

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

André Melato Pereira

Machine Learning para previsão de preço de carros

Belo Horizonte
2023

André Melato Pereira

MACHINE LEARNING PARA PREVISÃO DE PREÇO DE CARROS

Trabalho de Conclusão de Curso apresentado
ao Curso de Especialização em Ciência de
Dados e Big Data como requisito parcial à
obtenção do título de especialista.

Belo Horizonte
2023

SUMÁRIO

1. Introdução.....	5
1.1. Contextualização	5
1.2. O problema proposto	5
1.3. Objetivos.....	5
2. Coleta de Dados.....	9
3. Processamento/Tratamento de Dados	8
3.1 Tratamento do dataset preço de carro	8
3.1.1 Alteração do tipo do dado	8
3.1.2 Renomeação das colunas	11
3.2 Tratamento do dataset dólar	11
3.3 Merge das bases	12
3.4 Remoção de dados duplicados	12
3.5 Tratamento de dados nulos.....	13
3.5.1 Tratamento do Levy.....	13
3.5.2 Tratamento do Last_price	16
3.6 Tratamento outliers.....	17
3.6.1 Variável target	17
3.6.2 Análise multivariada	20
3.6.2.1 Criação de Dummies.....	20
3.6.2.2 KNN	21
4. Análise e Exploração dos Dados	24
4.1 Análise de features	24
4.2 Seleção de features	27
4.3 Análise multicolinearidade	28
5. Criação de Modelos de Machine Learning	28
5.1 Pre-processamento	29
5.2 Separação dos dados de treinamento e teste.....	30
5.3 Modelo.....	30
5.3.1 Random Forest Regression.....	30

5.3.2 ElasticNet Regression	33
5.3.3 XGBRegressor	35
6. Interpretação dos Resultados	38
6.1 Pré Modelagem	38
6.2 Modelos	38
7. Links.....	40
APÊNDICE.....	41

1. Introdução

1.1. Contextualização

Tendo em vista o fenômeno da globalização que ajudou a acelerar e facilitar meios produtivos mediante a facilitação logística e reduções de custo produtivos a mesma atualmente se mostrou como risco visto que a alta integração entre os mercados gera uma alta dependência susceptíveis a catástrofes locais que afetam diretamente tais malhas produtivas.

Nesse contexto, o mercado automobilístico tem vivenciado os impactos gerados pela pandemia e a escassez de produtos gerados pela crise logística que por sua vez tem impactado diretamente a indústria. Os altos preços de venda de carros base juntamente ao acréscimo de juros no Brasil e a alta do dólar tem afastado os consumidores reduzindo as vendas de novos carros no mercado interno, em consequência de tal cenário automobilísticas estão adotando medidas de paralizações de produção, férias coletivas e demissões para manter suas operações economicamente saudáveis no país.

Segundo a Fenabrave (Federação Nacional da Distribuição de Veículos Automotores) houve uma redução de 8,20% no número de emplacamentos de carros em fevereiro de 2023 em comparação com o mês anterior, o que por sua vez tem decepcionado o otimismo das montadoras no começo desse semestre.

1.2. O problema proposto

Conforme o contexto advindo do item anterior, tal projeto tem como foco minimizar os impactos que as montadoras automobilísticas vêm tendo nos dias atuais mediante a previsibilidade dos valores de venda dos carros a partir dos componentes dos carros.

Tal problema se torna relevante pelo fato da alta visibilidade dos preços de venda antes mesmo de serem produzidos ajudarem economicamente as montadoras nas estratégias de viabilização de veículos balanceando os custos de produção e a margem desejada na venda final minimizando impactos negativos de vendas, demissões e paralizações por alto volume de produto acabado.

Para a realização dessa análise foram utilizados 2 bases de dados, uma relacionada aos componentes de veículos com histórico de preço advindos da plataforma Kaggle (car_price_prediction.csv) e o histórico do preço do dólar resgatados da plataforma Investing (Índice Dólar Dados Históricos.csv). Para essa análise foram coletados dados de carros com ano de produção variando de 1939 a 2020.

1.3. Objetivos

O projeto vigente tem como objetivo prever o preço do automóvel tendo como base componentes do carro juntamente com o custo de produção, sendo assim tal visibilidade tem como foco auxiliar montadoras a projetarem seus carros objetivando melhor preço no mercado.

2. Coleta de Dados

Para o desenvolvimento do projeto foram utilizadas duas bases de dados principais sendo uma delas com histórico de dados do preço dos carros e as *features* relacionadas e outra com dados históricos mensal do dólar.

O primeiro dataset utilizado foi extraído da plataforma Kaggle (<https://www.kaggle.com/datasets/deepcontractor/car-price-prediction-challenge>) em formato CSV e possui a seguinte estrutura:

Nome da coluna/campo	Descrição	Tipo
ID	ID da tabela.	Int64
Price	Preço em dólar final do carro.	Int64
Levy	Tributo / Imposto sobre o contribuinte.	Object
Manufacturer	Montadora.	Object
Model	Modelo do carro.	Object
Prod. year	Ano em que o carro foi produzido.	Int64
Category	Categoria do carro.	Object
Leather interior	Couro no interior do carro.	Object
Fuel type	Tipo de combustível do carro.	Object
Engine volume	Volume do motor.	Object
Mileage	Km rodado.	Object
Cylinders	Número de cilindros do carro.	Float64
Gear box type	Tipo da caixa de câmbio.	Object
Drive wheels	Tipo de tração do carro.	object
Doors	Número de portas.	Object

Wheel	Posição do motorista.	Object
Color	Cor do carro.	Object
Airbags	Quantidade de airbags.	Int64

Já o segundo dataset foi extraído do site da Investing (<https://br.investing.com/currencies/usd-brl-historical-data>) no qual foi extraído dados mensais do dólar no primeiro momento em modelo CSV na seguinte estrutura:

Nome da coluna/campo	Descrição	Tipo
Data	Data mês.	Object
Último	Valor de fechamento do dólar no período.	Float64
Abertura	Valor de abertura do dólar no período.	Float64
Máxima	Valor máximo do dólar no período.	Float64
Mínima	Valor mínimo do dólar no período.	Float64
Vol.	Volume de transição no período.	Float64
Var%	Variação do dólar no período em %.	Object

3. Processamento/Tratamento de Dados

Tendo como base os dois *datasets* utilizados a primeira fase foi tratá-los separadamente para posteriormente forma uma base única

3.1. Tratamento do *dataset* preço de carro

3.1.1. Alteração do tipo do dado

Como observado no item 2 de coleta de dados, algumas *features* possuem tipo de dados divergente do senso comum, como por exemplo a quilometragem (*feature Mileage*) ser do tipo objeto, sendo assim foi feito num primeiro momento a adequação e tratamento dos tipos de dados para cada *feature* da tabela:

- *Feature Mileage*

```
car_price_df.Mileage.head()

0    186005 km
1    192000 km
2    200000 km
3    168966 km
4     91901 km
Name: Mileage, dtype: object
```

Figura 1 – Campo *Mileage* antes do tratamento

Como observado na figura 1 observa-se que a feature *Mileage* é constituída pelo campo numérico juntamente à string “km”, sendo assim foi feito num primeiro momento a remoção da string “km” e transformou-se o campo em float como mostrado na figura abaixo:

```
car_price_df['Mileage'] = car_price_df['Mileage'].str.replace('km', '').astype(float)

car_price_df.Mileage.head()

0    186005.0
1    192000.0
2    200000.0
3    168966.0
4     91901.0
Name: Mileage, dtype: float64
```

Figura 2 – Campo *Mileage* depois do tratamento

- *Feature Levy*

```
car_price_df.Levy.head()

0    1399
1    1018
2      -
3     862
4     446
Name: Levy, dtype: object
```

Figura 3 – Campo *Levy* antes do tratamento

Como observado na figura 3 o campo *Levy* era considerado “object” pelo fato de haver linhas com hífen (“-”), sendo assim tais casos foram substituídos por “Nan” e por fim o campo foi convertido para float como mostrado abaixo:

```
car_price_df['Levy'] = car_price_df['Levy'].replace({'-':np.nan}).astype(float)

car_price_df.Levy.head()

0    1399.0
1    1018.0
2      NaN
3     862.0
4     446.0
Name: Levy, dtype: float64
```

Figura 4 – Campo *Levy* depois do tratamento

- Engine Volume

```
: car_price_df['Engine volume'].tail()

: 19232    2.0 Turbo
   19233      2.4
   19234      2
   19235      2
   19236      2.4
Name: Engine volume, dtype: object
```

Figura 5 – Campo *Engine Volume* antes do tratamento

No caso da feature *Engine Volume* foi observado a existencia de alguns campo com complemento “Turbo”, sendo assim com o objetivo de acrescentar uma *feature* possivelmente relevante foi criado uma nova feature denominada *Exist_Turbo* booleana transformada a *feature Engine Volume* em *float* como observado abaixo:

```
car_price_df['Exist_Turbo'] = car_price_df['Engine volume'].str.contains('Turbo')

car_price_df['Exist_Turbo'].head()

0    False
1    False
2    False
3    False
4    False
Name: Exist_Turbo, dtype: bool

car_price_df['Engine volume'] = car_price_df['Engine volume'].str.replace(' Turbo', '').astype(float)

car_price_df['Engine volume'].tail()

19232    2.0
19233    2.4
19234    2.0
19235    2.0
19236    2.4
Name: Engine volume, dtype: float64
```

Figura 6 – Campos *Engine Volume* e *Exist_Turbo* após tratamento

- *Engine Volume*

```
car_price_df['Doors'].tail()

19232    02-Mar
19233    04-May
19234    04-May
19235    04-May
19236    04-May
Name: Doors, dtype: object
```

Figura 7 – Campos *Doors* antes do tratamento

Como evidenciado acima pode-se ver que o maior valor do campo *Doors* está em saber o número de portas contido no veículo, sendo assim houve o agrupamento do campo em 3 categorias e transformando o campo no formato *int* como mostrado abaixo:

```
l1 = list(set(car_price_df['Doors']))
l2 = [2,5,4]

replacement_map = {str(i1): int(i2) for i1, i2 in zip(l1, l2)}
car_price_df['Doors'] = car_price_df['Doors'].map(replacement_map)

car_price_df['Doors'].tail()

19232    4
19233    2
19234    2
19235    2
19236    2
Name: Doors, dtype: int64
```

Figura 8 – Campos *Doors* depois do tratamento

3.1.2. Renomeação das colunas

Com o objetivo de facilitar a manuseio do código foi renomeado alguns campos removendo os espaços em branco e substituindo por “_” como observado na figura 8 abaixo:

```
car_price_df.columns = ['ID', 'Price', 'Levy', 'Manufacturer', 'Model', 'Prod_year',
                        'Category', 'Leather_interior', 'Fuel_type', 'Engine_volume', 'Mileage',
                        'Cylinders', 'Gear_box_type', 'Drive_wheels', 'Doors', 'Wheel', 'Color',
                        'Airbags', 'Exist_Turbo']
```

Figura 8 – Renomeação das colunas

3.2. Tratamento do *dataset* dólar

O primeiro tratamento feito na tabela do índice dólar foi a transformação do tipo do campo data em *datetime* como observado abaixo:

```
dxy_df['Data'].head()

0    01.12.2020
1    01.11.2020
2    01.10.2020
3    01.09.2020
4    01.08.2020
Name: Data, dtype: object

dxy_df['date'] = pd.to_datetime(dxy_df['Data'], dayfirst=True)

dxy_df['date'].head()

0    2020-12-01
1    2020-11-01
2    2020-10-01
3    2020-09-01
4    2020-08-01
Name: date, dtype: datetime64[ns]
```

Figura 9 – Transformação da coluna Data

Após a renomeação foi criado a coluna *Year*, essencial para ser chave de ligação com a tabela de preços:

```
dxy_df['Year'] = dxy_df['date'].dt.year

dxy_df['Year'].head()

0    2020
1    2020
2    2020
3    2020
4    2020
```

Figura 9 – Criação da coluna Year

Poserior à criação foi realizada a padronização dos nomes das colunas removendo os espaços em branco e ao mesmo tempo foi transformada a coluna *var_%* em *float*:

```
dxy_df.columns = ['last_price', 'open_price', 'max_price', 'min_price', 'volume', 'var_%', 'date', 'Year']

dxy_df['var_%'] = dxy_df['var_%'].str.replace('%', '')
dxy_df['var_%'] = dxy_df['var_%'].str.replace('.', '.').astype(float)
```

Figura 10 – Renomeação e tratamento da coluna *var_%*

Tendo em vista que na tabela do índice dolar tempo dados mensais e o do preço está em ano de produção, foi realizado o agrupamento do índice por ano através do uso da média como demonstrado abaixo:

```
dxy_year_df = dxy_df.groupby('Year').last_price.mean().reset_index()
```

Figura 11 – Agrupamento do valor dólar pela média

3.3. Merge das bases

Como forma de unificar os dados relevantes de cada tabela, ambas foram unificadas mediante as colunas *Prod_Year* da tabela de preço com a coluna *Year* da tabela do índice:

```
union_df = car_price_df.merge(dxy_year_df, left_on='Prod_year', right_on='Year', how='left')
```

```
union_df.drop(columns='Year', inplace=True)
```

```
union_df.head(2)
```

type	Engine_volume	Mileage	Cylinders	Gear_box_type	Drive_wheels	Doors	Wheel	Color	Airbags	Exist_Turbo	last_price
brid	3.5	186005.0	6.0	Automatic	4x4	2	Left wheel	Silver	12	False	81.3600
etrol	3.0	192000.0	6.0	Tiptronic	4x4	2	Left wheel	Black	8	False	76.1375

Figura 12 – Merge das tabelas

3.4. Remoção de dados duplicados

Foi realizado no *dataset* unificado que os *ID's* estavam duplicados, sendo assim nessa fase foi realizado a remoção dessas linhas duplicadas, ao todo foram constatadas a existência de 313 linhas com *ID* duplicado:

```
print(union_df['ID'].duplicated().sum())
union_df.ID.value_counts().head()
```

```
313
45815365    8
45815361    8
45815363    7
45815368    7
45723475    7
Name: ID, dtype: int64
```

```
union_df.query("ID == 45815361")
```

```
...
```

```
# Shape antes
union_df.shape
```

```
(19237, 20)
```

```
#Shape depois
union_df.drop_duplicates('ID', inplace=True)
union_df.shape
```

```
(18924, 20)
```

Figura 13 – Remoção de ID's duplicados

3.5. Tratamento de dados nulos

Como observado na imagem abaixo foi observado a existência de dados nulos a serem tratados em duas *features* importantes, *Levy* e *last_price*.

```
union_df.isnull().sum()
```

```
ID          0
Price        0
Levy        5819
Manufacturer 0
Model        0
Prod_year    0
Category     0
Leather_interior 0
Fuel_type    0
Engine_volume 0
Mileage      0
Cylinders    0
Gear_box_type 0
Drive_wheels 0
Doors        0
Wheel        0
Color        0
Airbags      0
Exist_Turbo  0
last_price   15
dtype: int64
```

Figura 14 – Colunas com dados nulos

3.5.1 Tratamento do Levy

Tendo em vista que os dados nulos do *Levy* influenciavam em 30,16% das linhas do *dataset* que por sua vez é um número bastante significativo para serem excluídos foi-se adotado dois métodos para serem testados um utilizando o método KNN e outro excluindo todas as linhas.

