Proyecto 1 - Predicción de precios de vehículos usados

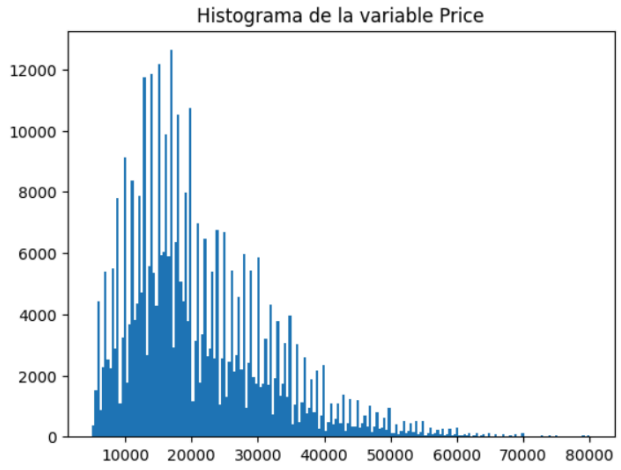
Viviana Villamil - Sebastián Barrera - Kevin Sánchez

1. Exploración y Procesamiento de datos

En este proyecto se utilizará el conjunto de datos de Car Listings de Kaggle, donde cada observación representa el precio de un automóvil usado en Estados Unidos. En primer lugar, se llevó a cabo un análisis exploratorio de los datos, en el cual se identificaron 6 variables principales en la base de datos destinada a la creación y entrenamiento del modelo:

- **Price:** Precio del automóvil usado. (Variable Numérica)
- **Year:** Año de fabricación del automóvil usado. (Variable Numérica)
- **Mileage:** Distancia recorrida por el automóvil en millas. (Variable Numérica)
- **State**: Estado de la placa del automóvil. (Variable Categórica)
- **Make:** Fabricante del automóvil. (Variable Categórica)
- **Model:** Modelo del automóvil. (Variable Categórica)

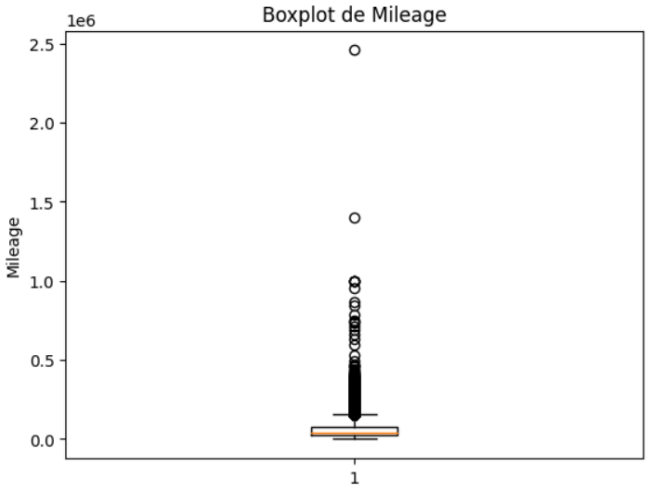
El conjunto de datos consta de un total de 400,000 observaciones. Al observar la variable a predecir, que es el precio del automóvil, se puede apreciar su distribución en el siguiente histograma:



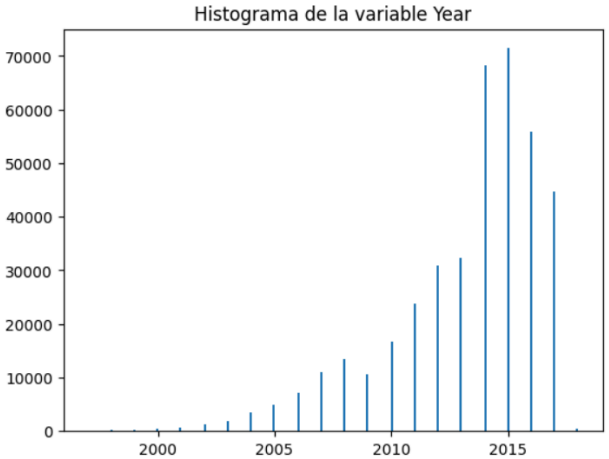
Como se puede observar en el gráfico, las observaciones parecen concentrarse en el lado izquierdo, donde el precio medio de los automóviles en la muestra es de 21,146 dólares. El automóvil con el precio más bajo en la muestra tiene un valor de 5,001 dólares, mientras que el de mayor precio alcanza los 7,999 dólares.

