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

Karen Yasmin de Oliveira Vicente

BOM NEGÓCIO - ANÚNCIO DE VENDAS DE CARROS

Belo Horizonte
2021

Karen Yasmin de Oliveira Vicente

BOM NEGÓCIO - ANÚNCIO DE VENDAS 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
2021

SUMÁRIO

1. Introdução

1.1. Contextualização

É inegável o grande avanço da tecnologia em todos os lugares, e com isso um grande aumento no volume de dados gerados em diversos setores, principalmente na área comercial. Essas informações disponibilizadas por toda parte torna-se essencial para as empresas, auxiliando em: tomadas de decisões, criação de Buyer-personas, desenvolvimento do mix de produtos, auxílio em campanhas publicitárias mais eficientes entre outros. (TRANSFORMAÇÃO DIGITAL, 2019).

Diante desses fatos diversas empresas na área comercial buscam investir em tecnologia para aumentar a eficiência, lucro e um diferencial nas suas vendas pela internet. O setor de vendas pela internet é um grande exemplo, estão cada vez mais inovando em tecnologias para agradar seus clientes. Os sites de marketplace estão investindo em marketing nos anúncios, abordagem tecnológica como em redes sociais por exemplo, algoritmos de recomendações para encontrar melhores opções de acordo com o perfil de cada cliente. Desta forma, a organização tem mais chances de identificar oportunidades lucrativas ou até mesmo de evitar riscos desconhecidos.

A raspagem de dados é uma das principais ferramentas para obter dados relevantes e realizar um mapeamento profundo para ajudar na tomada de decisões. Junto com ele também está associado algoritmos de Machine Learning, que por sua vez permite a criação de modelos que analisam conjuntos de dados e aprendem a reconhecer padrões ou a realizar predições. Com isso é possível analisar maiores volumes de dados de modo rápido e automático. (MEDIUM, 2020).

Diante deste contexto, este trabalho tem como objetivo aplicar técnicas de Mineração de Dados e Modelos Preditivos em Classificação a fim de classificar e encontrar padrões nos anúncios do vendedor, e identificá-los como um “Bom” ou “Mau” negócio para o cliente.

1.2. O problema proposto

Neste trabalho será utilizado Análise Exploratória e Modelagem Preditiva para extração de informações importantes dos anúncios de vendas de carros com o objetivo de realizar uma análise e encontrar padrões para classificá-lo como “Bom” ou “Mau” negócio para o comprador. Para isso todos os atributos serão classificados com um grau de importante, desta maneira conseguimos analisar os resultados e utilizá-los em predições futuras.

Para isso, serão analisados os anúncios de vendas de carros, disponibilizados no site da Webmotors. Os principais objetivos dessa análise são:

- Realizar uma análise nos dados dos anúncios de vendas de carros. Desta forma auxiliaremos e removemos a responsabilidade total de análise do comprador (no caso o usuário) em verificar qual o melhor negócio para compra.
- Os dados que serão analisados, foram coletados do site da WebMotors. Foi necessário coletar algumas informações separadamente, são elas:
    1. *Dataset dos anúncios:* neste dataset é apresentado informações sobre os anúncios das vendas, contendo informações como: preço de venda, descrição do anúncio, informações básicas do carro (modelo, ano, quilometragem, etc.).
    2. *Dataset dos valores da tabela Fipe do carro*: neste dataset é apresentado o valor da tabela Fipe do carro correspondente ao anúncio.
- As análises realizadas, têm como objetivo encontrar padrões, métricas e tendências que auxiliarão no entendimento das bases trabalhadas. E assim poderemos indicar quais características do carro faz dele um bom ou um negócio ruim .
- Em relação a aspectos geográficos, os resultados obtidos destinam-se exclusivamente a vendas online ocorridas no site da Webmotors.
- Os dados coletados são do período em tempo real, ordenados pela data de publicação decrescente.

[1] https://www.webmotors.com.br/api/

[2] https://www.webmotors.com.br/api/detail/car/

2. Coleta de Dados

Os dados coletados foram extraídos do site da Webmotors, utilizando a API disponibilizada por eles, onde as informações são retornadas em formato *json*. Para obter esses dados foi criado o arquivo ***script.py*** disponibilizado no mesmo link onde encontra-se o projeto. Ele foi desenvolvido utilizando a linguagem python, onde é realizado a chamada na API da Webmotors e os dados obtidos são salvos em um arquivo no formato *json* na pasta de datasets. Como observação, os dados já vieram formatados em *json*, por este motivo não foi necessário nenhuma conversão de formato. Abaixo é apresentado o código python disponibilizado no arquivo ***script.py***.

Figura 1: Arquivo script.py

```
#coding: utf-8

import json

from urllib.request import urlopen # Faz a requisição no servidor e
obtem a resposta
import urllib.error

# Definindo variáveis da url de busca
urlBase = 'https://www.webmotors.com.br/api/';
urlDetails = urlBase + 'detail/car/'

print('Obtendo os dados, aguarde!');

# Pegando dados
data = [];
for i in range(1, 500):
    url = urlBase +
'search/car?url=https://www.webmotors.com.br/carros%2Fsp%3Festadoci
dade%3DS%25C3%25A3o%2520Paulo%26tipoveiculo%3Dcarros&actualPage='+s
tr(i)

    # Exibir erro caso tenha problemas para obter os dados
    try:
        data += json.load(urlopen(url))['SearchResults'];
        print('Dados da Page: %s obtidos com sucesso, aguarde a
criação do arquivo!' % i)
```

```
    except urllib.error.HTTPError as e:
        print('Erro ao obter dados!' + e)
    except urllib.error.URLError as e:
        print('Erro ao obter dados!' + e)

data_fipe = []
for d in data:
    print("Obtendo dados do UniqueId: %d" % d["UniqueId"])

    dataDetails = None

    # Pega informações detalhadas de cada carro (necessário para
obter o valor da tabela FIPE)
    try:
        responseDetails = urlopen(urlDetails + str(d["UniqueId"]))
        dataDetails = json.load(responseDetails)
    except urllib.error.HTTPError as e:
        print('Erro ao obter dados!' + e)
    except urllib.error.URLError as e:
        print('Erro ao obter dados!' + e)

    fipe = {}
    fipe["Fipe"] = 0

    if dataDetails is None or "UniqueId" not in dataDetails:
        fipe["UniqueId"] = None
    else:
        fipe["UniqueId"] = dataDetails["UniqueId"]

    if dataDetails is not None and "Specification" in dataDetails
and "Evaluation" in dataDetails["Specification"]:
        fipe["Fipe"] =
dataDetails["Specification"]["Evaluation"]["FIPE"]

    data_fipe.append(fipe)

    # Parâmetros necessários para categorização
    # Dados bases necessários para a análise do modelo
    d["IPVApaid"] = False
    d["Licensed"] = False
    d["Warranty"] = False
    d["OnlyOwner"] = False
```

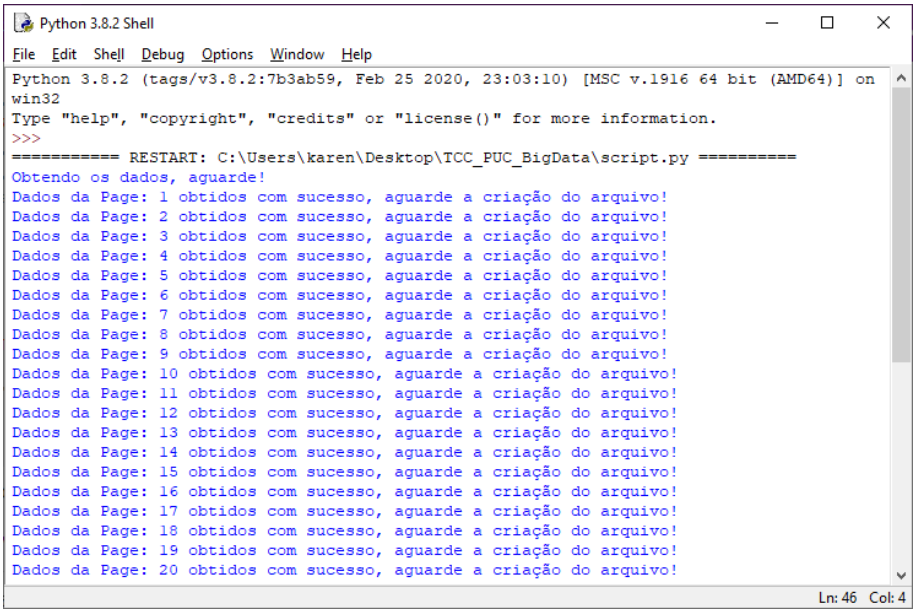
```
    if "Specification" in d and "VehicleAttributes" in
d["Specification"]:
        for atrr in d["Specification"]["VehicleAttributes"]:
            if (atrr["Name"] == "IPVA pago"):
                d["IPVApaid"] = True

            if (atrr["Name"] == "Licenciado"):
                d["Licensed"] = True

            if (atrr["Name"] == "Garantia de fábrica"):
                d["Warranty"] = True

            if (atrr["Name"] == "Único dono"):
                d["OnlyOwner"] = True

# Salvando os dados obtidos em um arquivo formato json
with open('data/data-cars.json', 'w', encoding='utf-8') as f:
    json.dump(data, f, ensure_ascii=False, indent=4)

# Salvando os dados FIPE obtidos em um arquivo formato json
    with open('data/data-cars-fipe.json', 'w', encoding='utf-8') as
f:
        json.dump(data_fipe, f, ensure_ascii=False, indent=4)

print('Dados salvos com sucesso!');
```

Fonte: Autor

Foi utilizado o pacote *urlib* do python para realizarmos as *requests* no site da Webmotors e obtermos as respectivas respostas em formato json.

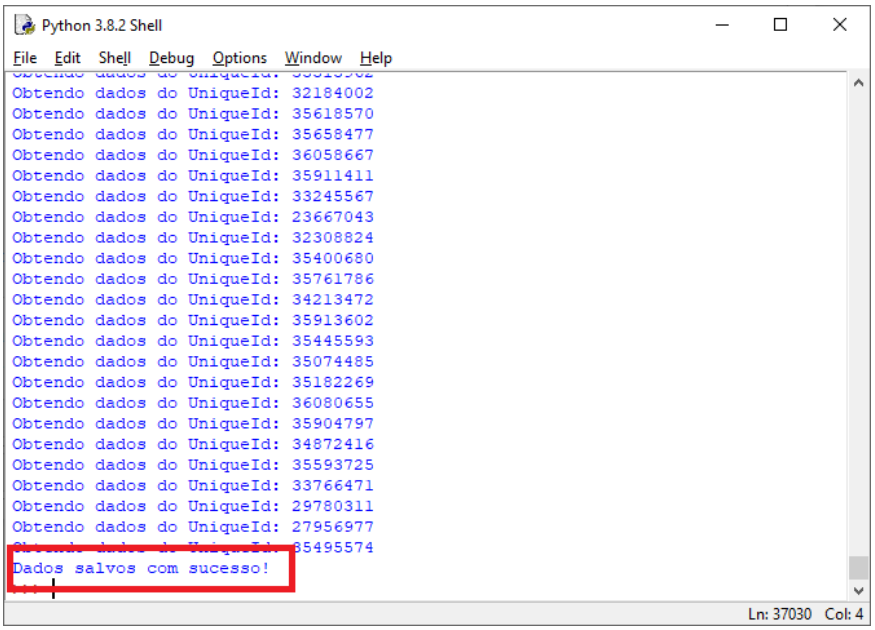
A Figura 2 abaixo, é o resultado da execução do *script.py*.

Figura 2: Arquivo script.py



Fonte: Autor

Figura 3: Arquivo script.py



Fonte: Autor

2.1 Dataset: data-cars.json

O Dataset *data-cars.json* contém informações referentes ao anúncio dos carros selecionados pelo script em python. Ele contém a seguinte estrutura para cada anúncio encontrado:

Figura 4: Dataset *data-cars.json*


```
[
 {
   "UniqueId": 34283122,
   "Media": {
     "Photos": [
       {
         "PhotoPath":
"2020\\202009\\20200925\\land-rover-discovery-sport-2.0-16v-td4-turbo-diesel-hse-4p-automatico-WMIMAGEM13485369695.jpg",
         "Order": 1
       },
       {
         "PhotoPath":
"2020\\202009\\20200925\\land-rover-discovery-sport-2.0-16v-td4-turbo-diesel-hse-4p-automatico-WMIMAGEM13485349115.jpg",
         "Order": 2
       },
       {
         "PhotoPath":
"2020\\202009\\20200925\\land-rover-discovery-sport-2.0-16v-td4-turbo-diesel-hse-4p-automatico-WMIMAGEM1348529746.jpg",
         "Order": 3
       },
       {
         "PhotoPath":
"2020\\202009\\20200925\\land-rover-discovery-sport-2.0-16v-td4-turbo-diesel-hse-4p-automatico-WMIMAGEM13485374775.jpg",
         "Order": 4
       },
       {
         "PhotoPath":
"2020\\202009\\20200925\\land-rover-discovery-sport-2.0-16v-td4-turbo-diesel-hse-4p-automatico-WMIMAGEM1348578526.jpg",
         "Order": 5
       },
       {
         "PhotoPath":
"2020\\202009\\20200925\\land-rover-discovery-sport-2.0-16v-td4-turbo-diesel-hse-4p-automatico-WMIMAGEM13485420725.jpg",
         "Order": 6
```

```
        },
        {
          "PhotoPath":
"2020\\202009\\20200925\\land-rover-discovery-sport-2.0-16v-td4-turbo-diesel-hse-4p-automatico-WMIMAGEM13485390426.jpg",
          "Order": 7
        },
        {
          "PhotoPath":
"2020\\202009\\20200925\\land-rover-discovery-sport-2.0-16v-td4-turbo-diesel-hse-4p-automatico-WMIMAGEM13485504239.jpg",
          "Order": 8
        }
      ]
    },
    "PhotoPath":
"2020\\202009\\20200925\\land-rover-discovery-sport-2.0-16v-td4-turbo-diesel-hse-4p-automatico-WMIMAGEM13485369695.jpg",
    "Specification": {
      "Title": "LAND ROVER DISCOVERY SPORT 2.0 16V TD4 TURBO DIESEL HSE 4P AUTOMÁTICO",
      "Make": {
        "id": 23,
        "Value": "LAND ROVER"
      },
      "Model": {
        "id": 3516,
        "Value": "DISCOVERY SPORT"
      },
      "Version": {
        "id": 346910,
        "Value": "2.0 16V TD4 TURBO DIESEL HSE 4P AUTOMÁTICO"
      },
      "YearFabrication": "2018",
      "YearModel": 2018,
      "Odometer": 23000,
      "Transmission": "Automática",
      "NumberPorts": "4",
      "BodyType": "Utilitário esportivo",
      "VehicleAttributes": [
        {
          "Name": "Aceita troca"
```

```
        },
        {
          "Name": "Todas as revisões feitas pela agenda do carro"
        },
        {
          "Name": "Único dono"
        },
        {
          "Name": "Todas as revisões feitas pela concessionária"
        },
        {
          "Name": "IPVA pago"
        },
        {
          "Name": "Licenciado"
        },
        {
          "Name": "Garantia de fábrica"
        }
      ],
      "Armored": "N",
      "Color": {
        "IdPrimary": "30405",
        "Primary": "Cinza"
      }
    },
    "Seller": {
      "Id": 3456580,
      "SellerType": "PF",
      "City": "Santos",
      "State": "São Paulo (SP)",
      "AdType": {
        "id": 4,
        "Value": "Pessoa Física"
      },
      "BudgetInvestimento": 0,
      "DealerScore": 0,
      "CarDelivery": false,
      "TrocaComTroco": false,
      "ExceededPlan": false
    },
    "Prices": {
```

```
      "Price": 219999,
      "SearchPrice": 219999
    },
    "ListingType": "U",
    "ProductCode": "674",
    "Channels": [
      {
        "id": 48,
        "Value": "Webmotors"
      }
    ],
    "LongComment": "? LAND ROVER - DIESEL\n????????????????\n?? Land
Rover Discovery Sport HSE 2.0 SD4\n????????????????\n?? Impecável
??\n??Ano 2018.\n??Apenas 19.000km.\n??GARANTIA ESTENDIDA ATE
2023\n??Airbags frontais / laterais / cortina\n??Alarme antifurto
perimétrico / volumétrico\n??Câmera de ré\n??Controle de tração /
estabilidade\n??Faróis de xenônio e de neblina \n??Faróis com
regulagem de altura\n??Faróis com refletores duplos\nUNICO DONO SEM
RETOQUES TODAS AS REVISÕES NA CS. 13-98142-1111 RENDEIRO
AUTOMOVEIS.\n",
    "FipePercent": 107,
    "IPVApaid": true,
    "Licensed": true,
    "Warranty": true,
    "OnlyOwner": true
  }
]
```


Fonte: Autor

2.1.1 Listagem descritiva das colunas:

A tabela abaixo mostra uma descrição da estrutura encontrada no Dataset *data-cars.json*, exibindo o nome do atributo, tipo e uma breve descrição da sua finalidade.

