

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

Bruno Philippe Maia

MACHINE LEARNING PARA
DETECÇÃO DE INCONSISTÊNCIAS
EM ANÚNCIOS DE AUTOMÓVEIS NA WEB

Belo Horizonte

2023

Bruno Philippe Maia

**MACHINE LEARNING PARA
DETECÇÃO DE INCONSISTÊNCIAS
EM ANÚNCIOS DE AUTOMÓVEIS NA WEB**

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

Belo Horizonte

2023

SUMÁRIO

1. Introdução	4
1.1. Contextualização	4
1.2. O problema proposto	5
1.3. Objetivos	6
2. Coleta de Dados	8
2.1. Web scraping	9
2.2. Olx	9
2.3. Seminovos	16
3. Processamento/Tratamento de Dados	24
3.1. Junção dos Arquivos	25
3.2. Extração e Padronização de Nomes de Marcas e Modelos	27
3.3. Extração e Padronização dos Atributos dos Automóveis	30
3.4. Tratamento de dados faltantes, informações inconsistentes e outliers	37
4. Análise e Exploração dos Dados	40
5. Criação de Modelos de Machine Learning	53
5.1. Modelo de Machine Learning	53
5.2. K-means	54
5.3. One-class Support Vector Machine	58
5.4. Principal Component Analysis - PCA	62
6. Apresentação e Interpretação dos Resultados	65
7. Considerações Finais	70
8. Links	72
REFERÊNCIAS	73

1. Introdução

1.1. Contextualização

A venda de carros seminovos e usados na internet tem se tornado cada vez mais comum, e com o aumento do acesso à internet e o desenvolvimento de plataformas online específicas para anúncios de carros usados, tem se tornado cada vez mais fácil e prático para os proprietários de veículos anunciarem seus carros e para os compradores pesquisarem e encontrarem veículos que atendam às suas necessidades.

O número de vendas de carros usados pela internet tem crescido consideravelmente nos últimos anos. As vendas de veículos usados e seminovos no Brasil atingiu a marca de 13,3 milhões de unidades em 2022, segundo dados da Federação Nacional das Associações de Revendedores de Veículos Automotores (Fenauto). Esse grande número se deve em parte à pandemia de Covid-19 nos últimos anos, que aumentou a demanda por veículos individuais, mas também reflete uma tendência de longo prazo de aumento da confiança e da comodidade das transações online.

As facilidades de anunciar carros na web incluem a possibilidade de criar anúncios com fotos e informações detalhadas sobre o veículo, a possibilidade de atingir um grande público em um curto espaço de tempo e a facilidade de atualizar ou modificar o anúncio conforme necessário. Além disso, muitas plataformas oferecem ferramentas de busca e filtragem que permitem aos compradores encontrar rapidamente os veículos que atendam às suas necessidades e orçamento.

No entanto, a venda de carros usados na web também apresenta alguns problemas e riscos. Por exemplo, os compradores podem se deparar com anúncios fraudulentos ou enganosos que apresentam informações imprecisas, fotos que não correspondem ao veículo real e até mesmo veículos de procedências duvidosas. Além disso, os compradores podem não ter a oportunidade de examinar o veículo pessoalmente antes de efetuar a compra, o que pode resultar em desapontamento ou até mesmo em problemas mecânicos ou de segurança mais graves. Com uma ampla gama de anúncios pode ser difícil definir se um automóvel anunciado está dentro do esperado no mercado, como por exemplo condições de conservação, preço, quilometragem, etc. Por fim, os vendedores também podem ser vítimas de fraudes, como

compradores que tentam negociar preços baixos ou que aplicam golpes no momento de efetivar a compra do automóvel.

1.2. O problema proposto

O usuário de plataformas Web de compra e venda de automóveis novos e seminovos pode se deparar com vários desafios na avaliação de um anúncio publicado. Estes anúncios envolvem possibilidade de fraude, informações fidedignas sobre o automóvel anunciado, precificação condizente com o mercado e até mesmo dificuldade de escolha de um automóvel, devido a ampla gama de anúncios e itens/características que os automóveis possuem.

Estes pontos são importantes inclusive para as plataformas. Pois a plataforma possui maior credibilidade quando possui anúncios com informações precisas, livres de fraudes e é facilitada a busca por automóveis por meio de padrões nos anúncios já existentes.

Para apresentar o problema proposta usou-se a técnica dos 5-Ws, que são um conjunto de perguntas que ajudam a entender e estruturar um problema. Eles são:

Why? O problema é importante porque a Internet é um espaço enorme para fraudes, e a venda de carros usados é um mercado que pode ser particularmente vulnerável. Os compradores podem perder muito dinheiro ao comprar um carro fraudulento, informações errôneas ou carros com característica fora do esperado no mercado; e os vendedores legítimos podem perder negócios se os compradores não confiarem nos anúncios. Portanto, ter um modelo de *machine learning* confiável para classificar anúncios de carros é crucial.

Who? Compradores e vendedores de carros usados na Web são diretamente afetados pelo problema. As empresas que hospedam esses anúncios também têm interesse em resolver o problema, pois podem perder negócios se os compradores não confiarem em seus serviços. Além disso, o governo, segurança pública e outras agências reguladoras podem querer usar esses modelos para monitorar fraudes em potencial.

What? O problema é a classificação de anúncios de carros na Web. O objetivo é criar um modelo de *machine learning* que possa analisar os dados dos anúncios de carros e prever se eles são anúncios legítimos ou fraudes, buscar padrões nos anúncios de carros a fim de auxiliar na busca por automóveis.

Where? O problema ocorre em plataformas de anúncios de carros na Web.

When? O problema pode ocorrer a qualquer momento, sempre que um anúncio de carro é publicado em uma plataforma na Web. No entanto, pode haver variações sazonais nas ações de compra e venda de carros na Web, por exemplo, durante as férias ou feriados, quando as pessoas podem estar mais dispostas a comprar um carro. O período analisado será de no máximo 1 ano, que é o período que anúncios ficam ativos nas plataformas.

1.3. Objetivos

O objetivo é utilizar algoritmos avançados para analisar dados de anúncios de automóveis na web. A partir desses dados, pretende-se identificar possíveis fraudes e padrões que possam ser úteis para auxiliar na compra de um carro.

A detecção de fraudes em anúncios de automóveis é uma tarefa importante para proteger os consumidores de informações falsas e enganosas. Através do uso de técnicas de *machine learning*, é possível analisar informações em larga escala, detectar anomalias e identificar padrões que possam indicar fraudes.

Além disso, ao identificar padrões nos anúncios de automóveis, é possível obter informações úteis para auxiliar na compra de um carro. Por exemplo, é possível identificar quais são as marcas e modelos de carros mais populares, quais são as características mais valorizadas pelos compradores, e quais são as faixas de preço mais comuns.

Dessa forma, o trabalho de *machine learning* pode contribuir para melhorar a

transparência do mercado de automóveis, proteger os consumidores de fraudes e oferecer informações úteis para auxiliar na compra de um carro.

2. Coleta de Dados

Os *datasets* de estudo foram extraídos de anúncios presentes em grandes plataformas de compra e venda de automóveis na Web. Hoje pode-se destacar três principais portais: Olx, Seminovos e Webmotors.

A Olx é uma plataforma de anúncio de produtos em geral, mas possui uma parte dedicada para o anúncio de automóveis, onde é possível detalhar o veículo anunciado, como sua marca, modelo, características e itens opcionais. A Olx é amplamente utilizada pois é bastante conhecida no mercado, além de permitir inserir anúncios gratuitos por um maior período de tempo.

O Seminovos é um website de compra e venda de anúncios que começou com anúncios de carro em Belo Horizonte e região metropolitana, mas com seu crescimento passou a também englobar anúncios de todo estado de Minas Gerais e até mesmo do país. É uma plataforma dedicado para anúncios de automóveis, dessa forma era possível inserir e obter informações mais detalhadas dos veículos em anúncio.

A Webmotors é uma plataforma mais recente em comparação com as duas anteriores, e mais completa quando se fala em detalhamento de informações do automóvel na hora de inserção e também na hora de buscar. Pelo ano do veículo já é possível filtrar somente as versões do modelo daquele ano, bem como informações de motorização, itens de série, opcionais e etc.

Neste estudo serão utilizados apenas os dados das plataformas Olx e Seminovos. A Webmotors, apesar de possuir dados mais estruturados e completos, não possui a data que o anúncio foi publicado, e essa informação é essencial para o estudo que deseja realizar sobre esses *datasets*.

Por fim, no Brasil existem milhões de carros anunciados na Web. Desta forma, tanto para performance dos algoritmos de *machine learning* quanto para performance dos algoritmos de extração de dados, a extração dos dados dos *datasets* foram reduzidas para automóveis apenas no estado de Minas Gerais.

2.1. Web scraping

Tanto a plataforma Olx quanto o Seminovos não possuem APIs para exportação de todos ou até mesmo parte dos automóveis anunciados. Dessa forma foi utilizada a técnica de *Web scraping* com as bibliotecas do Python para extração destes dados em ambos os portais. O *Web Scraping* é uma técnica de coleta de dados de plataformas online, como sites, redes sociais e etc. Os dados são capturados a partir de programas e *scripts*, que fazem o *download* das páginas e extraem as informações de acordo com os elementos presentes na página Web.

As bibliotecas de *Web scraping* do Python possui métodos que fazem o *download* da página Web e permitem navegar através dos elementos da página. Esses métodos buscam os elementos da página através dos tipos, classes, atributos e identificadores dos elementos dos elementos do código HTML da página.

2.2. Olx

Os dados dos anúncios de veículos da plataforma Olx foram extraídos entre os dias 07 de novembro de 2022 e 10 de novembro de 2022 e a página principal para navegação nesses anúncios é: <https://www.olx.com.br/autos-e-pecas/carros-vans-e-utilitarios/estado-mg>.

A plataforma Olx possui seus anúncios paginados. Dessa forma, para navegar entre os anúncios no momento do *Web scraping* basta requisitar a URL com a indicação da página. Por exemplo, a Figura 1 abaixo está mostrando os anúncios da primeira página, enquanto a Figura 2 mostra os anúncios da segunda página, sendo o parâmetro 'o=2' na URL é que realiza esse controle.

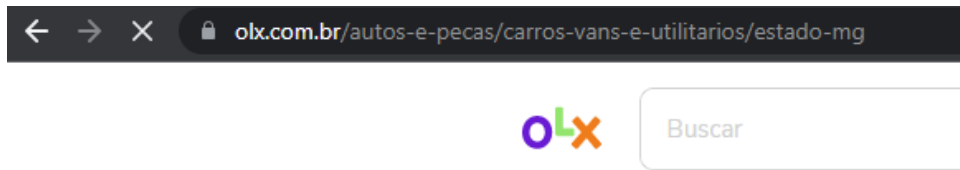


Figura 1: primeira página de anúncios na Olx.

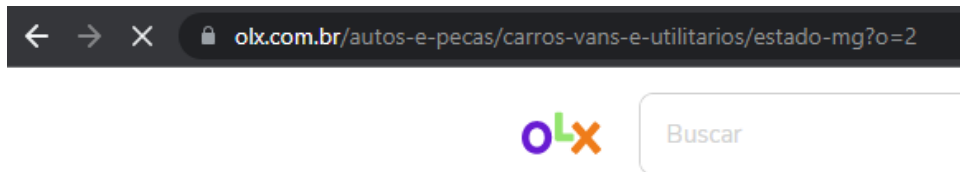


Figura2: segunda página de anúncios na Olx.

O grande problema dessa navegação na Olx é que são exibidas no máximo as 100 primeiras páginas de 50 anúncios cada, ou seja, dependendo do filtro realizado são exibidos no máximo 5000 anúncios. A Figura 3 mostra que ao chegar na 100ª página não é possível navegar para a próxima, mesmo havendo 93.127 anúncios, é possível visualizar apenas os 5000 primeiros automóveis.



Figura 3: Limitação de navegação nos anúncios na Olx.

Para a solução deste problema foi necessário segmentar a busca de tal forma que não haveria mais de 5000 anúncios por filtro. Para isso a busca é feita com três filtros e realizadas as combinações entre si. Os três filtros escolhidos foram as regiões de Minas Gerais, o ano do automóvel e o tipo de anúncio (se é particular ou anúncio profissional).

Dessa forma é buscado todos os anúncios de um ano, de uma região e de um tipo. Quando finalizado um destes itens do filtro, a iteração do algoritmo avança para o próximo item realizando assim todas as combinações de filtro e dessa forma foi garantido que não há mais de 5000 anúncios por filtro.

Na primeira parte do *script* que é feito o *Web scraping* são definidas as bibliotecas necessárias mais os filtros que são aplicados em cada busca. A biblioteca utilizada para o *Web scraping* é a *BeautifulSoup*. Na Figura 4 é possível observar as bibliotecas importadas, as 7 regiões de Minas Gerais; os tipos, onde 'c' é anúncio profissional e 'p' é par anúncio particular; e por fim o *array* de anos que será iterado na busca. Do ano de 1950 a 1990 a Olx permite filtrar de 5 em 5 anos, após 1990 o ano pode ser selecionado por unidade.

```

import requests
from bs4 import BeautifulSoup
import re
import pandas as pd
import math
import time

headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) '+
            'AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 '+'
            Safari/537.36'}

regions = ['belo-horizonte-e-regiao', 'regiao-de-juiz-de-fora',
            'regiao-de-governador-valadares-e-teofilo-otoni',
            'regiao-de-uberlandia-e-uberaba',
            'regiao-de-pocos-de-caldas-e-varginha',
            'regiao-de-divinopolis',
            'regiao-de-montes-claros-e-diamantina']
types = ['c', 'p']
startYear = 1990
endYear = 2022
years = [1950, 1955, 1960, 1965, 1970, 1975, 1980, 1985]
for i in range (startYear, endYear + 1):
    years.append(i)

print(years)

[1950, 1955, 1960, 1965, 1970, 1975, 1980, 1985, 1990, 1991, 1992, 1993, 1994,
1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022]

```

Figura 4: Filtros e bibliotecas *Web scraping* Olx.

O próximo trecho de código na Figura 5 é responsável por fazer os *loops* entre os filtros anteriormente citados: iteração para cada região, para cada ano e por fim para cada tipo. Para cada combinação do filtro é buscado o número de páginas deste filtro através do método *countPages*. Dessa forma é possível saber o número de páginas a serem navegadas dividindo a quantidade de anúncios por 50, sendo este o último é o número de anúncios por página.

Após buscada o número de páginas de cada combinação de filtro é feita a busca dos anúncios de cada página através de método *getCars*. Este método possui um *loop for* que vai percorrer por todas as páginas da combinação do filtro entre região, ano e tipo de anúncio. Em cada iteração deste *loop* são buscados os automóveis da página e atribuída na variável *cars*.

Na última parte do método *getCars* há mais um *loop* que percorre cada automóvel da lista de veículos da página. Cada iteração desse *loop* faz a chamada no método *getAd* onde serão buscadas todas as informações do anúncio do automóvel.

```
def countPages(url):
    site = requests.get(url, headers=headers)
    soup = BeautifulSoup(site.content, 'html.parser')
    count_itemsStr = soup.find('span', class_='sc-1mi5vq6-0 dQb0E sc-ifAKCX fCbscF').get_text().strip()
    count_items = int(count_itemsStr.split(' ')[4].replace('.', ''))
    count_pages = math.ceil(count_items / 50)
    return count_pages

def getCars(_countPage, region, year, _type):
    for i in range(1, _countPage + 1):
        print(_countPage, i, region, year, _type)
        url_pag = f'https://mg.olx.com.br/{region}/autos-e-pecas/carros-vans-e-utilitarios/{year}?f={_type}&o={i}'
        site = requests.get(url_pag, headers=headers)
        soup = BeautifulSoup(site.content, 'html.parser')
        cars = soup.find_all('a', {"class": "sc-12rk7z2-1 huFwya sc-htoDjs fpYhGm"})
        for car in cars:
            time.sleep(1)
            getAd(car['href'], region, year, _type)

for region in regions:
    for year in years:
        for _type in types:
            url = f'https://mg.olx.com.br/{region}/autos-e-pecas/carros-vans-e-utilitarios/{year}?f={_type}'
            getCars(countPages(url), region, year, _type)
```

Figura 5: Script para percorrer as páginas dos filtros.

A última parte do Script de *Web scraping* dos anúncios de automóveis da Olx é composta pelo método *getAd* da Figura 6 abaixo. Esse método recebe a *url* do anúncio e é responsável por entrar na página do anúncio e extrair as informações do automóvel que está sendo anunciado. Este método utilizará das funções das bibliotecas de *Web scraping* do Python para extrair informações de marca, modelo e características dos automóveis; e também informações relativas ao anúncio, como por exemplo, preço, data de publicação, localização e etc.

```

def getAd(urlAd, region, year, _type):
    site = requests.get(urlAd, headers=headers)
    soupAdd = BeautifulSoup(site.content, 'html.parser')

    publishDateElement = soupAdd.find('span', class_='ad__sc-10q8jzc-0 hSzkck sc-ifAKCX fizSrB')
    publishDate = publishDateElement.get_text().strip() if publishDateElement is not None else None

    codeElement = soupAdd.find('span', class_='ad__sc-16iz3i7-0 bTSFx0 sc-ifAKCX fizSrB')
    code = codeElement.get_text().strip() if codeElement is not None else None

    priceElement = soupAdd.find('h2', class_='ad__sc-12l420o-1 cuGsv0 sc-drMfKT fbofhg')
    price = priceElement.get_text().strip() if priceElement is not None else None

    diffsElements = map(lambda el: el.get_text().strip(), soupAdd.find_all('span', class_='info sc-iqzUVk leQbaP'))
    diffsList = list(diffsElements)

    divParentSpecs = soupAdd.find('div', class_='sc-hmzhuo ad__sc-1g2w54p-1 liYUIw sc-jTzLTM iwtNni')
    divSpecs = divParentSpecs.find('div', class_='sc-bwzfxH ad__h3us20-0 ikHgMx') if divParentSpecs is not None else None
    specsElements = divSpecs.find_all('div', {"class": "sc-hmzhuo ccYvDB sc-jTzLTM iwtNni"}) if divSpecs is not None else []

    specsdict = {}

    for specElement in specsElements:
        childs = specElement.findChildren()
        keySpec = childs[0].get_text().strip()
        valueSpec = childs[1].get_text().strip()
        specsdict.update({keySpec: valueSpec})

    divExtras = soupAdd.find('div', class_='sc-bwzfxH ad__h3us20-0 cyymI1')
    divsExtras = []

    if(divExtras is not None):
        divsExtras = divExtras.find_all('span', class_='ad__sc-1g2w54p-0 eDzIHZ sc-ifAKCX cmFKIN')

    extrasElements = map(lambda el: el.get_text().strip(), divsExtras)
    extrasList = list(extrasElements)

    divLocalParent = soupAdd.find('div', class_='ad__h3us20-6 crHfst')
    divLocal = divLocalParent.find('div', class_='sc-bwzfxH ad__h3us20-0 ikHgMx') if divLocalParent is not None else None
    divsItemsLocal = divLocal.find_all('div', {"class": "ad__duvuxf-0 ad__h3us20-0 kUfvdA"}) if divLocal is not None else []
    divCity = divsItemsLocal[1] if len(divsItemsLocal) > 1 else None
    city = divCity.find('dd', class_='ad__sc-1f2ug0x-1 cpGpXB sc-ifAKCX kaNiaQ').get_text().strip() if divCity is not None else None

    typeDesc = 'Particular' if _type == 'p' else 'Profissional'

    keys = []
    values = []

    for key in specsdict:
        keys.append(key)
        values.append(specsdict.get(key))

    row = f'{{publishDate}};{{code}};{{price}};{",".join(diffsList)};{",".join(values)};{",".join(keys)};{",".join(extrasList)};{{typeDesc}}'

    with open(f'cars_{region}_{year}.csv', 'a', encoding='utf-8') as fp:
        fp.write(row+'\n')

```

Figura 6: Método de extração das informações do anúncio Olx.

