

# **PredictCar**

## **Predicción del Precio de Vehículos**

Álvaro Martínez Flores

19/05/2025

<b>Resumen Ejecutivo</b>	<b>3</b>
<b>1. Introducción</b>	<b>3</b>
1.1 Contexto y motivación	3
1.2 Objetivos del proyecto	3
1.3 Alcance del proyecto	4
<b>2. Marco Teórico</b>	<b>4</b>
2.1 Conceptos clave	4
2.2 Estado del entorno	4
2.3 Herramientas y tecnologías seleccionadas	4
<b>3. Metodología</b>	<b>5</b>
3.1 Planificación del proyecto	5
3.2 Recopilación de datos	6
3.3 Procesamiento y análisis de datos	6
3.3.1 Limpieza de datos	6
3.3.2 Organización de carpetas	14
3.3.2 Creación del modelo	15
3.3.3 Guardado de los modelos en MYSQL	20
3.3.4 Prueba con modelo de HuggingFace	21
3.3.5 Web	22
3.4 Desarrollo del modelo de IA	24
<b>4. Resultados</b>	<b>24</b>
4.1 Resultados obtenidos	24
4.2 Comparativa con objetivos iniciales	28
4.3 Limitaciones de los resultados	28
<b>5. Conclusiones y Futuras Líneas de Trabajo</b>	<b>28</b>
5.1 Conclusiones principales	28
5.2 Impacto del proyecto	28
5.3 Líneas de trabajo futuro	29
<b>6. Bibliografía</b>	<b>29</b>

# Resumen Ejecutivo

La pregunta principal que ha llevado a la creación de este proyecto es, ¿cuánto debería valer un vehículo usado?. Para responderla, se ha desarrollado un sistema de predicción basado en técnicas de Machine Learning, con una interfaz web amigable y datos reales extraídos de publicaciones en Craigslist.

Durante el proceso, se emplearon herramientas como Python (y librerías como pandas, scikit-learn y matplotlib), así como tecnologías de despliegue como Flask y Grafana. Se compararon varios modelos, y finalmente se optó por Random Forest como el más equilibrado entre precisión y robustez.

La web está pensada para un uso sencillo: el usuario introduce las características de su coche (marca, año, tipo, condición, etc.) y obtiene una estimación del precio. Además, se ha diseñado con miras a ser fácilmente ampliable a otros tipos de vehículos en un futuro.

## 1. Introducción

### 1.1 Contexto y motivación

Este proyecto está creado para las personas que quieran vender su coche por cualquier motivo, pero no siempre es fácil saber si el precio es justo. Aquí es donde entra en juego este proyecto: ayudar a un usuario a saber el precio aproximado de su coche.

### 1.2 Objetivos del proyecto

#### Objetivo general:

Desarrollo de un sistema de predicción de precios para vehículos usados, accesible a través de una interfaz web.

#### Objetivos específicos:

- Utilizar técnicas de Machine Learning para estimar el precio según distintas variables.
- Comparar varios modelos y elegir el más eficaz.
- Implementar una solución visual mediante dashboards y una interfaz web.

## 1.3 Alcance del proyecto

Este proyecto se ha centrado exclusivamente en coches, por lo que, por ahora, no contempla la predicción de precios para motos, bicicletas o scooters. Tampoco se tienen en cuenta valoraciones subjetivas sobre el estado del vehículo, como por ejemplo el desgaste interior o exterior, ni se realiza ninguna conexión con APIs públicas para obtener datos en tiempo real. El enfoque está en una versión funcional y local, sin integrar aún fuentes externas ni extenderse a otros tipos de vehículos.

## 2. Marco Teórico

### 2.1 Conceptos clave

Respecto a BigData se usan datos reales de ventas de coches para alimentar el modelo, sacados de la web Kaggle.

Por otro lado, la Inteligencia Artificial, permite entrenar algoritmos para que aprendan patrones a partir de los datos y hagan predicciones. Entre las técnicas empleadas destacan los árboles de decisión, los bosques aleatorios (Random Forest) y los modelos de boosting como XGBoost.

A nivel práctico, también se utilizan herramientas como pandas y numpy para procesar datos, matplotlib y seaborn para visualizarlos, y scikit-learn para desarrollar y evaluar los modelos.

### 2.2 Estado del entorno

Si bien existen portales como AutoScout o Milanuncios que ofrecen tasaciones aproximadas, su funcionamiento no siempre está claro. El enfoque de este proyecto pretende ser más transparente.

En cuanto a investigaciones previas, muchos trabajos han explorado la regresión lineal como primera aproximación, pero se ha querido ir más allá y probar distintos modelos avanzados para ver cuál se ajusta mejor a la realidad de los datos.

### 2.3 Herramientas y tecnologías seleccionadas

Desde el principio se tuvo claro que se quería que el proyecto fuese funcional, práctico, y a ser posible, bonito visualmente. Por eso, se combinaron varias herramientas que ayudaron en cada parte:

- **Python:** para el preprocesamiento, modelado y evaluación de datos.
- **RapidMiner:** especialmente útil para la limpieza inicial del dataset.
- **Flask:** el motor que hace posible la web.

- **HTML + Tailwind + Axios:** dan vida al frontend.
- **MySQL:** donde se guardan todas las métricas.
- **Grafana:** la herramienta seleccionada para visualizar y comparar los resultados de los modelos.

## 3. Metodología

### 3.1 Planificación del proyecto

# Cronograma PredictCar

Álvaro Martínez Flores

PROCESOS	MAYO 2025																
	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Definición del objetivo del proyecto y recopilación de datos																	
Preprocesamiento de datos y selección de características relevantes																	
Evaluación y comparación de modelos de predicción																	
Desarrollo del frontend y backend de la aplicación web																	
Documentación técnica del proyecto																	
Pruebas de funcionamiento del sistema completo en distintos entornos																	

Desde el inicio, se organizó el trabajo en fases bien definidas: recopilación de datos, limpieza, modelado, evaluación, y por último, integración en la web. Aunque no se siguió una metodología formal como Scrum, se intentó mantener una estructura ágil.

Se dedicó una primera parte a entender a fondo el dataset, para luego pasar a la selección de variables relevantes, pruebas con distintos modelos, y finalmente el despliegue del sistema.

## 3.2 Recopilación de datos

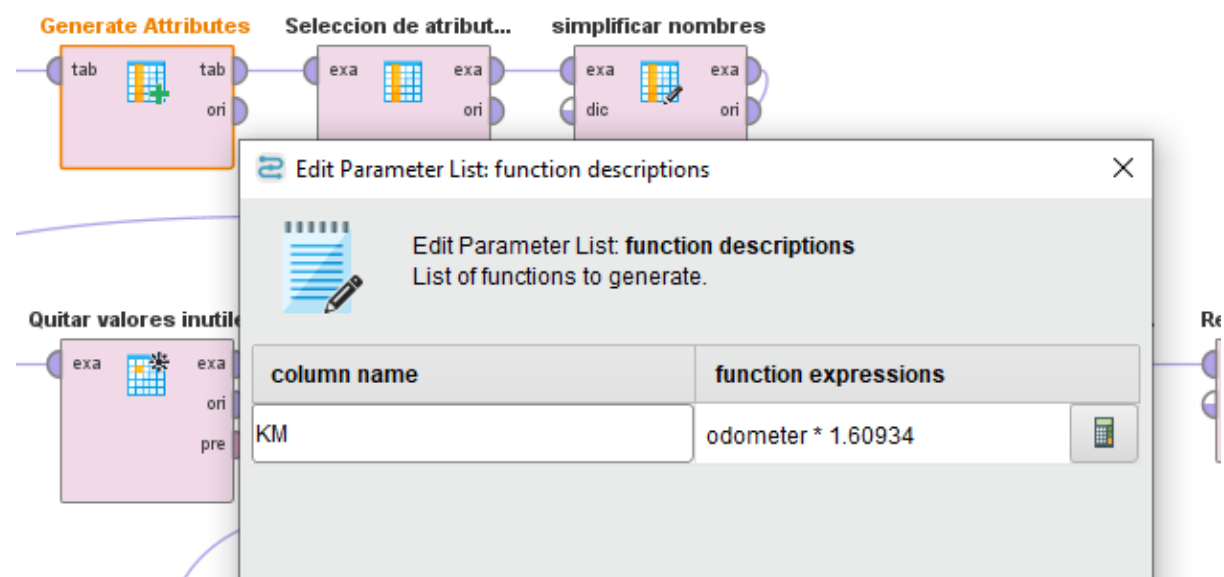
El dataset elegido proviene de [Kaggle](#), más concretamente del conjunto "Craigslist Cars and Trucks Data". Contiene más de 200.000 registros sobre vehículos en venta. Su formato original es CSV y no se recurrió a scraping ni APIs adicionales.

