**Pablo González**
**William Morales**
**Juan Sebastián Perdomo**
**Machine Learning y Procesamiento del Lenguaje Natural**
**Url Libro de trabajo**
**TEAM WPS en Kaggle, Puesto 18**

---

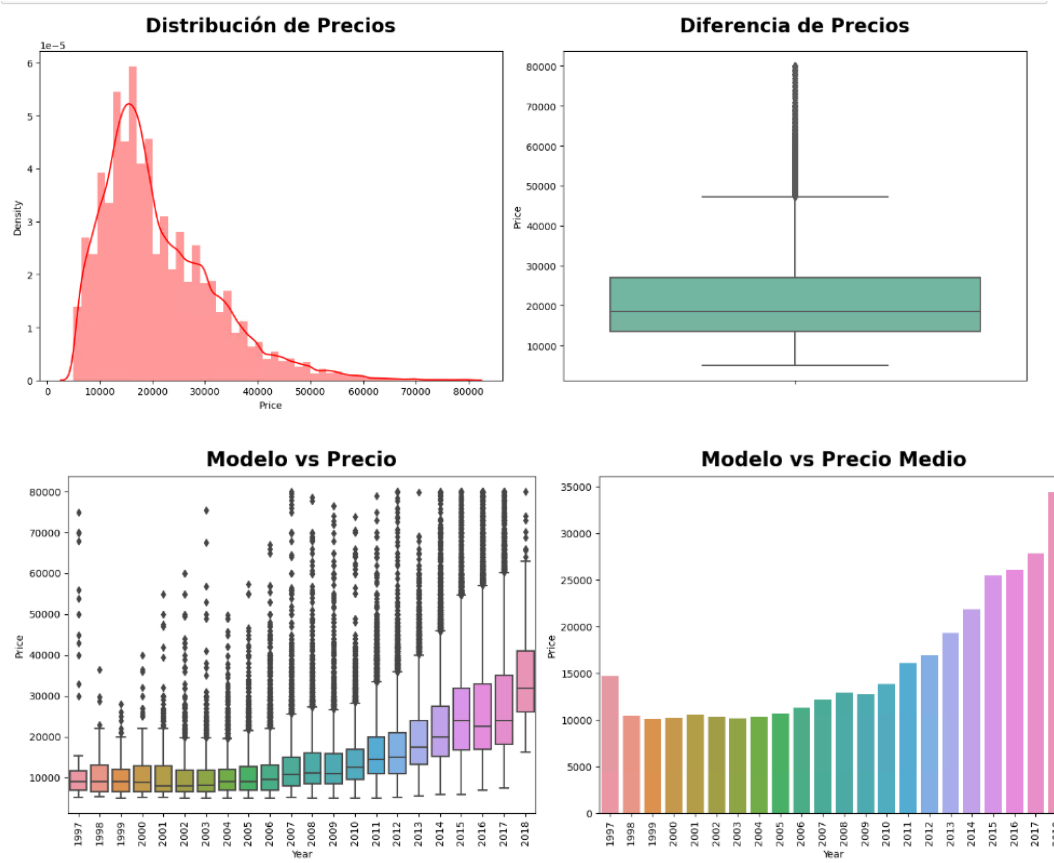
## Informe Predicción de Precios de Vehículos Usados

### 1. Procesamiento de datos

Una vez se carga la base de datos con la que se va a trabajar, se realizó un proceso exploratorio. En este proceso identificamos la estructura de la base que en este caso tiene 400.000 observaciones y 6 variables. Cabe mencionar que cada observación es un vehículo particular. Revisando las estadísticas descriptivas se identificó que el modelo más antiguo de un vehículo es del año 1997 y el modelo más reciente es del año 2018. Adicionalmente, en cuanto a los precios se puede decir que el vehículo más barato es de 5.001 USD, mientras que el más costoso es de 79.999 USD.