Este método salva os anúncios em arquivos .csv separados pela região e ano do anúncio. Dessa forma ao se deparar com possível problema de conexão de rede, ou qualquer problema que houvesse a interrupção do script de extração não seria necessário começar desde o início. Basta analisar o último arquivo gerado e então seria necessário reiniciar a extração a partir deste. A Figura 7 mostra como são armazenados os arquivos de anúncios da Olx.




















Name	Date modified	Type	Size
 cars_belo-horizonte-e-regiao_2014	08/11/2022 12:34	Planilha OpenOffice....	1.842 KB
 cars_belo-horizonte-e-regiao_2015	08/11/2022 14:11	Planilha OpenOffice....	1.705 KB
 cars_belo-horizonte-e-regiao_2016	08/11/2022 15:31	Planilha OpenOffice....	1.469 KB
 cars_belo-horizonte-e-regiao_2017	08/11/2022 16:41	Planilha OpenOffice....	1.276 KB
 cars_belo-horizonte-e-regiao_2018	08/11/2022 18:22	Planilha OpenOffice....	1.850 KB
 cars_belo-horizonte-e-regiao_2019	08/11/2022 20:16	Planilha OpenOffice....	2.081 KB
 cars_belo-horizonte-e-regiao_2020	08/11/2022 22:33	Planilha OpenOffice....	2.428 KB
 cars_belo-horizonte-e-regiao_2021	09/11/2022 10:42	Planilha OpenOffice....	1.860 KB
 cars_belo-horizonte-e-regiao_2022	09/11/2022 11:40	Planilha OpenOffice....	1.072 KB
 cars_regiao-de-juiz-de-fora_1950	09/11/2022 13:44	Planilha OpenOffice....	2 KB
 cars_regiao-de-juiz-de-fora_1955	09/11/2022 13:44	Planilha OpenOffice....	1 KB
 cars_regiao-de-juiz-de-fora_1960	09/11/2022 13:44	Planilha OpenOffice....	2 KB
 cars_regiao-de-juiz-de-fora_1965	09/11/2022 13:45	Planilha OpenOffice....	7 KB
 cars_regiao-de-juiz-de-fora_1970	09/11/2022 13:45	Planilha OpenOffice....	14 KB
 cars_regiao-de-juiz-de-fora_1975	09/11/2022 13:47	Planilha OpenOffice....	28 KB
 cars_regiao-de-juiz-de-fora_1980	09/11/2022 13:48	Planilha OpenOffice....	13 KB
 cars_regiao-de-juiz-de-fora_1985	09/11/2022 13:48	Planilha OpenOffice....	3 KB
 cars_regiao-de-juiz-de-fora_1990	09/11/2022 13:49	Planilha OpenOffice....	12 KB
 cars_regiao-de-juiz-de-fora_1991	09/11/2022 13:50	Planilha OpenOffice....	8 KB

Figura 7: Estrutura de arquivos de anúncios extraídos da Olx.

Na Tabela 1 abaixo estão todos os campos que foram extraídos pelo *Web scraping* de anúncios de automóveis na Olx.

Nome da coluna/campo	Descrição	Tipo
publish_date	Data e hora da publicação do anúncio.	string
code	Código do anúncio.	string
price	Preço do automóvel anunciado.	double
sell_diffs	Diferenciais atrativos para a venda separados por vírgula, por exemplo 'IPVA Pago', 'Aceita troca', etc.	string
attrs_values	Valores dos atributos (marca, modelo, ano, etc) do automóvel anunciado separados por vírgula.	string
attrs_keys	Chaves dos atributos (marca, modelo, ano, etc) do automóvel anunciado separados por vírgula.	string

items	Itens de série e opcionais presentes no veículo separados por vírgula.	string
type	Tipo de anúncio se é particular ou profissional.	string
city	Cidade onde se encontra o veículo anunciado.	string
url	URL do anúncio.	string
read_date	Data de leitura dos dados do anúncio.	date

Tabela 1: Campos do *dataset* extraído de anúncios da Olx.

2.3. Seminovos

Os dados dos anúncios de veículos da plataforma Seminovos foram extraídos no dia 29 de novembro de 2022 e a página principal para navegação nesses anúncios é: <https://seminovos.com.br>.

Os anúncios do portal Seminovos não são paginados. Para carregar os anúncios sob demanda a plataforma utiliza a forma do *infinite scroll*, onde à medida que o usuário rola a página para baixo mais anúncios são carregados, conforme Figura 8.

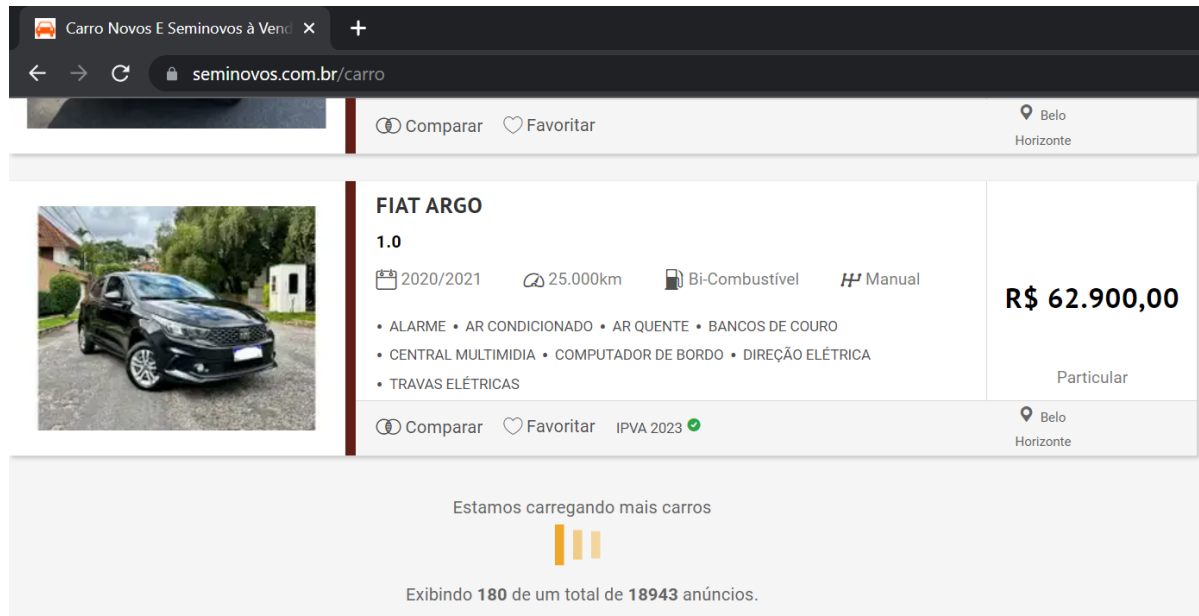


Figura 8: *Infinite scroll* para anúncios do Seminovos

O desafio desse tipo de *Web scraping* é que claramente os anúncios não são todos carregados logo que a página é aberta. É necessária uma ação de rolar a barra de *scroll* para baixo para que os próximos anúncios sejam carregados. Além disso, carregar todos os anúncios de uma única vez poderia ser um problema uma vez que milhares de anúncios sobrecarregariam a página.

Para fazer carregamento parciais dos anúncios foram utilizados filtros. Esses filtros são tipo de anúncio (anúncio particular ou profissional), ano do automóvel e marca do veículo. Dessa forma, são feitas todas as combinações entre esses três filtros. A cada combinação são buscados todos os anúncios, e quando finalizado passa-se para a próxima combinação.

Para carregar todos os anúncios de uma combinação de filtro foram utilizadas duas bibliotecas: *ChromeDriverManager* e *Selenium*. Essas duas bibliotecas em conjunto vão literalmente abrir o navegador Chrome na URL informada e executar comandos via *script* na página. Dessa forma, esses comandos, principalmente de *scroll*, vão permitir carregar todos os anúncios dinamicamente.

Na Figura 8 abaixo tem-se a parte inicial do *script* de *Web scraping* de anúncios do Seminovos. Primeiramente as declarações das bibliotecas utilizadas e logo após os filtros de

ano, marcas e tipos. A lista de marcas é obtida do combo de filtro por marcas e armazenada na variável *listBrands*.

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
import time
from bs4 import BeautifulSoup
import requests

driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

driver.get("https://seminovos.com.br")

time.sleep(3)

soup = BeautifulSoup(driver.page_source, "html.parser")
selectBrands = soup.find('select', {"id": "marca"})

brands = map(lambda el: el.get_text().strip(), selectBrands.findChildren())
_listBrands = list(brands)

listBrands = list(filter(lambda x: x != 'Selecione a marca' and x != '-', _listBrands))
listBrands = list(set(listBrands))
listBrands = list(map(lambda b: b.replace(" ", "-").lower(), listBrands))
listBrands.sort()

yearsRange = [1925, 2000, 2005, 2010, 2012, 2014, 2016, 2018, 2020, 2022]
types = ['origem-particular', 'origem-revenda']
```

Figura 8: Filtros e bibliotecas *Web scraping* Seminovos.

O próximo trecho de código é responsável por realizar os *loops* de iteração entre os três filtros anteriormente definidos, promovendo todas as combinações entre eles. A cada combinação gerada é chamado o método *infiniteScroll*, responsável por carregar todos os anúncios de uma combinação de filtros.

```
for brand in listBrands:
    for indexYear in range(0, len(yearsRange) - 1):
        for _type in types:
            print(str(yearsRange[indexYear]) + ' - ' + str(yearsRange[indexYear + 1]) + ' - ' + brand)
            infiniteScroll(brand, yearsRange[indexYear], yearsRange[indexYear + 1], _type)
```

Figura 9: Iteração entre os filtros definidos para busca de anúncios Seminovos.

O método *infiniteScroll* da Figura 10 é responsável por executar os comandos do *scroll* para carregar os anúncios dos filtros fornecidos. Este método é basicamente composto por um *loop while* que enquanto ainda está sendo carregado novos anúncios, permanece executando o comando *Javascript window.scrollTo*, rolando a barra de *scroll* até o fim da página.

Ao terminar de carregar todos os anúncios, os automóveis são extraídos e armazenados na variável *cars*. O fim deste método assim é um último *loop* que vai percorrer por cada anúncio, extrair a URL deste e passar para o método *getAdCar*, que será responsável por extrair as informações e atributos do carro em anúncio

```

def infiniteScroll(brand, yearStart, yearEnd, _type):
    url = f'https://seminovos.com.br/carro/{brand}/ano-{yearStart}-{yearEnd}/{_type}'
    driver.get(url)
    driver.execute_script('window.scrollTo(0, 0);')

    qntCurrent = 0
    qntTotal = 1
    qntScroll = 6000
    previous = qntScroll
    noAd = False

    while qntCurrent < qntTotal:
        previousCurrent = qntCurrent
        time.sleep(3)

        driver.execute_script('window.scrollTo(0, document.body.scrollHeight - 1000);')
        previous += qntScroll

        soup = BeautifulSoup(driver.page_source, "html.parser")
        divTotal = soup.find('div', class_='total-anuncios')
        sectionNoAd = soup.find('section', class_='nenhum-reseultado')

        if divTotal is None or sectionNoAd is not None:
            noAd = True
            break;

        strTotal = divTotal.get_text().strip()
        strSplitted = strTotal.split(' ')
        splittedFilter = filter(lambda x: x.isnumeric(), strSplitted)
        listFilter = list(splittedFilter)

        qntCurrent = int(listFilter[0])
        qntTotal = int(listFilter[1])

        if previousCurrent == qntCurrent:
            break

    if not noAd:
        soup = BeautifulSoup(driver.page_source, "html.parser")
        cars = soup.find_all('div', class_='anuncio-thumb-new anuncio-modo-list')
        for car in cars:
            urlAd = car.find_all("a")[0]["href"].split("?")[0]
            url = f'https://seminovos.com.br/{urlAd}'
            getAdCar(url, brand, yearStart, yearEnd, _type)

```

Figura 10: Método para buscar anúncios do *infinite scroll* do Seminovos.

A parte final do *Script* de *Web scraping* dos anúncios de automóveis do portal Seminovos é composta pelo método *getAdCar* da Figura 11. Esse método recebe a *url* do anúncio e é responsável por entrar na página do anúncio e extrair as informações do automóvel. Este método utilizará das funções das bibliotecas *BeautifulSoup* de *Web scraping* do Python para extrair informações de marca, modelo e características dos automóveis; e também informações relativas ao anúncio, como por exemplo, preço, data de publicação, localização e etc.

```

def getAdCar(urlAd, brand, yearStart, yearEnd, _type):
    print(urlAd)
    headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 S

    site = requests.get(urlAd, headers=headers)
    soupAd = BeautifulSoup(site.content, 'html.parser')

    divInfos = soupAd.find('div', class_='part-infos')

    if(divInfos is None):
        return

    divInfosChildren = divInfos.findChildren('div')

    publicacao = divInfosChildren[0].get_text().strip().split(' ')[2]
    codigo = divInfosChildren[2].get_text().strip().split(' ')[1]

    divMarcaModeloValor = soupAd.find('div', class_='part-marca-modelo-valor')
    spanValor = divMarcaModeloValor.find('span', class_='valor') if divMarcaModeloValor is not None else None

    valor = spanValor.get_text().strip().split(' ')[1] if spanValor is not None else None
    motorizacao = divMarcaModeloValor.find('span', class_='desc').get_text().strip().split(' ')[0] if divMarcaModeloValor is not
    marcaModelo = divMarcaModeloValor.find('h1').get_text().strip() if divMarcaModeloValor is not None else None

    divItemsDetalhes = soupAd.find('div', class_='part-items-detalhes-icone')
    items = divItemsDetalhes.find_all('div', class_='item')
    itemsKeys = []
    itemsValues = []

    for item in items:
        itemsKeys.append(item.find('div', class_='campo').get_text().strip())
        itemsValues.append(item.find('span', class_='valor').get_text().strip())

    ulAcessorios = soupAd.find('ul', class_='lista-acessorios')
    itensAcessorios = ulAcessorios.find_all('span', class_='description-print')
    listAcessorios = list(map(lambda el: el.get_text().strip(), itensAcessorios))

    cityElement = soupAd.find('address')

    city = cityElement.get_text().strip() if cityElement is not None else None

    typeDesc = 'Particular' if _type == 'origem-particular' else 'Profissional'

    row = f'{codigo};{publicacao};{valor};{motorizacao};{marcaModelo};{",".join(itemsKeys)};{",".join(itemsValues)};{",".join(listAcessorios)};{city};{typeDesc}'

    with open(f'seminovos_{brand}_{yearStart}_{yearEnd}.csv', 'a', encoding='utf-8') as fp:
        fp.write(row+'\n')

```

Figura 11: Método de extração das informações do anúncio Seminovos.

Da mesma forma que os anúncios da Olx, este método salva os anúncios da Seminovos em arquivos .csv separados pelos filtros marca e ano do automóvel. Dessa forma ao se deparar com possível problema de conexão de rede, ou qualquer problema que houvesse a interrupção do script de extração não seria necessário começar desde o início. Basta analisar o último arquivo gerado e então seria necessário reiniciar a extração a partir deste. A Figura 12 mostra como são armazenados os arquivos de anúncios da Seminovos.















Name	Date modified	Type	Size
 seminovos_honda_2005_2010	29/11/2022 12:58	Planilha OpenOffice....	20 KB
 seminovos_honda_2010_2012	29/11/2022 12:59	Planilha OpenOffice....	21 KB
 seminovos_honda_2012_2014	29/11/2022 13:00	Planilha OpenOffice....	21 KB
 seminovos_honda_2014_2016	29/11/2022 13:02	Planilha OpenOffice....	21 KB
 seminovos_honda_2016_2018	29/11/2022 13:03	Planilha OpenOffice....	22 KB
 seminovos_honda_2018_2020	29/11/2022 13:04	Planilha OpenOffice....	24 KB
 seminovos_honda_2020_2022	29/11/2022 13:05	Planilha OpenOffice....	25 KB
 seminovos_hyundai_2005_2010	29/11/2022 13:08	Planilha OpenOffice....	13 KB
 seminovos_hyundai_2010_2012	29/11/2022 13:09	Planilha OpenOffice....	23 KB
 seminovos_hyundai_2012_2014	29/11/2022 13:10	Planilha OpenOffice....	22 KB
 seminovos_hyundai_2014_2016	29/11/2022 13:12	Planilha OpenOffice....	21 KB
 seminovos_hyundai_2016_2018	29/11/2022 13:13	Planilha OpenOffice....	20 KB
 seminovos_hyundai_2018_2020	29/11/2022 13:14	Planilha OpenOffice....	22 KB
 seminovos_hyundai_2020_2022	29/11/2022 13:15	Planilha OpenOffice....	22 KB

Figura 12: Estrutura de arquivos de anúncios extraídos do Seminovos.

Na Tabela 2 abaixo estão todos os campos que foram extraídos pelo *Web scraping* de anúncios de automóveis no Seminovos.

Nome da coluna/campo	Descrição	Tipo
code	Código do anúncio.	string
publish_date	Data da publicação do anúncio.	date
price	Preço do automóvel anunciado;	double
motor	Motorização do veículo (motor 1.0 L, 1.6 L)	double
brand_model	Marca e modelo do automóvel	string
attrs_keys	Chaves dos atributos (quilometragem, câmbio, ano, etc.) do automóvel anunciado separados por vírgula.	string
attrs_values	Valores dos atributos (quilometragem, câmbio, ano, etc.) do automóvel anunciado separados por vírgula.	string

items	Itens de série e opcionais presentes no veículo separados por vírgula.	string
type	Tipo de anúncio se é particular ou profissional.	string
address	Endereço/cidade onde se encontra o veículo anunciado.	string
read_date	Data de leitura dos dados do anúncio.	date

Tabela 2: Campos do *dataset* extraído de anúncios do Seminovos.

3. Processamento/Tratamento de Dados

A etapa de processamento e tratamento de dados foi dividida em 4 etapas: Junção dos arquivos gerados na seção 2; extração dos nomes e modelos dos automóveis; extração dos atributos dos automóveis/anúncio; e por último o tratamento de dados faltantes, *outliers* e dados divergentes após a extração. Cada uma dessas etapas pode ter um ou mais *steps* gerados, que são processamentos intermediários gerados, para que não seja necessário todo um processamento após uma etapa já consolidada.

Os arquivos intermediários, bem como os *scripts* de cada etapa podem ser encontrados no *link* do repositório Github disponibilizado na seção 8, dentro da pasta **processamento_tratamento**. Nesta seção serão explicados os principais métodos, pois é grande o volume de código para o *script* de tratamentos de dados.

Na seção 3.1, a parte de junção de arquivos gera os arquivos para o *step1*. O *Step1* promove a junção dos arquivos gerados na seção anterior, em um único só arquivo de *dataset* por plataforma. A seção 3.2 é composta pelos *steps2*, *step3* e *step4*: extração de nomes de marcas e modelos; padronização de nome de marcas; e padronização de nomes de modelos, respectivamente. A seção 3.3 é responsável por extrair e padronizar os outros atributos do veículo/anúncio, gerando o *step5*. Por fim, a seção 3.4 é responsável por tratar dados em branco e inconsistências, gerando o *step6*. Todos os passos e as chamadas dos métodos principais são sumarizados na Figura 13 abaixo, arquivo de script **ads_etl.py**