Durante la exploración inicial, se detectaron varios problemas típicos: valores nulos, precios exageradamente altos o bajos, columnas irrelevantes, etc. Al final se consiguió extraer un subconjunto mucho más manejable y coherente para el modelo.

## 3.3 Procesamiento y análisis de datos

### 3.3.1 Limpieza de datos

Lo primero que se hizo fue hacer la conversión de millas a kilómetros, creando un atributo nuevo llamado KM



Después se hizo la selección de atributos más importantes, quitando todos los que no tenían nada que ver con la predicción del precio.

**Selección de atributos importantes** **simplificar nombres**

Select Attributes: **select subset**  
Click to select the attribute subset.

Attributes

Search

- county
- description
- # id
- image\_url
- # lat
- # long
- # odometer
- paint\_color
- posting\_date
- region
- region\_url
- state
- url
- VIN

Selected Attributes

Search

- condition
- cylinders
- drive
- fuel
- # KM
- manufacturer
- model
- # price
- size
- title\_status
- transmission
- type
- # year

Se cambiaron los nombres de algunos atributos para facilitar la comprensión.

**Selección de atribut...** **simplificar nombres**

Edit Parameter List: **rename attributes**  
Use this list to define the renaming of the attributes.

old name	new name
drive	traction
manufacturer	brand
size	carSize
title_status	legalStatus
type	vehicleType

La parte de filtrado es probablemente la más grande:

- Limitación en year para que los años de los coches no sean menores de 1990 ya que había algunos que eran del 1900.

year	≥	1990
year	<	2025

- Quitar valores erróneos por los que se clasificaban algunos coches respecto a la condición.

condition	is not in	2006;2017** ** GS
-----------	-----------	-------------------

- Quitar outliers del kilometraje para que no fuesen mayor a 500500, y así quitar outliers.

KM	≤	500500
----	---	--------

- Limitar el precio por el que se venden los coches ya que había algunos que llegaban al millón o superaban los 500000.

price	>	100
price	≤	200000

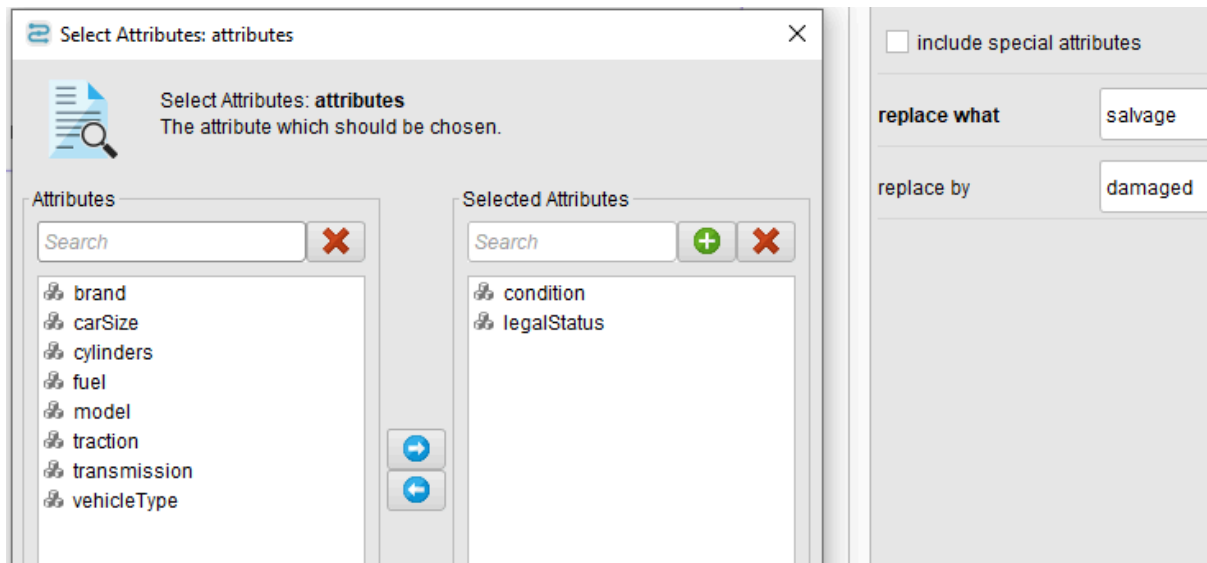
- Por último, quitar los valores nulos de atributos que no tenían tantos missing, por lo que no importaba borrarlos.

brand	is not missing	
model	is not missing	
fuel	is not missing	
KM	is not missing	
transmission	is not missing	

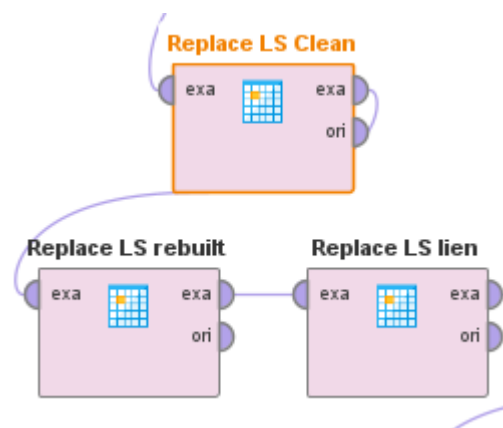
Haciendo toda esta limpieza se pasó de tener 239810 ejemplos a 189282. Gracias a la inmensa cantidad de datos se pudo hacer una limpieza un poco más rápida.



En esta parte se cambia el nombre de una categoría de dos atributos para así poder trabajar más cómodamente con ellos.

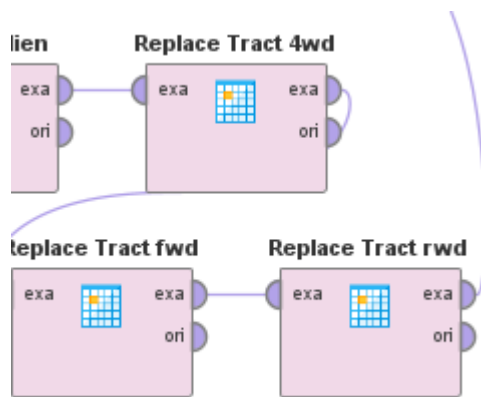


Siguiendo con las últimas partes, se hicieron varios replace para el cambio de nombres en LegalStatus y Traction.



Aquí se guarda como se encuentran los coches legalmente.

replace what	clean	replace what	rebuilt	replace what	lien
replace by	valid	replace by	repaired	replace by	debt



En estos se guarda el tipo de tracción que tiene el vehículo.

replace what	4wd	replace what	fwd	replace what	rwd
replace by	4x4	replace by	frontTraction	replace by	rearTraction

Esta es la selección de clases que pasarán de tener valores missing a unknown.

**Replace Missing Values** Replace ? por clase ... R

Select Attributes: attributes  
The attribute which should be chosen.


Attributes

- brand
- condition
- cylinders
- fuel
- # KM
- legalStatus
- model
- # price
- traction
- transmission
- # year

Selected Attributes


- carSize
- vehicleType

En este caso se hace lo mismo pero con legalStatus, para que pasen a llamarse missing ya que era el nombre de una de las categorías que tenía.

attribute filter type	single
attribute	legalStatus
<input type="checkbox"/> invert selection	
<input type="checkbox"/> include special attributes	
default	value
columns	 Edit List (0)
replenishment value	missing

Por último se reordenan los atributos.