Por otra parte, se revisaron si dentro de las variables había registros nulos o registros duplicados para realizar el debido proceso de limpieza en caso de encontrar. En este caso, la base de datos no contenía datos nulos, pero si tenía 169 registros duplicados que fueron eliminados, por lo que al final la base de datos quedó estructurada por 399.831 observaciones y 6 variables.

```
=========================================================
Validación de datos faltantes en el set de datos
         Total No. Faltantes
Price                      0
Year                       0
Mileage                    0
State                      0
Make                       0
Model                      0
=========================================================
Validación de duplicados en el set de datos
Registros duplicados: 169
```



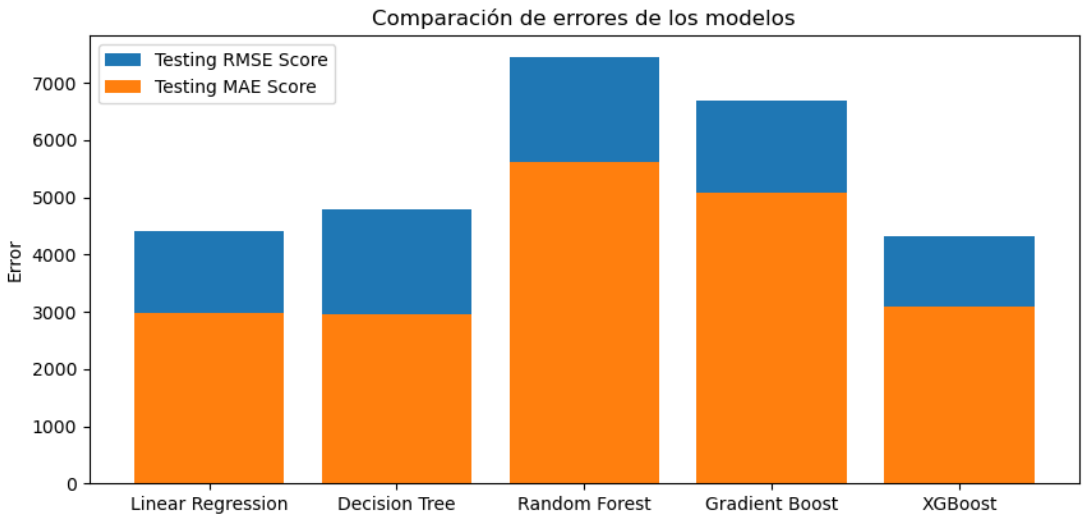

```python
# Separamos la variable objetivo (Price) y las características
X = df.drop('Price', axis=1)
y = df['Price']

# Dividimos el conjunto de datos en entrenamiento y prueba (80% entrenamiento, 20% prueba)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Definimos qué columnas son numéricas y cuáles son categóricas
numeric_features = ['Year', 'Mileage']
categorical_features = ['State', 'Make', 'Model']

# Creamos transformadores para escalar variables numéricas y codificar variables categóricas
numeric_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

# Aplicamos las transformaciones a las columnas correspondientes utilizando ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# Ajustamos y transformamos el conjunto de entrenamiento
X_train_transformed = preprocessor.fit_transform(X_train)

# Transformamos el conjunto de prueba (no ajustamos para evitar filtración de datos)
X_test_transformed = preprocessor.transform(X_test)
```

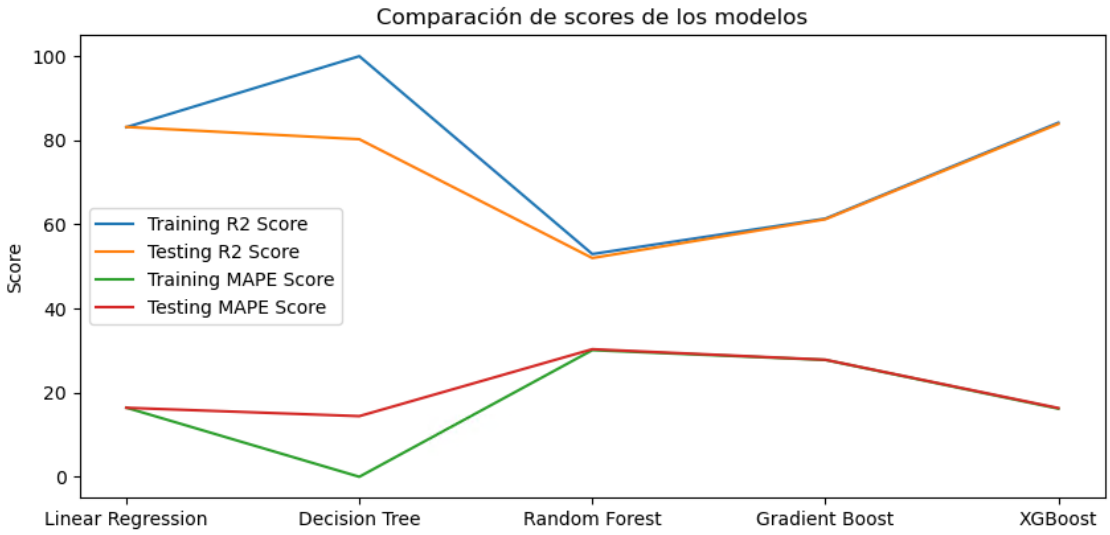
En las líneas de código de la imagen superior, se realizó el respectivo procesamiento de los datos. Lo primero que se hizo fue separar la variable *Price*, que en este caso va a ser la variable dependiente del modelo, del conjunto de

