PUC Minas

IEC - Instituto de Educação Continuada
Pós-Graduação em Ciência dos Dados e Big Data

Recuperação da Informação na Web e em Redes Sociais

Comparação de Produtos Petshops Online

Alunos: Mateus Girardi e Wesley Pedro Scorsatto
Professor: Zilton Cordeiro Jr.

Dezembro
2022

PUC Minas

IEC - Instituto de Educação Continuada
Pós-Graduação em Ciência dos Dados e Big Data

Projeto Final

Comparação de Produtos Petshops Online

Trabalho apresentado ao Instituto de Educação Continuada (IEC) da pós-graduação em Ciência dos Dados e Big Data da PUC Minas, como requisito parcial para a obtenção de créditos na disciplina de Recuperação da Informação na Web e em Redes Sociais.

Aluno: Mateus Girardi e Wesley Pedro Scorsatto
Professor: Zilton Cordeiro Jr.

Dezembro
2022

Conteúdo

1 Resumo

Este projeto tem como objetivo a utilização dos conhecimentos obtidos na disciplina, para a extração dos produtos dos e-commerces Petz e Cobasi, utilizando a linguagem 'Python', com a biblioteca 'scrapy'. Após a extração, realizamos os tratamentos necessários nos dados obtidos, identificamos a similaridade das bases e analisamos os resultados.

2 Introdução

Esse trabalho tem como objetivo a comparação de produtos dos dois maiores petshops onlines do Brasil, Petz e Cobasi, o trabalho aborda desde a primeira etapa do processo, a extração e tratamento dos dados, que foi realizada utilizando a biblioteca **python** chamada **scrapy**, nele explicamos o funcionamento dos spiders e as regras de negócio de cada site que tiveram de ser seguidas para obter as informações, depois, detalhamos o processo de união dos produtos através do cálculo de similaridade que desenvolvemos utilizando as propriedades que coletamos e por fim a análise de resultados, comparando principalmente o preço dos produtos entre os sites.

3 Descrição das Atividades

3.1 Extração dos dados

Iniciamos o trabalho realizando a extração dos produtos dos sites, para isso utilizamos a biblioteca **python** chamada **scrapy**, que nos permite realizar as requisições nas páginas web, analisando e navegando sobre o conteúdo, no nosso caso recebemos e tratamos respostas em HTML, JSON e XML.

A biblioteca scrapy funciona com a criação de spiders (scripts que fazem a extração dos dados), nesse trabalho criamos dois deles, um para a Petz (petzProductSpider) e outro para a Cobasi (cobasiProductSpider).

Mas antes de qualquer codificação dos spiders é necessário analisar os sites, entender sua estrutura e traçar uma estratégia para buscar os produtos e suas informações. Em ambos os sites nossa estrategia foi buscar todas as categorias de produtos do site e através delas consultar os produtos, dessa forma temos uma garantia de que não estamos deixando nenhum produto para trás, a única falha seria se a empresa do site omiti-se alguma categoria, o que é improvável.

Outra coisa que tivemos de lidar em ambos os sites foi a paginação das informações, pois, por questões de performance, não é possível obter todos os produtos de uma categoria com apenas uma requisição. Então, para cada site tivemos que entender a forma de paginação e como utiliza-lá.

A principal dificuldade na extração dos dados se deu principalmente em entender como cada página funcionava e traçar a estratégia, mas também tivemos dificuldades com o scrapy, foi necessário testar várias abordagens e realizar algumas configurações na biblioteca para que tudo funciona-se corretamente.

3.1.1 Petz spider

No spider da Petz buscamos as categorias diretamente do HTML da página. No primeiro método parse navegamos pelo elemento da classe ".dropdown-menu"até obtermos os submenus com a classe ".submenu-text"e por fim buscamos as URLs das categorias nos elementos de link "a::attr(href)".

```
import scrapy
import json
from ..items import PetzcrawlerItemPage
from ..items import PetzcrawlerItemNumberPage
from ..items import PetzcrawlerItemProduct
```

```
class PetzProductSpider(scrapy.Spider):
    name = 'petzProductSpider';
    allowed_domains = ['petz.com.br'];
    start_urls = ['https://www.petz.com.br'];

    def start_requests(self):
        urls = [
            'https://www.petz.com.br'
        ];

        for url in urls:
            yield scrapy.Request(url = url,callback=self.parse);


    def parse(self, response):
        page_item = PetzcrawlerItemPage();
        for menu in response.css('.dropdown-menu'):
            for page in menu.css('.submenu-text'):
                page_item['page'] = page.css('a::attr(href)').get();
                next_page = response.urljoin(page_item['page']);
                yield scrapy.Request(next_page,
                callback=self.numberPageParse);
```

