Universidad Simón Bolívar
IA 2 - CI-5438
Miguel Perez 15-11126
Gabriel Chaurio 17-10126

Proyecto 1: Descenso de Gradiente

PROBLEMA

El objetivo de este proyecto es realizar una implementación del algoritmo de descenso de gradiente con error cuadrático como función de pérdida, para regresión lineal multivariada. Se debe probar con un set de datos definido (CarDehko.csv) y evaluar el modelo que lo implemente para revisar y estudiar su funcionamiento para la predicción de los valores de “Precio”.

PROCESAMIENTO DE DATOS

El archivo CarDehko.csv, está conformado por una muestra de datos correspondientes a las especificaciones de distintos autos y el precio de los mismos.

Los datos que se pueden recolectar del archivo son:

- Make (fabricante)
- Model (modelo)
- Price (precio de venta)
- Year (año)
- Kilometer (kilometraje)
- Fuel Type (tipo de combustible)
- Transmission (tipo de transmisión)
- Location (ubicación)

- Color (color)
- Owner (nro. de dueño actual)
- Seller Type (tipo de vendedor)
- Engine (motor)
- Max Power (potencia máxima)
- Max Torque (par máximo del motor)
- Drivetrain (tren motriz)
- Length (largo)
- Width (ancho)
- Height (alto)
- Seating Capacity (cantidad de asientos)
- Fuel Tank Capacity (capacidad del tanque de combustible)

Lo primero que se hace para procesar dicha data, es importar el archivo como un dataframe de Pandas.

Posteriormente, se eliminan dos columnas las cuales son Max Power y Max Torque, ya que su manejo como variables categóricas es especialmente engorroso y no tienen una correlación alta con los otros valores de el dataframe.

El dataframe contiene tanto variables categóricas, como continuas. Algunas columnas tienen datos faltantes, y se manejaron de la siguiente forma:

- Se creó una copia del dataframe original, llamada car_data_fill_na, donde los valores faltantes para las columnas se trataron de la siguiente manera:
    - Para las variables continuas, se sustituyeron los valores nulos por la media total de todos los valores no nulos de la columna. Esto, para no perder la data correspondiente al resto de las columnas para las filas afectadas, ya que puede haber información relevante en dichas columnas que en caso de ser eliminadas, afectaron el desempeño de aprendizaje del modelo y los pesos calculados. Se utilizó la media, porque de esta manera ese valor para dichas filas tendría un impacto menor a la hora de calcular los pesos que si se colocaran valores

aleatorios los cuales pueden provenir de otras filas donde se encuentre un valor atípico.

- Para las variables categóricas, se utilizó la moda, el cual es el valor que más se repite. Esto con la misma finalidad que utilizar la media en las variables continuas, minimizar el impacto en el entrenamiento del modelo a la hora de evaluar el valor de dichas filas donde se sustituye el valor nulo por la moda.

```python
car_data_fill_na = car_data_fill_na.fillna(car_data.mean())
```

Python

```python
for column in ["Length","Width","Height","Seating Capacity","Fuel Tank Capacity"]:
    car_data_fill_na[column] = car_data_fill_na[column].round(0)

for column in ["Engine","Drivetrain"]:
    mode = str(car_data_fill_na[column].mode())
    car_data_fill_na[column] = car_data_fill_na[column].replace(np.NaN, mode)
```

Python

- Se creo otra copia del dataframe original, llamado car_data_notna, en donde en lugar de sustituir los valores, se eliminaban las filas que contienen valores faltantes, de esta manera, el universo de muestra queda reducido y no existe una variación en los pesos provenientes de la existencia de un valor repetido numerosas veces a raiz de la sustitución.

```python
car_data_notna = car_data_notna.dropna()
```

Una vez definidos estos dataframes, se taclean la normalización de valores y el tratamiento de los outliers. Para esto se crean dos copias de los arrays anteriores, uno para la data que tendrá los valores manejados en el segmento anterior. A estos se le eliminaran los outliers mientras que la data de los cuales se copian, los mantendrán, esto para estudiar el comportamiento del algoritmo ante distintos casos.

Los outliers son valores que se desvían significativamente del resto de los datos en un conjunto de datos. Pueden ser causados por errores de medición, valores

atípicos o datos de una población diferente. En el caso del experimento, con la funcion value_counts() para el dataframe original, se puede ver que existen solo 1 ocurrencia para autos de las marcas Lamborghini, Ferrari y Maserati, los cuales afectan enormemente la escala de precios ya que estos son outliers que superan enormemente la media de la población.