Tabela 1: Estrutura Dataset data-cars.json

Nome	Tradução	Tipo	Descrição
UniqueId	ID Único	Integer	Identificador único do anúncio.

Media	Mídia	Object	Contém um objeto com as imagens do carro anunciado.
PhotoPath	Diretório Imagem	String	Contém o diretório da imagem principal do carro no anúncio.
Specification	Especificação	Object	Contém um objeto com toda a especificação do carro, como: Título, Marca, Modelo e Versão.
Seller	Venda	Object	Contém informações da venda, como: Cidade, Estado, o Tipo de Vendedor etc.
Prices	Preços	Object	Contém informações relacionadas aos preços, como: Preço da venda, Preço antigo (caso o vendedor tenha alterado o preço).
ListingType	Tipo de Listagem	String	Tipo de listagem (ordenação dos registros).
ProductCode	Código do Produto	String	Código do produto.
Channels	Canais	Object	Contém informações do canal de venda (origem do anúncio).
LongComment	Comentário Longo	LongText	Contém um comentário feito pelo vendedor sobre o carro.
FipePercent	Porcentagem Fipe	Integer	Porcentagem do valor do carro anunciado com o valor da tabela Fipe.
IPVApaid	IPVA Pago	Boolean	Informação se o IPVA do carro está pago (true ou falso).
Licensed	Licenciado	Boolean	Informação se o Licenciamento do carro está feito (true ou false).
Warranty	Garantia	Boolean	Informação se o carro está na garantia (true ou false).
OnlyOwner	Único Dono	Boolean	Informação se o carro é de um único dono (true ou falso).
GoodDeal	Bom Negócio	Boolean	Informação se a compra é um bom negócio (true ou false).

Fonte: Autor

2.2 Dataset: data-cars-fipe.json

O Dataset *data-cars-fipe.json* contém informações referentes ao anúncio dos carros selecionados pelo script em python. Ele contém a seguinte estrutura para cada anúncio encontrado:

Figura 5: Dataset data-cars-fipe.json

```
[
    {
        "Fipe": 206373.0,
        "UniqueId": 34283122
    }
]
```

Fonte: Autor

2.2.1 Listagem descritiva das colunas:

A tabela abaixo mostra uma descrição da estrutura encontrada no Dataset *data-cars-fipe.json*, exibindo o nome do atributo, tipo e uma breve descrição da sua finalidade.

Tabela 2: Estrutura Dataset data-cars-fipe.json

Nome	Tipo	Descrição
UniqueId	Integer	Identificador único do anúncio.
Fipe	Float	Contém o valor do carro na tabela Fipe.

Fonte: Autor

Podemos observar que em ambos os datasets foram mantidos os nomes dos atributos em inglês, garantindo assim uma padronização e internacionalização futura.

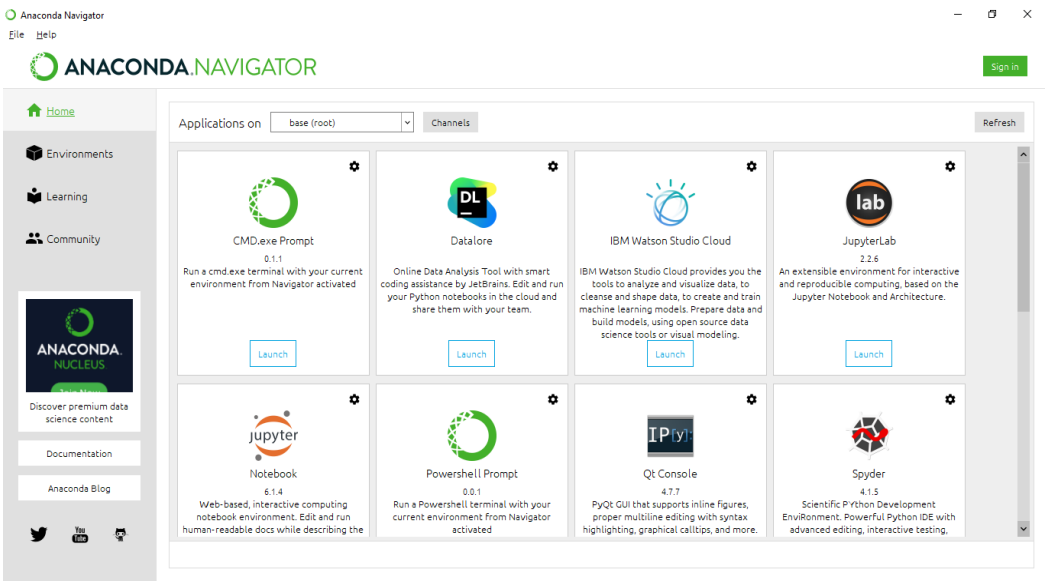
3. Processamento/Tratamento de Dados

Nessa seção será apresentado todas as ferramentas e bibliotecas utilizadas para o processamento e o tratamento dos dados.

3.1 Ferramentas utilizadas

Como ferramenta para desenvolvimento dos scripts em python, foi escolhida a distribuição Anaconda, versão 1.10.0 (figura 6), disponível em https://www.anaconda.com/distribution/.

Figura 6: Screenshot do Anaconda Navigator, da distribuição Anaconda

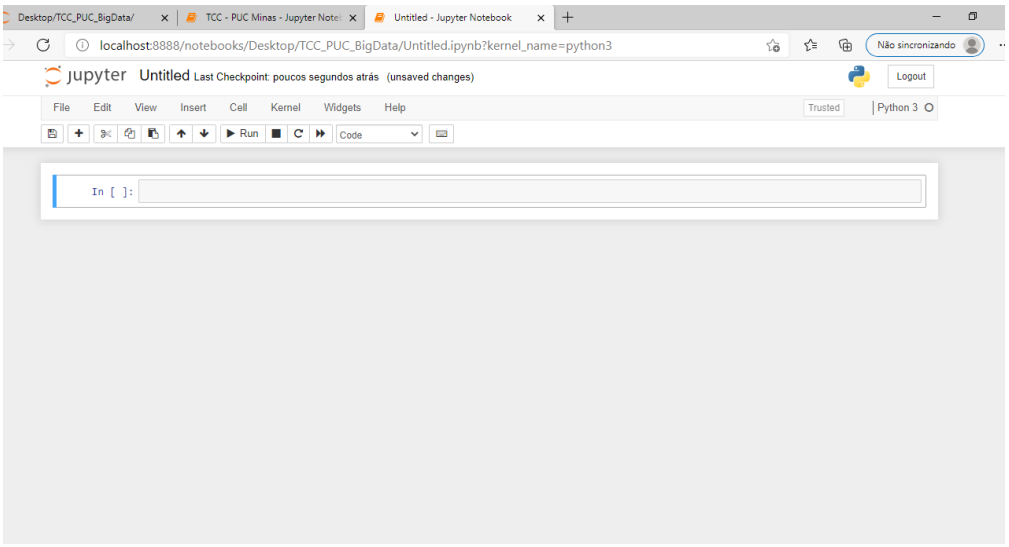


Fonte: Autor

Foi escolhida a distribuição *Anaconda* pois possui as principais ferramentas e bibliotecas para realizarmos toda a codificação necessária para análise e tratamento dos dados. Desta forma não é necessário realizar a importação dos pacotes separadamente, pois o *Anaconda* já nos fornece tudo em uma só instalação.

Utilizamos o Jupyter Notebook (figura 7) como principal editor deste projeto. Esta ferramenta nos permite unir código e texto, facilitando nossa organização no projeto. (MEDIUM, 2017).

Figura 7: Screenshot do editor Jupyter Notebook.



Fonte: Autor

O *Jupyter Notebook* possui o ambiente instalado e configurado com o python 3 (versão utilizada no projeto) incluso ao pacote do *Anaconda*.

3.1.2 Bibliotecas utilizadas

Para realizar o processamento e o tratamento dos dados, foi necessário importar algumas bibliotecas conforme a Figura 8 abaixo.

Figura 8: Screenshot importação das bibliotecas.

```
import json
import pandas as pd # lib pandas
import numpy as np # lib numpy
import datetime # lib datetime

import matplotlib.pyplot as plt # lib para utilização dos gráficos
import seaborn as sns # lib para exibir dados estátisticos

# Necessário para visualização automática dos gráficos no Jupyter
%matplotlib inline

from pathlib import Path
from urllib.request import urlopen # Faz a requisição no servidor e obtem a resposta
from pandas import json_normalize # package for flattening json in pandas df

# lib sklearn
from sklearn.model_selection import train_test_split # lib para definir os dados de treino e de teste
from sklearn.neighbors import KNeighborsClassifier # lib classe KNeighborsClassifier - Classificação
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn import preprocessing # Import LabelEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier # lib classe RandomForestClassifier - Classificação
```

Fonte: Autor

A tabela 3 abaixo mostra de forma detalhada as bibliotecas importadas.

Tabela 3: Bibliotecas utilizadas

Biblioteca	Descrição	Comando(s) utilizados
Json	Pacote de ferramentas para realizar processamento de dados em formatos Json, construída sobre a base da linguagem de programação python Informações: https://docs.python.org/3/library/json.html	`import json`
Pandas	Pacote de ferramentas para análise de dados e manipulação, construída sobre a base da linguagem de programação python. Informações: https://pandas.pydata.org/	`import pandas as pd`
Numpy	Pacote de ferramentas utilizada para realizar cálculos em Arrays Multidimensionais, construída sobre a base da linguagem de programação python. Informações: https://numpy.org/	`import numpy as np`
Datetime	Pacote de ferramenta que fornece classes para manipulação de datas, construída sobre a base da linguagem de programação python. Informações: https://docs.python.org/pt-br/3/library/datetime.html	`import datetime`
Matplotlib	Pacote de ferramentas utilizada para criação de gráficos e visualização de dados, construída sobre a base da linguagem de programação python.	`import matplotlib.pyplot as plt`

	Informações: https://matplotlib.org/	
Seaborn	Pacote de ferramentas utilizadas para criação de gráficos e visualização de dados de alto nível baseada na lib Matplotlib. Informações: https://docs.python.org/pt-br/3/library/datetime.html	`import seaborn as sns`
Pathlib	Pacote de ferramentas utilizadas para manipular diretórios do sistema. Informações: https://docs.python.org/3/library/pathlib.html	`from pathlib import Path`

Sklearn	Pacote de ferramentas utilizadas para trabalhar com Machine Learning, com ela são importados diversos métodos, algoritmos e técnicas para facilitar a codificação.
	
	Informações: https://scikit-learn.org/stable/
		`from sklearn.model_selection import train_test_split`
		
		`from sklearn.neighbors import KNeighborsClassifier`
		
		`from sklearn.metrics import accuracy_score, classification_report, confusion_matrix`
		
		`from sklearn import preprocessing`
		
		`from sklearn.model_selection import GridSearchCV`
		
		`from sklearn.ensemble import RandomForestClassifier`

Fonte: Autor

3.2 Obtendo dados

Nessa seção será apresentado como foi coletado os dados após a execução do ***script.py*** e salvo as informações em arquivo *Json*.

3.2.1 Obtendo dados: data-cars.json

Na Figura abaixo é apresentado a parte do código utilizada para obter os dados salvos em formato json e transformá-los em um formato Data Frame para iniciarmos a nossa análise.

Figura 9: Screenshot obtendo dados do arquivo data-cars.json

```
# Abrindo arquivo json de coleta dos dados na fonte
p = Path(r'C:\Users\karen\Desktop\TCC_PUC_BigData\data\data-cars.json')

# read json
with p.open('r', encoding='utf-8') as f:
    data = json.loads(f.read())

df_cars = json_normalize(data) # converte json
df_cars.head() # exibe DataFrame
```

Fonte: Autor

A seguir é apresentado o resultado em Data Frame dos dados coletados, selecionando apenas os 5 primeiros registros encontrados através do comando *df_cars.head()*.

Figura 10: Screenshot exibindo dados dataset do data-cars.json

	UniqueId	PhotoPath	ListingType	ProductCode	Channels	LongComment	FipePercent	IPVApaid	Licensed	Warrant
0	34283122	2020\202009\20200925\land-rover-discovery-spor...	U	674	[{'id': 48, 'Value': 'Webmotors'}]	? LAND ROVER - DIESEL\n??????????????? \n?? Lan...	107.0	True	True	Tru
1	9532158	2021\202103\20210306\hyundai-azera-3.3-mpfi-gl...	U	674	[{'id': 48, 'Value': 'Webmotors'}]	NaN	84.0	True	True	Fals
2	18996954	2017\201703\20170301\land-rover-discovery-4-3....	U	674	[{'id': 48, 'Value': 'Webmotors'}]	Veiculo com 7 lugares, único dono, 4 pneus nov...	135.0	True	True	Fals
3	35205229	2020\202012\20201213\land-rover-discovery-4-5....	U	674	[{'id': 48, 'Value': 'Webmotors'}]	NaN	82.0	True	True	Fals
4	36073749	2021\202103\20210301\volkswagen-jetta-2.0-tsi-...	U	674	[{'id': 48, 'Value': 'Webmotors'}]	NaN	88.0	True	True	Fals

Fonte: Autor

3.2.2 Obtendo dados: data-cars-fipe.json

Na Figura abaixo é apresentado a parte do código utilizada para obter os dados salvos em formato json e transformá-los em um formato Data Frame para iniciarmos a nossa análise.

Figura 11: Screenshot obtendo dados do arquivo data-cars-fipe.json

```
# Abrindo arquivo json de coleta dos dados da tabela Fipe
p = Path(r'C:\Users\karen\Desktop\TCC_PUC_BigData\data\data-cars-fipe.json')

# read json
with p.open('r', encoding='utf-8') as f:
    data = json.loads(f.read())

df_cars_fipe = json_normalize(data)  # converte json
df_cars_fipe.head() # exibe DataFrame
```

Fonte: Autor

Na Figura abaixo é apresentado o resultado em Data Frame dos dados coletados dos 5 primeiros registros.

Figura 12: Screenshot exibindo dados do arquivo data-cars-fipe.json

	Fipe	UniqueId
0	206373.0	34283122.0
1	31520.0	9532158.0
2	118552.0	18996954.0
3	91943.0	35205229.0
4	51408.0	36073749.0

Fonte: Autor

3.3 Informações dos DataFrames

Para cada objeto DataFrame foi utilizado a função *info()* para visualizarmos algumas informações como a quantidade de registros, quantidade de colunas, informações de cada coluna e o tipo dela.

Podemos observar na Figura abaixo, que no DataFrame df_cars foram encontrados 11976 registros com um total de 46 colunas.