Al examinar las otras variables, notamos que en la variable "Mileage" los datos están extremadamente concentrados en la zona izquierda, lo que resulta en una

gran cantidad de observaciones fuera del percentil 75 de la muestra, como se puede observar a continuación:



Dentro de la variable "Mileage", observamos que el mínimo es de 5 millas, el percentil 25 es de 25,841 millas y el percentil 75 es de 77,433 millas. Para completar el análisis de las variables numéricas, nos queda la variable "Year". En esta variable, se observa que los automóviles de la muestra tienen un rango desde 1997 hasta 2018, distribuidos de la siguiente manera:

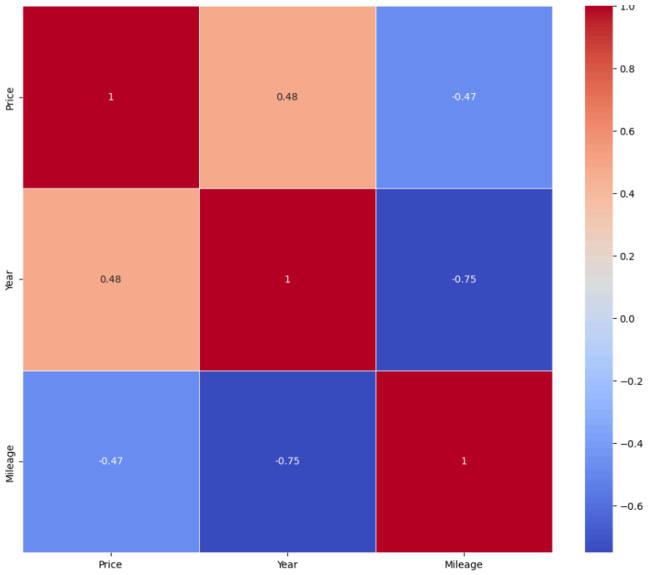


En el análisis de las variables categóricas, al examinar la variable "State", se observa que las tres principales ubicaciones de registro de vehículos en la base de autos usados son Texas, California y Florida. Estos estados representan conjuntamente el 29% de la muestra, la cual incluye un total de 51 estados, destacando que la mayoría de los vehículos pertenecen al periodo entre 2014 y 2017.

Para la variable "Make", se identifican 38 categorías, siendo las más frecuentes Ford, Chevrolet, Toyota y Honda, cada una con más de 30,000 autos registrados en la muestra.

Por último, la variable "Model" cuenta con 525 categorías, que registran los modelos de los distintos fabricantes de automóviles.

Al revisar las correlaciones entre nuestras variables numéricas, identificamos una relación negativa de -0.75 entre la variable "Mileage" y la variable "Year". Esto tiene sentido, ya que a medida que disminuye el tiempo transcurrido desde la fabricación del automóvil, es esperable que la distancia recorrida en millas sea menor.



Para el procesamiento de datos, hemos decidido convertir las variables categóricas en valores booleanos (Verdadero / Falso) para simplificar su cuantificación en el modelo. Antes de aplicar este método, hemos utilizado la función "Strip" en la columna "State" para eliminar espacios en blanco.

```
dataTraining['State'] = dataTraining['State'].str.strip()
# Convertir variables categóricas en variables dummy
dataTraining = pd.get_dummies(dataTraining, columns=['State', 'Make', 'Model'])

# Visualizar los primeros registros del conjunto de datos preprocesado
print(dataTraining.head())
```

Finalmente, llevamos a cabo la partición de los datos, asignando el 80% del conjunto de datos para el entrenamiento del modelo y el 20% restante para la evaluación del mismo.

```
# Definir las características (X) y la variable objetivo (y)
X = dataTraining.drop('Price', axis=1)
y = dataTraining['Price']

# Dividir los datos en conjuntos de entrenamiento y prueba
XTrain, XTest, yTrain, yTest = train_test_split(X, y, test_size=0.2, random_state=42)
```