Para lidiar con estos, se utiliza el siguiente código:

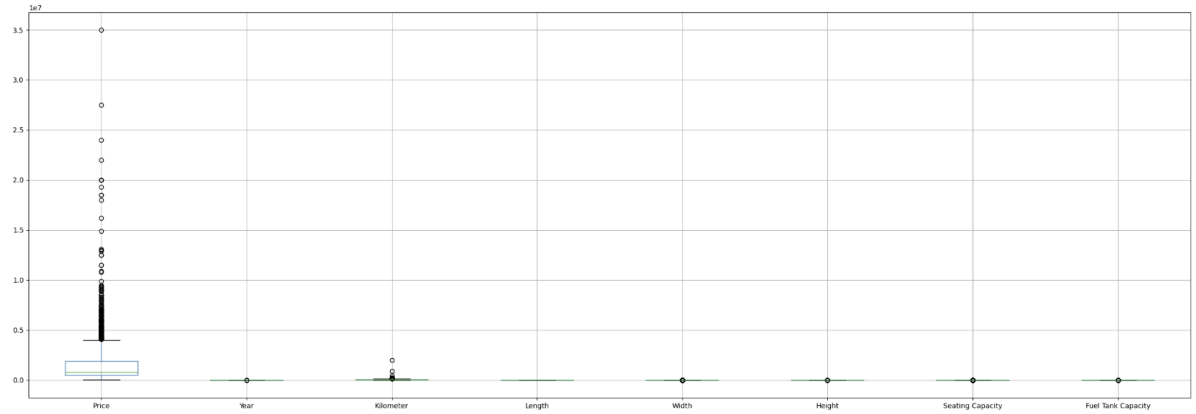
```
iqr = np.percentile(car_data_fill_na_no_outliers['Price'], 75) -
np.percentile(car_data_fill_na_no_outliers['Price'], 25)
upper_outlier_bound =
np.percentile(car_data_fill_na_no_outliers['Price'], 75) + 1.5 * iqr
lower_outlier_bound =
np.percentile(car_data_fill_na_no_outliers['Price'], 25) - 1.5 * iqr
outliers =
car_data_fill_na_no_outliers[(car_data_fill_na_no_outliers['Price'] >
upper_outlier_bound) | (car_data_fill_na_no_outliers['Price'] <
lower_outlier_bound)]
```

El código calcula el rango intercuartílico, utilizando la función percentil de numpy, encontrando los cuartiles superiores e inferiores donde yacen los outliers. Posteriormente, se identifican los outliers y luego se eliminan de los dataframes que contendrá tanto los valores nulos sustituidos como los eliminados, los cuales serán llamados car_data_fill_na_no_outliers, el cual contiene las variables nulas sustituidas anteriormente con los outliers eliminados; y car_data_nona_no_outliers, el cual contendra la data con las variables nulas eliminadas sin los outliers.

*El código mostrado anteriormente solo crea uno de los arreglos, el otro implementado es exactamente igual pero con el nombre car_data_nona_no_outliers en lugar de car_data_fill_na_no_outliers.

Luego de definir entonces estos 4 dataframes(car_data_fill_na_no_outliers, car_data_nona_no_outliers, car_data_nona y car_data_fill_na), se realiza un proceso de normalización para cada uno de ellos. La normalización es un

escalamiento de datos con el fin de tener una escala común, para facilitar el análisis y la precisión del modelo, ya que facilita la comparación entre los datos.



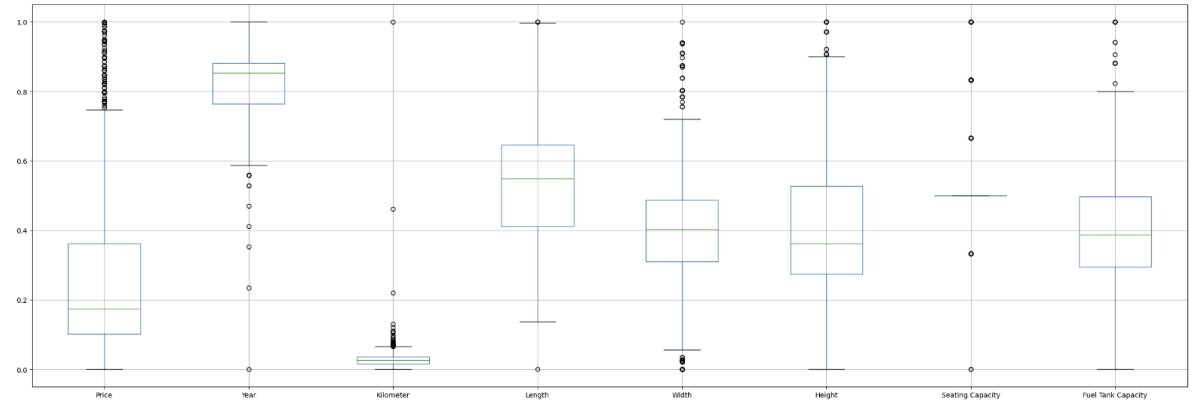
Boxplot donde se puede apreciar la diferencia entre las escalas de precio y kilometraje contra las otras variables continuas.

A partir de aqui, se aplico normalización mediante el siguiente algoritmo:

```
to_scale = ['Price',"Year","Kilometer","Length","Width","Height","Seating Capacity","Fuel Tank Capacity"]

for column in to_scale:

    x_min = car_data_notna[column].min()
    x_max = car_data_notna[column].max()

    car_data_notna[column] = (car_data_notna[column] - x_min)/(x_max - x_min)
```

Este es un algoritmo min max, donde se resta el valor mínimo de cada valor, y luego divide dicho resultado por el rango de valores. De esta manera, todos los valores de las columnas especificadas, son escalados de su valor original a valores en el rango entre 1 y 0.

