecen dos modos de uso (esta separación permite equilibrar usabilidad y precisión del modelo, adaptándose a distintos perfiles de usuario:
    - Formulario Express (campos esenciales).
    - Formulario Avanzado (campos opcionales adicionales).
- El resultado de la predicción se muestra de forma destacada para evitar que el usuario tenga que buscarlo en la página.







7. Manual de instalación y ejecución

7.1 Requisitos previos

Para ejecutar la aplicación es necesario disponer de:

- **Python 3.x** instalado (recomendado 3.11 o superior).
- Un editor como **VS Code** (opcional, pero recomendado).
- Acceso al proyecto con su estructura de carpetas.

7.2 Preparación del entorno

1. Abrir una terminal en la carpeta raíz del proyecto.
2. Crear un entorno virtual:

Windows (PowerShell):

python -m venv .venv

3. Activar el entorno virtual:

.\.venv\Scripts\Activate.ps1

4. Actualizar pip:

python -m pip install --upgrade pip

7.3 Instalación de dependencias

Instalar los paquetes necesarios desde requirements.txt:

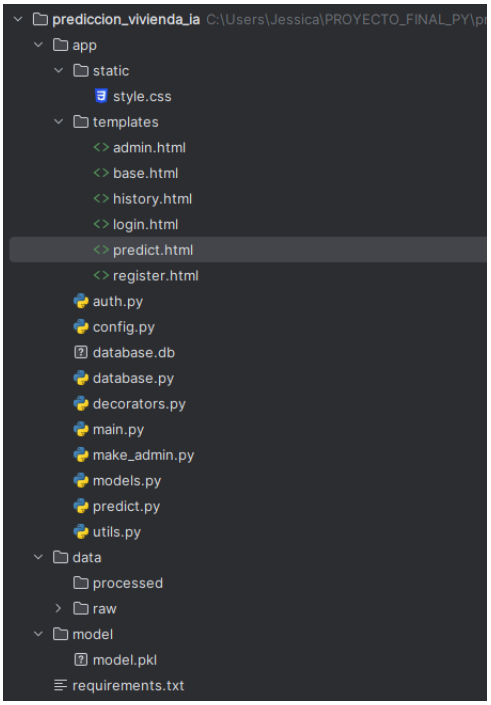
pip install -r requirements.txt

En caso de no disponer de requirements.txt, instalar manualmente los principales paquetes:

pip install flask flask_sqlalchemy flask_login werkzeug joblib pandas scikit-learn

7.4 Estructura del proyecto

La aplicación está organizada de forma modular:



- app/ → código principal de Flask (rutas, modelos, BD, plantillas)
- app/templates/ → vistas HTML (login, registro, predicción, historial, admin)
- app/static/ → CSS y recursos visuales
- model/ → modelo entrenado serializado (model.pkl)
- data/raw/ → dataset original utilizado (train.csv, test.csv)

7.5 Ejecución de la aplicación

Con el entorno virtual activado, ejecutar:

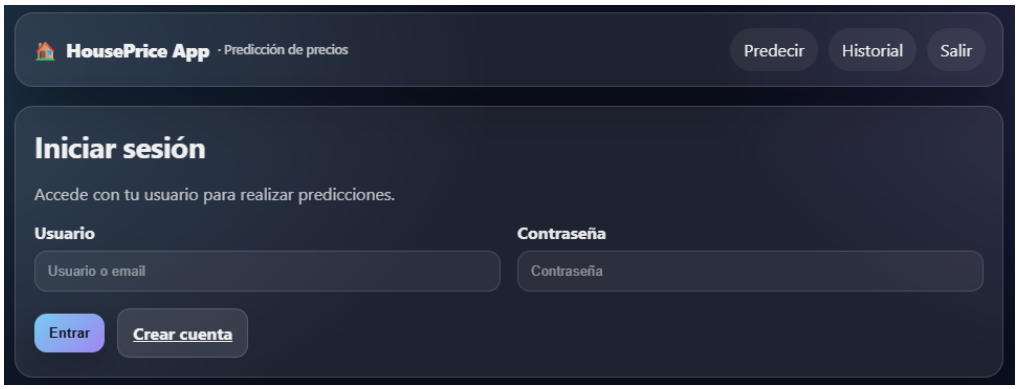
python app/main.py

Al iniciar, Flask mostrará una URL similar a:

- http://127.0.0.1:5000/

7.6 Uso básico de la aplicación

1. Acceder a la aplicación desde el navegador.
2. Crear un usuario desde **Crear cuenta**.
3. Iniciar sesión con el usuario creado.
4. Acceder al apartado **Predecir** para realizar una predicción.
5. Consultar el apartado **Historial** para ver predicciones previas.