```

# step 1
joinFiles()
# step 1

# step 2
executeCombosScraping()
extractBrandsModelAndSaveFile()
# step 2

# step 3
df_olx, df_seminovos = getDataFrames('step_2_brands_models_extracted')
printDiffsBrands(df_olx, df_seminovos)
renameBrandsAndSaveFile(df_olx, df_seminovos)
# step 3

# step 4
df_olx, df_seminovos = getDataFrames('step_3_brands_renamed')
generateModelsDiffs(df_olx, df_seminovos)
renameModelsAndSaveFile(df_olx, df_seminovos)
# step 4

# step 5
df_olx, df_seminovos = getDataFrames('step_4_models_renamed')
print_attributes_olx(df_olx,)
print_attributes_seminovos(df_seminovos)
olx_attrs_etl(df_olx)
seminovos_attrs_etl(df_seminovos)
join_dfs(df_olx, df_seminovos)
# step 5

# step 6
df = getDataFrameJoined('step_5_items_extracted')
removeDuplicate(df)
outliers(df)
fill_category_missing_values(df)
fill_value_missing_values(df)
process_completo_final_dataframe(df)
df = delete_missing_cities(df)
write_csv(df)
# step 6

```

Figura 13: Métodos de cada passo gerado no processamento dos dados. Arquivo inicial:

3.1. Junção dos Arquivos

Como dito na seção anterior, os *datasets* extraídos através de *Web scraping* tanto da plataforma Olx quanto do portal Seminovos foram separados em vários arquivos, em detrimento de problemas de carregar muitos arquivos e como forma de facilitar o *script* de extração quando houvesse alguma falha.

Dessa forma, a primeira etapa do processamento e tratamento de dados foi juntar

esses arquivos das Figuras 7 e 12 em somente um arquivo por plataforma, pois ainda há diferença entre os dados coletados em cada portal. A Figura 14 mostra o *script* executado para fazer a junção dos arquivos tanto da Olx quanto do Seminovos, se diferem apenas na pasta de origem, arquivo final de destino e cabeçalho de dados, variáveis *dir_path*, *fileName* e *headers*, respectivamente. Método **joinFiles()** do arquivo **join_files.py**

```
import os

def readFileAndWrite(path, fileName):
    file = open(path, 'r', encoding="utf8")
    lines = file.readlines()

    for index, line in enumerate(lines):
        with open(fileName, 'a', encoding='utf-8') as fp:
            fp.write(line)
    file.close()

def join(dir_path, fileName, headers):
    with open(fileName, 'a', encoding='utf-8') as fp:
        fp.write(headers)

    for path in os.listdir(dir_path):
        if os.path.isfile(os.path.join(dir_path, path)):
            readFileAndWrite(os.path.join(dir_path, path), fileName)

def joinFiles():
    join('olx_files', 'step_1_joined_files\\olx_all.csv', 'publish_date;code;price;sell_diffs;attrs_')
    join('seminovos_files', 'step_1_joined_files\\seminovos_all.csv', 'code;publish_date;price;motor')
```

Fi-

gura 14: *Script* de junção de arquivos.

Neste momento é realizado o primeiro passo: **step_1_joined_files**, gerando os arquivos *olx_all.csv* e *seminovos_all.csv*, neste momento com 87.317 e 5.752 registros respectivamente cada um.

Cada plataforma possui sua especificidade em exibir os atributos de um automóvel, bem como considerar que alguns itens são relevantes ou não. Por exemplo, a Olx coloca a marca e modelo do carro juntamente com a descrição da versão do automóvel, enquanto na Seminovos marca e modelos são obtidos separadamente. No Seminovos é possível visualizar ano de fabricação e modelo do carro, na Olx é possível apenas o ano modelo, que é o comumente utilizado na precificação. Um último exemplo, é na Seminovos é possível informar quantos *Air bags* tem o carro; na Olx é uma informação binária: possui ou não.

Dessa forma, as próximas etapas de processamento e tratamento de dados serão analisar as informações em comum dos anúncios da Olx e Seminovos, padronizar nomes de marcas e modelos dos automóveis padronizar nomes de atributos dos anúncios, de tal forma que no fim os dois *datasets* possuirão os mesmos campos e podendo ser unificados em apenas um.

3.2. Extração e Padronização de Nomes de Marcas e Modelos

A próxima etapa de processamento e tratamento dos dados agora é a padronização dos nomes das marcas e modelos dos automóveis. Por exemplo, na Olx temos as marcas Mercedes-Benz e Kia Motors, enquanto no portal Seminovos esses nomes são Mercedes Bens e Kia, respectivamente.

Dessa forma, é necessário fazer um tratamento de caracteres nos textos de marcas e modelos, e até mesmo identificar manualmente nomes diferentes, mas que representam o mesmo, fazendo um *de/para* e padronizar os nomes. Além da padronização de nomes também é necessário extrair separadamente esses valores, pois por exemplo, na Seminovos a marca e modelo são uma única *string*; e a Olx ainda possui a versão do modelo juntamente com o modelo do carro, não sendo possível saber onde começa e termina exatamente o modelo do automóvel.

Foi realizado assim um *Web Scraping* para a extração das marcas e modelos da Olx e Seminovos para a padronização do nome. O método inicial para isso é o **executeCombosScraping()**, e todo esse *script* está contido no arquivo **brands_models_combo_scraping.py**.

Essas informações são salvas na pasta *brands_models_scraping* e posteriormente o método **extractBrandsModelAndSaveFile()** do arquivo **brands_models_extract.py** os utiliza para fazer a extração dos nomes e marcas de modelos que não estão estruturados. Neste momento é gerado o **step_2_brands_models_extracted**.

Métodos principais *brands_models_extract.py*:

- `getBrandsScrapingOlx()`: Faz o *Web Scraping* das marcas e modelos os automóveis da Olx e armazenando-os na pasta `brands_models_scraping\olx`.
- `getBrandsScrapingSeminovos()`: Faz o *Web Scraping* apenas das marcas dos automóveis da Seminovos. Obtendo a marca o restante da *string* é composto apenas pelo modelo. Nomes da marcas são armazenados e armazenando-os na pasta `brands_models_scraping\seminovos\brands.txt`.

O próximo passo é fazer a padronização dos nomes das marcas dos automóveis. Esse será o *step3*, e é formado por duas funções principais **`printDiffsBrands()`** e **`renameBrandsAndSaveFile()`**, ambas do arquivo de *scripts* **`diffs_rename_brands_models.py`**. A primeira será responsável por mostrar na tela (Figura 15) as diferenças entre as marcas de Olx e Seminovos, removendo caracteres especiais, espaços e letras em caixa alta para facilitar a comparação. Após feita a verificação manual, o segundo método é responsável por renomear as marcas e salvar essa etapa como mais um passo, gerando assim o **`step_3_brands_renamed`**.

Na Figura 15 pode-se notar que há várias marcas que estão no *dataset* de uma plataforma, mas não no da outra. Mas 4 marcas em específico são apenas grafias diferentes. Sendo essas CHEVROLET e GMCHEVROLET, KIA e KIAMOTORS, VOLKSWAGEN e VW VOLKSWAGEN, WILLYS e WILLYSOVERLAND.

```
Seminovos Not In Olx:

['BENTLEY', 'BUGGY', 'CHEVROLET', 'COMIL', 'FARUS', 'KIA', 'LINCOLN', 'MPLAFER', 'SANTAMATILDE', 'VOLKSWAGEN', 'WILLYS']

Olx Not In Seminovos:

['ACURA', 'AGRALE', 'AMERICAR', 'AMGEN', 'ASIAMOTORS', 'BABY', 'BRM', 'BUGRE', 'CABMOTORS', 'CBTJIPE', 'CHANA', 'DKWEMAG', 'ENVEMO', 'GEELY', 'GMCHEVROLET', 'GREATWALL', 'HAFEI', 'HITECHELETRIC', 'ISUZU', 'JPX', 'KIAMOTORS', 'LADA', 'LOBINI', 'MAHINDRA', 'MAZDA', 'MG', 'SEAT', 'SHINERAY', 'TAC', 'VENTURA', 'VWVOLKSWAGEN', 'WAKE', 'WALK', 'WILLYSOVERLAND']
```

Figura 15: Diferenças entre marcas nas plataformas.

Após renomeados os nomes das marcas dos automóveis a próxima etapa é fazer o mesmo procedimento, porém para os nomes dos modelos. Essa etapa é composta por dois principais métodos **generateModelsDiffs()** e **renameModelsAndSaveFile()**. O primeiro método é responsável por analisar os modelos da mesma marca entre as plataformas e gerar as diferenças entre eles, ou seja, quais modelos para uma marca X contém na Olx e que não contém na Seminovos e vice-versa. Essas diferenças são armazenadas em arquivos por marca (pasta *models_diff*), onde cada marca uma possui dois arquivos *existsInOlxNotInSeminovos* e *existsInSeminovosNotInOlx*. No exemplo da Figura 16, destacou-se a marca Fiat, e percebe-se que por exemplo na Olx existe o modelo GRANDSIENA, enquanto na Seminovos o mesmo modelo é SIENAGRAND. Ou mesmo na Seminovos existem diferenças entre o modelo na Strada (cabine dupla, estendida e simples), na Olx existe apenas Strada.

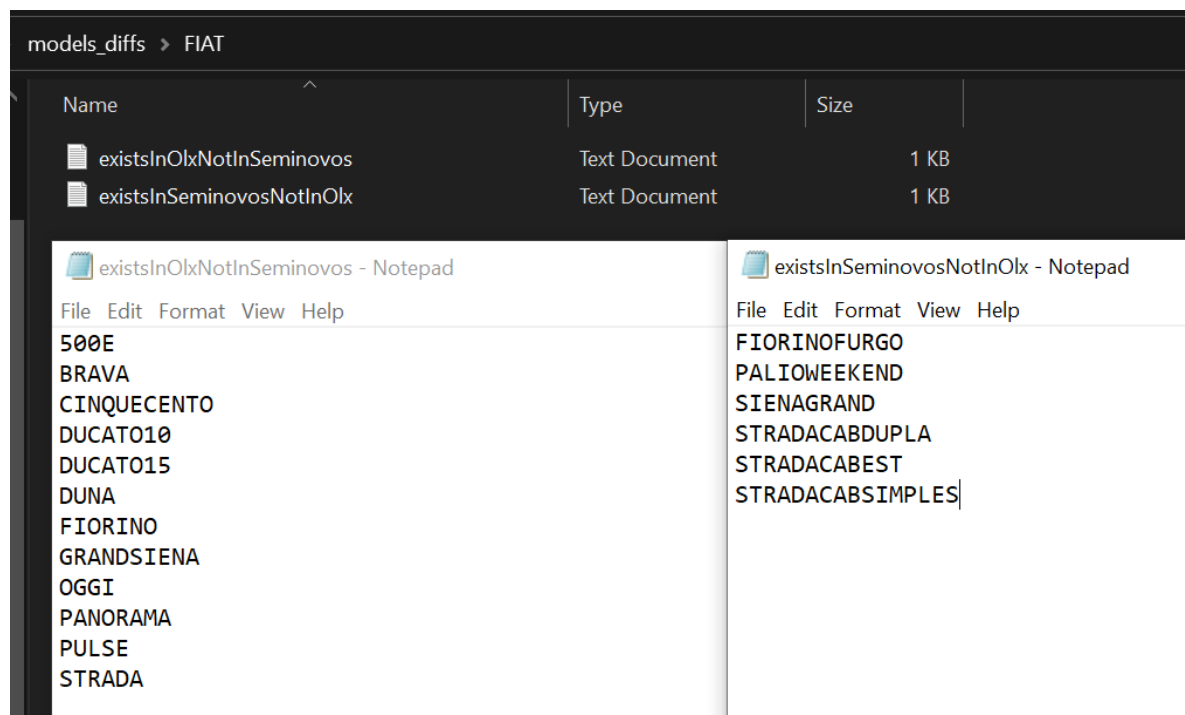


Figura 16: Diferença entre modelos de uma mesma marca.

Analisada a manualmente diferença, é construído então o método *renameModelsAndSaveFile()* onde será feito um de/para padronizando os nomes dos modelos dos automóveis.

Todas as alterações para padronização dos nomes dos modelos automóveis podem ser visualizadas dentro dos métodos `renameModels AndSaveFileSeminovos()` e `renameModelsAndSaveFileOlx()`, dentro do arquivo de *scripts* `diffs_rename_brands_models.py`. Essa é o `step_4_models_renamed`, o último passo da padronização dos nomes de marcas e modelos dos automóveis da Olx e Seminovos.

3.3. Extração e Padronização dos Atributos dos Automóveis

Boa parte dos atributos dos veículos e anúncios possuem atributos apresentados de formas diferente entre as duas plataformas. Por exemplo, o Seminovos diferencia a quantidade de *airbags* no carro, enquanto a Olx é uma informação binária: possui ou não. O preço do carro no Seminovos possui 'R\$', enquanto na Olx esse atributo é apenas composto por dígitos. Dessa forma essa seção mostrará o tratamento e extração dos dados em comum das duas plataformas, a fim de padronizar para posterior junção em único *dataset*.

Para fazer a análise manual dos atributos é necessário verificar todos os itens, chaves e valores presentes nos *datasets* Olx e Seminovos. Para isso foi criado dois métodos, `print_attributes_olx()` e `print_attributes_seminovos()`, Figura 17. O primeiro percorre os diferenciais de venda de anúncios da Olx, chaves dos atributos e itens de série de anúncios na Olx. O segundo tem o mesmo propósito, porém na Seminovos não existe a campo diferenciais de venda.

```

def print_attributes_olx(df_olx):

    set_diffs = set()
    set_keys_olx = set()
    set_items_olx = set()

    for index, row in df_olx.iterrows():
        sell_diffs = row['sell_diffs'].split(',')
        keys = row['attrs_keys'].split(',')
        items = row['items'].split(',')

        for set_diff in sell_diffs:
            set_diffs.add(set_diff)
        for key in keys:
            set_keys_olx.add(key)
        for item in items:
            set_items_olx.add(item)

    print('\nDiferencias de venda Olx:\n')
    print(set_diffs)
    print('\nChaves atributos Olx:\n')
    print(set_keys_olx)
    print('\nItems Olx:\n')
    print(set_items_olx)

def print_attributes_seminovos(df_seminovos):

    set_keys_seminovos = set()
    set_items_seminovos = set()

    for index, row in df_seminovos.iterrows():
        keys = row['attrs_keys'].split(',')
        items = row['items'].split(',')

        for key in keys:
            set_keys_seminovos.add(key)
        for item in items:
            set_items_seminovos.add(item)

    print('\n\nChaves atributos Seminovos:\n')
    print(set_keys_seminovos)
    print('\nItems Seminovos:\n')
    print(set_items_seminovos)
    print('\n')

```

Figura 17: Métodos para análise dos atributos das plataformas Olx e Seminovos.

Dessa forma foram geradas para Olx as informações ‘Diferenciais de Venda Olx’, ‘Items Olx’ que são já valores, e ‘Chaves Atributos’ que não as chaves de atributos disponíveis. Para a seminovos foram gerados os valores ‘Itens Seminovos’ e as chaves de atributos ‘Chaves Atributos Seminovos’, conforme Figura 18.

```

Diferenciais de venda Olx:
{'', 'Carro Financiado', 'Aceita troca', 'IPVA pago', 'Carro de leilão', 'Com multa', 'Único dono', 'Não aceita troca'}

Chaves atributos Olx:
{'Modelo', 'Categoria', 'Ano', 'Final de placa', 'Portas', 'Direção', 'Câmbio', 'Marca', 'Quilometragem', 'Kit GNV', 'Potência do motor', 'Tipo de veículo', 'Cor', 'Combustível'}

Items Olx:
{'', 'Sensor de ré', 'Trava elétrica', 'Alarme', 'Câmera de ré', 'Direção hidráulica', 'Som', 'Air bag', 'Blindado', 'Ar condicionado', 'Vidro elétrico'}

Chaves atributos Seminovos:
{'Troca?', 'Leilão', 'Portas', 'Ano - Modelo', 'Câmbio', 'cor', 'Quilometragem', 'Placa', 'Combustível'}

Items Seminovos:
{'AIR BAGS 10', 'CONTROLE TRAÇÃO', 'COMPUTADOR DE BORDO', 'CÂMBIO AUTOMATIZADO', 'CD / MP3', 'CAPOTA MARÍTIMA', 'ALARME', 'TRAVAS ELÉTRICAS', 'DIREÇÃO HIDRÁULICA', 'AIR BAGS 7', 'AIR BAGS 2', 'AR QUENTE', 'PROTETOR DE CAÇAMBA', 'AIR BAGS 8', 'REBAIXADO', 'AIR BAGS 9', 'ASSIST. PARTIDA EM RAMP', 'AR CONDICIONADO', 'AWD', 'AIR BAGS 4', 'TETO-SOLAR', 'RODAS DE LIGA LEVE', 'CONVERSÍVEL', 'AIR BAGS 12', 'AIR BAGS 1', 'DESEMPAÇADOR', 'VOLANTE AJUSTÁVEL', 'TURBO', 'VOLANTE COMANDO MULTIMÍDIA', 'ENGATE', 'DVD', 'FARÓIS AUXILIARES', 'BANCOS DE COURO', 'EBD', 'BLINDADO', 'BANCO AJUSTE ALTURA', 'AIR BAGS 11', 'FARÓIS LED', 'TRAÇÃO 4x4', 'CONTROLE ESTABILIDADE', 'FARÓIS AUXILIARES', 'ABS', 'CÂMBIO MANUAL', 'CENTRAL MULTIMÍDIA', 'CÂMBIO DE RÉ', 'RETROVISORES ELÉTRICOS', 'LIMPADOR TRASEIRO', 'AIR BAGS 5', 'MP3 / USB', 'CÂMBIO AUTOMÁTICO', 'AIR BAGS 6', '7 LUGARES', 'AR-COND. DIGITAL', 'GPS', 'PILOTO AUTOMÁTICO', 'DIREÇÃO ELÉTRICA', 'AIR BAGS 3', 'FAROL XENÔNIO', 'SENSOR DE ESTACIONAMENTO', 'VIDROS ELÉTRICOS'}

```

Figura 18: Item e chaves de atributos das plataformas Olx e Seminovos.

Os itens e diferenciais de vendas já são os valores finais, porém, é necessário saber os valores presentes dentro das chaves dos atributos de cada plataforma. Por exemplo, quais os valores de combustível, câmbio e direção dos anúncios de Olx e Seminovos? Para isso foi criado mais um método auxiliar, *printValuesFromKey()*, que dado o *dataset* e a chave do atributo são gerados todos os valores presentes no *dataset*, Figura 19. Pegando como por exemplo o câmbio dos anúncios de ambas as plataformas, temos os seguintes valores na Figura 20.