- KNN

```
# Número de linhas com esse critério nulo chega a 30,16% do DF
```

```
union_df.Levy.isnull().sum()/len(union_df)
```

```
0.30168040583386174
```

Figura 15 – Constatação de perda

Para tal substituição foi observado quais eram os parametros existentes que mais tinha relevancia com o *Levy* através da análise de correlação de *Spearman*:

```
corr = union_df.corr(method='spearman')
plt.figure(figsize = (8,5))
plt.title(f'Correlation | Spearman')
sns.heatmap(corr, annot = True, vmin=-1, vmax=1, annot_kws={"size":6})
plt.show()
```

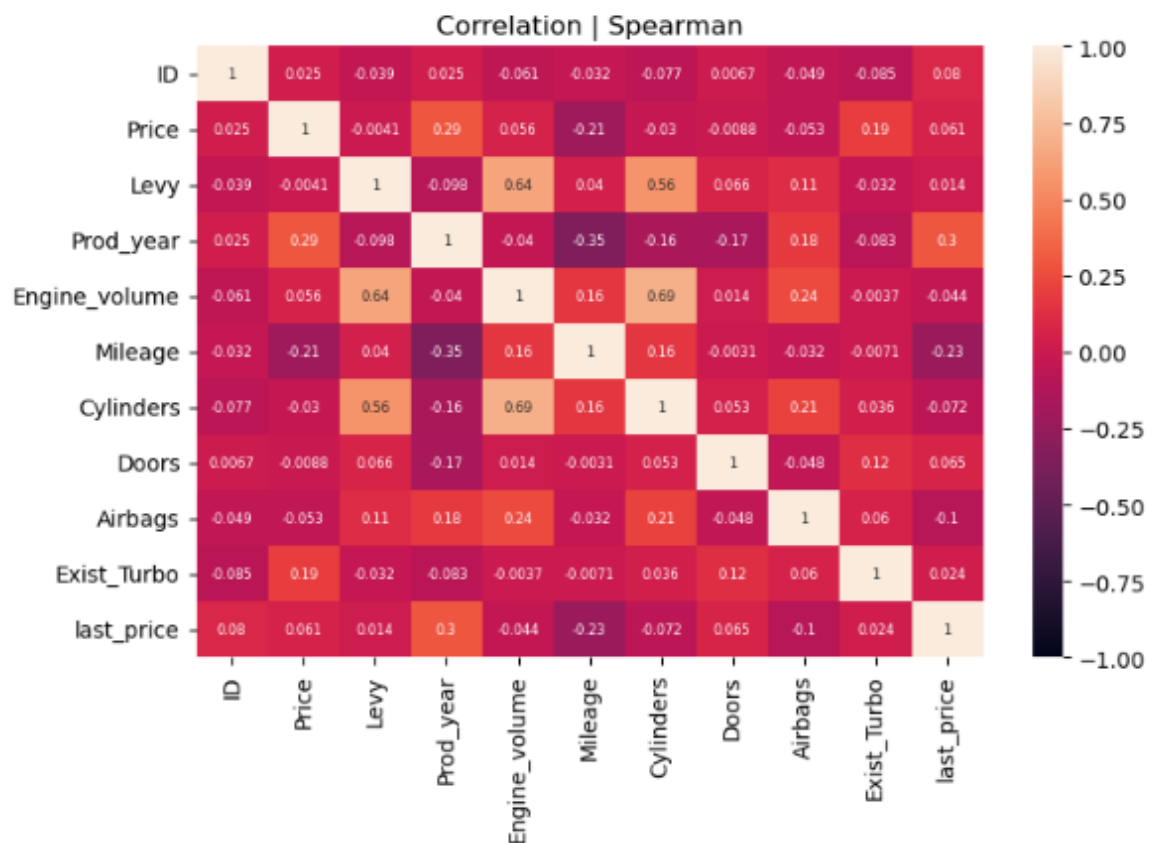


Figura 16 – Correlação spearman

Tendo como base a figura 15 acima pode-se ver que os parametros Engine_volume e Cylinders tem uma boa correlação com Levy, sendo assim tais colunas foram selecionados como critérios relevantes para o preenchimento dos dados nulos do Levy:

```
from sklearn.impute import KNNImputer
from sklearn import metrics

# Aplicação
df_fill_levy = union_df[param].copy()

imputer_KNN = KNNImputer()
df_KNN = imputer_KNN.fit_transform(df_fill_levy)
imputer_KNN_imputed = pd.DataFrame(df_KNN, columns=param)

print("ESTATÍSTICA DF ORIGINAL")
print(f"{df_fill_levy.Levy.describe()} \n")
print("ESTATÍSTICA DF COM PREENCHIMENTO")
print(imputer_KNN_imputed.Levy.describe())

union_df_knn = union_df.copy()

# Substituição da coluna preenchida no DF original
union_df_knn['Levy'] = list(imputer_KNN_imputed['Levy'])
```

Figura 17 – Preenchimento de dados nulos com KNNImputer

Como visto acima foi usado a biblioteca KNN Imputer para a substituição dos dados nulos na coluna Levy e tais resultados gravados num novo dataframe denominado union_df_knn, sendo assim foi realizado uma comparação estatística mostrada abaixo mostrando que tal imputação não revelou grandes distorções na distribuição se comparado ao caso de ter excluído os campos nulos, para isso foi usado a função *describe*:


```

ESTATÍSTICA DF ORIGINAL
count      13215.000000
mean        906.299205
std         463.296871
min          87.000000
25%         640.000000
50%         781.000000
75%        1058.000000
max        11714.000000
Name: Levy, dtype: float64

ESTATÍSTICA DF COM PREENCHIMENTO
count      18924.000000
mean        936.291270
std         479.966682
min          87.000000
25%         691.600000
50%         831.000000
75%        1053.000000
max        11714.000000
Name: Levy, dtype: float64

```

Figura 18 - Constatação de não grande alteração na distribuição

- Excluir todas as linhas nulas

Já para essa segunda abordagem de tratamento de outliers foi usada a função *drop* do pandas e os resultados gravados como outro *dataframe* denominado *union_df_drop* como mostrado abaixo:

```

: union_df_drop = union_df.copy()

: union_df_drop.dropna(axis=0, inplace=True)

```

Figura 19 – Criação do novo dataframe excluindo linhas nulas

3.5.2 Tratamento do *Last_price*

Devido ao pequeno número e dos dados serem de carros muito antigos, foi usado *dropna* para excluir as 15 linhas do *dataframe* tratado pelo método KNN descrito no item anterior e assim finalizando a etapa de substituição de dados nulos:

```
union_df_knn.dropna(axis=0, inplace=True)
```

```
union_df_knn.isnull().sum()
```

```
ID          0
Price        0
Levy         0
Manufacturer 0
Model        0
Prod_year    0
Category     0
Leather_interior 0
Fuel_type    0
Engine_volume 0
Mileage       0
Cylinders    0
Gear_box_type 0
Drive_wheels 0
Doors         0
Wheel         0
Color         0
Airbags       0
Exist_Turbo   0
last_price    0
dtype: int64
```

Figura 20 – Remoção de dados nulos da coluna *Last_price*

3.6. Tratamento outliers

Em tal processo foi usado dois passos para a remoção de outliers, um com foco na variável target e outro com uma análise multivariada.

3.6.1. Variável target

Tendo em vista a existência de carros muito antigos e carros muito luxuosos com preços altíssimos e com pouca frequência no *dataset* foi feita inicialmente uma limpeza de outliers referente ao preço utilizando os intervalos interquartis para cada *dataframe* de daída criado no item 3.5:

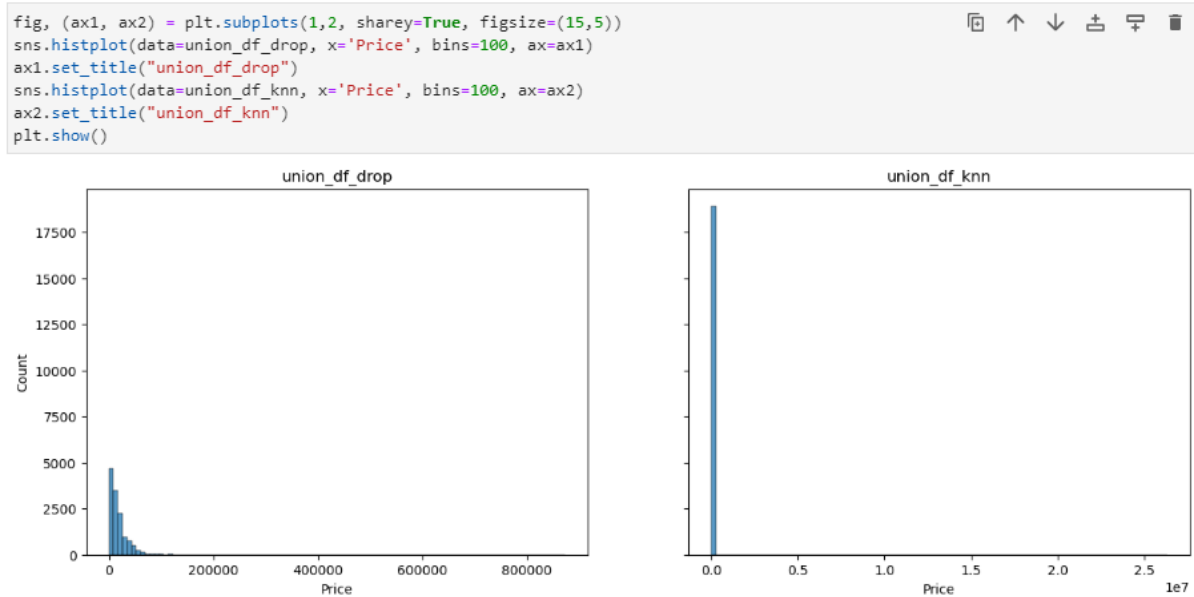


Figura 20 – Distribuição do preço antes da remoção de outliers

Como observado acima vemos uma difícil interpretação devido a existência de veículos com preços muito extremos.

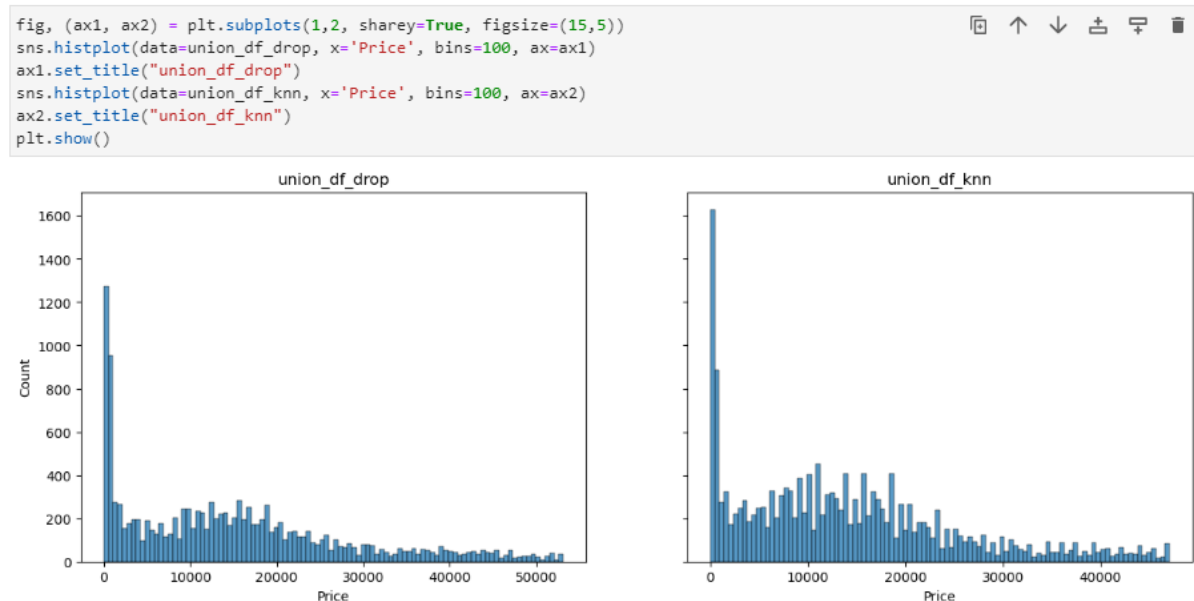


Figura 23 – Remoção dos pontos extremos através do uso dos intervalos interquartis

Após a utilização da técnica mostrada acima conseguimos ver abaixo como a distribuição foi melhorada e os pontos extremos excluídos.

3.6.2 Análise multivariada

3.6.2.1 Criação de Dummies

Antes da remoção de multivariada foi realizada o processo de criação de *dummies* para facilitar no uso das variáveis categóricas em métodos como KNN.

Antes da realização de transformação foi verificada o número de dados diferentes por coluna categórica existente a fim de saber quais recursos utilizar e evitar certos problemas:

```
cat_features = ['Manufacturer', 'Model', 'Category', 'Leather_interior', 'Fuel_type', 'Gear_box_type', 'Drive_wheels',
num_features = ['Price', 'Levy', 'Prod_year', 'Engine_volume', 'Mileage', 'Cylinders', 'Doors', 'Airbags', 'Age', 'Mileage_p

for col in cat_features:
    print(f"{col} - {union_df[cat_features][col].nunique()}")

Manufacturer - 60
Model - 1492
Category - 11
Leather_interior - 2
Fuel_type - 7
Gear_box_type - 4
Drive_wheels - 3
Wheel - 2
Color - 16
Exist_Turbo - 2
```

Figura 20 – Estudo de valores distintos por coluna categórica

Como observado na figura acima há existência de *features* com muitos valores distintos como é o caso do *Model*, *Manufacturer* e *Color* por exemplo. Como forma de não provocar a maldição da dimensionalidade gerando muitas colunas com o método *get_dummies*, foi usado para tais *features* o uso do *label encoder* para cada uma das dataframes de saída da etapa 3.6.1:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

def dummies(df):
    # Encode labels
    df['Manufacturer'] = le.fit_transform(df['Manufacturer'])
    df['Model'] = le.fit_transform(df['Model'])
    df['Color'] = le.fit_transform(df['Color'])

    # Outras colunas categóricas
    df = pd.get_dummies(df)

    return df
```

Figura 21 – Função criada para gerar as dummies

Ao executar a função acima para cada um dos dataframes existentes, os resultados foram gravados nas seguintes variáveis: `df_dummies_drop` e `df_dummies_knn`.