7.7 Creación de un administrador

Por defecto, los usuarios registrados tienen rol user. Para convertir un usuario a administrador se utiliza el script make_admin.py.

1. Ejecutar el script:

python app/make_admin.py

2. Introducir el nombre exacto del usuario cuando se solicite.
3. El sistema actualizará su rol a admin.

Tras esto, al iniciar sesión con ese usuario aparecerá el acceso al panel **Admin**.

7.8 Base de datos

La base de datos SQLite se crea automáticamente al ejecutar la aplicación por primera vez, generando el archivo:

- app/database.db

En dicha base de datos se almacenan:

- usuarios (users)
- predicciones (predictions)

8. Descripción técnica del funcionamiento de la aplicación

8.1 Arquitectura general

La aplicación sigue una arquitectura modular basada en el patrón **MVC simplificado** (Modelo–Vista–Controlador), adaptado al framework Flask. Cada componente de la aplicación tiene una responsabilidad clara, lo que facilita el mantenimiento y la escalabilidad del proyecto.

La arquitectura se compone de los siguientes elementos principales:

- **Controlador**: rutas Flask y lógica de negocio.
- **Modelo**: base de datos SQLite y modelo de Machine Learning.
- **Vista**: plantillas HTML renderizadas con Jinja2.

8.2 Backend (Flask)

El backend está desarrollado con Flask y organizado mediante **blueprints**, lo que permite separar las funcionalidades principales:

- **Blueprint de autenticación (auth)**:
    - Registro de usuarios.
    - Inicio y cierre de sesión.
    - Control de acceso mediante Flask-Login.
- **Blueprint de predicción (predict)**:
    - Formulario de predicción.

- Ejecución del modelo de Machine Learning.
- Guardado de resultados en la base de datos.
- Visualización del historial de predicciones.
- Panel de administración para usuarios con rol admin.

El control de permisos se realiza mediante el rol del usuario almacenado en la base de datos.

8.3 Modelo de Machine Learning

El modelo de Machine Learning se entrena de forma independiente en un notebook y posteriormente se integra en la aplicación web.

Características del modelo:

- Se utiliza un **pipeline de scikit-learn** que incluye:
  - Preprocesado de variables numéricas y categóricas.
  - Imputación de valores nulos.
  - Codificación One-Hot para variables categóricas.
  - Algoritmo de regresión (Random Forest).
- El modelo entrenado se guarda en un archivo model.pkl mediante **joblib**.
- En la aplicación web, el modelo se carga una única vez y se reutiliza para todas las predicciones.

Esto garantiza que el preprocesado aplicado en producción es el mismo que se utilizó durante el entrenamiento.

8.4 Flujo de una predicción

El proceso completo de una predicción es el siguiente:

1. El usuario introduce los datos de la vivienda mediante el formulario web.
2. Flask recoge los valores enviados y construye un diccionario de entrada.
3. Los datos se transforman en un DataFrame compatible con el modelo.
4. El pipeline del modelo aplica automáticamente el preprocesado necesario.

5. El modelo genera la predicción del precio.
6. El resultado se muestra al usuario en la interfaz.
7. La predicción y los datos de entrada se almacenan en la base de datos.

8.5 Base de datos y persistencia

La persistencia de datos se gestiona mediante **SQLAlchemy**, que permite interactuar con la base de datos SQLite de forma orientada a objetos.

- Los usuarios se almacenan en la tabla users.
- Las predicciones se almacenan en la tabla predictions.
- Cada predicción está asociada a un usuario mediante una clave foránea.

Este diseño permite mantener un historial de predicciones y facilita su consulta tanto por usuarios normales como por administradores.

8.6 Frontend (HTML, CSS y Jinja2)

La interfaz de usuario se desarrolla con HTML y CSS, utilizando **Jinja2** como motor de plantillas.

- Se utiliza una plantilla base común (base.html) para garantizar coherencia visual.
- Las vistas se renderizan dinámicamente según el usuario y su rol.
- El diseño está orientado a la claridad y facilidad de uso, mostrando los resultados de forma destacada.

9. Conclusiones y mejoras futuras

9.1 Conclusiones

Durante el desarrollo de este proyecto se ha creado una aplicación web completa en Python que integra un modelo de Machine Learning para la predicción del precio de viviendas. A lo largo del proyecto se han aplicado los conocimientos adquiridos sobre desarrollo web con Flask, gestión de bases de datos y autenticación de usuarios.

La aplicación cumple con todos los requisitos funcionales planteados inicialmente, permitiendo a los usuarios realizar predicciones, consultar su historial y gestionar el acceso

mediante roles diferenciados. La integración del modelo de inteligencia artificial dentro de la aplicación demuestra la viabilidad de combinar técnicas de ciencia de datos con desarrollo de software.

