Alunos: Gustavo Borges Maranhão Paulino e Henrique Barros Silva

Relatório de Regressão

Introdução

A regressão é uma técnica essencial em Inteligência Artificial (IA) e análise de dados que desempenha um papel fundamental na previsão de valores numéricos com base em dados históricos. Ela é uma ferramenta poderosa que permite modelar e compreender relações entre variáveis, fornecendo insights valiosos para tomada de decisões informadas em uma variedade de aplicações.

Neste contexto, este relatório introdutório explorará os conceitos básicos da regressão, descrição do projeto, resultados e análises das aplicações dos códigos explicados em sala. À medida que avançamos, vamos mergulhar mais fundo nos detalhes dessa técnica estatística essencial para entender seu funcionamento e potencial nos algoritmos de regressão linear e gradiente descendente.

Descrição do projeto

O nosso projeto se baseia em um DataSet chamado autos.csv, este conjunto de dados é uma coleção abrangente de dados valiosos sobre carros usados e fornece informações sobre como os carros estão sendo vendidos, por qual preço estão sendo vendidos e todos os detalhes sobre sua condição.

Cada anúncio contém informações como modelo do carro, marca à qual ele pertence, tipo de anúncio (particular ou revendedor), tipo de oferta, preço, informações de teste A/B, tipo de veículo, ano de registro (em que ano o carro foi registrado pela primeira vez), tipo de câmbio, potência em PS (cavalos de potência), quantos quilômetros ele percorreu até agora, mês de registro (quando foi registrado pela primeira vez, dando uma ideia de sua idade), tipo de combustível utilizado (gasolina/diesel/eletricidade/GLP, etc.), danos não reparados – se houver algum dano no veículo que ainda não foi reparado. Grande parte desses fatores são instrumentais para determinar um preço adequado para veículos usados.

Nosso objetivo com este DataSet, será fazer previsões dos preços de carros usados utilizando algoritmos de Aprendizado De Máquina. Neste exercício, daremos ênfase aos algoritmos de Regressão Linear e Gradiente Descendente.

Exploração dos dados

Primeiramente, analisamos a forma em que os dados estavam preenchidos no DataSet escolhido. Fazendo esta análise, verificamos que muitos dos atributos presentes, tinham dados não preenchidos. Isto ocorre, porque em várias plataformas estes campos são digitáveis pelo usuário. A **imagem A**, ilustra a porcentagem de valores não preenchidos para cada coluna.