Luego de la normalización, podemos apreciar que los valores ahora se encuentran en una misma escala donde son comparables de una forma mas sencilla



Boxplot donde se puede apreciar una similitud de las escalas entre todas las variables continuas.

Para las variables categóricas, se aplicó One Hot Encoding, mediante la librería category_encoders. El one-hot encoding es una técnica de preparación de datos que se utiliza para convertir variables categóricas en variables numéricas. Esto es necesario para que los modelos de aprendizaje automático puedan entender y procesar las variables categóricas. Este crea para cada variable categórica, una columna binaria para cada valor existente de la variable categórica, y luego se le asigna un valor de 1 o 0, a cada fila, dependiendo de si el valor original pertenencia a esta categoría (se coloca 1 si pertenecía, 0 si no)

```
encoder = OneHotEncoder(handle_unknown='ignore', use_cat_names=True,
cols = ["Make", "Fuel Type", "Transmission", "Owner"])

encoder.fit(car_data_notna)

car_data_notna = encoder.transform(car_data_notna)
```

Esto se realiza para todos los dataframes a ser estudiados, puesto que es necesario que el modelo use valores numéricos para su entrenamiento.

Finalmente, se exportan los 4 dataframes anteriormente mencionados como archivos .csv, para poder ser evaluados en el modelo.

IMPLEMENTACIÓN

Se implementó el algoritmo de descenso de gradiente aprovechando las facilidades de las librerías de Python para análisis de datos (pandas, numpy, matplotlib) de forma que se aprovechaban para tratar el problema con un enfoque matricial, eliminando así la necesidad de programar utilizando múltiples ciclos.

El código del algoritmo es el siguiente:

```
    # Algoritmo de descenso gradiente
    for i in range(iters):
          hw = np.dot(X,W)
           E = Y - hw
          err_it = np.mean(abs(E))
          self.errores.append(err_it)
          W = W + (learning_rate * (np.dot(X.T, E) * (2/len(X))))
          if err_it <= epsilon:
                # Convergencia Alcanzada
                print(f'Iteracion de Convergencia: {i}')
                break
```

- X: Es la matriz que contiene los valores de las variables del conjunto de datos de entrenamiento.
- Y: Es el vector de valores reales de la función del conjunto de datos de entrenamiento.
- W: Es el vector de pesos calculado y actualizado en cada iteración. La actualización del vector consiste en la aplicación de la siguiente fórmula:

$$w_i \leftarrow w_i + \alpha \sum_j x_{j,i}(y_j - h_{\mathbf{w}}(\mathbf{x}_j)) \ .$$

- hw: Es el vector de hipótesis. Contiene los valores estimados para la función en cada iteración. El cálculo de este vector consiste en multiplicar el vector de pesos (W) por la matriz de variables (X).
- E: Es el vector de errores. Contiene los errores calculados en cada iteración según el vector de hipótesis y el vector de valores reales de la función. El

cálculo de este vector consiste en restar el vector de hipótesis (hw) al vector de valores reales (Y).

- err_it: Es el valor del promedio de los errores de la iteración.

*El resto de la implementación del modelo y sus métodos están documentados dentro del código.

ENTRENAMIENTO

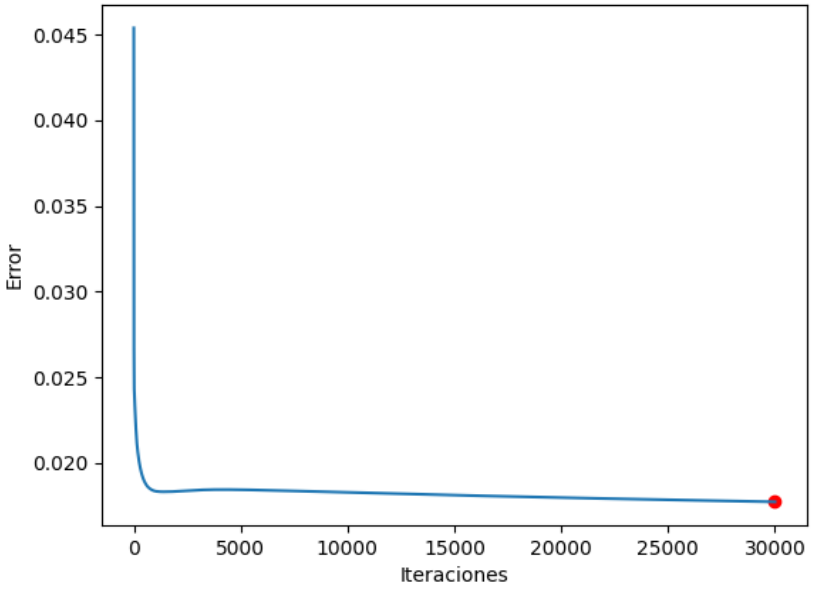
Para el entrenamiento del modelo, se dividió el conjunto de datos en 2 partes. 80% fue utilizado para el entrenamiento y 20% fue utilizado para las pruebas. Ambas partes son escogidas de forma aleatoria con la función “train_test_split” de *scikit-learn*. Al escoger “random_state = 42” se garantiza que se obtenga la misma división para distintas ejecuciones. El objetivo de realizar un train_split_split, es evitar el overfitting para el modelo, puesto que si se entrenará con el conjunto de datos completo, este predeciría perfectamente cualquier entrada proveniente de nuestro único set de datos. Utilizando train_test_split, el universo se reduce a una muestra del conjunto total, para así dejar espacio a ver como el modelo se comporta ante data desconocida.