 Select Ordering Rules: attribute ordering

 Select Ordering Rules: **attribute ordering**  
Rules to order attributes.

Attributes

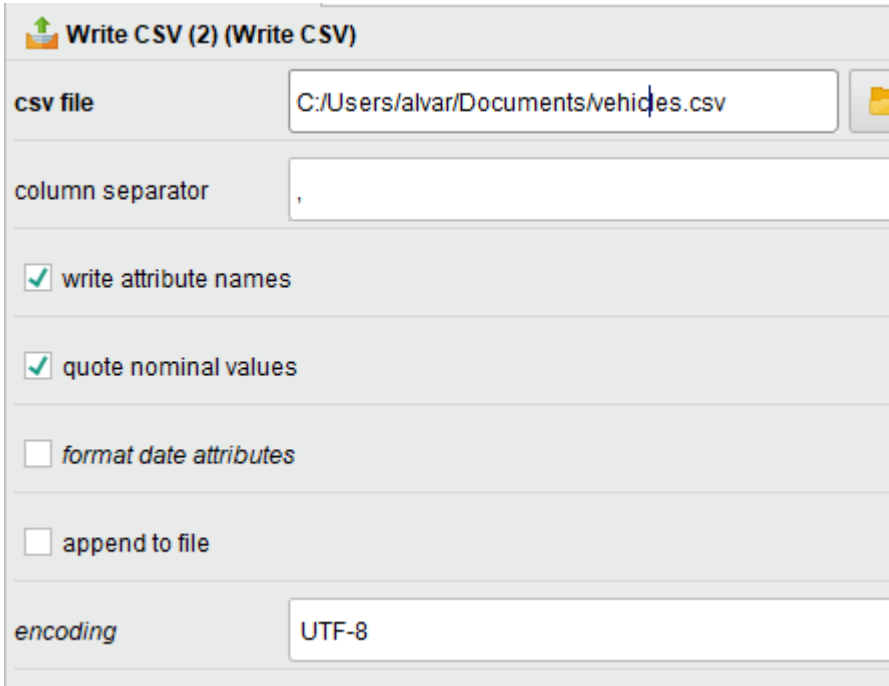
Attribute Ordering

price  
brand  
model  
year  
condition  
legalStatus  
KM  
cylinders  
fuel  
transmission  
traction  
carSize  
vehicleType

→

←

Y se guarda el dataset limpio.



**Write CSV (2) (Write CSV)**

csv file: C:/Users/alvar/Documents/vehicules.csv

column separator: ,

☒ write attribute names

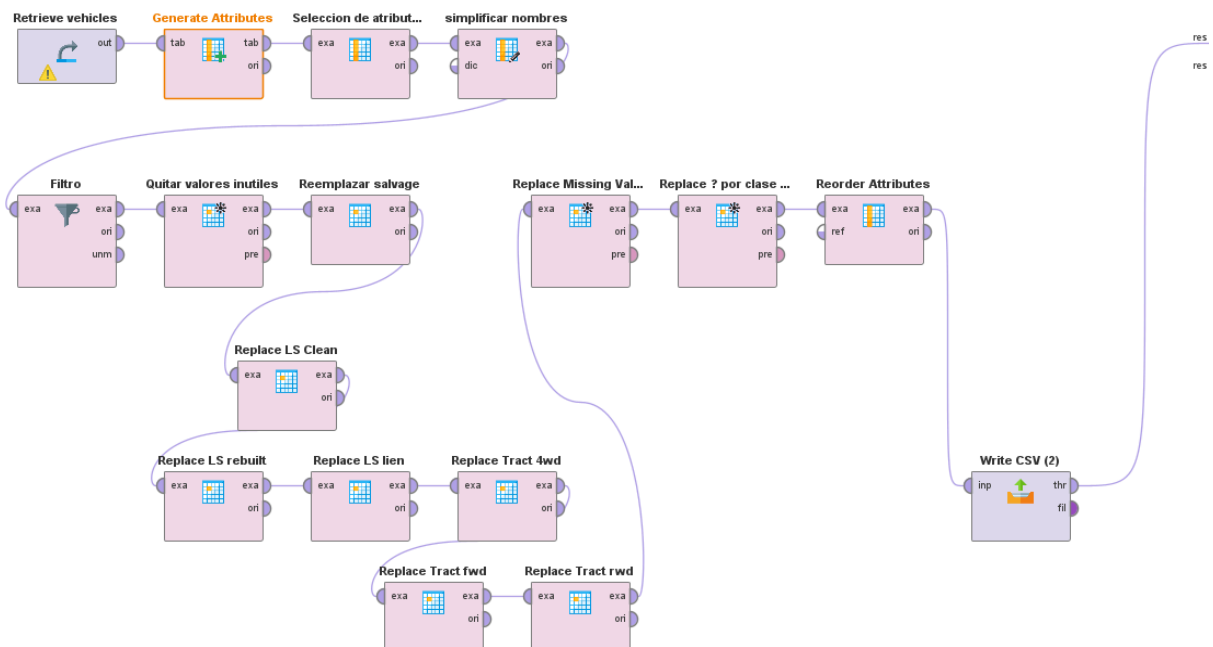
☒ quote nominal values

☐ format date attributes

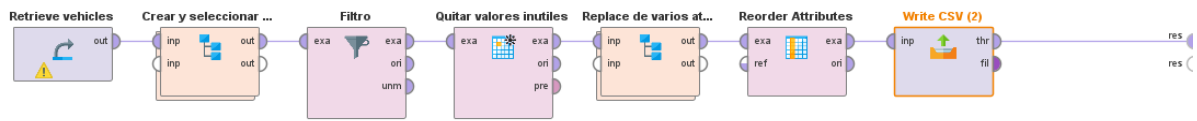
☐ append to file

encoding: UTF-8

Al final el flujo de trabajo se ve así.



Pero con un poco de orden se puede ver así.

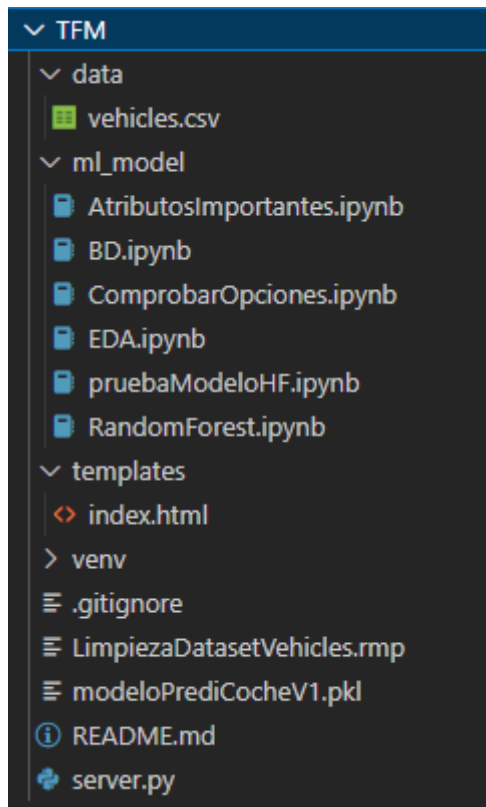


Durante el proceso de limpieza y transformación de datos, se generaron múltiples versiones del dataset. Cada una fue guardada permitiendo mantener un historial de cambios.

Nombre	Fecha de modificación	Tipo	Tamaño
5atrib	10/05/2025 14:47	Hoja de cálculo d...	8.954 KB
9atrib	10/05/2025 14:52	Hoja de cálculo d...	15.227 KB
10atrib	10/05/2025 15:00	Hoja de cálculo d...	17.410 KB
cleanVehicleVers1	09/05/2025 14:22	Hoja de cálculo d...	23.039 KB
Condition	09/05/2025 19:29	Hoja de cálculo d...	22.676 KB
ModelAtrib	10/05/2025 22:34	Hoja de cálculo d...	17.644 KB
PRECIOS	11/05/2025 16:56	Hoja de cálculo d...	15.149 KB
PRECIOS75K	11/05/2025 17:19	Hoja de cálculo d...	15.149 KB
pruebaManual	09/05/2025 19:03	Hoja de cálculo d...	21.566 KB
tractiCondition	09/05/2025 19:26	Hoja de cálculo d...	22.221 KB
vehiclesLimpio	09/05/2025 19:33	Hoja de cálculo d...	23.302 KB
vehiclesLimpioAtrib	09/05/2025 22:06	Hoja de cálculo d...	11.686 KB

Finalmente, la versión que ofreció mejores resultados en términos de precisión fue PRECIO75K (renombrada vehicles.csv), en la cual se aplicó un filtro que limitaba los precios a un rango entre 100 y 75,000 dólares. Esta decisión permitió reducir la dispersión de los datos y mejorar notablemente el rendimiento de los modelos predictivos.

