

BUSCA



Início

E-book (Gratuito)



Sobre



Contato

Se junte a nossa comunidade e receba atualizações de artigos, tutoriais e muito mais!



✉ Insira seu email aqui

QUERO RECEBER!



Início > 2018 > julho > 3 > Data Cleaning com Similaridade de Strings em Python.



Data Cleaning com Similaridade de Strings em Python.



RODRIGO SANTANA



JULHO 3, 2018



4 COMMENTS



CAFÉ COM CÓDIGO, DATA ANALYSIS



Olá, hoje vamos **Data Cleaning com Similaridade de Strings** .

Mais uma vez quero mostrar um recurso para você que trabalha ou quer trabalhar com dados.

Como já falamos anteriormente, a tarefa mais trabalhosa de um cientista de dados é o tal de **Data Cleaning**.

Data Cleaning ? o que é isso?

É só mais um termo bonito para se referir a *Limpeza de dados, tratamento de dados, pré-processamento*, entre outras.

Pois é, esse trabalho de “arrumar” os dados é bem custoso, as vezes levam horas ou até mesmo dias.

Imagina você que tem que percorrer uma base de dados para tratar *missing values*, remover pontuações, números, dados duplicados, e por ai vai.

Para nossa felicidade, existem ferramentas e bibliotecas que facilitam nossa vida.



É o caso da biblioteca **fuzzywuzzy**

Com esta, podemos usar o recurso de similaridade de strings para ajudar na tarefa de tratameto de strings.

Antes de avançar, gostaria de agradecer a todos que tem interagido com a gente por e-mail.

É realmente muito legal receber feedbacks, críticas e sugestões.

Caso você queira entrar em contato é só responder esse e-mail 😊

Vamos ver alguns exemplos de como usar essa biblioteca.

Obs: Se você não sabe o que são missing values, te convindo a ler o nosso artigo: [Tratando Valores Faltantes com Pandas e Python](#)

Continuando aqui, imagine que tenhamos uma base de dados com colunas com dados do tipo string.

Por exemplo, dados de diagnóstico de pacientes digitados por médicos, enfermeiros e demais profissionais.

Com a **fuzzywuzzy** podemos calcular o índice de similaridade entre duas strings, por exemplo:

‘Doença cardiovascular’ e ‘Doença cardiovasculhar’

A diferença aqui é que na string da direita temos um ‘h’ entre a letra ‘a’ e a letra ‘l’.

Nitidamente conseguimos ver essa diferença, agora queremos que a biblioteca nos der um valor de similaridade entre essas strings.

Para isso usamos a função **ratio**, veja:

```
In [139]: 1 # Teste de similaridade exata
          2 fuzz.ratio('Doença cardiovascular', 'Doença cardiovasculhar')

Out[139]: 98
```

Isso mostra que de 0 a 100 o índice de similaridade foi de 98, ou seja, apenas uma letra de diferença resultou em 2 pontos a menos.

Se não houvesse diferença o valor seria 100, veja:

```
In [142]: 1 # Teste de similaridade exata
          2 fuzz.ratio('Doença cardiovascular', 'Doença cardiovascular')

Out[142]: 100
```

Se colocamos pontuações ou letras trocadas (minúsculas) taxa de similaridade cai ainda mais, veja:

```
In [144]: 1 # Tesde de similaridade com pontuação.  
          2 fuzz.ratio('Doença cardiovascular','Doença cardiovascular!!!')
```

Out[144]: 93

```
In [145]: 1 # Teste de similaridade com letra minuscula e pontuação.  
          2 fuzz.ratio('Doença cardiovascular','doença cardiovascular...')
```

Out[145]: 89

Mas, não precisamos ser tão rígidos concorda?

Talvez o nosso dado está apenas ‘sujo’ o que realmente importa está lá.

Veja o caso abaixo, podemos querer checar se a string ‘Doença cardiovascular’ está presente na string, se estiver, desconsidere o resto.

Se executarmos a função **ratio()** veja como será o valor de similaridade:

```
In [153]: 1 # Similaridade parcial  
          2 fuzz.ratio('Doença cardiovascular','###$Doença cardiovascular!!!####&"...')
```

Out[153]: 71

A similaridade foi bem baixa. Para cumprir a nossa missão podemos usar a função **partial_ratio()**.

