Previsão do Preço de Carros Usados

Aline Carvalho

Instituto Politécnico de Coimbra, Coimbra Business School, ISCAC, Coimbra, Portugal

a2020106700@alumni.iscac.pt

***Resumo* — O mercado de carros usados está em crescente desenvolvimento. Surge então o desafio de encontrar um correto preço, seja para vender ou para comprar. Este artigo pretende prever o preço dos carros usados com base nas suas caraterísticas, recorrendo à regressão linear e à *random forest regression*. Os resultados indicam que o último método produz resultados mais precisos.**

Palavras-chave — previsão, machine learning, regressão linear, random forest regression, python.

I. Introdução

Os carros usados ocupam uma grande porção no mercado automóvel. Nos E.U.A., o número de transações de carros usados é três vezes maior do que o número de transações de carros novos [1]. Uma avaliação correta do preço dos carros usados é importante para o mercado automóvel pois permite: aos clientes a compra e venda de automóveis sem preocupações; às empresas de aluguer calcular o preço adequado para os seus serviços, ao banco para controlar melhor o valor concebido de empréstimo [2].

O crescente desenvolvimento do mercado de itens usados levanta um desafio aos comerciantes e particulares que pretendam realizar uma venda: o preço a aplicar [3]. Aliás, este problema também surge do outro lado: o consumidor. Coloca-se a questão se o preço aplicado é um preço justo. Para resolver tal problema, os investigadores têm recorrido aos métodos de previsão.

Em 2005, Kuiper [4] utilizou o modelo de regressão multivariada para prever o preço dos carros da General Motors. Já Listiani [5] recorreu ao SVM (*Support Vector Machines*) para calcular o preço dos carros para alugar. Noor e Jan [6] exploraram a ideia de utilizar a regressão linear para prever o preço de carros usados. Portanto, é possível perceber a panóplia de possíveis técnicas para prever o preço dos carros.

O artigo encontra-se organizado da seguinte maneira: na parte II é feita a revisão da literatura com base em artigos científicos relacionados com o tema. Na parte III é definida a metodologia a aplicar. De seguida, na parte IV, ocorre o processo de limpeza dos dados e de organização do ficheiro de modo a permitir a análise estatística das variáveis em questão. Na parte V, é demonstrada a aplicação dos modelos de previsão e os seus resultados. Por fim, surge a conclusão e as referências bibliográficas.

II. Revisão da Literatura

A previsão do preço de carros usados tem sido tema de diversos projetos na área de análise de dados e programação. Um dos métodos utilizados é a regressão linear. O termo "regressão" define um conjunto de técnicas estatísticas utilizadas para modelar relações entre variáveis e predizer o valor de uma variável dependente a partir de um conjunto de variáveis independentes (preditores) [4].

O método *random forest regression* é um algoritmo de aprendizagem supervisionada que serve para executar tarefas de regressão e classificação. Este método pode ser utilizado mesmo quando os dados não apresentam uma relação linear [8], revelando-se bastante útil e preciso. Contudo, é difícil dizer se é melhor do que a regressão linear, dependendo esta resposta, da natureza dos dados e do adequado tratamento dos mesmos.

Gupta, Kumar e Singh [5], com o intuito de prever o preço de carros usados, recorreram a três métodos: regressão linear, *random forest* e árvore de decisão. A maior precisão nas previsões foi obtida com recurso à regressão linear (48.03% em comparação com 45,34% e 44.67%). Kiran [9] também recorreu à mesma técnica e obtive um modelo cuja precisão foi de 90%. Monburinon e colaboradores [10], também o objetivo de prever o preço de carros usados, recorreu a modelos de regressão: regressão linear, *gradient boosted regression trees* e *random forest regression*. O segundo método obteve valores de média de erros absoluto menor, seguido da regressão linear. Já em outros estudos, o *random forest regressor* obteve valores elevados na precisão das previsões. No estudo de Pal e colaboradores, este método foi capaz de prever os preços de carros usados com uma precisão de 95.82% nos dados de treino e 83.63% nos dados de teste.

III. Metodologia

A base de dados utilizada neste projeto foi retirada do *kaggle* e retrata dados de carros da Índia. Esta encontra-se agrupada em dois ficheiros: train-data.csv e test-data.csv [9]. O primeiro corresponde aos dados utilizados para treinar o modelo e é composto por 6019 linhas e 14 colunas. O segundo tem como função testar o modelo, contendo 1234 linhas e 13 colunas.

O objetivo é implementar técnicas estatísticas e de *machine learning* de modo a conseguir prever o preço dos carros usados, com base nas caraterísticas destes. Para tal, dois métodos são utilizados: regressão linear e *random forest regression*. Com isto, pretende-se avaliar qual é o melhor método para os dados em análise.

A metodologia seguida pode ser segmentada em três partes: pre-processamento, implementação do modelo, verificação da precisão do modelo. Na primeira parte será analisado os tipos de dados presentes no ficheiro, encontrar valores nulos e eliminar os mesmos, compreender a distribuição dos dados e analisar a sua relação com a variável preço. Na segunda parte são aplicados os modelos de regressão linear e de *random forest regression*. Por fim, é analisada a precisão do modelo.

Toda a parte de programação foi realizada no Google Colab, com recurso à linguagem de programação Python.

IV. Implementação e Análise

A. *Limpeza dos dados*

Para trabalhar com as bases de dados em formato csv no Google Colab, foi feito o upload dos ficheiros diretamente da drive local.

```
[1]  # Upload files from local drive

     from google.colab import files
     files.upload()
```

```
Escolher arquivos  2 arquivos
• test-data.csv(application/vnd.ms-excel) - 127946 bytes, last modified: 13/04/2021 - 100% done
• train-data.csv(application/vnd.ms-excel) - 655278 bytes, last modified: 13/04/2021 - 100% done
Saving test-data.csv to test-data.csv
Saving train-data.csv to train-data.csv
```

Os ficheiros foram alocados como *dataframe* através da função do Pandas *read_csv()*.

```
[4]  # Store dataset as dataframe

     train = pd.read_csv("train-data.csv")
     test = pd.read_csv("test-data.csv")
```

O título da coluna "mileage" foi alterado para "fuel economy" por traduzir melhor os dados, uma vez que apresentam informação relativamente ao quão longe um carro pode ir utilizando determinada quantidade de combustível. Os dados apresentam-se com as unidades km/kg ou kmpl (kilometers per liter), de acordo com o tipo de combustível utilizado por cada veículo. É de notar que estas unidades não são comparáveis. De modo a garantir que a variável "fuel_economy" tenha apenas uma unidade, foram eliminadas as linhas cuja unidade fosse km/kg. O ficheiro de treino ficou então com 5909 linhas e o ficheiro de teste com 1215 linhas.

```
# Maintain only rows with kmpl as unit in column fuel_economy

train = train[train["Fuel_economy"].str.contains("km/kg")==False]
test = test[test["Fuel_economy"].str.contains("km/kg")==False]

# Maintain only the numbers as values

train["Fuel_economy"] = train["Fuel_economy"].str.replace("kmpl","")
test["Fuel_economy"] = test["Fuel_economy"].str.replace("kmpl","")

train["Engine"] = train["Engine"].str.replace("CC","")
test["Engine"] = test["Engine"].str.replace("CC","")

train["Power"] = train["Power"].str.replace("bhp","")
test["Power"] = test["Power"].str.replace("bhp","")
```

