

**ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA**
Universidad de Córdoba



MANUAL DE USUARIO DEL TFG
Grado en Ingeniería Informática

**Asistente para la configuración de Pipelines de
Aprendizaje Supervisado mediante LLMs**

Autor: Isaac Manuel Cejudo Alfaro
Directores: Dr. José Raúl Romero Salguero
Dr. Carlos García Martínez



UNIVERSIDAD DE CÓRDOBA



Tabla de Contenidos

1. Introducción.....	3
2. Instalación	3
2.1. Requisitos previos	3
2.2. Pasos de instalación	3
3. Configuración	5
4. Guía de uso	5
4.1. Interfaz principal	5
4.2. Ejemplo de flujo completo	6



Tabla de Figuras

Figura 1. Vista general del sistema	6
Figura 2. Vista previa del dataset	7
Figura 3. Resultados del EDA.....	7
Figura 4. Preprocesamiento realizado	8
Figura 5. Selección de características aplicada	8
Figura 6. Selección de instancias aplicada	9
Figura 7. Entrenamiento y Evaluación del modelo	10
Figura 8. Métricas generadas del modelo.....	10
Figura 9. Modelo entrenado	11
Figura 10. Ejemplo de logs de CrewAI.....	11



1. Introducción

El sistema que se ha desarrollado es un asistente multiagente pensado para guiar la configuración de pipelines de aprendizaje supervisado a través de instrucciones en lenguaje natural. La idea central se basa en que cualquier persona pueda crear flujos de trabajo de *machine learning* sin necesidad de codificar nada. Así, tareas que normalmente suelen ser complejas (el análisis exploratorio, el preprocesamiento de datos, la selección de características o instancias y el entrenamiento y evaluación de modelos) se convierten accesibles gracias a una interfaz conversacional donde el usuario podrá conversar con el asistente para conseguir dichas tareas.

El asistente se apoya en *CrewAI* para organizar a los agentes especializados, en *scikit-learn* como base para las tareas de *machine learning* y *Panel* para ofrecer un *dashboard* interactivo e intuitivo.

2. Instalación

2.1. Requisitos previos

- **Sistema Operativo:** Windows 11, Linux o Mac.
- **Python:** versión 3.10 o superior.
- **pip**
- **Hardware recomendado:** CPU i5 o equivalente, 16 GB RAM, GPU opcional.

2.2. Pasos de instalación

Los pasos para la instalación del sistema son los siguientes:



1. Clonar el repositorio o descargar el proyecto.

```
git clone https://github.com/p82ceali/proyecto-tfg.git  
cd proyecto-tfg
```

2. Crear y activar un entorno virtual.

- En Windows.

```
python -m venv venv  
venv\\Scripts\\activate
```

- En Linux/macOS:

```
python3 -m venv venv  
source venv/bin/activate
```

3. Instalar las dependencias.

```
pip install ".[ui,agents,xgb]"
```

Esto instalará todas las dependencias necesarias.

4. Configurar el entorno.

Crear un archivo `.env` en la raíz del proyecto y añadir tu *Google API key*:

```
GOOGLE_API_KEY="your_api_key"
```



5. Ejecutar la aplicación.

```
panel serve src/tfg_ml/interface/ui_panel.py --autoreload --show
```

Al indicar `--show` se abrirá directamente una ventana en el navegador con el sistema.

3. Configuración

La única configuración que necesita el sistema es indicar la *Google API key* dentro del archivo `.env` como se ha visto en la sección anterior. Una vez realizados los pasos de la instalación, el sistema está preparado para funcionar correctamente (ver 2. Instalación).

4. Guía de uso

4.1. Interfaz principal

En la Figura 1 se observa cómo está dividido el *dashboard* del sistema.