```
def printValuesFromKey(df, key):
    setValues = set()

    for index, row in df.iterrows():
        keys = row['attrs_keys'].split(',')
        values = row['attrs_values'].split(',')

        indexKey = None

        try:
            indexKey = keys.index(key)
        except ValueError:
            indexKey = -1

        if indexKey >= 0:
            setValues.add(values[indexKey])

    print(setValues)
```

Figura 19: Método para geração de valores de uma chave no *dataset*.

```
print('\nValores Câmbio Olx:\n')
printValuesFromKey(df_olx, 'Câmbio')
print('\nValores Câmbio Seminovos:\n')
printValuesFromKey(df_seminovos, 'Câmbio')
print('\n')
```

```
Valores Câmbio Olx:

{'Automático', 'Semi-Automático', 'Manual'}

Valores Câmbio Seminovos:

{'Não Informado', 'Automático', 'Automatizado', 'Manual'}
```

Figura 20: Valores do atributo Câmbio nas plataformas Olx e Seminovos.

Somente no exemplo acima é possível visualizar que é necessário fazer um tratamento nos nomes do câmbio ‘Semi-Automático’ e ‘Automatizado’ que representam a mesma informação.

Todos os métodos anteriores serão utilizados para a padronização dos nomes dos atributos nos dois principais métodos dessa etapa de tratamento: *olx_attrs_etl()* e

seminovos_attrs_etl()). Ambos os métodos são iterações entre os anúncios de cada plataforma, fazendo a extração e padronização dos itens: preço; tempo em dias anunciado; aceita troca; possui passagem por leilão; quantidade de portas; tipo de câmbio; tipo de direção; quilometragem rodada; ano do automóvel; tipo de combustível; potência do motor (litragem); cor do veículo; possui vidros elétricos; possui airbags; possui sensor de estacionamento; possui som; é blindado ou não; possui alarme; possui câmera de ré; possui ar condicionado; possui travas elétricas; se é completo de itens básico; a cidade onde está anunciado o automóvel. Os métodos dos processamentos desses itens estão enumerados na Figura 21 e Figura 22, Olx e Seminovos, respectivamente, de números 1 até 23.

```
def olx_attrs_etl(df_olx):
    ad_new_cols(df_olx)
    cities = getCitiesProcessed()

    for index, row in df_olx.iterrows():
        1 process_price_olx(row, index, df_olx)
        2 process_posted_days_olx(row, index, df_olx)
        3 process_aceita_troca_olx(row, index, df_olx)
        4 value_in_items(df_olx, index, row, 'leilao', 'sell_diffs', ['Carro de leilão'], 'Sim', 'Não')
        5 attr_in_keys(df_olx, index, row, 'portas', 'Portas', {'4 portas': '4portas', '2 portas': '2portas'}, '', False)
        6 attr_in_keys(df_olx, index, row, 'cambio', 'Câmbio', {'Manual': 'Manual', 'Automático': 'Automático', 'Semi-Automático': 'Automatizado'}, '', False)
        7 attr_in_keys(df_olx, index, row, 'direcao', 'Direção', {'Hidráulica': 'Hidráulica', 'Mecânica': 'Mecânica', 'Elétrica': 'Elétrica', 'Assistida': 'Assistida'}, '', False)
        8 process_km_olx(df_olx, index, row, 'quilometragem', 'Quilometragem')
        9 process_ano_olx(df_olx, index, row, 'ano', 'Ano')
        10 process_combustivel_olx(df_olx, index, row, 'combustivel', 'Combustível', 'Kit GNV')
        11 attr_in_keys(df_olx, index, row, 'motorizacao', 'Potência do motor', {'2.0 - 2.9': '2.0', '3.0 - 3.9': '3.0', '4.0 ou mais': '4.0'}, '', True)
        12 attr_in_keys(df_olx, index, row, 'cor', 'Cor', {'Outra': ''}, '', True)
        13 value_in_items(df_olx, index, row, 'vidros_eletricos', 'items', ['Vidro elétrico'], 'Sim', 'Não')
        14 value_in_items(df_olx, index, row, 'air_bag', 'items', ['Air bag'], 'Sim', 'Não')
        15 value_in_items(df_olx, index, row, 'sensor_estacionamento', 'items', ['Sensor de ré'], 'Sim', 'Não')
        16 value_in_items(df_olx, index, row, 'som', 'items', ['Som'], 'Sim', 'Não')
        17 value_in_items(df_olx, index, row, 'blindado', 'items', ['Blindado'], 'Sim', 'Não')
        18 value_in_items(df_olx, index, row, 'alarme', 'items', ['Alarme'], 'Sim', 'Não')
        19 value_in_items(df_olx, index, row, 'camera_re', 'items', ['Câmera de ré'], 'Sim', 'Não')
        20 value_in_items(df_olx, index, row, 'ar_condicionado', 'items', ['Ar condicionado'], 'Sim', 'Não')
        21 value_in_items(df_olx, index, row, 'trava_eletrica', 'items', ['Trava elétrica'], 'Sim', 'Não')
        22 process_completo(df_olx, index, row, 'completo')
        23 process_city_olx(df_olx, index, row, 'city_processed', cities)

    df_olx.to_csv('step_5_items_extracted\\olx.csv', sep=';', encoding='utf-8', index=False)
```

Figura 21: Chamadas dos métodos de processamento itens Olx.

```

def seminovos_attrs_etl(df_seminovos):
    ad_new_cols(df_seminovos)
    cities = getCitiesProcessed()

    for index, row in df_seminovos.iterrows():
        1 process_price_seminovos(row, index, df_seminovos)
        2 process_posted_days_seminovos(row, index, df_seminovos)
        3 attr_in_keys(df_seminovos, index, row, 'exchange', 'Troca?', {'Aceito Troca': 'Sim', 'Não Aceita Troca': 'Não'}, '', False)
        4 attr_in_keys(df_seminovos, index, row, 'leilao', 'Leilão', {'Proveniente de leilão': 'Sim'}, 'Não', False)
        5 attr_in_keys(df_seminovos, index, row, 'portas', 'Portas', {'0': '2portas', '1': '2portas', '2': '2portas', '3': '4portas', '4': '4portas', '5': '4portas'}, '2portas', False)
        6 attr_in_keys(df_seminovos, index, row, 'cambio', 'Câmbio', {'Manual': 'Manual', 'Automático': 'Automático', 'Automatizado': 'Automatizado'}, 'Não', False)
        7 process_direcao_seminovos(df_seminovos, index, row, 'direcao', 'items')
        8 process_km_seminovos(df_seminovos, index, row, 'quilometragem', 'Quilometragem')
        9 process_ano_seminovos(df_seminovos, index, row, 'ano', 'Ano - Modelo')
        10 process_combustivel_seminovos(df_seminovos, index, row, 'combustivel', 'Combustível')
        11 process_motor_seminovos(df_seminovos, index, row, 'motorizacao', 'Motorização')
        12 attr_in_keys(df_seminovos, index, row, 'cor', 'Cor', {}, '', True)
        13 value_in_items(df_seminovos, index, row, 'vidros_eletricos', 'items', ['VIDROS ELÉTRICOS'], 'Sim', 'Não')
        14 value_in_items(df_seminovos, index, row, 'air_bag', 'items', ['AIR BAGS 12', 'AIR BAGS 9', 'AIR BAGS 4', 'AIR BAGS 11', 'AIR BAGS 7', 'AIR BAGS 6'], 'Sim', 'Não')
        15 value_in_items(df_seminovos, index, row, 'sensor_estacionamento', 'items', ['SENSOR DE ESTACIONAMENTO'], 'Sim', 'Não')
        16 value_in_items(df_seminovos, index, row, 'som', 'items', ['DVD', 'MP3 / USB', 'CD / MP3', 'CENTRAL MULTIMÍDIA'], 'Sim', 'Não')
        17 value_in_items(df_seminovos, index, row, 'blindado', 'items', ['BLINDADO'], 'Sim', 'Não')
        18 value_in_items(df_seminovos, index, row, 'alarme', 'items', ['ALARME'], 'Sim', 'Não')
        19 value_in_items(df_seminovos, index, row, 'camera_re', 'items', ['CÂMERA DE RÉ'], 'Sim', 'Não')
        20 value_in_items(df_seminovos, index, row, 'ar_condicionado', 'items', ['AR CONDICIONADO', 'AR-COND. DIGITAL'], 'Sim', 'Não')
        21 value_in_items(df_seminovos, index, row, 'trava_eletrica', 'items', ['TRAVAS ELÉTRICAS'], 'Sim', 'Não')
        22 process_completo(df_seminovos, index, row, 'completo')
        23 process_city_seminovos(df_seminovos, index, row, 'city_processed', cities)

    df_seminovos.to_csv('step_5_items_extracted\\seminovos.csv', sep=';', encoding='utf-8', index=False)

```

Figura 22: Chamadas dos métodos de processamento itens Seminovos.

Alguns métodos possuem especificidades, por exemplo, processamento do preço, combustível e cidade, mas outros possuem métodos em comum para processamento. Para isso foi criado os métodos `attrs_in_keys()` e `value_in_items()`, que dado valores e/ou chaves, analisam dentro do conjunto informado se contém a informação ou não, podendo ainda fazer padronizações passadas como parâmetro.

Para extrair e padronizar as cidades dos anúncios Olx e Seminovos foi utilizada uma fonte externa com as cidades de Minas Gerais. Essa fonte foi necessária pois nomes de municípios são escritos de forma diferente, principalmente quando se tem numerais: 7 Lagoas/Sete Lagoas, 3 Corações/Três Corações, por exemplo. Foi buscado um *array* com as cidades no repositório *GitHub*: <https://gist.github.com/letanure/3012978>.

Ainda nos anúncios do tipo Profissional do Seminovos o campo cidade contém todo o endereço da loja, sendo necessário extrair a cidade deste campo. Nessa etapa também foram removidas as cidades não pertencentes a Minas Gerais, alguns anúncios do Seminovos havia automóveis de outros estados. Foram criados dois arquivos para visualização manual das cidades `mg_cities\\city_not_olx.txt` e `mg_cities\\city_not_seminovos.txt`, dessa forma foi possível fazer a padronização de cidades que estavam com grafias erradas, ou mesmo se não era uma cidade pertencente a Minas Gerais. Esses métodos para geração destes arquivos são os

`printCitiesNotInSeminovos()` `printCitiesNotInOlx()` estão no arquivo `mg_cities.cities.py`, sendo essenciais para detecção de padronização das cidades da Figura 23 abaixo.

```
def process_city_olx(df_olx, index, row, column, cities):

    city = df_olx['city']
    city = row['city']
    city = unicode.decode(city).upper()
    city = city.split(',')[0].strip() if ',' in city else city.strip()

    city = 'AMPARO DO SERRA' if city == 'AMPARO DA SERRA' else city
    city = 'BRASOPOLIS' if city == 'BRAZOPOLIS' else city
    city = 'PASSA VINTE' if city == 'PASSA 20' else city
    city = 'SAO JOAO DEL REI' if city == 'SAO JOAO DEL REY' else city
    city = 'TRES CORACOES' if city == '3 CORACOES' else city
    city = 'SETE LAGOAS' if city == '7 LAGOAS' else city

    if city not in cities:
        city = ''

    df_olx.at[index, column] = city

def process_city_seminovos(df_seminovos, index, row, column, cities):
    address = row['address']
    type = row['type']

    city = unicode.decode(address).upper().strip() if type == 'Particular' else getCityFromSemiProfessional

    if city not in cities:
        city = ''

    df_seminovos.at[index, column] = city
```

Figura 23: Métodos de padronização das cidades dos anúncios do Olx e Seminovos.

Após extraídos os atributos dos *datasets* de Olx e Seminovos agora eles são unidos em somente um *dataframe*, através do método `join_dfs()`. Essa assim foi a última etapa de extração e padronização de dados dos anúncios da Olx. Todos os métodos dessa etapa 3.3 estão no arquivo `attributes_ads_etl.py` e após esta etapa é gerado o arquivo do passo `step_5_items_extracted`.

3.4. Tratamento de dados faltantes, informações inconsistentes e *outliers*

A primeira etapa desta seção é remover os registros duplicados através de método *removeDuplicate()*. Após isso, é feito o tratamento de *outliers*, que são valores atípicos que estão fora dos parâmetros normais. Isso acontece bastante no *dataset*, onde o usuário que não deseja informar o valor ou quilometragem do veículo muitas vezes insere valores muito pequenos ou quantidade de dígitos muito grande.

O tratamento de *outliers* será feito para os atributos preço e quilometragem. Nesta etapa são removidos preços e quilometragens abaixo de 1000. Muitos usuários como forma de fazer com que o anúncio dele permaneça em busca de quilometragens baixas, insere por exemplo 160 em vez de 160.000. Assim, quem procura por automóveis com até 100mil km tem sua busca prejudicada por esta técnica.

Após removidos os valores mínimos são analisados veículos de mesma marca, modelo e ano, ou quando não é atingido o número mínimo de 4 veículos na amostra são analisados também de anos próximos.

Após obtidos os valores da amostra é utilizada o método de John Tukey (John Tukey, Exploratory Data Analysis, Addison-Wesley, 1977) para análise de *outliers*. Neste método são utilizados quartis 1 e 3, onde se concentram 25 e 75% dos valores da amostrar, e é aplicado um fator onde o que está fora do intervalo é considerado um *outlier*.

Em vez de remover os registros com atributos considerados *outliers*, o valor é calculado com base nos valores médios dentro dos quartis definidos anteriormente. São criadas duas novas colunas para cada atributo *<atributo>_processed* e *<atributo>_outlier*, contendo novo valor calculado e informação de possível *outlier* ou não, Figura 24.

Esse processamento de *outliers* é feito pelos métodos ***outliers()*** e ***outliers_by_key()***, onde o primeiro itera pelos atributos preço e quilometragem, e o segundo busca as amostras e verifica se é *outlier*.

```

if len(values) > 1:
    fator = 1.5
    q3, q1 = np.percentile(values, [75, 25])
    iqr = q3 - q1
    lowpass = q1 - (iqr * fator)
    highpass = q3 + (iqr * fator)