Depois de obter a URL da categoria, juntamos ela com a URL base do site para formar a página a ser consultada, isso é feito na variável **next_page**. Com a próxima página realizamos o request passando a próxima função como callback para tratar a resposta **self.numberPageParse**.

```
def numberPageParse(self, response):
    page_number = PetzcrawlerItemNumberPage();
    for pages in response.css('p#paginas'):
        for page in pages.css('span.paginaAtual'):
            page_number['page'] = response.url;
            page_number['pageNumber'] = response.url+"?page=1";
            yield scrapy.Request(page_number['pageNumber'],
            callback=self.productParse);
            #yield page_number;
        for page in pages.css('.pagina:not([class*="setaCor"])'):
            page_number['page'] = response.url;
            page_number['pageNumber'] = response
            .urljoin(page.css('a::attr(href)').get());
            yield scrapy.Request(page_number['pageNumber'],
```

```
            callback=self.productParse);
            #yield page_number;
```

Na função acima tratamos da paginação para obter todas as URLs válidas possíveis. No site da Petz a paginação é consideravelmente mais simples que a paginação no site da Cobasi, já que as páginas ficam inseridas no elemento HTML "p#paginas"que contem elementos de link "a"indicando todas as páginas que a determinada categoria possui e a página que você está atualmente.

Na primeira parte do metódo tratamos especificamente para buscar a primeira página da categoria, já que o site não entrega diretamente a URL como é possível ver na imagem, na segunda parte buscamos todos os elementos da class "pagina"exceto a página com a classe "setaCor", dessa forma conseguimos todas as páginas de cada categoria.
Por fim realizamos o request com as páginas obtidas incovando o método **"self.productParse"** no callback para obter os produtos.

```
def productParse(self, response):
    item_product = PetzcrawlerItemProduct()
    for jproduct in response.css('textarea.jsonGa'):
        j_inf = json.loads(jproduct.css(".jsonGa::text").get());
        item_product['url'] = response.url;
        item_product['price'] = j_inf['price'];
        item_product['name'] = j_inf['name'];
        item_product['id'] = j_inf['id'];
        item_product['sku'] = j_inf['sku'];
        item_product['category'] = j_inf['category'];
        item_product['brand'] = j_inf['brand'];
        yield item_product;
```

O método **"productParse"** busca o JSON que o próprio site da Petz insere no HTML dentro do elemento "textarea"classe "jsonGa"e realiza o parse dele utilizando o método "json.loads()"para obter os campos necessários,

[URL,price,name,id,sku,category,brand], por fim, a variável "item_product" contendo as informações é retornada.

Para rodar o spider no scrapy e ter o output com os valores retornados pelo método, utilizamos o seguinte comando dentro da pasta do projeto.

```
scrapy crawl petzProductSpider –O petzResults.json
```

3.1.2 Cobasi spider

No site da Cobasi a estrutura das páginas e seu funcionamento é significativamente mais complexa, preferimos obter as categorias através do sitemap oferecido, é possível acessar em `https://cobasi.vteximg.com.br/arquivos/categorias.xml`. Com isso foi possível desenvolver o spider para conseguir os produtos.

```
import scrapy
import json
from ..items import CobasicrawlerItemCategory
from ..items import CobasicrawlerItemProduct

class CobasiProductSpider(scrapy.Spider):
    name = 'cobasiProductSpider';
    allowed_domains = ['cobasi.com.br'];
    start_urls = ['https://cobasi.com.br',
    'https://mid-back.cobasi.com.br/catalog/products/categories/'];
    split = "/c/";
    def start_requests(self):
        urls = [
            'https://cobasi.vteximg.com.br/arquivos/categorias.xml'
        ];

        for url in urls:
            yield scrapy.Request(url = url,callback=self.parse);

    def parse(self, response):
        item = CobasicrawlerItemCategory();
        for loc in response.xpath("//url/loc"):
            item["category"] = loc.xpath("text()").get();
            next_page = item["category"];
            next_page = self.start_urls[1]
            +next_page[next_page.find(self.split) + len(self.split):];
            for i in range(1,51):
```

```
                yield scrapy
                .Request(next_page + f"?page={i}&pageSize=50",
                callback=self.productParse);
```

