Universidad de los Andes
MIAD – Machine Learning y Procesamiento de Lenguaje Natural
30/04/2023

Integrantes

Daniel Rozo Isaza
Gabriel Gómez
Yolanda Franco

Parte 1. Análisis descriptivo.

Las bases de datos a utilizar en este ejercicio ya habían sido segmentadas previamente en datos de entrenamiento y de prueba. La base de entrenamiento cuenta con un total de 400.000 observaciones, mientras que los datos de prueba constan de 100.000 observaciones. Adicionalmente, se cuentan con 6 columnas que corresponden al precio del vehículo, el año de fabricación, el kilometraje, el estado donde se vende el vehículo, el fabricante y el modelo. Las primeras tres variables son numéricas, dos de tipo continuo (precio y kilometraje) y una de tipo discreto (año de fabricación). Las tres variables restantes son de tipo categórico. La variable a predecir es el precio. Con los descriptivos realizados se tiene que el precio promedio es de 21,146.92 dólares, con una desviación estándar de 10,753.66 dólares, lo que indica una gran variabilidad en los precios. El precio mínimo registrado es de 5,001 dólares, mientras que el precio máximo es de 79,999 dólares. El 25% de los carros tienen un precio inferior a 13,499 dólares, el 50% tienen un precio inferior a 18,450 dólares y el 75% tienen un precio inferior a 26,999 dólares. Estos descriptivos indican que la distribución de precios es sesgada hacia la derecha, ya que la mediana (18,450 dólares) es menor que la media (21,146.92 dólares). El kilometraje promedio de los coches es de 55,072.96 millas, con una desviación estándar de 40,881.02 millas, lo que indica una gran variabilidad en los datos. El kilometraje mínimo registrado es de 5 millas, lo que podría ser atribuible a coches nuevos o a errores de entrada de datos. La mediana, que es el valor que separa la mitad inferior y la mitad superior de los datos, es de 42,955 millas, lo que sugiere que la mayoría de los coches han recorrido un número moderado de millas. El 25% de los coches tienen un kilometraje de menos de 25,841 millas, lo que indica que un cuarto de los coches son relativamente nuevos. El 75% de los coches tienen un kilometraje de menos de 77,433 millas, lo que indica que la mayoría de los coches han sido usados. El kilometraje máximo registrado es de 2,457,832 millas, lo que es una cifra bastante inusual y probablemente sea un error de entrada de datos o una medida exagerada. La siguiente tabla resume lo mencionado.

Tabla 1. Estadísticas descriptivas de las variables numéricas.

	Price	Year	Mileage
count	400000.000000	400000.000000	4.000000e+05
mean	21146.919312	2013.198125	5.507296e+04
std	10753.664940	3.292326	4.088102e+04
min	5001.000000	1997.000000	5.000000e+00
25%	13499.000000	2012.000000	2.584100e+04
50%	18450.000000	2014.000000	4.295500e+04
75%	26999.000000	2016.000000	7.743300e+04
max	79999.000000	2018.000000	2.457832e+06

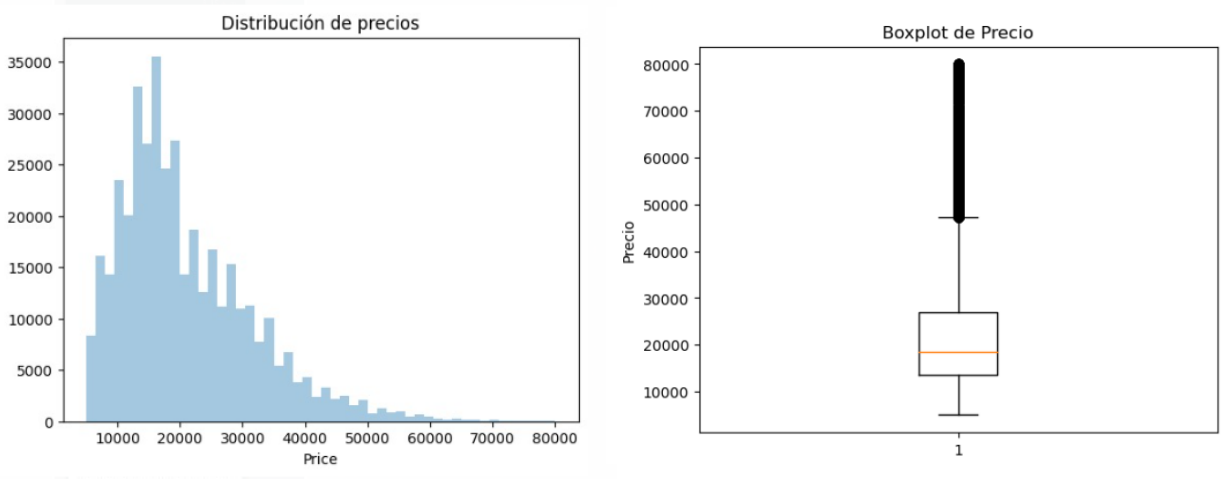
En general, no se encontraron valores nulos en ninguna de las columnas. En cambio, se cuenta con un alto número de valores únicos, incluso para las variables de tipo categóricas. La siguiente tabla muestra que, se tiene información de 22 años, 51 estados, 38 fabricantes y 525 modelos. El alto número de categorías que se presenta en una variable como modelos, eventualmente puede dificultad su inclusión en un modelo predictivo ya que, al transformar cada categoría en una variable dicotómica, el número de variables sube de manera exagerada. Por esta razón, más adelante se observará qué tipo de tratamiento recibirá esta variable.