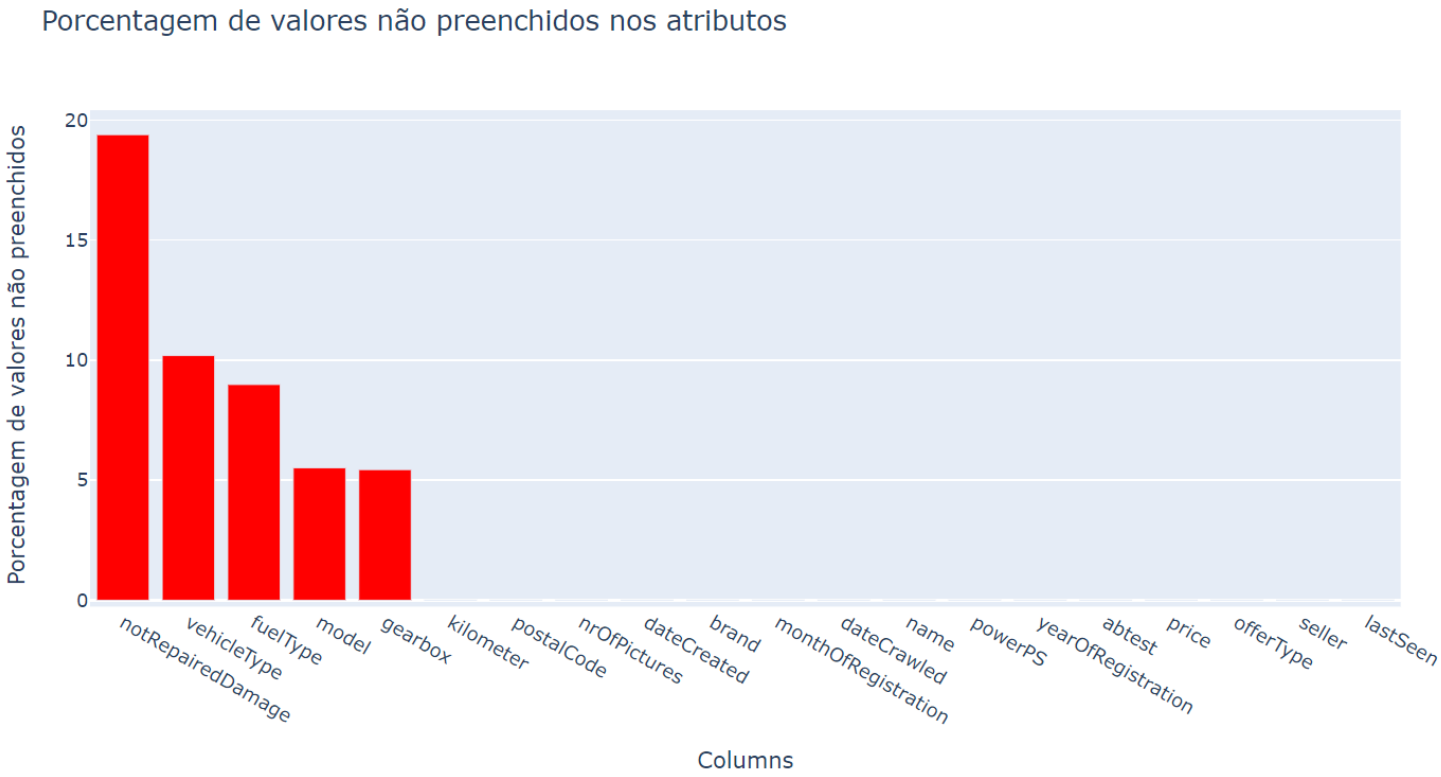

Imagem A

Para não perdermos as instâncias que possuem valores não preenchidos, uma alternativa é definirmos um valor "padrão" para elas. Nos campos "notRepairedDamage", "vehicleType", "fuelType" e "gearbox", bastará analisarmos o valor que mais se repete nestes atributos e atribuí-los aos dados não preenchidos. Já o atributo "model", por ser algo muito específico, iremos colocar como "unknow".

Por indução, podemos deduzir que alguns atributos do Dataset não fazem sentido para predição do valor dos veículos. Como "nrOfPictures", que significa o número de imagens no anúncio do veículo.

Estes campos, sendo eles: número de imagens (nrOfPictures), última visualização (lastSeen), data do anúncio (dateCrawled), código postal (postalCode), nome (name), mês de registro (monthOfRegistration), tipo de anúncio (seller) e tipo de oferta (offerType), somente iriam atrapalhar nossa análise, portanto decidimos removê-los do Dataset.

Analisando a forma com que os dados estão sendo preenchidos, reparamos que "notRepairedDamage", "vehicleType", "fuelType", "gearbox" e "model" estão sendo preenchidos com strings. Com estes dados representados desta forma, não podemos fazer análises estatísticas sobre eles.

Para solucionar este problema, utilizamos a função LabelEncoder da biblioteca sklearn para transformar esses atributos classificativos em numéricos. A **imagem B** ilustra o antes e depois desta transformação. Um adento aqui, é que posteriormente, no ato do treinamento, será utilizado OneHotEncoding nestes atributos.

	model	notRepairedDamage	vehicleType	fuelType	gearbox
0	golf	nein	limousine	benzin	manuell
1	unknown	ja	coupe	diesel	manuell
2	grand	nein	suv	diesel	automatik
3	golf	nein	kleinwagen	benzin	manuell
4	fabia	nein	kleinwagen	diesel	manuell
5	3er	ja	limousine	benzin	manuell
6	2_reihe	nein	cabrio	benzin	manuell
7	andere	nein	limousine	benzin	manuell
8	c_max	nein	bus	benzin	manuell
9	golf	nein	kleinwagen	benzin	manuell

ANTES DE UTILIZAR ENCODER

	model	notRepairedDamage	vehicleType	fuelType	gearbox
0	117	1	6	1	1
1	228	0	3	3	1
2	118	1	7	3	0
3	117	1	4	1	1
4	102	1	4	3	1
5	11	0	6	1	1
6	8	1	2	1	1
7	40	1	6	1	1
8	61	1	1	1	1
9	117	1	4	1	1

DEPOIS DE UTILIZAR O ENCODER

Imagem B

Após obtermos todos os dados preenchidos com números, o próximo passo, é encontrar a relação entre os atributos. Para isto, utilizamos a Matriz De Correlação para identificarmos quais atributos mais influenciam no preço do veículo. A **imagem C** mostra como ficou a Matriz de Correlação.