    if valKey < lowpass or valKey > highpass:
        df.at[index, f'{key}_processed'] = round((q3 + q1)/2)
        df.at[index, f'{key}_outlier'] = 'CORRIGIDO'
    else:
        df.at[index, f'{key}_processed'] = valKey
        df.at[index, f'{key}_outlier'] = 'AMOSTRA PEQUENA' if len(values) <= TAMANHO_AMOSTRA else 'NÃO'
else:
    df.at[index, f'{key}_processed'] = valKey
    df.at[index, f'{key}_outlier'] = 'TALVEZ'

```

Figura 24: Trecho do método para verificação de *outliers*.

Após processados os valores outliers a próxima etapa é fazer o tratamento de dados faltantes, tanto para atributos categóricos quanto para quantitativos. Os atributos categóricos tratados são 'portas', 'cambio', 'direcao', 'combustivel', 'motorizacao' e 'cor'; e os atributos quantitativos são 'preço' e 'quilometragem'.

Os métodos principais para tratamento de atributos categóricos são *fill_category_missing_values()* e *fill_cat_values()*. O primeiro é responsável por iterar por cada um dos atributos categóricos; o segundo para cada um dos atributos percorre o dataset para fazer o tratamento.

A técnica utilizada para substituir dados categóricos é buscar todos os valores para um mesmo modelo de carro e ano e a quantidade de vezes que ele aparece. Dentro desse conjunto é escolhido um desses valores de acordo com o peso da quantidade de vezes que ele aparece.

Já para tratar os atributos quantitativos, são utilizados os métodos *fill_value_missing_values()* e *fill_val_values()*. O primeiro é responsável por iterar por cada um dos atributos quantitativos; o segundo para cada um dos atributos percorre o *dataset* para fazer o tratamento de valores faltantes.

Para tratar dados quantitativos faltantes a técnica utilizada foi buscar a média dos valores para o mesmo modelo e ano. Caso nenhum valor seja encontrado para o mesmo ano é buscado nos anos seguintes.

Após tratados os dados categóricos, novamente é chamado o método *process_completo_final_dataframe()* para processar se um automóvel possui ou não os itens que o tornam um carro 'completo'. Após isso o método *delete_missing_cities()* é chamado para deletar anúncios com cidades em branco, esses anúncios não pertenciam ao estado de Minas Gerais,

algum erro na hora de coletar esses dados no portal do Seminovos.

Por fim, o *dataset* final é armazenado no diretório **step_6_etl_done**, através do método *write_csv()*. Foram assim totalizados **89.453** registros. Todos os métodos dessa etapa de tratamento de *outliers*, dados faltantes e inconsistências estão armazenados no arquivo de *scripts values_treatment.py*.

4. Análise e Exploração dos Dados

Para a análise e exploração dos dados, serão utilizados alguns gráficos e indicadores para apresentar dados sobre cidades, preço e atributos categóricos dos anúncios de carros. Para isso foi utilizada a biblioteca *matplotlib* do *Python*, e os algoritmos para geração dos gráficos estão nos arquivos **analise_exploracao\analysis.py** e **analise_exploracao\scripts\charts.py**.

A primeira análise realizada é fazer um top 10 das marcas com mais anúncios. Percebe-se que as quatro primeiras marcas são que nas últimas décadas se destacaram como mais populares no Brasil, conforme Gráfico 1 abaixo.

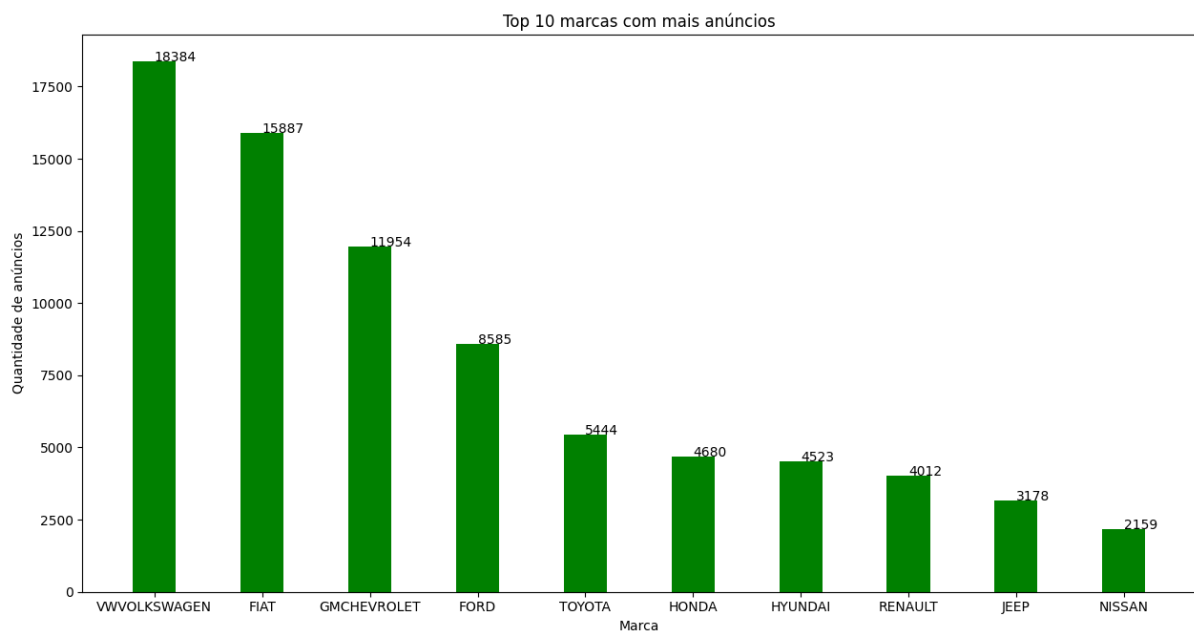


Gráfico 1: Top 10 das marcas com mais anúncios.

A próxima análise é o Gráfico 2 onde é realizada uma análise dos dez modelos de veículos mais anunciados. Nota-se que todos eles estão dentro das cinco marcas mais anunciadas. Ou seja, não existem nenhuma outra marca que possui apenas um modelo que se destaca entrando no top 10.

Além disso, dentro deste pode-se destacar que o veículo VW – Gol possui 30% a mais de anúncios em relação ao segundo colocado. E essa diferença é bem menor em comparação com as próximas posições do ranking, não chegando a 10%.

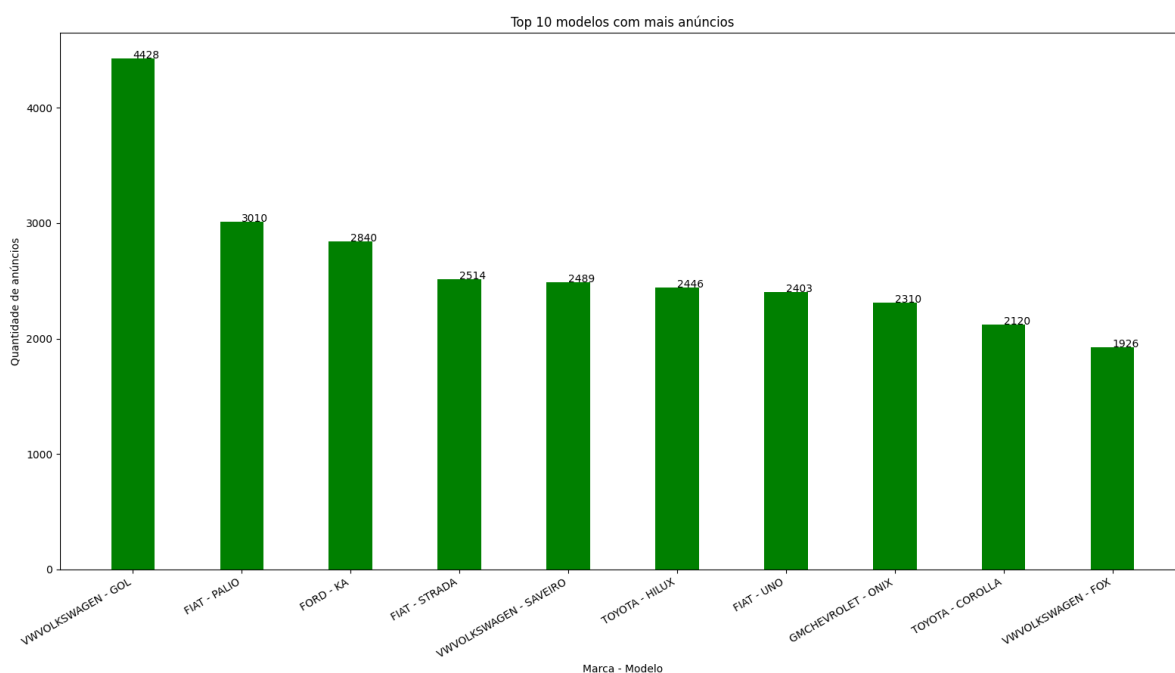


Gráfico 2: Top 10 dos modelos com mais anúncios.

A próxima análise do Gráfico 3 é entender a distribuição de anúncios entre as cidades de Minas Gerais. Nota-se o grande destaque da capital de Minas Geais, Belo Horizonte, que somente ela contempla mais que 1/3 dos anúncios de automóveis.

Através da Figura 25 abaixo, com as 10 cidades mais populosas de Minas Gerais, percebe-se que a relação entre número de anúncios com a quantidade de pessoas é proporcional: as mesmas cidades estão presentes em ambos os *rankings*, alterando um pouco na ordenação.

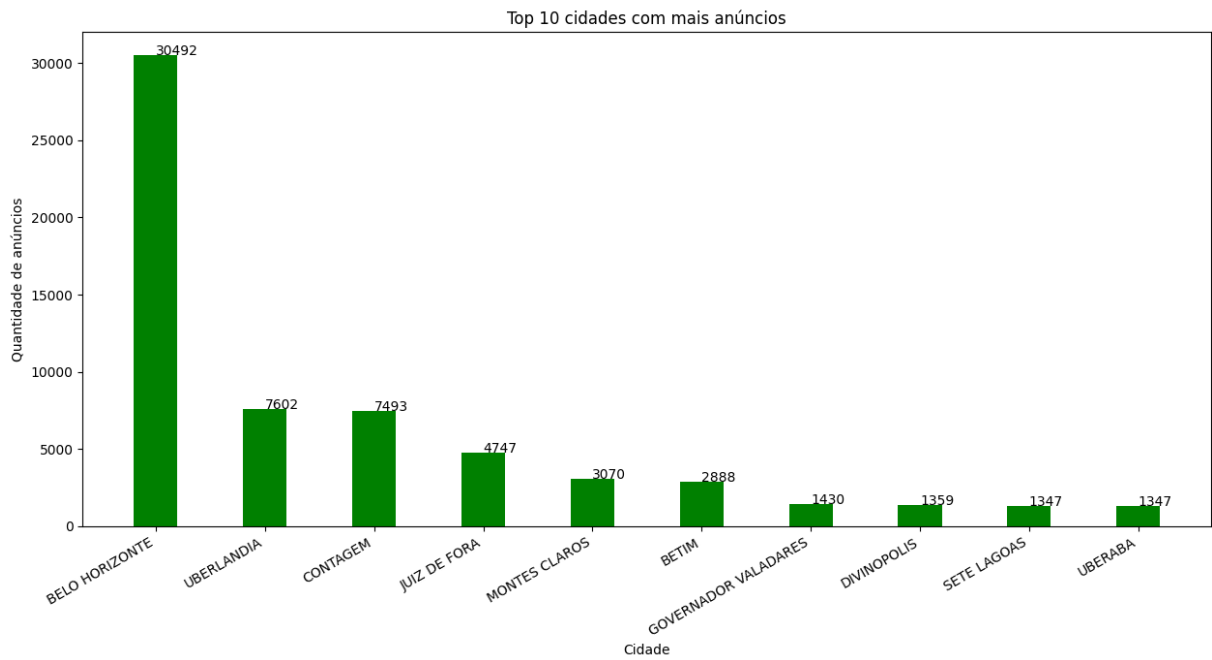


Gráfico 3: Top 10 cidades com mais anúncios.

Populações estimadas das maiores cidade de MG

Cidade	2020
Belo Horizonte	2.521.564
Uberlândia	699.097
Contagem	668.949
Juiz de Fora	573.285
Betim	444.784
Montes Claros	413.487
Ribeirão das Neves	338.197
Uberaba	337.092
Governador Valadares	281.046

Figura 25: Top 10 cidades de Minas Gerais com maiores populações. Fonte: g1.globo.

A próxima análise é entender os dias que os anúncios estão *online*. Foram criados 4 intervalos: menos de 1 dia postado; entre 1 dia e 1 semana; de 1 semana a 1 mês; e mais que 1 mês. Nota-se que quase 70% dos veículos anunciados possuem mais de 1 semana que estão disponíveis para venda.

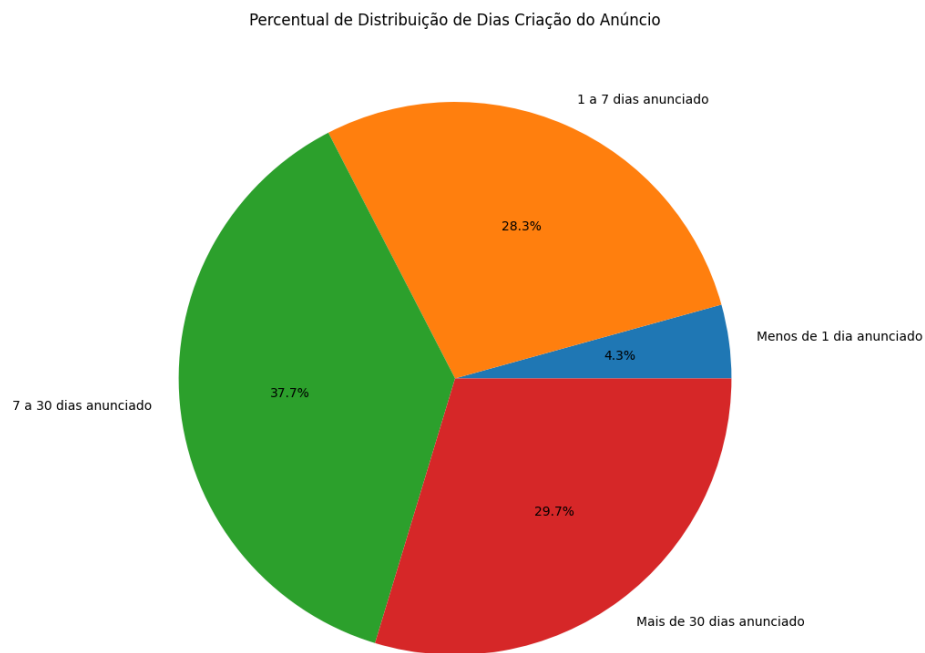


Gráfico 4: Distribuição dias anunciados.

Os próximos Gráficos 5, 6 e 7 mostram um *ranking* com as cidades com maiores médias de preço, a diferença que consideram todas as cidades, cidades com no mínimo 10 anúncios e cidades com no mínimo 50 anúncios, respectivamente.

No primeiro gráfico (Gráfico 5) nota-se que muitas cidades se destacam por muitas vezes possuir apenas alguns carros com preços muito alto, por exemplo a cidade de Serra dos Aimorés possui apenas três anúncios com valores altos, Figura 26.

Quando se filtra por quantidade de anúncios esses casos não aparecem, mas nota-se que a cidade Monte Alegre de Minas permanece nos Gráficos 6 e 7, indicando que é uma cidade que possui automóveis com preços altos.

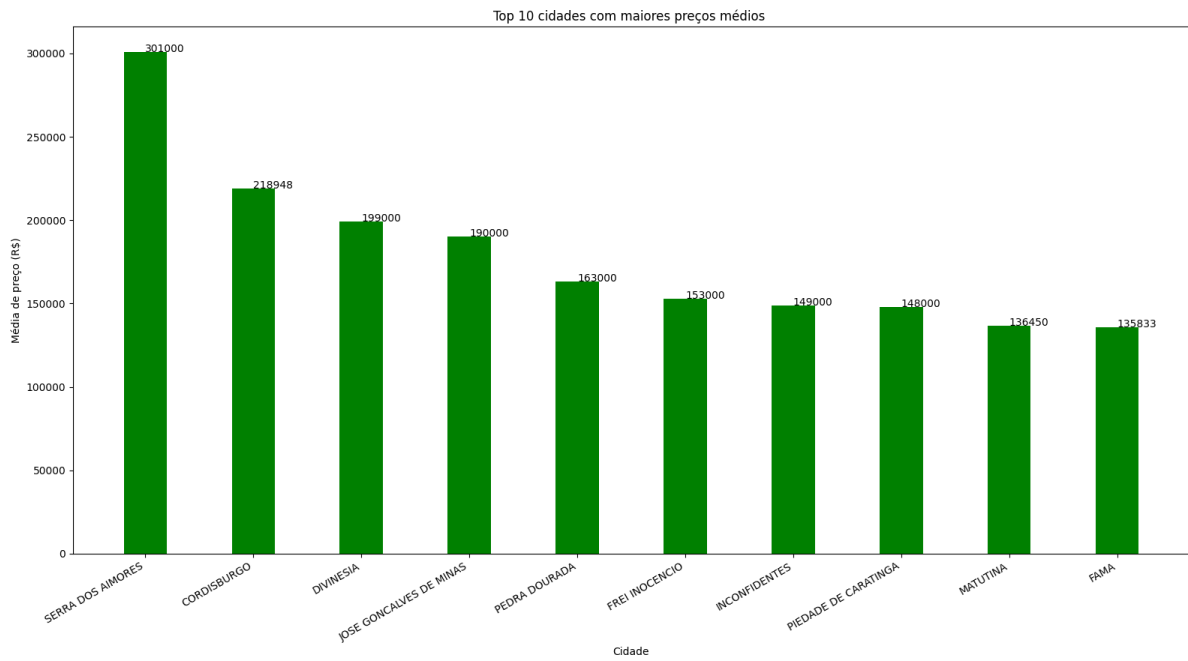


Gráfico 5: Top 10 cidades com maiores preços médios.

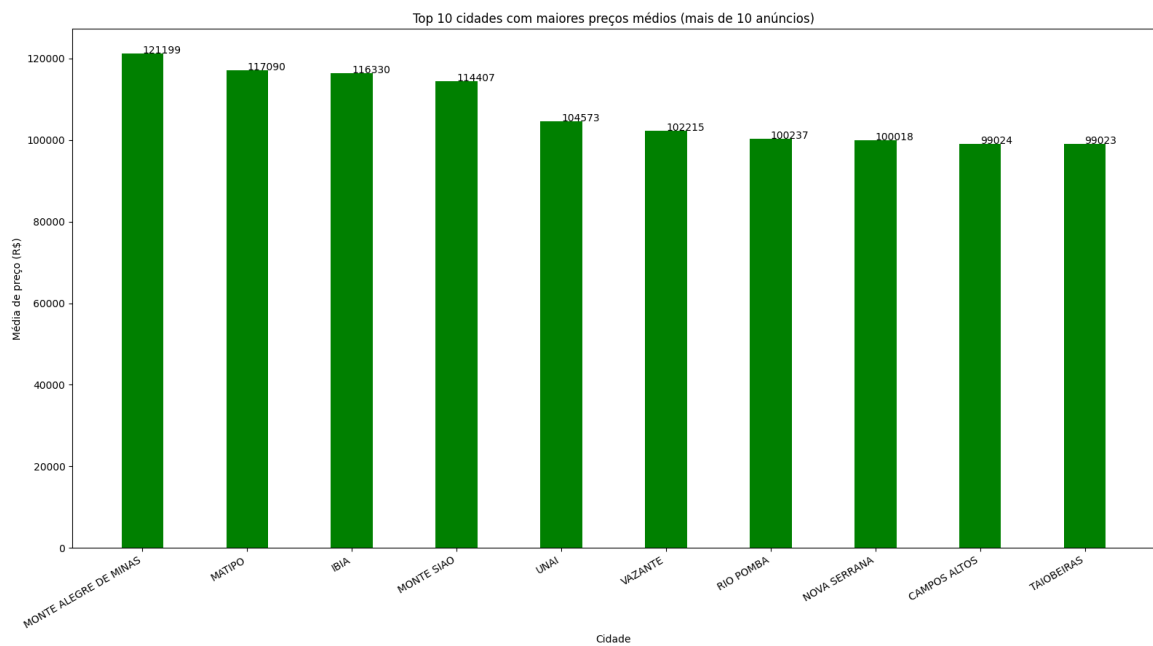


Gráfico 6: Top 10 cidades com maiores preços médios (mais de 10 anúncios).

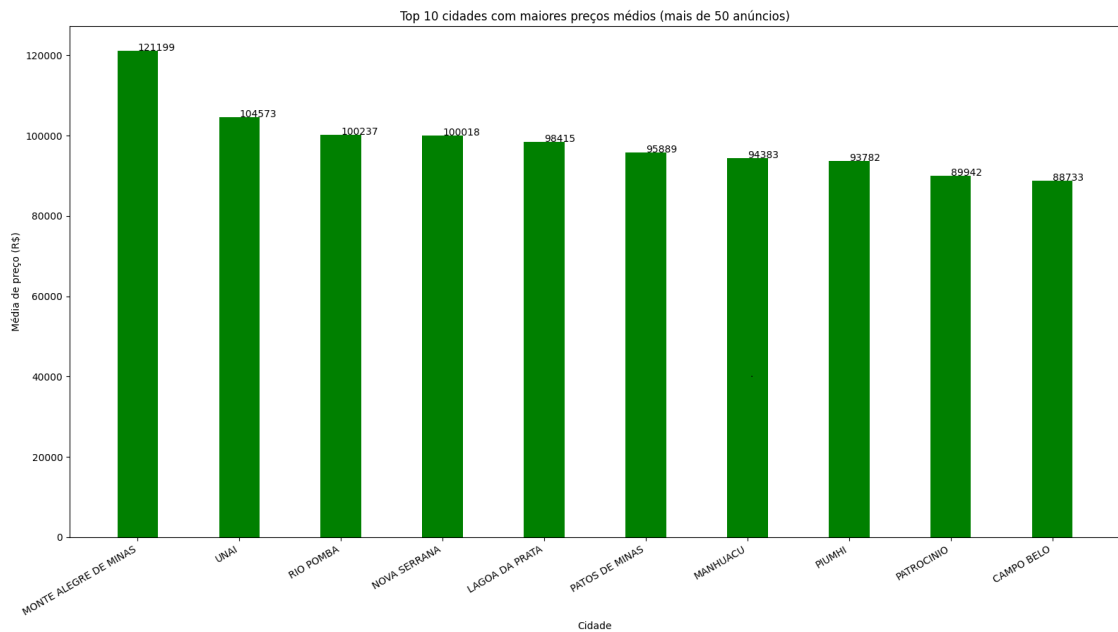


Gráfico 7: Top 10 cidades com maiores preços médios (mais de 50 anúncios).

Uma rápida análise mostra que esses municípios, que são cidades do interior do estado de Minas Gerais, possuem caminhonetes em sua maioria, indicando que é um veículo muito necessário em zonas rurais e longe dos grandes centros.

brand_parsed	model_parsed	city_processed	price_processed
TOYOTA	HILUX	SERRA DOS AIMORES	262000
HYUNDAI	HR	SERRA DOS AIMORES	125000
TOYOTA	HILUX	SERRA DOS AIMORES	340000

Figura 26: Anúncios do município de Serra dos Aimorés.

A próxima análise é identificar a distribuição de preços através de um box plot. Percebe-se que a grande maioria dos anúncios estão situados abaixo dos R\$100 mil reais. Quando o preço passa dos 500mil reais pode-se encontrar valores mais isolados, Gráfico .

Através dos dados da Figura 27, pode-se entender melhor esses números: o maior

anúncio tem valor de R\$3.500.000,00; os quartis de 20, 50 e 75% são R\$34.000, R\$54.900 e R\$87.500, respectivamente, e a média de preços é R\$71.488,73.

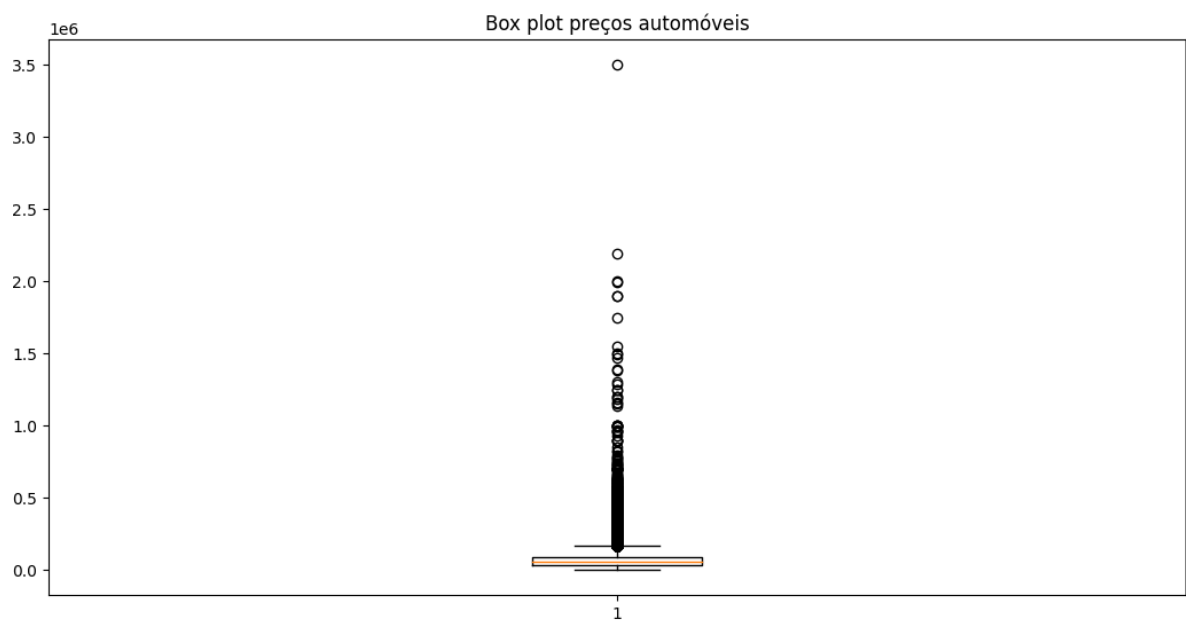


Gráfico 8: Box plot preço dos automóveis.