Tabla 2. Número de valores únicos por columna para las bases de prueba y entrenamiento.

```
Valores únicos por columna

Price       35867
Year           22
Mileage    130600
State          51
Make           38
Model         525
dtype: int64

Year           22
Mileage     66697
State          51
Make           37
Model         525
dtype: int64
```

Tanto el histograma como el boxplot corroboran la asimetría de la variable precio hacia la derecha. Además, el boxplot muestra la presencia de datos atípicos altos. Para verificar si estos datos altos tienen sentido, se ubicó el fabricante y la marca del precio máximo encontrado. En esta base de datos el precio máximo es de 79,999 dólares y corresponde a un Land Rover del 2018. La búsqueda indica: "The Manufacturer's Suggested Retail Price (MSRP) for a base-model 2018 Land Rover Range Rover starts at about 88,500, which isn't bad, while long-wheelbase versions start at about 110,000", siendo el de la base de datos más bajo que el valor encontrado en la búsqueda, pero esto puede deberse a que no es un auto de primera mano, sino que cuenta con 649 millas.

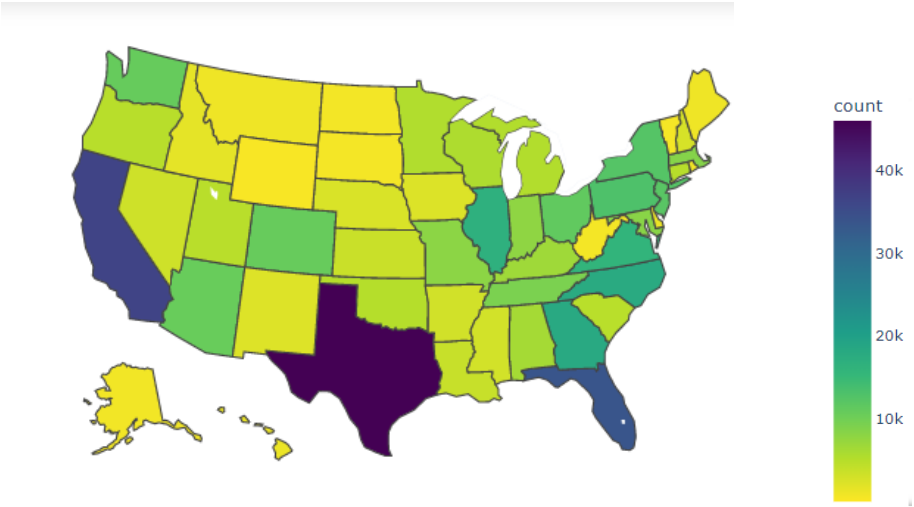
Gráfico 1. Distribución de la variable Precio.



Sobre los estados en los cuales se listan los automóviles, se construyó un mapa para comprender su dinámica, además de una tabla con los primeros 10 estados con el mayor número de observaciones. Se encuentra que los vehículos de la muestra tienden a concentrarse en el sur, sobre todo en los estados de Texas, California y Florida.