No spider da Cobasi buscamos as URLs das categorias através do sitemap fornecido, depois no método **parse** lemos o XML buscando a tag "loc"onde se encontra a URL da categoria, e diferente da Petz já realizamos a paginação diretamente, pois, o site da Cobasi possui um método de paginação através de API e não entrega essa informação na página, logo não conseguimos extrair as páginas e temos que criar a "própria paginação", fazemos isso utilizando um laço de repetição que para cada categoria busca 50 páginas de 50 produtos cada, em nosso análise, a maior categoria tinha 1900 produtos, então, esse método cobre todas as possíveis páginas.

Outra diferença em relação a Petz, é que a Cobasi utiliza um middleware para retornar os produtos, a pagina da Cobasi é sempre a mesma, ela apenas realiza consultas no middleware recolhendo os produtos conforme você troca de página e apresenta a informação na página, então, na paginação concatenamos as categorias obtidas com a URL `https://mid-back.cobasi.com.br/catalog/products/categories/` + as páginas e realizamos o request para obter os produtos, o seu retorno é tratado pelo método **self.productParse**.

```
def productParse(self, response):
    item_product = CobasicrawlerItemProduct();
    j_response = json.loads(response.text);
    for product in j_response["products"]:
        self.cleanProduct(product);
        for sp in product["items"]:
            for sell in sp["sellers"]:
                item_product["brand"] = product["brandName"];
                item_product["url"] = product["link"];
                item_product["name"] = sp["completeName"];
                item_product["id"] = sp["id"];
                item_product["price"] = sell["price"];
                yield item_product;
```

Por fim, o método **productParse** recebe a resposta em JSON do site e realiza o parse para o objeto item_product retornando apenas os campos necessários [URL,id,name,brand,price], diferente da Petz o campo "category"não é retornado de forma descritiva, apenas seu ID, então, decidimos não trazer na extração.

Para rodar o spider no scrapy e ter o output com os valores retornados pelo método, utilizamos o seguinte comando dentro da pasta do projeto.

```
scrapy crawl cobasiProductSpider –O cobasiResults.json
```

3.2 Cálculo da similaridade e união das informações

Após a extração das duas bases dos e-commerces, realizamos o seguinte código em python para extração do Similidade de Produtos e a geração de uma nova base com os itens com maior similaridade:

```
# Importando as bibliotecas do Python
import pandas as pd
import heapq
import json
from cdifflib import CSequenceMatcher

# Extraindo os arquivos json
df = pd.read_json('petzResults.json')
df2 = pd.read_json('cobasiResults.json')

# Removendo as linhas duplicadas das duas bases
df = df.drop_duplicates(subset='id', keep='first')
print(len(df))
df2 = df2.drop_duplicates(subset='id', keep='first')
print(len(df2))

# Fun  o desenvolvidada para identificar
# a Similaridade dos produtos das duas bases
def similary_product(lista1, lista2):
    produtos = {"produtos": []}
    for i, j in lista1.iterrows():
        p = []
        f = []
        for c, n in lista2.iterrows():
            sim_name_brand = CSequenceMatcher(None,
            f"Produto: {lista1['name'][i]} - Marca:
            {lista1['brand'][i]}",
            f"Produto: {lista2['name'][c]} - Marca:
            {lista2['brand'][c]}").ratio()
            cal_sim_price = float(lista1['price'][i])
            / float(lista2['price'][c])
            sim_price = 1 if cal_sim_price >= 0.85
            and cal_sim_price <= 1.25 else 0
```

```
            sim = sim_name_brand + sim_price

            #Adiciona a lista
            p.append([sim, i, c])

        # Busca o item com maior similaridade
        # e salva na variavel f
        f = heapq.nlargest(1, p)

        # Adiciona o item em formato JSON
        # na lista produtos criada
        d = {
                'nome_petz': lista1['name'][f[0][1]],
                'nome_cobasi': lista2['name'][f[0][2]],
                'marca_petz': lista1['brand'][f[0][1]],
                'marca_cobasi': lista2['brand'][f[0][2]],
                'preco_petz': lista1['price'][f[0][1]],
                'preco_cobasi': lista2['price'][f[0][2]],
                'similaridade': f[0][0]
            }
        produtos['produtos'].append(d)

        # Prints para valida  o dos resultados
        print(heapq.nlargest(1, p))
        print(f"Produto Petz: {lista1['name'][f[0][1]]}")
        print(f"Produto Cobasi: {lista2['name'][f[0][2]]}")
        print(f"Marca Petz: {lista1['brand'][f[0][1]]}")
        print(f"Marca Cabasi: {lista2['brand'][f[0][2]]}")
        print(f"Pre o Petz: {lista1['price'][f[0][1]]}")
        print(f"Pre o Cabasi: {lista2['price'][f[0][2]]}")
        print(f"Similaridade: {f[0][0]}\n")

        # Para de executar no produto de indice 1000,
        # pois a execu  o em todos os itens ir
        # demorar um tempo maior
        if i == 1000:
            break

# Gera um arquivo JSON com os arquivos de maior similaridade
js = json.dumps(produtos, ensure_ascii=False).encode("utf-8")
file = open('produtos.json', 'w+', encoding="utf-8")
```

```
    file.write(js.decode())
    file.close()

# Executando a fun    o desenvolvida
similary_product(df, df2)
```