```
df_dummies_drop = dummies(union_df_drop)
df_dummies_knn = dummies(union_df_knn)
```

Figura 22 – Execução e gravação dos resultados

3.6.2.2 KNN

Tendo as variáveis categóricas prontas para serem usadas no modelo, criou-se a função de para remoção de outliers via KNN.

```
from pyod.models.knn import KNN

def knn_outliers(df_dummies, df_orig):
    x = df_dummies.drop(columns=['ID'], axis=1).copy()

    # Treinar KNN
    clf = KNN()
    clf.fit(x)
    # Obter Labels e número de outliers
    y = clf.labels_ # binary labels (0: inliers, 1: outliers)
    #Saídas
    print(np.unique(y, return_counts=True))
    print(f"Shape antes:{df_orig.shape}")

    outliers = []
    for i in range(len(y)):
        if y[i] == 1:
            outliers.append(i)

    outliers_df = df_orig.iloc[outliers,:]
    df_without_outliers = df_orig.drop(outliers_df.index)

    print(f"Shape depois:{df_without_outliers.shape}")
    return df_without_outliers
```

Figura 23 – Criação da função para rodar KNN

Tendo em vista a criação da função, foi a mesma executada para cada dataframe criada no item 3.6.2.1. Sendo assim obte-se os seguintes resultados:

```
df_without_outliers_knn = knn_outliers(df_dummies_knn, union_df_knn)

(array([0, 1]), array([16072, 1786], dtype=int64))
Shape antes:(17858, 20)
Shape depois:(16072, 20)

df_without_outliers_drop = knn_outliers(df_dummies_drop, union_df_drop)

(array([0, 1]), array([11428, 1270], dtype=int64))
Shape antes:(12698, 20)
Shape depois:(11428, 20)
```

Figura 24 – Resultados após execução da função

Com auxílio do modelo KNN, foi identificado a existência de 1786 linhas como outliers para o dataframe com tratamento de nulos com KNN e 1270 linhas identificadas como outliers para o dataframe com tratamento de nulos com eliminação de linhas.

4. Análise e Exploração dos Dados

4.1 Análise de features

Visto que a limpeza de dados após a remoção de outliers nos gerou duas bases denominadas por *df_without_outliers_knn* e *df_without_outliers_drop*, tais bases foram submetidas a uma análise inicial de correlação como critério de escolha de qual base seguir a análise exploratória.

Ao executar a análise de correlação obteve-se os seguintes resultados:

```
corr = df_without_outliers_knn.corr(method='pearson')
plt.figure(figsize = (8,5))
plt.title(f'Correlation | df_without_outliers_knn')
sns.heatmap(corr, annot = True, vmin=-1, vmax=1, annot_kws={"size":6})
plt.show()
```

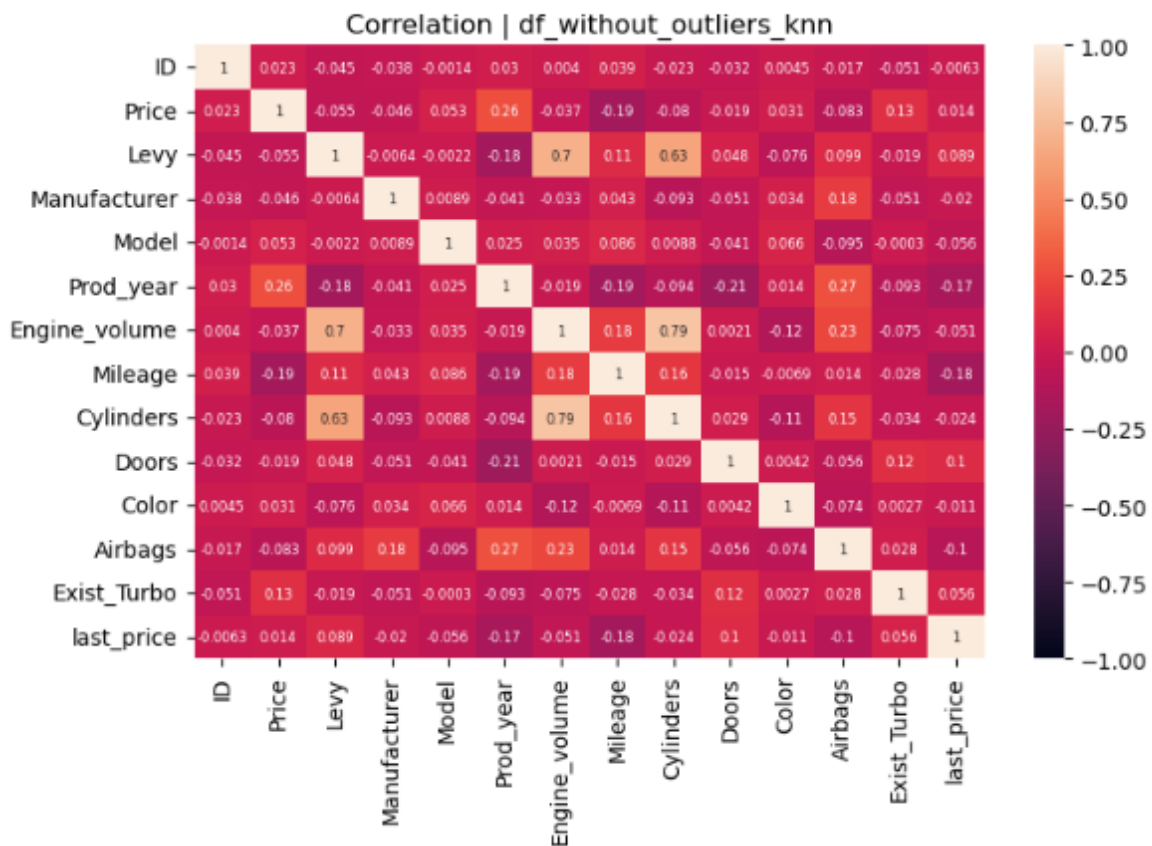


Figura 25 – Correlação com *dataframe* *df_without_outliers_knn*

```
corr = df_without_outliers_drop.corr(method='pearson')
plt.figure(figsize = (8,5))
plt.title(f'Correlation | df_without_outliers_drop')
sns.heatmap(corr, annot = True, vmin=-1, vmax=1, annot_kws={"size":6})
plt.show()
```

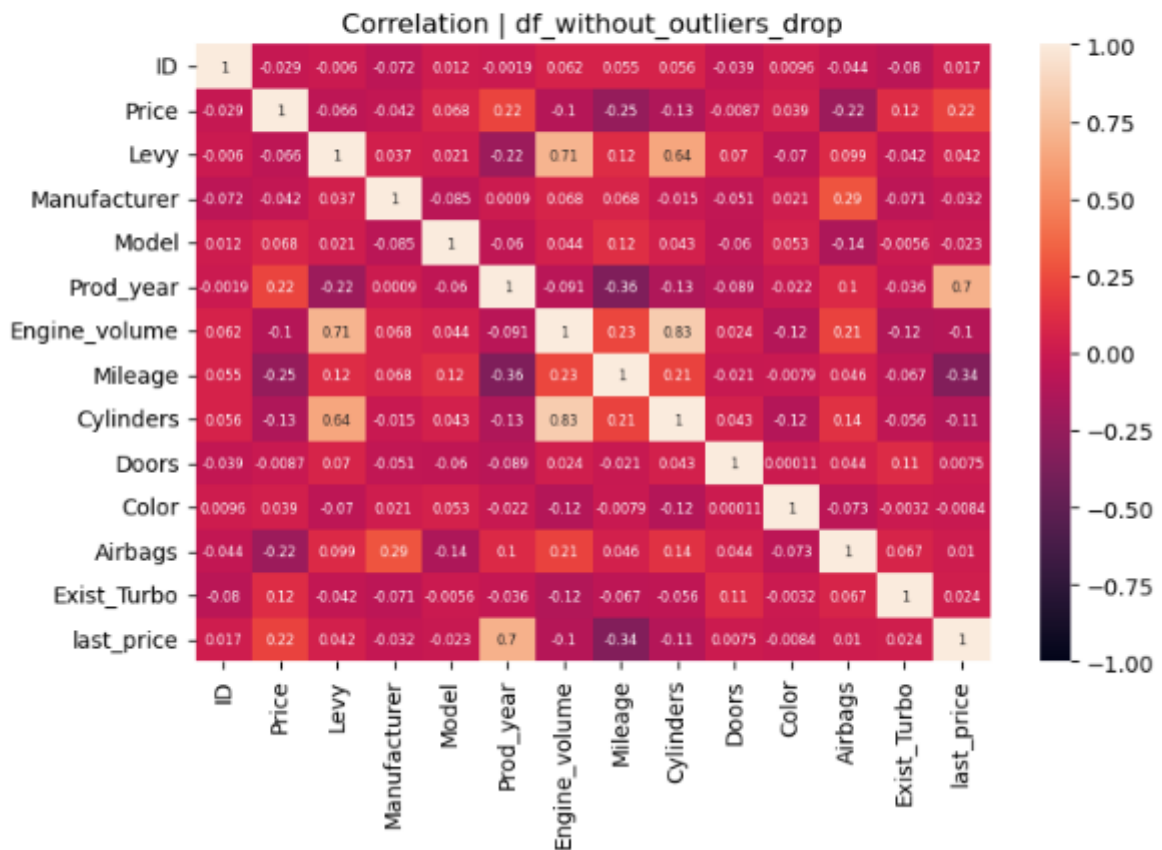


Figura 26 – Correlação com *dataframe* *df_without_outliers_drop*

Ao comparar os resultados, observamos que algumas features ganham um pouco mais de força quando analisamos a tabela *df_without_outliers_drop* com relação a tabela *df_without_outliers_knn*. Dentre as principais features estão a *last_price*, indicando uma influencia da variação do dólar e a influencia da quantidade de airbags.

Sendo assim, essa etapa foi determinante para a escolha *dataframe* escolhido baseado na base que mais possui feature diretamente correlacionado com a variável alvo que é o preço, neste caso o *dataframe* escolhido foi *df_without_outliers_drop*.

Após tal escolha foi realizado uma análise categórica para saber quais features categóricas podem ser relevantes para alterar a média do preço, para isso foi usado ANOVA para avaliar tais relações:

```

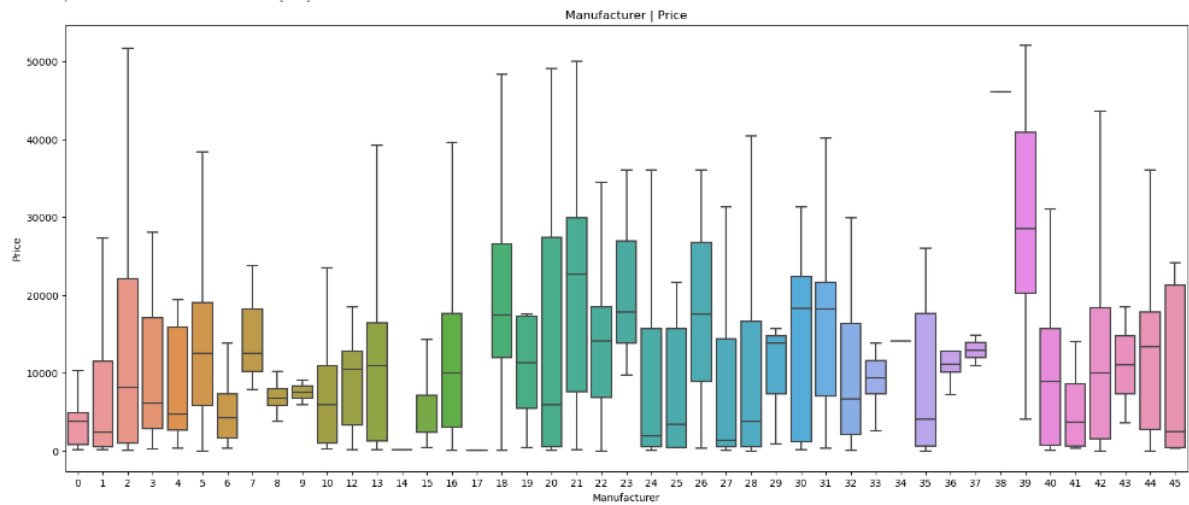
for col in cat_features:
    df = df_without_outliers_drop
    var = 'Price'
    # ANOVA
    category_group = df.groupby(col)[var].apply(list)
    anova = f_oneway(*category_group)
    print(f"P-Value para variavel {col} é: ",{anova[1]})
    # BOXPLOT
    plt.figure(figsize=(20,8))
    plt.title(f'{col} | {var}')
    sns.boxplot(data = df, x = col, y = var, showfliers = False);
    plt.show();

```

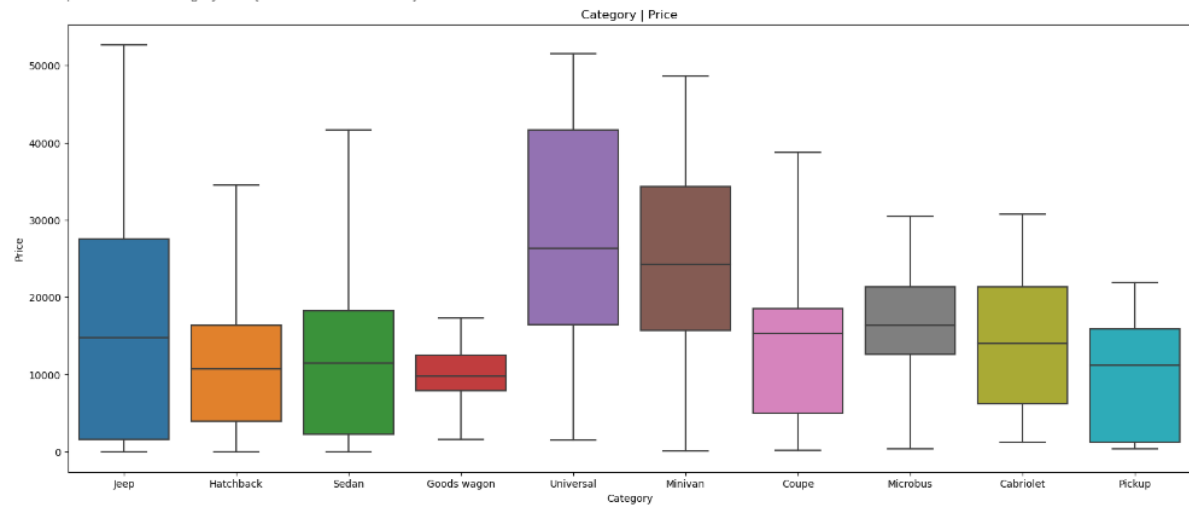
Figura 27 - Looping para rodar ANOVA

A partir da função acima executada obteve-se os seguintes resultados:

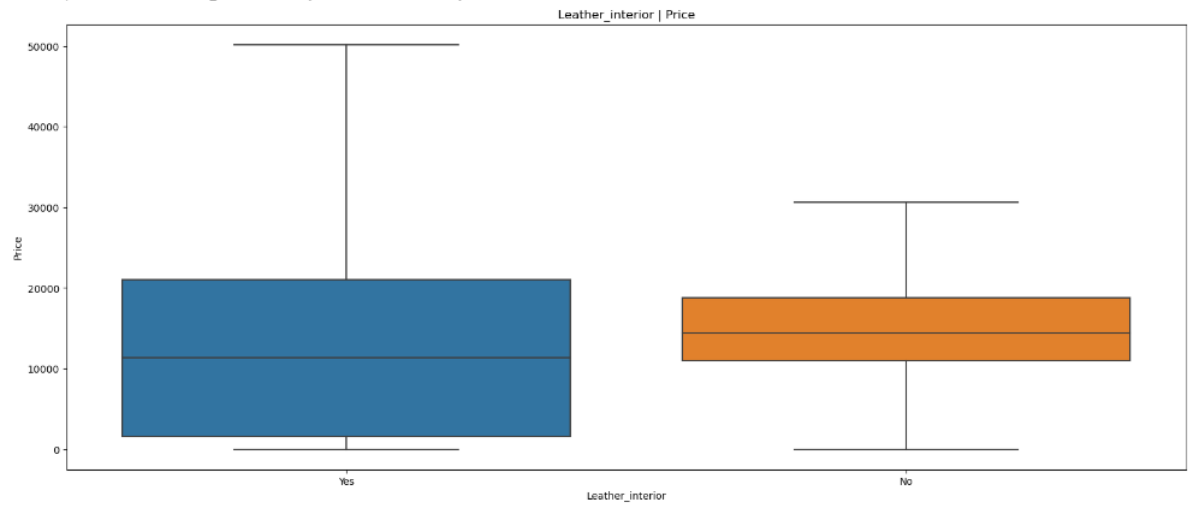
P-Value para variavel Manufacturer é: {0.0}



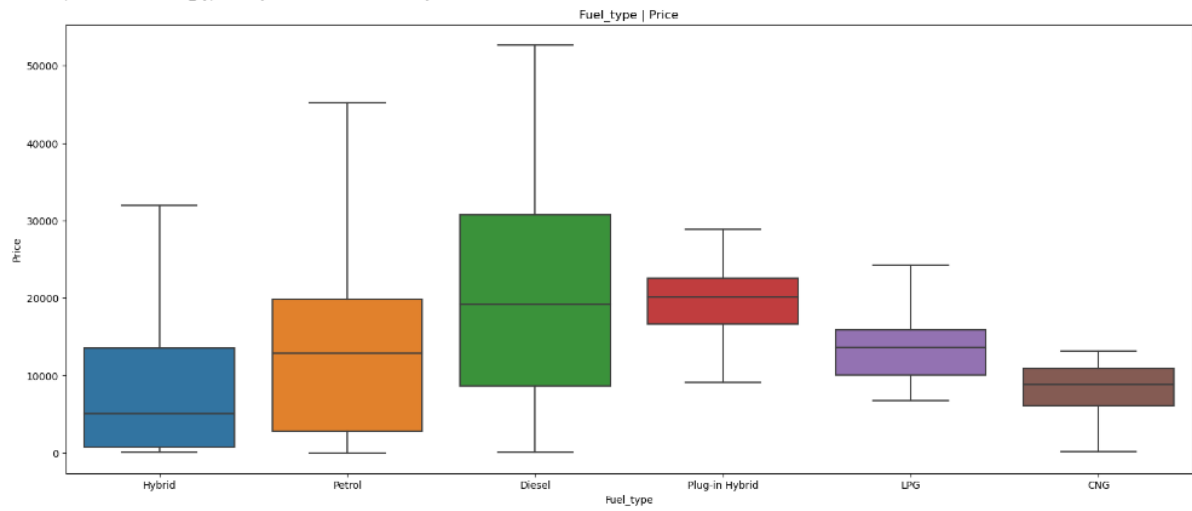
P-Value para variavel Category é: {9.036511766005489e-201}



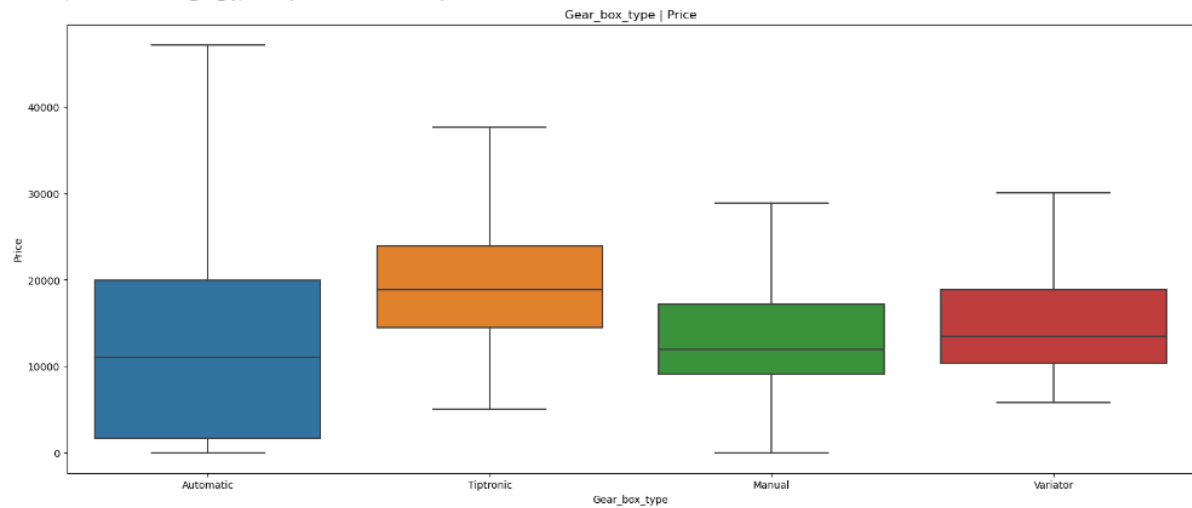
P-Value para variável Leather_interior é: {8.948577672633194e-09}



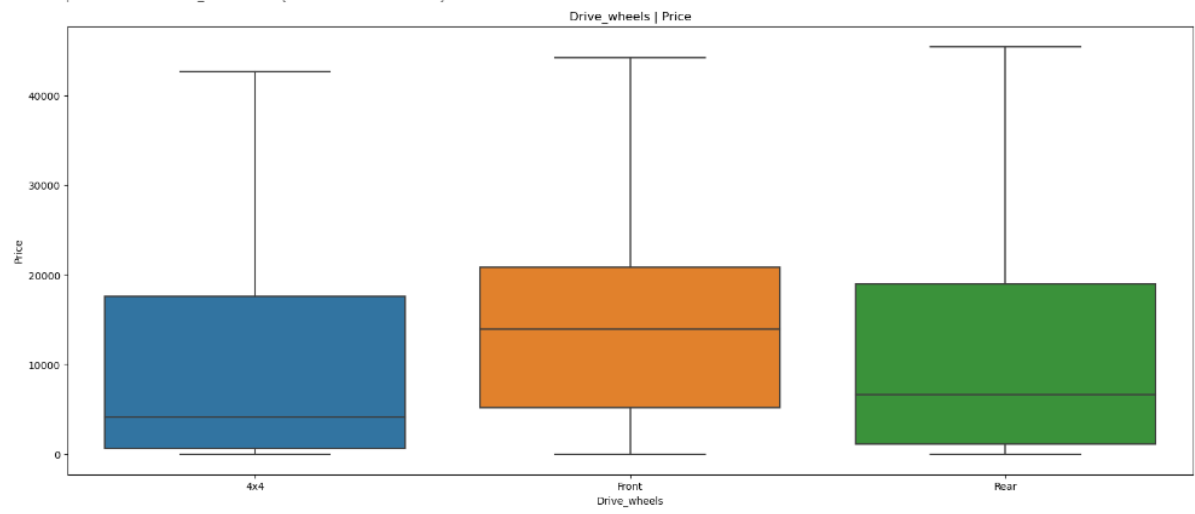
P-Value para variável Fuel_type é: {1.8762064104744488e-300}



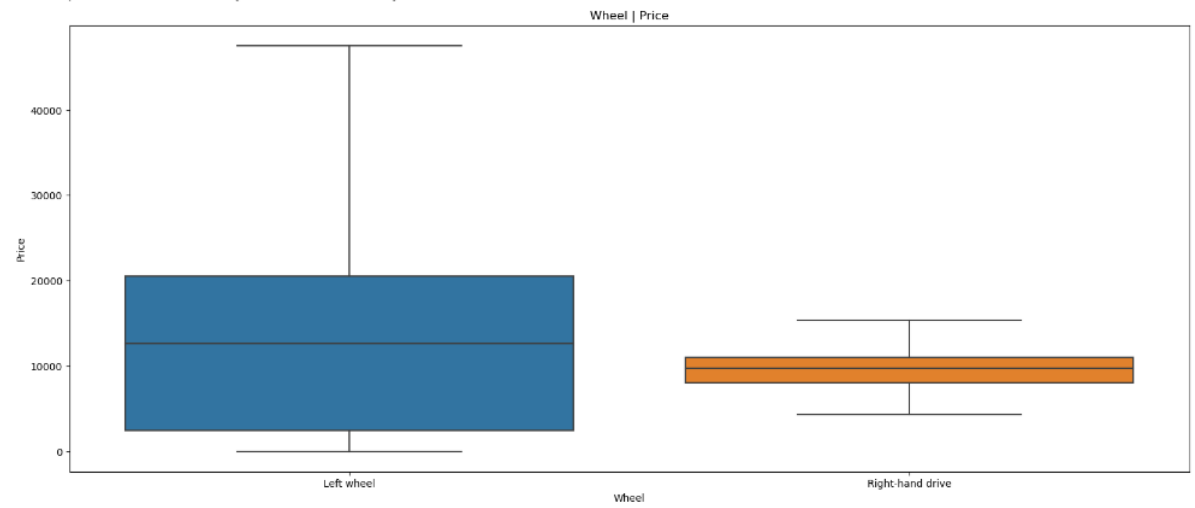
P-Value para variável Gear_box_type é: {6.820932934398122e-79}



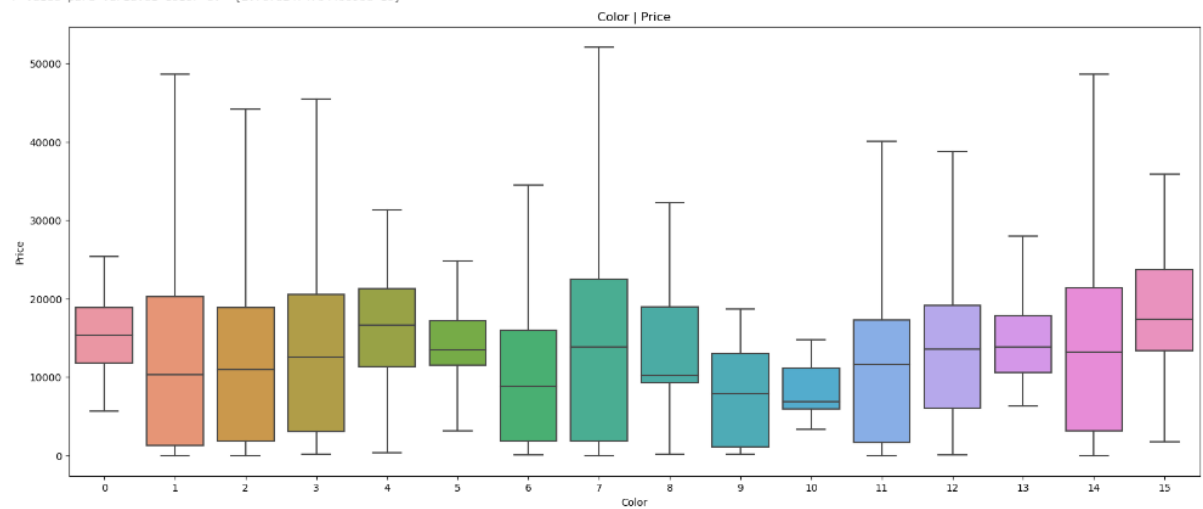
P-Value para variavel Drive_wheels é: {2.4986467416718592e-54}



P-Value para variavel Wheel é: {2.8760502238479326e-07}



P-Value para variavel Color é: {2.7678247475445093e-10}



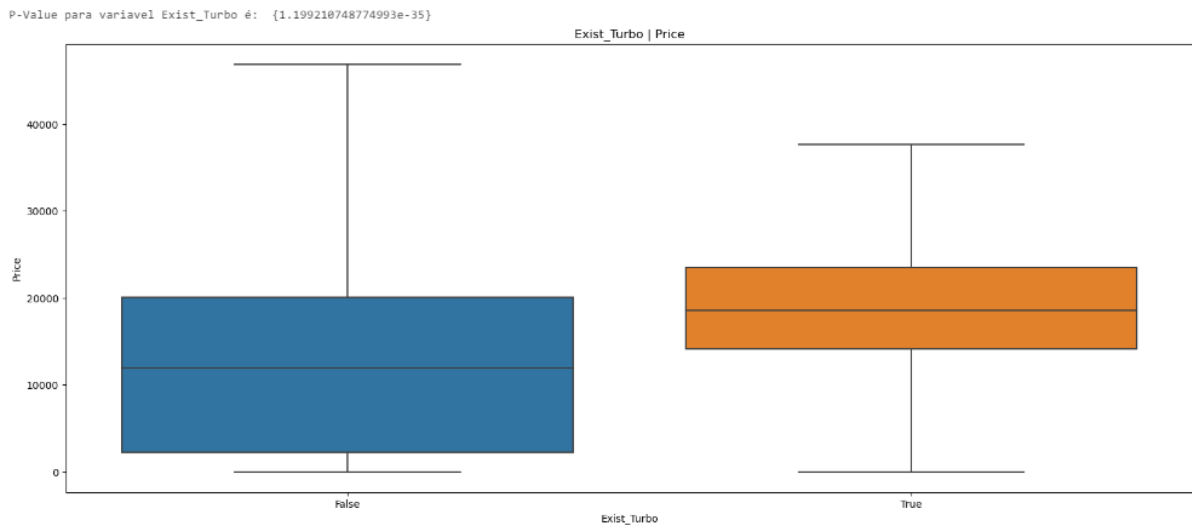


Figura 28 – Resultado ANOVA

Analisando os resultados acima e com base na hipótese que H_0 está relacionado a uma mesma média entre os grupo e H_1 que as médias entre os grupos diferem chegamos à conclusão para o nível de confiança de 5% que as features Manufacturer, Category, Fuel_type, Gear_box_type, Drive_wheels, Wheel, Color e Exist_Turbo são features relevantes para a entrada do modelo.

4.2. Seleção de features

Tendo como base a análise de correlação do item anterior foram selecionados algumas *features* de entrada para o modelo de previsão de preço, sendo assim as *features* selecionadas foram:

- 'Manufacturer'
- 'Prod_year'
- 'Category'
- 'Fuel_type'
- 'Mileage'
- 'Gear_box_type'
- 'Drive_wheels'
- 'Wheel'
- 'Color'
- 'Airbags'
- 'Exist_Turbo'
- 'last_price'

4.3. Análise multicolinearidade

Sabendo que a multicolinearidade é uma situação em que duas ou mais variáveis independentes em um modelo de regressão encontram-se altamente correlacionadas e que a alta correlação pode afetar a qualidade dos resultados do modelo, nessa etapa foi realizada a análise do fator de inflação da variância (VIF).

O VIF avalia o quanto a variância de um coeficiente de regressão estimado aumenta se as suas preditoras estiverem correlacionadas, ou seja, caso o VIF resulte em 1 isso quer dizer que nenhum fator está correlacionado.

Sendo assim a implementação da análise foi executada segundo a função abaixo:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant
def get_VIF(dataFrame , target):
    X = add_constant(dataFrame.loc[:, dataFrame.columns != target])
    seriesObject = pd.Series([variance_inflation_factor(X.values,i) for i in range(X.shape[1])] , index=X.columns,)
    return seriesObject

target = 'Price'
print(get_VIF(df_features_selected.select_dtypes(['int64', 'float64']),target))
```