### 3.3.2 Organización de carpetas



Esta es la estructura del proyecto:

- **data**: Es donde se encuentra el dataset limpio.
- **ml\_model**: Es donde se ejecutan todas las notebooks.
- **templates**: Es donde se encuentra el html con el diseño y el javascript
- **LimpiezaDatasetVehicles.rmp**: Es la hoja de Rapidminer en la que se hace la limpieza.
- **modeloPredictCoche.pkl**: Modelo final con el que se hace la predicción del precio.
- **server.py**: Donde se ejecuta todo el backend de la web, junto al modelo creado y al modelo importado de HuggingFace.

### 3.3.2 Creación del modelo

En esta notebook (AtributosImportantes.ipynb) se ven los atributos más importantes para predecir el precio de los vehículos. Los resultados muestran que el año y el kilometraje son los factores con mayor peso, seguidos por el tipo de combustible y la tracción.

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

df = pd.read_csv("../data/vehicles.csv")

# Codificar las variables categoricas
df_encoded = pd.get_dummies(df, drop_first=True)

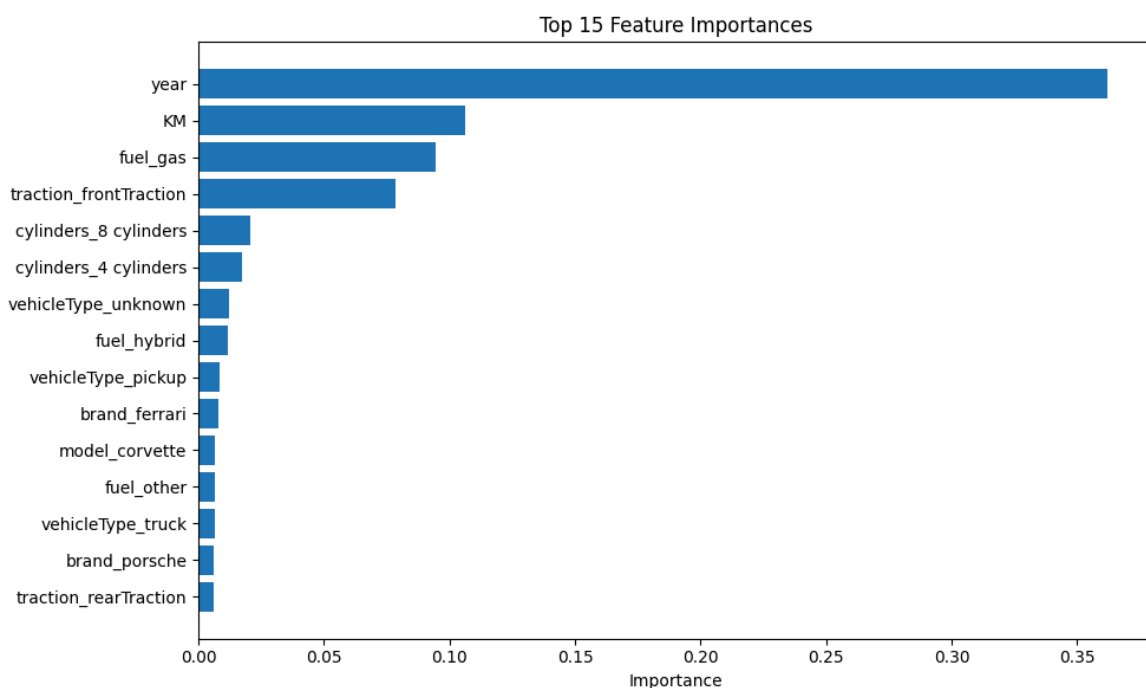
# Separar X e y
X = df_encoded.drop("price", axis=1)
y = df_encoded["price"]

# Separacion de train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Entrenamiento
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Obtener importancias
importances = rf.feature_importances_
feature_names = X.columns
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
importance_df = importance_df.sort_values(by="Importance", ascending=False)

# Mostrar el grafico
plt.figure(figsize=(10,6))
plt.barh(importance_df['Feature'][:15][::-1], importance_df['Importance'][:15][::-1])
plt.title("Top 15 Feature Importances")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()
```



En esta notebook (ComprobarOpciones.ipynb) se hace la comparativa entre los modelos para ver cual es el mejor. El modelo que obtuvo mejores resultados fue Random Forest, con un mayor coeficiente  $R^2$  y menor error en las predicciones respecto a los demás

```
# Cargar de datos
df = pd.read_csv("../data/vehicles.csv")

# Codificación
df_encoded = pd.get_dummies(df, drop_first=True)

# Separar precio
X = df_encoded.drop("price", axis=1)
y = df_encoded["price"]

# Dividir train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Normalización
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Modelos diferentes para hacer la comparación
models = {
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(n_estimators=100, random_state=42), # Ganador
    "Gradient Boosting": GradientBoostingRegressor(n_estimators=100, random_state=42),
    "XGBoost": xgb.XGBRegressor(n_estimators=100, random_state=42)
}

# Entrenamiento
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    preds = model.predict(X_test_scaled)

    mae = mean_absolute_error(y_test, preds)
    rmse = np.sqrt(mean_squared_error(y_test, preds))
    r2 = r2_score(y_test, preds)

    print(f"\n-- {name} --")
    print(f"R²: {r2:.3f}")
    print(f"MAE: {mae:.2f}")
    print(f"RMSE: {rmse:.2f}")
```

```
-- Decision Tree --
R²: 0.714
MAE: 3409.48
RMSE: 6448.26

-- Random Forest --
R²: 0.835
MAE: 2777.78
RMSE: 4888.08

-- Gradient Boosting --
R²: 0.719
MAE: 4176.82
RMSE: 6386.58

-- XGBoost --
R²: 0.801
MAE: 3433.85
RMSE: 5371.43
```



En la notebook EDA.ipynb nos encontramos con código para generar las gráficas que se pueden ver en el punto [4.1 del documento](#).

En esta notebook (RandomForest.ipynb) se crea el modelo final, el que se usará en la web

```
df = pd.read_csv("../data/vehicles.csv")

# Variables
features = ['brand', 'condition', 'cylinders', 'fuel', 'KM', 'traction', 'vehicleType', 'year']
target = 'price'

# Codificar variables categóricas
df_encoded = pd.get_dummies(df[features], drop_first=True)
X = df_encoded
y = df[target]

# Separar para train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Entrenamiento
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predicciones
y_pred = rf.predict(X_test)