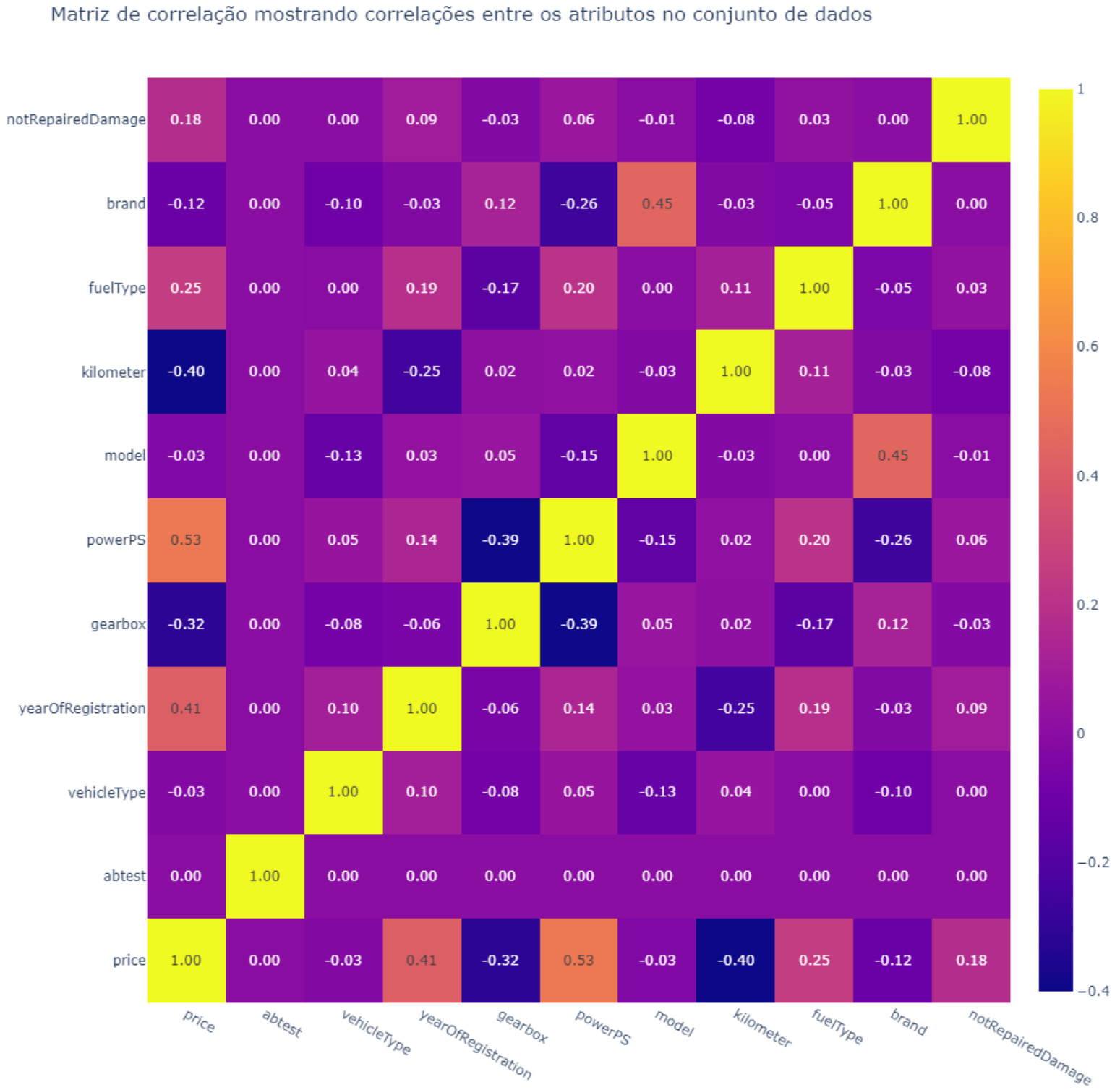

Imagem C

Com a Matriz de Correlação em mãos, podemos notar que o atributo Preço (primeira coluna), está fortemente correlacionado ao Atributo Potência em Cavalos (PowerPS), onde quanto maior a potência, maior o preço do carro. Outro fator importante, é o atributo KM rodados (kilometer), onde esta relação é negativa, ou seja, quanto maior a quilometragem de um veículo, menor é o seu valor, já que está desgastado.