Además, el uso de una base de datos SQLite garantiza la persistencia de la información y permite mantener un registro completo de las predicciones realizadas, lo que aporta trazabilidad y valor al sistema.

9.2 Mejoras futuras

Como posibles evoluciones del proyecto se proponen las siguientes mejoras:

- Despliegue de la aplicación en un entorno de producción.
- Incorporación de más opciones de visualización de resultados y estadísticas.
- Optimización del modelo de Machine Learning mediante ajuste de hiperparámetros.
- Inclusión de más variables en el formulario avanzado de predicción.
- Exportación de predicciones a formatos externos (CSV, Excel).
- Mejora del diseño visual y experiencia de usuario.
- Integración de nuevos modelos de predicción para comparar resultados.

SEGUNDA PARTE: INTELIGENCIA ARTIFICIAL Y MACHINE LEARNING

1. Definición del problema y objetivos

1.1 Definición del problema

El objetivo del proyecto de Inteligencia Artificial es desarrollar un modelo capaz de predecir el precio de venta de una vivienda a partir de un conjunto de características descriptivas de la misma, como su tamaño, calidad, año de construcción o ubicación.

Este problema se enmarca dentro del ámbito de la regresión supervisada, ya que se dispone de un conjunto de datos históricos etiquetados, donde para cada vivienda se conoce el precio real de venta. A partir de estos datos, el modelo aprende patrones y relaciones entre las variables de entrada y el valor objetivo.

El problema abordado es relevante en el contexto inmobiliario, donde la estimación del valor de una vivienda es una tarea habitual y de gran interés tanto para compradores como para vendedores.

1.2 Variable objetivo

La variable objetivo del modelo es:

- SalePrice: precio de venta de la vivienda.

Se trata de una variable numérica continua, lo que justifica el uso de algoritmos de regresión.

1.3 Objetivos del modelo de IA

Los objetivos específicos del proyecto de Inteligencia Artificial son los siguientes:

- Analizar y comprender el conjunto de datos utilizado para el entrenamiento.
- Limpiar y preprocesar los datos para que sean adecuados para el modelado.
- Explorar las relaciones entre las variables y el precio de venta.
- Entrenar distintos modelos de regresión y comparar su rendimiento.
- Evaluar los modelos mediante métricas adecuadas para problemas de regresión.
- Seleccionar el modelo con mejor rendimiento general.

- Integrar el modelo final en una aplicación web para su uso práctico.

1.4 Justificación del enfoque

Se ha optado por un enfoque de aprendizaje supervisado debido a la disponibilidad de datos históricos con etiquetas conocidas. Este enfoque permite entrenar un modelo capaz de generalizar y realizar predicciones sobre nuevas viviendas no vistas durante el entrenamiento.

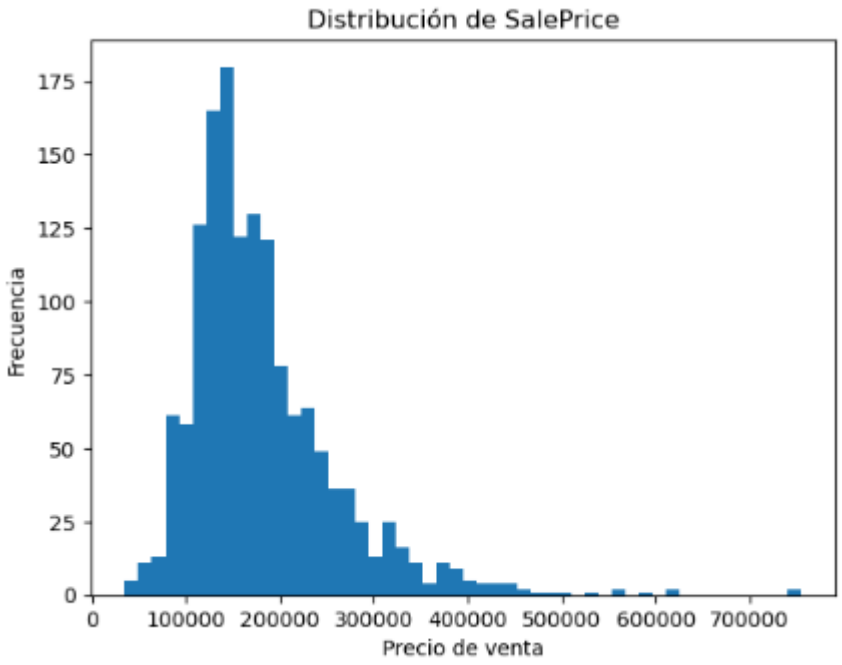
El uso de un modelo de regresión permite obtener estimaciones numéricas continuas, lo que resulta adecuado para el objetivo del proyecto.