```
count      89375.000000
mean       71488.733885
std        65705.559933
min         1111.000000
25%        34000.000000
50%        54900.000000
75%        87500.000000
max       3500000.000000
```

Figura 27: Distribuição dos preços dos automóveis.

Analisando ainda os preços dos automóveis, o próximo gráfico mostra os dez carros com maiores valores anunciados. Nota-se que praticamente todos eles acima de R\$1.500.000.

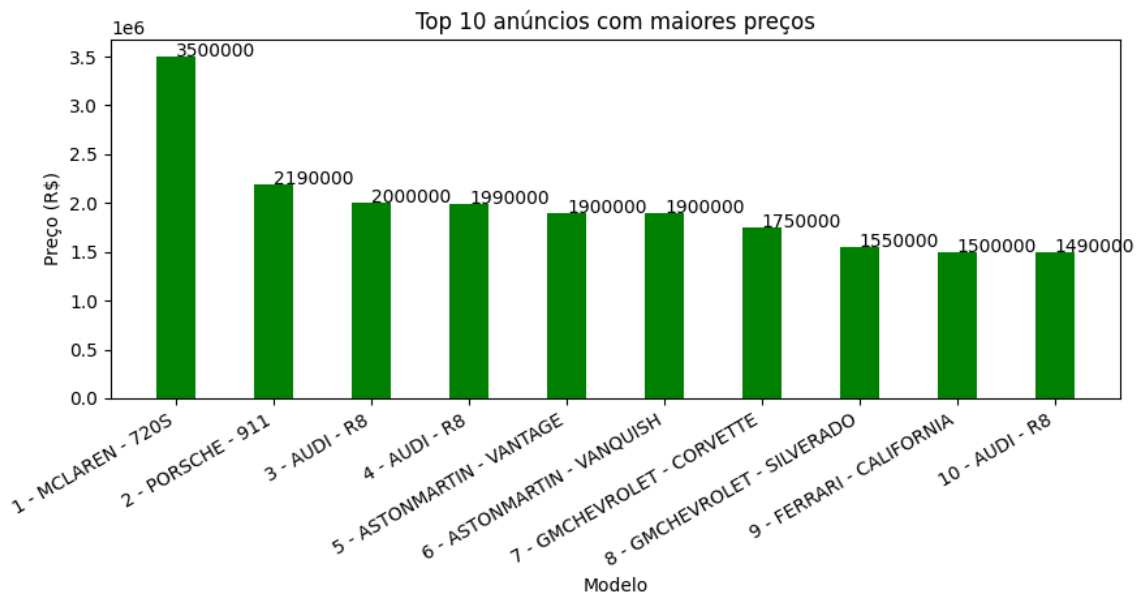


Gráfico 9: Top 10 anúncios com maiores valores.

O Gráfico 10 abaixo mostra o *ranking* dos 10 modelos de carros elétricos mais anunciados. Pode-se notar que esse tipo de motorização ainda não é bem difundida nos anúncios de automóveis de Minas Gerais. Isso indica que talvez por ser uma tecnologia teoricamente recente, ainda não possuem muitos veículos seminovos à venda.

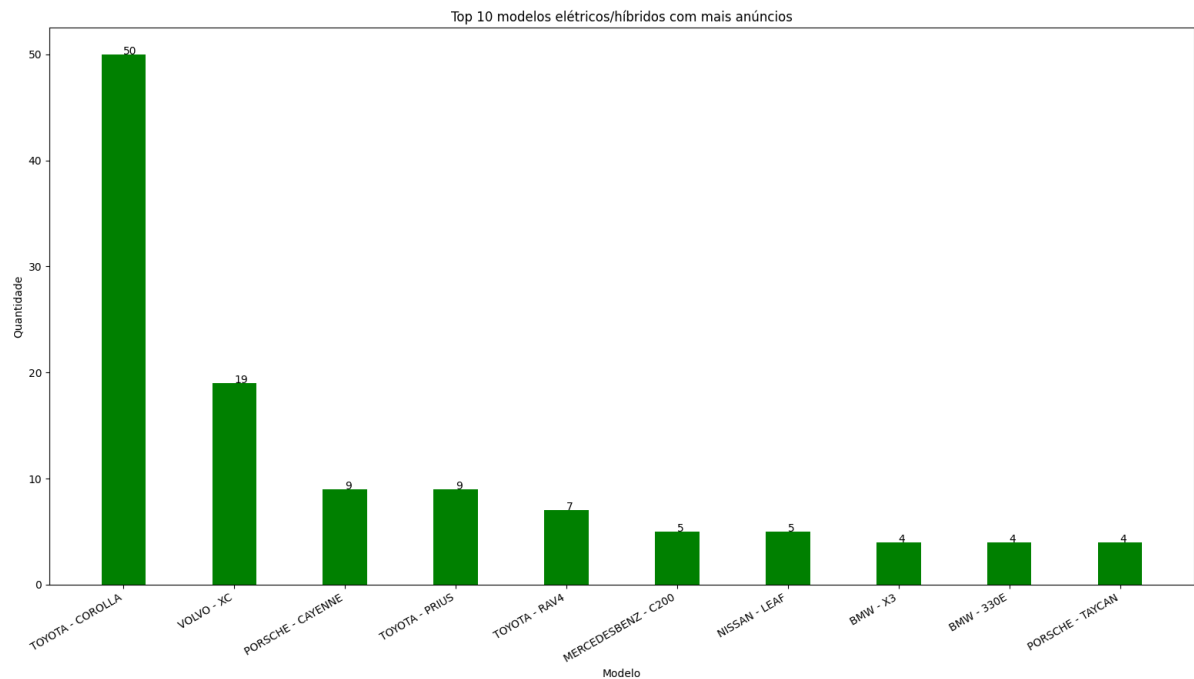


Gráfico 10: Top 10 elétricos/híbridos com mais anúncios.

Pode ser também que a razão dessa baixa quantidade de anúncios de carros elétricos pode ser devido a análise que pode ser feita no Gráfico 11 – da média de preço desses mesmos modelos - boa parte deles possuem média de preço acima de 200 mil reais ou bem mais. Apenas o Toyota Prius, um modelo com quase 10 anos desde o seu lançamento, possui preço médio na casa dos 90 mil reais.

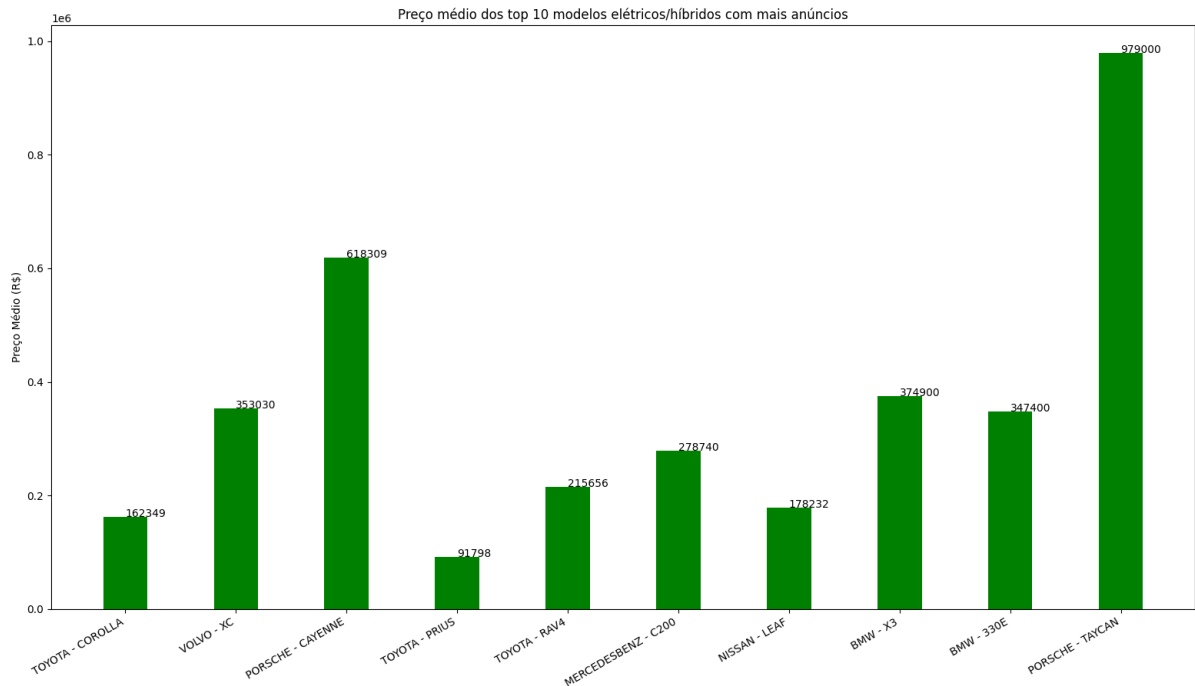


Gráfico 11: Preço médio do top 10 elétricos/híbridos com mais anúncios.

O Gráfico 12 mostra um dado interessante sobre a evolução dos tipos de câmbios ao longo dos anos. Para anúncios dos anos 2000, praticamente 95% dos anúncios são de carros com câmbio manual. Percebe-se que ao passar dos anos a quantidade de carros com câmbio automático (automático e automatizados) vai crescendo, até a partir de 2014 já se tornar maioria em relação ao câmbio manual.

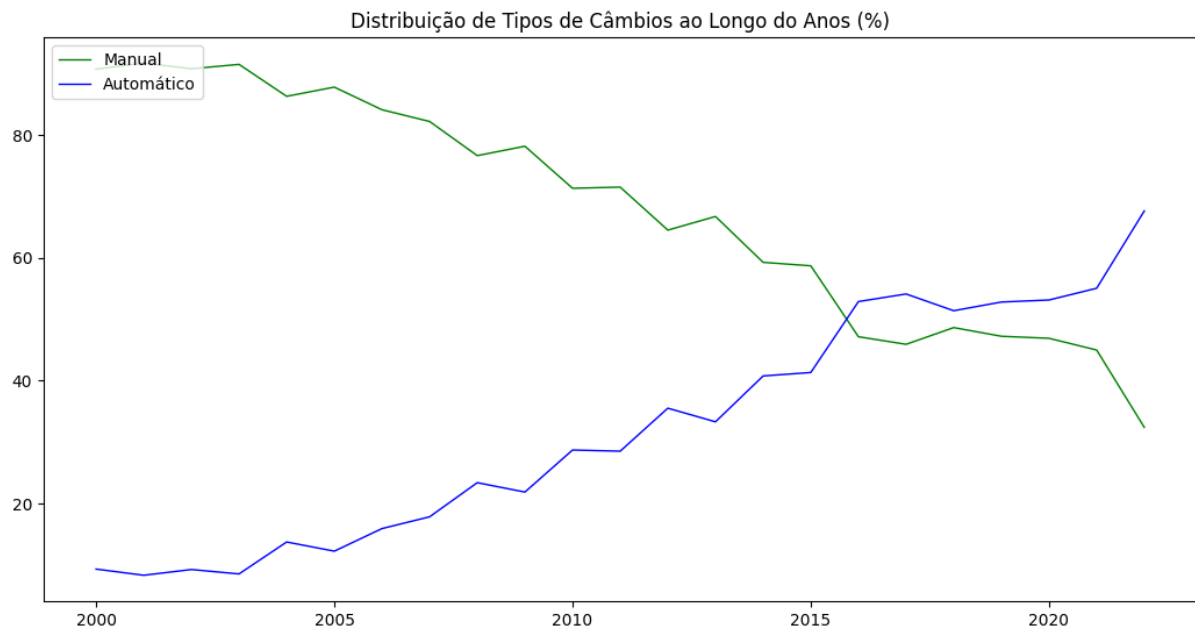


Gráfico 12: Percentual de tipos de câmbios ao longo dos anos.

Por fim, a última análise feita nesta etapa é identificar o percentual de anúncios profissionais e particulares, no Gráfico 13. Percebe-se que este número é bem próximo, com pequena porcentagem a mais de anúncios dos tipos particulares.

Percentual de distribuição dos tipos anúncios

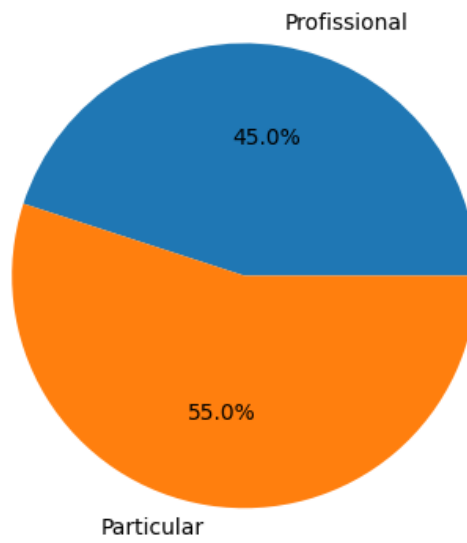


Gráfico 13: Percentual distribuição tipo de anúncios.

5. Criação de Modelos de Machine Learning

5.1. Modelo de Machine Learning

Dentre os modelos de machine learning, o aprendizado supervisionado pode ser útil para identificar padrões nos dados e prever inconsistências ou fraudes com base em rótulos conhecidos. Se o *dataset* fosse rotulado, onde sabe-se quais anúncios são fraudulentos e quais são autênticos; quais anúncios tem alguma inconsistência ou quais possui informações precisas, seria possível treinar um modelo de aprendizado supervisionado para classificar novos anúncios como fraudulentos/precisos. No entanto, é importante ter um conjunto de dados rotulados confiáveis e representativos para o treinamento.

O aprendizado não-supervisionado é útil para identificar padrões e inconsistências nos dados sem a necessidade de rótulos conhecidos. Dessa forma, algoritmos não supervisionados, pode ajudar a identificar grupos de anúncios que se comportam de maneira suspeita ou atípica em relação aos demais, visualizar padrões ou separações entre os anúncios.

Quando se analisa o aspecto de anúncios fraudulentos não se tem esse rótulo nos dados para saber se é fraude ou não. E sobre dados inconsistentes, seria necessário ter um conhecimento bem grande sobre cada modelo de automóvel, e ainda sim pode haver particularidades que não são inerentes a característica do automóvel, por exemplo como quilometragem. Dessa forma, não se tem nenhuma informação sobre rótulos, sendo assim o melhor modelo de machine learning a ser utilizado é o **aprendizado não-supervisionado**.

Os algoritmos desse modelo implementados nesse trabalho utilizam a biblioteca *sklearn* do Python para esses aprendizados. Os scripts com as chamadas dos métodos dos algoritmos estão localizados em **machine_learning\ads_machine_learning.py**.

5.2. K-means

O K-means é um algoritmo de agrupamento. É um dos algoritmos de machine learning não supervisionados mais simples e frequentemente usado. Sua ideia é basicamente *clusterizar* os dados semelhantes e descobrir padrões subjacentes.

Apesar de ser bastante utilizado para agrupar dados e descobrir padrões em dados não rotulados, o K-means também tem sido bastante utilizado para buscar inconsistências nos dados. E esse é um dos principais objetivos do trabalho, encontrar anúncios que podem ser considerados fraudulentos ou contém informações inconsistentes.

Dessa forma, dados os *clusters*, registros que possuem a maior distância do centroide do *cluster* são possíveis registros inconsistentes, uma vez que são os que mais se diferem do grupo a que pertencem.

O primeiro passo do K-means é encontrar a melhor quantidade de *clusters*. Para isso é usada. Um modelo de agrupamento não é relevante se não é encontrado o número correto de agrupamentos a serem considerados. Existem várias técnicas para encontrar o número ideal de clusters. Para este algoritmo utilizou-se o método Elbow, que é um método heurístico e um dos mais usados para encontrar o número ótimo de *clusters*.

A primeira função auxiliar *find_best_clusters()* cria para cada valor de K o modelo K-means correspondente e salva sua inércia junto com o valor atual de K. A segunda função *generate_elbow_plot()* usa essas inércias e valores de K para gerar o gráfico de Elbow final, conforme Figura 28.