Nosso próximo passo, é a remoção de outliers, uma vez que, observamos que os valores máximos e mínimos dos campos numéricos não condiziam com a realidade. O atributo ano de registro "yearOfRegistration", por exemplo, possuía o valor mínimo de 1000 e máximo de 9999. Para solucionar este problema, removemos outliers usando o método Intervalo Interquartil (IQR). A **imagem D** ilustra a discrepância dos dados antes e depois da remoção de outliers.

	price	yearOfRegistration	powerPS	kilometer
count	3.715280e+05	371528.000000	371528.000000	371528.000000
mean	1.729514e+04	2004.577997	115.549477	125618.688228
std	3.587954e+06	92.866598	192.139578	40112.337051
min	0.000000e+00	1000.000000	0.000000	5000.000000
25%	1.150000e+03	1999.000000	70.000000	125000.000000
50%	2.950000e+03	2003.000000	105.000000	150000.000000
75%	7.200000e+03	2008.000000	150.000000	150000.000000
max	2.147484e+09	9999.000000	20000.000000	150000.000000

ANTES DA REMOÇÃO

	price	yearOfRegistration	powerPS	kilometer
count	365154.000000	365154.000000	365154.000000	365154.000000
mean	5128.230440	2003.467359	109.930254	126648.290858
std	5885.062633	7.110265	65.927772	39125.523789
min	0.000000	1968.000000	0.000000	5000.000000
25%	1100.000000	1999.000000	69.000000	125000.000000
50%	2900.000000	2003.000000	105.000000	150000.000000
75%	6950.000000	2008.000000	145.000000	150000.000000
max	34200.000000	2019.000000	481.000000	150000.000000

DEPOIS DA REMOÇÃO

Imagem D

Ainda observando a **imagem D**, percebemos que os campos numéricos possuem valores bastante distante uns dos outros, ou seja, em escalas muito grandes. Isto impacta negativamente no resultado final do aprendizado de máquina.

Para tratar isto, iremos normalizar estes dados, colocando todos eles em uma escala entre 0 e 1. Para isto aplicaremos a fórmula:

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Imagem E

Agora que já temos os campos numéricos normalizados entre 0 e 1, outro problema importante surge à tona. Nossos atributos de classificação, embora estejam numéricos, estão recebendo números em sequência como pudemos ver na **imagem B.**

Isto nos trará problemas, pois o algoritmo de Aprendizado de Máquina irá entender erroneamente, que valores mais altos destas classificações podem inferir algo sobre o modelo. Para evitar que isto aconteça, nossa alternativa escolhida é utilizar OneHotEncoding nestas colunas de classificação.

Após utilizarmos o OneHotEncoding nos atributos "notRepairedDamage", "gearbox", "fuelType", "brand" e "model" obtemos uma matriz com 306 colunas. Uma para cada item de classificação, como observado na **imagem F**.

	onehotencoder__notRepairedDamage_ja	onehotencoder__notRepairedDamage_nein	onehotencoder__gearbox_automatik	onehotencoder__gearbox_manuell	oneh
0	0.0	1.0	0.0	1.0	
1	1.0	0.0	0.0	1.0	
2	0.0	1.0	1.0	0.0	
3	0.0	1.0	0.0	1.0	
4	0.0	1.0	0.0	1.0	
5	1.0	0.0	0.0	1.0	
6	0.0	1.0	0.0	1.0	
7	0.0	1.0	0.0	1.0	
8	0.0	1.0	0.0	1.0	
9	0.0	1.0	0.0	1.0	

rows × 306 columns

Imagem F

Por fim, uma vez que os dados já estejam todos normalizados, dividimos nosso Dataframe em duas matrizes. Sendo uma contendo 80% dos dados para treino e outra com 20% dos dados para teste.

Resultados

Com as matrizes de teste e de treino em mãos, ambas já normalizadas, utilizamos SGDRegressor da biblioteca sklearn para treinarmos o modelo aplicando o algoritmo de Gradiente Descendente.

No Gradiente Descendente, fizemos três abordagens diferentes, a primeira com a learning_rate = "adaptive", onde o modelo ajusta a taxa de aprendizado à medida que o treinamento progride. Com essa abordagem conseguimos um Mean Squared Error (MSE) de 12.368.003 e um R2 Score de 63%. **A imagem G** ilustra o resultado desta abordagem.