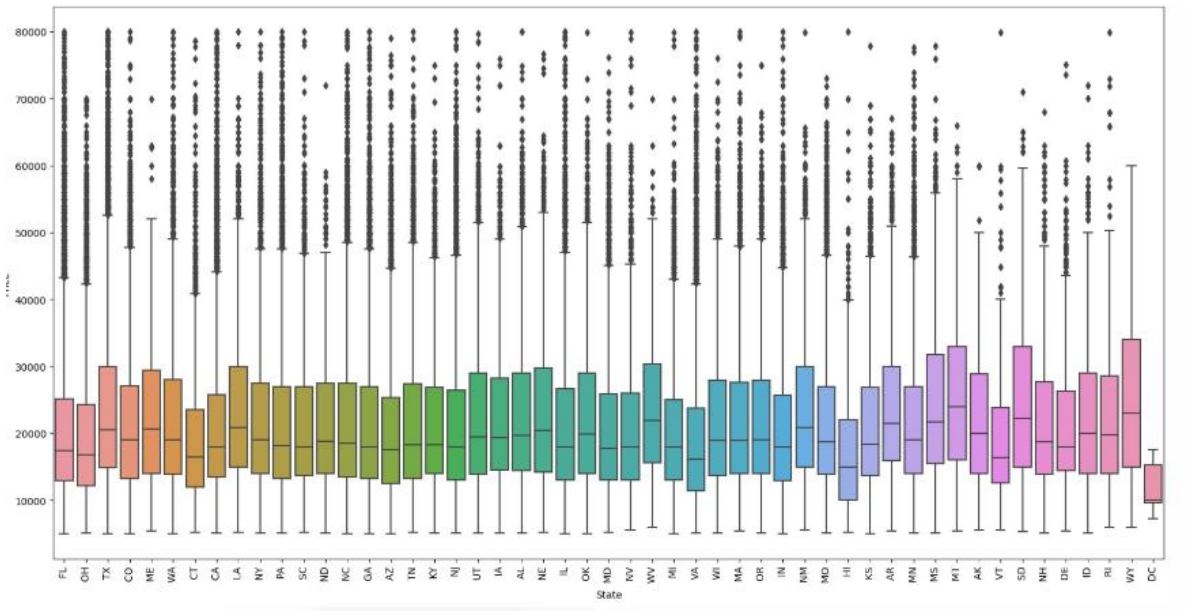
Gráfico 2. Distribución de la muestra por estados.

	index	state_full
0	Texas	45918
1	California	36534
2	Florida	33759
3	Georgia	18182
4	North Carolina	17930
5	Illinois	16793
6	Virginia	15894
7	Pennsylvania	13039
8	New York	12447
9	New Jersey	12132
10	Ohio	11364



La distribución de los precios por estado muestra que no hay diferencias muy marcadas de precio entre un estado y otro. En realidad, la única excepción parece ser Washington DC, donde los precios se tienden a concentrar en valores inferiores a los que se encuentran en los demás estados.

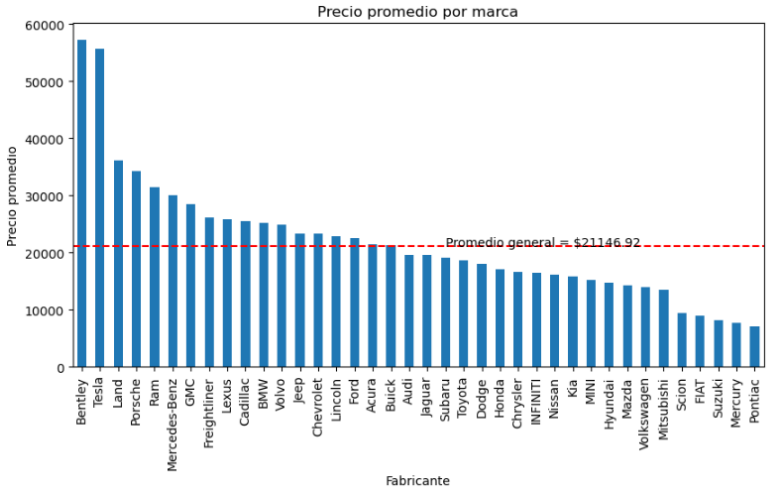
Gráfico 3. Precios de los vehículos por estado.