```
def find_best_clusters(df, maximum_K):
    clusters_centers = []
    k_values = []

    for k in range(1, maximum_K):
        kmeans_model = KMeans(n_clusters = k)
        kmeans_model.fit(df)

        clusters_centers.append(kmeans_model.inertia_)
        k_values.append(k)

    return clusters_centers, k_values

def generate_elbow_plot(clusters_centers, k_values):
    plt.subplots(figsize = (12, 6))
    plt.plot(k_values, clusters_centers, 'o-', color = 'orange')
    plt.xlabel("Number of Clusters (K)")
    plt.ylabel("Cluster Inertia")
    plt.title("Elbow Plot of KMeans")
    plt.show()
```

Figura 28: Métodos para encontrar a quantidade ideal de *clusters*.

Após gerado o Gráfico 14 Elbow de *clusters* nota-se que quando o número é superior a 6, a inércia não se altera muito. Dessa forma, o número de *clusters* a ser utilizado para o K-means será 6.

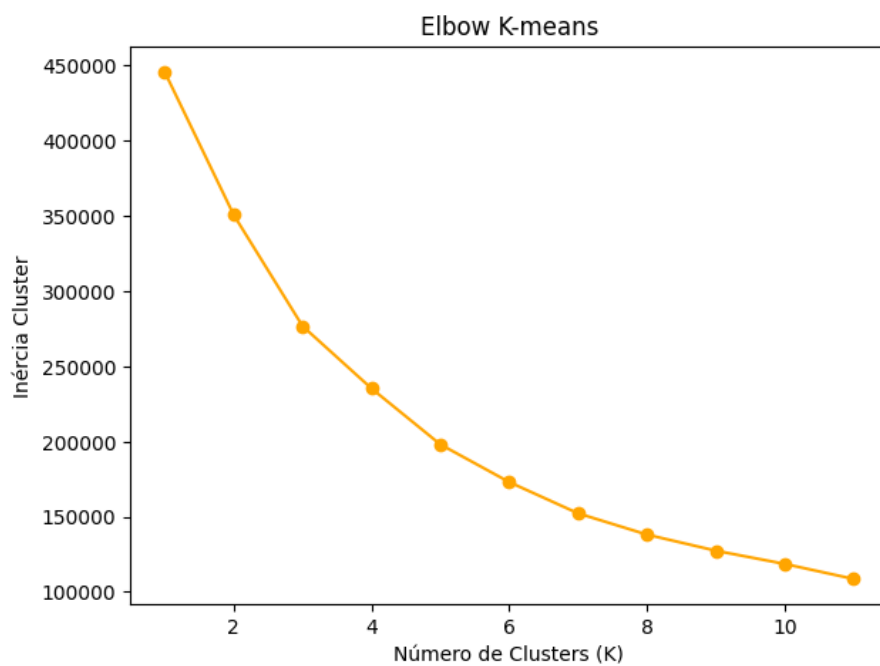


Gráfico 14: Elbow quantidade de *clusters*.

Explorando agora o código para geração do K-means algumas tratativas foram realizadas antes de iniciar os cálculos dos *clusters*. Primeiramente, foram analisados os campos relevantes para o estudo. Para isso foram considerados preço, quilometragem, dias de anúncio, ano e uma nova coluna foi criada – *extras_score*. Nessa coluna é dado uma pontuação de acordo com a quantidade de acessórios ou itens de série que o automóvel do anúncio contém, para poder contemplar os dados categóricos do anúncio. Isso é feito através do método *generate_extras_score()*.

Após isso são removidos os registros que algum desses atributos não contém valores (*remove_missing_values*), filtradas somente as colunas que serão usadas no K-means (*keep_relevant_columns*) e por fim chamados todos os métodos do K-means do *sklearn*, conforme Figura 29.

```
relevant_cols = ['price_processed', 'quilometragem_processed', 'ano', 'posted_days',

df_all_cols = get_dataframe()
df_all_cols = generate_extras_score(df_all_cols)
df_all_cols = remove_missing_values(df_all_cols, relevant_cols)
df = keep_relevant_columns(df_all_cols, relevant_cols)

scaler = StandardScaler()
scaler.fit(df)
scaled_data = scaler.transform(df)

clusters_centers, k_values = find_best_clusters(scaled_data, 12)
generate_elbow_plot(clusters_centers, k_values)

CLUSTERS = 6

kmeans_model = KMeans(n_clusters = CLUSTERS)

kmeans_model.fit(scaled_data)
distances = kmeans_model.transform(scaled_data)

df_all_cols["clusters"] = kmeans_model.labels_

for i in range(CLUSTERS):
    df_all_cols[f'Distance_to_Cluster_{i}'] = distances[:, i]
```

Figura 29: Implementação do K-means com 6 *clusters*.

Após executado o K-means cada um dos 6 clusters ficaram com a quantidade abaixo de registros:

Cluster	Quantidade de Registros
0	17.922
1	15.640
2	6.199
3	3
4	8.786
5	40.595

Tabela 3: Quantidade de registros por *cluster*.

O fato interessante disso é o *cluster* 3 conter apenas três registros. E analisando esses três registros foi possível concluir a razão disso: todos eles possuem quilometragem muito acima do normal. O algoritmo de detecção de *outliers* da seção 3.4 até sinalizou que dois desses registros são potenciais *outliers*, mas o que contém a *quilometragem_outlier* como corrigido ainda sim é um erro, conforme Figura 30.

code	brand_parsed	model_parsed	ano	quilometragem_processed	quilometragem_outlier	cluster	
cód. 1087922026	GM	CHEVROLET	VERANEIO	1975	1111111111	AMOSTRA PEQUENA	3
cód. 1093931213	FORD	LANDAU	2022	1000000000	TALVEZ		3
cód. 1106853160	FORD	F350	1999	694686069	CORRIGIDO		3

Figura 30: Anúncios pertencentes ao *cluster* 3.

5.3. One-class Support Vector Machine

O One-class Support Vector Machine é um algoritmo de aprendizado não supervisionado que é uma variação do tradicional SVM. Ao contrário do SVM supervisionado, o One-class SVM não possui rótulos de destino para o processo de treinamento do modelo. Em vez disso, ele aprende o limite para os pontos de dados normais e identifica os dados fora da fronteira como anomalias.

Para o objetivo do projeto, o One-Class SVM pode identificar outros tipos de anomalias ou outliers nos dados. Isso pode incluir anúncios de carros que não seguem os padrões típicos do conjunto de dados, como preços extremamente altos, quilometragem anormalmente baixa ou características de veículos incomuns. O One-Class SVM é capaz de identificar essas instâncias que estão distantes da região que contém a maioria dos exemplos normais, permitindo a detecção de casos incomuns ou de anomalias.

A primeira parte do algoritmo é trabalhar o *dataset* para contemplar apenas as colunas de estudo. Neste caso os atributos de estudo serão quilometragem, ano e preço. Após feito isso o *dataset* é dividido entre treino e teste, com proporção de 80% e 20%, para posteriormente, ser verificado a performance do modelo através do método *classification_report*.

Dividido o *dataset* o SVM é instanciado com parâmetros $\text{nu} = 0.1$, que indica o quanto vai ser tolerante a inconsistências; e $\text{kernel} = \text{rbf}$, radial basis function, que é a função de utilizada para projetar os dados. Com isso as funções *fit* e *predict* são usadas para treinar e validar o modelo, conforme Figura 30.

```

relevant_cols = ['price_processed', 'ano', 'quilometragem_processed']

df_all_cols = get_dataframe()
df_all_cols = remove_missing_values(df_all_cols, relevant_cols)

X = df_all_cols[relevant_cols]

scaler = StandardScaler()
dados = scaler.fit_transform(X)

dados_treino, dados_teste = train_test_split(dados, test_size=0.2, random_state=42)

modelo = OneClassSVM(kernel='rbf', gamma='scale', nu=0.1)
modelo.fit(dados_treino)

anomalias_teste = modelo.predict(dados_teste)

anomalias_teste[anomalias_teste == 1] = 0 # Convertendo instâncias normais para 0
anomalias_teste[anomalias_teste == -1] = 1 # Convertendo anomalias para 1

y_test = [0] * len(dados_teste)
y_test.extend([1] * len(dados_teste[anomalias_teste == -1]))

report = classification_report(y_test, anomalias_teste)
print(report)

```

Figura 30: Implementação One-Class SVM.

Após treinado e testado o modelo o algoritmo apresentou a performance exibido na Figura 31 abaixo.

	precision	recall	f1-score	support
0	0.99	0.99	0.99	17668
1	0.06	0.06	0.06	161
accuracy			0.98	17829
macro avg	0.53	0.53	0.53	17829
weighted avg	0.98	0.98	0.98	17829

Figura 31: Performance do modelo do One-class SVM.

Uma parte interessante do One-class SVM é poder visualizar esses dados de anomalias e normais. É fácil a visualização de quando os dados que passam das fronteiras do conjunto considerado normal são assinalados inconsistências.

Durante o estudo dessa implementação notou-se que o SVM pode comportar melhor quando se analisa casos para um só modelo de automóvel em específico. Pois quando se analisa todo o *dataset*, pode acontecer de um veículo com o valor muito alto e quilometragem extremamente baixa pode ser considerado uma anomalia – e essas características são comuns em carros superesportivos.

Dessa forma foram gerados dois gráficos, o primeiro, o Gráfico 15, contém apenas os anúncios do modelo Volkswagen Gol, o modelo mais anunciado de acordo com o Gráfico 2 da seção 4. Já o Gráfico 16 contém todos os anúncios do *dataset*.

Detecção de Anomalias em Anúncios do modelo VW - Gol

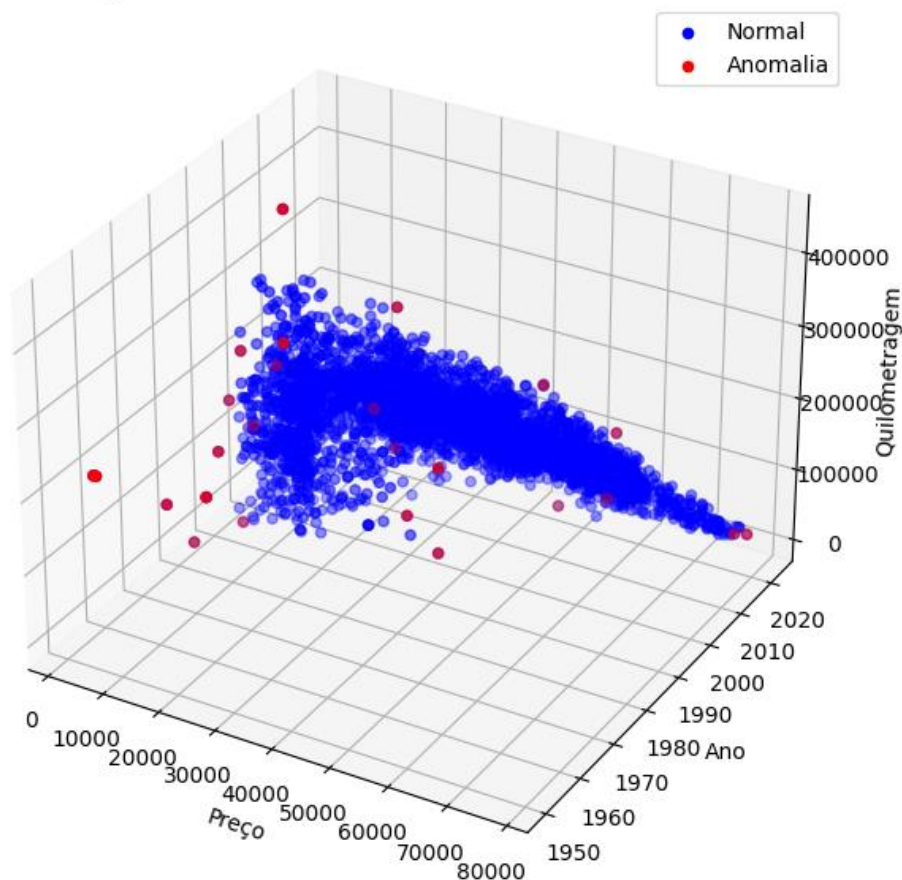


Gráfico 15: One-class SVM para detecção de anomalias do modelo VW – Gol.

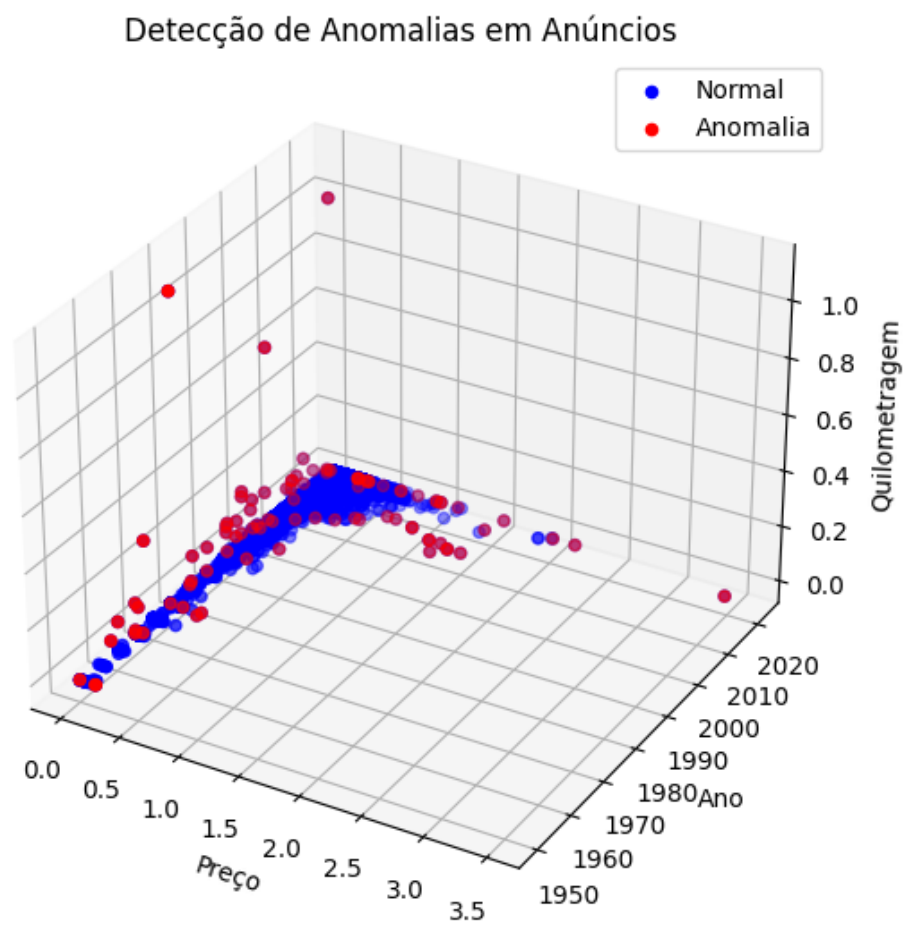


Gráfico 15: One-class SVM para detecção de anomalias em todo *dataset*.

5.4. Principal Component Analysis - PCA

O Principal Component Analysis (PCA) é uma técnica de redução de dimensionalidade que pode ser utilizada para identificar padrões em conjuntos de dados de alta dimensionalidade. Ele busca transformar as variáveis originais em um conjunto de novas variáveis, que são chamados os componentes principais. A principal do PCA é capturar a maior parte da variabilidade dos dados em um número menor de componentes principais, tentando preservar as informações contidas nos dados originais.

Sobre os anúncios de carros, o PCA pode ajudar a encontrar anomalias nos dados. Ao aplicar o PCA nos atributos dos anúncios de carro, é possível identificar se algum dos componentes principais captura um comportamento incomum ou fora do padrão esperado, analisando as dispersões desses componentes gerados.

Como a ideia do PCA é a redução de grandes dimensionalidades, o algoritmo implementado para os anúncios de carro foi analisar a dispersão de dados dos atributos quilometragem, ano, preço, dias de anúncio e motorização e reduzi-los em dois principais componentes apenas, analisando a dispersão.

Outro experimento a ser realizado com PCA é usar a combinação dele com o Support Vector Classification, para tentar fazer a predição de possíveis anúncios de leilão, bem como a cross validation e eficácia dessa modelagem.

A primeira parte do algoritmo é buscar os atributos do *dataframe*, normalizá-los e instanciar o PCA para geração de 2 componentes principais. Após isso os componentes são criados a partir do método *fit_transform*. Por fim é gerado o gráfico de dispersão.

```

relevant_cols = ['price_processed', 'ano', 'quilometragem_processed', 'posted_days', 'motorizacao']

df_all_cols = get_dataframe()
df_all_cols = remove_missing_values(df_all_cols, relevant_cols)

df = df_all_cols[relevant_cols]

scaler = StandardScaler()
normal_data = scaler.fit_transform(df)

pca = PCA(n_components=2)
principal_components = pca.fit_transform(normal_data)

print("Variância componentes principais:")
print(pca.explained_variance_ratio_)

plt.scatter(principal_components[:, 0], principal_components[:, 1])
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('Dispersão dos Componentes Principais')
plt.show()

```

Figura 32: Implementação da dispersão do PCA.

Analisando os dois componentes principais gerados, cada um teve a proporção de variância de 31% e 23%, indicando a porcentagem que cada um representa na variância total dos dados.

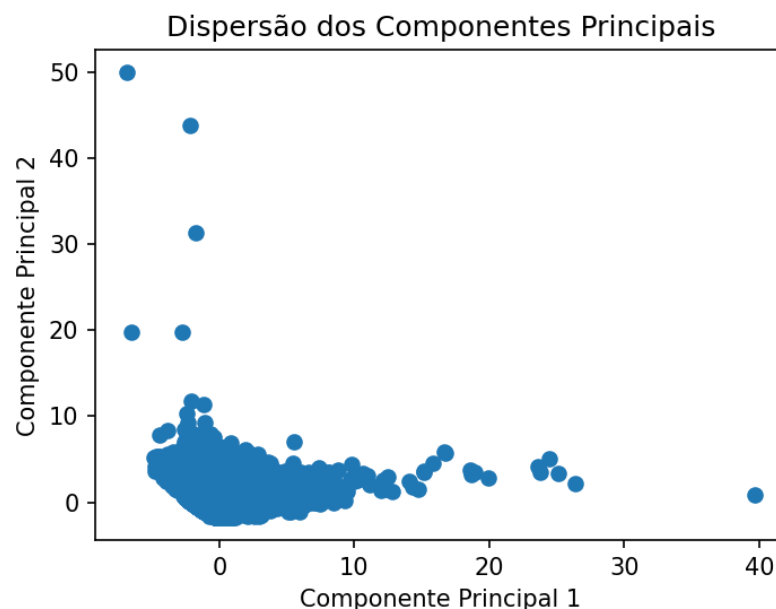


Gráfico 16: Dispersão dos componentes principais.

O gráfico 16 mostra as dispersões dos dados após gerados os componentes principais. Pontos fora da concentração podem indicar casos de anomalias no conjunto de dados.

A outra análise anteriormente citada é tentar estabelecer uma relação entre os componentes principais gerados com o rótulo de leilão contidos nos anúncios. Para isso é carregado o *dataframe*, da mesma forma do algoritmo anterior, porém agora é criada uma variável *y* que contém os dados de leilão. O *dataset* é dividido entre treino e teste na proporção de 80% e 20%, é instanciado o PCA, treinado o modelo pelo método *fit* e testado pelo *predict* do SVC. Após isso são geradas as informações de performance do modelo contidas na Figura 34.