A coluna preço apresenta valores em Lakh rúpias que corresponde a 150.000 rúpias. Para utilizar a moeda euro, foi criada uma coluna denominada de "price_euro". A coluna "new price" foi eliminada uma vez que esta representa o objetivo: prever os preços de carros usados, e será criada com o modelo proposto. Para o projeto, só será considerada a coluna "price_euro", ignorando "price".

```
[14] # Convert Lakh Rupees to Euro and create a new column

     train["Price_euro"] = train["Price"].apply(lambda x: x*1100)

     # Delete price column

     train.drop("Price",inplace=True, axis=1)

     # Delete new_price column

     train.drop("New_Price",inplace=True, axis=1)
     test.drop("New_Price",inplace=True, axis=1)
```

Os valores nulos foram identificados através da função *isnull()* e foram somados por variável/coluna. No ficheiro de treino haviam 114 valores nulos e no ficheiro de teste 31. As linhas com valores nulos foram eliminadas, recorrendo à função *dropna()*.

```
# Delete rows with NaN values

train.dropna(axis=0, how="any", thresh=None, subset=None, inplace=True)
test.dropna(axis=0, how="any", thresh=None, subset=None, inplace=True)
```

Posteriormente, ao verificar os tipos de variáveis presentes no ficheiro, pelo método *.dtypes*, as colunas que deveriam ser do tipo *int* ou *float* continuavam como *object*. Pelo que tais colunas tinham como valores "null" em texto, reconhecidos como *string* e não NaN. Foram eliminadas as linhas que contivessem esse texto.

```
[31] # Check each column with "object" type for values with "null" as a string and delete rows

     train=train[train["Fuel_economy"].str.contains("null")== False]
     train=train[train["Engine"].str.contains("null")== False]
     train=train[train["Power"].str.contains("null")== False]

     test=test[test["Fuel_economy"].str.contains("null")== False]
     test=test[test["Engine"].str.contains("null")== False]
     test=test[test["Power"].str.contains("null")== False]

     # Convert to float

     train["Fuel_economy"] = train.Fuel_economy.astype(float)
     test["Fuel_economy"] = test.Fuel_economy.astype(float)

     train["Engine"] = train.Engine.astype(float)
     test["Engine"] = test.Engine.astype(float)

     train["Power"] = train.Power.astype(float)
     test["Power"] = test.Power.astype(float)
```

Após conversão para *float*, foi utlizada a função *.describe()* e mais um erro surgiu. Tanto no ficheiro train como no test, verificou-se que a coluna "fuel_economy" possui como mínimo o valor zero. Ora, tal implicaria que o carro não conseguiria deslocar-se mesmo tendo um litro de combustível. Portanto, assume-se que os valores zero correspondem a *null*, pelo que as linhas com os mesmos devem ser eliminadas.

```
[33] # Delete rows where train or test["Fuel_economy"] == 0

     train.drop(train.index[train['Fuel_economy'] == 0], inplace = True)
     test.drop(test.index[test['Fuel_economy'] == 0], inplace = True)
```

A variável "Name" representa a marca e o modelo, havendo no ficheiro de treinos 1 776 valores diferentes. Já no ficheiro de teste, existem 735 nomes únicos. Contudo, existem 781 nomes que só têm um carro no ficheiro de treino e 501 no ficheiro de teste. Por haver muitos valores únicos e isso limitar o modelo, a coluna "Name" foi eliminada. Assim, a marca e o modelo serão desprezados na previsão do preço que irão ser vendidos os carros, apoiando-se apenas nas caraterísticas inerentes ao carro.

Por fim, foram eliminadas as primeiras colunas de cada ficheiro que continham uma numeração desatualizada após as alterações. Também foi efetuado o *reset* do index. Assim, o ficheiro de treino passou a conter 5779 linhas e 11 variáveis, e o ficheiro de teste 1187 linhas e 10 colunas.

B. *Variáveis e Análise descritiva*

Os ficheiros apresentam as mesmas dez variáveis, exceto o ficheiro train que apresenta mais duas variáveis: "price_euro" e "price", sendo que a última será desprezada. Eis a definição de cada variável:

- *Location*: local onde o carro está a ser vendido
- *Year*: ano de produção do modelo
- *Kilometers_Driven*: total de km percorridos
- *Fuel_Type*: tipo de combustível
- *Transmission*: tipo de transmissão
- *Owner_Type*: ordem do dono atual
- *Fuel_economy*: km percorridos por litro de combustível (kmpl)
- *Engine*: capacidade/cilindrada do motor (cc)

· *Power*: potência do carro (bhp)

· *Seats*: número de lugares que o carro possui

· *Price_euro*: preço do carro (€)

As variáveis podem ser agrupadas em dois grupos: qualitativas e quantitativas. As quantitativas são: *price_euro*, *kilometers_driven*, *fuel_economy*, *engine*, *power* e *seats*. As qualitativas são as seguintes: *name*, *location*, *fuel_type*, *transmission*, *owner_type* e *year*. Nos modelos de *machine learning*, as variáveis cujos valores sejam *strings* não podem ser comparadas a valores como *integers*. Portanto, é necessário converter *strings* para *integers*, ou seja, converter variáveis qualitativas em numéricas. Tal será demonstrado na secção das respetivas variáveis.

Variável			
Qualitativa		***Quantitativa***	
Location	*Nominal*	Price_euro	*Contínua*
Fuel_Type	*Nominal*	Year	*Discreta*
Transmission	*Nominal*	Kilometers_Driven	Contínua
Owner_Type	*Ordinal*	Fuel_economy	*Contínua*
		Engine	*Contínua*
		Power	*Contínua*
		Seats	*Discreta*

Foi aplicada a função *.describe()* para ver as principais estatísticas de cada variável quantitativa – média, desvio padrão, mínimo, máximo, 1º quartil, mediana e 3º quartil.

```
[43] # Descriptive Statistics

     train.describe()
```

	Year	Kilometers_Driven	Fuel_economy	Engine	Power	Seats	Price_euro
count	5779.000000	5.779000e+03	5779.000000	5779.000000	5779.000000	5779.000000	5779.000000
mean	2013.479149	5.835925e+04	18.295070	1628.925074	113.691476	5.287939	10576.175463
std	3.169719	9.284468e+04	4.107155	599.172172	53.834573	0.808495	12352.785556
min	1998.000000	1.710000e+02	6.400000	624.000000	34.200000	2.000000	484.000000
25%	2012.000000	3.342000e+04	15.290000	1198.000000	78.000000	5.000000	3949.000000
50%	2014.000000	5.238300e+04	18.200000	1496.000000	98.600000	5.000000	6325.000000
75%	2016.000000	7.246700e+04	21.100000	1991.000000	138.100000	5.000000	11093.500000
max	2019.000000	6.500000e+06	28.400000	5998.000000	560.000000	10.000000	176000.000000

```
test.describe()
```