El fabricante que más aparece en la muestra es Ford con el 15.6% de las observaciones, seguido por Chevrolet con el 14.6% y Toyota con el 11.5%. Por otra parte, los fabricantes con los automóviles de mayor precio promedio son Bentley (57256) y Tesla (55646), mientras que el precio menor se encuentra para Pontiac (7167.8). Estos resultados se presentan en el gráfico 4.

Gráfico 4. Número y proporción de automóviles por fabricante y precio promedio.

Año	Valor Absoluto	Proporción
Ford	62899	0.157248
Chevrolet	58383	0.145957
Toyota	45941	0.114852
Honda	33191	0.082977
Jeep	24369	0.060922
GMC	20834	0.052085
Kia	16352	0.040880
Dodge	16159	0.040398
Hyundai	15057	0.037643
Lexus	13664	0.034160
BMW	12326	0.030815
Volkswagen	11110	0.027775



Para la parte de los modelos, debido al alto número de categorías resulta difícil realizar una descripción más detallada. En general, el modelo más habitual es el Silverado que corresponde al 4% de los datos de entrenamiento y el Grand que corresponde al 3%.

Parte 2. Preprocesamiento.

En el preprocesamiento no se presentó la necesidad de realizar imputación de datos faltantes ni valores atípicos. Dado que las bases de datos ya habían sido separadas en entrenamiento y prueba con anterioridad, tampoco existió la necesidad de incorporar este paso. En general, se hicieron algunas transformaciones. Como se mencionó anteriormente, el número de categorías en algunas variables es alto. Si bien, los modelos a utilizar podrían, en teoría trabajar con este número elevado de variables, ya que el tamaño de las observaciones también lo permite, fue necesario tener en cuenta el tiempo de ejecución de los modelos y, por lo tanto, buscar una estrategia que permitiera disminuir estas categorías. Para el caso de los estados, se crearon variables dicotómicas para los primeros 10 estados con mayor número de observaciones. El resto de los estados se agruparon dentro de la categoría “Otro”. Para la variable de fabricante y modelo se generaron variables dummies para tener mayor poder de predicción dado que esto como lo vimos en el análisis descriptivo discrimina el precio de forma determinante.

Imagen 1. Transformación de las variables.

```
# Pre procesar los datos
top10_estados = list(dataTraining['State'].value_counts().head(10).index)

#Agrupar los estados que no estan en el top 10 en un solo estado
dataTraining['State'] = dataTraining['State'].apply(lambda x: x if x in top10_estados else 'Otro')
dataTesting['State'] = dataTesting['State'].apply(lambda x: x if x in top10_estados else 'Otro')

# Convertir variables categóricas
dataTraining['State'] = dataTraining['State'].astype('category')
dataTraining['Make'] = dataTraining['Make'].astype('category')
dataTraining['Model'] = dataTraining['Model'].astype('category')

dataTesting['State'] = dataTesting['State'].astype('category')
dataTesting['Make'] = dataTesting['Make'].astype('category')
dataTesting['Model'] = dataTesting['Model'].astype('category')

# Crear variables dummies
dataTraining = pd.get_dummies(dataTraining, columns=['State', 'Make', 'Model'], drop_first=True)
dataTesting = pd.get_dummies(dataTesting, columns=['State', 'Make', 'Model'])

# Separar la variable objetivo
X_total = dataTraining.drop(['Price', 'Make_Freightliner'], axis=1)
y_total = dataTraining['Price']

# Separar los datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_total, y_total, test_size=0.2, random_state=123)
```

El anterior procesamiento concluyó en un set de 517 variables y 400.000 Observaciones para poder calibrar y posteriormente entrena el modelo.

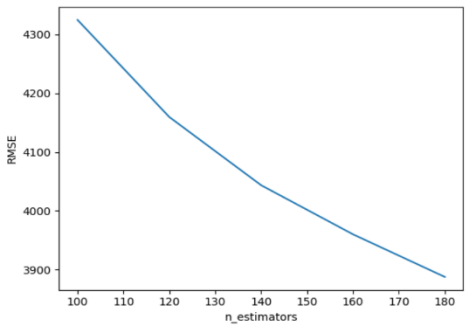
Parte 3. Calibración del modelo

Dado su versatilidad y poder de predicción decidimos calibrar un modelo de XGBoost para poder predecir el precio. Asi mismo, para poder ahorrar tiempo en computación los parámetros de calibración (n_estimators, Learning_rate, max_depth y min_child) se evaluaron de forma separada para así poder llegar a una solución buena, más no optima ya que el tiempo no lo permitía, pero teniendo una lógica clara para mejorar las predicciones.