# Métricas para la evaluación
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

# Resultados
print("Random Forest")
print(f"R²: {r2:.3f}")
print(f"MAE: {mae:.2f}")
print(f"RMSE: {rmse:.2f}")
```

En esta parte se analiza la distribución del error entre los precios reales y los predichos por el modelo. La mayoría de los errores se concentran cerca del 0, lo que indica que las predicciones fueron bastante ajustadas al valor real en la mayoría de los casos.

## Predicciones del modelo

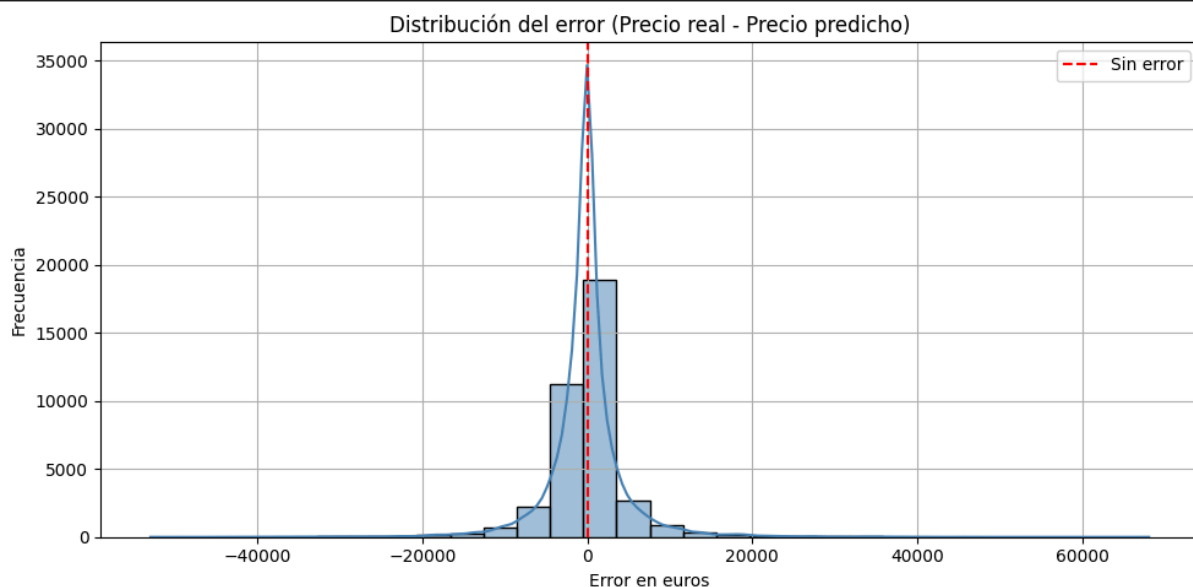
```
features = ['brand', 'condition', 'cylinders', 'fuel', 'KM', 'traction', 'vehicleType', 'year']
target = 'price'

# Codificar variables categóricas
df_encoded = pd.get_dummies(df[features], drop_first=True)
X = df_encoded
y = df[target]

# Dividir train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

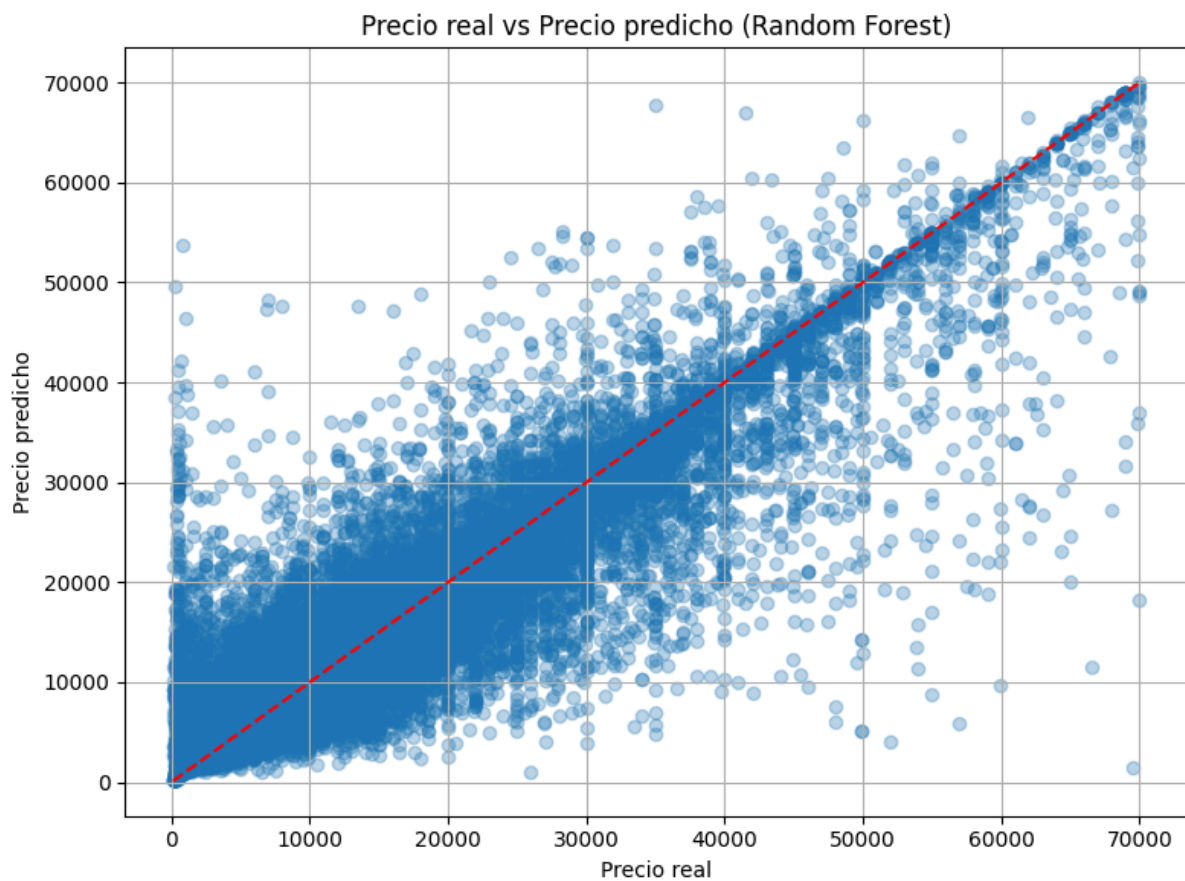
# Calcular errores
errores = y_test - y_pred

# Graficar distribución del error
plt.figure(figsize=(10, 5))
sns.histplot(errores, kde=True, bins=30, color='steelblue')
plt.title("Distribución del error (Precio real - Precio predicho)")
plt.xlabel("Error en euros")
plt.ylabel("Frecuencia")
plt.axvline(0, color='red', linestyle='--', label="Sin error")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Aquí se genera una gráfica de dispersión para comparar visualmente los precios reales con los predichos por el modelo Random Forest. Los puntos cercanos a la línea indican buenas predicciones, y en general se observa una alineación bastante aceptable, lo que refleja un desempeño sólido del modelo.

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.3)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--') # línea ideal
plt.xlabel("Precio real")
plt.ylabel("Precio predicho")
plt.title("Precio real vs Precio predicho (Random Forest)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



En este apartado simplemente se guarda el modelo

## Guardado del modelo

```
# Guardar el modelo
joblib.dump(rf, '../modeloPrediCoche.pkl')
```

### 3.3.3 Guardado de los modelos en MYSQL

En la notebook BD.ipynb se hace el guardado de las métricas de los diferentes modelos para luego ser mostradas en Grafana. Se hace la creación de tablas y el insert con los datos.

```
# Conexion
conn = mysql.connector.connect(
    user="alv_mar",
    password="16_alv_mar_93",
    host="82.165.173.36",
    port="31234"
)
c = conn.cursor()

# Usar la bd de Rodolfo
c.execute("USE bd_alv_mar_graf1")
conn.commit()

# Creacion de tablas
c.execute('''
CREATE TABLE IF NOT EXISTS experimentos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    modelo VARCHAR(50),
    fecha DATETIME DEFAULT CURRENT_TIMESTAMP
);
''')

c.execute('''
CREATE TABLE IF NOT EXISTS metricas_modelos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_experimento INT,
    r2 FLOAT,
    mae FLOAT,
    rmse FLOAT,
    FOREIGN KEY (id_experimento) REFERENCES experimentos(id)
);
''')

conn.commit()
```

```

# Modelos y las metricas que han dado
modelos = [
    ("Decision Tree", 0.661, 4339.98, 6483.49),
    ("Gradient Boosting", 0.763, 3265.88, 5469.47),
    ("XGBoost", 0.778, 3087.79, 5291.72),
    ("Random Forest", 0.835, 2777.78, 4888.08)
]

# Insercion de cada modelo en la tabla
for modelo, r2, mae, rmse in modelos:
    c.execute("INSERT INTO experimentos (modelo) VALUES (%s)", (modelo,))
    conn.commit()
    id_exp = c.lastrowid

    # Insercion de las metricas
    c.execute("""
        INSERT INTO metricas_modelos (id_experimento, r2, mae, rmse)
        VALUES (%s, %s, %s, %s)
    """, (id_exp, r2, mae, rmse))
    conn.commit()

print("Todos los modelos guardados correctamente.")

c.close()
conn.close()