	Year	Kilometers_Driven	Fuel_economy	Engine	Power	Seats
count	1187.000000	1187.000000	1187.000000	1187.000000	1187.000000	1187.000000
mean	2013.503791	58099.955350	18.346521	1599.800337	110.740830	5.295703
std	3.089821	35546.730977	4.003990	564.473279	51.603221	0.834141
min	2000.000000	1000.000000	7.940000	624.000000	34.200000	2.000000
25%	2011.000000	33572.500000	15.300000	1198.000000	75.940000	5.000000
50%	2014.000000	54000.000000	18.490000	1493.000000	93.700000	5.000000
75%	2016.000000	74000.000000	21.100000	1968.000000	130.000000	5.000000
max	2019.000000	350000.000000	28.400000	5998.000000	616.000000	10.000000

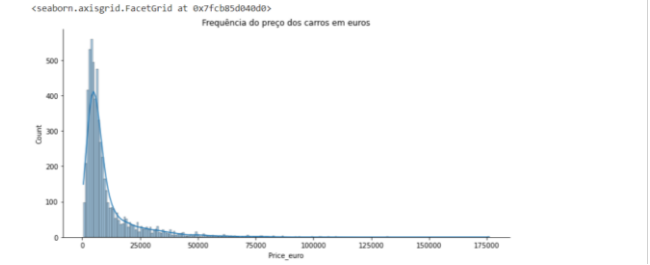
1) Price_euro

De acordo com os resultados da função *.describe()*, os carros custam, em média, 10 576 €. Os valores variam de 484 € a 176 000 €. É de notar que 25% dos carros da amostra custam menos 3 949 € e que outros 25% dos carros custam mais de 11 093 €.

Recorrendo ao *Seaborn*, verificou-se que a distribuição do preço é assimétrica positiva, pelo que se assume que a média é maior do que a mediana. Ou seja, existem mais carros com preço baixos mas como alguns deste possuem valores muito grandes, a média é arrastada para a direita.

```
# Price_euro frequency in a histogram

sns.displot(train['Price_euro'],kde=True, alpha=0.4,aspect=16/8).set(title='Frequência do preço dos carros em euros')
```

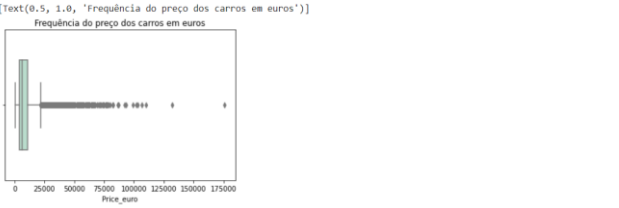
```
<seaborn.axisgrid.FacetGrid at 0x7fcb85d040d0>
```



A caixa de bigodes também permite visualizar a assimetria positiva:

```
sns.boxplot(x=train['Price_euro'],palette=("Pastel2"),width=0.6).set(title='Frequência do preço dos carros em euros')
```

```
[Text(0.5, 1.0, 'Frequência do preço dos carros em euros')]
```



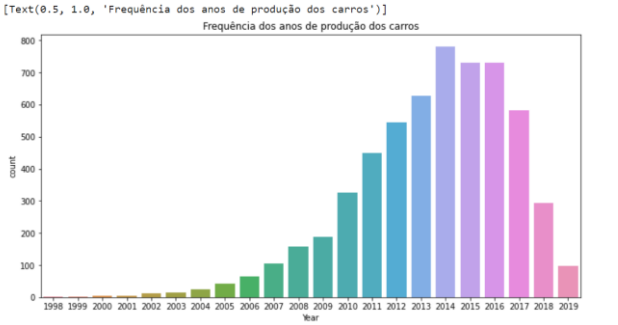
Pelos dois gráficos anteriores é possível identificar outliers, pelo que se procedeu à sua remoção uma vez que podem enviesar a amostra. Foram eliminados carros que custassem mais de 125 000, que correspondia a dois elementos.

2) Year

Os carros da amostra foram lançados no mercado entre 1998 e 2019, sendo que há mais carros de 2014 do que doutros anos.

```
[49] # Check number of cars from x year

     plt.figure(figsize=(12, 6))
     sns.countplot(x="Year", data=train).set(title="Frequência dos anos de produção dos carros")
```

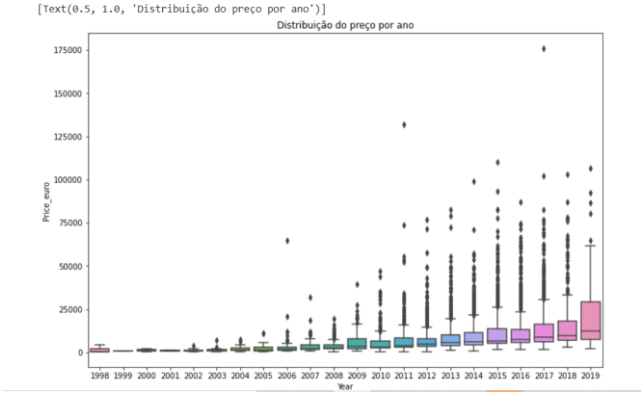
```
[Text(0.5, 1.0, 'Frequência dos anos de produção dos carros')]
```



Pelos boxplots, é possível visualizar a distribuição do preço por ano. Os carros mais recentes parecem ter valores mais elevados. De acordo com o coeficiente de correlação de Spearman, existe uma correlação positiva moderada e significativa (r=0.47, p-value=0.0).

```
[133] # Distribution of price per year

      plt.figure(figsize=(12, 8))
      sns.boxplot(x="Year", y="Price_euro", data=train).set(title="Distribuição do preço por ano")
```

```
[Text(0.5, 1.0, 'Distribuição do preço por ano')]
```

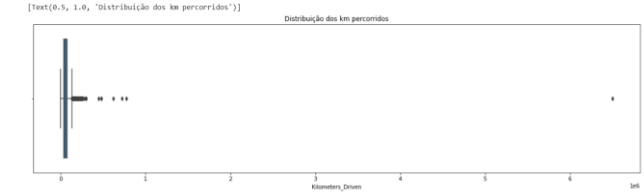


3) *Kilometrers_Driven*

A distribuição dos quilómetros percorridos apresenta um outlier na extrema direita, ou seja, um dos carros possui um valor demasiado elevado, discrepante relativamente ao resto da amostra. É possível visualizar alguns outliers no boxplot seguinte:

```
[65] plt.figure(figsize=(20, 5))
     sns.boxplot(x="Kilometers_Driven", data=train).set(title="Distribuição dos km percorridos")
```

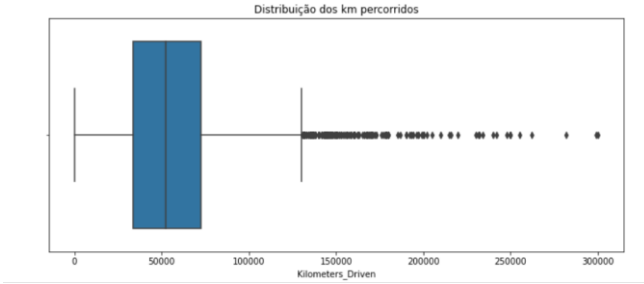
```
[Text(0.5, 1.0, 'Distribuição dos km percorridos')]
```