PRUEBAS Y VALIDACIÓN

Se ejecutaron pruebas sobre 4 conjuntos de datos distintos. Cada conjunto de datos fue tratado de manera distinta para poder analizar los efectos de dichos tratamientos en la precisión final del modelo.
Al ejecutar cada prueba también se calculó el error relativo, el cual es una estimación o predicción en relación con el valor real. Se calcula dividiendo el error absoluto entre el valor real. Este representa la diferencia porcentual entre el valor real y el valor obtenido. Es decir, un error relativo bajo indica que se está haciendo una estimación más precisa.

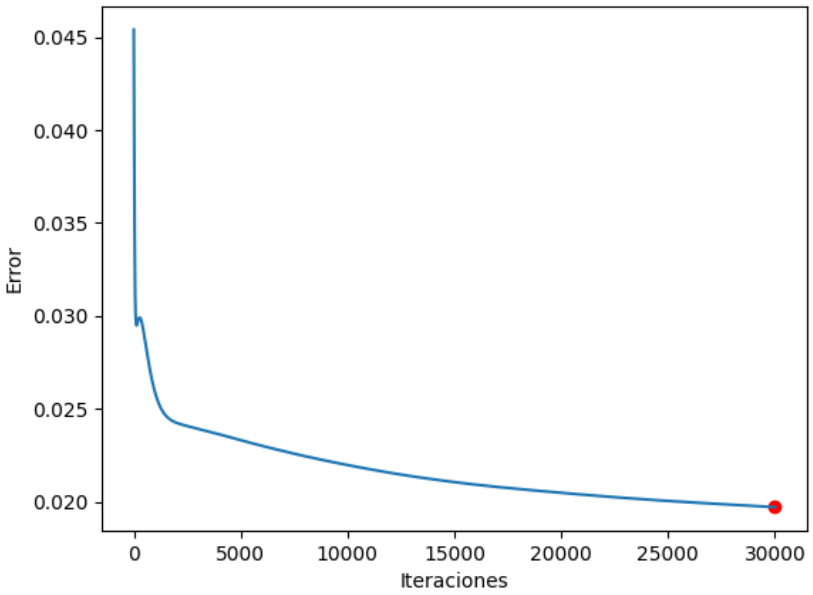
Con el conjunto de datos al que se le aplicó **eliminación de valores nulos** Se obtuvieron las siguientes curvas de aprendizaje:

- Tasa de aprendizaje de 0.1, 30000 iteraciones:



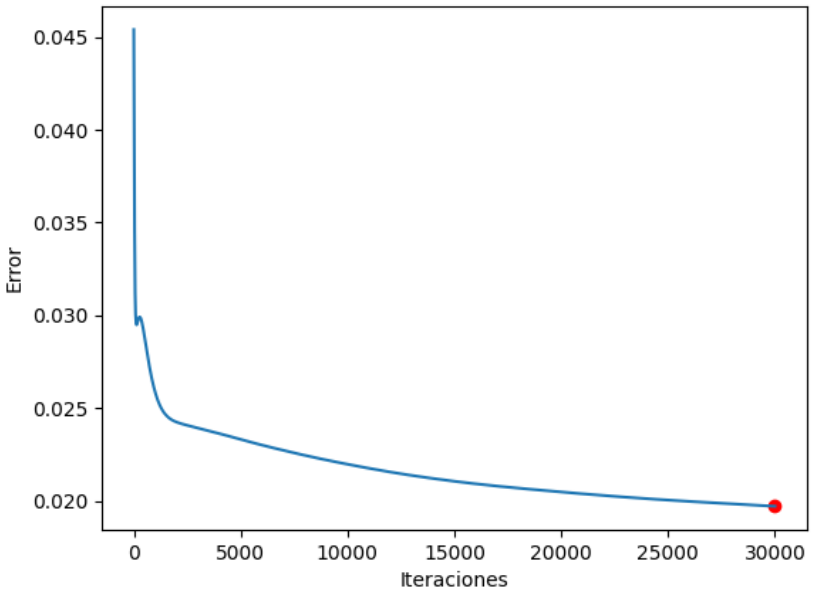
error relativo = 0.4385387682039966

- Tasa de aprendizaje 0.01, 30000 iteraciones:



error relativo = 0.47064498348016043

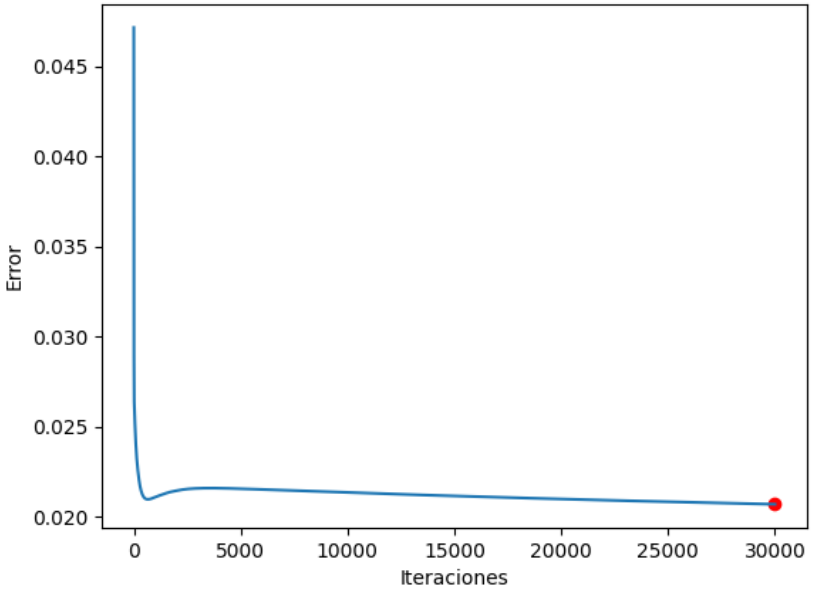
- Tasa de aprendizaje 0.001, 30000 iteraciones:



error relativo = 0.47064498348016043

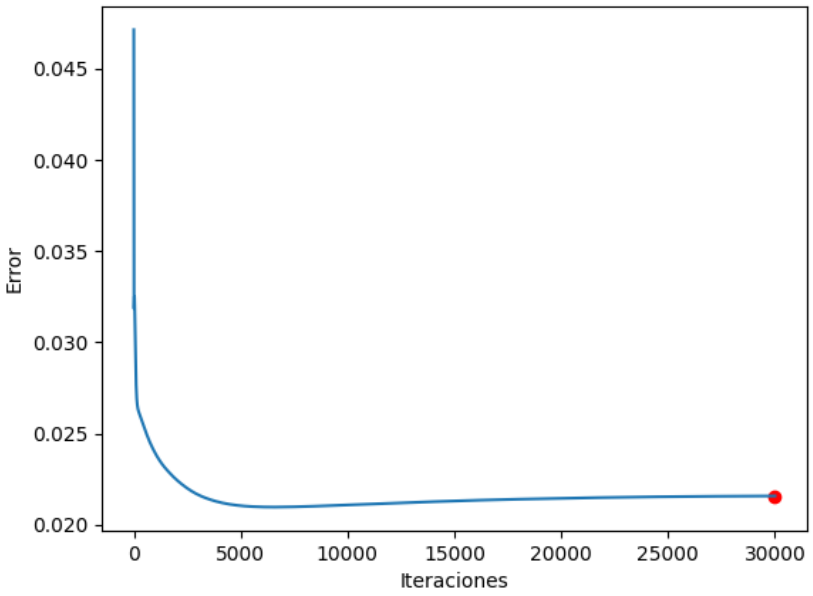
Con el conjunto de datos al que se le aplicó **sustitución de los valores nulos por la moda para variables categóricas y la media para variables continuas** se obtuvieron las siguientes curvas de aprendizaje:

- Tasa de aprendizaje de 0.1, 30000 iteraciones:



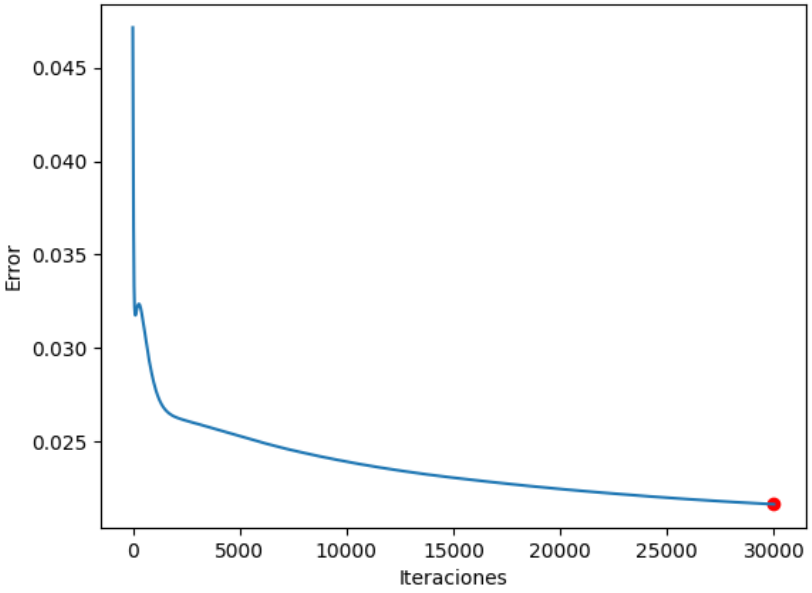
error relativo = 0.4624940442552044

- Tasa de aprendizaje 0.01, 30000 iteraciones:



error relativo = 0.48240084380913156

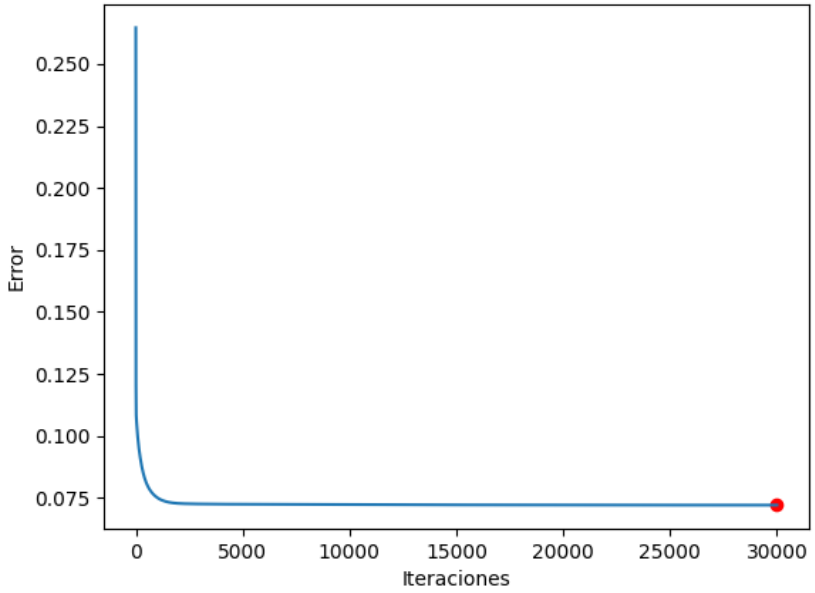
- Tasa de aprendizaje 0.001, 30000 iteraciones:



error relativo = 0.46524817321228495

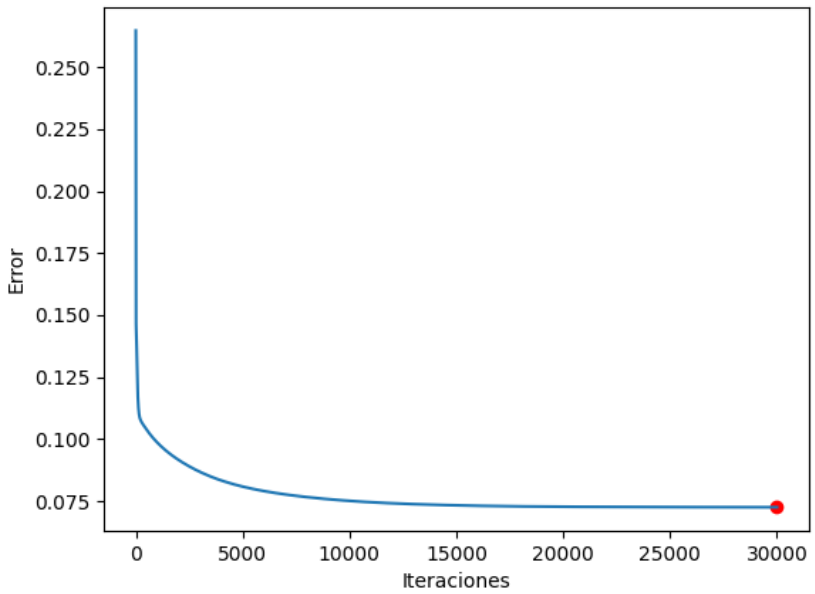
Con el conjunto de datos donde **se eliminaron los outliers, los valores nulos fueron sustituidos por la moda para variables categóricas y la media para las variables continuas** se obtuvieron las siguientes curvas de aprendizaje:

- Tasa de aprendizaje de 0.1, 30000 iteraciones:



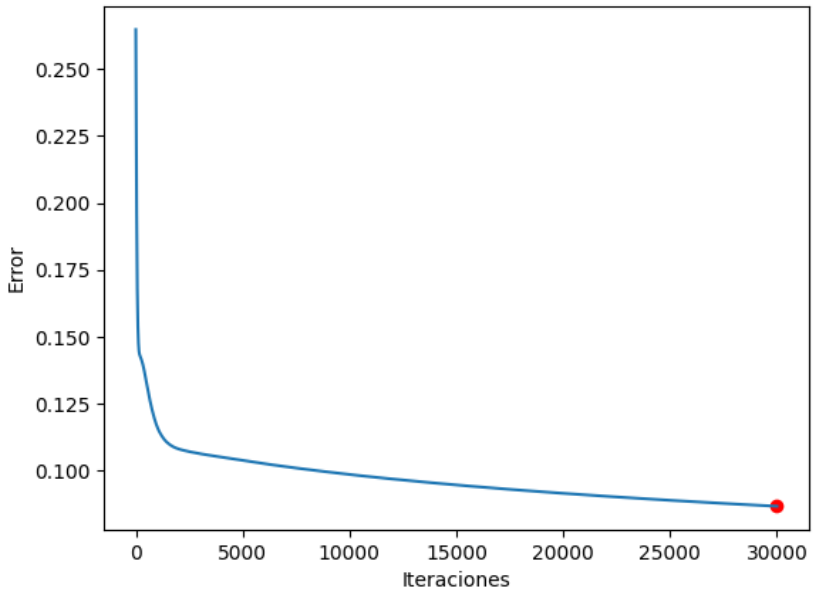
error relativo = 0.2844872433379333

- Tasa de aprendizaje 0.01, 30000 iteraciones:



error relativo = 0.29072708976984996

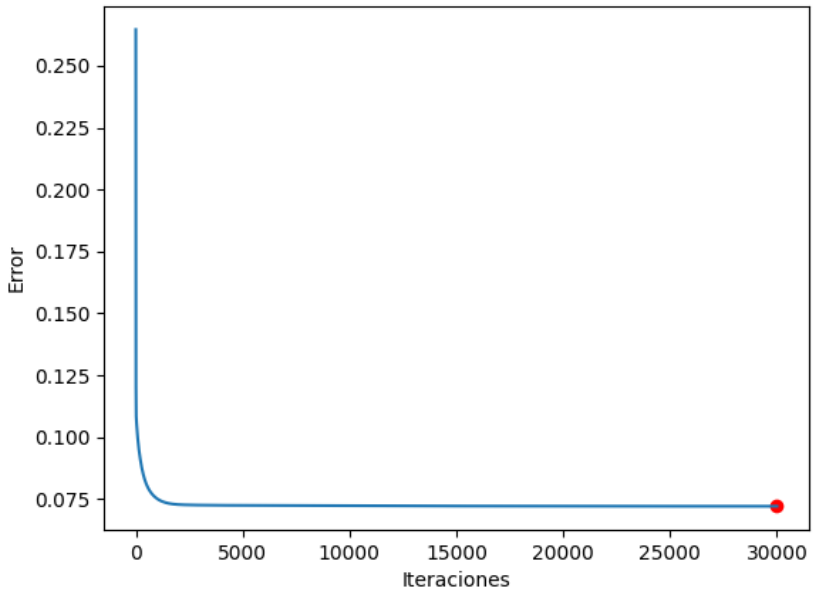
- Tasa de aprendizaje 0.001, 30000 iteraciones:



error relativo = 0.36525572457426225

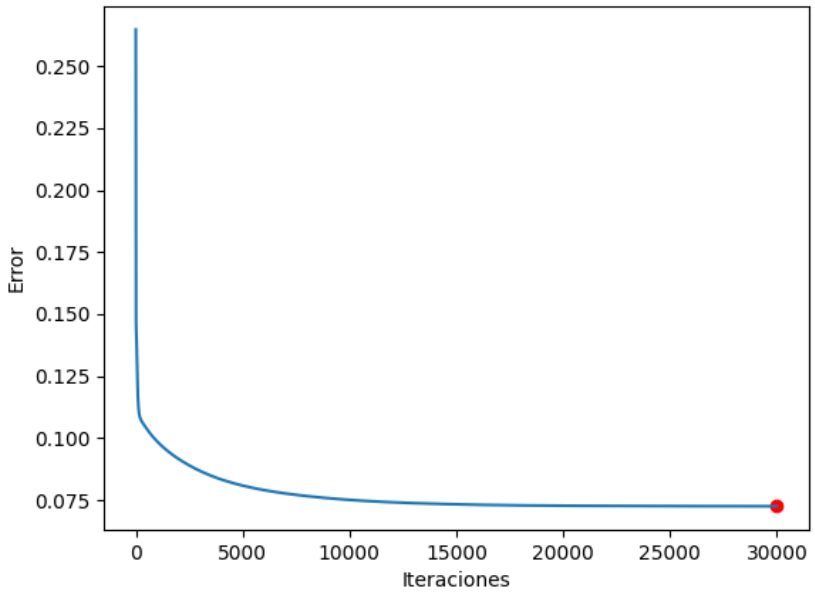
Con el conjunto de datos donde **se eliminaron los outliers y los valores nulos** se obtuvieron las siguientes curvas de aprendizaje:

- Tasa de aprendizaje de 0.1, 30000 iteraciones:



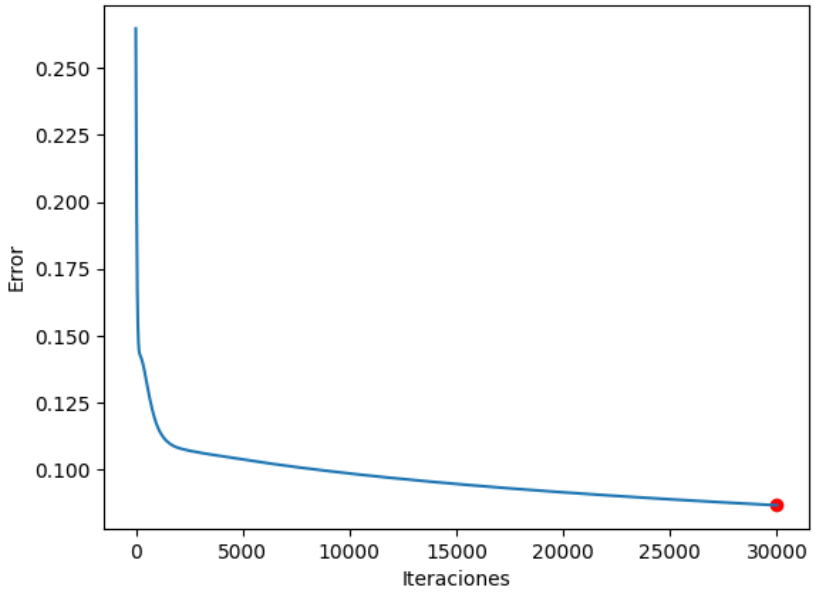
error relativo = 0.2844872433379333

- Tasa de aprendizaje 0.01, 30000 iteraciones:



error relativo = 0.29072708976984996

- Tasa de aprendizaje 0.001, 30000 iteraciones:



error relativo = 0.36525572457426225

Hipótesis final seleccionada: **Dataframe sin outliers y sin valores nulos**

	Valores	Variables
0	-0.110315	Make_Honda
1	-0.150469	Make_Maruti Suzuki
2	-0.134738	Make_Hyundai
3	-0.037681	Make_Toyota
4	0.213343	Make_BMW
5	-0.065492	Make_Skoda
6	-0.154024	Make_Nissan
7	-0.170062	Make_Renault
8	-0.149291	Make_Tata
9	-0.138966	Make_Volkswagen
10	-0.085118	Make_Ford
11	0.126392	Make_Mercedes-Benz
12	0.094470	Make_Audi
13	-0.199875	Make_Mahindra
14	-0.022598	Make_MG
15	-0.031503	Make_Jeep
16	-0.090467	Make_Kia
17	0.031341	Make_Volvo
18	0.342525	Make_Porsche
19	-0.076841	Make_Isuzu
20	0.304248	Make_Jaguar
21	0.333218	Make_Land Rover
22	0.294100	Make_MINI
23	-0.127182	Make_Mitsubishi
24	-0.203001	Make_Datsun
25	-0.181167	Make_Chevrolet
26	-0.289351	Make_Ssangyong
27	-0.131412	Make_Fiat
28	0.383459	Make_Lexus
29	0.868626	Year
30	-0.297019	Kilometer
31	-0.062179	Fuel Type_Petrol
32	-0.030090	Fuel Type_Diesel
33	-0.088750	Fuel Type_CNG
34	-0.019123	Fuel Type_CNG + CNG
35	0.012732	Fuel Type_LPG
36	0.000000	Fuel Type_Petrol + CNG
37	-0.248673	Transmission_Manual
38	-0.177782	Transmission_Automatic
39	0.077183	Owner_First
40	0.059753	Owner_Second
41	0.080053	Owner_Third
42	0.065714	Owner_UnRegistered Car
43	0.175608	Seating Capacity
44	0.386554	Fuel Tank Capacity
45	-0.426455	t_ind

CONCLUSIONES

La hipótesis seleccionada como solución del problema, es la hipótesis generada por el entrenamiento del modelo con el dataframe sin outliers y con la eliminación de valores nulos y un learning_rate de 0.1, ya que este es el que obtiene el menor valor de error relativo.

La razón por la que el learning_rate de 0.1 funciona mejor que otros valores, es que el learning rate de 0.1 hace que el algoritmo de descenso de gradiente converja más rápidamente. A pesar de esto, utilizar valores de learning rate resulta en variaciones relativamente pequeñas del error relativo, por lo que podemos atribuir también gran parte de los resultados obtenidos, al conjunto de datos procesados con el que fue entrenado.

Cabe destacar, que a pesar de que los estudios ante los dataframes que tuvieron eliminación de valores nulos y sustitución de valores nulos tuvieran resultados similares, y se eligiera aquel donde se eliminan las filas con valores nulos, esto se debe a la presencia de una cantidad pequeña de filas con valores nulos, ya que de haber una proporción más grandes de los mismos, lo preferible hubiera sido optar por el que contiene la sustitución de los datos en lugar de los eliminados, debido a que mientras mas data tenga un modelo para entrenar, mejor es su desempeño; eliminar variables solo resultaría en una pérdida de data que puede ser significativa en estos casos.

Ante los resultados observados, podemos concluir que el modelo implementado con el algoritmo de descenso de gradiente, no es un algoritmo de predicción infalible, al igual que ningún otro algoritmo lo será. De ser perfecto, sería meramente por overfitting y el modelo tendría un peor desempeño ante data desconocida. Para este, caso, dado que el universo de muestras era solo lo contenido en el archivo CarDekho, no era viable entrenar el modelo con todo el set de datos puesto que no había más data de donde probar como funciona la predicción ni hubiera un fin real si se buscara predecir precios reales con dicho modelo, no es el sujeto de estudio.

Se observó también que el modelo entrenado con la data donde se eliminaron los outliers, tiene un mejor desempeño de predicción, esto debido a que los outliers tienen un valor muy grande que desestabiliza los pesos y el modelo busca entonces de encontrar un balance entre dichos valores tan gigantes con la media de valores mucho más pequeños. Al buscar dicho balance, los valores de autos con menor precio tienden a tener un valor mucho más alto que el que deberían de tener y los autos con un precio muy grande, tienen un valor muy reducido comparable con lo imaginado. Si el universo fuera más diverso, tal vez se pudiera encontrar un modelo que funcione de mejor forma, pero este no es el caso. Se puede afirmar entonces, que los modelos de predicción tienen un mejor funcionamiento ante un conjunto de datos que se encuentra en una escala similar.