2. Descripción del conjunto de datos

2.1 Origen del conjunto de datos

El conjunto de datos utilizado en este proyecto procede de la plataforma **Kaggle**, concretamente del dataset **Ames Housing**, ampliamente utilizado en problemas de regresión y aprendizaje automático.

Este dataset contiene información detallada sobre viviendas vendidas en la ciudad de Ames (Iowa, EE. UU.) e incluye tanto características físicas de las viviendas como información sobre su ubicación y calidades.



2.2 Tamaño y estructura del dataset

El conjunto de datos de entrenamiento está compuesto por:

- **Número de registros:** 1.460 viviendas.
- **Número de variables:** 81 columnas en total.
    - 80 variables explicativas (features).
    - 1 variable objetivo (SalePrice).

Las variables incluyen tanto datos **numéricos** como **categóricos**, lo que requiere un preprocesado adecuado antes del entrenamiento del modelo.

```
import pandas as pd  # Importa la librería pandas y la renombra como 'pd'

# Carga el archivo train.csv desde la ruta indicada y lo guarda en un DataFrame llamado 'train'
train = pd.read_csv("../data/raw/train.csv")

# Carga el archivo test.csv desde la ruta indicada y lo guarda en un DataFrame llamado 'test'
test  = pd.read_csv("../data/raw/test.csv")

# Muestra las dimensiones (filas, columnas) de ambos DataFrames
train.shape, test.shape
```

```
((1460, 81), (1459, 80))
```

2.3 Tipos de variables

El dataset contiene distintos tipos de variables:

- **Variables numéricas**: superficies, número de habitaciones, año de construcción, número de baños, etc.
- **Variables categóricas**: calidad de materiales, tipo de vivienda, barrio, tipo de tejado, zonificación, entre otras.

Esta diversidad de variables hace necesario aplicar técnicas de imputación y codificación para poder utilizar modelos de Machine Learning.

2.4 Variable objetivo

La variable objetivo del dataset es:

- **SalePrice**: precio de venta de la vivienda.

Esta variable representa el valor que el modelo debe predecir y se expresa como un valor numérico continuo.

2.5 Justificación del uso del dataset

El dataset Ames Housing se ha elegido por las siguientes razones:

- Es un dataset realista y ampliamente utilizado en proyectos de Machine Learning.
- Contiene un número elevado de variables que permiten analizar distintos factores que influyen en el precio de una vivienda.
- Incluye tanto variables numéricas como categóricas, lo que lo hace adecuado para aplicar técnicas de preprocesado y modelado más completas.
- Es un dataset suficientemente grande para entrenar y evaluar modelos de regresión de forma fiable.

3. Limpieza y preprocesado de los datos

3.1 Análisis inicial de los datos

Antes de entrenar los modelos, se realizó un análisis inicial del conjunto de datos para identificar posibles problemas de calidad, como valores nulos, tipos de datos incorrectos o variables no adecuadas para el modelado.

Durante este análisis se observó que el dataset contiene un número significativo de **valores nulos**, especialmente en variables categóricas relacionadas con características opcionales de la vivienda (por ejemplo, sótano, garaje o calidad de ciertos elementos).

3.2 Tratamiento de valores nulos

Para gestionar los valores nulos de forma adecuada, se aplicaron distintas estrategias según el tipo de variable:

- **Variables numéricas**:
  Los valores nulos se imputaron utilizando la **mediana** de cada variable. Esta medida es robusta frente a valores extremos y resulta adecuada para distribuciones asimétricas, comunes en datos inmobiliarios.
- **Variables categóricas**:
  Los valores nulos se imputaron utilizando el valor más frecuente o una categoría

específica que indica ausencia de la característica. De este modo, se evita la pérdida de información y se mantiene la coherencia del dataset.

Estas técnicas permiten conservar todos los registros del conjunto de datos sin necesidad de eliminar filas, lo que es especialmente importante dada la cantidad limitada de ejemplos.

3.3 Separación de variables

Se realizó una separación explícita entre:

- **Variables explicativas (X)**: todas las características de la vivienda.
- **Variable objetivo (y)**: SalePrice.

Asimismo, las variables explicativas se dividieron en dos grupos:

- **Variables numéricas**
- **Variables categóricas**

Esta separación facilita la aplicación de diferentes técnicas de preprocesado según el tipo de dato.

```
# Selecciona los nombres de las columnas numéricas (enteros y decimales)
num_cols = X.select_dtypes(include=["int64", "float64"]).columns

# Selecciona los nombres de las columnas categóricas (tipo object)
cat_cols = X.select_dtypes(include=["object"]).columns

# Muestra cuántas columnas numéricas y categóricas hay
len(num_cols), len(cat_cols)
```

```
(37, 43)
```

3.4 Codificación de variables categóricas

Dado que los modelos de Machine Learning no pueden trabajar directamente con variables categóricas en formato texto, se aplicó una **codificación One-Hot Encoding** para transformar estas variables en representaciones numéricas.