Imagen 1. Calibración n_estimators

```python
# Calibración de un modelo de regresión XGBoost
# Definición de parámetros
n_estimators_range = list(range(100, 200, 20))
learning_rate_range = np.linspace(0.01, 0.5, 20, endpoint=True)
max_depth_range = list(range(8, 20, 2))
min_child_weight_range = list(range(28, 37, 2))

# Definición de variables para guardar los resultados
rmse_list = []

# Ciclo para evaluar n_estimators
for n_estimators in n_estimators_range:
    print(f'Probando con n_estimators={n_estimators}')
    # Crear el modelo
    model = XGBRegressor(n_estimators=n_estimators, random_state=123, n_jobs=-1)
    # Entrenar el modelo
    model.fit(X_train, y_train)
    # Predecir
    y_pred = model.predict(X_test)
    # Calcular métricas
    rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
    print(f'RMSE={rmse}')
    # Guardar resultados
    rmse_list.append(rmse)
```

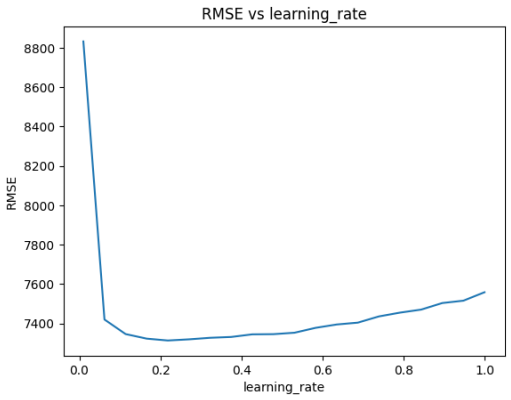
```python
# Mejor n_estimators
n_estimators_best = n_estimators_range[np.argmin(rmse_list)]
print(f'El mejor valor de n_estimators es {n_estimators_best}')

# Mejor error
rmse_best = np.min(rmse_list)
print(f'El mejor valor de RMSE es {rmse_best}')

# Gráfica de RMSE vs n_estimators
plt.plot(n_estimators_range, rmse_list)
plt.xlabel('n_estimators')
plt.ylabel('RMSE')
plt.show()
```



En este acaso tomamos un poco más de los 180, llegando a los 220 para seguir reduciendo el RMSE del modelo como muestra la tendencia y asi poderlo tener como base para el entrenamiento.

Imagen 2. Calibración Learning Rate

```python
# Mejor learning_rate
learning_rate_best = learning_rate_range[np.argmin(rmse_list)]
print(f'El mejor valor de learning_rate es {learning_rate_best}')
# Mejor error
rmse_best = np.min(rmse_list)
print(f'El mejor valor de RMSE es {rmse_best}')

# Graficar RMSE vs learning_rate
plt.plot(learning_rate_range, rmse_list)
plt.xlabel('learning_rate')
plt.ylabel('RMSE')
plt.title('RMSE vs learning_rate')
plt.show()
```



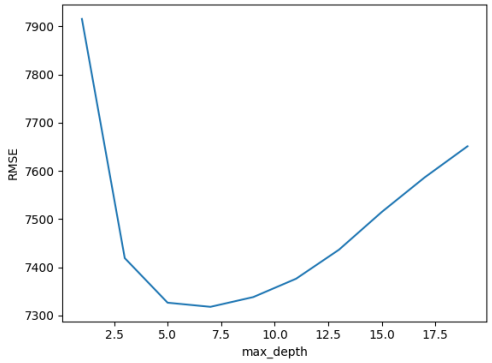
Con el learning rate el valor que mejor desempeño tuvo fue de 0.2184210 ya que después de este el RMSE tendía a subir nuevamente.

Ahora bien, para poder calibrar la maxima profundidad tomamos el valor del n_estimator de 220 y el resultado del mejor valor para este parámetro fue 7.