En la barra superior, se puede observar el nombre del sistema, junto con una breve descripción de lo que realiza. Además, en la esquina izquierda se encuentra un botón, que, si se pulsa, hará que el sector izquierdo alterne entre plegarse y desplegarse logrando que el usuario sea capaz de personalizarlo a su gusto y mejore así su experiencia. En la esquina derecha se observa otro botón para alternar entre el modo claro y oscuro (usar preferiblemente el claro ya que algunos navegadores no convierten los elementos correctamente a dicho estilo)

En el sector izquierdo, se observa una ventana para cargar el *dataset* que se quiera utilizar. Debajo aparece el número de filas y columnas que tiene el *dataset* cargado, para tener una idea básica del conjunto de datos. Mas abajo, se visualiza una barra de estado para conocer si se está ejecutando el pipeline y, si ha resultado con éxito o con error, lo indica.

Manual de Usuario de Trabajo Fin de Grado

En el sector central, se puede visualizar el núcleo de este sistema, es decir, el chat para conversar con el asistente. Cuando se ejecuta el sistema, aparece un mensaje de bienvenida para conocer todo lo que puede realizar el asistente y así tener una idea básica del alcance de este.

Para finalizar, en el sector inferior se puede visualizar los datos del *dataset* cargado. Esta ventana se irá actualizando conforme se vaya actualizando el *dataset* para que el usuario pueda seguir el progreso de los datos.

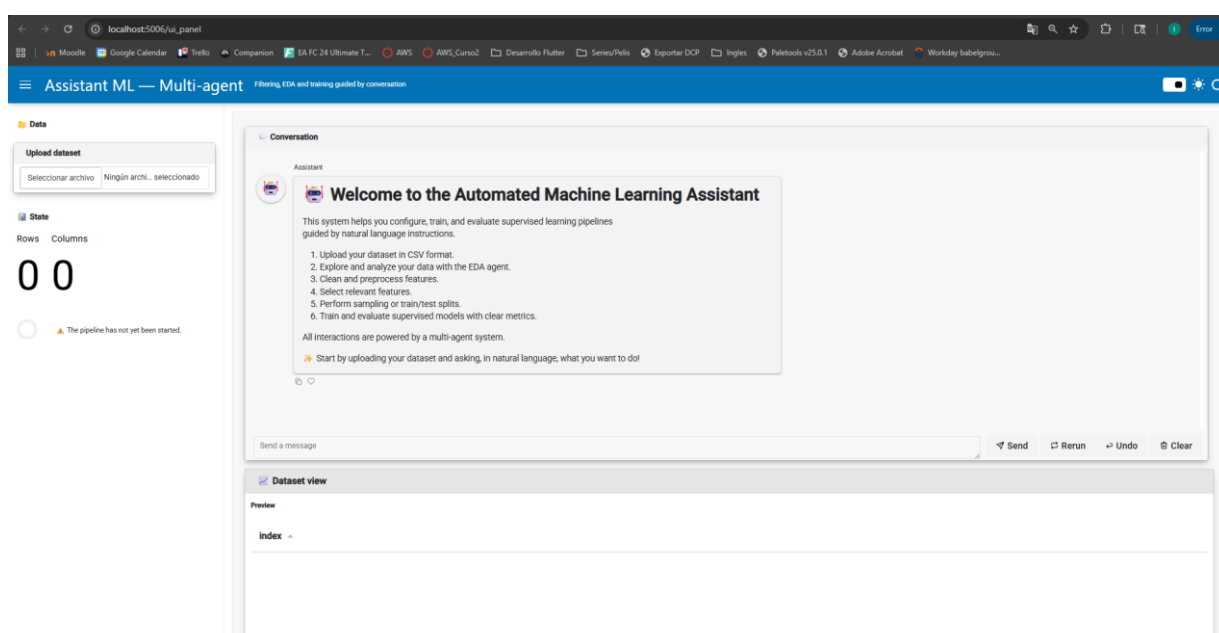


Figura 1. Vista general del sistema

4.2. Ejemplo de flujo completo

1. Carga y visualización de datos.

Se pulsa sobre *Seleccionar Archivo* y se carga el *dataset*. En la Figura 2 se puede observar cómo se visualizan los datos y el número de filas y columnas del *dataset*.