Figura 13: Screenshot exibindo informações DataFrame *df_cars*

```
# Visualizando informações sobre os dados antes da formatação
df_cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11976 entries, 0 to 11975
Data columns (total 46 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   UniqueId                  11976 non-null  int64
 1   PhotoPath                 11886 non-null  object
 2   ListingType               11976 non-null  object
 3   ProductCode               11976 non-null  object
 4   Channels                  11976 non-null  object
 5   LongComment               11159 non-null  object
 6   FipePercent               11265 non-null  float64
 7   IPVApaid                  11976 non-null  bool
 8   Licensed                  11976 non-null  bool
 9   Warranty                  11976 non-null  bool
 10  OnlyOwner                 11976 non-null  bool
 11  Media.Photos              11886 non-null  object
 12  Specification.Title       11976 non-null  object
 13  Specification.Make.id     11976 non-null  int64
```

Fonte: Autor

Para o DataFrame com os valores da tabela Fipe, temos o seguinte resultado:

Figura 14: Screenshot exibindo informações DataFrame *df_cars_fipe*

```
# Visualizando informações sobre os dados antes da formatação
df_cars_fipe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11976 entries, 0 to 11975
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Fipe      11976 non-null  float64
 1   UniqueId  11920 non-null  float64
dtypes: float64(2)
memory usage: 187.2 KB
```

Fonte: Autor

Logo podemos ver que foram encontrados 11976 registros exatamente a quantidade do DataFrame dos anúncios, porém somente com duas colunas.

3.4 Tratamento dos Dados

Nessa seção será detalhado cada passo realizado no tratamento dos dados antes de começarmos as análises.

3.4.1 Unindo os DataFrames

Antes de realizarmos o tratamento em cada Dataset, para facilitar foi feita a união dos DataFrames. A figura 15 abaixo mostra a utilização da função *join* para unir os dos DataFrames pelo atributo *UniqueId.*

Figura 15: Screenshot unindo os DataFrame: df_cars e *df_cars_fipe*

```
#Unindo os datasets infos detalhes do carro
df_cars = df_cars.join(df_cars_fipe.set_index('UniqueId')[['Fipe']], on='UniqueId')
df_cars.head()
```

Fonte: Autor

3.4.2 Removendo as colunas

Após realizado análise em cima do DataFrame *df_cars,* verificou-se que existem colunas que não serão necessárias e não terão impacto em análises futuras. Desta forma essas colunas foram removidas conforme a Figura 16.

Figura 16: Remoção colunas DataFrame: df_cars

```
del df_cars['Channels'] # Coluna Channels - Canal de origem da
venda
del df_cars['ListingType'] # Coluna Tipo de Listagem
del df_cars['ProductCode'] # Coluna Código Produto
del df_cars['PhotoPath'] # Coluna PhotoPath
del df_cars['FipePercent'] # Coluna FipePercent
del df_cars['LongComment'] # Coluna LongComment - comentários do
vendedor

del df_cars['VipAutopago'] # Coluna VipAutopago

# Group - Media
del df_cars['Media.Photos'] # Coluna Media.Photos
del df_cars['Media.Videos'] # Coluna Media.Videos
```

```
# Group - Seller
del df_cars['Seller.Id'] # Coluna tipo de venda Id
del df_cars['Seller.SellerType'] # Coluna tipo de venda Id (PF, PJ)
del df_cars['Seller.AdType.id'] # Coluna tipo de venda Id (Pessoa
Física, Pessoa Jurídica)
del df_cars['Seller.AdType.Value'] # Coluna tipo de venda Value
(Pessoa Física, Pessoa Jurídica)
del df_cars['Seller.CarDelivery'] # Coluna Entrega do Carro (true,
false)
del df_cars['Seller.TrocaComTroco'] # Coluna Seller.TrocaComTroco
del df_cars['Seller.BudgetInvestimento'] # Coluna
BudgetInvestimento
del df_cars['Seller.DealerScore'] # Coluna DealerScore
del df_cars['Seller.City'] # Coluna Seller.City
del df_cars['Seller.State'] # Coluna Seller.State
del df_cars['Seller.ExceededPlan'] # Coluna Seller.ExceededPlan
#del df_cars['Prices.OldPrice'] # Coluna Prices.OldPrice

# Group - Prices
del df_cars['Prices.SearchPrice'] # Coluna Prices.SearchPrice valor
utilizado no campo busca do site

# Group - Specification
del df_cars['Specification.NumberPorts'] # Coluna NumberPorts
del df_cars['Specification.Transmission'] # Coluna Transmission
del df_cars['Specification.Make.id'] # Coluna Specification.Make.id
del df_cars['Specification.Version.id'] # Coluna
Specification.Version.id
del df_cars['Specification.Version.Value'] # Coluna
Specification.Version.Value
del df_cars['Specification.Model.id'] # Coluna Especificação do
Modelo Id
del df_cars['Specification.Color.IdPrimary'] # Coluna
Specification.Color.IdPrimary id da cor
del df_cars['Specification.VehicleAttributes'] # Coluna
Specification.VehicleAttributes (atributos já organizados em forma
de colunas)
del df_cars['Specification.YearFabrication'] # Coluna
Specification.YearFabrication, considerarmos o ano do Modelo do
carro
del df_cars['Specification.BodyType'] # Coluna BodyType
```

```
del df_cars['Specification.Title'] # Coluna Title
```

Fonte: Autor

3.4.2 Renomeando as colunas

Para deixar nossos dados formatados e padronizados, algumas colunas foram renomeadas utilizando a função *rename(),* conforme a Figura 17 abaixo.

Figura 17: Renomeando colunas DataFrame: df_cars

```
df_cars.rename({'Specification.Make.Value': 'Make',
                'Specification.Model.Value': 'Model',
                'Specification.Version.Value': 'Version',
                'Specification.YearModel': 'YearModel',
                'Specification.YearFabrication': 'YearFabrication',
                'Specification.Odometer': 'Odometer',
                'Specification.Armored': 'Armored',
                'Prices.Price': 'Price',
                'Prices.OldPrice': 'OldPrice',
                'Specification.Color.Primary': 'Color',
                'Specification.Armored': 'Armored',
               }, axis=1, inplace=True)
```

Fonte: Autor

Ao renomear removemos todos os sub-nós das colunas, deixando apenas o nome da coluna no nó principal.

3.4.2 Padronizando os tipos das colunas

Para deixar os tipos de cada coluna padronizados, foi realizada a conversão dos tipos em algumas colunas utilizando a função *astype()*, conforme Figura 18.

Figura 18: Padronizando colunas DataFrame: df_cars

```
# Convertendo o ano de Float64 para Int64 (obs: o ano modelo está
vindo como float 1 casa decimal)
df_cars['YearModel'] = df_cars['YearModel'].astype('int64')
```

```
# Convertendo dados que estão como object para string
df_cars['Make'] = df_cars['Make'].astype('string')
df_cars['Model'] = df_cars['Model'].astype('string')
df_cars['Armored'] = df_cars['Armored'].astype('string')
```

Fonte: Autor

O valor da coluna *YearModel* estava em *String,* foi convertido para um formato inteiro, correspondendo a quantidade de anos do modelo do carro.

As colunas *Make*, *Model* e *Armored* foram convertidas para string, pois corresponde a valor em texto.

3.4.3 Anúncios sem valor tabela Fipe

Para melhorar a precisão em nossas análises, consideramos apenas os registros de carros que encontraram o valor correspondente ao valor da tabela Fipe, ou seja, que possui um valor válido para a coluna *Fipe* (valor do carro na tabela Fipe).

A Figura X abaixo faz uma contagem total de registros onde o valor de *Fipe* seja igual a zero. Para isso, foi utilizado o comando *count()* no DataFrame df_cars.

Figura 19: Screenshot - Verificando registros inválidos da tabela Fipe

```
df_cars['Fipe'][df_cars['Fipe'] == 0].count()
```

```
711
```

Fonte: Autor

Temos como resultado 711 registros onde o valor *Fipe* está zerado.

Neste caso, removemos os registros encontrados do DataFrame principal e consideramos apenas os valores válidos para a coluna Fipe.

Figura 20: Screenshot - Verificando registros inválidos da tabela Fipe

```
# Seleciona os carros que estão com o valor Fipe em <= 0
df_cars_fipe_ok = df_cars[(df_cars['Fipe'].isnull()) | (df_cars['Fipe'] <= 0)]

# Remove os registros que estão com o Fipe <= e considera apenas os que estão com Fipe > 0
df_cars = df_cars.drop(df_cars_fipe_ok.index, axis=0)
```

Fonte: Autor

Os registros foram removidos utilizando o comando *drop()*, onde é passado por parâmetro o *index* da tabela e o valor de *axis.* Lembrando que o axis corresponde ao valor do eixo 0 ou 1, sendo 0 para linhas e 1 para colunas.

3.4.4 Anúncios sem ajuste de valor

O DataFrame *df_cars* possui uma coluna chamada *OldPrice* (Preço Antigo). Essa coluna armazena o valor antigo postado no anúncio do carro, caso o preço tenha sido ajustado pelo vendedor. Porém se o preço não tiver recebido ajuste, esta coluna vem como valor nulo na estrutura do dataset, neste caso consideramos que o *OldPrice* que não for alterado, receberá o mesmo valor atual do anúncio. Ou seja, receberá o mesmo valor da coluna *Price*. Ao fazermos desta forma, padronizamos a coluna, pois utilizaremos ela em análises futuras.

Figura 21: Definindo valores para a coluna OldPrice.

```
df_cars['OldPrice'].loc[df_cars['OldPrice'].isnull()] = df_cars['Price']
```

Fonte: Autor

A figura 21 mostra a utilização da função *loc()* para alterar o valor da coluna, onde selecionamos apenas os registros onde o *OldPrice* estiver nulo, recebendo o valor de *Price* do registro.

3.4.5 Anúncios sem valor para “Bom Negócio”

A coluna *GoodDeal* quando o anúncio não é identificado como “Bom Negócio”, ou seja, possui o valor *True*, esta coluna está vindo com um valor nulo. Neste caso consideramos que a coluna receberá o valor *False*.

Figura 22: Atribuindo valores para a coluna *GoodDeal*

```
df_cars['GoodDeal'].loc[df_cars['GoodDeal'].isnull()] = False
```

Fonte: Autor

3.4.6 Informações dos DataFrames

Para certificar que os dados estão padronizados e que não existem valores nulos no DataFrame, foi executado a função *isnull()* onde recupera todos os registros que estão com valores nulo e logo em seguida aplicado a função *sum()*, pois somamos a quantidade de registros nulos da coluna correspondente, conforme a Figura 23.

Figura 23: Screenshot - Definindo valores para a coluna OldPrice

```
#Verifica se existe algum valor nulo após o tratamento dos dados
df_cars.isnull().sum()
```

```
UniqueId     0
IPVApaid     0
Licensed     0
Warranty     0
OnlyOwner    0
Make         0
Model        0
YearModel    0
Odometer     0
Armored      0
Color        0
Price        0
OldPrice     0
Fipe         0
dtype: int64
```

Fonte: Autor

O resultado obtido apresenta a listagem de todas as colunas e o valor da soma de valores nulos obtidos. Neste caso não existe nenhuma coluna com registro nulo, por isso os valores aparecem zerados.

Para exibir as informações do DataFrame de forma detalhada, utilizamos a função *info()*, ela retorna informações importantes de cada coluna, tais como: nome da coluna, tipo da coluna e se a coluna aceita valores nulos, conforme a Figura 24 abaixo.

Figura 24: Screenshot - Informações detalhadas do DataFrame

```
# Visualizando informações sobre os dados já formatados
df_cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11209 entries, 0 to 11975
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   UniqueId   11209 non-null  int64
 1   IPVApaid   11209 non-null  bool
 2   Licensed   11209 non-null  bool
 3   Warranty   11209 non-null  bool
 4   OnlyOwner  11209 non-null  bool
 5   Make       11209 non-null  string
 6   Model      11209 non-null  string
 7   YearModel  11209 non-null  int64
 8   Odometer   11209 non-null  float64
 9   Armored    11209 non-null  string
 10  Color      11209 non-null  object
 11  Price      11209 non-null  float64
 12  OldPrice   11209 non-null  float64
 13  Fipe       11209 non-null  float64
dtypes: bool(4), float64(4), int64(2), object(1), string(3)
memory usage: 1.3+ MB
```

Fonte: Autor

Foi utilizado a função *shape(),* onde podemos visualizar as dimensões do DataFrame.

Figura 25: Screenshot - Informações dimensões do DataFrame

```
df_cars.shape
```

```
(11209, 14)
```

Fonte: Autor

Neste caso obtemos como resultado 11.209 linhas e 14 colunas.

4. Análise e Exploração dos Dados

Nessa seção será mostrado todas as análises e exploração dos dados tratados anteriormente. Analisaremos as ocorrências, padrões e informações importantes que levantamos do dataset.

4.1 Análise coluna *GoodDeal (Bom Negócio)*

A coluna *GoodDeal* (Bom Negócio), identifica se o anúncio é ou não um bom negócio.

Figura 26: Screenshot - Agrupamento Bom Negócio

	Bom Negócio	Quantidade	Porcentagem
0	Pode não ser um Bom Negócio	10466	93.371398
1	Bom Negócio	743	6.628602

Fonte: Autor

Podemos dizer que a maioria dos anúncios do dataset, se classificam como não sendo um Bom Negócio, sendo 10.466 e 743 como Bom negócio. Podemos ver um claro desbalanceamento na base.

4.2 Análise coluna *GoodDeal (Bom Negócio)* em relação aos outros atributos

Nessa seção será apresentado alguns dos resultados das análises feitas na Coluna GoodDeal relacionada aos outros atributos.

Para essa análise, consideramos 743 registros encontrados como *GoodDeal* recebendo um valor verdadeiro.

As análises encontradas conforme o arquivo notebook em python foram:

- Relação com a coluna *PriceFipeOk:* verificamos que foram encontrados 743 registros como GoodDeal, e curiosamente todos os registros estão com o Preço Fipe Ok.
- Relação com a coluna *OdometerRecommended*: foram encontrados 281 registros com a quilometragem recomendada recebendo um valor verdadeiro e considerados um Bom Negócio.
- Relação com a coluna *OldPrice*: não foi encontrado nenhum registro onde o preço inicial do anúncio tenha sido ajustado por um menor preço.

Conclui-se que as colunas com os valores da Tabela Fipe/Preço e quilometragem do carro, tem uma relação muito forte com a coluna de “Bom negócio”. Desta forma nas seções seguintes será realizado uma análise aprofundada nesses atributos.

4.3 Análise preço e valor na tabela Fipe

O valor do carro no anúncio está diretamente relacionado ao valor correspondente a tabela Fipe e consideramos como um atributo de extrema importância para nossa análise. Pois desta maneira podemos dizer se o valor está ou não na média do valor recomendado na tabela Fipe.

Foi feito uma análise detalhada nos valores dos carros anunciados e os respectivos valores da tabela Fipe correspondente ao modelo e ano do carro. Com essa análise será possível identificarmos se o valor da coluna *Fipe,* juntamente com o valor do IPVA pago, são atributos relevantes para ser considerado no nosso modelo de classificação.

A figura 27 mostra quantos registros do Dataframe que estão com o valor do anúncio abaixo ou igual ao valor da tabela Fipe e com IPVA Pago.

Figura 27: Screenshot - Carros Preço <= Valor Fipe

```
# Carros que estão abaixo da tabela FIPE e IPVA Pago
cars = (df_cars['Price'] <= df_cars['Fipe']) & (df_cars['IPVApaid'])

# Utilizando a função shape para pegar o tamanho de linhas do DataFrame com o filtro de carros
df_cars[cars].shape[0]
```

```
1304
```

Fonte: Autor

Neste caso utilizamos a condição: o valor do carro no anúncio tem que ser menor do que o valor do carro na tabela Fipe e o IPVA já pago, ou seja a coluna *IPVApaid* deverá ser verdadeira. Como resultado obtemos o valor de 1.304 em um total de 11.209, que satisfazem essa condição.

Consideremos que se o carro anunciado estiver com um valor abaixo ou igual ao valor da tabela Fipe, pode ser um “Bom negócio”, pois ele corresponde a um valor apropriado para venda, já que está entre o valor da tabela do carro e queremos filtrar os melhores preços para os compradores.