```

### 3.3.4 Prueba con modelo de HuggingFace

En esta la notebook pruebaModeloHF.ipynb se hacen las pruebas con el modelo que se usará en la web para hacer la lectura de comentarios.

```

# Modelos de la web Hugging Face
# bhadresh-savani/bert-base-uncased-emotion
# nateraw/bert-base-uncased-emotion

# Traducir frase al inglés
frase = "me alegra mucho que haya paginas asi"
frase_en = GoogleTranslator(source='auto', target='en').translate(frase)

# Cargar pipeline de Hugging Face
clasificador = pipeline("text-classification", model="nateraw/bert-base-uncased-emotion")

# Predecir emoción
resultado = clasificador(frase_en)

# Mostrar resultado
print("Frase original:", frase)
print("Traducción:", frase_en)
print("Emoción detectada:", resultado[0]['label'])

```

```

Device set to use cpu
Frase original: me alegra mucho que haya paginas asi
Traducción: I'm very happy that there are pages like this
Emoción detectada (modelo BERT): joy

```

También se comprobó cuantas emociones cargaba el modelo para poder trabajar mejor en el javascript de la web

```
# Prueba para ver cuantas emociones coge el modelo
resultado = clasificador("I don't know what to feel anymore", top_k=None)

for emocion in resultado:
    print(f"{emocion['label']}: {emocion['score']:.2f}")
```

```
Device set to use cpu
sadness: 0.97
fear: 0.02
joy: 0.00
anger: 0.00
love: 0.00
surprise: 0.00
```

### 3.3.5 Web

Dentro de index.html se encuentra todo el frontend de la web. En la captura se puede ver parte del código javascript en el que se hace la petición al back con todos los datos del formulario que se necesita para usar el modelo.

```
const data = {
  brand: compBrand,
  model: compModel,
  vehicle_type: compVehiT,
  year: compYear,
  KM: compkmValue,
  condition: compCondi,
  traction: compTract,
  fuel_type: compFuelT,
  cylinders: compCylin,
  comments: compComme,
};

try {
  // Mostrar spinner de carga
  document.getElementById('loadingContainer').classList.remove('hidden');
  document.getElementById('resultContainer').classList.add('hidden');

  // Enviar los datos recogidos al backend
  const res = await axios.post('http://127.0.0.1:5000/precidir', data);

  // Mostrar resultado del modelo
  console.log(res.data.precio_estimado)
  document.getElementById('predictedPrice').textContent = `€${res.data.precio_estimado}`;

  document.getElementById('loadingContainer').classList.add('hidden');
  document.getElementById('resultContainer').classList.remove('hidden');

  // Rellenar los datos con los que se ve un resumen del coche
  document.getElementById('carTitle').textContent = `${data.brand} ${data.model} ${data.year}`;
  document.getElementById('resBrand').textContent = data.brand;
  document.getElementById('resModel').textContent = data.model;
  document.getElementById('resYear').textContent = data.year;
  document.getElementById('resKM').textContent = `${data.KM.toLocaleString()} km`;
```

Esta es una parte del backend, donde se usa el modelo y procesan los datos que llegan del front. Los datos deben ser procesados ya que llegan como string.

```
# instalaciones necesarias pip install transformers flask pandas joblib flask-cors
# pip install deep_translator
# pip install scikit_learn ns si esta bien escrito
# pip install torch

app = Flask(__name__)
CORS(app)

# Carga del modelo
modelo = joblib.load('modeloPrediCoche.pkl')
columnas_modelo = modelo.feature_names_in_

# Ruta para el main de la web y /predecir
@app.route('/')
def home():
    return render_template('index.html')

# Uso del modelo para mandar el precio estimado
@app.route('/predecir', methods=['POST'])
def predecir():
    # Datos reecibidos del frontend
    data = request.get_json()
    df_entrada = pd.DataFrame([data])
    df_transformado = pd.DataFrame([[0]*len(columnas_modelo)], columns=columnas_modelo)
    # cambio de los nombres ya que el modelo usa unos diferentes
    for col in df_entrada.columns:
        val = df_entrada[col][0]
        if col in ['brand', 'fuel', 'condition', 'traction', 'vehicleType', 'cylinders']:
            nombre_col = f"{col}_{val}"
            if nombre_col in df_transformado.columns:
                df_transformado[nombre_col] = 1
        elif col in ['year', 'KM', 'mileage']:
            if col == 'mileage' and 'KM' in df_transformado.columns:
                df_transformado['KM'] = val
            elif col in df_transformado.columns:
                df_transformado[col] = val
    # Uso del modelo con los datos recibidos y procesados
    try:
        prediccion = modelo.predict(df_transformado)
```

### 3.4 Desarrollo del modelo de IA

Se probaron varios algoritmos: Árbol de Decisión, Random Forest, Gradient Boosting y XGBoost. Cada uno se evaluó con métricas como MAE, RMSE y  $R^2$ . Y aunque todos aportaron algo, Random Forest fue el mejor ya que su precisión era alta, su comportamiento estable, y además era bastante rápido de entrenar.

```
-- Decision Tree --  
R2: 0.714  
MAE: 3409.48  
RMSE: 6448.26  
  
-- Random Forest --  
R2: 0.835  
MAE: 2777.78  
RMSE: 4888.08  
  
-- Gradient Boosting --  
R2: 0.719  
MAE: 4176.82  
RMSE: 6386.58  
  
-- XGBoost --  
R2: 0.801  
MAE: 3433.85  
RMSE: 5371.43
```

El dataset fue dividido en un 80% para entrenamiento y 20% para pruebas. No se aplicó validación cruzada por razones de eficiencia, ya que el volumen de datos era grande y el enfoque estaba más orientado a la aplicación práctica que a la optimización.

## 4. Resultados

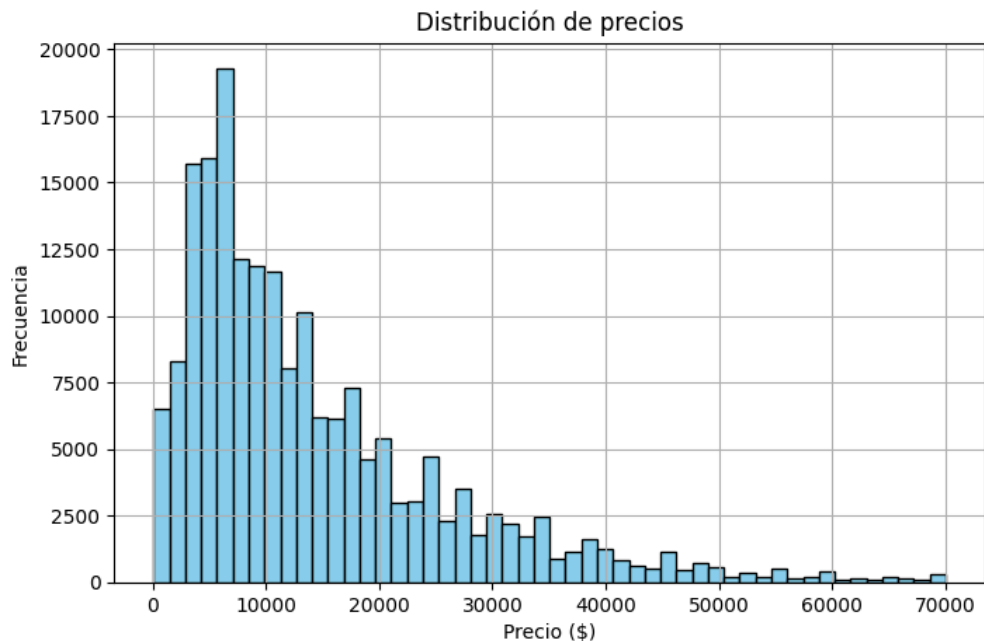
### 4.1 Resultados obtenidos

Con un  $R^2$  de 0.835 y un MAE bajo, este modelo se posicionó como el más preciso. XGBoost también mostró un rendimiento sólido, pero algo por debajo. Los modelos de Gradient Boosting y Árbol de Decisión, aunque útiles para probar y comparar, quedaron más rezagados.