O 1º quartil é 33 420 km e o 3º quartil corresponde a 72 467 km. Contudo, o outlier mais severo tem o valor de 6 500 000 km. Como estes valores podem conduzir ao enviesamento deverá ser eliminado. Para tal, foram eliminados todos os carros que tivessem mais do que 400 000 km percorridos. Foi criado um novo boxplot. Existe uma assimetria positiva, indicando que existem mais carros com poucos km percorridos. Portanto, os carros com muitos km percorridos são mais escassos.

```
[37] # Drop the outlier

     train.drop(outlier, inplace = True)

     # Reset index

     train = train.reset_index(drop=True)

     # Verify the km driven distribution

     plt.figure(figsize=(12, 5))
     sns.boxplot(x="Kilometers_Driven", data=train).set(title="Distribuição dos km percorridos")
```

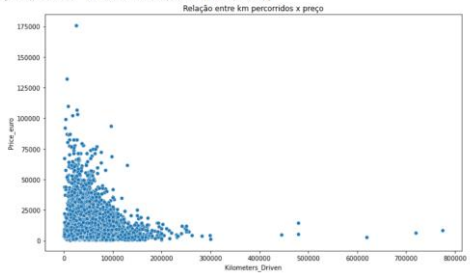
```
[Text(0.5, 1.0, 'Distribuição dos km percorridos')]
```



Ao visualizar o gráfico de dispersão da relação entre os km percorridos e o preço, parece não haver relação entre as duas variáveis. De facto, o coeficiente de correlação de Pearson, apesar de significativo (p-value ≈ 0), apresenta uma correlação negativa fraca (r = -0.16)

```
# Relation between price and km driven

plt.figure(figsize=(12, 7))
sns.scatterplot(x="Kilometers_Driven", y="Price_euro", data=train).set(title="Relação entre km percorridos x preço")
```

```
[Text(0.5, 1.0, 'Relação entre km percorridos x preço')]
```

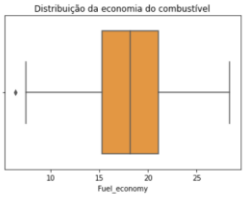


4) *Fuel_economy*

A economia do combustível apresenta uma distribuição quase simétrica, tendo média de 18.39 kmpl e mediana de 18.20 kmpl.

```
[144] # Fuel economy distribution

      sns.boxplot(x="Fuel_economy", data=train, palette="YlOrBr").set(title="Distribuição da economia do combustível")
```

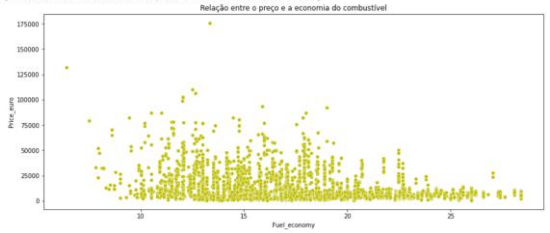
```
[Text(0.5, 1.0, 'Distribuição da economia do combustível')]
```



Esta variável também parece não ter relação com a variável preço uma vez que os valores estão dispersos no gráfico de dispersão. O coeficiente de correlação de Pearson indica que a correlação é fraca e negativa (r=-0.34) mas significativa (p-value ≈ 0)

```
# Relation between price and fuel economy

plt.figure(figsize=(15, 6))
sns.scatterplot(x="Fuel_economy", y="Price_euro", data=train, color="y").set(title="Relação entre o preço e a economia do combustível")
```

```
[Text(0.5, 1.0, 'Relação entre o preço e a economia do combustível')]
```

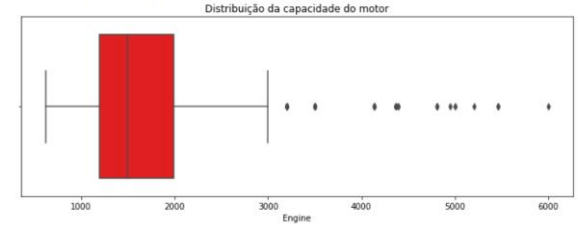


5) *Engine*

A distribuição da capacidade do motor também apresenta assimetria positiva. Portanto, nesta amostra é mais frequente carros com um motor com menor capacidade. Cerca de 75% dos carros têm motor com menos de 1 991 cc, sendo a média de 1 628 cc.

```
# Engine capacity distribution

plt.figure(figsize=(12, 4))
sns.boxplot(x="Engine", data=train, color="r").set(title="Distribuição da capacidade do motor")
```

```
[Text(0.5, 1.0, 'Distribuição da capacidade do motor')]
```



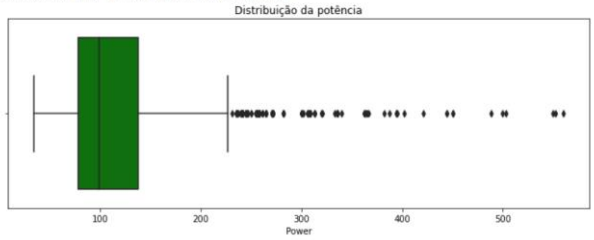
Pelo gráfico de dispersão é difícil perceber se existe relação entre as variáveis. Recorrendo ao coeficiente de correlação de Pearson, verifica-se uma correlação positiva moderada (r=0.66) e significativa (p-value = 0.0).

```
[146] plt.figure(figsize=(12, 5))
      sns.scatterplot(x="Engine", y="Price_euro", data=train, color="r").set(title="Relação entre o preço e a capacidade do motor")

      [Text(0.5, 1.0, 'Relação entre o preço e a capacidade do motor')]
```

6) Power

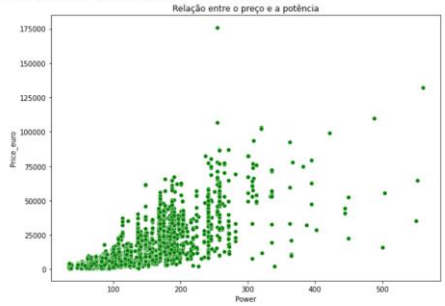
Novamente, é identificada uma distribuição assimétrica positiva.

```
[152] # Variable Power distribution

      plt.figure(figsize=(12, 4))
      sns.boxplot(x="Power", data=train, color="g").set(title="Distribuição da potência")

      [Text(0.5, 1.0, 'Distribuição da potência')]
```



Esta variável parece ter uma relação com o preço, segundo o gráfico de dispersão. O coeficiente de correlação de Pearson indica uma correlação positiva forte (r=0.77) e significativa (p-value=0.0). Pode-se dizer que quanto maior a potência, maior o preço do carro.