variables que conforman la base de datos. Posteriormente, se dividió el conjunto total de datos en los conjuntos de entrenamiento y prueba. Se decidió utilizar el 30% de los datos en el conjunto de prueba y adicionalmente se estableció un estado aleatorio igual a 42. También, se realizaron transformaciones con el objetivo de escalar las variables numéricas y codificar las variables categóricas para luego aplicar las respectivas transformaciones.

## 2. Calibración del modelo

Se decidieron correr 5 algoritmos distintos y las métricas que se utilizaron para determinar el modelo con el mejor desempeño posible fue el RMSE, el MAE y el MAPE. El primero fue una regresión lineal donde para el conjunto de entrenamiento se obtuvo un RMSE de 4418.6, un MAE de 2972.0 y un MAPE de 16.4. Para el conjunto de prueba se obtuvo un RMSE de 4413.3, un MAE de 2972.5 y un MAPE de 16.4. El segundo fue un árbol de decisión donde para el conjunto de entrenamiento se obtuvo un RMSE de 119.9, un MAE de 2.0 y un MAPE de 0.0. Para el conjunto de prueba se obtuvo un RMSE de 4780.4, un MAE de 2949.2 y un MAPE de 14.4. El tercero fue un bosque aleatorio, en el que se estableció un máximo de profundidad igual a 10 y 50 como el número de observaciones, donde para el conjunto de entrenamiento se obtuvo un RMSE de 7373.0, un MAE de 5565.2 y un MAPE de 30.1. Para el conjunto de prueba se obtuvo un RMSE de 7451.6, un MAE de 5624.2 y un MAPE de 30.3. El cuarto fue un Gradient Boosting donde para el conjunto de entrenamiento se obtuvo un RMSE de 6681.2, un MAE de 5072.2 y un MAPE de 27.8. Para el conjunto de prueba se obtuvo un RMSE de 6698.0, un MAE de 5088.5 y un MAPE de 27.8. Finalmente, el quinto fue un XGBoost donde para el conjunto de entrenamiento se obtuvo un RMSE de 4276.4, un MAE de 3049.0 y un MAPE de 16.2. Para el conjunto de prueba se obtuvo un RMSE de 4317.2, un MAE de 3080.4 y un MAPE de 16.3.




Con base en los resultados de ejecución de cada uno de los modelos, hemos decidido utilizar el algoritmo de **GradientBoostingRegresor** para utilizarlo como algoritmo de regresión para las predicciones.

Para la calibración de los hiperparámetros del modelo se utilizó la función GridSearchCV, la cual realiza el entrenamiento del modelo con todas las posibles combinaciones establecidas y retorna la mejor combinación a través de validación cruzada, el único contra de la función utilizada es el tiempo de ejecución ya que realiza el entrenamiento de todas las posibles combinaciones.

```python
# Se definen los hiperparámetros a utilizar dentro del ajuste
param_grid = {
    'learning_rate': [0.1],
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7, 10, 20, 30]
}

# Crear un modelo Gradient Boosting Regressor
gb = GradientBoostingRegressor()

# Realizar una búsqueda de cuadrícula para encontrar los mejores hiperparámetros
grid_search = GridSearchCV(estimator=gb, param_grid=param_grid, cv=5, verbose=2)
grid_search.fit(X_train_transformed, y_train)

# Imprimir los mejores hiperparámetros y la puntuación de validación cruzada
print("Mejores hiperparámetros: ", grid_search.best_params_)
print("Puntuación de validación cruzada: ", grid_search.best_score_)
```