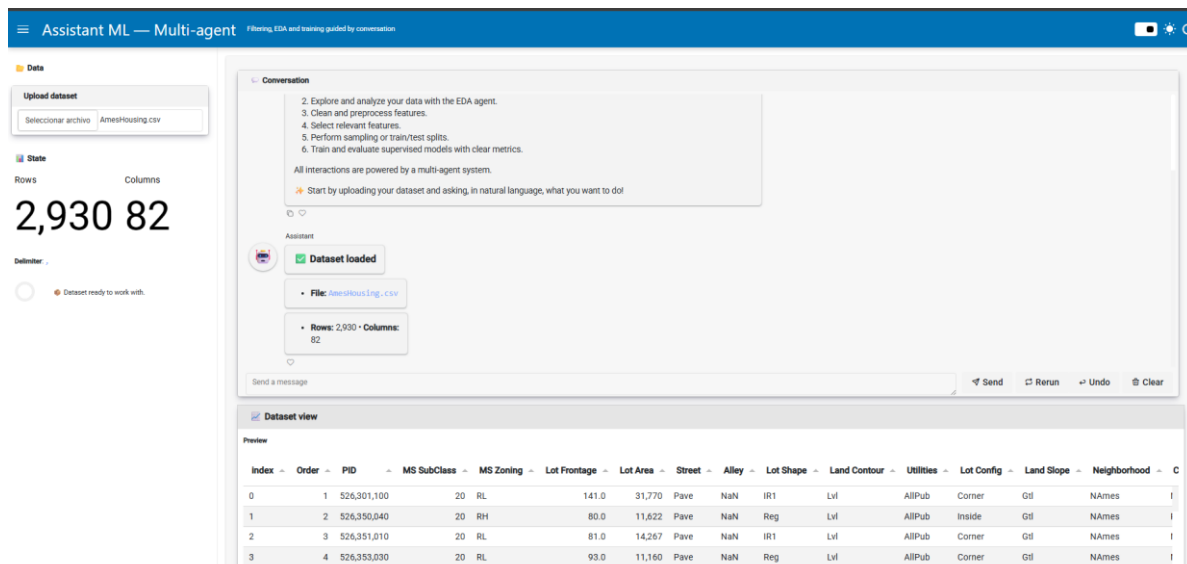


Figura 2. Vista previa del dataset

2. Análisis Exploratorio (EDA)

El usuario indica la siguiente orden: “Describe SalePrice column”. El asistente responde con las estadísticas descriptivas. En la Figura 3 se observa esta interacción.