2. Calibración del modelo

Dado que el objetivo del proyecto es desarrollar un modelo capaz de predecir el precio de un automóvil usado en Estados Unidos, hemos explorado varios modelos para determinar cuál ofrece el mejor rendimiento en esta tarea. Evaluamos el poder predictivo de cuatro modelos diferentes: árboles de decisión, Random Forest,

Gradient Boosting y XGBoost. Se mantienen los hiperparámetros por defecto para cada modelo.

```
# Inicializar modelos
decision_tree = DecisionTreeRegressor(random_state=42)
random_forest = RandomForestRegressor(random_state=42)
gradient_boosting = GradientBoostingRegressor(random_state=42)
xgboost = XGBRegressor(random_state=42)

# Entrenar modelos con la mejor combinación de hiperparámetros
decision_tree.fit(XTrain, yTrain)
random_forest.fit(XTrain, yTrain)
gradient_boosting.fit(XTrain, yTrain)
xgboost.fit(XTrain, yTrain)

# Imprimir mensaje de finalización
print("Entrenamiento completado.")
```

- ***Árboles de Decisión***
  Los árboles de decisión son modelos supervisados de Machine Learning que buscan predecir una variable objetivo utilizando reglas de decisión binarias sobre las variables predictoras. Una de sus ventajas principales es su interpretabilidad, ya que permiten comprender fácilmente cómo se toman las decisiones. Además, estos modelos pueden manejar variables categóricas, y son aplicables tanto a problemas de clasificación como de regresión. Otra ventaja significativa es su robustez ante datos atípicos o valores nulos en los conjuntos de datos.

- ***Random Forest***
  Random Forest es un algoritmo de Machine Learning supervisado que utiliza Bagging de árboles con un ajuste adicional. Es ampliamente utilizado por su excelente rendimiento en problemas predictivos. Ofrece ventajas como una estimación precisa de la importancia de las variables predictoras y la capacidad de estimar el error fuera de la muestra. Sin embargo, tiene la desventaja de requerir más tiempo para el entrenamiento y la predicción en comparación con otros modelos.

- ***Gradient Boosting***
  Gradient Boosting es una técnica que mejora el rendimiento de AdaBoost y adopta conceptos de la agregación Bootstrap para mejorar aún más los modelos. Implica tres elementos principales: una función de pérdida para la optimización, un "Weak Learner" para las predicciones y un modelo aditivo para combinar los "Weak Learners" y minimizar la función de pérdida. La función de pérdida evalúa la discrepancia entre las predicciones y los valores reales. Los "Weak Learners" son modelos básicos que, aunque tienen un rendimiento limitado por sí solos, se combinan para formar modelos más predictivos. En el modelo aditivo, los "Weak Learners" se agregan uno por

uno utilizando el método de gradiente descendente para minimizar la pérdida y mejorar la precisión del modelo.

- ***XGBoost***
  Extreme Gradient Boosting, conocido como XGBoost, es un algoritmo de aprendizaje automático supervisado que se basa en ensamblajes de árboles de decisión. Es ampliamente reconocido en la comunidad de Machine Learning por su alta eficiencia, escalabilidad y precisión en la predicción. XGBoost sigue el principio de construir secuencialmente un ensamblaje de árboles de decisión, donde cada "Weak Learner" aprende de los errores anteriores para mejorar la predicción. Introduce mejoras algorítmicas y de optimización del sistema, como la regularización para evitar el sobreajuste de modelos complejos y mejorar la generalización a datos nuevos. Además, realiza validación cruzada en cada iteración, eliminando la necesidad de especificar manualmente el número de validaciones. Por último, XGBoost es capaz de manejar valores nulos en los datos, aprendiendo automáticamente el mejor valor según la pérdida durante el entrenamiento.

```
# Evaluar el rendimiento en el conjunto de prueba
y_pred_dt = decision_tree.predict(XTest)
mse_dt = mean_squared_error(yTest, y_pred_dt)

y_pred_rf = random_forest.predict(XTest)
mse_rf = mean_squared_error(yTest, y_pred_rf)

y_pred_gb = gradient_boosting.predict(XTest)
mse_gb = mean_squared_error(yTest, y_pred_gb)

y_pred_xgb = xgboost.predict(XTest)
mse_xgb = mean_squared_error(yTest, y_pred_xgb)

# Imprimir las métricas de rendimiento
print("MSE Decision Tree:", mse_dt)
print("MSE Random Forest:", mse_rf)
print("MSE Gradient Boosting:", mse_gb)
print("MSE XGBoost:", mse_xgb)
```

```
MSE Decision Tree: 22432060.730712265
MSE Random Forest: 14099652.58585069
MSE Gradient Boosting: 44904078.2162739
MSE XGBoost: 18135800.237015836
```