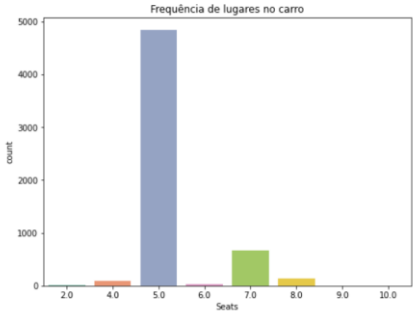
```
[153] plt.figure(figsize=(10, 7))
      sns.scatterplot(x="Power", y="Price_euro", data=train, color="g").set(title="Relação entre o preço e a potência")

      [Text(0.5, 1.0, 'Relação entre o preço e a potência')]
```



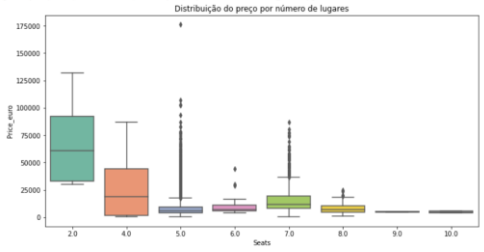
7) Seats

Relativamente ao número de lugares que cada carro tem, maior parte possui 5 lugares. Estes carros apresentam um preço médio de 9 417 € enquanto que o carros com 2 lugares custam em média 67 471 €.

```
# Frequency of seats

plt.figure(figsize=(8, 6))
sns.countplot(x="Seats", data=train, palette="Set2").set(title="Frequência de lugares no carro")

[Text(0.5, 1.0, 'Frequência de lugares no carro')]
```




```
[167] plt.figure(figsize=(12, 6))
      sns.boxplot(x=train["Seats"],y=train["Price_euro"], palette="Set2").set(title='Distribuição do preço por número de lugares')

      [Text(0.5, 1.0, 'Distribuição do preço por número de lugares')]
```



O coeficiente de correlação de Spearman mostra uma correlação positiva e significativa entre o preço e o número de lugares, mas fraca (r=0.22, p-value ≈ 0.0).

8) Location

De acordo com a função *.value_counts()*, a base de dados inclui 11 cidades diferentes, sendo que Mumbai é onde estão a ser vendidos o maior número de carros.

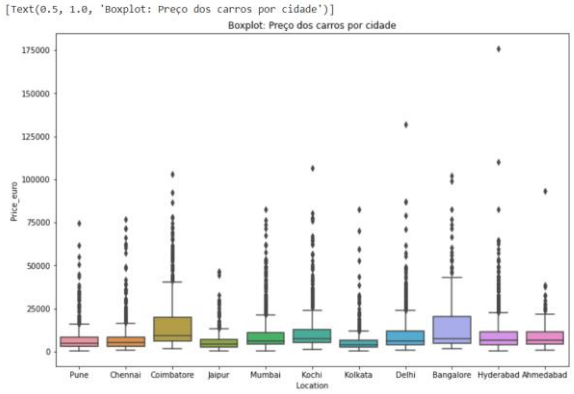
```
# Number of cars to be sold in certain city

train["Location"].value_counts()
```

```
Mumbai        755
Hyderabad     704
Kochi         635
Coimbatore    626
Pune          577
Delhi         533
Kolkata       519
Chennai       473
Jaipur        399
Bangalore     343
Ahmedabad     215
Name: Location, dtype: int64
```

Ao produzir um boxplot para cada cidade em função do preço em euros, pode-se assumir que as medianas de cada cidade não sejam muito distintas. Em todas as cidades é possível identificar uma assimetria positiva, com outliers nos preços elevados.

```
plt.figure(figsize=(12, 8))
sns.boxplot(x='Location', y='Price_euro', data=train).set(title='Boxplot: Preço dos carros por cidade')
```

```
[Text(0.5, 1.0, 'Boxplot: Preço dos carros por cidade')]
```



As médias de preço em euro por cidade variam desde os 16 498 € e os 6 276 €, havendo assim uma diferença de 10 000 €. Contudo, a média é uma medida de tendência central muito afetada pelos outliers e pela figura anterior é visível a presença dos mesmos nesta amostra de treino.

```
lop=train[["Location","Price_euro"]]
mean_location = lop.groupby(["Location"], as_index = False).mean()

mean_location
```

	Location	Price_euro
0	Ahmedabad	9243.632558
1	Bangalore	14845.734694
2	Chennai	8714.930233
3	Coimbatore	16498.506390
4	Delhi	10900.525328
5	Hyderabad	10994.406250
6	Jaipur	6515.060150
7	Kochi	12423.209449
8	Kolkata	6276.421965
9	Mumbai	10491.552318
10	Pune	7579.114385

Resta transformar esta variável em numérica. Para tal, será utilizado a função *.get_dummies*, uma vez que esta permite a definição de nomes para as colunas. A função *OneHotEncoder* teria como output colunas denominadas com numeração. A correta denominação das colunas é importante dado que haverão mais variáveis a serem convertidas para numéricas.

```
[206] # Encode variable "Location" in train data

Location = train[["Location"]]
Location = pd.get_dummies(Location,drop_first=False)
Location=Location.astype(int)

# Merge with main dataframe

train=train.join(Location)
train
```

Location_Ahmedabad	Location_Bangalore	Location_Chennai	Location_Coimbatore	Location_Delhi	Location_Hyderabad
0	0	0	0	0	0
0	0	1	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	0	0

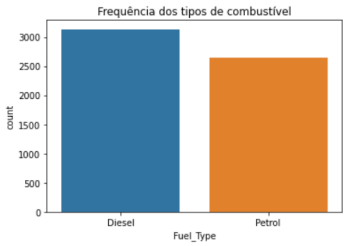
A função *.get_dummies* cria colunas para cada valor da variável. Neste caso, designou-se 1 para os carros de cidade x e 0 para os carros que não estivessem a ser vendidos nessa cidade.

9) *Fuel_Type*

Os tipos de combustível dos carros da amostra são: Diesel e Gasolina. No ficheiro de treino, mais de metade dos carros funcionam a Diesel, 3134 em comparação com 2645 que usam gasolina.

```
[27] # Fuel type distribution

sns.countplot(x='Fuel_Type', data=train).set(title='Frequência dos tipos de combustível')
```

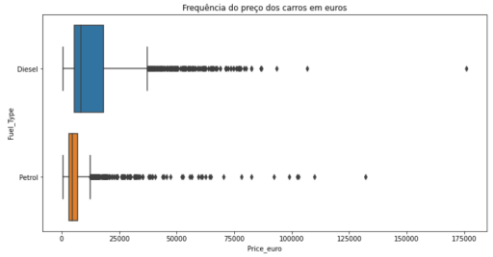
```
[Text(0.5, 1.0, 'Frequência dos tipos de combustível')]
```



Relativamente aos preços dos carros tendo em conta o tipo de combustível, os dados demonstram que os carros a diesel tendem a ser mais caros. A média de preço de um carro a diesel é de 14 144 € em contraste com 6 348 €, o preço médio de um carro a gasolina.

```
[30] # Price distribution per fuel type

plt.figure(figsize=(12, 6))
sns.boxplot(x=train["Price_euro"],y=train["Fuel_Type"], orient="h").set(title='Frequência do preço dos carros em euros')
```

```
[Text(0.5, 1.0, 'Frequência do preço dos carros em euros')]
```




```
ftp=train[["Fuel_Type","Price_euro"]]
price_fuel = ftp.groupby(["Fuel_Type"], as_index = False).describe()

price_fuel
```