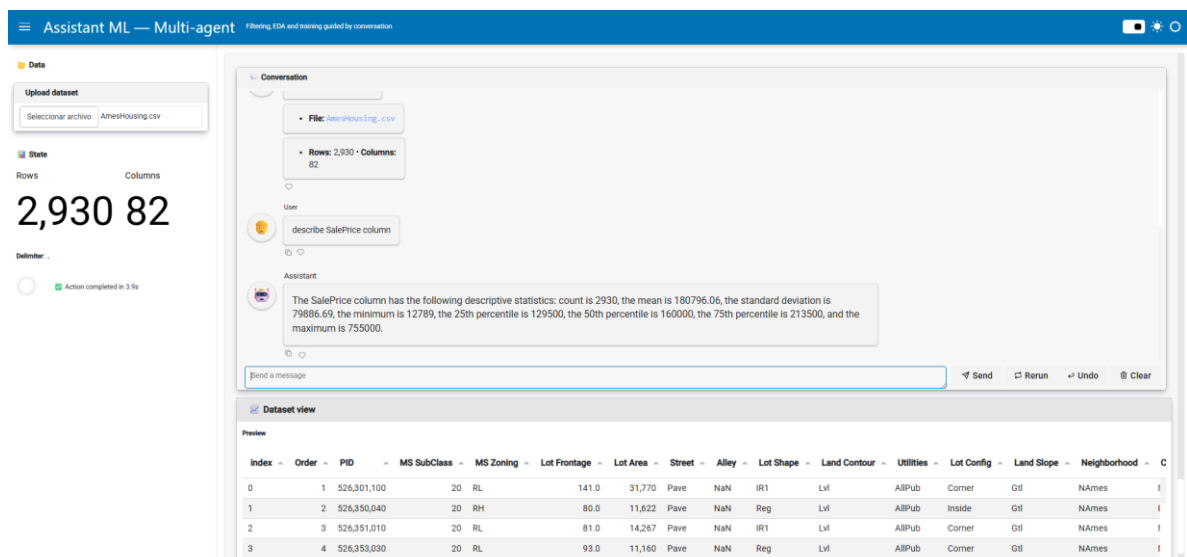


Figura 3. Resultados del EDA

3. Preprocesamiento

El usuario indica la siguiente orden: “*Apply one-hot encoding to Neighborhood column*”.

El asistente responde realizando el preprocesamiento en esa variable. En la Figura 4 se puede observar dicha interacción.

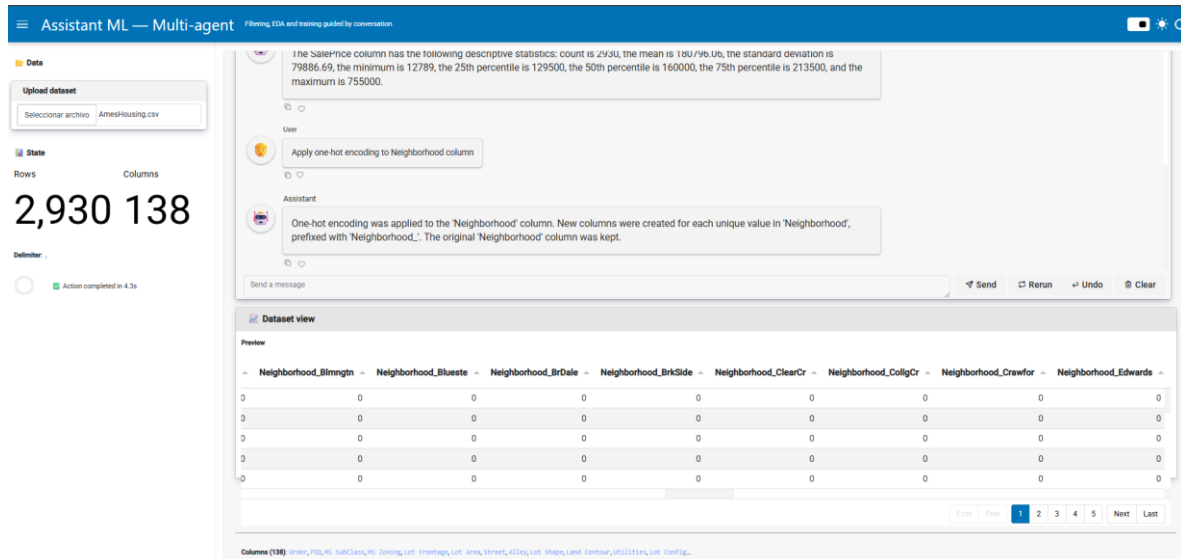


Figura 4. Preprocesamiento realizado

4. Selección de características

El usuario solicita una reducción del *dataset* a través de la selección de características usando la siguiente orden: “*Select 5 features most relevant for SalePrice using RandomForest*”. El asistente responde con un informe de todo lo que ha realizado con esa instrucción. En la Figura 5 se muestra esta interacción.