Este método crea una columna binaria por cada categoría, evitando la introducción de relaciones ordinales artificiales entre categorías.

```python
# Importa OneHotEncoder: convierte variables categóricas (texto) en columnas binarias (0/1)
# para que los modelos puedan trabajar con ellas
from sklearn.preprocessing import OneHotEncoder

# Importa SimpleImputer: rellena valores faltantes (NaN) con estrategias como media, mediana o moda
from sklearn.impute import SimpleImputer
```

3.5 Uso de pipelines

Todo el proceso de preprocesado se integró dentro de un **pipeline de scikit-learn**, que incluye:

- Imputación de valores nulos.
- Transformación de variables categóricas mediante One-Hot Encoding.
- Entrenamiento del modelo de regresión.

El uso de pipelines garantiza que el mismo preprocesado se aplique de forma consistente tanto durante el entrenamiento como durante la fase de predicción, evitando errores y fugas de información (*data leakage*).

```python
# Pipeline para el tratamiento de variables numéricas
numeric_transformer = Pipeline(steps=[
    # Rellena los valores faltantes usando la mediana
    # (robusta frente a outliers)
    ("imputer", SimpleImputer(strategy="median"))
])

# Pipeline para el tratamiento de variables categóricas
categorical_transformer = Pipeline(steps=[
    # Rellena valores faltantes con la categoría "None"
    # (útil cuando la ausencia tiene significado)
    ("imputer", SimpleImputer(strategy="constant", fill_value="None")),

    # Convierte las categorías en variables binarias (One-Hot Encoding)
    # handle_unknown="ignore" evita errores con categorías nuevas
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])
```

3.6 Ventajas del enfoque aplicado

El enfoque de limpieza y preprocesado utilizado presenta varias ventajas:

- Permite trabajar con datasets que contienen valores nulos.
- Evita la pérdida innecesaria de información.
- Facilita la reutilización del modelo en producción.

- Mejora la mantenibilidad y robustez del sistema.

4. Análisis exploratorio de los datos (EDA)

4.1 Objetivo del análisis exploratorio

El análisis exploratorio de los datos (EDA) se realizó con el objetivo de comprender mejor la distribución de las variables, identificar relaciones relevantes con la variable objetivo (SalePrice) y detectar posibles patrones o anomalías en el conjunto de datos.

Este análisis permitió obtener información valiosa para orientar las decisiones posteriores de preprocesado y selección de modelos.

4.2 Distribución del precio de venta

Se analizó la distribución de la variable objetivo SalePrice, observándose una distribución **asimétrica hacia la derecha**, característica habitual en datos inmobiliarios. La mayoría de las viviendas se concentran en rangos de precio medios, mientras que existen algunos valores elevados correspondientes a viviendas de alto valor.

Este comportamiento justifica el uso de métricas robustas y modelos capaces de manejar distribuciones no perfectamente normales.

4.3 Relación entre variables y el precio

Durante el análisis exploratorio se identificaron varias variables con una relación clara con el precio de venta:

- **OverallQual**: la calidad general de la vivienda presenta una fuerte correlación positiva con el precio.
- **GrLivArea**: la superficie habitable muestra una relación directa con el precio, aumentando a medida que crece el tamaño de la vivienda.
- **TotalBsmtSF** y **GarageCars**: las características relacionadas con el sótano y el garaje también influyen significativamente en el precio.
- **YearBuilt**: las viviendas más recientes tienden a presentar precios más elevados.

Estas observaciones confirman la relevancia de las variables seleccionadas para el formulario de predicción.

La relación entre las variables numéricas y el precio se analizó mediante correlaciones calculadas únicamente sobre variables numéricas inicialmente, evitando interferencias de variables categóricas.