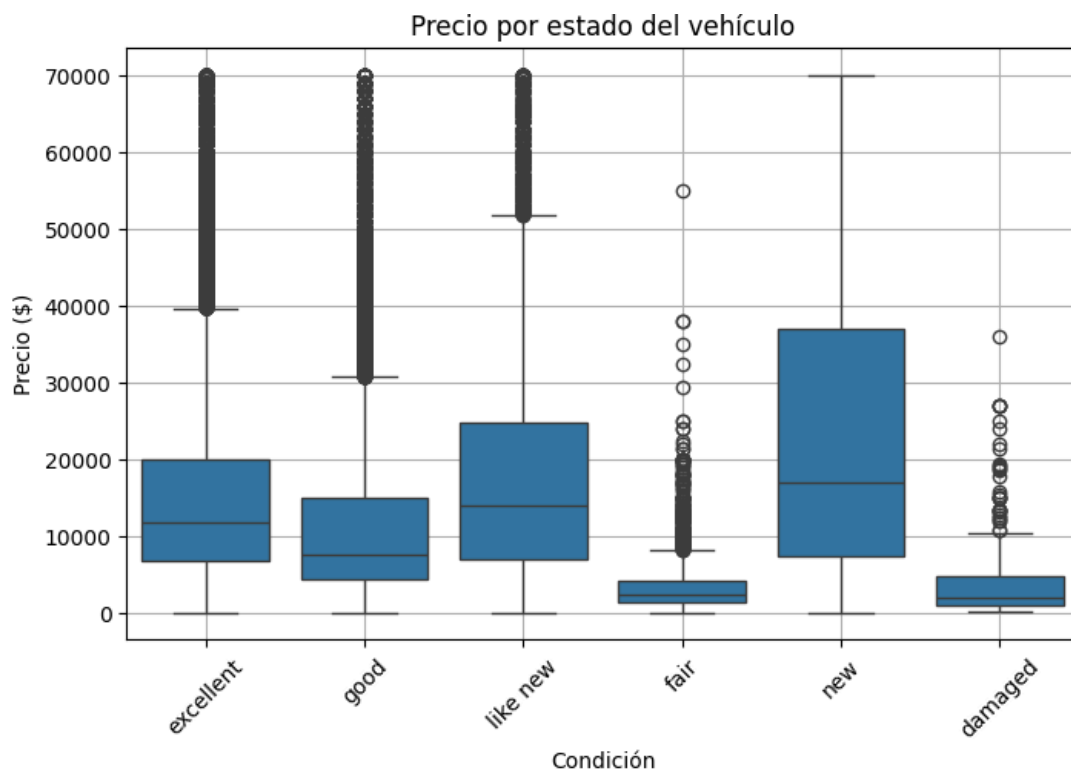


También se realizó un análisis exploratorio apoyado en visualizaciones. Esto ayudó a detectar valores atípicos, relaciones entre variables y a guiar la selección de características.

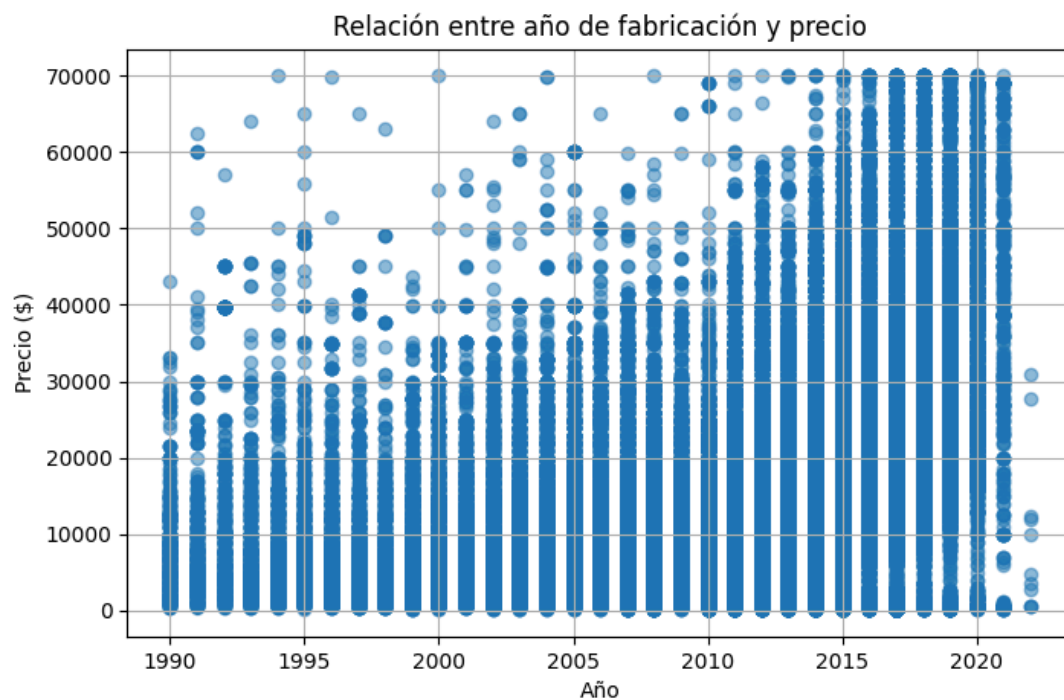
Se observa una distribución asimétrica a la derecha, donde la mayoría de los vehículos tienen un precio inferior a 20,000 \$. Esta observación justificó aplicar un filtrado en el dataset, estableciendo un límite máximo de 75,000 \$ para reducir la influencia de outliers.



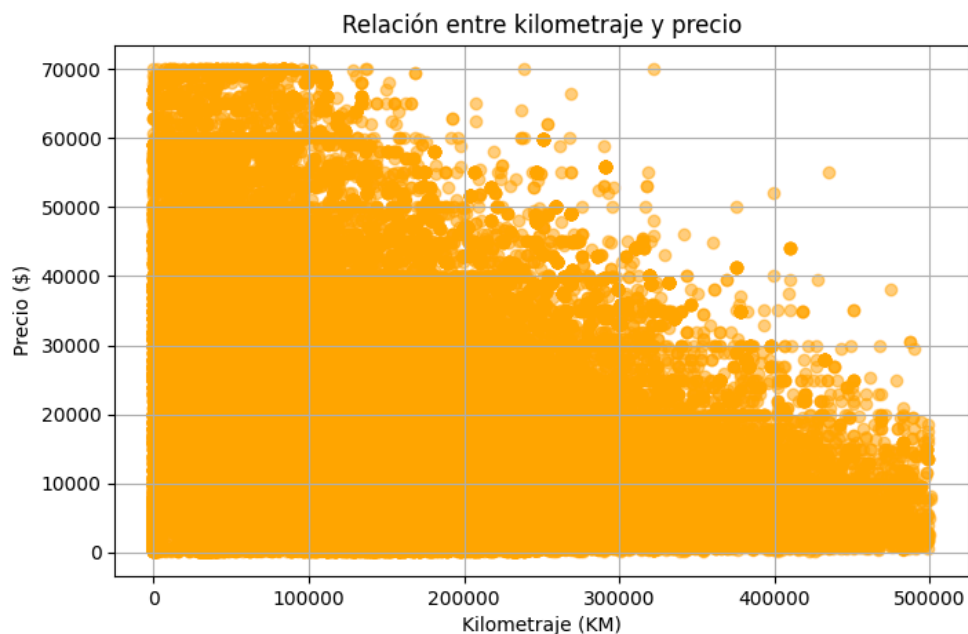
El estado del vehículo (condition) tiene un efecto significativo sobre el precio. Los coches nuevos y como nuevos presentan los precios más altos, mientras que los dañados tienen los precios más bajos.