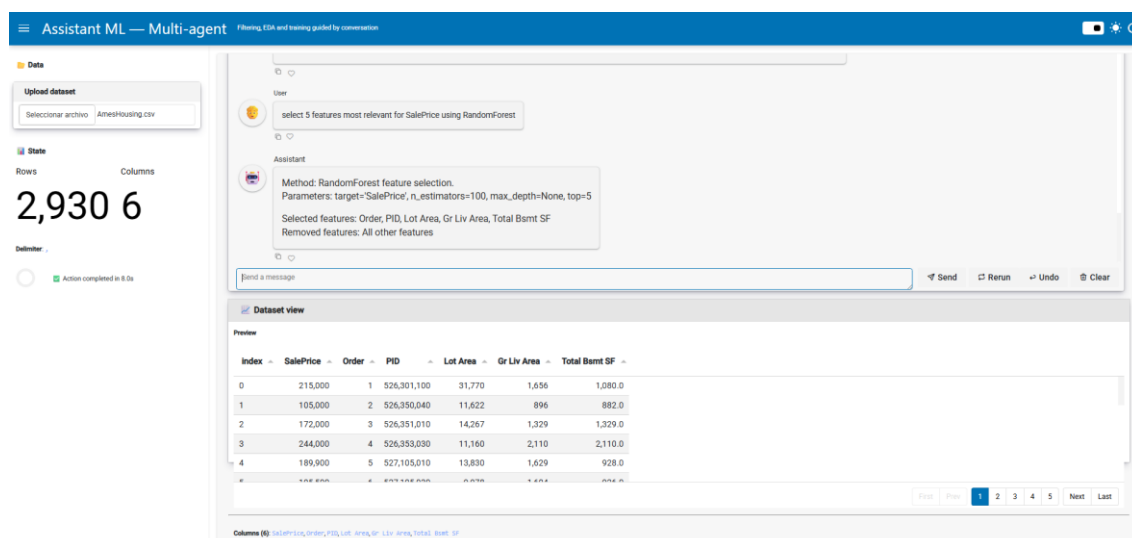
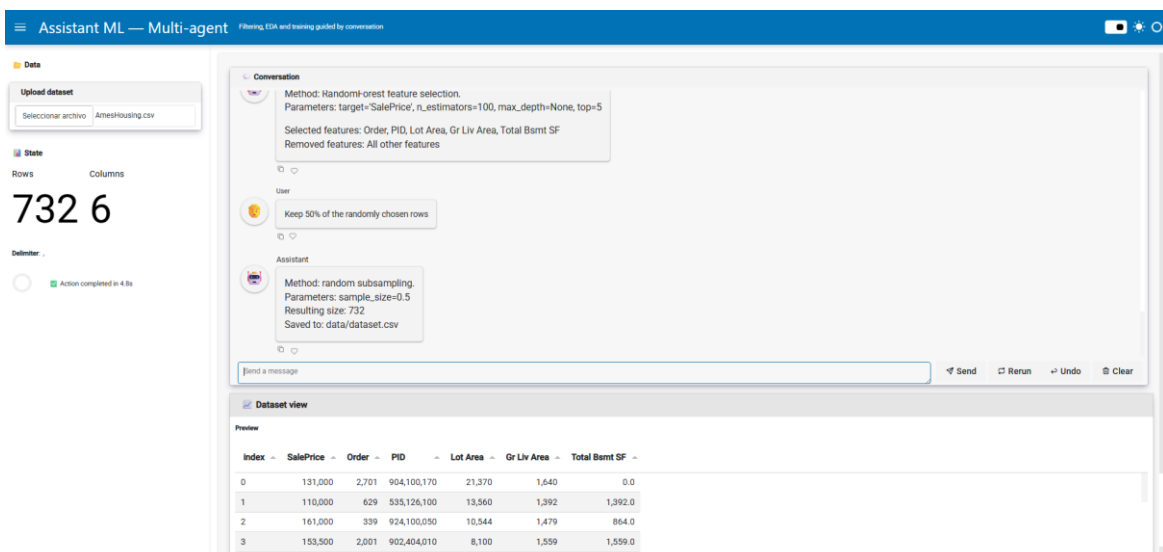


Figura 5. Selección de características aplicada

5. Selección de instancias

El usuario solicita una selección de instancias para reducir el *dataset* usando la siguiente orden: “*Keep 50% of the randomly chosen rows*”. El asistente responde informando de todos los cambios realizados. En la Figura 6 se puede comprobar la interacción entre ellos.