Esta função verifica se a string em questão é similar, se sim, desconsidera o que não é similar, veja:

```
In [154]: 1 # Similaridade parcial  
          2 fuzz.partial_ratio('Doença cardiovascular','###$Doença cardiovascular!!!####&"...')
```

Out[154]: 100

Interessante, não?

A função **partial_ratio()** ainda considera a similaridade das strings em questão, se tivermos letras diferentes o índice será menor, veja:

```
In [155]: 1 # Similaridade parcial
          2 fuzz.partial_ratio('Doença cardiovascular', '###$Doença Dardiovascular!!!####&"...')

Out[155]: 95
```

E se tivermos ordem diferentes entre as strings?

Se na base tivermos 'cardiovascular Doença' ao invés de 'Doença Cardiovascular'? Como ficamos?

```
In [156]: 1 # Similaridade parcial
          2 fuzz.partial_ratio('Doença cardiovascular', 'cardiovascular Doença')

Out[156]: 67
```

O índice de similaridade ficou bem baixo, porém, sabemos que muitas vezes isso é apenas erro de digitação.

Talvez seria interessante aproveitar isso, ou seja, poderíamos considerar casos como esses como similares?

A função **partial_token_sort_ratio()** surge para resolver esse problema.

Essa função separa as palavras por espaço e ordena por ordem alfabética.

Dessa forma, para a função, a similaridade entre as strings será total, veja:

```
In [160]: 1 fuzz.partial_token_sort_ratio('Doença cardiovascular', 'cardiovascular doença#!!@')

Out[160]: 100
```

Perceba que além de ordenar as strings em ordem alfabética, esta também não diferencia letras maiúscula de minúsculas.

Isso acontece, pois, a função coloca as duas strings em letras minúsculas.

E ainda como ela é uma função do tipo **partial** ela considera se a string contém nos dados, isso significa que pontuação e caracteres estranhos não são relevantes.

Outro recuso interessante dessa lib é a possibilidade de fazer uma verificação baseado em um conjunto de valores.

Por exemplo, você pode querer ler uma base de dados com milhares de linhas e quer verificar a similaridade destas.

Veja um exemplo:

```
In [61]: 1 # Importa o método process
        2 from fuzzywuzzy import process
```

```
In [169]: 1 # Lista de Strings
        2 lista = ['Doença cardiovascular', 'doença cardiovascular!!!!...', 'Cardiovascular', 'Doenca Cardio.']
```

```
In [170]: 1 # Extrai as strings das mais similares para a menos similar.
        2 process.extract('Doença Cardiovascular', lista, scorer=fuzz.ratio)
```

```
Out[170]: [('doença cardiovascular!!!!...', 100),
            ('Doença cardiovascular', 98),
            ('Cardiovascular', 80),
            ('Doenca Cardio.', 71)]
```

Ou ainda você pode usar o parâmetro **limit** para limitar o número de strings a ser retornado.

```
In [171]: 1 process.extract('Doença Cardiovascular', lista, scorer=fuzz.ratio, limit=3)
```

```
Out[171]: [('doença cardiovascular!!!!...', 100),
            ('Doença cardiovascular', 98),
            ('Cardiovascular', 80)]
```

Mas e se você quer apenas a string mais similar e quer que retorne apenas strings com uma similaridade igual ou superior a 98, use a função **extractOne()** com o parâmetro **score_cutoff=98**. Veja:

```
In [172]: 1 process.extractOne('Cardiovascular', lista, scorer=fuzz.partial_ratio, score_cutoff=98)
Out[172]: ('doença cardiovascular!!!...', 100)
```

Junte-se a mais de 3.500 Mineradores

Tenha acesso gratuito ao Data Science Drops e outras atrações da nossa lista VIP!

✉ Seu melhor E-mail

QUERO ASSINAR! ➔

Data Cleaning em um Dataset

Vimos as principais funções presentes na biblioteca.

Agora vamos ver como isso realmente pode ser usado em projeto de data Science.

Imagine que você tem um dataset, com milhares de linhas e quer usar a **fuzzywuzzy** para te ajudar a manipular os dados da base.