Como resultado del proceso se obtienen los siguientes valores para los parámetros del modelo:

```
[CV] END ..learning_rate=0.1, max_depth=20, n_estimators=200; total time=21.3min
[CV] END ..learning_rate=0.1, max_depth=20, n_estimators=200; total time=21.4min
[CV] END ..learning_rate=0.1, max_depth=20, n_estimators=200; total time=21.2min
[CV] END ...learning_rate=0.1, max_depth=30, n_estimators=50; total time=17.6min
[CV] END ...learning_rate=0.1, max_depth=30, n_estimators=50; total time=17.5min
[CV] END ...learning_rate=0.1, max_depth=30, n_estimators=50; total time=17.5min
[CV] END ...learning_rate=0.1, max_depth=30, n_estimators=50; total time=17.5min
[CV] END ...learning_rate=0.1, max_depth=30, n_estimators=50; total time=17.7min
[CV] END ..learning_rate=0.1, max_depth=30, n_estimators=100; total time=24.8min
[CV] END ..learning_rate=0.1, max_depth=30, n_estimators=100; total time=24.7min
[CV] END ..learning_rate=0.1, max_depth=30, n_estimators=100; total time=25.0min
[CV] END ..learning_rate=0.1, max_depth=30, n_estimators=100; total time=25.2min
[CV] END ..learning_rate=0.1, max_depth=30, n_estimators=100; total time=25.0min
[CV] END ..learning_rate=0.1, max_depth=30, n_estimators=200; total time=41.4min
[CV] END ..learning_rate=0.1, max_depth=30, n_estimators=200; total time=40.7min
[CV] END ..learning_rate=0.1, max_depth=30, n_estimators=200; total time=42.4min
[CV] END ..learning_rate=0.1, max_depth=30, n_estimators=200; total time=42.3min
[CV] END ..learning_rate=0.1, max_depth=30, n_estimators=200; total time=42.9min
Mejores hiperparámetros:  {'learning_rate': 0.1, 'max_depth': 20, 'n_estimators': 200}
Puntuación de validación cruzada:  0.8790005520776083
```

### 3. Entrenamiento del modelo

Dentro del punto 1 del presente informe, se muestran las transformaciones realizadas a las variables categóricas y numéricas para la predicción, por otro lado se realizo la medición de las siguientes métricas dentro del modelo entrenado

tomando como base los hiperparámetros resultado de la etapa de calibración R2, RMSE, MAE y MAPE

**Predicción con los mejores hiperparámetros**

```
In [219]: evaluacion_y_prediccion(GradientBoostingRegressor(learning_rate=0.1, max_depth=20, n_estimators=200))

r2_Score de GradientBoostingRegressor(max_depth=20, n_estimators=200) en datos de Trainig es: 94.38423010104172
r2_Score de GradientBoostingRegressor(max_depth=20, n_estimators=200) en datos de Testing es: 88.1595651862012
rmse_Score de GradientBoostingRegressor(max_depth=20, n_estimators=200) en datos de Trainig es: 2547.522384423502
rmse_Score de GradientBoostingRegressor(max_depth=20, n_estimators=200) en datos de Testing es: 3701.1921831614177
mae_Score de GradientBoostingRegressor(max_depth=20, n_estimators=200) en datos de Trainig es: 1696.679257778223
mae_Score de GradientBoostingRegressor(max_depth=20, n_estimators=200) en datos de Testing es: 2383.8540272806235
mape_Score de GradientBoostingRegressor(max_depth=20, n_estimators=200) en datos de Trainig es: 8.733959002887982
mape_Score de GradientBoostingRegressor(max_depth=20, n_estimators=200) en datos de Testing es: 11.976473258117025
```

Se selecciono el algoritmo de GradientBoostingRegressor ya que es capaz de ajustarse a patrones no lineales y puede manejar grandes conjuntos de datos, teniendo en cuenta su capacidad para reducir el sesgo y la varianza de los modelos de árboles de decisión individuales, lo que resulta en un mejor rendimiento general del modelo.

Los resultados de las diferentes métricas evaluadas indican que el modelo tiene un buen desempeño en los datos de entrenamiento y también en los datos de prueba, aunque el error en los datos de prueba es mayor que en los datos de entrenamiento. El coeficiente de determinación R2 indica que el modelo explica el 94.38% de la variabilidad en los datos de entrenamiento y el 88.16% en los datos de prueba, lo que sugiere que el modelo tiene una buena capacidad predictiva.

El error cuadrático medio (RMSE) indica que el modelo tiene un error promedio de 2547.52 en los datos de entrenamiento y 3701.19 en los datos de prueba.

El error absoluto medio (MAE) indica que el modelo tiene un error promedio de 1696.68 en los datos de entrenamiento y 2383.85 en los datos de prueba.

El error porcentual absoluto medio (MAPE) indica que el modelo tiene un error promedio de 8.73% en los datos de entrenamiento y 11.98% en los datos de prueba.