The screenshot shows the Assistant ML — Multi-agent interface. On the left, the 'Data' section shows an uploaded dataset 'AmesHousing.csv' with 732 rows and 6 columns. The 'State' section shows a progress bar and a message 'Action completed in 4.6s'. The 'Conversation' section shows a dialogue between the User and the Assistant. The User asks to keep 50% of the randomly chosen rows. The Assistant responds with the method 'random subsampling', parameters 'sample_size=0.5', resulting size '732', and saved to 'data/dataset.csv'. Below the conversation, the 'Dataset view' section shows a preview of the dataset with columns: Index, SalePrice, Order, PID, Lot Area, Gr Liv Area, and Total Bsmt SF. The preview shows the first four rows of data.

Index	SalePrice	Order	PID	Lot Area	Gr Liv Area	Total Bsmt SF
0	131,000	2,701	904,100,170	21,370	1,640	0.0
1	110,000	629	535,126,100	13,560	1,392	1,392.0
2	161,000	339	924,100,090	10,544	1,479	864.0
3	153,500	2,001	902,404,010	8,100	1,559	1,559.0

Figura 6. Selección de instancias aplicada

6. Entrenamiento y evaluación del modelo.

El usuario solicita el entrenamiento del modelo usando la siguiente instrucción: “*Train a Random Forest to predict SalePrice column*”. El asistente responde con un informe describiendo lo que ha usado para entrenar el modelo y las métricas generadas en ese modelo. En la Figura 6 se muestra la interacción.