Después de comparar los modelos utilizando el error cuadrático medio (MSE), identificamos que Random Forest presenta el mejor rendimiento predictivo seguido de XGBoost.

Pero decidimos seleccionar XGBoost como nuestro modelo final, al seleccionar XGBoost en lugar de Random Forest, se optó por un enfoque que equilibra eficiencia computacional y rendimiento predictivo. XGBoost ha demostrado ser más rápido y eficiente en el procesamiento de grandes conjuntos de datos, lo cual es

crucial para optimizar el tiempo de entrenamiento del modelo. Además, su flexibilidad en la optimización de hiperparámetros y su capacidad para ofrecer un rendimiento ligeramente superior en términos de precisión de predicción fueron factores clave en la decisión.

Esta elección permitirá maximizar la precisión, contribuyendo así a resultados más sólidos y una mejor interpretación del modelo en el contexto del problema de predicción de precios de automóviles usados.

Luego, procedimos a calibrar con mayor detalle los hiperparámetros del modelo XGBoost para optimizar su desempeño.

3. Entrenamiento del modelo

XGBoost superó a los otros algoritmos debido a su capacidad para optimizar la precisión del modelo mientras controla el sobreajuste y mantiene la eficiencia computacional. Sus técnicas de regularización, manejo de datos faltantes y eficiencia en el tiempo de entrenamiento lo hicieron destacar en la reducción del error cuadrático medio (MSE), demostrando su superioridad en la predicción precisa y escalable.

```
# Definir rangos para cada hiperparámetro
param_grid = {
    'booster': ['gbtree'],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'eta': [0.1, 0.2, 0.3],
    'gamma': [0, 1, 5],
    'max_depth': [4, 6, 9],
    'max_leaves': [0],
    'n_estimators': [50, 150, 200, 250, 300, 350, 400, 250, 500, 550, 600, 650, 700],
    'objective': ['reg:linear'],
    'reg_alpha': [0, 1, 2, 3],
    'reg_lambda': [0, 1, 2, 3],
    'subsample': [0.6, 0.7, 0.8],
    'tree_method': ['auto'],
    'random_state': [42]
}

# Inicializar el modelo
xgb_model = XGBRegressor()

# Realizar la búsqueda de cuadrícula
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', verbose=1, n_jobs=-1)

# Entrenar el modelo con los datos de entrenamiento
grid_search.fit(XTrain, yTrain)

# Obtener el mejor modelo
best_xgb_model = grid_search.best_estimator_
```

En el proceso de calibración de hiperparámetros realizado para el modelo de regresión XGBoost en el contexto del análisis de precios de automóviles usados, se empleó una técnica de búsqueda exhaustiva conocida como Grid Search Cross-Validation (CV). Esta estrategia implica la evaluación sistemática de una cuadrícula de combinaciones de hiperparámetros predefinidos, donde cada combinación se prueba utilizando validación cruzada. El objetivo principal es encontrar la combinación óptima de hiperparámetros que maximice el rendimiento

del modelo en términos de una métrica de evaluación específica, en este caso, el error cuadrático medio negativo (neg_mean_squared_error).

Durante la búsqueda de hiperparámetros, se consideraron múltiples aspectos clave del modelo, como el número de árboles en el ensamblado (n_estimators), la profundidad máxima de cada árbol (max_depth), la tasa de aprendizaje (eta), y los parámetros de regularización (reg_alpha y reg_lambda), entre otros. Esta exploración exhaustiva permitió identificar las combinaciones de hiperparámetros que ofrecen un mejor rendimiento predictivo en términos de la métrica seleccionada. Además, se utilizó validación cruzada con un esquema de k-fold (con k = 5) para garantizar una evaluación robusta y evitar el sobreajuste del modelo.