```
X = df.values
y = np.array(df_all_cols['leilao'])

scaler = StandardScaler()
normal_X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(normal_X, y, test_size=0.2, random_state=42)

y_train=y_train.astype('int')
y_test=y_test.astype('int')

pca = PCA(n_components=2)
principal_components = pca.fit_transform(X_train)

modelo_svm = SVC()
modelo_svm.fit(principal_components, y_train)

test_pc = pca.transform(X_test)

y_pred = modelo_svm.predict(test_pc)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recallscore = recall_score(y_test, y_pred)
f1score = f1_score(y_test, y_pred)
```

Figura 33: Implementação de predição do PCA em combinação com SVC.

```
Acurácia do modelo: 97.87%
Precisão do modelo: 0.00%
Recall score do modelo: 0.00%
F1-score do modelo: 0.00%
```

Figura 34: Performance do modelo.

6. Apresentação e Interpretação dos Resultados

Na seção anterior foram apresentados os algoritmos utilizados para possível de detecção de anomalias dos anúncios de carros dos portais Olx e Seminovos. O objetivo dessa seção agora é interpretar os resultados obtidos, se os algoritmos são performáticos o suficiente para detecção destes problemas propostos.

Para a análise dos resultados obtidos do K-means a ideia é analisar os 3 resultados mais próximo e os 3 mais longe do centroide do cluster.

Cluster 0:

Cluster: 0							
Próximos							
Code	Marca	Modelo	Preço	Quilometragem	Ano	Dias Anuncio	Extras
cód. 1100514275	VWVOLKSWAGEN	SAVEIRO	39900	100000	2012	18	4
cód. 1098820916	GMHEVROLET	MONTANA	39000	72000	2012	19	4
cód. 1098389465	HONDA	CITY	37900	159000	2012	20	4
Distantes							
Code	Marca	Modelo	Preço	Quilometragem	Ano	Dias Anuncio	Extras
cód. 1090318549	FIAT	DUCATO	50000	99999999	2009	42	2
cód. 1090338041	FIAT	DUCATO	50000	99999999	2009	42	1
cód. 1071757616	VWVOLKSWAGEN	PARATI	34000	139011665	2013	6	6

Uma rápida análise no *cluster* 0 mostra que os mais distantes possuem uma quilometragem bem fora dos padrões normais.

Cluster 1:

Cluster: 1							
Próximos							
Code	Marca	Modelo	Preço	Quilometragem	Ano	Dias Anuncio	Extras
cód. 1084242879	VWVOLKSWAGEN	SAVEIRO	65000	140000	2014	55	8
cód. 1084215917	VWVOLKSWAGEN	SAVEIRO	62000	114000	2014	57	8
cód. 1084050702	VWVOLKSWAGEN	SAVEIRO	61900	101000	2014	57	8
Distantes							
Code	Marca	Modelo	Preço	Quilometragem	Ano	Dias Anuncio	Extras
cód. 977954758	BMW	530i	253890	48000	2018	302	11
cód. 992940413	MERCEDES BENZ	S500L	439900	35000	2016	270	3
2556673	HONDA	CIVIC	21800	100000	1999	1154	6

O *cluster* 1 apresenta uma particularidade a ser estudada: os 3 anúncios mais próximos do centroide são do mesmo modelo com características bem próximas.

Cluster 2:

Cluster: 2							
Próximos							
Code	Marca	Modelo	Preço	Quilometragem	Ano	Dias Anuncio	Extras
cód. 1096004361	VWVWOLKSWAGEN	GOL	23000	93857	1993	26	3
cód. 1098077694	VWVWOLKSWAGEN	GOL	15000	80000	1993	24	3
3455623	FIAT	ELBA	14900	210000	1993	26	2
Distantes							
Code	Marca	Modelo	Preço	Quilometragem	Ano	Dias Anuncio	Extras
cód. 1095050189	FORD	PAMPA	22000	181622986	1995	28	1
cód. 1087047321	BMW	318	25000	200000000	1995	47	4
cód. 1101374984	FORD	F350	103000	347393034	1965	15	0

O *cluster 2* também mostra que os mais distantes possuem uma quilometragem bem fora dos padrões normais.

Cluster 3: somente possui 3 anúncios, não sendo necessária a análise, conforme seção interior.

Cluster 4:

Cluster: 4							
Próximos							
Code	Marca	Modelo	Preço	Quilometragem	Ano	Dias Anuncio	Extras
cód. 1084597321	VWVWOLKSWAGEN	TIGUAN	199000	69000	2019	18	9
cód. 1025105567	BMW	218i	209900	17900	2020	16	9
cód. 969526449	VWVWOLKSWAGEN	AMAROK	212990	122586	2019	12	9
Distantes							
Code	Marca	Modelo	Preço	Quilometragem	Ano	Dias Anuncio	Extras
cód. 1059927013	AUDI	R8	1990000	2000	2021	113	0
cód. 1086777506	PORSCHE	911	2190000	4400	2021	1	7
cód. 1046781783	MCLAREN	720S	3500000	5000	2019	3	0

O *cluster 4* parece agrupar carros de luxo e mais caros, porém pode-se notar um ponto importante, dois anúncios mais distantes, que são carros de luxo/esportivos, possuem zero extras, aparentando ser uma inconsistência.

Cluster 5:

Cluster: 5							
Próximos							
Code	Marca	Modelo	Preço	Quilometragem	Ano	Dias Anuncio	Extras
cód. 1079531799	FORD	ECOSPORT	64900	91183	2016	10	9
cód. 1103887002	HYUNDAI	HB20X	66990	49970	2016	9	9
cód. 1103595774	GMCHEVROLET	SPIN	62000	104981	2016	9	9
Distantes							
Code	Marca	Modelo	Preço	Quilometragem	Ano	Dias Anuncio	Extras
cód. 1092159880	FORD	F250	145000	53000	1999	2	7
cód. 1102036202	RENAULT	MEGANE	10500	15111111	2021	14	7
cód. 1095579294	VOLVO	C30	32000	18890750	2007	30	9

O *cluster 5* também mostra que os mais distantes possuem uma quilometragem bem fora dos padrões normais.

A Figura 35 abaixo é uma análise de alguns itens considerados como anomalias pelo algoritmo de One-class SVM sobre todo o *dataset*. Percebe-se que boa parte das anomalias estão relacionadas a quilometragem alta; e supercarros (Audi R8, McLaren, Lamborguini, Porsche Taycan) de preço bem alto e quilometragem baixa – esse último grupo não deve ser considerado anomalia.

<u>Marca</u>	<u>Modelo</u>	<u>Preço</u>	<u>Ano</u>	<u>Quilometragem</u>
GMCHEVROLET	OPALA	R\$ 45,000.00	1980	125073125
RENAULT	SCENIC	R\$ 5,000.00	2000	22660375
FERRARI	CALIFORNIA	R\$ 1,500,000.00	2010	21000
MERCEDESSENZ	SLS63	R\$ 1,300,000.00	2011	26000
VWVOLKSWAGEN	PARATI	R\$ 34,000.00	2013	139011665
VWVOLKSWAGEN	PARATI	R\$ 35,000.00	2013	46435245
MCLAREN	720S	R\$ 3,500,000.00	2019	5000
RENAULT	CLIO	R\$ 5,500.00	2021	10000000
GMCHEVROLET	SILVERADO	R\$ 1,550,000.00	2022	1500
AUDI	R8	R\$ 825,000.00	2014	8000
AUDI	Q5	R\$ 360,000.00	2022	9100
AUDI	R8	R\$ 2,000,000.00	2021	2438
GMCHEVROLET	OPALA	R\$ 50,000.00	1979	31325531
FORD	MAVERICK	R\$ 260,000.00	1974	86945
LAMBORGHINI	GALLARDO	R\$ 1,199,000.00	2011	13200
MERCEDESSENZ	SPRINTER	R\$ 26,800.00	1998	34871161
MERCEDESSENZ	C200	R\$ 359,900.00	2022	6575
NISSAN	GTR	R\$ 995,900.00	2013	17000
PORSCHE	TAYCAN	R\$ 999,000.00	2022	2400
PORSCHE	TAYCAN	R\$ 959,000.00	2022	3300

Figura 35: Anomalias One-class SVM.

A Figura 36 já é uma análise sobre todos os anúncios considerados anomalias do modelo VW – Gol, o modelo com mais anúncios no *dataset*. Percebe-se anúncios de carros bem antigos com quilometragem bem baixa; ano 1950 que não existe esse modelo nesse ano; e outros casos tipos dois anúncios do ano 2022 com diferença de mais de 20 mil reais entre eles.

Preço	Ano	Quilometragem
R\$ 5,450.00	1980	80000
R\$ 3,000.00	1980	10000
R\$ 16,800.00	1985	300000
R\$ 5,500.00	1990	2000
R\$ 40,000.00	1992	30000
R\$ 40,000.00	1992	149910
R\$ 2,000.00	1992	160000
R\$ 12,500.00	1992	445715
R\$ 2,500.00	1998	100000
R\$ 3,000.00	2004	163250
R\$ 20,000.00	2012	50000
R\$ 55,500.00	2020	118693
R\$ 75,700.00	2022	12405
R\$ 53,000.00	2022	10000
R\$ 6,000.00	1950	238056
R\$ 77,900.00	2022	17000
R\$ 5,450.00	1980	80000
R\$ 16,000.00	2012	98000
R\$ 6,450.00	1950	238056
R\$ 5,000.00	1970	111111
R\$ 5,000.00	1991	240000
R\$ 35,000.00	1991	75853
R\$ 44,000.00	2017	174000
R\$ 7,900.00	1980	150000
R\$ 47,000.00	2017	7000
R\$ 24,700.00	2005	288000

Figura 36: Anomalias One-class SVM do modelo VW-Gol.

Agora a análise é sobre o gráfico de dispersão do algoritmo de Principal Component Analysis. A ideia é interpretar os dados dos anúncios dispersos situados no destaque em vermelho da Figura 37.

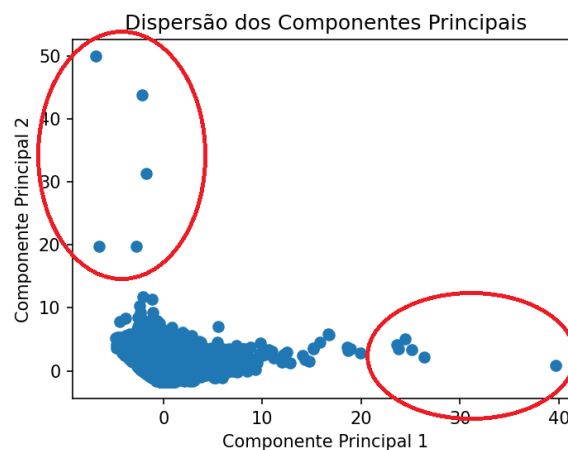


Figura 37: Gráfico anúncios dispersos PCA.

Marca	Modelo	Preço	Ano	Quilometragem
GMCHEVROLET	VERANEIO	R\$ 60,000.00	1975	1111111111
ASTONMARTIN	VANTAGE	R\$ 1,900,000.00	2020	4600
PORSCHE	911	R\$ 2,190,000.00	2021	4400
AUDI	R8	R\$ 1,990,000.00	2021	2000
FORD	LANDAU	R\$ 14,000.00	2022	1000000000
FORD	F350	R\$ 103,000.00	1965	347393034
FORD	F350	R\$ 130,000.00	1999	694686069
MCLAREN	720S	R\$ 3,500,000.00	2019	5000
ASTONMARTIN	VANQUISH	R\$ 1,900,000.00	2020	4600
AUDI	R8	R\$ 2,000,000.00	2021	2438
HONDA	CIVIC	R\$ 21,800.00	1999	100000

Figura 38: Anúncios dispersos PCA.

Os anúncios destacados nessa análise se concentram em carros de valores muito alto, acima da casa de milhões; e alguns casos interessantes a serem estudados, como por exemplo um Honda Civic do ano de 1999 com 100mil km, absolutamente existe a possibilidade, mas é um caso raro; e um Ford Landau 2022, ano inexistente para esse modelo.

7. Considerações Finais

Este projeto teve o objetivo de usar modelos de *machine learning* para identificação de padrões e inconsistências em anúncios de carros disponibilizados nas plataformas Web Olx e Seminovos. Foram usadas técnicas de *Web scraping* para extração de dados, que desde o desenvolvimento inicial já se notava um certo desafio, pois os dados não eram estruturados e seria necessária uma grande etapa de processamento e tratamento para as análises e algoritmos de aprendizagem.

Na etapa de análise e exploração dos dados identificou-se padrões com esperado, por exemplo, marcas e modelos mais anunciados; quantidade de anúncios por cidade em correlação com o número de habitantes; e análises interessantes como a proporção de veículos elétricos; evolução do tipo de transmissão ao longo dos anos.

Para a etapa de implementação dos algoritmos de *machine learning* foram utilizados algoritmos de aprendizado não supervisionado, pois para o objetivo do trabalho, encontrar inconsistências, não se tem essa informação para os anúncios. Alguns padrões e anomalias foram encontrados, principalmente relacionados a quilometragem errada, preços e ano não condizentes com os modelos de veículos. Mas por outro lado, alguns veículos de valor realmente alto e baixa quilometragem foram considerados erroneamente inconsistências.

Este projeto é repleto de trabalhos futuros para melhoria dos algoritmos, e também processo de extração de dados. Por exemplo, em vez de *Web scraping* poderia acordar com as plataformas uma disponibilização de API dos anúncios, pois os dados já são públicos, isso facilitaria o trabalho, fornecendo dados estruturados e de maneira mais rápida. Para os algoritmos, seria possível trabalhar em conjunto com informações dos usuários proprietários dos anúncios, detectar contas não validadas. Existe também a opção de reportar problema em um anúncio. Se algum anúncio foi reportado, é um importante fator para ser uma inconsistência e servir de aprendizado para outros. Até mesmo uma técnica de aprendizado por reforço poderia ser aplicada, pois como dito anteriormente, alguns anúncios foram reportados com anomalias, mas uma rápida pesquisa é possível saber corretamente as informações e atributos dos carros e dessa forma assinalar aquele automóvel como inconsistência ou não, ajudando no processo de aprendizado.

Abaixo o projeto foi resumido no *Workflow Canvas* proposto por Jasmine Vasandani para projetos de *Data Science*.

Data Science Workflow Canvas*
Start here. The sections below are ordered intentionally to make you state your goals first, followed by steps to achieve those goals. You're allowed to switch orders of these steps!

Title:		
1 Problem Statement What problem are you trying to solve? What larger issues do the problem address? Melhorar processo de compra e venda de carros usados e seminovos, fornecendo anúncios mais confiáveis e precisos.	2 Outcomes/Predictions What prediction(s) are you trying to make? Identify applicable predictor (X) and/or target (Y) variables. Identificar anúncios fraudulentos e inconsistentes.	3 Data Acquisition Where are you sourcing your data from? Is there enough data? Can you work with it? Os dados são de plataformas de anúncios de automóveis de MG na Web. Aproximadamente 90 mil anúncios extraídos por Web scraping.
4 Modeling What models are appropriate to use given your outcomes? Aprendizado não supervisionado, uma vez que os dados não são rotulados. Serão usados algoritmos de identificação de anomalias.	5 Model Evaluation How can you evaluate your model's performance? Através da acurácia, F1-score e recall. Além da análise manual dos registros identificados como inconsistentes.	6 Data Preparation What do you need to do to your data in order to run your model and achieve your outcomes? Necessário extrair dados de nomes de marcas e modelos dos automóveis nos dados não estruturados, identificar atributos em comum nas plataformas Olx e Seminovos, e padronização dos nomes desses atributos.

✓ Activation
When you finish filling out the canvas above, now you can begin implementing your data science workflow in roughly this order:

1 Problem Statement → 2 Data Acquisition → 3 Data Prep → 4 Modeling → 5 Outcomes/Preds → 6 Model Eval

* Note: This canvas is intended to be used as a starting point for your data science projects. Data science workflows are typically nonlinear.

Figura 39: Workflow Canvas no contexto de projeto de Machine Learning em anúncios de automóveis na Web.

8. Links

URL vídeo:

<https://youtu.be/ZK8ooV9eeJU>

URL repositório:

<https://github.com/brunophmaia/cars-ads-machine-learning>

REFERÊNCIAS

Venda de automóveis. Disponível em: <<https://autopapo.uol.com.br/curta/venda-carro-usado-2022/>> Acesso em: Outubro 2022.

Tratamento de *outliers*. Disponível em: <<https://datatest.readthedocs.io/en/stable/how-to/outliers.html>> Acesso em: Janeiro 2023.

Municípios do Brasil em JSON. Disponível em: <<https://gist.github.com/letanure/3012978>> Acesso em: Fevereiro 2023.

População municípios de Minas Gerais. Disponível em: <<https://g1.globo.com/mg/triangulo-mineiro/noticia/2021/09/01/uberlandia-passa-de-706-mil-habitantes-e-segue-como-a-2a-mais-populosa-de-mg-veja-as-principais-cidades-do-triangulo-alto-paranaiba-e-noroeste.ghtml>> Acesso em: Fevereiro 2023.

K-means em Python: < <https://towardsdatascience.com/how-to-perform-kmeans-clustering-using-python-7cc296cec092> > Acesso em: Maio 2023.

One-class SVM Python: < <https://medium.com/grabngoinfo/one-class-svm-for-anomaly-detection-6c97fdd6d8af> > Acesso em: Maio 2023.