A Figura 28 exibe as colunas *Fipe* e *Price* de todos os carros que estão com o valor acima do valor na tabela Fipe.

Figura 28: Screenshot - Carros Preço > Valor Fipe

```
# Carros que estão acima da tabela FIPE
cars = (df_cars['Price'] > df_cars['Fipe'])
df_cars[cars][['Fipe', 'Price']]
```

	Fipe	Price
0	206373.0	219999.0
2	118552.0	159900.0
7	25991.0	29900.0
9	95229.0	115000.0
13	126460.0	134900.0
...	...	...
11970	58464.0	65990.0
11972	98242.0	118880.0
11973	48609.0	53990.0
11974	38175.0	42290.0
11975	40421.0	45190.0

9138 rows × 2 columns

Fonte: Autor

Podemos observar que são 9.138 registros encontrados com o valor acima do valor da Tabela Fipe.

4.4 Análise preço OK e carros com mais de 10 anos

Para darmos continuidade na análise dos dados, foram criadas duas colunas que são atributos fundamentais para fazer as comparações e encontrarmos padrões. São elas:

- *PriceFipeOK:* coluna do tipo Boolean (True - Sim e False - Não). Esta coluna identifica como um valor verdadeiro (True) caso o valor do carro no anúncio for menor ou igual ao valor correspondente ao valor da tabela Fipe.
- *Year>10Years:* coluna do tipo Boolean (True - Sim e False - Não). Esta coluna identificará como um valor verdadeiro (True) caso o carro possuir mais de 10 anos, caso contrário ela receberá falso (False).

A Figura 29 abaixo mostra como as colunas *PriceFipeOK* e a *Year>10Years* foram criadas.

Figura 29: Criando colunas *PriceFipeOk* e *Year>10Years*

```
# Essa coluna identifica se o valor do carro está acima ou abaixo
do valor da tabela Fipe (True-sim, False-não)
df_cars['PriceFipeOk'] = (df_cars['Price'] <= df_cars['Fipe'])

# Essa coluna identifica se o veículo tem mais de 10 anos (True-
sim, False-não)
now = datetime.datetime.now() # Pega a data atual

df_cars['Year>10Years'] = (now.year - df_cars['YearModel'] > 10) #
verifica se a diferença do ano é superior a 10

df_cars.head()
```

Fonte: Autor

Logo abaixo podemos obter o resultado do comando *head()*, exibindo os 5 primeiros registros do DataFrame.

Figura 30: Screenshot - Data Frame Atualizado

	UniqueId	IPVApaid	Licensed	Warranty	OnlyOwner	Make	Model	YearModel	Odometer	Armored	Color	Price	OldPrice	Fipe	I
0	34283122	True	True	True	True	LAND ROVER	DISCOVERY SPORT	2018	23000.0	N	Cinza	219999.0	219999.0	206373.0	
1	9532158	True	True	False	True	HYUNDAI	AZERA	2010	71496.0	N	Preto	26500.0	26500.0	31520.0	
2	18996954	True	True	False	True	LAND ROVER	DISCOVERY 4	2013	97411.0	N	Preto	159900.0	159900.0	118552.0	
3	35205229	True	True	False	True	LAND ROVER	DISCOVERY 4	2011	107200.0	S	Preto	75500.0	75500.0	91943.0	
4	36073749	True	True	False	True	VOLKSWAGEN	JETTA	2012	92000.0	N	Branco	45000.0	45000.0	51408.0	

Fonte: Autor

4.5 Resumo Estatístico do DataFrame

Para exibirmos um resumo estatístico de cada uma das colunas, utilizamos a função *describe()* e com isso teremos como resultado um Data Frame com as colunas: *count* (frequência), *mean* (média), *std* (desvio padrão), *min* (valor mínimo), *25%* (primeiro quartil), *50%* (mediana), *75%* (terceiro quartil) e *max* (valor máximo).

A figura X exibe o resultado obtido utilizando a função *describe()* para exibir as estatísticas descritivas das colunas.

Figura 31: Screenshot - Análise estatísticas das colunas

```
# Utilizando a função describe para exibir as Estatísticas descritivas das colunas
df_cars.describe().round(2)
```

	UniqueId	YearModel	Odometer	Price	OldPrice	Fipe
count	11209.00	11209.00	11209.00	11209.00	11209.00	11209.00
mean	34810534.04	2016.55	59045.87	110560.34	110561.41	83973.96
std	3719016.56	5.08	56160.48	998500.56	998500.50	95139.47
min	4369950.00	1985.00	0.00	6000.00	6000.00	2380.00
25%	35396411.00	2014.00	26980.00	47790.00	47790.00	43647.00
50%	35904828.00	2019.00	47064.00	66900.00	66900.00	61667.00
75%	36075268.00	2020.00	80700.00	99990.00	99990.00	93200.00
max	36156739.00	2022.00	2039882.00	75000000.00	75000000.00	3100000.00

Fonte: Autor

4.6 Análise Carros e Marcas

Foram feitas algumas análises mais profundas utilizando os atributos do carro e a sua respectiva marca na tentativa de verificar alguns padrões e quais marcas estão entre as mais anunciadas.

4.6.1 Agrupando Marca, Preço Médio e Quantidade

Agrupamos os registros pela Marca, Preço Médio e Quantidade, desta maneira conseguimos identificar quais as marcas mais populares em vendas de carros e quais a média de preço colocada. A figura 32 mostra o código utilizado para realizar o agrupamento desses registros, a alteração dos nomes das colunas para exibi-las e a ordenação pela marca.

Figura 32: Agrupamento Marca, Preço Médio e Quantidade

```
# Agrupa e exibe carros pelo preço médio de cada marca.
cars_group_make_mean = df_cars.groupby(['Make'], as_index=False)

# Exibe colunas:
# Model: valor único (agrupado)
# Preço: média de preço daquela Marca
cars_group_make_mean = cars_group_make_mean.agg({'Price':np.mean,
'UniqueId': np.size}).round(2)
```

```
# Renomeando as colunas para apresentá-las
cars_group_make_mean =
cars_group_make_mean.rename(columns={'Make':'Marca','Price':'Preço
Médio', 'UniqueId': 'Quantidade'})

# Ordenando em ordem Decrescente pela coluna modelo
cars_group_make_mean = cars_group_make_mean.sort_values('Marca',
ascending=True)

cars_group_make_mean
```

Fonte: Autor

Abaixo o resultado obtido ao exibir o Data Frame agrupado.

Figura 33: Screenshot - Resultado Agrupamento: marca, preço médio e quantidade

	Marca	Preço Médio	Quantidade
0	ALFA ROMEO	59708.33	12
1	ASTON MARTIN	520000.00	1
2	AUDI	181651.93	291
3	BMW	186277.03	346
4	CHERY	108409.11	125
5	CHEVROLET	60701.00	1432
6	CHRYSLER	49100.74	27
7	CITROËN	48193.03	253
8	DODGE	70395.83	36
9	FERRARI	1654750.00	4
10	FIAT	80108.93	952
11	FORD	67448.56	1327
12	GURGEL	18000.00	1
13	HAFEI	20500.00	1
14	HONDA	78518.15	285
15	HYUNDAI	69738.90	626
16	IVECO	196000.00	1
17	JAC	38004.38	16
18	JAGUAR	248645.68	37
19	JEEP	110898.31	516
20	JINBEI	32875.00	4
21	KIA	117432.64	211
22	LAMBORGHINI	2656666.67	3
23	LAND ROVER	168776.98	316
24	LEXUS	175936.25	16
25	LIFAN	63506.00	10
26	MASERATI	464950.00	4
27	MAZDA	58300.00	3
28	MCLAREN	3100000.00	1
29	MERCEDES-BENZ	404799.36	365
30	MG	53725.00	4
31	MINI	116118.58	38
32	MITSUBISHI	113433.91	201
33	NISSAN	84365.82	264
34	PEUGEOT	61017.77	334
35	PORSCHE	483092.32	94
36	RELY	24500.00	1
37	RENAULT	57972.29	815
38	SMART	63163.33	3
39	SSANGYONG	44722.50	4
40	SUBARU	82226.50	26
41	SUZUKI	89795.89	27
42	TOYOTA	191081.01	429
43	TROLLER	159602.63	19
44	VOLKSWAGEN	113048.32	1619
45	VOLVO	185615.99	109

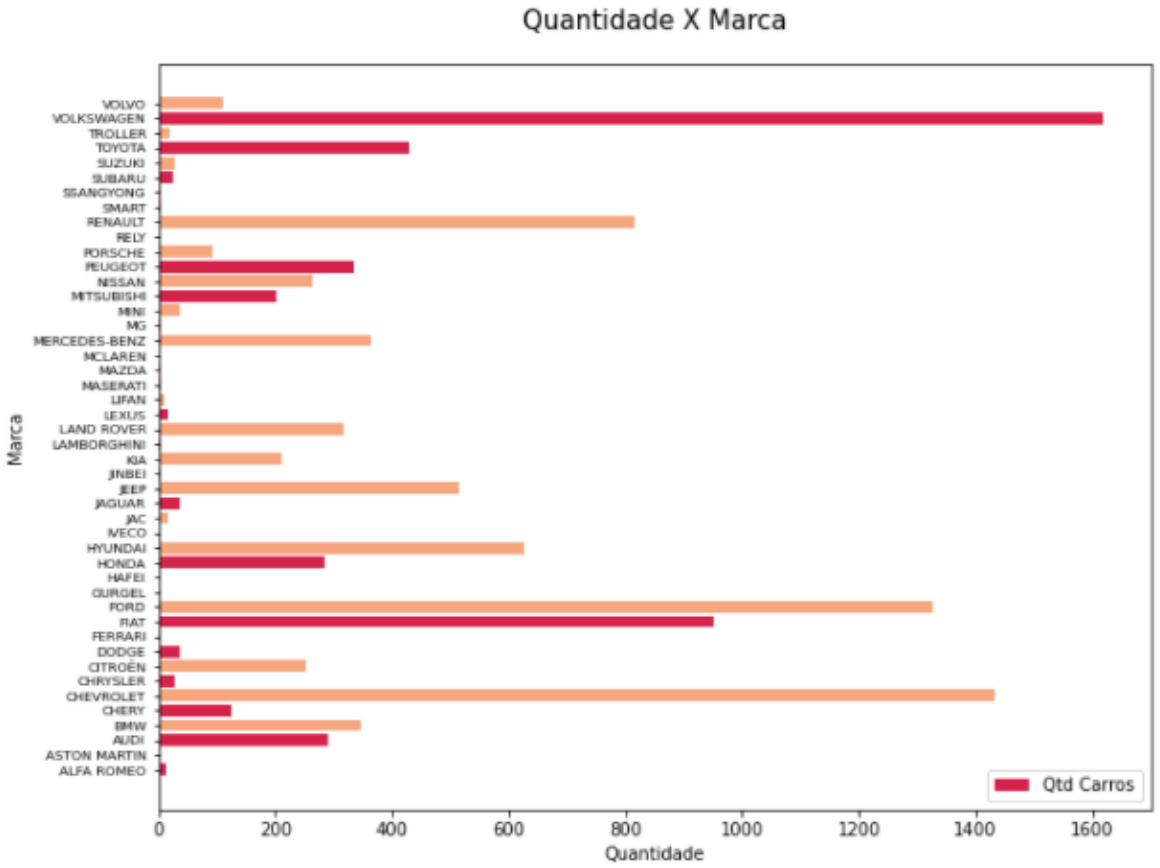
Fonte: Autor

Podemos dizer que neste caso a marca Volkswagen possui mais anúncios de carros no site da Webmotors, são 1.619 registros encontrados. Concluímos também que os carros com menos anúncios no site são geralmente de marcas importadas ou carros com modelos atípicos, tais como: Gurgel, Iveco, Rely e Aston Martin.

4.6.2 Gráfico Agrupamento: Marca e Quantidade

Para avaliarmos melhor os dados obtidos do agrupamento da seção anterior, criamos um gráfico de Marcas e Quantidades de carros encontrados de cada marca.

Figura 34: Screenshot - Gráfico Marca e Quantidade



Fonte: Autor

Para desenvolver os dados em forma gráfica, foi utilizado o seguinte código, exibido na Figura 35 abaixo.

Figura 35: Código - Gráfico Marca e Quantidade

```
# Gráfico Quantidade de Carros de cada Marca

# Pega o resultado do agrupamento (Preço Médio + Modelo) para
exibir no gráfico
fig = plt.figure(figsize=(8,6))
eixo = fig.add_axes([0, 0, 1, 1])

# Adicionando os Índices das marcas
indice = np.arange(len(cars_group_make_mean))

eixo.barh(indice, cars_group_make_mean['Quantidade'].round(),
align='center', height=0.8,
tick_label=cars_group_make_mean['Marca'], color=["#d5224a",
'#f6a67e'])

# Alterando Título e Labels
eixo.set_title('Quantidade X Marca', fontsize=15, pad=20)
eixo.set_xlabel('Quantidade')
eixo.set_ylabel('Marca')

eixo.set_yticks(indice)
eixo.set_yticklabels(cars_group_make_mean['Marca'], fontsize=7.8)

# Adicionando Legenda
eixo.legend(['Qtd Carros'], loc = 'lower right')
```

Fonte: Autor

Para construir o gráfico em barras, foi utilizado a biblioteca *Matpotlib* e a função *barth()* onde passamos por parâmetros os dados que iremos exibir. Também foi feita toda a customização do gráfico, como alteração das cores, títulos e legendas.

4.7 Análise carros onde o preço foi ajustado

Foi realizada uma análise na coluna *OldPrice,* ela representa os preços de carros que foram ajustados, identificando se o valor inicial do anúncio foi abaixado.

4.7.1 Agrupamento dos carros onde o preço foi ajustado

Realizamos um agrupamento por Marca e Quantidade, dos carros que tiveram uma alteração de preço para um valor menor ao inicial. A figura 36 mostra como foi feito esse agrupamento.

Figura 36: Agrupamento carros com preço ajustado

```
# Agrupa os carros por Marca, somente os que houve uma alteração do
preço para um menor valor

# Agrupa por Make
cars_group_make_old_price = df_cars[df_cars['OldPrice'] >
df_cars['Price']].groupby(['Make'], as_index=False)

# Exibe colunas:
# UniqueId: quantidade de registros
cars_group_make_old_price = cars_group_make_old_price.agg({
'UniqueId': np.size}).round(2)

# Renomeando as colunas para apresentá-las
cars_group_make_old_price =
cars_group_make_old_price.rename(columns={'Make':'Marca',
'UniqueId':'Quantidade'})

cars_group_make_old_price
```

Fonte: Autor

Logo abaixo podemos visualizar o resultado obtido, exibindo as colunas Marca e Quantidade.

Figura 37: Screenshot - Agrupamento Marca e Quantidade preços alterados

	Marca	Quantidade
0	AUDI	1
1	CHEVROLET	1
2	CITROËN	1
3	FIAT	1
4	FORD	1
5	HONDA	1
6	HYUNDAI	4
7	NISSAN	1
8	PEUGEOT	1
9	VOLKSWAGEN	1

Fonte: Autor

Podemos observar que houve poucos anúncios onde o preço inicial foi alterado para um valor menor. Os carros da marca Hyundai tiveram uma maior alteração, aparecendo 4 vezes nos registros.

4.7.2 Gráfico Agrupamento: carros onde o preço foi ajustado

Para exibirmos os dados agrupados anteriormente utilizamos também a exibição em barras conforme a Figura 38.