Imagen 3. Max-Depth

```python
# Calibración de max_depth
n_estimators_best = 220
# Definición de variables para guardar los resultados
rmse_list = []

# Ciclo para evaluar max_depth
for max_depth in max_depth_range:
    print(f'Probando con max_depth={max_depth}')
    # Crear el modelo
    model = XGBRegressor(n_estimators=n_estimators_best, learning_rate=0.1, max_depth=max_depth, random_state=123,
    # Entrenar el modelo
    model.fit(X_train, y_train)
    # Predecir
    y_pred = model.predict(X_test)
    # Calcular métricas
    rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
    print(f'RMSE={rmse}')
    # Guardar resultados
    rmse_list.append(rmse)
```

```python
# Mejor max_depth
max_depth_best = max_depth_range[np.argmin(rmse_list)]
print(f'El mejor valor de max_depth es {max_depth_best}')
# Mejor error
rmse_best = np.min(rmse_list)
print(f'El mejor valor de RMSE es {rmse_best}')

# Graficar RMSE vs max_depth
plt.plot(max_depth_range, rmse_list)
plt.xlabel('max_depth')
plt.ylabel('RMSE')
plt.show()
```

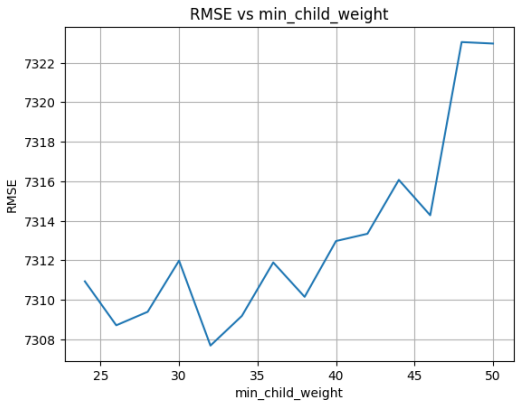


Por último, calibramos el min_child_Weight_range con todos los parámetros anteriores y encontramos que el mejor número que nos dio fue el 32.

Imagen 4. Max-Depth

```python
# Calibrar min_child_weight
min_child_weight_range = list(range(24, 52, 2))
# Definición de variables para guardar los resultados
rmse_list = []

# Ciclo para evaluar min_child_weight
for min_child_weight in min_child_weight_range:
    print(f'Probando con min_child_weight={min_child_weight}')
    # Crear el modelo
    model = XGBRegressor(n_estimators=n_estimators_best, learning_rate=learning_rate_best, max_depth=max_depth_best,
    # Entrenar el modelo
    model.fit(X_train, y_train)
    # Predecir
    y_pred = model.predict(X_test)
    # Calcular métricas
    rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
    print(f'RMSE={rmse}')
    # Guardar resultados
    rmse_list.append(rmse)
```

```python
# Mejor min_child_weight
min_child_weight_best = min_child_weight_range[np.argmin(rmse_list)]
print(f'El mejor valor de min_child_weight es {min_child_weight_best}')
# Mejor error
rmse_best = np.min(rmse_list)
print(f'El mejor valor de RMSE es {rmse_best}')

# Graficar RMSE vs min_child_weight
plt.plot(min_child_weight_range, rmse_list)
plt.xlabel('min_child_weight')
plt.ylabel('RMSE')
plt.title('RMSE vs min_child_weight')
plt.grid()
plt.show()
```



Parte 4. Entrenamiento del modelo

Ya habiendo determinado los mejores parámetros calibrados continuamos con el entrenamiento del modelo de XGBoost acotando la búsqueda de los parámetros a los que vimos en el punto anterior, pero realizando escenarios en un ciclo for para evaluar los los diferentes escenarios que correremos.

Imagen 1. Código Entrenamiento con búsqueda del mejor escenario de error

```python
# Acortar rango de búsqueda en parámetros
n_estimators_range = list(range(120, 201, 20))
learning_rate_range = [0.1, 0.2, 0.3]
max_depth_range = list(range(4, 15, 2))
min_child_weight_range = list(range(26, 37, 2))

# Calibrar todas las combinaciones de los modelos
# Definición de variables para guardar los resultados
rmse_list = []

# Ciclo para evaluar n_estimators
for n_estimators in n_estimators_range:
    for rate in learning_rate_range:
        for depth in max_depth_range:
            for weight in min_child_weight_range:
                print(f'Probando con n_estimators={n_estimators}, learning_rate={rate}, max_depth={depth}, min_child_weight={weight}')
                # Crear el modelo
                model = XGBRegressor(n_estimators=n_estimators, learning_rate=rate, max_depth=depth, min_child_weight=weight, random_state=1
                # Entrenar el modelo
                model.fit(X_train, y_train)
                # Predecir
                y_pred = model.predict(X_test)
                # Calcular métricas
                rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
                print(f'RMSE={rmse}')
                # Guardar resultados
                rmse_list.append(rmse)
```

Una vez, habido corrido los diferentes escenarios, guardamos los resultados en un dataframe y buscamos el escenario con el menor error y visibilizamos los resultados.