Nela, estivemos realizando os seguintes passos:

- Buscamos os dois arquivos gerados após a extração dos dados;
- Removemos as linhas duplicadas das duas bases;
- Criamos a função similary product, que possui dois parâmetros a serem passados (duas listas tratadas acima). Nessa função, percorremos as duas listas com metódo 'for', para identificar a similaridade entre os produtos.
- Para calcularmos a similaridade do Nome e da Marca do produto, utilizamos a função 'CSequenceMatcher', da biblioteca 'cdifflib' (duas vezes mais rápida que a função 'SequenceMatcher', da biblioteca 'difflib');
- Para calcularmos a similaridade do preço, realizamos a divisão entre os preços dos produtos e definimos uma margem de 0,85 a 1,25 do resultado do cálculo, para aceitarmos que os dois preços estão em conformidade;
- Em seguida, somamos essas similiradidades e pegamos a maior similiradade de cada item da lista 1;
- Por fim, geramos um novo arquivo JSON, que agrupa as duas bases, onde ele identificou os itens com maior similaridade. (Pegamos somente 1000 registros, pois a execução de todos os itens irá levar dois dias em média).

4 Análise dos Resultados

Realizamos o seguinte código em python para retornar os resultados:

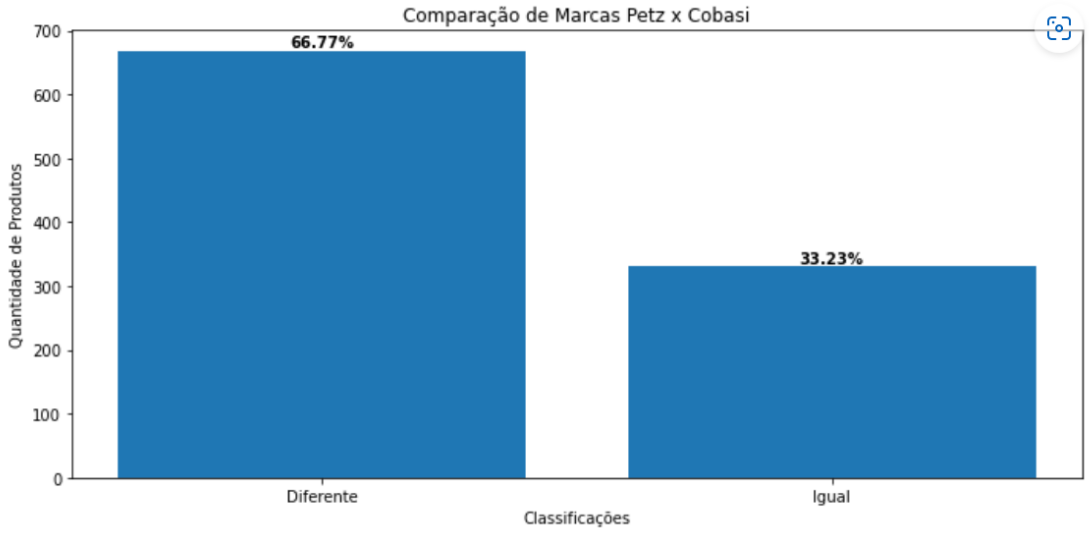
```
#Importando as bibliotecas
import json
import pandas as pd
import matplotlib.pyplot as plt

#Transformando o arquivo JSON de produtos em DataFrame
with open('produtos.json','r',encoding="utf-8") as f:
    data = json.loads(f.read())
df = pd.json_normalize(data, record_path =['produtos'])

# Gr fico da diferen a de marcas entre as duas bases de dados
x1 = ("Diferente","Igual")
total = df["nome_petz"].count()

y1 = (df[df["marca_petz"]!=df["marca_cobasi"]]["nome_petz"].count(),