Figura 38: Gráfico - Agrupamento Marca e Quantidade

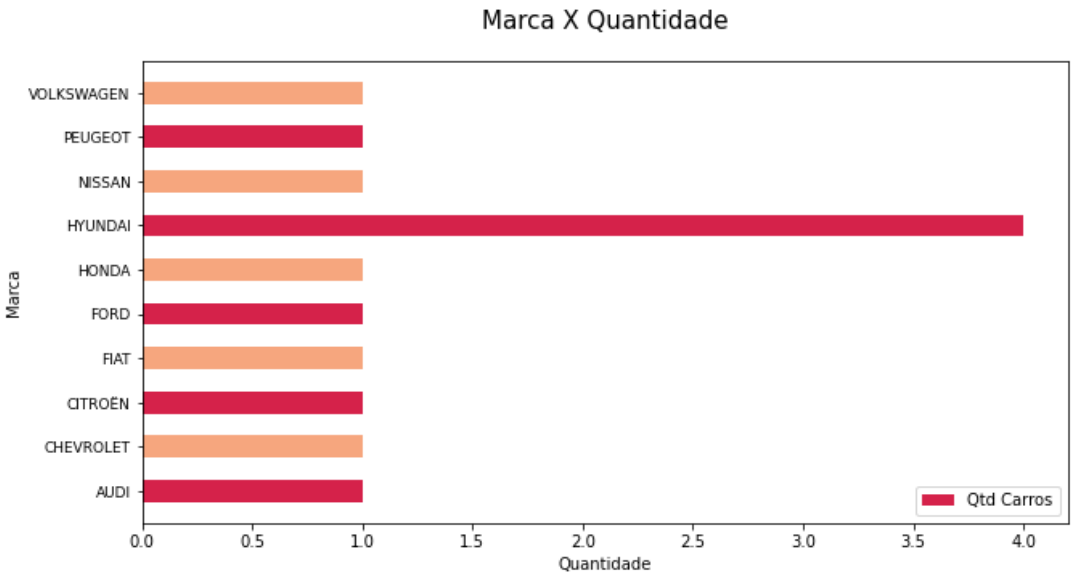
```
# Gráfico Marca X Quantidade

# Pega o resultado do agrupamento (Marca e Qtd de registros) para
exibir no gráfico
fig = plt.figure(figsize=(8,4))
eixo = fig.add_axes([0, 0, 1, 1])

# Adicionando os Índices das marcas
indice = np.arange(len(cars_group_make_old_price))

eixo.barh(indice, cars_group_make_old_price['Quantidade'].round(),
```

```
align='center', height=0.5,
tick_label=cars_group_make_old_price['Marca'], color=["#d5224a",
'#f6a67e'])

# Alterando Título e Labels
eixo.set_title('Marca X Quantidade', fontsize=15, pad=20)
eixo.set_xlabel('Quantidade')
eixo.set_ylabel('Marca')

eixo.set_yticks(indice)
eixo.set_yticklabels(cars_group_make_old_price['Marca'],
fontsize=9)

# Adicionando Legenda
eixo.legend(['Qtd Carros'], loc = 'lower right')
```

Fonte: Autor

O resultado obtido foi o seguinte:

Figura 39: Screenshot - Gráfico Marca e Quantidade



Fonte: Autor

4.8 Análise quilometragem dos carros

A quilometragem do carro também pode ser um atributo relevante a ser considerado no momento da compra. Realizamos uma análise de quilometragem ano e marca, será demonstrado nas seções seguintes.

4.8.1 Agrupamento quilometragem dos carros

Para analisarmos a média de quilometragem dos carros por ano, foi feito um agrupamento dos registros por ano, quilometragem e quantidade encontrada. As colunas agrupadas foram: *YearModel* (ano do modelo) e *Odometer* (quilometragem).

Figura 40: Agrupamento Ano do Modelo e Quilometragem

```
# Agrupa os carros por Ano e Calcula a Média e a Quantidade por Ano

# Agrupa por ano e remove o index (YearModel)
cars_group_mean_year = df_cars.groupby(['YearModel'],
as_index=False)

# Exibe colunas:
# YearModel: valor único (agrupado)
# Odometer: média de valores (Km)
# UniqueId: quantidade de registros
cars_group_mean_year = cars_group_mean_year.agg({'YearModel':
np.unique, 'Odometer':np.mean, 'UniqueId': np.size}).round(2)

# Renomeando as colunas para apresentá-las
cars_group_mean_year =
cars_group_mean_year.rename(columns={'YearModel':'Ano','Odometer':'
Km', 'UniqueId': 'Quantidade'})

# Ordenando em ordem Decrescente pela coluna ano
cars_group_mean_year = cars_group_mean_year.sort_values('Ano',
ascending=False)
#.reset_index()

cars_group_mean_year.head()
```

Fonte: Autor

Teremos como resultado do código acima:

Figura 41: Screenshot - Agrupamento Ano do Modelo e Quilometragem

	Ano	Km	Quantidade
37	2022	5.96	100
36	2021	2333.54	1218
35	2020	34938.99	2797
34	2019	38896.65	1553
33	2018	51397.67	811

Fonte: Autor

O resultado exibido foi apenas dos 5 primeiros registros, onde claramente podemos identificar que quanto mais novo o carro, menor será sua média de quilometragem. Os registros foram ordenados por ano de forma decrescente.

Identificamos também que entre um ano e outro existe uma média de quilometragem muito próxima a uma diferença de 10.000 a 20.000km rodados.

4.8.2 Gráfico Agrupamento quilometragem dos carros

Para melhorar nossa análise, construímos também um gráfico do agrupamento realizado na seção anterior. Porém, para exemplificar, consideramos apenas os carros com ano a partir de 2000 até o ano atual.

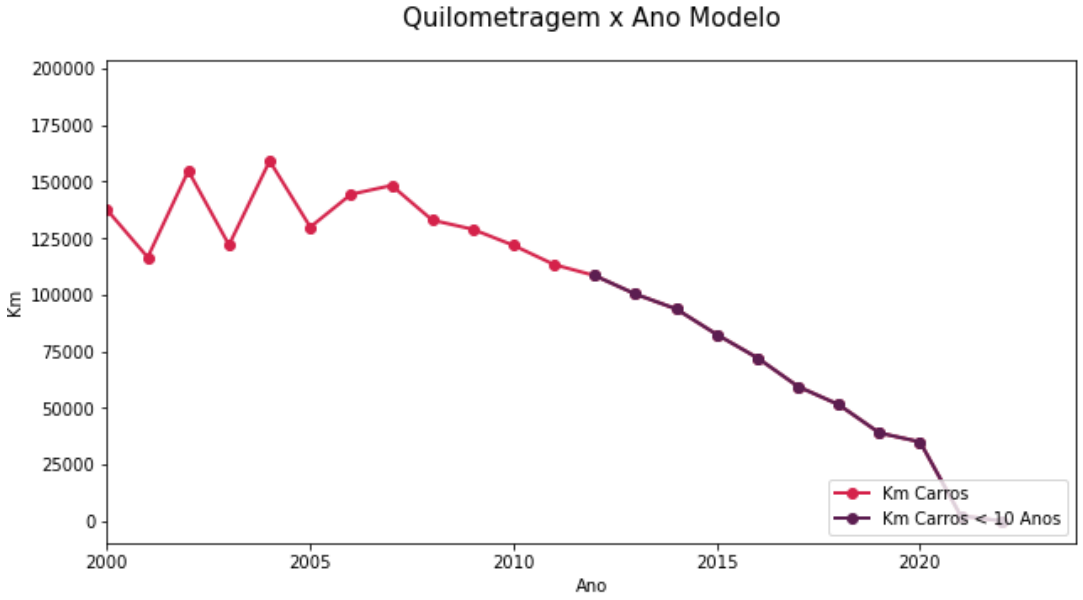
Figura 42: Código Gráfico Agrupamento ano e quilometragem

```
# Gráfico variação de Quilometragem e Ano do carro

# Pega o resultado do agrupamento (Ano + Quilometragem) para exibir
no gráfico
fig = plt.figure(figsize=(8,4))
eixo = fig.add_axes([0, 0, 1, 1])

# Adicionando gráfico
eixo.plot(cars_group_mean_year['Ano'], cars_group_mean_year['Km'],
color= '#d5224a', lw=2, marker = 'o')

# Exibe no gráfico somente os carros entre [2000 - Ano Atual]
eixo.set_xlim(2000) # a partir de 2000
```

```
# Adicionando cor de destaque para carros que ainda estão entre 10
anos
cars_10_years = now.year - cars_group_mean_year['Ano']  < 10
eixo.plot(cars_group_mean_year[cars_10_years]['Ano'],
cars_group_mean_year[cars_10_years]['Km'], color= '#5c1e51', lw=2,
marker = 'o')

# Alterando Título e Labels
eixo.set_title('Quilometragem x Ano Modelo', fontsize=15, pad=20)
eixo.set_ylabel('Km')
eixo.set_xlabel('Ano')

# Adicionando Legenda
eixo.legend(['Km Carros', 'Km Carros < 10 Anos'], loc = 'lower
right')
```

Fonte: Autor

Figura 43: Screenshot - Gráfico Ano e Quilometragem



Fonte: Autor

A figura 43 acima mostra graficamente os resultados do agrupamento ano e quilometragem, considerando apenas os carros a partir de 2000. No gráfico destacamos os carros que tem até 10 anos com a cor mais forte (cor roxa) e os carros mais velhos na cor mais fraca. Identificamos a curva do gráfico de forma

ascendente, ou seja quanto maior o ano do carro, mais velho ele é e maior será a quilometragem.

4.9 Análise valor Fipe OK dos carros

Nesta seção analisaremos a coluna *PriceFipeOK, ela* é considerada um atributo importante para nossa análise, pois identifica se o valor do carro no anúncio corresponde a um valor ideal para o carro, através da média de valor na tabela Fipe. Caso o valor do carro no anúncio seja menor ou igual ao valor do carro na tabela Fipe, a coluna receberá *True*, caso contrário receberá *False*.

4.9.1 Análise valor Fipe OK e Fipe não OK das 10 maiores marcas

Agrupamos os registros dos valores *PriceFipeOk* sendo eles com valores True ou False. Para melhor análise dos dados e aplicarmos os gráficos, consideramos apenas registros das 10 maiores marcas listadas pelo Notícias Automotivas em 2019. (NOTÍCIAS AUTOMOTIVAS, 2019). Pois desta forma conseguimos visualizar melhor os resultados, uma vez que na nossa base é possível uma variedade de mais de 30 marcas em anúncios.

Consideramos as seguintes marcas: Toyota, Volkswagen, Ford, Honda, Nissan, Chevrolet, Kia, Mercedes-Benz e BMW.

A figura 44 mostra como foi feito o agrupamento dos dados pela classe True ou False da coluna *PriceFipeOK*.

Figura 44: Agrupamento carros *PriceFipeOk*

```
# Agrupa os carros por Make e quantidade de carros com PriceFipeOk
(True or False)
# obs: Consideremos apenas as 10 maiores marcas de carro
make_top_10 =
['TOYOTA','VOLKSWAGEN','FORD','HONDA','NISSAN','HYUNDAI','CHEVROLET
','KIA','MERCEDES-BENZ','BMW']

# Agrupa por marca e remove o index (Make)
cars_group_make_fipe = pd.get_dummies(df_cars,
columns=['PriceFipeOk']).groupby(['Make'], as_index=False).sum()
```

```
# Renomeando as colunas para apresentá-las
cars_group_make_fipe =
cars_group_make_fipe.rename(columns={'Make':'Marca',
'PriceFipeOk_True':'Preço Fipe OK', 'PriceFipeOk_False':'Preço Fipe
Não OK'})

# Exibe colunas:
# Make: valor único (agrupado)
# PriceFipeOk_True: quantidade de registros Preço Fipe OK
# PriceFipeOk_False: quantidade de registros Preço Fipe Não OK
cars_group_make_fipe = cars_group_make_fipe[['Marca', 'Preço Fipe
OK', 'Preço Fipe Não OK']]

# Pega apenas os registros que estão com o Preço Fipe OK
(PriceFipeOk) e que estão entre as 10 maiores marcas veículos
cars_group_make_fipe =
cars_group_make_fipe[(cars_group_make_fipe['Marca'].isin(make_top_1
0))]

cars_group_make_fipe
```

Fonte: Autor

O resultado do agrupamento está ilustrado na figura 45 abaixo.

Figura 45: Screenshot - Agrupamento Preço Fipe OK e Preço Fipe Não OK

	Marca	Preço Fipe OK	Preço Fipe Não OK
3	BMW	86.0	260.0
5	CHEVROLET	164.0	1268.0
11	FORD	306.0	1021.0
14	HONDA	29.0	256.0
15	HYUNDAI	64.0	562.0
21	KIA	38.0	173.0
29	MERCEDES-BENZ	80.0	285.0
33	NISSAN	44.0	220.0
42	TOYOTA	48.0	381.0
44	VOLKSWAGEN	182.0	1437.0

Fonte: Autor

Podemos observar que as marcas Ford, Volkswagen e Chevrolet possuem mais anúncios no site do que as demais.

4.9.2 Gráfico Agrupamento valor Fipe OK e Fipe não OK das 10 maiores marcas

Para melhorar nossa análise no agrupamento anterior, aplicamos o gráfico de barras conforme a figura 46.

Figura 46: Gráfico - Agrupamento carros *PriceFipeOk*

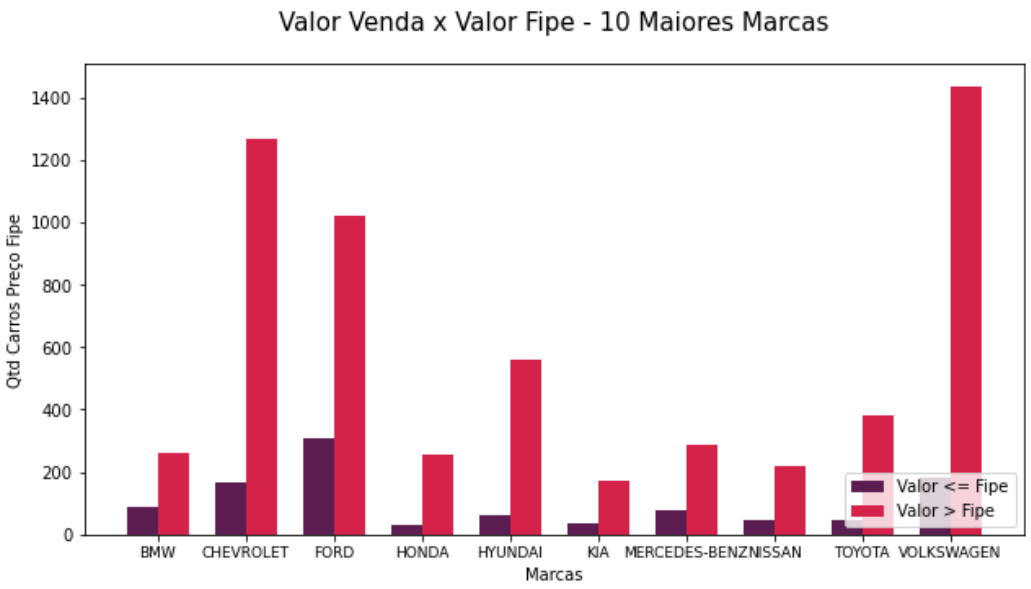
```
# Gráfico variação de Marcas e Preço Fipe

# Pega o resultado do agrupamento (Marca + Preço Fipe) para exibir no gráfico
fig = plt.figure(figsize=(8,4))
eixo = fig.add_axes([0, 0, 1, 1])

# Adicionando os Índices das marcas
indice = np.arange(len(cars_group_make_fipe))

width = 0.35  # tamanho das barras
```

```
# Adicionando barras Preço <= Valor Fipe
eixo.bar(indice - width/2, cars_group_make_fipe['Preço Fipe OK'],
width, color= '#5c1e51')

# Adicionando barras Preço > Valor Fipe
eixo.bar(indice + width/2, cars_group_make_fipe['Preço Fipe Não
OK'], width, color= '#d5224a')

# Alterando Título e Labels
eixo.set_title('Valor Venda x Valor Fipe - 10 Maiores Marcas',
fontsize=15, pad=20)
eixo.set_xlabel('Marcas')
eixo.set_ylabel('Qtd Carros Preço Fipe')

eixo.set_xticks(indice)
eixo.set_xticklabels(cars_group_make_fipe['Marca'], fontsize=9)

# Adicionando Legenda
eixo.legend(['Valor <= Fipe', 'Valor > Fipe'], loc = 'lower right')
```

Fonte: Autor

A figura x mostra o resultado obtido graficamente.