Imagen 2. Código Entrenamiento con búsqueda del mejor escenario de error

```python
# Crear dataframe con escenarios
results = pd.DataFrame()
index = 0

# Agregar escenarios
for estimator in n_estimators_range:
    for rate in learning_rate_range:
        for depth in max_depth_range:
            for weight in min_child_weight_range:
                results.loc[index, 'Estimators'] = estimator
                results.loc[index, 'Rate'] = rate
                results.loc[index, 'Depth'] = depth
                results.loc[index, 'Weight'] = weight
                results.loc[index, 'RMSE'] = rmse_list[index]
                index += 1

display(results.head())

# Obtener los menores RMSE
display(results[results['RMSE'] == results['RMSE'].min()])
```

	Estimators	Rate	Depth	Weight	RMSE
0	120.0	0.1	4.0	26.0	7547.707704
1	120.0	0.1	4.0	28.0	7549.221990
2	120.0	0.1	4.0	30.0	7547.962118
3	120.0	0.1	4.0	32.0	7547.782154
4	120.0	0.1	4.0	34.0	7545.572766

```python
best_rmse = results[results['RMSE'] == results['RMSE'].min()]

n_estimators_best = best_rmse.iloc[0,0].astype(int)
learning_rate_best = best_rmse.iloc[0,1]
max_depth_best = best_rmse.iloc[0,2].astype(int)
min_child_weight_best = best_rmse.iloc[0,3].astype(int)

print(f'El mejor valor de n_estimators es {n_estimators_best}')
print(f'El mejor valor de learning_rate es {learning_rate_best}')
print(f'El mejor valor de max_depth es {max_depth_best}')
print(f'El mejor valor de min_child_weight es {min_child_weight_best}')
```

```
El mejor valor de n_estimators es 160
El mejor valor de learning_rate es 0.1
El mejor valor de max_depth es 12
El mejor valor de min_child_weight es 34
```

Una vez obtenido estos resultados, lo que hicimos fue correr el modelo de predicción XGBoost con estos parámetros y además tomamos toda la data_train con el total de observaciones para garantizar mayor robustes en los resultados del precio de los carros, los cuales los guardamos en un dataframe que fue el que subimos a Kaggle para competir.

```python
from xgboost import XGBRegressor
xgboost_reg = XGBRegressor(n_estimators=n_estimators_best, learning_rate=learning_rate_best, max_depth=max_depth_best, min_child_weight=min_child_w
xgboost_reg.fit(X_total, y_total)
```

Python

```
XGBRegressor
```

```python
# Predecir
y_pred = xgboost_reg.predict(dataTesting.drop(['Model', 'State_ CA','Make_Acura'], axis=1))

# Exportar resultados para Kaggle
y_pred = pd.DataFrame(y_pred, index=dataTesting.index, columns=['Price'])
y_pred.to_csv('submission_xgboost_2.csv', index_label='ID', header=['Price'])
y_pred.head()
```