Los mejores hiperparámetros fueron:

```
XGBRegressor(booster="gbtree",colsample_bytree=1,eta=0.3,
    gamma=1,max_depth=9,max_leaves=0,n_estimators=550,
    objective="reg:linear",reg_alpha=2,
    reg_lambda=2,tree_method="auto", subsample=0.7,random_state=42)
```

Además de la búsqueda exhaustiva de hiperparámetros, se realizaron ajustes manuales específicos en variables clave del modelo XGBoost, como número de estimadores, profundidad máxima y regularización de L1 y L2 . Estos ajustes adicionales se basaron en la comprensión del problema e intuición sobre cómo mejorar el rendimiento del modelo. Esta optimización manual complementaria refinó aún más la capacidad predictiva del modelo, resultando en una mejora significativa en la precisión de las predicciones de precios de automóviles usados.

Dejando el modelo con los siguientes hiperparámetros:

```
# Inicializar el modelo con los hiperparámetros ajustados
xgb_model = XGBRegressor(
    booster="gbtree",
    colsample_bytree=1,
    eta=0.3,
    gamma=1,
    max_depth=9,
    max_leaves=0,
    n_estimators=540,
    objective="reg:linear",
    reg_alpha=2.3,
    reg_lambda=1.7,
    tree_method="auto",
    subsample=0.7,
    random_state=42
)

# Entrenar el modelo con los datos de entrenamiento
xgb_model.fit(XTrain, yTrain)

# Imprimir mensaje de finalización
print("Entrenamiento completado.")
```

En el caso de XGBoost, la optimización de hiperparámetros como **n_estimators**, **max_depth**, **reg_alpha** y **reg_lambda** demostro mejorar el modelo. Por ejemplo, **n_estimators** controla el número de árboles en el ensamblado, permitiendo que el modelo capture relaciones más complejas entre las características. Aumentar este valor puede mejorar la capacidad del modelo para generalizar patrones en los datos.

Por otro lado, **max_depth** limita la profundidad máxima de cada árbol en el ensamblado, lo que ayuda a prevenir el sobreajuste al restringir la complejidad del modelo. **reg_alpha** y **reg_lambda** son términos de regularización que controlan la penalización de los coeficientes de las características, lo que ayuda a evitar el sobreajuste al restringir la magnitud de los coeficientes.

A la final obtuvimos una mejoría del modelo, sus métricas de rendimiento:

```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Realizar predicciones en el conjunto de prueba
y_pred = xgb_model.predict(XTest)

# Calcular métricas de rendimiento
mse = mean_squared_error(yTest, y_pred)
mae = mean_absolute_error(yTest, y_pred)
r2 = r2_score(yTest, y_pred)
rmse = np.sqrt(mse)

# Imprimir las métricas de rendimiento
print("Métricas de rendimiento del modelo:")
print("Error cuadrático medio (MSE):", mse)
print("Raíz del error cuadrático medio (RMSE):", rmse)
print("Error absoluto medio (MAE):", mae)
print("Coeficiente de determinación (R^2):", r2)
```

```
Métricas de rendimiento del modelo:
Error cuadrático medio (MSE): 11997074.91404786
Raíz del error cuadrático medio (RMSE): 3463.679389615595
Error absoluto medio (MAE): 2171.490903515625
Coeficiente de determinación (R^2): 0.8963045725274584
```

Después de realizar ajustes en los hiperparámetros del modelo XGBoost, observamos una mejora significativa en el rendimiento del modelo en términos de varias métricas clave de evaluación. El error cuadrático medio (MSE) disminuyó a 11,997,074.91, lo que indica una mayor precisión en las predicciones de los precios de los automóviles usados. Esto se tradujo en una raíz del error cuadrático medio (RMSE) de 3,463.68, lo que implica una desviación menor de las predicciones con respecto a los valores reales. Además, el error absoluto medio (MAE) se redujo a 2,171.49, lo que indica una mejora en la precisión promedio de nuestras predicciones. Finalmente, el coeficiente de determinación (R^2) alcanzó un valor notable de 0.8963, lo que sugiere que nuestro modelo explica aproximadamente el 89.63% de la variabilidad en los precios de los automóviles usados. Estos resultados reflejan una mejora sustancial en la capacidad predictiva del modelo XGBoost después de ajustar cuidadosamente los hiperparámetros, lo que respalda su utilidad en la predicción precisa de precios de automóviles usados.