df[df["marca_petz"]==df["marca_cobasi"]]["nome_petz"].count())

plt.figure(figsize=(10,5))

graph = plt.bar(x1, y1)
plt.ylabel("Quantidade de Produtos")
plt.xlabel("Classifica  es")
plt.title("Compara  o de Marcas Petz x Cobasi")

for i,p in enumerate(graph):
    width = p.get_width()
    height = p.get_height()
    x, y = p.get_xy()
    plt.text(x+width/2,
             y+height*1.01,
             str(round(y1[i]/total*100,2))+'%',
             ha='center',
             weight='bold')

plt.tight_layout()

#Gr ficos comparando os valores de cada e-commerce
```

```
x1 = ("Maior Valor","Mesmo Valor","Menor Valor")
x2 = x1
total = df["nome_petz"].count()

y1 = (df[df["preco_petz"]>df["preco_cobasi"]]["nome_petz"]
.count(),df[
df["preco_petz"]==df["preco_cobasi"]]["nome_petz"].count(),
df[df["preco_petz"]< df["preco_cobasi"]]["nome_petz"].count())
y2 = (df[df["preco_petz"]<df["preco_cobasi"]]["nome_petz"].count(),
df[df["preco_petz"]==df["preco_cobasi"]["nome_petz"].count(),
df[df["preco_petz"]>df["preco_cobasi"]]["nome_petz"].count())

plt.figure(figsize=(10,5))

plt.subplot(1, 2, 1)
graph = plt.bar(x1, y1)
plt.ylabel("Quantidade de Produtos")
plt.xlabel("Classifica   es")
plt.title("Pre os do E-commerce Petz x Cobasi")

for i,p in enumerate(graph):
    width = p.get_width()
    height = p.get_height()
    x, y = p.get_xy()
    plt.text(x+width/2,
             y+height*1.01,
             str(round(y1[i]/total*100,2))+'%',
             ha='center',
             weight='bold')

plt.subplot(1, 2, 2)
graph = plt.bar(x2, y2, color = "green")
plt.ylabel("Quantidade de Produtos")
plt.xlabel("Classifica   es")
plt.title("Pre os do E-commerce Cobasi x Petz")

for i,p in enumerate(graph):
    width = p.get_width()
    height = p.get_height()
    x, y = p.get_xy()
    plt.text(x+width/2,
```