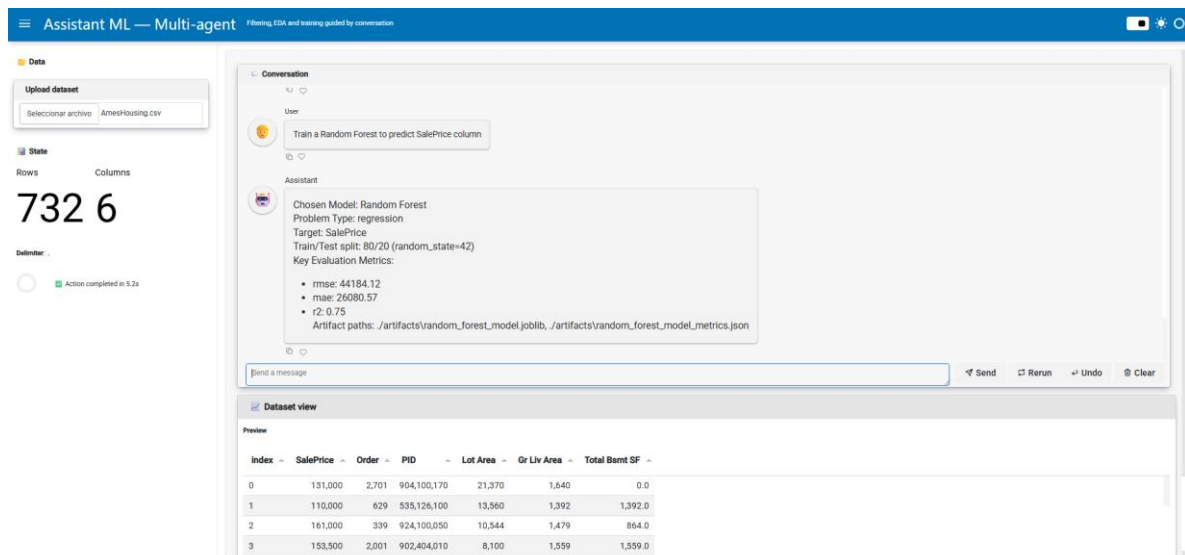


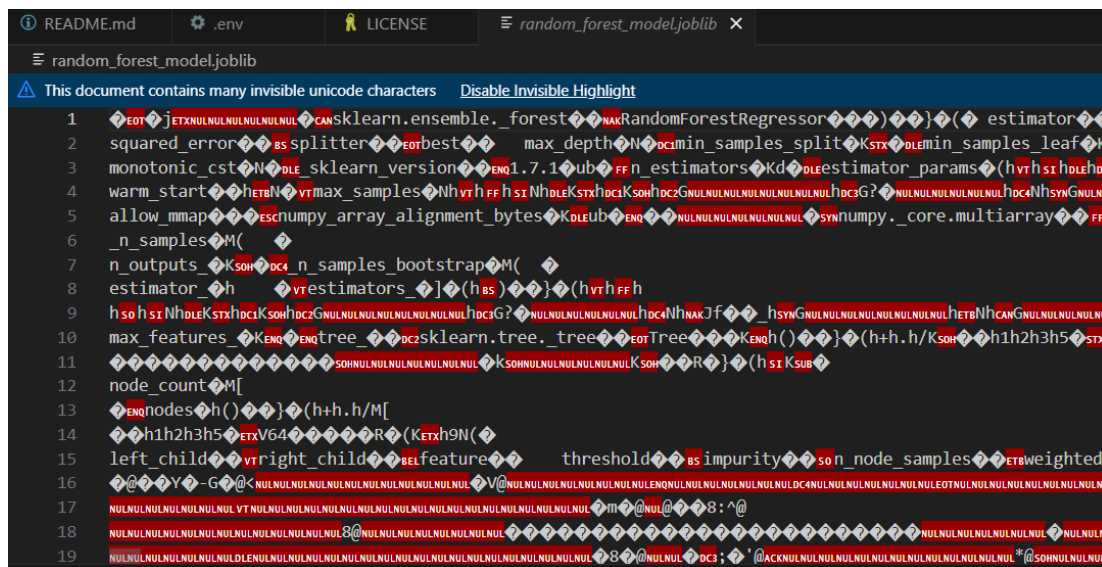
Figura 7. Entrenamiento y Evaluación del modelo

En las Figuras 8 y 9 se observa los distintos archivos generados a causa de entrenar el modelo. El modelo generado es un archivo binario por lo que no es legible.

Por otro lado, en la Figura 10 se observa un ejemplo de los logs que muestra CrewAI.

```
{} random_forest_model_metrics.json > ...
1  {
2    "problem_type": "regression",
3    "model": "random_forest",
4    "target": "1st Flr SF",
5    "rmse": 237.37775124078013,
6    "mae": 148.02093856655287,
7    "r2": 0.6764969975726369
8  }
```