Utilizando Taxa de Aprendizado "Adaptativa"

eta = eta0, desde que o treinamento continue diminuindo. Cada vez que n_iter_no_change épocas consecutivas falham em diminuir a perda de treinamento em tol ou falham em aumentar a pontuação de validação em tol se early_stopping for True, a taxa de aprendizado atual é dividida por 5

```
In [87]: model = SGDRegressor(max_iter = 1000, learning_rate = "adaptive")
         model.fit(X_train, y_train)

Out[87]: SGDRegressor(learning_rate='adaptive')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [88]: resultado = model.predict(X_test)
         print("\nMean Squared Error (MSE):", mean_squared_error(y_test, resultado))
         print(f"R2 Score : {r2_score(y_test, resultado)}")

         Mean Squared Error (MSE): 12367907.904777259
         R2 Score : 0.6367509323439127
```

Imagem G

A segunda foi com a learning_rate = "constant" e com a taxa de aprendizagem (eta0) = 0,01, isso significa que a taxa de aprendizado será mantida constante durante todo o processo de treinamento e o valor inicial da taxa de aprendizad0. Com essa abordagem conseguimos um Mean Squared Error (MSE) de 12.487.921 e um Coeficiente de Determinação (R2) de 63%. Um resultado um pouco pior, porém, com um processamento bem mais rápido. Já que não varia o cálculo da taxa de aprendizado.

A terceira foi utilizando os mesmos parâmetros da segunda, porém utilizando uma taxa de penalidade “L1”, em que adiciona a soma dos valores absolutos dos coeficientes como termo de penalidade à função de perda. Com essa abordagem conseguimos um Mean Squared Error (MSE) de 12.717.488 e um Coeficiente de Determinação (R2) de 62%.

Após testar diferentes cenários no Gradiente Descendente, passamos para o algoritmo de Regressão Linear para treinar nosso modelo. Utilizando este método, obtivemoso valor 12.336.712 como MSE e Coeficiente de Determinação (R2) de 63%, conforme ilustrado na **imagem H.**

Aplicando Regressão Linear

```
In [89]: model = LinearRegression()
         model.fit(X_train, y_train)
         resultado = model.predict(X_test)
         print("\nMean Squared Error (MSE):", mean_squared_error(y_test, resultado))

         Mean Squared Error (MSE): 12336712.317435939

In [90]: model.score(X_test, y_test)

Out[90]: 0.6376671558559215
```

Imagem H

Análise

Como utilizamos uma base de dados bem extensa, contendo incialmente 371.825 instâncias, preenchida com dados reais e com altíssima dispersão dos dados, não conseguimos um Coeficiente de Determinação (R2) muito alto.

Mesmo após a normalização e remoção de outliers, foram processadas 365.154 instâncias nos algoritmos de Regressão Linear e Gradiente Descendente, conforme ilustrado na **imagem J.**

```
In [67]: one_hot_df["remainder__yearOfRegistration"] = (one_hot_df["remainder__yearOfRegistration"] - one_hot_df["remainder__yearOfRegistr
         one_hot_df["remainder__powerPS"] = (one_hot_df["remainder__powerPS"] - one_hot_df["remainder__powerPS"].min()) / (one_hot_df["rem
         one_hot_df["remainder__kilometer"] = (one_hot_df["remainder__kilometer"] - one_hot_df["remainder__kilometer"].min()) / (one_hot_c

         one_hot_df.head(10)
```

Out[67]:

	onehotencoder__notRepairedDamage_ja	onehotencoder__notRepairedDamage_nein	onehotencoder__gearbox_automatik	onehotencoder__gearbox_manuell	oneh
0	0.0	1.0	0.0	1.0	
1	1.0	0.0	0.0	1.0	
2	0.0	1.0	1.0	0.0	
3	0.0	1.0	0.0	1.0	
4	0.0	1.0	0.0	1.0	
5	1.0	0.0	0.0	1.0	
6	0.0	1.0	0.0	1.0	
7	0.0	1.0	0.0	1.0	
8	0.0	1.0	0.0	1.0	
9	0.0	1.0	0.0	1.0	

10 rows × 306 columns

```
In [77]: one_hot_df.shape
Out[77]: (365154, 306)
```

Imagem J

Mesmo com um grande número dados processados (365.154 instâncias), obtivemos um Mean Squared Error (MSE) aceitável (12.368.032,64), isso tendo em vista que a média de preços dos veículos é um valor alto (5.128,23).