Figura 47: Screenshot - Gráfico Agrupamento Preço Fipe OK e Preço Fipe Não OK



Fonte: Autor

Podemos observar que em todas as marcas o preço do anúncio está maior do que o valor da tabela Fipe.

4.9.3 Análise valor Fipe OK e Fipe não OK em porcentagem

Agrupamos os dados considerando o Preço Fipe OK, Quantidade a Porcentagem da quantidade de carros encontrados, filtrando apenas carros menores ou iguais a 10 anos.

Figura 48: Agrupamento Preço Fipe OK, Quantidade e Porcentagem

```
# Agrupa os carros onde PriceFipeOk (o valor está <= ao valor da
tabela Fipe ou valor > valor da tabela Fipe)

# Fazer agrupamento somente com carros com ano <= 10 anos
cars_10_years = now.year - df_cars['YearModel']  <= 10

# Agrupa por PriceFipeOk e remove o index (PriceFipeOk)
# somente para carros onde ano <= 10
cars_group_fipe_10_years =
df_cars[cars_10_years].groupby(['PriceFipeOk'], as_index=False)

# Exibe colunas:
```

```
# PriceFipeOk: valor único (agrupado)
# UniqueId: quantidade de registros
cars_group_fipe_10_years =
cars_group_fipe_10_years.agg({'PriceFipeOk': np.unique, 'UniqueId':
np.size}).round(2)

# Renomeando as colunas para apresentá-las
cars_group_fipe_10_years =
cars_group_fipe_10_years.rename(columns={'PriceFipeOk':'Preço Fipe
OK', 'UniqueId': 'Quantidade'})


# Alterando os valores das linhas da coluna Preço Fipe OK para o
label text
# obs: Isso facilitará no uso do gráfico logo abaixo
cars_group_fipe_10_years.loc[cars_group_fipe_10_years['Preço Fipe
OK'] == True, ['Preço Fipe OK']] = 'Valor <= Fipe'
cars_group_fipe_10_years.loc[cars_group_fipe_10_years['Preço Fipe
OK'] == False, ['Preço Fipe OK']] = 'Valor > Fipe'

# Calculando a Porcentagem da coluna Quantidade
# obs: Essa coluna será usada no gráfico abaixo
cars_group_fipe_10_years['Porcentagem'] =
(cars_group_fipe_10_years['Quantidade'] /
cars_group_fipe_10_years['Quantidade'].sum()) * 100

# ordena o Data Frame para apresentar primeiro os valores Valor >
Fipe e posteriormente Valor <= Fipe
# obs: isso garantirá no gráfico esta ordem para utilização da
configuração de cores
cars_group_fipe_10_years =
cars_group_fipe_10_years.sort_values(by=['Preço Fipe OK'],
ascending=False)

cars_group_fipe_10_years
```

Fonte: Autor

O resultado do agrupamento obtido foi:

Figura 49: Screenshot - Agrupamento Preço Fipe OK, Quantidade e Porcentagem

	Preço Fipe OK	Quantidade	Porcentagem
0	Valor > Fipe	8366	82.496795
1	Valor <= Fipe	1775	17.503205

Fonte: Autor

Podemos verificar que a grande maioria em quantidade e porcentagem são de carros com valores maiores ao valor da tabela Fipe.

4.9.4 Gráfico Análise valor Fipe OK e Fipe não OK em porcentagem

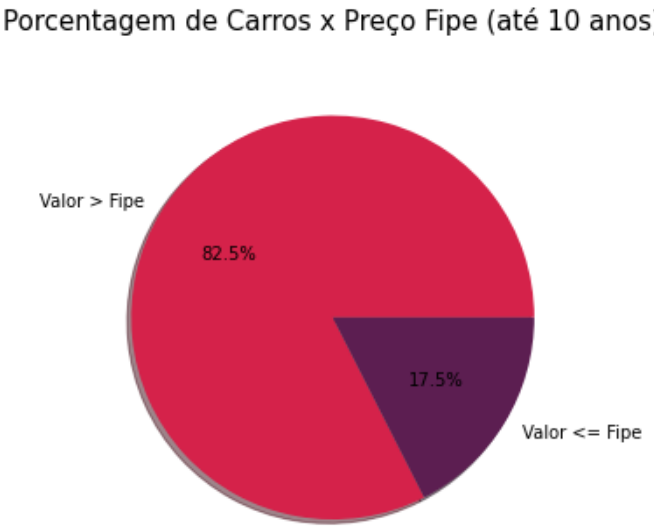
Nesta seção mostraremos os resultados obtidos no agrupamento valor Fipe OK, Fipe não OK e agrupamento de forma gráfica.

Figura 50: Gráfico - Agrupamento carros Preço Fipe OK e Porcentagem

```
# Gráfico Quantidade de Carros Preço Fipe (valor <= valor Fipe e
valor > valor Fipe)

# Pega o resultado do agrupamento (Marca + Preço Fipe) para exibir
no gráfico
fig = plt.figure(figsize=(8,4))
eixo = fig.add_axes([0, 0, 1, 1])

eixo.pie(cars_group_fipe_10_years['Porcentagem'],
labels=cars_group_fipe_10_years['Preço Fipe OK'], autopct='%.1f%%',
colors=['#d5224a', '#5c1e51'], shadow=True)

# Alterando Título e Labels
eixo.set_title('Porcentagem de Carros x Preço Fipe (até 10 anos)',
fontsize=15, pad=20)
```

Fonte: Autor

Figura 51: Screenshot - Agrupamento carros Preço Fipe OK e Porcentagem



Fonte: Autor

4.10 Criação da coluna *OdometerRecommended*

Após as análises realizadas, concluímos que a quilometragem também pode ser considerada um atributo relevante para classificar o anúncio como um Bom Negócio. A quilometragem pode estar diretamente ligada a quantidade de vezes que um veículo precisa ser revisado e isso acaba tendo impacto na sua parte mecânica, caso o atual dono não tenha feito as revisões corretamente. De um modo geral, o mercado considera de 10 a 15 mil km rodados por ano a média ideal para o veículo para que ele não seja considerado desgastado. A média também pode ser considerada como um dos critérios para classificar o veículo como seminovo ou usado. (CAR FLIX, 2020).

Neste caso criamos a coluna *OdometerRecommended*, atribuindo o valor True para Quilometragem dos carros menores que 15 mil km rodados, conforme a figura 52.

Figura 52: Agrupamento carros Preço Fipe OK e Porcentagem

```
df_cars['OdometerRecommended'] = (df_cars['Odometer'] / (now.year -
df_cars['YearModel']) < 15000)
```

Fonte: Autor

Para realizar o cálculo da média da quilometragem , pegamos a quantidade de km rodados e dividimos pelo ano atual subtraído pelo ano do modelo do carro e desta forma conseguimos comparar a média de quilometragem ideal por ano.

4.11 Correlação dos atributos

Nessa seção será apresentado os resultados relacionados a correlação entre os atributos do dataset.

A relação entre duas variáveis é chamada de correlação. Em estatística, o método mais comum para calcular a correlação é o coeficiente de correlação de Pearson. Pode ter três valores da seguinte forma:

- *Valor do coeficiente = 1*: Representa a correlação positiva total entre as variáveis.
- *Valor do coeficiente = -1*: Representa a correlação negativa total entre as variáveis.
- *Valor do coeficiente = 0*: Não representa nenhuma correlação entre as variáveis.

É de extrema importância verificar as correlações de pares dos atributos do dataset antes de usá-lo no projeto de ML, pois alguns algoritmos de aprendizado de máquina, como regressão linear e regressão logística, terão um desempenho ruim se tiver atributos altamente correlacionados.

Em Python, podemos calcular facilmente uma matriz de correlação de atributos de conjunto de dados com a função *corr()* no Pandas DataFrame.

4.11.1 Matriz de Correlação

Antes de aplicarmos a matriz de correlação, preparamos os dados que serão utilizados na análise. Neste momento já estamos preparando os dados para classificação e excluindo alguns atributos que não são relevantes, isso facilitará a análise.

Figura 53: Preparando o dataset para o ML

```
# Fazendo uma cópia do data frame original
df_cars_class = df_cars.copy()

# Deletando atributos que não serão utilizados
del df_cars_class['UniqueId']
del df_cars_class['Make']
del df_cars_class['Model']
del df_cars_class['Licensed']

# Convertendo o tipo dos atributos
df_cars_class['OdometerRecommended'] =
df_cars_class['OdometerRecommended'].astype('int64')
df_cars_class['IPVApaid'] =
df_cars_class['IPVApaid'].astype('int64')
#df_cars_class['Licensed'] =
df_cars_class['Licensed'].astype('int64')
df_cars_class['Warranty'] =
df_cars_class['Warranty'].astype('int64')
df_cars_class['OnlyOwner'] =
df_cars_class['OnlyOwner'].astype('int64')
df_cars_class['PriceFipeOk'] =
df_cars_class['PriceFipeOk'].astype('int64')
df_cars_class['Year>10Years'] =
df_cars_class['Year>10Years'].astype('int64')

# Convertendo dados categóricos e numéricos
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
df_cars_class['Armored']=le.fit_transform(df_cars_class['Armored'])
df_cars_class['Color']=le.fit_transform(df_cars_class['Color'])
```

Fonte: Autor

A figura 53, mostra as técnicas utilizadas para preparar os dados para as próximas análises. Algumas colunas consideradas menos importantes foram removidas, e algumas foi preciso converter seu tipo, assim como a coluna Armored e Color que são atributos categóricos, foi necessário transformá-los em atributos em escala numérica.

4.11.2 Gráfico Matriz de Correlação

Foi desenvolvido um gráfico da Matriz de Correlação para identificarmos visualmente a correlação dos atributos. Para obter os resultados foi utilizado a função *corr()* do Pandas, juntamente com a biblioteca *Matplotlib* para exibir o gráfico.

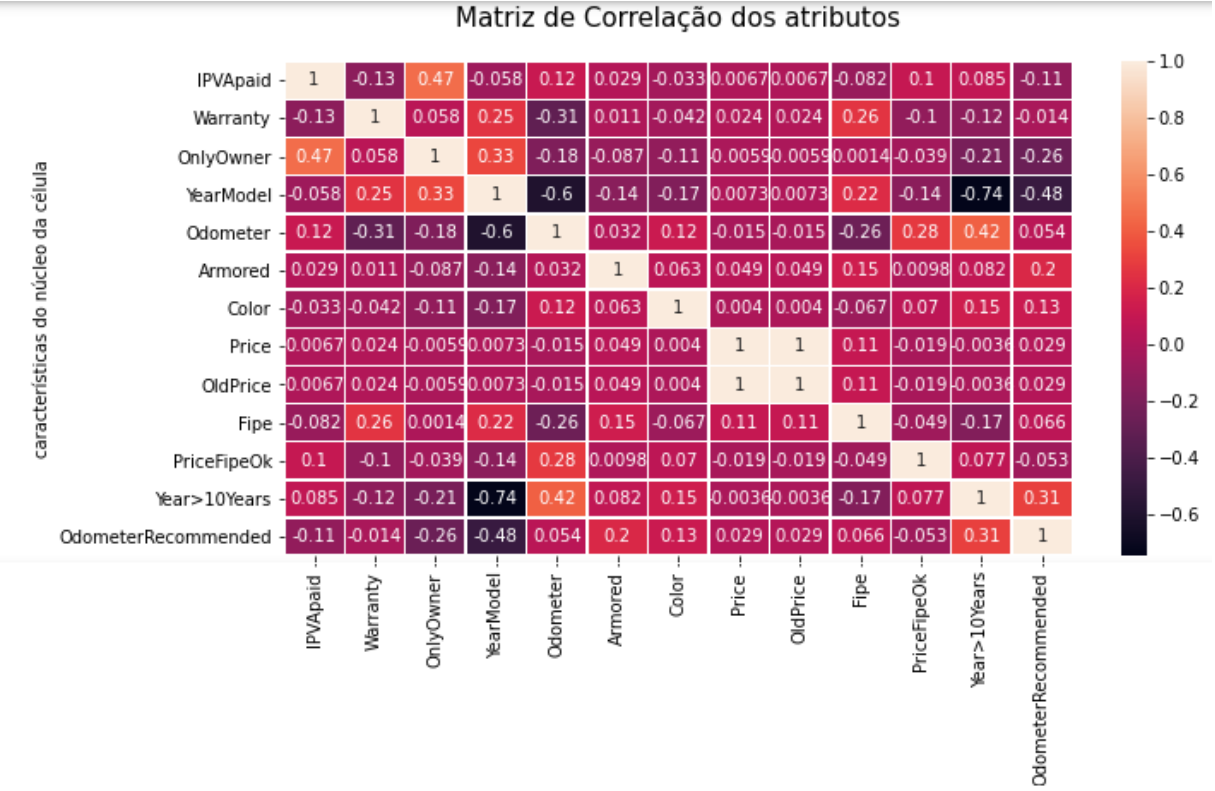
Figura 54: Gráfico - Matriz de Correlação

```
# Gráfico de Matriz de Correlação
df_small = df_cars_class.copy()

# O corr() método do Pandas DataFrame  é usado para calcular a
matriz.
# Por padrão, ele calcula o coeficiente de correlação de Pearson
correlation_mat = df_small.corr(method='pearson')

# Definindo as configurações do Gráfico
fig = plt.figure(figsize=(8,4))
eixo = fig.add_axes([0, 0, 1, 1])

# Usando o método heatmap para traçar a Matriz
# O parâmetro ' annot=True' exibe os valores do coeficiente de
correlação em cada célula.
sns.heatmap(correlation_mat, annot = True, linewidth=0.5)

# Definindo título e labels do gráfico
eixo.set_title('Matriz de Correlação dos atributos', fontsize=15,
pad=20)
eixo.set_xlabel ("características do núcleo da célula")
eixo.set_ylabel ("características do núcleo da célula")
```

Fonte: Autor

O resultado obtido do gráfico podemos verificar na figura 55.

Figura 55: Screenshot - Gráfico Matriz de Correlação


Matriz de Correlação dos atributos
características do núcleo da célula
IPVApaid 1 -0.13 0.47 -0.058 0.12 0.029 -0.033 0.0067 0.0067 -0.082 0.1 0.085 -0.11
Warranty -0.13 1 0.058 0.25 -0.31 0.011 -0.042 0.024 0.024 0.26 -0.1 -0.12 -0.014
OnlyOwner 0.47 0.058 1 0.33 -0.18 -0.087 -0.11 -0.0059 -0.0059 0.0014 -0.039 -0.21 -0.26
YearModel -0.058 0.25 0.33 1 -0.6 -0.14 -0.17 0.0073 0.0073 0.22 -0.14 -0.74 -0.48
Odometer 0.12 -0.31 -0.18 -0.6 1 0.032 0.12 -0.015 -0.015 -0.26 0.28 0.42 0.054
Armored 0.029 0.011 -0.087 -0.14 0.032 1 0.063 0.049 0.049 0.15 0.0098 0.082 0.2
Color -0.033 -0.042 -0.11 -0.17 0.12 0.063 1 0.004 0.004 -0.067 0.07 0.15 0.13
Price 0.0067 0.024 -0.0059 0.0073 -0.015 0.049 0.004 1 1 0.11 -0.019 -0.0036 0.029
OldPrice 0.0067 0.024 -0.0059 0.0073 -0.015 0.049 0.004 1 1 0.11 -0.019 -0.0036 0.029
Fipe -0.082 0.26 0.0014 0.22 -0.26 0.15 -0.067 0.11 0.11 1 -0.049 -0.17 0.066
PriceFipeOk 0.1 -0.1 -0.039 -0.14 0.28 0.0098 0.07 -0.019 -0.019 -0.049 1 0.077 -0.053
Year>10Years 0.085 -0.12 -0.21 -0.74 0.42 0.082 0.15 0.0036 0.0036 -0.17 0.077 1 0.31
OdometerRecommended -0.11 -0.014 -0.26 -0.48 0.054 0.2 0.13 0.029 0.029 0.066 -0.053 0.31 1
IPVApaid
Warranty
OnlyOwner
YearModel
Odometer
Armored
Color
Price
OldPrice
Fipe
PriceFipeOk
Year>10Years
OdometerRecommended
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

Fonte: Autor

Consideremos o seguinte entendimento para a Matriz de correlação:

- Cada célula da grade representa o valor do coeficiente de correlação entre duas variáveis.
- O valor na posição (*a*, *b*) representa o coeficiente de correlação entre os recursos na linha *a* e coluna *b*. Isso será igual ao valor na posição (*b*, *a*)
- É uma matriz quadrada - cada linha representa uma variável e todas as colunas representam as mesmas variáveis que linhas, portanto, o número de linhas = número de colunas.
- É uma matriz simétrica - isso faz sentido porque a correlação entre a, b será a mesma que entre b, a.
- Todos os elementos diagonais são 1 . Como os elementos diagonais representam a correlação de cada variável consigo mesma, ela sempre será igual a 1.
- Os pontos dos eixos denotam o recurso que cada um deles representa.
- Um grande valor positivo (próximo a 1,0) indica uma forte correlação positiva, ou seja, se o valor de uma das variáveis aumenta, o valor da outra variável também aumenta.
- Um grande valor negativo (próximo a -1,0) indica uma forte correlação negativa, ou seja, o valor de uma variável diminui com o aumento da outra e vice-versa.
- Um valor próximo a 0 (positivo ou negativo) indica a ausência de qualquer correlação entre as duas variáveis e, portanto, essas variáveis são independentes uma da outra.
- Cada célula na matriz acima também é representada por tons de uma cor. Aqui, os tons mais escuros da cor indicam valores menores, enquanto os tons mais brilhantes correspondem a valores maiores (perto de 1). Esta escala é dada com a ajuda de uma barra colorida no lado direito do gráfico.