Figura 8. Métricas generadas del modelo

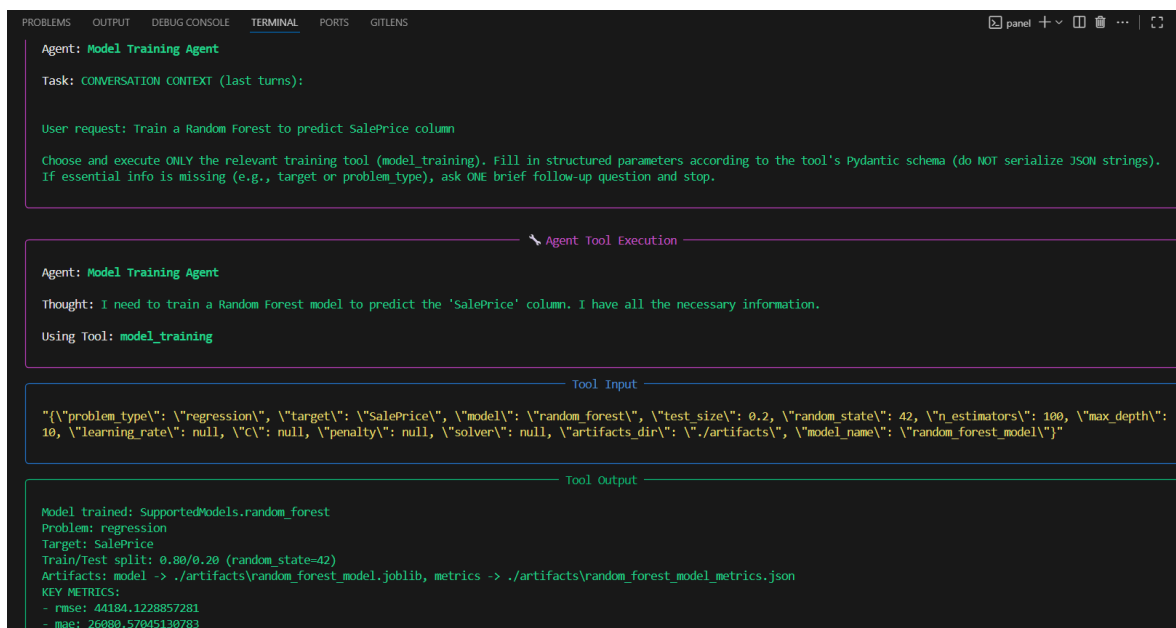


```

1  from sklearn.ensemble import RandomForestRegressor
2  from sklearn.metrics import mean_squared_error
3  from sklearn.model_selection import train_test_split
4  from sklearn.preprocessing import StandardScaler
5  from sklearn.utils import shuffle
6  import numpy as np
7  import joblib
8  import random
9  import sys
10 import os
11
12 # Parameters
13 n_estimators = 100
14 max_depth = 5
15 min_samples_split = 2
16 min_samples_leaf = 1
17 min_samples_bootstrap = 1
18 bootstrap = True
19
20 # Load data
21 data = pd.read_csv('data.csv')
22 X = data[['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10']]
23 y = data['SalePrice']
24
25 # Split data
26 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
27
28 # Scale data
29 scaler = StandardScaler()
30 X_train = scaler.fit_transform(X_train)
31 X_test = scaler.transform(X_test)
32
33 # Train model
34 model = RandomForestRegressor(n_estimators=n_estimators, max_depth=max_depth, min_samples_split=min_samples_split, min_samples_leaf=min_samples_leaf, min_samples_bootstrap=min_samples_bootstrap, bootstrap=bootstrap)
35 model.fit(X_train, y_train)
36
37 # Save model
38 joblib.dump(model, 'random_forest_model.joblib')
39
40 # Predict
41 y_pred = model.predict(X_test)
42
43 # Calculate metrics
44 rmse = mean_squared_error(y_test, y_pred)
45 mae = mean_absolute_error(y_test, y_pred)
46
47 # Print metrics
48 print('RMSE: ', rmse)
49 print('MAE: ', mae)

```

Figura 9. Modelo entrenado



```

Agent: Model Training Agent

Task: CONVERSATION CONTEXT (last turns):

User request: Train a Random Forest to predict SalePrice column

Choose and execute ONLY the relevant training tool (model_training). Fill in structured parameters according to the tool's Pydantic schema (do NOT serialize JSON strings).
If essential info is missing (e.g., target or problem_type), ask ONE brief follow-up question and stop.

----- Agent Tool Execution -----

Agent: Model Training Agent

Thought: I need to train a Random Forest model to predict the 'SalePrice' column. I have all the necessary information.

Using Tool: model_training

----- Tool Input -----

{"problem_type": "regression", "target": "SalePrice", "model": "random forest", "test_size": 0.2, "random_state": 42, "n_estimators": 100, "max_depth": 5, "learning_rate": null, "c": null, "penalty": null, "solver": null, "artifacts_dir": "./artifacts", "model_name": "random_forest_model"}

----- Tool Output -----

Model trained: SupportedModels.random_forest
Problem: regression
Target: SalePrice
Train/Test split: 0.80/0.20 (random_state=42)
Artifacts: model -> ./artifacts/random_forest_model.joblib, metrics -> ./artifacts/random_forest_model_metrics.json
KEY METRICS:
- rmse: 44184.1228857281
- mae: 26080.57045130783

```

Figura 10. Ejemplo de logs de CrewAI