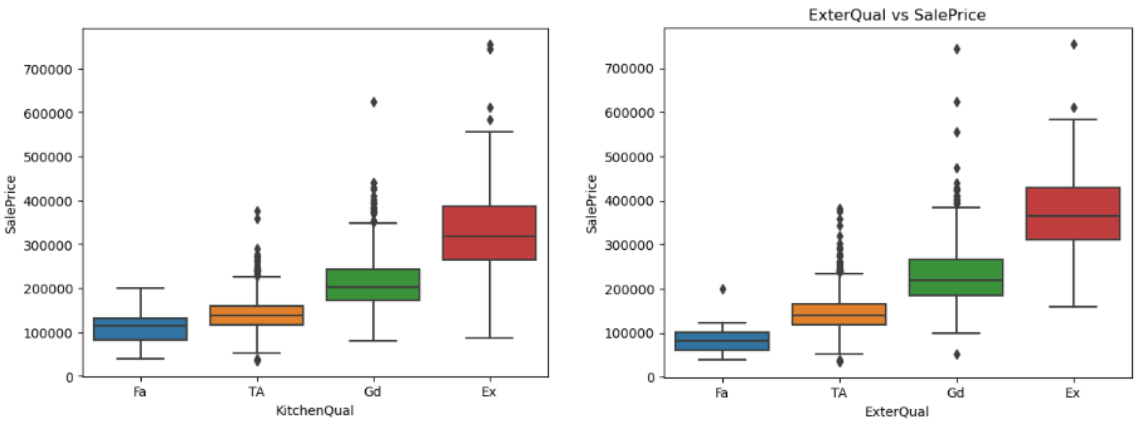
```
# Seleccionar solo columnas numéricas
num_data = train.select_dtypes(include=["int64", "float64"])

# Correlación de variables numéricas con el precio
corr = num_data.corr()["SalePrice"].sort_values(ascending=False)
corr.head(10)
```

```
SalePrice       1.000000
OverallQual     0.790982
GrLivArea       0.708624
GarageCars      0.640409
GarageArea      0.623431
TotalBsmtSF     0.613581
1stFlrSF        0.605852
FullBath        0.560664
TotRmsAbvGrd    0.533723
YearBuilt       0.522897
Name: SalePrice, dtype: float64
```

4.4 Variables categóricas

El análisis de las variables categóricas mostró que aspectos como el **barrio (Neighborhood)**, la **calidad de la cocina (KitchenQual)** y la **calidad exterior (ExterQual)** tienen un impacto significativo en el precio final de la vivienda.

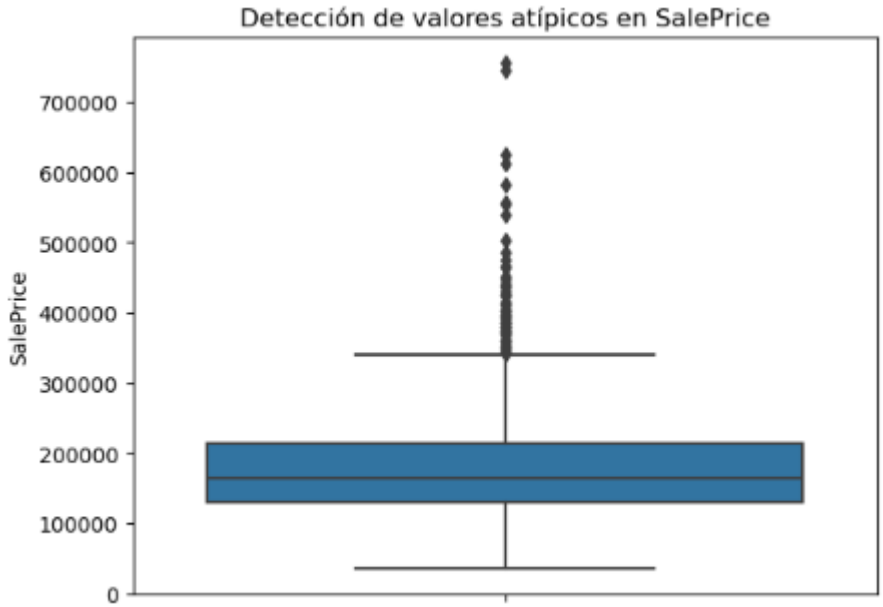
La codificación One-Hot aplicada posteriormente permite capturar estas diferencias sin introducir relaciones artificiales entre categorías.



4.5 Detección de valores atípicos

Durante el EDA se detectaron algunos **valores atípicos** en variables relacionadas con grandes superficies y precios elevados. No obstante, estos valores se consideraron representativos del mercado inmobiliario real, por lo que no se eliminaron del conjunto de datos.

En su lugar, se optó por utilizar modelos robustos capaces de manejar este tipo de observaciones.



4.6 Conclusiones del EDA

El análisis exploratorio permitió confirmar que el conjunto de datos contiene información relevante para la predicción del precio de viviendas. Las variables seleccionadas presentan relaciones coherentes con la variable objetivo, y el dataset es adecuado para el entrenamiento de modelos de regresión.

Las conclusiones obtenidas durante esta fase guiaron las decisiones posteriores de preprocesado y selección del modelo final.