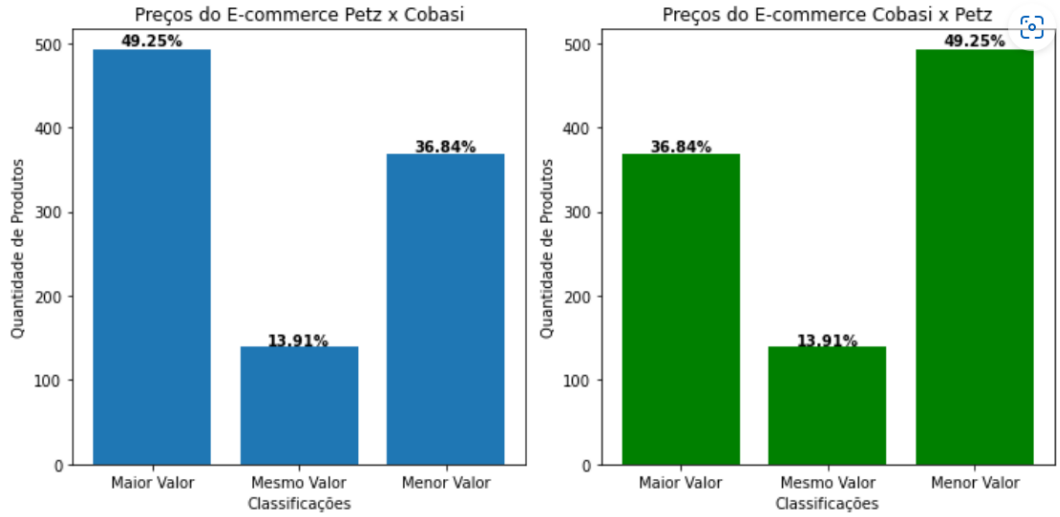
```
                y+height*1.01,
                str(round(y2[i]/total*100,2))+'%',
                ha='center',
                weight='bold')

plt.tight_layout()

#Gr ficos de M dia de Similaridade por Marca
sp = df[["marca_petz","similaridade"]].groupby("marca_petz")
.mean("similaridade")
.sort_values(by='similaridade',ascending=False)
.head(10) - 1
sp.reset_index(inplace=True)

sc = df[["marca_cobasi","similaridade"]]
.groupby("marca_cobasi").mean("similaridade")
.sort_values(by='similaridade',ascending=False)
.head(10) - 1
sc.reset_index(inplace=True)

x1 = sp["marca_petz"]
x2 = sc["marca_cobasi"]

y1 = sp.similaridade
y2 = sc.similaridade

plt.figure(figsize=(30,20))

plt.subplot(2, 1, 1)
graph = plt.bar(x1, y1)
plt.ylabel("Similaridade")
plt.xlabel("Marca")
plt.title("10 MAIORES M dias de Similaridade por Marca - Petz")
for i,p in enumerate(graph):
    width = p.get_width()
    height = p.get_height()
    x, y = p.get_xy()
    plt.text(x+width/2,
                y+height*1.01,
                round(y1[i],4),
                ha='center',
```

```
                weight='bold')


plt.subplot(2, 1, 2)
graph = plt.bar(x2, y2, color = "green")
plt.ylabel("Similaridade")
plt.xlabel("Marca")
plt.title("10 MAIORES M dias de Similaridade
por Marca - Cobasi")
for i,p in enumerate(graph):
    width = p.get_width()
    height = p.get_height()
    x, y = p.get_xy()
    plt.text(x+width/2,
             y+height*1.01,
             round(y2[i],4),
             ha='center',
             weight='bold')

plt.tight_layout()
```

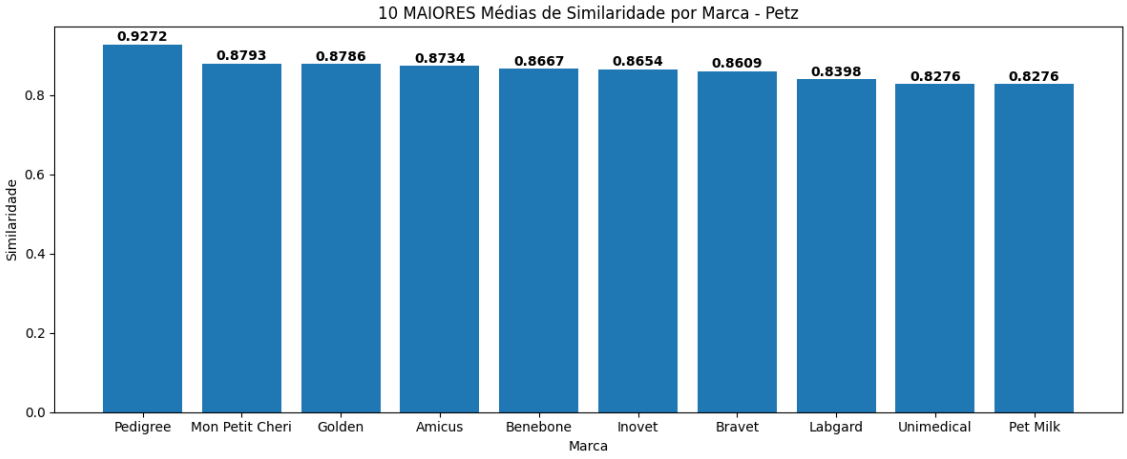
Dos 1000 registros analisados, obtivemos os seguintes resultados:
- Identificamos que 66.77% das marcas de produtos possuem diferente definicões nos e-commerces. O restante possuí a mesma marca nos produtos analisados.