4.11.3 Classificando a Matriz de Correlação

Se os dados fornecidos tiverem um grande número de recursos, a matriz de correlação pode se tornar muito grande e, portanto, difícil de interpretar. Neste caso

seria interessante classificar os valores da matriz e ver a intensidade da correlação entre os vários pares de recursos em ordem crescente ou decrescente.

Para isso, precisamos converter a matriz fornecida em uma série de valores unidimensionais.

O método *unstack()* no Pandas DataFrame retorna uma Série com MultiIndex, ou seja, cada valor da Série é representado por mais de um índice, que neste caso são os índices de linha e coluna que são os nomes dos recursos.

Figura 56: Classificando a Matriz de Correlação

```
correlation_mat = df_small.corr(method='pearson')

corr_pairs = correlation_mat.unstack()

corr_pairs
```

Fonte: Autor

Figura 57: Screenshot - Classificando a Matriz de Correlação

```
IPVApaid              IPVApaid                1.000000
                      Warranty               -0.126517
                      OnlyOwner               0.466906
                      YearModel              -0.057668
                      Odometer                0.115902
                                               ...
OdometerRecommended   OldPrice                0.028934
                      Fipe                    0.065949
                      PriceFipeOk            -0.052857
                      Year>10Years            0.306212
                      OdometerRecommended     1.000000
Length: 169, dtype: float64
```

Fonte: Autor

Para classificar esses valores, foi utilizado a função da Série Pandas: *sort_values()*.

Figura 58: Classificando a Matriz de Correlação

```
sorted_pairs = corr_pairs.sort_values(kind="quicksort")

print(sorted_pairs)
```

Fonte: Autor

Figura 59: Screenshot - Classificando a Matriz de Correlação

```
Year>10Years          YearModel              -0.743747
YearModel             Year>10Years           -0.743747
                      Odometer               -0.598858
Odometer              YearModel              -0.598858
OdometerRecommended   YearModel              -0.480630
                                                ...
YearModel             YearModel               1.000000
OnlyOwner             OnlyOwner               1.000000
Warranty              Warranty                1.000000
Year>10Years          Year>10Years            1.000000
OdometerRecommended   OdometerRecommended     1.000000
Length: 169, dtype: float64
```

Fonte: Autor

Podemos ver que cada valor é repetido duas vezes na saída classificada. Isso ocorre porque a matriz de correlação é uma matriz simétrica e cada par de características ocorria duas vezes nela. No entanto, agora os valores de coeficiente de correlação classificados de todos os pares de recursos e podemos tomar decisões de acordo.

4.11.3.1 Seleção de pares de correlação negativa

Podemos selecionar pares de características com uma faixa particular de valores do coeficiente de correlação. Neste caso selecionaremos pares com correlação negativa a partir dos pares classificados na seção anterior.

Figura 60: Selecionando pares da correlação negativa

```
negative_pairs = sorted_pairs[sorted_pairs < 0]

print(negative_pairs)
```

Fonte: Autor

Figura 61: Screenshot - Selecionando pares da correlação negativa

```
Year>10Years          YearModel      -0.743747
YearModel             Year>10Years   -0.743747
                      Odometer       -0.598858
Odometer              YearModel      -0.598858
OdometerRecommended   YearModel      -0.480630
                                        ...
OnlyOwner             OldPrice       -0.005889
Year>10Years          Price          -0.003641
Price                 Year>10Years   -0.003641
OldPrice              Year>10Years   -0.003641
Year>10Years          OldPrice       -0.003641
Length: 70, dtype: float64
```

Fonte: Autor

4.11.3.2 Seleção de pares de correlação fortes (magnitude que 0,5)

Nesta seção selecionamos os recursos fortemente relacionados. Ou seja, tentaremos filtrar os pares de recursos cujos valores de coeficiente de correlação são maiores que 0,5 ou menores que -0,5.

Figura 62: Selecionando pares da correlação fortes

```
strong_pairs = sorted_pairs[abs(sorted_pairs) > 0.5]

print(strong_pairs)
```

Fonte: Autor

Figura 64: Screenshot - Selecionando pares da correlação fortes

```
YearModel           Odometer            -0.598858
Odometer            YearModel           -0.598858
OldPrice            Price                1.000000
Price               OldPrice             1.000000
IPVApaid            IPVApaid             1.000000
Fipe                Fipe                 1.000000
OldPrice            OldPrice             1.000000
Price               Price                1.000000
Color               Color                1.000000
Armored             Armored              1.000000
Odometer            Odometer             1.000000
YearModel           YearModel            1.000000
OnlyOwner           OnlyOwner            1.000000
Warranty            Warranty             1.000000
PriceFipeOk         PriceFipeOk          1.000000
OdometerRecommended OdometerRecommended  1.000000
dtype: float64
```

Fonte: Autor

4.11.4 Verificando assimetria dos atributos

A assimetria pode ser definida como a distribuição que se presume ser gaussiana, mas parece distorcida ou deslocada em uma direção ou outra, para a esquerda ou direita. Rever a assimetria de atributos é uma das tarefas importantes devido aos seguintes motivos:

- A presença de assimetria nos dados requer a correção no estágio de preparação dos dados para que possamos obter mais precisão do nosso modelo.
- A maioria dos algoritmos de ML assume que os dados têm uma distribuição Gaussiana, ou seja, dados normais ou cursos em sino.

A partir da saída da Figura 63 abaixo, uma inclinação positiva ou negativa pode ser observada. Se o valor estiver mais próximo de zero, ele mostra menos inclinação.

Figura 63: Verificando assimetria dos atributos

```
# Realizando a ordenação dos resultados
df_cars_class.skew().sort_values(kind="quicksort")
```

Fonte: Autor

Figura 64: Screenshot - Verificando Assimetria dos Atributos

```
YearModel              -2.267743
IPVApaid               -0.082694
Color                   0.025632
OnlyOwner               0.216288
OdometerRecommended     0.256008
PriceFipeOk             1.624716
Warranty                2.097096
Armored                 3.327714
GoodDeal                3.487174
Odometer                6.520480
Fipe                   11.044795
Price                  62.367798
OldPrice               62.367806
dtype: float64
```

Fonte: Autor

4.11.5 Gráficos univariados: Compreendendo os atributos de forma independente

O tipo mais simples de visualização é a visualização de variável única ou “univariada”. Com a ajuda da visualização univariada, podemos entender cada atributo do nosso conjunto de dados de forma independente. O Histograma é uma das técnicas em Python para implementar a visualização univariada.

4.11.5.1 Gráficos de Histogramas

Os histogramas agrupam os dados em caixas e é a maneira mais rápida de se ter uma ideia sobre a distribuição de cada atributo no conjunto de dados. A seguir estão algumas das características do histograma:

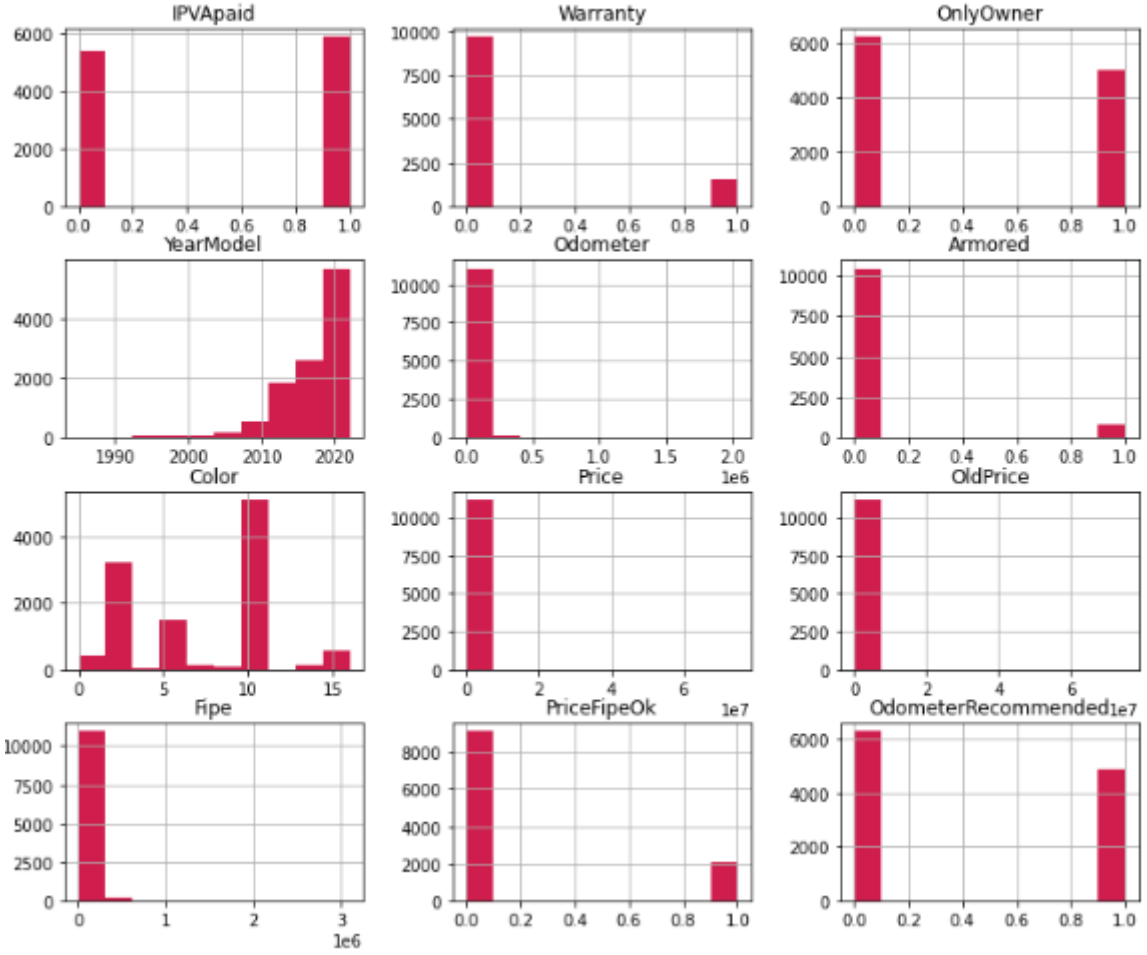
- Ele nos fornece uma contagem do número de observações em cada compartimento criado para visualização.
- Pela forma da caixa, podemos facilmente observar a distribuição, ou seja, se ela é gaussiana, distorcida ou exponencial.
- Os histogramas também nos ajudam a ver possíveis outliers.

Figura 65: Gráfico de Histogramas

```
# Gráfico Histogramas
df_small = df_cars_class.copy()

# Definindo as configurações do Gráfico
df_small.hist(color="#cf1e4d", figsize=(12,10))

plt.show()
```

Fonte: Autor

Figura 66: Screenshot - Gráfico de Histogramas



Fonte: Autor

5. Criação de Modelos de Machine Learning

Após as análises realizadas nas seções anteriores, iremos aplicar modelos de Machine Learning, utilizando algoritmos de classificação sobre os dados. O principal objetivo é identificar e classificar os atributos de maior importância para classificar se o anúncio é um bom ou mau negócio para o comprador.

5.1 Analisando e preparando os dados

Para poder classificar os dados, foi necessário criar uma coluna para categorizar e separar os anúncios como “Class” (1-Sim ou 2-Não), baseado no valor da coluna booleana GoodDeal.

A coluna *Class* é binária e tem como referência se o anúncio é ou não um Bom Negócio.

Figura 67: Criando coluna “Class”

```
# criando coluna Class
df_cars_class['Class'] = (df_cars['GoodDeal']).astype('int64')

# Deletando a coluna GoodDeal, pois agora seu valor está na coluna Class
del df_cars_class['GoodDeal']
```

Fonte: Autor

Para verificar a quantidade de cada classe (se é ou não um bom negócio), utilizamos a função *count()* do Pandas conforme a Figura 68.

Figura 68: Exibindo a quantidade de cada classe

```
df_cars_class.groupby('Class')['PriceFipeOk'].count()
```

Fonte: Autor

Figura 69: Screenshot - Exibindo a quantidade de cada classe

```
Class
0    10466
1      743
Name: PriceFipeOk, dtype: int64
```

Fonte: Autor

Na figura 69 acima temos:

- Não é um Bom Negócio: 10.466
- É um Bom Negócio: 743

Temos 743 anúncios classificados como “Bom Negócio”, e 10.466. Observamos que existe um desbalanceamento entre os anúncios considerados “Bom Negócio” e os que não foram.

5.2 Aplicando Algoritmos de Classificação

Para iniciar a classificação dos dados, foi necessário definir duas variáveis que foram utilizadas no modelo. São elas:

- X: todas as colunas, removendo apenas a coluna de classificação “Class”.
- Y: pega somente a coluna de classificação “Class”.

Figura 70: Definindo variáveis: X e Y

```
# Selecionando todas as colunas exceto a coluna “Class”
X = df_cars_class.drop('Class', axis=1)

# Selecionando apenas valores da coluna Class
y = df_cars_class['Class']
```

Fonte: Autor

Foi necessário também dividir a base de Teste e de Treino.

Nesse caso, consideramos 70% da base para treino e 30% para teste.

Figura 71: Exibindo a quantidade de cada classe

```
# Divide o DataFrame em teste e treino
# 70% treino
# 30% teste
```

```
train_X, test_X, train_y, test_y = train_test_split(X, y,
train_size=0.70, test_size=0.30, stratify=y)
```

Fonte: Autor

5.2.1 KNN (K-Nearest Neighbor)

O algoritmo KNN é um tipo de algoritmo de ML supervisionado que pode ser usado tanto para classificação quanto para problemas preditivos de regressão. No entanto, é usado principalmente para problemas de previsão de classificação na indústria.

A figura 72 mostra como é criado a instância KNeighborsClassifier.

Figura 72: Criando instância KNeighborsClassifier

```
# Instânciando KNeighborsClassifier
knn = KNeighborsClassifier()
knn = knn.fit(train_X, train_y)
```

Fonte: Autor

Os resultados obtidos utilizando o algoritmo KNeighborsClassifier foram conforme a figura 73.