5. Modelos entrenados y metodología

5.1 División de los datos

Para entrenar y evaluar los modelos se dividió el conjunto de datos en dos subconjuntos:

- **Conjunto de entrenamiento**: utilizado para ajustar los modelos.
- **Conjunto de validación/prueba**: utilizado para evaluar el rendimiento de los modelos sobre datos no vistos.

La división se realizó utilizando una proporción estándar (por ejemplo, 80% entrenamiento y 20% prueba), garantizando la reproducibilidad mediante una semilla aleatoria fija.

5.2 Modelos evaluados

Durante el proyecto se entrenaron y evaluaron distintos modelos de regresión con el objetivo de comparar su rendimiento:

- **Regresión Lineal**: utilizada como modelo base por su simplicidad e interpretabilidad.
- **Random Forest Regressor**: modelo de conjunto que combina múltiples árboles de decisión para mejorar la capacidad de generalización.

Todos los modelos se integraron dentro de un pipeline que incluye el preprocesado de los datos.

5.3 Métricas de evaluación

Para evaluar el rendimiento de los modelos se utilizaron métricas habituales en problemas de regresión:

- **MAE (Mean Absolute Error)**: error medio absoluto entre el valor real y el predicho.
- **RMSE (Root Mean Squared Error)**: raíz del MSE, expresada en las mismas unidades que la variable objetivo.
- **R^2 (Coeficiente de determinación)**: indica la proporción de la variabilidad del precio explicada por el modelo.

Estas métricas permiten evaluar tanto la precisión como la capacidad explicativa de los modelos.

5.4 Selección del modelo final

Tras comparar los resultados obtenidos por los distintos modelos, se seleccionó el **Random Forest Regressor** como modelo final, ya que presentó un mejor equilibrio entre precisión y capacidad de generalización.

Este modelo mostró un rendimiento superior frente a la regresión lineal, especialmente en la gestión de relaciones no lineales y en la reducción del sobreajuste.

RMSE (más bajo = mejor)

- Random Forest comete errores grandes **menos graves** que la regresión lineal.
- Mejora de ~2 000 € en error típico penalizado.

MAE (más bajo = mejor)

- En promedio, el Random Forest se equivoca **unos 3 000 € menos** por vivienda.
- Esto es **una mejora clara y relevante**.

R^2 (más alto = mejor)

- Random Forest explica **más variabilidad del precio**.
- Aunque la diferencia parece pequeña, es **significativa** en este tipo de problemas.

5.5 Integración del modelo en producción

El modelo seleccionado se entrenó utilizando todo el conjunto de entrenamiento y se guardó mediante **joblib**. Posteriormente, se integró en la aplicación web, donde se utiliza para generar predicciones a partir de los datos introducidos por los usuarios.

El uso de un pipeline garantiza que el mismo preprocesado aplicado durante el entrenamiento se utilice también en la fase de predicción.

6. Resultados y evaluación final

6.1 Resultados obtenidos

Tras entrenar y evaluar los distintos modelos de regresión, se obtuvieron resultados satisfactorios en la predicción del precio de las viviendas. El modelo seleccionado (Random Forest Regressor) mostró un rendimiento superior, tanto en términos de error como de capacidad de generalización.

Las métricas de evaluación se calcularon sobre un conjunto de validación, compuesto por datos no utilizados durante el entrenamiento.

6.2 Métricas

A continuación se muestran las métricas:

```
import pandas as pd

results = pd.DataFrame({
    "Modelo": ["Regresión Lineal", "Random Forest"],
    "RMSE": [rmse, rmse_rf],
    "MAE": [mae_lr, mae_rf],
    "R2": [r2_lr, r2_rf]
})

results
```

	Modelo	RMSE	MAE	R2
0	Regresión Lineal	31054.985973	20466.007197	0.874267
1	Random Forest	28998.496545	17541.673893	0.890368

6.3 Interpretación de los resultados

El modelo presenta un buen rendimiento general, siendo capaz de realizar predicciones razonables para distintos tipos de viviendas. El valor de R^2 obtenido indica que una parte significativa de la variabilidad del precio puede ser explicada por las variables incluidas en el modelo.

No obstante, como ocurre en la mayoría de problemas reales, existen factores externos no incluidos en el dataset (estado del mercado, ubicación exacta, contexto económico) que pueden influir en el precio final y limitar la precisión del modelo.

6.4 Limitaciones del modelo

Entre las principales limitaciones del modelo se encuentran:

- Dependencia del dataset utilizado para el entrenamiento.
- Menor interpretabilidad debido al uso de modelos de conjunto.

Estas limitaciones son inherentes a muchos modelos de Machine Learning aplicados a problemas reales.