Existe una relación positiva entre el año de fabricación y el precio. Los vehículos más recientes tienden a tener precios más altos. Se aprecia una gran concentración de registros entre los años 2005 y 2022.



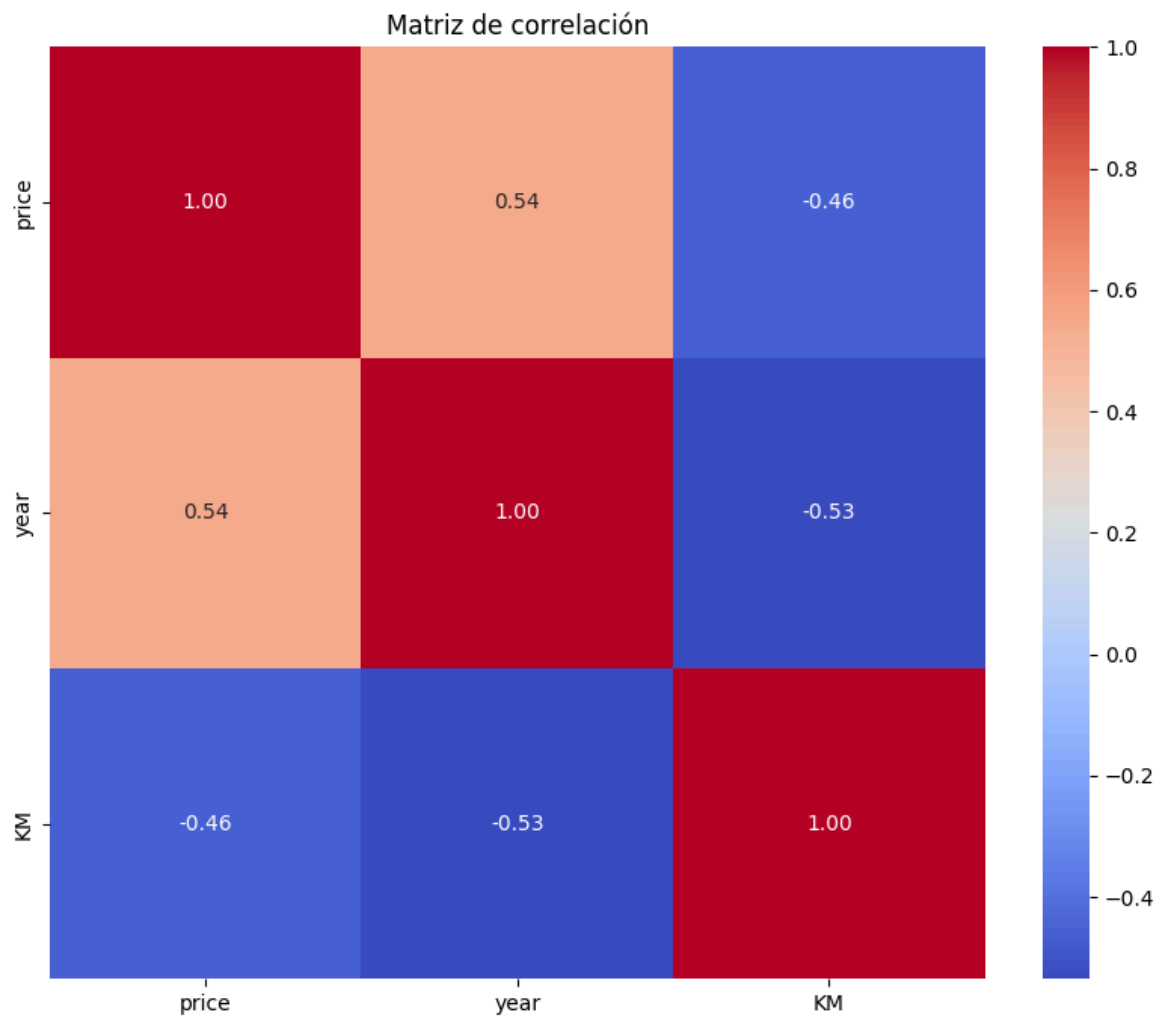
Como se puede ver hay una correlación negativa entre el kilometraje (KM) y el precio: cuanto mayor es el uso del vehículo, menor es su valor estimado. Esta relación también sugiere que KM es una variable relevante.



También generó una matriz de correlación para cuantificar las relaciones entre las variables numéricas, confirmando:

- price y year están moderadamente correlacionados de forma positiva ( $r = 0.54$ )
- price y KM tienen una correlación negativa ( $r = -0.46$ )
- year y KM también presentan correlación negativa ( $r = -0.53$ )

Esto valida visualmente las observaciones anteriores.



Además, se visualizó la distribución de errores para entender mejor dónde fallaban los modelos, y se descubrió que la mayoría de los errores más grandes ocurrían en coches con precios extremos o atributos poco frecuentes.

## **4.2 Comparativa con objetivos iniciales**

El modelo elegido ofrece resultados consistentes, y la interfaz web facilita la interacción sin necesidad de conocimientos técnicos.

También se consiguió almacenar todas las métricas en MySQL y mostrar los resultados de forma visual en Grafana.

## **4.3 Limitaciones de los resultados**

Por supuesto, el modelo no es infalible del todo. A pesar del buen rendimiento general, existen varios factores que lo limitan. Uno de ellos es la calidad de los datos: algunos registros, aunque filtrados, siguen sin representar fielmente el mercado real. También hay atributos que no se contemplan, como el mantenimiento del coche o su historial de accidentes, que claramente influirían en su valor.

Además, no se aplicó validación cruzada ni se exploraron técnicas de optimización exhaustiva por cuestiones de tiempo y recursos. Esto se considera una mejora pendiente.

# **5. Conclusiones y Futuras Líneas de Trabajo**

## **5.1 Conclusiones principales**

Después de todo esto, se puede decir que el proyecto ha cumplido con creces su propósito: ofrecer una estimación razonable y fundamentada del precio de un coche usado.

El modelo seleccionado (Random Forest) ha mostrado un rendimiento sólido, y el sistema, en conjunto, funciona de manera fluida, con una interfaz clara y una visualización potente de resultados. Todo esto ha sido posible gracias a una integración bien pensada entre diferentes tecnologías.

## **5.2 Impacto del proyecto**

PredictCar puede ayudar a usuarios particulares, concesionarios o incluso plataformas de compraventa a tener una referencia objetiva del valor de un coche. Y es que, en un mercado tan variable como el de los vehículos usados, contar con una herramienta así puede marcar la diferencia.

Además, el enfoque modular y reproducible del proyecto permite repetir experimentos fácilmente, actualizar modelos con nuevos datos o incluso integrar nuevas funcionalidades.

### 5.3 Líneas de trabajo futuro

Existen muchas posibilidades de mejora y expansión. Algunas de las más prometedoras son:

- Incluir nuevos tipos de vehículos, como motos o bicicletas.
- Conectar el sistema a APIs para recibir datos en tiempo real.
- Añadir explicaciones más claras sobre las predicciones del modelo.
- Desplegar el sistema en la nube para hacerlo accesible desde cualquier parte.

## 6. Bibliografía

**Grafana Labs.** (n.d.). *Grafana Documentation*. <https://grafana.com/docs/>

**MySQL.** (n.d.). *MySQL Reference Manual*. <https://dev.mysql.com/doc/>

**Reese, A. (2021).** *Craigslist Cars and Trucks Data* [Dataset]. Kaggle. <https://www.kaggle.com/datasets/austinreese/craigslist-carstrucks-data>

The Pallets Projects. (n.d.). *Flask Documentation*. <https://flask.palletsprojects.com/>

**Nateraw. (2020).** *nateraw/bert-base-uncased-emotion* [Modelo de lenguaje]. Hugging Face. <https://huggingface.co/nateraw/bert-base-uncased-emotion>

Parte del desarrollo y redacción del presente trabajo ha contado con el apoyo de herramientas de inteligencia artificial generativa (como ChatGPT), utilizadas como asistencia para estructurar ideas, depurar código y mejorar la expresión escrita.