Figura 73: Screenshot - Resultados KNeighborsClassifier

```
Confusion Matrix:
[[3103   37]
 [ 139   84]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97      3140
           1       0.69      0.38      0.49       223

    accuracy                           0.95      3363
   macro avg       0.83      0.68      0.73      3363
weighted avg       0.94      0.95      0.94      3363

Accuracy: 0.9476657746060065
```

Fonte: Autor

Na coluna de Acurácia (accuracy), podemos ver o quanto o modelo acertou nas previsões possíveis.

Para melhorar nossos resultados, consideramos a seguinte Matriz de Confusão:

Figura 74: Screenshot - Matriz de Confusão KNeighborsClassifier

Predito	0	1	All
Real			
0	3103	37	3140
1	139	84	223
All	3242	121	3363

Fonte: Autor

Dos 3.140 valores, o modelo previu como sendo classe “0”, 3.103 elementos foram classificados corretamente, 37 foram classificados como sendo classe “1”. Ou seja, nesse caso o modelo errou apenas 37 registros na previsão da classe 0.

5.2.2 Random Forest Classifier

Nessa seção será apresentado a utilização do algoritmo Random Forest Classifier para classificação dos dados.

Random Forest Classifier é um algoritmo de aprendizado supervisionado que é usado tanto para classificação quanto para regressão. Porém, ele é usado principalmente para problemas de classificação. Como sabemos que uma floresta é formada por árvores e mais árvores significa floresta mais robusta. Da mesma forma, o algoritmo cria árvores de decisão em amostras de dados e, em seguida, obtém a previsão de cada uma delas e, finalmente, seleciona a melhor solução por meio de votação. É um método de conjunto melhor do que uma única árvore de decisão porque reduz o sobreajuste ao calcular a média do resultado.

Para utilizar o algoritmo RandomForestClassifier, precisamos instanciar a classe RandomForestClassifier conforme a figura 75.

Figura 75: Criando instância RandomForestClassifier

```
rfc = RandomForestClassifier(n_estimators=50)
rfc = rfc.fit(train_X, train_y)
```

Fonte: Autor

Os resultados obtidos utilizando o algoritmo RandomForestClassifier foram conforme a figura 76.

Figura 76: Screenshot - Resultados RandomForestClassifier

```
Confusion Matrix:
[[3100   40]
 [ 101  122]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      3140
           1       0.75      0.55      0.63       223

    accuracy                           0.96      3363
   macro avg       0.86      0.77      0.81      3363
weighted avg       0.95      0.96      0.95      3363

Accuracy: 0.9580731489741302
```

Fonte: Autor

Para melhorar nossos resultados, consideramos a seguinte Matriz de Confusão:

Figura 57: Screenshot - Matriz de Confusão RandomForestClassifier

Predito	0	1	All
Real			
0	3100	40	3140
1	101	122	223
All	3201	162	3363

Fonte: Autor

O RandomForestClassifier teve um desempenho um pouco melhor do que o KNeighborsClassifier, de 3.140 registros classificados como classe 0, ele errou 40 registros apenas, porém de 223 classificados na classe 1, ele errou 122.

Podemos concluir que em ambos os algoritmos o valor da acurácia é considerado alto e com grande assertividade na classificação da classe 0. Porém ele possui mais erros na classificação da classe 1, devido ao desbalanceamento do dataset.

6. Apresentação dos Resultados

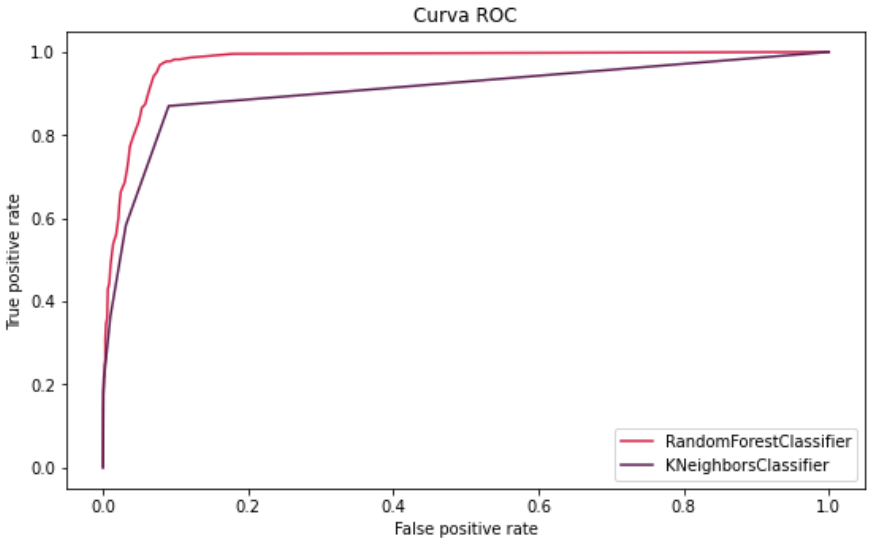
Nessa seção será apresentado os resultados obtidos. Para exemplificar foi desenvolvido o modelo Canvas proposto pelo Vasandani (clique aqui)..

Título: Bom Negócio - Anúncio de vendas de carros		
Problema Analisar o dataset de anúncios de vendas de carros da Webmotors e investigar atributos relacionados ao atributo “Bom Negócio.”	**Resultados e Previsões** Avaliar os atributos relacionados a atribuição verdadeira do atributo “Bom Negócio”, com a finalidade de tentar prever e classificar os atributos de maior importância e assim atribuir verdadeiro ou falso para o atributo “Bom Negócio”	**Aquisição de Dados** Os dados de ambos os datasets data-cars.json e data-cars-fipe.json) foram coletados do site da Webmotors.
Modelagem Realizado análises no dataset coletado, tanto de forma gráfica quanto análise descritiva dos dados utilizando a biblioteca *Pandas* em *Pyrthon.* Desta forma foi possível identificar um dataset adequado para aplicar modelo de classificação de ML.	**Avaliação do Modelo** Para avaliação dos resultados obtidos no modelo de classificação, foram avaliados a Matriz de Confusão e o Relatório de Classificação conforme o notebook em Python no diretório deste projeto.	**Preparação dos Dados** Após a união dos datasets, os dados foram tratados, as colunas foram renomeadas, os dados duplicados foram removidos e dados desnecessários para a análise também foram removidos.

A classificação ML utilizando o algoritmo Random Forest Classifier obteve um melhor resultado comparado ao algoritmo K Neighbors Classifier. Para facilitar essa compreensão, foi desenvolvido o Gráfico da Curva ROC. A curva ROC é um gráfico que mostra o desempenho de um modelo de classificação em todos os limites de classificação.

A figura 58 exibe o gráfico e nele cada linha de cada modelo na curva ROC, uma curva que está mais próxima do topo e a esquerda é a melhor avaliada, neste caso a Random Forest Classifier.

Figura 58: Screenshot - Gráfico Curva ROC

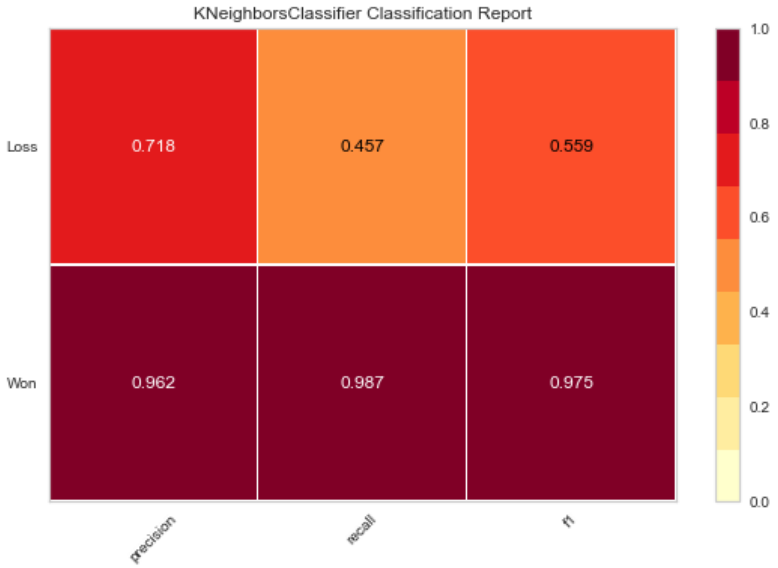


Fonte: Autor

Devido ao desbalanceamento do dataset , observamos que a categorização da classe 1 se tornou mais complicada, obtendo um resultado inferior comparado a classificação da classe 0.

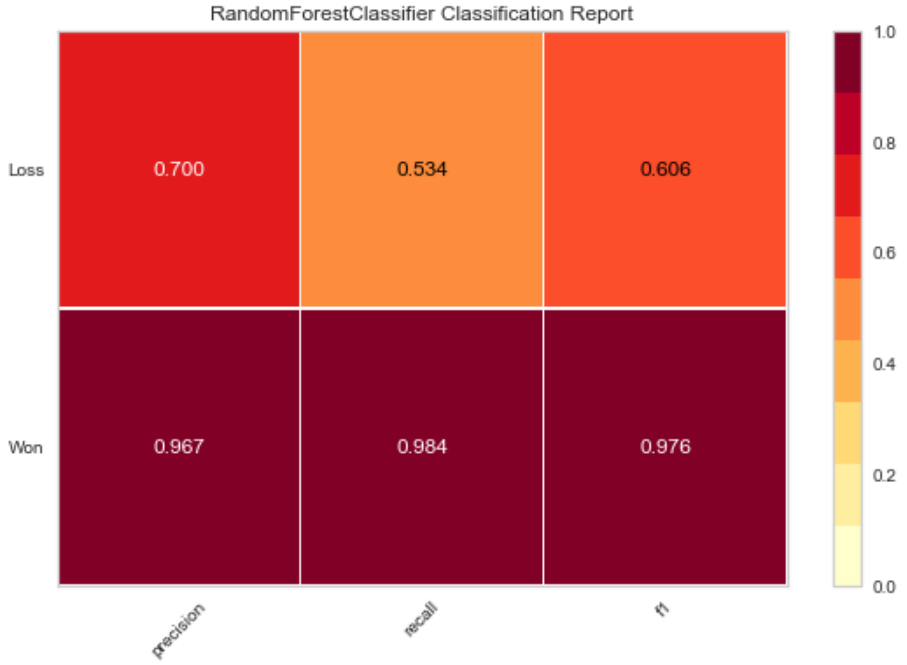
Para mostrar o relatório de resultados de cada algoritmo, consideremos o gráfico da figura 59 referente ao algoritmo KNeighborsClassifier e figura 60 referente ao algoritmo RandomForestClassifier.

Figura 59: Screenshot - Gráfico Resultados KNeighborsClassifier



Fonte: Autor

Figura 60: Screenshot - Gráfico Resultados RandomForestClassifier

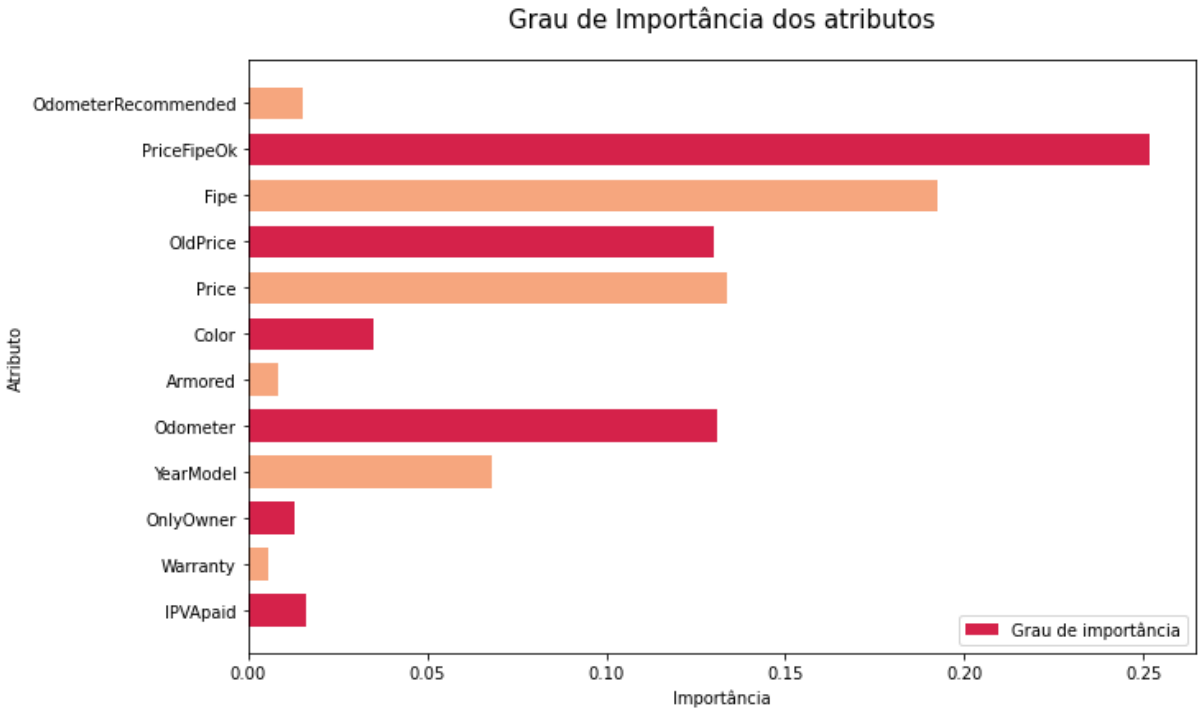


Fonte: Autor

Conforme os gráficos acima, podemos perceber uma pequena diferença de resultado, colocando o algoritmo RandomForestClassifier em melhor desempenho.

Para exibir os atributos de maior importância do algoritmo Random Forest Classifier, foi desenvolvido um gráfico com todos os atributos e seu grau de importância, conforme a figura 61.

Figura 61: Screenshot - Gráfico Grau de Importância RandomForestClassifier



Fonte: Autor

Conforme o gráfico acima, os atributos com maior grau de importância são os relacionados ao preço Fipe, ao preço e quilometragem do carro. Podemos ver que os atributos de menor importância são os referentes ao IPVA, garantia e blindagem do carro. Logo podemos concluir que os atributos diretamente ligados a vida útil do carro (Quilometragem e Ano) são proporcionais na definição do Preço, por isso foram apresentados com maior Grau de Importância na categorização de “Bom Negócio”.

7. Links

Todos os códigos desenvolvidos e a documentação utilizada são disponibilizados no repositório do Github.

Link para o vídeo: https://drive.google.com/file/d/11WKz7NBY41jevE6eWWklSHIDl62sa6TK/view?usp=sharing

Link Github: https://github.com/karenyov/TCC_PUC_BigData.

REFERÊNCIAS

CAR FLIX. **Quais as vantagens do big data em vendas?** Disponível em https://blog.carflix.com.br/quilometragem-usado-seminovo/#:~:text=De%20modo%20geral%2C%20o%20mercado,at%C3%A9%2020%20mil%20km%20rodados. Acesso em: 12/03/2021.

MEDIUM. **Machine Learning — O que é, tipos de aprendizagem de máquina, algoritmos e aplicações.** Disponível em https://medium.com/camilawaltrick/introducao-machine-learning-o-que-e-tipos-de-aprendizado-de-maquina-445dcfb708f0. Acesso em: 09/03/2021.

MEDIUM. **Por que usar Jupyter Notebook?** Disponível em https://suzana-svm.medium.com/por-que-usar-jupyter-notebook-77d5a59b42a1. Acesso em: 12/03/2021.

NOTÍCIAS AUTOMOTIVAS. **As 20 maiores e melhores marcas do mundo.** Disponível em https://www.noticiasautomotivas.com.br/marcas-de-carro-as-10-maiores-e-melhores-do-mundo. Acesso em: 18/02/2021.

TRANSFORMAÇÃO DIGITAL. **Quais as vantagens do big data em vendas?** Disponível em http://www.mma.gov.br/sitio/index.php?ido=conteudo.monta&idEstrutura=18. Acesso em: 12/02/2021.