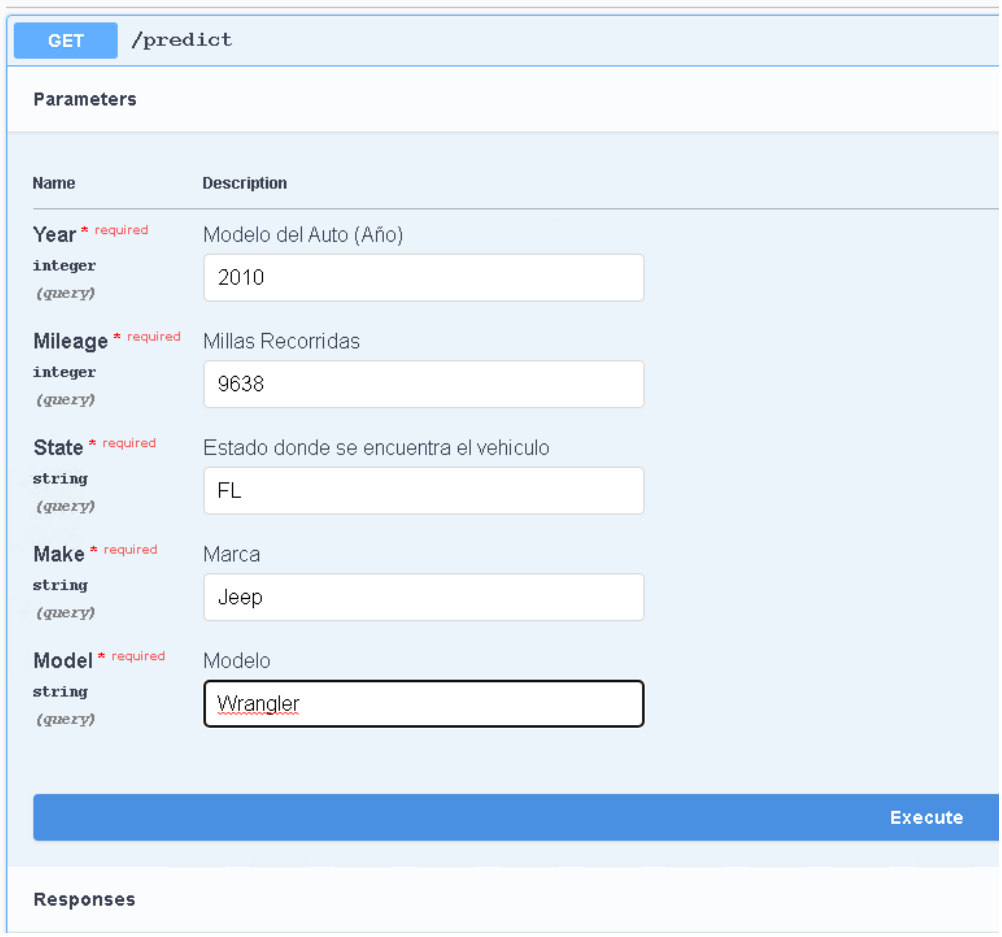
En general, los resultados sugieren que el modelo tiene un buen potencial para predecir los precios de los autos en función de sus características.

## 4. Disponibilización del modelo

Para la disponiblizacion del modelo se creó un archivo API.py el cual hace uso del modelo entrenado para la predicción, esta información fue desplegada en un servidor de AWS es cual es accesible desde la siguiente URL

http://3.144.136.253:5000/

Se realizaron las siguientes predicciones validando que el modelo es funcional.




## 5. Conclusiones

Es fundamental comprender qué función cumple cada hiperparámetro dentro del modelo seleccionado, ya que, dependiendo del algoritmo utilizado, existen parámetros que, aunque no suelen utilizarse con frecuencia, pueden mejorar significativamente el rendimiento del modelo.

Después de elegir el modelo, el paso que más tiempo nos llevó fue seleccionar los mejores hiperparámetros. Esto se debió a que empleamos un proceso que realiza combinaciones y, generalmente, toma horas analizar los distintos parámetros que generan el mejor resultado del modelo.

Poner a disposición el modelo en la API es un proceso relativamente sencillo. Se debe tener muy claro qué parámetros recibe el modelo y, además, algo que nos llevó un poco de tiempo fue importar las transformaciones que se realizaron en el entrenamiento a los datos enviados desde la URL a través del método GET.

De manera general el proyecto fue bastante provechoso, pues tuvimos que indagar más sobre los algoritmos utilizados para así poder generar el mejor resultado.