const	996257.431070
Prod_year	2.054348
Mileage	1.173596
Airbags	1.023432
last_price	1.999927
dtype:	float64

Figura 29 – Resultado VIF

Segundo a figura 29 conseguimos observar valores bem próximo de 1 o que sugere que as variáveis numéricas com boa correlação selecionadas não possuem correlações entre si, não havendo a necessidade de exclusão de *features*.

5. Criação de Modelos de Machine Learning

Com base nas *features* selecionadas e sendo tal problema um problema de regressão nesse projeto foi usado três tipos de modelos para serem comparados, sendo *eles Random Forest Regression, ElasticNet Regression e XGB Regressor*. Para tal antes da execução dos modelos foi realizado 2 etapas de pré-processamento para a entrada dos modelos que foi a conversão dos dados categóricos em numéricos e a normalização dos dados.

5.1 Pre-processamento

Como indicado a primeira etapa foi a conversão dos dados categóricos em dados numéricos através da criação de Dummies seguindo o mesmo raciocínio do item 3.6.2.1, sendo assim a execução pode ser vista na imagem abaixo:

```
# Encode Labels
df_features_selected['Manufacturer'] = le.fit_transform(df_features_selected['Manufacturer'])
df_features_selected['Color'] = le.fit_transform(df_features_selected['Color'])

# Outras colunas categóricas
df_model_dummies = pd.get_dummies(df_features_selected)

df_model_dummies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11428 entries, 0 to 19236
Data columns (total 33 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Price                                11428 non-null  int64
1   Manufacturer                         11428 non-null  int64
2   Prod_year                           11428 non-null  int64
3   Mileage                             11428 non-null  float64
4   Color                               11428 non-null  int64
5   Airbags                             11428 non-null  int64
6   Exist_Turbo                         11428 non-null  bool
7   last_price                          11428 non-null  float64
8   Category_Cabriolet                  11428 non-null  uint8
9   Category_Coupe                      11428 non-null  uint8
10  Category_Goods wagon                11428 non-null  uint8
11  Category_Hatchback                  11428 non-null  uint8
12  Category_Jeep                       11428 non-null  uint8
13  Category_Microbus                   11428 non-null  uint8
14  Category_Minivan                    11428 non-null  uint8
15  Category_Pickup                     11428 non-null  uint8
16  Category_Sedan                      11428 non-null  uint8
17  Category_Universal                  11428 non-null  uint8
18  Fuel_type_CNG                       11428 non-null  uint8
19  Fuel_type_Diesel                    11428 non-null  uint8
20  Fuel_type_Hybrid                    11428 non-null  uint8
21  Fuel_type_LPG                       11428 non-null  uint8
22  Fuel_type_Petrol                    11428 non-null  uint8
23  Fuel_type_Plug-in Hybrid            11428 non-null  uint8
24  Gear_box_type_Automatic              11428 non-null  uint8
25  Gear_box_type_Manual                 11428 non-null  uint8
26  Gear_box_type_Tiptronic              11428 non-null  uint8
27  Gear_box_type_Variator               11428 non-null  uint8
28  Drive_wheels_4x4                    11428 non-null  uint8
29  Drive_wheels_Front                  11428 non-null  uint8
30  Drive_wheels_Rear                   11428 non-null  uint8
31  Wheel_Left wheel                    11428 non-null  uint8
32  Wheel_Right-hand drive              11428 non-null  uint8
dtypes: bool(1), float64(2), int64(5), uint8(25)
```

Figura 30 – Criação de dummies para entrada do modelo

Posterior à conversão dos elementos categóricos foi realizada a etapa de normalização dos dados, visto que a não normalização pode conceder pesos de importância para cada *feature* dependendo da sua ordem de grandeza. Para isso foi realizada a conversão via *StandardScaler* no qual se padroniza as *features* removendo a média e escala a variância a uma unidade. Tal processo pode ser visto na figura 31 abaixo:

```

from sklearn.preprocessing import StandardScaler

col_nb = df_model_dummies.select_dtypes(['int64', 'float64']).columns.to_list()

scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_model_dummies[col_nb])

df_normalize = df_model_dummies.copy()
df_normalize[col_nb] = scaled_data

```

Figura 31 – Normalização das variáveis de entrada do modelo

5.2 Separação dos dados de treinamento e teste

Antes da execução dos modelos e com o objetivo de se ter o mesmo cenário para comparação entre eles foi realizada a separação do dataset em treinamento e teste, sendo 70% e 30% respectivamente e selecionado o random_state em 42.

Sendo assim, o código de separação executado foi:

```

X = df_normalize.drop(columns=['Price'])
y = df_normalize['Price']

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 42,
                                                    shuffle = True)

print('X_train:', X_train.shape)
print('X_test:', X_test.shape)
print('y_train:', len(y_train))
print('y_test:', len(y_test))

X_train: (11250, 34)
X_test: (4822, 34)
y_train: 11250
y_test: 4822

```

Figura 32 – Split treino e teste

5.3 Modelo

A execução de cada modelo foi feita em 2 etapas, sempre começada pelo processo de tuning de parâmetros a fim de se economizar tempo de desenvolvimento e por seguinte a execução e coleta de resultados dos modelos.

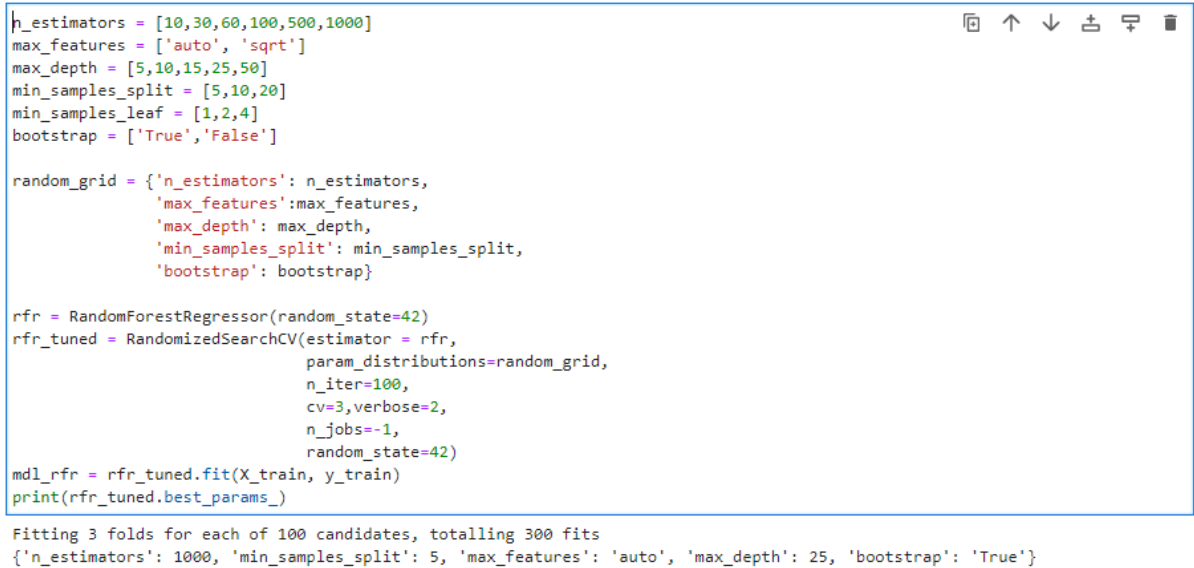
5.3.1 Random Forest Regression

O uso do algoritmo do random forest (floresta aleatória), apesar das árvores serem estruturas mais simples, é um algoritmo que cria de forma aleatória várias arvores de decisão e utiliza da combinação delas para chegar no melhor resultado, o que torna muito poderoso esse algoritmo.

Sendo assim foi realizado inicialmente o tuning do modelo variando os seguintes parâmetros:

- `n_estimators = [10,30,60,100,500,1000]`
- `max_features = ['auto', 'sqrt']`
- `max_depth = [5,10,15,25,50]`
- `min_samples_split = [5,10,20]`
- `bootstrap = ['True','False']`

Após rodar o algoritmo para encontrar a melhor combinação de parâmetros para a base de entrada proposta encontramos o seguinte resultado:



```

n_estimators = [10,30,60,100,500,1000]
max_features = ['auto', 'sqrt']
max_depth = [5,10,15,25,50]
min_samples_split = [5,10,20]
min_samples_leaf = [1,2,4]
bootstrap = ['True','False']

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'bootstrap': bootstrap}

rfr = RandomForestRegressor(random_state=42)
rfr_tuned = RandomizedSearchCV(estimator = rfr,
                              param_distributions=random_grid,
                              n_iter=100,
                              cv=3,verbose=2,
                              n_jobs=-1,
                              random_state=42)
mdl_rfr = rfr_tuned.fit(X_train, y_train)
print(rfr_tuned.best_params_)

```

Fitting 3 folds for each of 100 candidates, totalling 300 fits
{'n_estimators': 1000, 'min_samples_split': 5, 'max_features': 'auto', 'max_depth': 25, 'bootstrap': 'True'}

Figura 33 – Tuning dos parâmetros

- `n_estimators = 1000`
- `max_features = auto`
- `max_depth = 25`
- `min_samples_split = 5`
- `bootstrap = True`

Com isso executamos o primeiro modelo usando *RandomForestRegressor* do *Sklearn* e desenvolvido em *python* como mostrado abaixo:

```
# Fit do modelo
rfr_tuned_final = RandomForestRegressor(n_estimators = 1000,
                                       min_samples_split = 5,
                                       max_features = 'auto',
                                       max_depth = 25,
                                       bootstrap = 'True',
                                       random_state = 42)

mdl_rfr_final = rfr_tuned_final.fit(X_train, y_train)

# Predição do teste
y_pred_rfr = mdl_rfr_final.predict(X_test)
y_true_rfr = y_test.tolist()

# Métricas de validação
r2_rfr = r2_score(y_true_rfr, y_pred_rfr)
rmse_rfr = mean_squared_error(y_true_rfr, y_pred_rfr)**(1/2)
mae_rfr = mean_absolute_error(y_true_rfr, y_pred_rfr)
```

Figura 32 – Execução do fit do modelo, predição e validação

Como observado na figura 32 foi realizado o fit do modelo juntamente com a execução de 3 métricas de validação, sendo eles: R2, RMSE e MAE obtidos pelo “sklearn.metrics”.

Como forma de avaliar o desempenho do modelo nessas condições foi elaborado *plot* do desempenho do modelo, análise de resíduo e análise de parâmetros importantes adotados pelo modelo.