	Price_euro							
	count	mean	std	min	25%	50%	75%	max
0	3134.0	14144.143267	14083.696226	660.0	5390.0	8525.0	18150.0	176000.0
1	2645.0	6348.572023	8090.596983	484.0	3080.0	4675.0	6820.0	132000.0

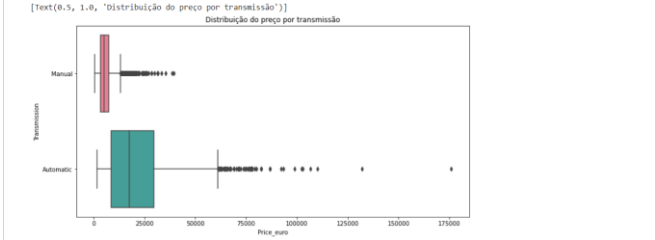
A variável Fuel_Type sofreu o mesmo processo que a variável anterior, de modo a ser convertida em numérica.

10) *Transmission*

Esta variável contém dois valores: manual e automático. A maioria dos carros desta amostra são manuais (n=4 098), sendo apenas 1 681 automáticos. Contudo, pelo boxplot, os carros automáticos tendem a ser mais caros. A distribuição do preço por transmissão é assimétrica positiva. Em média, um carro automático custa 21 789 €, chegando a 176 000 €. Já os manuais, 75% destes custam menos de 7 381 €.

```
# Price distribution per transmission

plt.figure(figsize=(12, 6))
sns.boxplot(x=train["Price_euro"],y=train["Transmission"], orient="h", palette="husl").set(title='Distribuição do preço por transmissão')
```

```
[Text(0.5, 1.0, 'Distribuição do preço por transmissão')]
```




```
# Descriptive analysis of price per transmission

tpp=train[["Transmission","Price_euro"]]
transm = tpp.groupby(["Transmission"], as_index = False).describe()

transm
```

	Price_euro							
	count	mean	std	min	25%	50%	75%	max
0	1681.0	21789.436050	17624.474967	1650.0	8690.0	17600.0	29700.0	176000.0
1	4098.0	5976.494876	3882.630750	484.0	3410.0	5170.0	7381.0	39402.0

Esta variável também foi convertida com recurso à função *.get_dummies()*.

11) Owner_Type

O tipo de dono foi convertido para valores inteiros correspondentes ao número de donos que o carro teve, uma vez que é uma variável qualitativa ordinal. Por exemplo, "segundo" implica que tenha tido 2 donos, logo será substituído por valor 2.

```
# Replace string for integers - represents number of owners

train.replace({"First":1,"Second":2,"Third": 3,"Fourth & Above":4},inplace=True)
train.head()
```

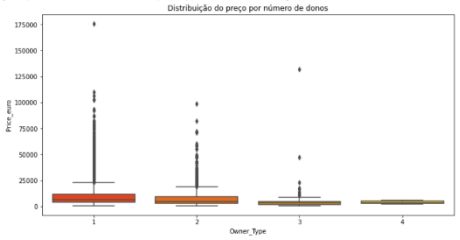
	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Fuel_economy
0	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	1	19.67
1	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	1	18.20
2	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	1	20.77
3	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	2	15.20
4	Nissan Micra Diesel XV	Jaipur	2013	86999	Diesel	Manual	1	23.08

Ao aplicar *.value_counts()* é possível identificar quantos carros tiveram x donos: 4765 carros tiveram apenas um dono, 906 carros tiveram dois donos, 101 tiveram 3 donos e 7 carros tiveram quatro donos.

Relativamente ao preço, pelo boxplot percebe-se que os carros que tiveram apenas um dono apresentam preços mais elevados. Pela tabela do preço em função do número de donos é possível verificar que os preços médios diminuem consoante se aumenta o número de donos.

```
[247] # Price distribution per number of owners

    plt.figure(figsize=(12, 6))
    sns.boxplot(x=train["Owner_Type"],y=train["Price_euro"], palette="autumn").set(title='Distribuição do preço por número de donos')
```

```
[Text(0.5, 1.0, 'Distribuição do preço por número de donos')]
```




```
# Descriptive analysis of price per number of owners

ppo=train[["Owner_Type","Price_euro"]]
owner=ppo.groupby(["Owner_Type"], as_index = True).describe()

owner
```

	Price_euro							
	count	mean	std	min	25%	50%	75%	max
Owner_Type								
1	4764.0	11060.437657	12551.759410	550.0	4235.00	6600.0	11825.0	176000.0
2	906.0	8543.102649	10547.816899	484.0	2763.75	4950.0	9350.0	99000.0
3	101.0	5829.455446	13934.823912	495.0	1650.00	3245.0	4950.0	132000.0
4	7.0	3923.857143	1506.527733	2167.0	2832.50	3575.0	5005.0	6050.0

V. RESULTADOS

A. *Regressão Linear*

Os dados foram introduzidos no modelo de regressão. O objetivo desta técnica estatística é prever o valor da variável dependente com recursos às variáveis independentes. Neste caso, a variável explicada é o preço e as variáveis explicativas são: localização, ano de produção, total de km percorridos, tipo de combustível, tipo de transmissão, número de donos, economia do combustível, capacidade/cilindrada do motor, potência do carro e número. É de notar que as variáveis independentes foram estandardizadas através do método *StandardScaler()* e o *.fit_transform()*.

O coeficiente de determinação é a proporção da variância na variável dependente explicada pelos fatores. Este indicador é suscetível ao número de fatores, tornando necessário recorrer ao coeficiente de determinação ajustado. No modelo em questão, o R^2 ajustado é de 72.40%. Ou seja, 72.40% da variância do preço é explicada pelos fatores presente no modelo.

O teste F testa se existe pelo menos um coeficiente que seja significativamente diferente de zero, ou seja, testa a significância global do modelo. Os resultados indicam um p-value = 0.0, pelo que rejeito a hipótese nula em que os coeficientes são iguais a zero. Os testes de significância individual (teste t) indica que todos os fatores são significativos, excepto "owner_type" (p-value=0.504) e "engine" (p-value=0.074). Portanto, o número de donos e o motor parecem não ser relevantes para explicar o preço dos carros usados.