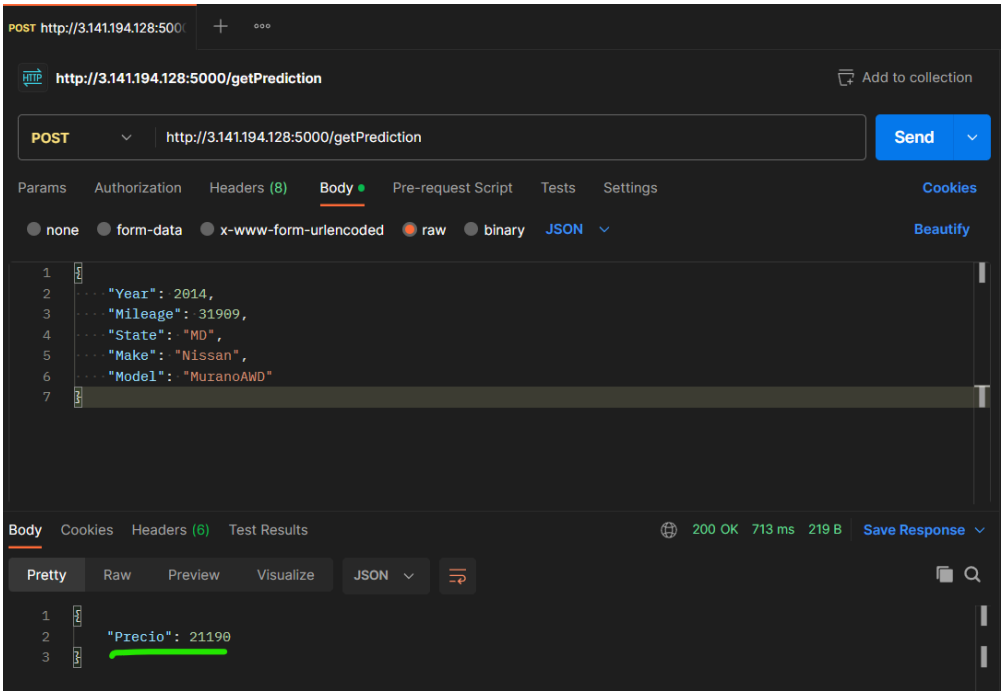
4. Disponibilización del modelo

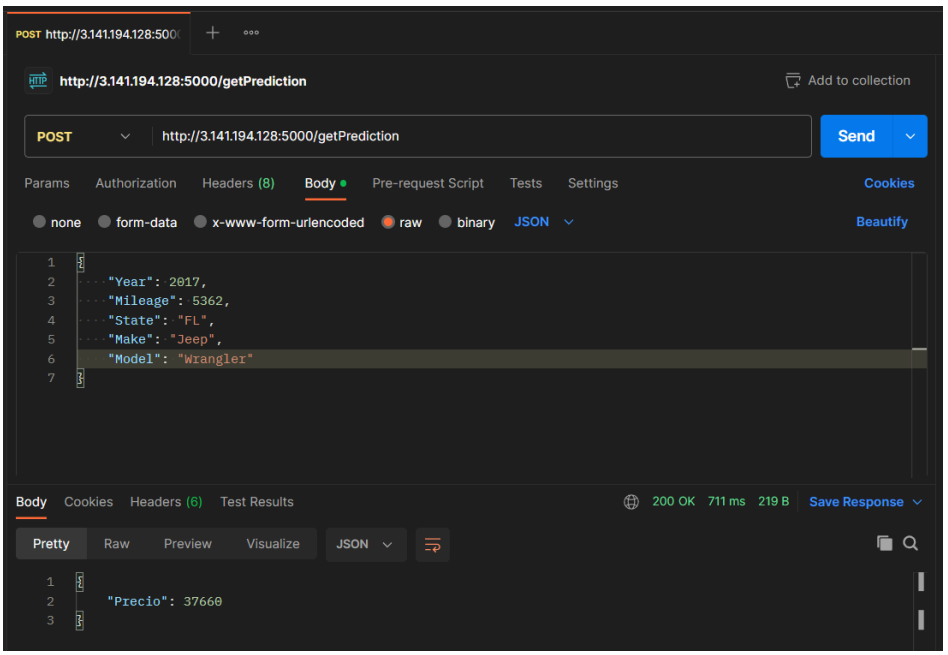
Para hablar sobre la disponibilización del modelo primero es necesario definir que es una API. Una API es una herramienta que hace que los datos de un sitio web sean comprensibles para una computadora. A través de él, una computadora puede ver y editar datos al igual que una persona puede cargar páginas y enviar formularios. Cuando los sistemas se enlazan a través de una API decimos que están integrados. En un lado, el servidor que sirve a la API, y en el otro, el cliente que consume la API y puede manipularla**.**