Com a finalização do processamento do modelo obteve-se o seguinte resultado:

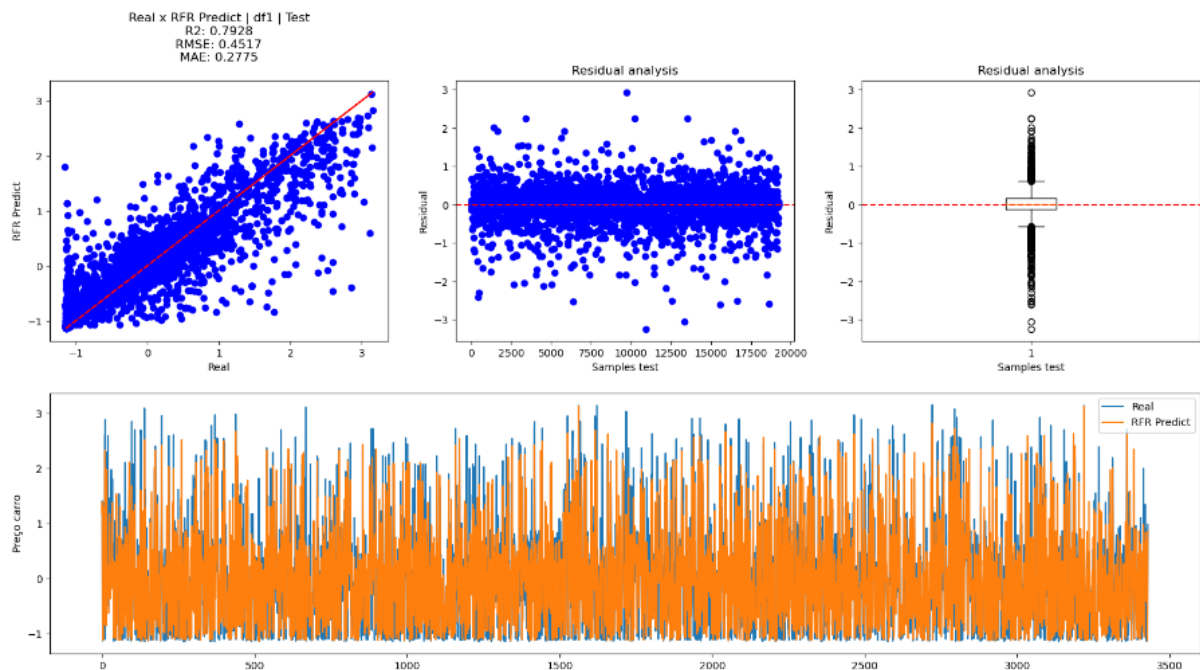


Figura 33 – Resultado do modelo

Para tal modelo foi considerado como 10 fatores mais importantes as seguintes features:

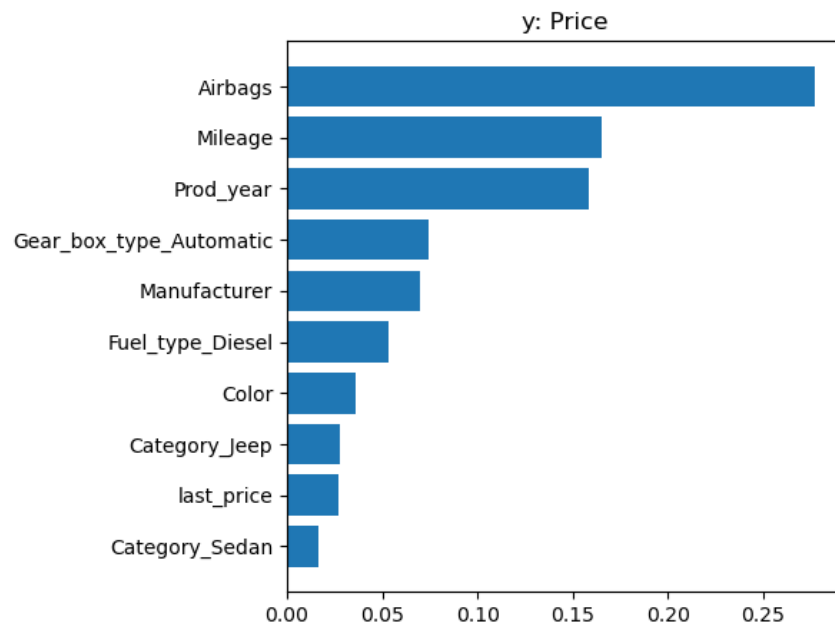


Figura 34 – Feature Importance do modelo

5.3.2 ElasticNet Regression

O algoritmo ElasticNet é um modelo intermediário entre Lasso e Ridge, ou seja, utiliza-se de meios de regularização ajudando a diminuir a variância evitando um possível overfitting e tendendo a melhorar a qualidade de predição. Esse foi o principal ponto que levou tal projeto a utilizar tal modelo.

Dentro desse contexto, o modelo Elastic tem como parâmetro α . Sendo assim, o primeiro passo para o *fit* do modelo assim como já observado no Random Forest foi executar o tuning do modelo:

```
# Tuning do modelo
from sklearn.linear_model import ElasticNetCV

results = ElasticNetCV(cv = 10).fit(X_train,y_train)
```

```
results.alpha_
```

```
0.0005074276890633889
```

Figura 35 – Tuning modelo ElasticNet

Na figura acima é importante salientar que foi utilizado para geração dos cenários o *ElasticNetCV* advindo do *sklearn* com dados de entrada considerando *cv* igual a 10.

No caso observado acima os valores ótimos do *tuning* foi aproximadamente α 0.00051. Com isso o fit do modelo foi executado da seguinte maneira:

```
# Let's create the final model according to optimum alpha.
enet_tuned = ElasticNet(alpha = results.alpha_).fit(X_train,y_train)

# Predict test data
y_pred_enet_tuned = enet_tuned.predict(X_test)
y_true_enet = y_test.tolist()

# Metrics RFR Predicts
r2_enet_tuned = r2_score(y_test,y_pred_enet_tuned)
rmse_enet_tuned = mean_squared_error(y_test,y_pred_enet_tuned)**(1/2)
mae_enet_tuned = mean_absolute_error(y_test,y_pred_enet_tuned)
```

Figura 36 – Execução do fit do modelo, predição e validação

Com a finalização do processamento do modelo obteve-se o seguinte resultado:

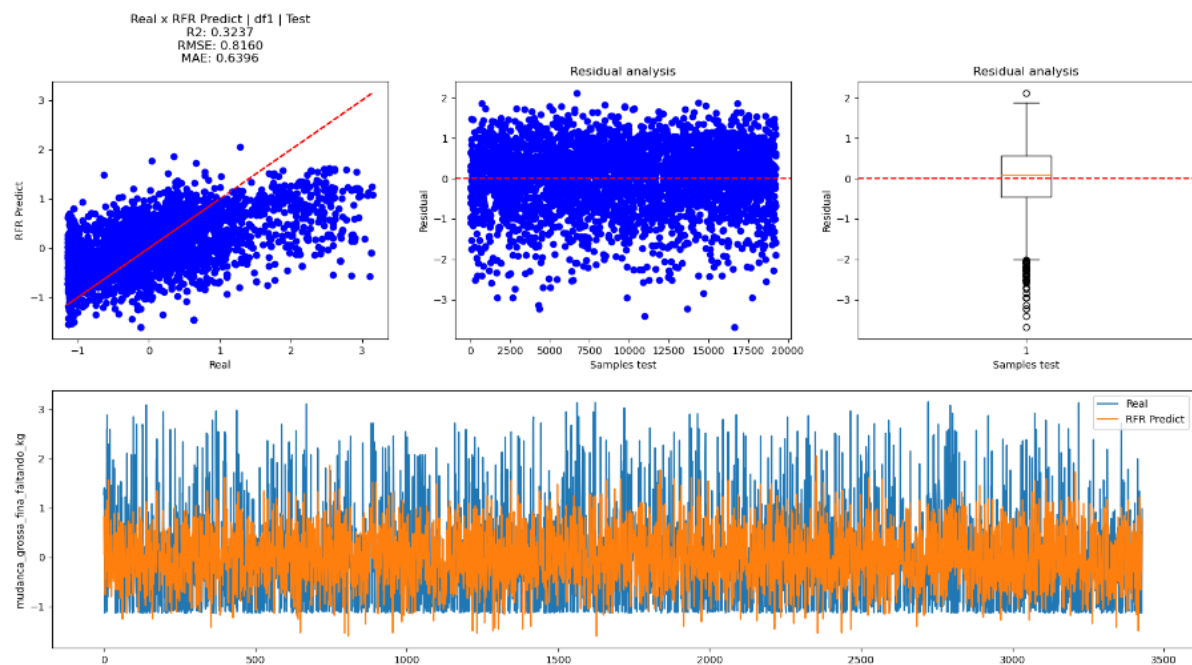


Figura 37 – Execução do fit do modelo, predição e validação

Para tal modelo foi considerado como 10 fatores mais importantes as seguintes features:

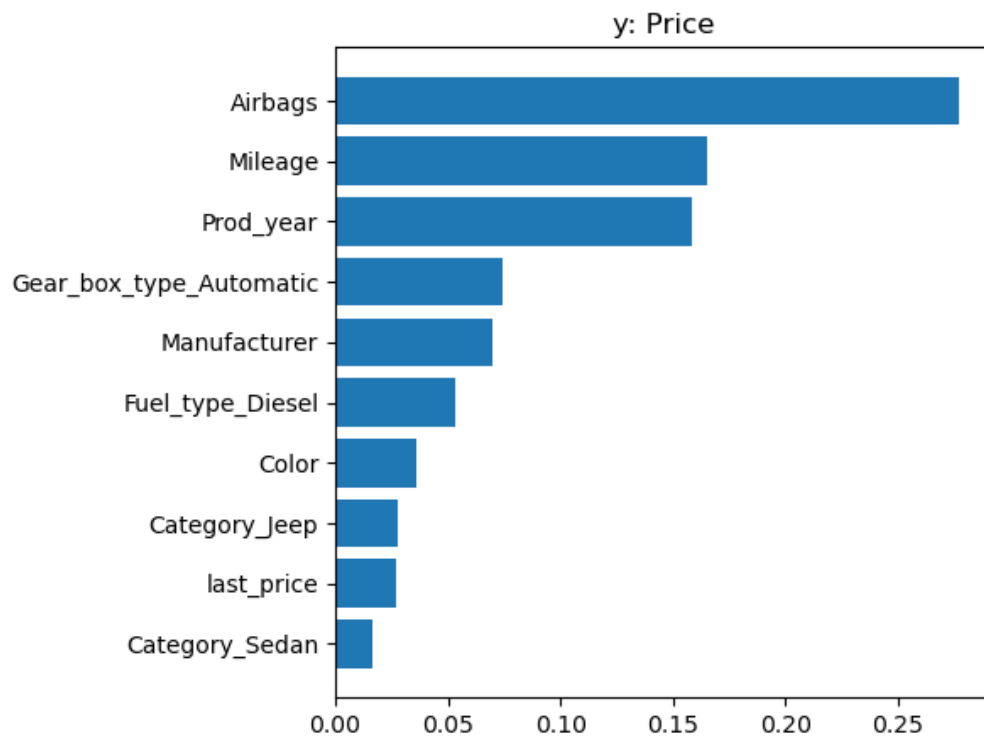


Figura 38 – *Feature Importance* do modelo

5.3.3. XGBRegressor

Dentre os principais pontos e qualidade para se escolher o algoritmo XGBRegressor está o fato dele conseguir combinar resultados de árvores de decisão e usar o algoritmo *Gradient Descent* para minimizar a perda além de ser um algoritmo extremamente rápido de se executar em comparação com outros modelos.

Sendo assim no projeto o modelo passou pelo processo de tuning mediante os seguintes cenários de parâmetros:

- `n_estimators = [10,30,60,100,500,1000]`
- `max_depth = [6,10,15,25]`
- `eta = [0.1,0.3,0.5]`
- `subsample = [1,2,3,4]`
- `colsample_bytree = [1,2,3]`

Na figura abaixo é possível de se ver o código escrito utilizando `RandomizedSearchCV` do `sklearn`:

```

n_estimators = [10,30,60,100,500,1000]
max_depth = [6,10,15,25]
eta = [0.1,0.3,0.5]
subsample = [1,2,3,4]
colsample_bytree = [1,2,3]

param = {'max_depth':max_depth,
         'n_estimators': n_estimators,
         'eta': eta,
         'subsample': subsample,
         'colsample_bytree': colsample_bytree}

xb = XGBRegressor(random_state=42)
xb_tuned = RandomizedSearchCV(estimator = xb,
                             param_distributions=param,
                             n_iter=100,
                             cv=3,
                             verbose=2,
                             n_jobs=-1,
                             random_state=42)

mdl_xb = xb_tuned.fit(X_train, y_train)
print(xb_tuned.best_params_)

Fitting 3 folds for each of 100 candidates, totalling 300 fits
{'subsample': 1, 'n_estimators': 500, 'max_depth': 10, 'eta': 0.3, 'colsample_bytree': 1}

```

Figura 39 – Tuning modelo XGB

Com os parâmetros que melhor se adaptam ao modelo foi executado o fit do modelo nos mesmos padrões dos modelos apresentados até aqui:

```

# Fit tuned model
xb_tuned_final = XGBRegressor(subsample = 1,
                             n_estimators = 500,
                             max_depth = 10,
                             eta = 0.3,
                             colsample_bytree = 1)

mdl_xb_final = xb_tuned_final.fit(X_train, y_train)

# Predict test data
y_pred_xb = mdl_xb_final.predict(X_test)
y_true_xb = y_test.tolist()

# Metrics RFR Predicts
r2_xb = r2_score(y_true_xb,y_pred_xb)
rmse_xb = mean_squared_error(y_true_xb,y_pred_xb)**(1/2)
mae_xb = mean_absolute_error(y_true_xb,y_pred_xb)

```

Figura 40 – Execução do fit do modelo, predição e validação

Com a finalização do processamento do modelo obteve-se o seguinte resultado:

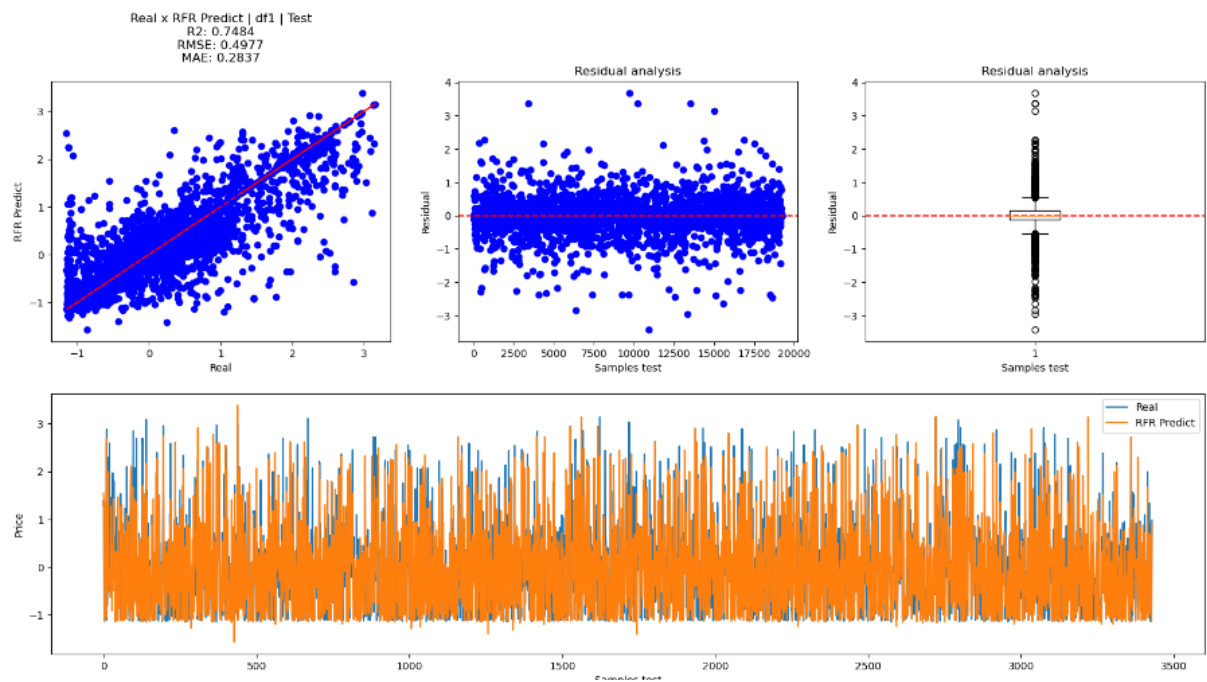


Figura 41 – Execução do fit do modelo, predição e validação

Para tal modelo foi considerado como 10 fatores mais importantes as seguintes features:

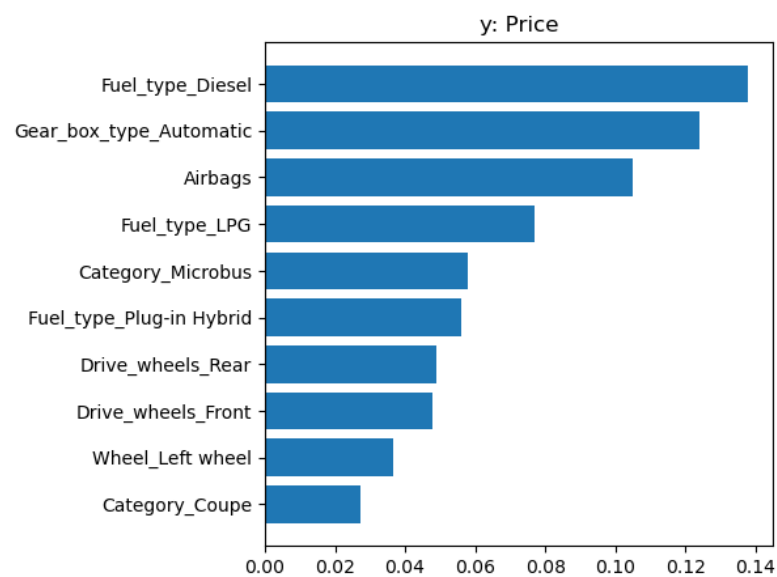


Figura 42 – Feature Importance do modelo

6. Interpretação dos Resultados

6.1 Pré Modelagem

Partindo do princípio de que o uso do método de tratamento de dados nulos foi um ponto crucial para a escolha do *dataframe* de entrada para a análise exploratória, vimos que apesar da perda de aproximadamente 30% das linhas do *dataset* inicial, o uso da técnica de remoção de outliers via KNN baseado em poucas *features* levou à uma tendenciosidade dos dados preenchidos, fazendo com que *features* significantes deixasse de ser input para o modelo. Sendo assim vimos que o tratamento de dados abordado fez com que alguns parâmetros ganhassem relevância como foi o caso das colunas *last_price*, *Airbags*, *Cylinders*, *Mileage*.

A abordagem ANOVA para os parâmetros categóricos revelou diferenças significativas das médias entre a variável target, no caso a coluna *Price* em relação às colunas *Category*, *Fuel_type*, *Gear_box_type*, *Drive_wheels*, *Wheel*, *Color* e *Exist_Turbo* tendo como base uma significância de 5%.

Sendo assim, com base no processo acima foi possível a escolha dos principais parâmetros a serem utilizados em 3 modelos escolhidos (*Random Forest*, *ElasticNet*, *XGB Regressor*) que por sua vez gerou o seguinte resultado:

6.2 Modelos

Quando analisamos o modelo *RandomForest* na figura 33, observamos que dentre os três modelos testados foi o que melhor performou tendo em vista o conjunto de resultados dos 3 parâmetros de validação utilizadas.

No caso, a utilização do R^2 em conjunto do RMSE e MAE foi proposital visto que o uso apenas de um parâmetro pode nos gerar falsos bons modelos. Sendo assim, observamos no modelo *RandomForest* um MAE próximo do RMSE o que indica uma pequena variância sendo $MAE < RMSE$. A não discrepância do RMSE perante o MAE nos mostra a baixa influencia dos outliers no modelo.

Como forma de avaliar melhor os resultados obtidos, foi visto também com certa importância o comportamento dos resíduos, tal etapa é importante pois através da análise de resíduos pode-se identificar quando os resíduos se comportam de forma aleatória, como foi o caso mostrado nas tabelas “*Residual analysis*” dos modelos, ou seja, quando avaliamos o resíduo do modelo *Random Forest* que melhor performou entre os parâmetros vemos que os pontos não seguem um padrão ou tendência, demonstrando uma boa especificação do modelo para o conjunto de dados.

Por fim, tendo escolhido o modelo mais performático foi possível de se ver quais foram as features que mais tiveram importância para boa previsibilidade do modelo, e ao verificarmos o modelo vemos que dentre as principais características temos os airbags, mileage, prod_year, gear_box_automatic, Manufacturer, ou seja, 2 componentes técnicos juntamente com o ano de produção (reflexo da inflação como visto pela alta correlação com índice dólar) e no tipo da montadora são os principais fatores que explicam a variação do preço final no mercado do veículo.

7. Links

Link para o vídeo: <https://youtu.be/XX-y2OhWBpQ>

Link para o repositório: <https://github.com/andremelato/TCC-PucMinas-2023.git>

APÊNDICE

Programação/Scripts

Importar bibliotecas

```
import pandas as pd
from pandas.tseries.offsets import DateOffset
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt

from scipy.stats import f_oneway

from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
```

Puxar dataframes

```
#Base tabela Car_Price
car_price_df = pd.read_csv("Database/pos/car_price_prediction.csv")
# Base histórica do real perante o dólar
dxy_df = pd.read_csv("Database/pos/Índice Dólar Dados Históricos.csv", decimal=',')
```

```
car_price_df.head()
```

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume	Mileage	Cylinders	Gear box type	Drive wheels	Doors	Wheel	Color	Airbags
0	45654403	13328	1399	LEXUS	RX 450	2010	Jeep	Yes	Hybrid	3.5	186005 km	6.0	Automatic	4x4	04-May	Left wheel	Silver	12
1	44731507	16621	1018	CHEVROLET	Equinox	2011	Jeep	No	Petrol	3	192000 km	6.0	Tiptronic	4x4	04-May	Left wheel	Black	8
2	45774419	8467	-	HONDA	FIT	2006	Hatchback	No	Petrol	1.3	200000 km	4.0	Variator	Front	04-May	Right-hand drive	Black	2
3	45769185	3607	862	FORD	Escape	2011	Jeep	Yes	Hybrid	2.5	168966 km	4.0	Automatic	4x4	04-May	Left wheel	White	0
4	45809263	11726	446	HONDA	FIT	2014	Hatchback	Yes	Petrol	1.3	91901 km	4.0	Automatic	Front	04-May	Left wheel	Silver	4

Tratamento dos dados tabela car_price

```
car_price_df['Engine volume'].tail()
```

```
19232    2.0 Turbo
19233         2.4
19234         2
19235         2
19236         2.4
Name: Engine volume, dtype: object
```

```
car_price_df['Mileage'] = car_price_df['Mileage'].str.replace('km', '').astype(float)
```

```
car_price_df['Levy'] = car_price_df['Levy'].replace({'-':np.nan}).astype(float)
```

```
car_price_df['Exist_Turbo'] = car_price_df['Engine volume'].str.contains('Turbo')
```

```
car_price_df['Exist_Turbo'].head()
```

```
0    False
1    False
2    False
3    False
4    False
Name: Exist_Turbo, dtype: bool
```

```
car_price_df['Engine volume'] = car_price_df['Engine volume'].str.replace(' Turbo', '').astype(float)
```

```
car_price_df['Doors'].tail()
```

```
19232    02-Mar
19233    04-May
19234    04-May
19235    04-May
19236    04-May
Name: Doors, dtype: object
```

```
l1 = list(set(car_price_df['Doors']))
l2 = [2,5,4]

replacement_map = {str(i1): int(i2) for i1, i2 in zip(l1, l2)}
car_price_df['Doors'] = car_price_df['Doors'].map(replacement_map)
```

```
car_price_df.columns = ['ID', 'Price', 'Levy', 'Manufacturer', 'Model', 'Prod_year',
                        'Category', 'Leather_interior', 'Fuel_type', 'Engine_volume', 'Mileage',
                        'Cylinders', 'Gear_box_type', 'Drive_wheels', 'Doors', 'Wheel', 'Color',
                        'Airbags', 'Exist_Turbo']
```

Tratamento dos dados tabela DXY (índice dólar)

```
dxy_df['date'] = pd.to_datetime(dxy_df['Data'], dayfirst=True)
```

```
dxy_df['Year'] = dxy_df['date'].dt.year
```

```
dxy_df.drop(columns='Data', inplace=True)
```

```
dxy_df.columns = ['last_price', 'open_price', 'max_price', 'min_price', 'volume', 'var_%', 'date', 'Year']
```

```
dxy_df['var_%'] = dxy_df['var_%'].str.replace('%', '')
dxy_df['var_%'] = dxy_df['var_%'].str.replace(',', '.').astype(float)
```

```
dxy_year_df = dxy_df.groupby('Year').last_price.mean().reset_index()
```

Unificação da base

```
union_df = car_price_df.merge(dxy_year_df, left_on='Prod_year', right_on='Year', how='left')
```

```
union_df.drop(columns='Year', inplace=True)
```

```
union_df.head(2)
```

	ID	Price	Levy	Manufacturer	Model	Prod_year	Category	Leather_interior	Fuel_type	Engine_volume	Mileage	Cylinders	Gear_box_type	Drive_wheels	Doors	W
0	45654403	13328	1399.0	LEXUS	RX 450	2010	Jeep	Yes	Hybrid	3.5	186005.0	6.0	Automatic	4x4	2	W
1	44731507	16621	1018.0	CHEVROLET	Equinox	2011	Jeep	No	Petrol	3.0	192000.0	6.0	Tiptronic	4x4	2	W

Análise de dados duplicados

```
print(union_df['ID'].duplicated().sum())
union_df.ID.value_counts().head()
```

```
313
45815365    8
45815361    8
45815363    7
45815368    7
45723475    7
Name: ID, dtype: int64
```

```
# Shape antes
union_df.shape
```

```
(19237, 20)
```

```
#Shape depois
union_df.drop_duplicates('ID', inplace=True)
union_df.shape
```

```
(18924, 20)
```

Tratamento de Null value

```
union_df.isnull().sum()
```

```
ID          0
Price        0
Levy        5709
Manufacturer 0
Model        0
Prod_year    0
Category     0
Leather_interior 0
Fuel_type    0
Engine_volume 0
Mileage      0
Cylinders    0
Gear_box_type 0
Drive_wheels 0
Doors        0
Wheel        0
Color        0
Airbags      0
Exist_Turbo  0
last_price   15
dtype: int64
```

Tratando LEVY

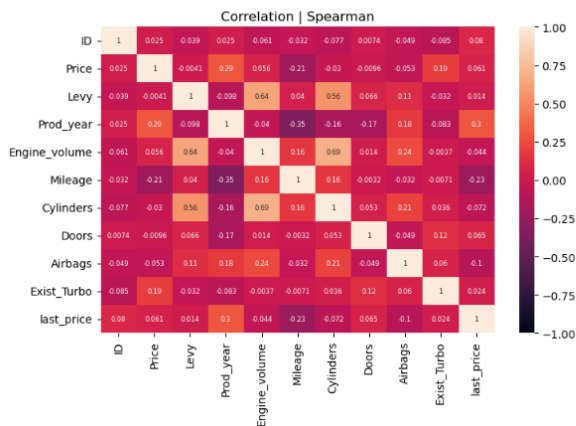
Método Implace KNN

```
# Número de linhas com esse critério nulo chega a 30,16% do DF
```

```
union_df.Levy.isnull().sum()/len(union_df)
```

```
0.30168040583386174
```

```
corr = union_df.corr(method='spearman')
plt.figure(figsize=(8,5))
plt.title('Correlation | Spearman')
sns.heatmap(corr, annot=True, vmin=-1, vmax=1, annot_kws={"size":6})
plt.show()
```



```
# Preenchimento via KNN
```

```
param = ['Engine_volume', 'Cylinders', 'Levy']
```

```
from sklearn.impute import KNNImputer
from sklearn import metrics

# Aplicação
df_fill_levy = union_df[param].copy()

imputer_KNN = KNNImputer()
df_KNN = imputer_KNN.fit_transform(df_fill_levy)
imputer_KNN_imputed = pd.DataFrame(df_KNN, columns=param)

print("ESTATÍSTICA DF ORIGINAL")
print(f"{df_fill_levy.Levy.describe()}\n")
print("ESTATÍSTICA DF COM PREENCHIMENTO")
print(imputer_KNN_imputed.Levy.describe())

union_df_knn = union_df.copy()

# Substituição da coluna preenchida no DF original
union_df_knn['Levy'] = list(imputer_KNN_imputed['Levy'])
```

Método Dropna()

```
union_df_drop = union_df.copy()
```

```
union_df_drop.dropna(axis=0, inplace=True)
```

Tratando DXY

```
union_df_knn.isnull().sum()
```

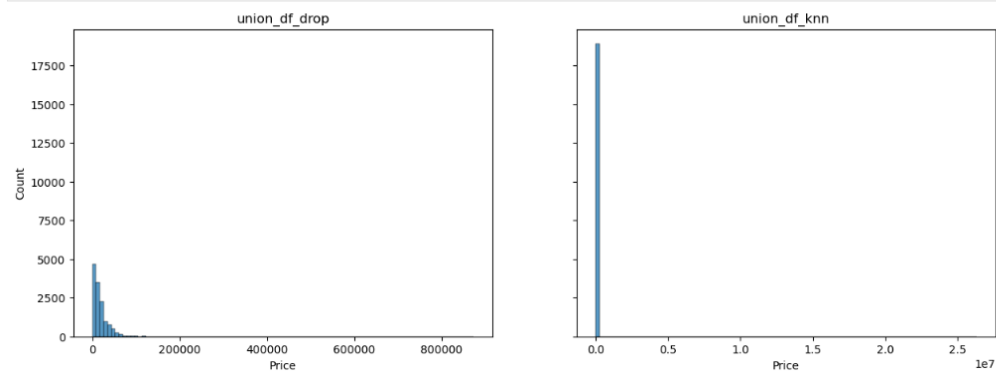
```
ID          0
Price       0
Levy        0
Manufacturer 0
Model       0
Prod_year   0
Category    0
Leather_interior 0
Fuel_type   0
Engine_volume 0
Mileage     0
Cylinders   0
Gear_box_type 0
Drive_wheels 0
Doors       0
Wheel       0
Color       0
Airbags     0
Exist_Turbo 0
last_price  15
dtype: int64
```

```
# Devido ao pequeno número e dos dados serem de carros muito antigos, foi usado dropna para excluir as 15 linhas.
```

```
union_df_knn.dropna(axis=0, inplace=True)
```