OLS Regression Results

Dep. Variable:	Price_euro	R-squared:	0.725
Model:	OLS	Adj. R-squared:	0.724
Method:	Least Squares	F-statistic:	798.1
Date:	Mon, 10 May 2021	Prob (F-statistic):	0.00
Time:	11:18:41	Log-Likelihood:	-58675.
No. Observations:	5770	AIC:	1.174e+05
Df Residuals:	5750	BIC:	1.175e+05
Df Model:	19		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-8.545e+05	3.55e+04	-24.090	0.000	-9.24e+05	-7.85e+05
Year	889.5657	36.947	24.077	0.000	817.136	961.995
Kilometers_Driven	-0.0421	0.003	-12.642	0.000	-0.049	-0.036
Owner_Type	-195.7269	208.045	-0.941	0.347	-603.574	212.120
Fuel_economy	-198.8641	37.163	-5.351	0.000	-271.718	-126.010
Engine	0.7494	0.424	1.767	0.077	-0.082	1.581
Power	137.0109	4.109	33.343	0.000	128.956	145.066
Seats	-744.0231	145.060	-5.129	0.000	-1028.396	-459.650
Location_Ahmedabad	-7.846e+04	3235.138	-24.251	0.000	-8.48e+04	-7.21e+04
Location_Bangalore	-7.631e+04	3235.928	-23.581	0.000	-8.27e+04	-7e+04
Location_Chennai	-7.694e+04	3224.067	-23.863	0.000	-8.33e+04	-7.06e+04
Location_Coimbatore	-7.593e+04	3287.023	-23.098	0.000	-8.24e+04	-6.95e+04
Location_Delhi	-7.861e+04	3233.031	-24.315	0.000	-8.5e+04	-7.23e+04
Location_Hyderabad	-7.604e+04	3220.012	-23.614	0.000	-8.24e+04	-6.97e+04
Location_Jaipur	-7.706e+04	3236.670	-23.808	0.000	-8.34e+04	-7.07e+04
Location_Kochi	-7.822e+04	3285.016	-23.812	0.000	-8.47e+04	-7.18e+04
Location_Kolkata	-7.99e+04	3195.518	-25.002	0.000	-8.62e+04	-7.36e+04
Location_Mumbai	-7.947e+04	3221.646	-24.668	0.000	-8.58e+04	-7.32e+04
Location_Pune	-7.762e+04	3235.098	-23.992	0.000	-8.4e+04	-7.13e+04
Fuel_Type_Diesel	-4.257e+05	1.77e+04	-23.986	0.000	-4.6e+05	-3.91e+05
Fuel_Type_Petrol	-4.289e+05	1.77e+04	-24.193	0.000	-4.64e+05	-3.94e+05
Transmission_Automatic	-4.26e+05	1.77e+04	-24.003	0.000	-4.61e+05	-3.91e+05
Transmission_Manual	-4.286e+05	1.77e+04	-24.176	0.000	-4.63e+05	-3.94e+05

Omnibus:	2940.293	Durbin-Watson:	2.020
Prob(Omnibus):	0.000	Jarque-Bera (JB):	55200.911
Skew:	1.999	Prob(JB):	0.00
Kurtosis:	17.616	Cond. No.	9.05e+20

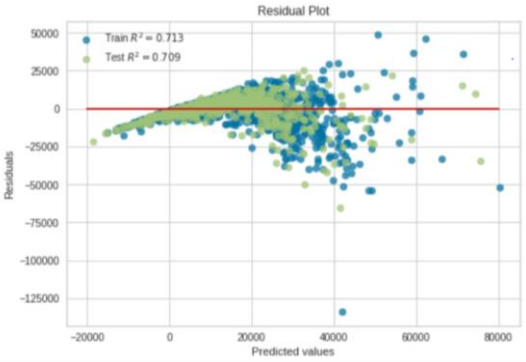
O modelo parece bom, contudo é necessário perceber se os dados preenchem os pressupostos que garantem a adequação da regressão linear aos dados:

1. Distribuição normal dos erros;
2. Homogeneidade da variância dos erros (homocedasticidade);
3. Independência dos erros.

O teste de Omnibus testa a hipótese nula de que a distribuição dos erros é normal. Como o valor de prob(omibus), presente na tabela, é praticamente zero, então os erros não estão normalmente distribuídos. Surge assim uma violação do pressuposto de distribuição normal dos erros. Este teste é corroborado pelo teste de Jarque-Bera que afirma que não há distribuição normal.

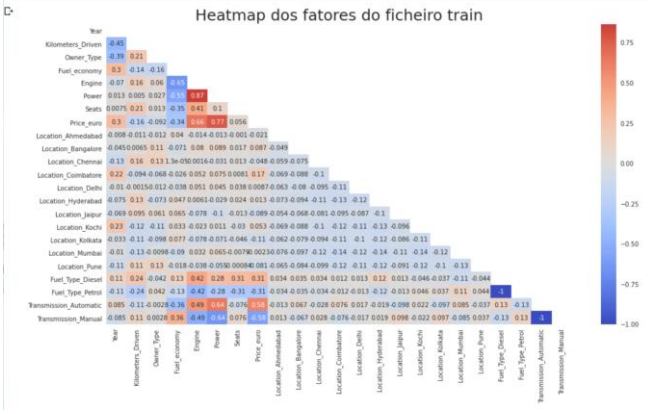
O teste de Durbin-Watson é utilizado para detectar a presença de autocorrelação nos resíduos. Os valores variam entre 0 e 4. A tabela indica um valor de 2.02, pelo que para haver independência dos erros.

Para verificar a homocedasticidade, foi criado um gráfico de dispersão entre os resíduos e os valores previstos. Até y=20000, os resíduos não estão dispersos aleatoriamente, sugerindo que a regressão linear não é adequada para os dados.



O *condition number* mede a sensibilidade do output da função relativamente ao seu input. Ou seja, avalia a existência de multicolinearidade. O seu valor deve ser menor do que 30. Neste caso, é bem superior, sugerindo a existência de multicolinearidade. Ao analisar o heatmap, concluímos a existência de multicolinearidade por existir correlações entre as variáveis independentes. Por exemplo, "power" e "engine" possuem uma correlação forte (r=0.87).

```
# Correlation between variables

plt.style.use('ggplot')
sns.set_style('whitegrid')
plt.subplots(figsize = (17,9))

mask = np.zeros_like(train.corr(), dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(train.corr(), cmap="coolwarm", annot=True, mask=mask, center = 0 );
plt.title("Heatmap dos fatores do ficheiro train", fontsize = 25);
```



Como suma, a regressão linear não parece ser a técnica adequada para prever o preço dos carros usados com base nos dados destes ficheiros. A exclusão da regressão linear também pode ser corroborada pela precisão dos resultados obtidos através da função *linear_reg.score()*. A precisão foi de aproximadamente 70% tanto no set de treino como no de teste.

```
[197] # Linear regression model and its accuracy

      from sklearn.linear_model import LinearRegression
      linear_reg = LinearRegression()
      linear_reg.fit(X_train, y_train)
      y_pred= linear_reg.predict(X_test)
      print("Accuracy on Training set: ",linear_reg.score(X_train,y_train))
      print("Accuracy on Testing set: ",linear_reg.score(X_test,y_test))

      Accuracy on Training set:  0.713219952985974
      Accuracy on Testing set:  0.7090562899721569
```

B. *Random Forest Regressor*

A variável explicada (y) é o preço e as variáveis explicativas (x_i) são: localização, ano de produção, total de km percorridos, tipo de combustível, tipo de transmissão, número de donos, economia do combustível, capacidade/cilindrada do motor, potência do carro e número de lugares.