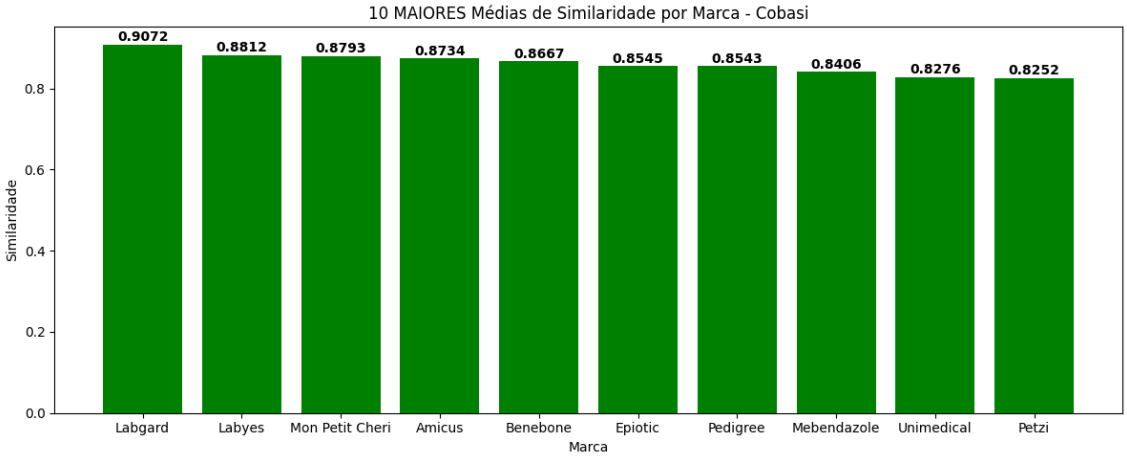


- Identificamos que os melhores preços dos produtos analisados são no e-commerce Petz (49.25%) em comparação com o e-commerce Cobasi (36.84%).



- Identificamos que a melhor média de similiridade das marcas analisadas no e-commerce Petz foi a Pedigree (0.9272). Já no e-commerce Cobasi, a mellhor média é da marca Labgard (0.9072).




10 MAIORES Médias de Similaridade por Marca - Cobasi
Similaridade
Marca
0.0
0.2
0.4
0.6
0.8
0.9072
0.8812
0.8793
0.8734
0.8667
0.8545
0.8543
0.8406
0.8276
0.8252
Labgard
Labyes
Mon Petit Cheri
Amicus
Benebone
Epiotic
Pedigree
Mebendazole
Unimedical
Petzi

5 Trabalhos Futuros

Tratar diferentes apresentações de produtos, no site da Petz a apresentação dos produtos é tratada de uma forma específica na página web e neste trabalho buscamos apenas a primeira apresentação que é mostrada no site, não buscamos coletar e tratar as apresentações nesse momento.

Evitar consultas repetidas na extração de dados, isso foi algo que teve um impacto significativo principalmente no site da Cobasi, onde em várias situações possuíamos a categoria mãe e as categorias filhas mais específicas, porém, consultavamos todas elas sem distinção, isso teve como consequências, um aumento no tempo de processamento e dados repetidos, que foram excluídos no cálculo da similaridade, uma possível melhoria futura seria analisar e ordenar a hierarquia das categorias, removendo categorias redundantes.

Utilizar as imagens dos produtos para o cálculo de similaridade, as imagens dos produtos podem facilmente serem baixadas de ambos os sites, inclusive no site Cobasi em diferentes tamanhos, porém, devido a complexidade de realizar essa análise, optamos por não realiza-lá agora.

Bibliografia

https://docs.scrapy.org/en/2.7/
https://towardsdatascience.com/sequencematcher-in-python-6b1e6f3915fc

Anexo

https://github.com/MateusGirardi/PetShopCrawler