Tratamento target

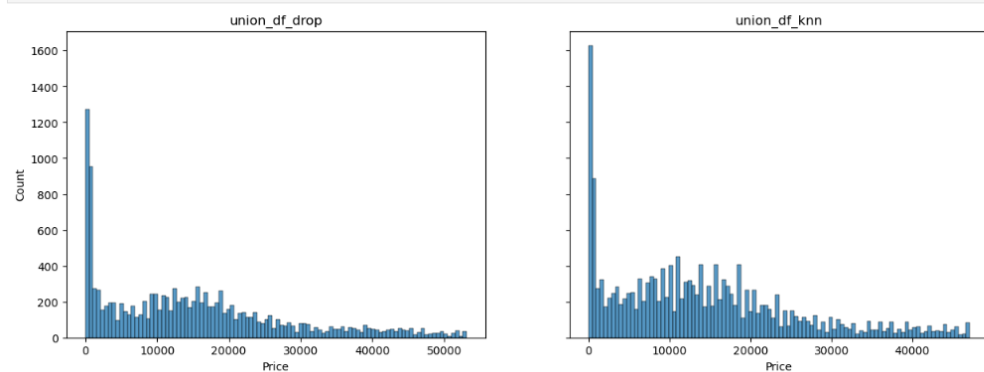
```
fig, (ax1, ax2) = plt.subplots(1,2, sharey=True, figsize=(15,5))
sns.histplot(data=union_df_drop, x='Price', bins=100, ax=ax1)
ax1.set_title("union_df_drop")
sns.histplot(data=union_df_knn, x='Price', bins=100, ax=ax2)
ax2.set_title("union_df_knn")
plt.show()
```



```
q1 = union_df_knn.Price.quantile(0.25)
q3 = union_df_knn.Price.quantile(0.75)
IQR = q3 - q1
union_df_knn = union_df_knn[~((union_df_knn.Price < (q1 - 1.5 * IQR)) | (union_df_knn.Price > (q3 + 1.5 * IQR)))]

q1 = union_df_drop.Price.quantile(0.25)
q3 = union_df_drop.Price.quantile(0.75)
IQR = q3 - q1
union_df_drop = union_df_drop[~((union_df_drop.Price < (q1 - 1.5 * IQR)) | (union_df_drop.Price > (q3 + 1.5 * IQR)))]
```

```
fig, (ax1, ax2) = plt.subplots(1,2, sharey=True, figsize=(15,5))
sns.histplot(data=union_df_drop, x='Price', bins=100, ax=ax1)
ax1.set_title("union_df_drop")
sns.histplot(data=union_df_knn, x='Price', bins=100, ax=ax2)
ax2.set_title("union_df_knn")
plt.show()
```



Dummies

Método de clusterização (Análise multivariada)

- Testar inicialmente KNN

```
cat_features = ['Manufacturer', 'Model', 'Category', 'Leather_interior', 'Fuel_type', 'Gear_box_type', 'Drive_wheels', 'Wheel', 'Color', 'Exist_Turbo']
num_features = ['Price', 'Levy', 'Prod_year', 'Engine_volume', 'Mileage', 'Cylinders', 'Doors', 'Airbags', 'Age', 'Mileage_per_Age', 'last_price']
```

```
for col in cat_features:
    print(f'{col} - {union_df[cat_features][col].nunique()}')
```

```
Manufacturer - 65
Model - 1598
Category - 11
Leather_interior - 2
Fuel_type - 7
Gear_box_type - 4
Drive_wheels - 3
Wheel - 2
Color - 16
Exist_Turbo - 2
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

```
def dummies(df_union):
    df = df_union.copy()
    # Encode Labels
    df['Manufacturer'] = le.fit_transform(df['Manufacturer'])
    df['Model'] = le.fit_transform(df['Model'])
    df['Color'] = le.fit_transform(df['Color'])
    # Outras colunas categóricas
    df = pd.get_dummies(df)
    return df
```

```
df_dummies_drop = dummies(union_df_drop)
df_dummies_knn = dummies(union_df_knn)
```

KNN

```
from pyod.models.knn import KNN
```

```
def knn_outliers(df_dummies, df_orig):
    x = df_dummies.drop(columns=['ID'], axis=1).copy()

    # Treinar KNN
    clf = KNN()
    clf.fit(x)
    # Obter labels e número de outliers
    y = clf.labels_ # binary labels (0: inliers, 1: outliers)
    #Saídas
    print(np.unique(y, return_counts=True))
    print(f"Shape antes: {df_orig.shape}")

    outliers = []
    for i in range(len(y)):
        if y[i] == 1:
            outliers.append(i)

    outliers_df = df_orig.iloc[outliers,:]
    df_without_outliers = df_orig.drop(outliers_df.index)

    print(f"Shape depois: {df_without_outliers.shape}")
    return df_without_outliers
```

```
df_without_outliers_knn = knn_outliers(df_dummies_knn, union_df_knn)
```

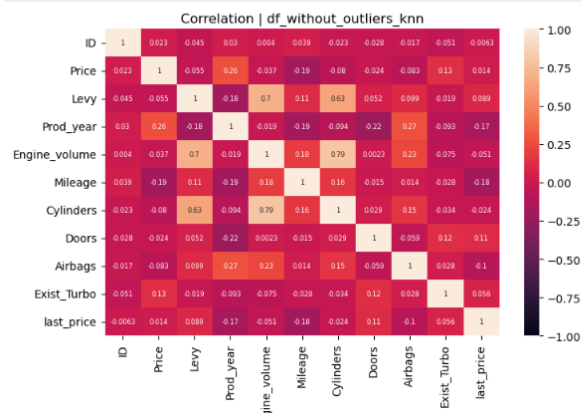
```
(array([0, 1]), array([16072, 1786], dtype=int64))
Shape antes: (17858, 20)
Shape depois: (16072, 20)
```

```
df_without_outliers_drop = knn_outliers(df_dummies_drop, union_df_drop)
```

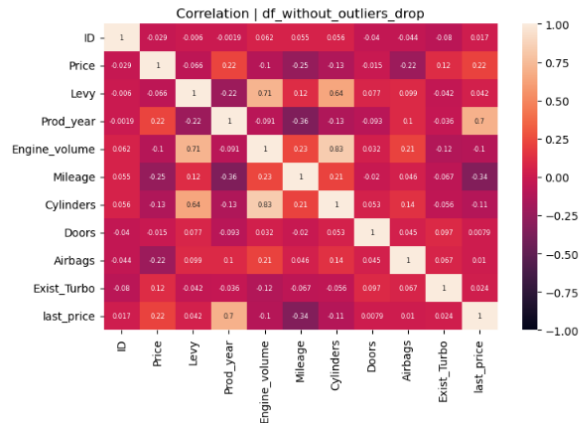
```
(array([0, 1]), array([11428, 1270], dtype=int64))
Shape antes: (12698, 20)
Shape depois: (11428, 20)
```

Numerical

```
corr = df_without_outliers_knn.corr(method='pearson')
plt.figure(figsize=(9,5))
plt.title(f'Correlation | df_without_outliers_knn')
sns.heatmap(corr, annot=True, vmin=-1, vmax=1, annot_kws={"size":6})
plt.show()
```



```
corr = df_without_outliers_drop.corr(method='pearson')
plt.figure(figsize = (8,5))
plt.title('Correlation | df_without_outliers_drop')
sns.heatmap(corr, annot = True, vmin=-1, vmax=1, annot_kws={"size":6})
plt.show()
```



Categorical

```
for col in cat_features:
    df = df_without_outliers_drop
    var = 'Price'
    # ANOVA
    category_group = df.groupby(col)[var].apply(list)
    anova = f_oneway(*category_group)
    print(f"P-Value para variavel {col} é: ",{anova[1]})
    # BOXPLOT
    plt.figure(figsize=(20,8))
    plt.title(f'{col} | {var}')
    sns.boxplot(data = df, x = col, y = var, showfliers = False);
    plt.show();
```

Feature selection

```
features_selected = ['Price', 'Manufacturer', 'Prod_year', 'Category',
                    'Fuel_type', 'Mileage', 'Gear_box_type', 'Drive_wheels',
                    'Wheel', 'Color', 'Airbags', 'Exist_Turbo', 'last_price']
```

```
df_features_selected = df_without_outliers_drop[features_selected]
```

```
df_features_selected.columns
```

```
Index(['Price', 'Manufacturer', 'Prod_year', 'Category', 'Fuel_type',
      'Mileage', 'Gear_box_type', 'Drive_wheels', 'Wheel', 'Color', 'Airbags',
      'Exist_Turbo', 'last_price'],
      dtype='object')
```

Multicolinearidade

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

def get_VIF(dataFrame, target):
    X = add_constant(dataFrame.loc[:, dataFrame.columns != target])
    seriesObject = pd.Series([variance_inflation_factor(X.values,i) for i in range(X.shape[1])], index=X.columns,
                             return seriesObject)

target = 'Price'
print(get_VIF(df_features_selected.select_dtypes(['int64', 'float64']),target))

const          996257.431878
Prod_year      2.054348
Mileage        1.173596
Airbags        1.023432
last_price     1.999927
dtype: float64
```

Dummies model

```
cat_columns = df_features_selected.select_dtypes(['object', 'category']).columns.tolist()

for col in cat_columns:
    print(f'{col} - {df_features_selected[col].nunique()}')
```

```
Manufacturer - 45
Category - 10
Fuel_type - 6
Gear_box_type - 4
Drive_wheels - 3
Wheel - 2
Color - 16
```

```
# Encode Labels
df_features_selected['Manufacturer'] = le.fit_transform(df_features_selected['Manufacturer'])
df_features_selected['Color'] = le.fit_transform(df_features_selected['Color'])

# Outras colunas categóricas
df_model_dummies = pd.get_dummies(df_features_selected)

df_model_dummies.info()
```

Normalize features

```
from sklearn.preprocessing import StandardScaler

col_nb = df_model_dummies.select_dtypes(['int64', 'float64']).columns.tolist()

scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_model_dummies[col_nb])

df_normalize = df_model_dummies.copy()
df_normalize[col_nb] = scaled_data
```

Modeling

Split (Train | Test)

```
X = df_normalize.drop(columns=['Price'])
y = df_normalize['Price']

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 42,
                                                    shuffle = True)

print('X_train:', X_train.shape)
print('X_test:', X_test.shape)
print('y_train:', len(y_train))
print('y_test:', len(y_test))

X_train: (7999, 32)
X_test: (3429, 32)
y_train: 7999
y_test: 3429
```

Random Forest Regressor

Tuning

```
n_estimators = [10, 30, 60, 100, 500, 1000]
max_features = ['auto', 'sqrt']
max_depth = [5, 10, 15, 25, 50]
min_samples_split = [5, 10, 20]
min_samples_leaf = [1, 2, 4]
bootstrap = ['True', 'False']

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'bootstrap': bootstrap}

rfr = RandomForestRegressor(random_state=42)
rfr_tuned = RandomizedSearchCV(estimator = rfr,
                              param_distributions=random_grid,
                              n_iter=100,
                              cv=3, verbose=2,
                              n_jobs=-1,
                              random_state=42)

mdl_rfr = rfr_tuned.fit(X_train, y_train)
print(rfr_tuned.best_params_)
```

Model

```
# Fit do modelo
rfr_tuned_final = RandomForestRegressor(n_estimators = 1000,
                                       min_samples_split = 5,
                                       max_features = 'auto',
                                       max_depth = 25,
                                       bootstrap = 'True',
                                       random_state = 42)

mdl_rfr_final = rfr_tuned_final.fit(X_train, y_train)

# Predição do teste
y_pred_rfr = mdl_rfr_final.predict(X_test)
y_true_rfr = y_test.tolist()

# Métricas de validação
r2_rfr = r2_score(y_true_rfr, y_pred_rfr)
rmse_rfr = mean_squared_error(y_true_rfr, y_pred_rfr)**(1/2)
mae_rfr = mean_absolute_error(y_true_rfr, y_pred_rfr)
```

ElasticNet Regression

```
from sklearn.linear_model import ElasticNet
```

Tuning

```
# Tuning do modelo
from sklearn.linear_model import ElasticNetCV

results = ElasticNetCV(cv = 10).fit(X_train, y_train)

results.alpha_
```

```
0.0013199477789637567
```

Model

```
# Let's create the final model according to optimum alpha.
enet_tuned = ElasticNet(alpha = results.alpha_).fit(X_train, y_train)

# Predict test data
y_pred_enet_tuned = enet_tuned.predict(X_test)
y_true_enet = y_test.tolist()

# Metrics RFR Predicts
r2_enet_tuned = r2_score(y_test, y_pred_enet_tuned)
rmse_enet_tuned = mean_squared_error(y_test, y_pred_enet_tuned)**(1/2)
mae_enet_tuned = mean_absolute_error(y_test, y_pred_enet_tuned)
```

XGBRegressor

```
from xgboost import XGBRegressor
```

Tuning

```
import warnings
warnings.filterwarnings('ignore')

n_estimators = [10, 30, 60, 100, 500, 1000]
max_depth = [6, 10, 15, 25]
eta = [0.1, 0.3, 0.5]
subsample = [1, 2, 3, 4]
colsample_bytree = [1, 2, 3]

param = {'max_depth': max_depth,
         'n_estimators': n_estimators,
         'eta': eta,
         'subsample': subsample,
         'colsample_bytree': colsample_bytree}

xb = XGBRegressor(random_state=42)
xb_tuned = RandomizedSearchCV(estimator=xb,
                             param_distributions=param,
                             n_iter=100,
                             cv=3,
                             verbose=2,
                             n_jobs=-1,
                             random_state=42)

mdl_xb = xb_tuned.fit(X_train, y_train)
print(xb_tuned.best_params_)
```

Model

```
# Fit tuned model
xb_tuned_final = XGBRegressor(subsample = 1,
                              n_estimators = 500,
                              max_depth = 10,
                              eta = 0.3,
                              colsample_bytree = 1)

mdl_xb_final = xb_tuned_final.fit(X_train, y_train)

# Predict test data
y_pred_xb = mdl_xb_final.predict(X_test)
y_true_xb = y_test.tolist()

# Metrics RFR Predicts
r2_xb = r2_score(y_true_xb, y_pred_xb)
rmse_xb = mean_squared_error(y_true_xb, y_pred_xb)**(1/2)
mae_xb = mean_absolute_error(y_true_xb, y_pred_xb)
```