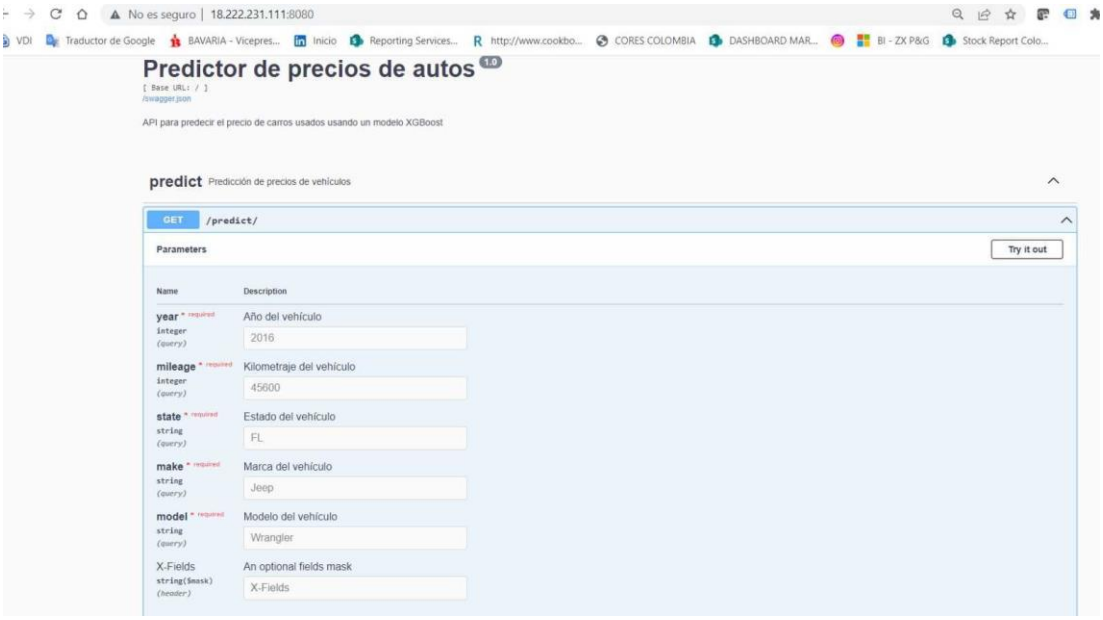
Python

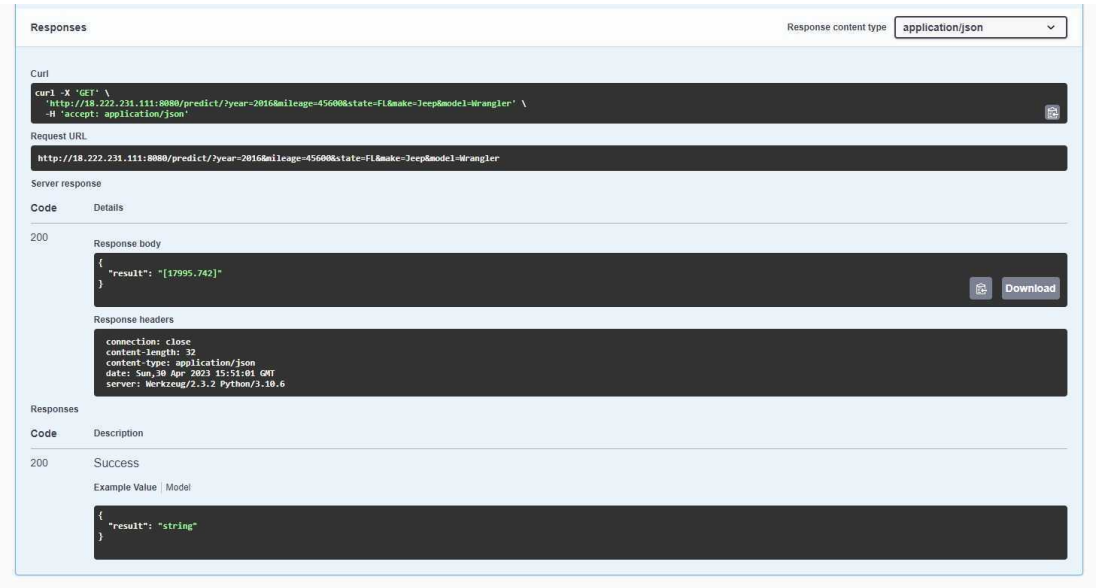
ID	Price
0	21442.773438
1	32192.921875
2	23124.072266
3	7977.369629

Parte 5. Disponibilización del modelo

Ya habiendo corrido el modelo, lo que hicimos fue guardar este algoritmo y configurar el API en AWS para recibir las variables correspondientes para poder estimar el precio el cual se puede ver ya publicado a continuación:

Imagen 1- Modelo publicado en AWS





Esta API quedó disponible en el siguiente link publico. http://18.222.231.111:8080 para que cualquier persona en conclusión pueda disponer de una estimación del precio de un carro solamente ingresando su año, kilometraje, Estado al que pertenece el vehículo, marca y modelo.