Después de afinar y optimizar nuestro modelo XGBoost, se realizo el proceso de disponibilizarlo a traves de un API. Para lograr esto, hemos implementado el modelo en una instancia de EC2 de Amazon Web Services (AWS).

Previamente, exportamos el modelo entrenado en formato .pkl para garantizar su portabilidad y compatibilidad. Ahora, para acceder al modelo y realizar predicciones, simplemente debe dirigirse a la URL http://3.141.194.128:5000/getPrediction.
Es importante tener en cuenta que para probar el modelo, se recomienda utilizar **Postman**, ya que la información necesaria se envía un JSON en el cuerpo de la solicitud **POST**, tal como se muestra a continuación con dos observaciones del conjunto de datos de Test:

ID	Year	Mileage	State	Make	Model
0	2014	31909	MD	Nissan	MuranoAWD
1	2017	5362	FL	Jeep	Wrangler





5. Conclusiones

Tras completar este proyecto de predicción de precios de automóviles usados, hemos alcanzado varias conclusiones significativas. A pesar de que inicialmente observamos que el modelo Random Forest obtuvo resultados ligeramente mejores en términos de precisión de predicción, después de una evaluación, decidimos optar por el modelo XGBoost por varias razones importantes. En primer lugar, XGBoost demostró ser más eficiente computacionalmente, lo que nos permitió reducir el tiempo de entrenamiento del modelo y mejorar la escalabilidad de nuestro sistema. Además, su flexibilidad en la optimización de hiperparámetros y su capacidad para proporcionar un rendimiento ligeramente superior en una variedad de conjuntos de datos y problemas fueron factores clave en nuestra decisión.

Si bien no pudimos proporcionar una interfaz gráfica fácil de usar para acceder a las predicciones, implementamos una API que permite a los usuarios enviar solicitudes y obtener predicciones de manera programática. Al implementar el modelo XGBoost en una instancia de EC2 de AWS y exponer una API accesible en la URL proporcionada, hemos habilitado el acceso a la capacidad predictiva del modelo. Aunque la interacción con la API requiere un conocimiento básico de cómo realizar solicitudes HTTP, esta implementación aún facilita la integración del modelo en aplicaciones y sistemas existentes.

Este enfoque pragmático y accesible no solo mejora la eficiencia en la toma de decisiones relacionadas con los precios de los automóviles usados, sino que también demuestra la aplicabilidad práctica de los modelos de machine learning. Estamos entusiasmados con el impacto positivo que este proyecto puede tener en la industria automotriz y esperamos continuar explorando y aplicando técnicas avanzadas de aprendizaje automático para abordar desafíos futuros. A lo largo del proceso, hemos aprendido la importancia de la selección cuidadosa de algoritmos, la optimización de hiperparámetros y la implementación efectiva de soluciones técnicas. Aunque enfrentamos desafíos y limitaciones, en el desarrollo de la API con nuevas tecnologías, hemos logrado proporcionar una solución práctica y accesible a través de una API que facilita la integración del modelo en aplicaciones reales.