A eficácia do modelo é de 98.5% no ficheiro de treino e 89.3% no ficheiro de teste.

```
[203] # Random Forest Regressor

      from sklearn.ensemble import RandomForestRegressor

      rf_reg = RandomForestRegressor()
      rf_reg.fit(X_train, y_train)
      y_pred= rf_reg.predict(X_test)

      print("Accuracy on Training set: ",rf_reg.score(X_train,y_train))
      print("Accuracy on Testing set: ",rf_reg.score(X_test,y_test))

      Accuracy on Training set:  0.9849643037533619
      Accuracy on Testing set:  0.8931667876053809
```

Agora, se definirmos o número de estimadores para 20, a precisão do modelo diminui, tal como demonstrado a seguir:

```
# Random Forest Regressor

from sklearn.ensemble import RandomForestRegressor

rf_reg = RandomForestRegressor(n_estimators=20, random_state=0)
rf_reg.fit(X_train, y_train)
y_pred= rf_reg.predict(X_test)

print("Accuracy on Training set: ",rf_reg.score(X_train,y_train))
print("Accuracy on Testing set: ",rf_reg.score(X_test,y_test))
```

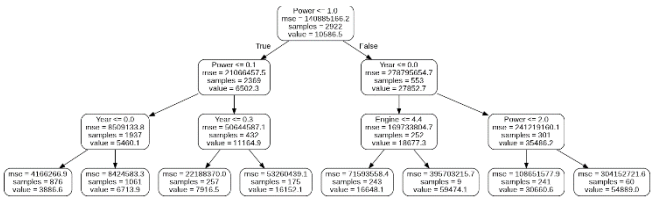
```
Accuracy on Training set:  0.9794161283630028
Accuracy on Testing set:  0.8879934931436785
```

Se este parâmetro for aumentado para 200 árvores de decisão, o tempo de resolução é maior, mas permite um aumento da precisão, pelo menos no ficheiro de teste.

```
# Random Forest Regressor

from sklearn.ensemble import RandomForestRegressor

rf_reg = RandomForestRegressor(n_estimators=200, random_state=0)
rf_reg.fit(X_train, y_train)
y_pred= rf_reg.predict(X_test)

print("Accuracy on Training set: ",rf_reg.score(X_train,y_train))
print("Accuracy on Testing set: ",rf_reg.score(X_test,y_test))
```

```
Accuracy on Training set:  0.9837220858173825
Accuracy on Testing set:  0.8964956272730048
```

Foi possível extrair uma árvore de decisão com 3 níveis. A interpretação é simples: se power for menor que dado valor, segue-se para o nó do lado esquerdo (true) e assim sucessivamente até ao fim, onde *value* representa o preço do carro. É de notar que as variáveis independentes foram estandardizadas.



O RMSE (Root Mean Squared Error) é uma boa medida para comparar os modelos de previsão, para além do R^2. Este serve para estimar o desvio padrão do valor observado do valor previsto pelo modelo. Ao comparar o RMSE dos dois modelos, é possível identificar que o Random Forest Regressor é um melhor modelo, comparativamente à regressão linear, para estes dados, uma vez que o RMSE é menor.

```
[79] # Linear regression metrics

     from sklearn import metrics

     print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
     print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
     print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 4231.280772917813
Mean Squared Error: 47743949.97355113
Root Mean Squared Error: 6909.699702125348
```

```
# Random forest regressor metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 1774.061050986571
Mean Squared Error: 18007066.4699665
Root Mean Squared Error: 4243.47339687272
```

VI. Conclusão

A regressão linear é um método robusto para prever os valores da variável dependente com base nos fatores. Contudo, a aplicabilidade deste método é limitada pelos requisitos que este impõe. Os dados desta amostra violam tais requisitos, pelo que a regressão linear não se revela o melhor método para ser aplicado nestes dados.

Todavia, existem outros métodos para o mesmo efeito que não requerem uma relação linear entre as variáveis. O *Random Forest Regression* demonstrou uma maior precisão em prever o preço dos carros usados com as caraterísticas disponíveis (98.5% nos dados de treino e 89.3% nos dados de teste). Assim, se um sujeito pretender prever o preço de um carro usado que queira vender ou um comerciante que pretenda saber por quanto deve vender no seu stand, sugiro o uso da *Random Forest Regression*. Contudo, alerto para o facto de que a amostra em si possa ter peso na escolha do melhor método a utilizar.

Para trabalho futuro e para uma maior precisão do modelo, é necessário a adição de mais caraterísticas relevantes tais como a marca, o modelo, a cor do carro, entre outras. E mais caraterísticas não bastam, uma vez que é necessário haver um grande número de carros com essas caraterísticas. Ou seja, um carro de cada modelo não permite avaliar o impacto no modelo no preço. Seria também interessante utilizar outros modelos de previsão e comparar os seus resultados.

Referências

[1] A. Gavazza, A. Lizzeri e N. Roketskiy, "A quantitative analysis of the used-car market," *American Economic Review*, vol. 104, nº 11, pp. 3668-3700, 2014.

[2] C. Chen, L. Hao e C. Xu, "Comparative analysis of used car price evaluation models," *AIP Conference Proceedings*, vol. 1893, nº 1, p. 020165, 2017.

[3] A. Fathalla, A. Salah, K. Li, K. Li e P. Francesco, "Deep end-to-end learning for price prediction of second-hand items," *Knowledge and Information Systems*, vol. 62, nº 12, pp. 4541-4568, 2020.

[4] S. Kuiper, "Introduction to Multiple Regression: How Much Is Your Car Worth?," *Journal of Statistics Education*, vol. 16, nº 3, 2008.

[5] M. Listiani, "Support Vector Regression Analysis for Price Prediction in a Car Leasing Application," *Dissertação de Mestrado, Hamburg University of Technology*, 2009.

[6] K. Noor e S. Jan, "Vehicle price prediction system using machine learning techniques," *International Journal of Computer Applications*, vol. 167, nº 9, pp. 27-31, 2017.

[7] J. Marôco, Análise Estatísitica com o SPSS Statistics, 6ª ed., Report Number, 2014.

[8] M. Segal, "Machine learning benchmarks and random forest regression.," em *Machine Learning*, 2004.

[9] P. Gupta, P. Kumar e N. Singh, "Comparative analysis of car sales using supervised algorithms," *Advances and Applications in Mathematical Sciences*, vol. 20, nº 3, pp. 367-375, 2021.

[10] S. Kiran, "Prediction of Resale Value of the Car Using Linear Regression Algorithm," *International Journal of Innovative Science and Research Technology*, vol. 5, nº 7, pp. 382-386, 2020.

[11] N. Monburinon, P. Chertchom, T. Kaewkiriya, S. Rungpheung, S. Buya e P. Boonpou, "Prediction of prices for used car by using regression models," *5th International Conference on Business and Industrial Research (ICBIR)*, pp. 115-119, 2018.

[12] "Kaggle," [Online]. Available: https://www.kaggle.com/avikasliwal/used-cars-price-prediction. [Acedido em 13 abril 2021].

[13] "Scikit-Learn: Machine learning in Python," sklearn.preprocessing.LabelEncoder, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html. [Acedido em 05 maio 2021].

[14] "Scikit-Learn: Machine learning in Python," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html. [Acedido em 05 maio 2021].