6.5 Validación del enfoque

A pesar de estas limitaciones, los resultados obtenidos validan el enfoque seguido, demostrando que el uso de técnicas de Machine Learning resulta adecuado para el problema planteado y que el modelo puede ser utilizado como una herramienta de estimación de precios de viviendas.

7. Conclusiones finales

7.1 Conclusiones

En este proyecto se ha desarrollado un modelo de Inteligencia Artificial capaz de predecir el precio de venta de una vivienda a partir de un conjunto de características descriptivas. A lo largo del proceso se han aplicado las distintas fases propias de un proyecto de ciencia de datos, desde el análisis del problema y la exploración de los datos hasta el entrenamiento, evaluación e integración del modelo.

Los resultados obtenidos muestran que el modelo seleccionado es capaz de realizar predicciones razonables, capturando relaciones relevantes entre las variables y el precio de venta.

7.2 Valor del proyecto

Este proyecto demuestra la viabilidad de integrar modelos de Machine Learning en aplicaciones web reales, permitiendo que usuarios finales puedan interactuar con modelos predictivos de forma sencilla. Además, el trabajo realizado evidencia la importancia de un correcto preprocesado de los datos y de la evaluación adecuada de los modelos.

7.3 Líneas de mejora

Como posibles líneas de mejora en el ámbito de la Inteligencia Artificial se proponen:

- Incorporar técnicas de ajuste de hiperparámetros para mejorar el rendimiento del modelo.
- Evaluar otros modelos avanzados de regresión.
- Analizar la importancia de las variables para mejorar la interpretabilidad del modelo.
- Incorporar validación cruzada para una evaluación más robusta.
- Ampliar el conjunto de datos utilizado para el entrenamiento.

INTEGRACIÓN DEL MODELO IA EN LA APLICACIÓN PYTHON HOUSEPRICE APP

En este proyecto se han desarrollado dos partes complementarias: por un lado, el entrenamiento y evaluación de un modelo de Machine Learning (IA) para predecir el precio de una vivienda, y por otro lado, una aplicación web en Python con Flask (parte de software). La integración consiste en convertir el modelo entrenado en un componente reutilizable dentro de la aplicación web para que cualquier usuario pueda utilizarlo mediante un formulario.

Flujo de integración

1. Entrenamiento del modelo en el notebook

En la fase de IA se cargaron los datos (Ames Housing), se realizó el análisis exploratorio y se preparó un pipeline de scikit-learn que incluye:

- imputación de valores nulos (numéricas y categóricas),
- codificación One-Hot de variables categóricas,
- y el algoritmo final de regresión (Random Forest).

Este pipeline es importante porque encapsula tanto el preprocesado como el modelo, garantizando que el tratamiento de los datos en producción sea exactamente el mismo que durante el entrenamiento.

2. Guardado del modelo entrenado

Una vez seleccionado el modelo final, se guardó utilizando joblib en un archivo (por ejemplo, model.pkl). De esta forma, el modelo puede cargarse posteriormente sin necesidad de reentrenar, lo que reduce tiempos y permite reutilización.

3. Carga del modelo en la aplicación Flask

En la parte Python, la aplicación web carga el archivo model.pkl al iniciar. Esto permite mantener el modelo disponible en memoria para realizar predicciones de forma rápida cuando los usuarios envían el formulario.

4. Captura de datos desde el formulario

El usuario introduce las características de la vivienda en la interfaz web. La aplicación ofrece un modo "Express" con variables principales y un modo "Avanzado" con variables adicionales opcionales. Los valores del formulario se recogen en Flask y se organizan en una estructura (diccionario) con los nombres de variables que espera el modelo.

5. Construcción del DataFrame y predicción

Con los valores recogidos, la aplicación construye un DataFrame con la misma estructura que los datos usados en el entrenamiento. A continuación, el pipeline realiza automáticamente el preprocesado y genera la predicción del precio mediante model.predict().

6. Presentación del resultado y almacenamiento en base de datos

La predicción se muestra al usuario en la interfaz web (resultado destacado). Además, se guarda en la base de datos SQLite:

- el usuario que realizó la predicción,
- los datos de entrada (en formato JSON),
- el precio predicho,
- y la fecha/hora.

Esto permite que el usuario consulte su historial y que el administrador pueda revisar todas las predicciones desde el panel de administración.

Ventajas de esta integración

- **Consistencia**: el pipeline asegura que el preprocesado en la app es idéntico al entrenamiento.
- **Reproducibilidad**: el modelo se guarda y reutiliza sin reentrenar.
- **Usabilidad**: el usuario final no necesita conocimientos técnicos para obtener una predicción.
- **Trazabilidad**: cada predicción queda registrada en la base de datos con sus inputs y resultado.