Para simular essa situação, vou criar um DataFrame usando o pandas para exemplificar.

```
In [184]: 1 import pandas as pd
          2 from collections import OrderedDict
          3 data = OrderedDict(
          4 {
          5 'NumPaciente': [123465, 456789, 987654, 456789, 98765],
          6 'Diagnostico': ['Doenc.Cardiovascular',
          7                  'Doença Cardio.',
          8                  'Doença Cardiovascular.', 'Cardiovascular!!',
          9                  '%$#Doença Cardiovascular&&'
         10                ]
         11 })
         12
         13 dataset = pd.DataFrame(data)
```

O meu dataset agora contém as colunas **NumPaciente** e **Diagnóstico**. Veja:

```
In [186]: 1 dataset
```

Out[186]:

	NumPaciente	Diagnostico
0	123465	Doenc.Cardiovascular
1	456789	Doença Cardio.
2	987654	Doença Cardiovascular.
3	456789	Cardiovascular!!
4	98765	%\$#Doença Cardiovascular&&

O que queremos é tratar os dados da coluna **Diagnostico**.

Para isso, irei criar uma função que irá aplicar a fuzzywuzzy nos dados para tratar os dados.

Segue o código da função:

```
In [136]: 1 def Aplica_Fuzzy(dado, lista, tipo_score, valor_corte):  
          2     return process.extractOne(dado, choices=lista, scorer=tipo_score, score_cutoff=valor_corte)|
```

Veja que é uma função extremamente simples, esta recebe apenas 4 parâmetros. Que são:

1. **dado**: O dado será a coluna que será analisada, no nosso caso a coluna Diagnostico.
2. **lista**: A lista de dados que será usada para análise.
3. **tipo_score**: A função que vamos usar para calcular a similaridade, pode ser a ratio, partial_ratio ou qualquer outra mostrada anteriormente.
4. **valor_corte**: O índice mínimo de similaridade que queremos.

A função usa a função usa o método **extractOne()** para retornar a string mais similar possível igual ou superior ao um índice informado.

Com a função acima, vou usa-la para inspecionar os dados e obter a string mais similar a uma string correta.

Quero substituir todas as strings da coluna Diagnostico por uma mais próxima a string 'Doença cardiovascular'.

Antes de alterar o DataFrame, vamos ver qual será a string mais similar com a função extractOne:

```
In [189]: 1 process.extractOne('Doença Cardiovascular', choices=dataset.Diagnostico, scorer=fuzz.ratio, score_cutoff=95)  
Out[189]: ('Doença Cardiovascular.', 100, 2)
```

Podemos ver que a string mais similar é 'Doença Cardiovascular.' bem melhor que as outras né?

Agora irei gerar uma outra coluna no DataFrame preenchendo esta com uma string mais similar. Veja:

```
In [190]: 1 dataset['Diagnostico2'] = Aplica_Fuzzy('Doença Cardiovascular',dataset.Diagnostico, fuzz.ratio,95)[0]
```

```
In [188]: 1 dataset
```

```
Out[188]:
```

	NumPaciente	Diagnostico	Diagnostico2
0	123465	Doenc.Cardiovascular	Doença Cardiovascular.
1	456789	Doença Cardio.	Doença Cardiovascular.
2	987654	Doença Cardiovascular.	Doença Cardiovascular.
3	456789	Cardiovascular!!	Doença Cardiovascular.
4	98765	%%\$#Doença Cardiovascular&&	Doença Cardiovascular.

O comando acima chama a função `Aplica_Fuzzy()` passando os parâmetros e no final usei o `[0]` para obter apenas o primeiro valor da tupla retornada, que no caso é a string 'Doença Cardiovascular.'

Veja que temos agora uma coluna chamada '**Diagnostico2**' com os dados mais corretos.

Junte-se a mais de 3.500 Mineradores

Tenha acesso gratuito ao Data Science Drops e outras atrações da nossa lista VIP!

QUERO ASSINAR! ➔

Conclusão

Neste artigo vimos como a biblioteca **fuzzywuzzy** pode ser aplicada na tarefa de limpeza dos dados.

Essa é uma aplicação desta biblioteca, mas podemos usá-la para diversas tarefas.

Caso tenha alguma consideração, basta responder esse e-mail, será um prazer lhe responder.

Um Abraço!

 Sobre **Rodrigo Santana**

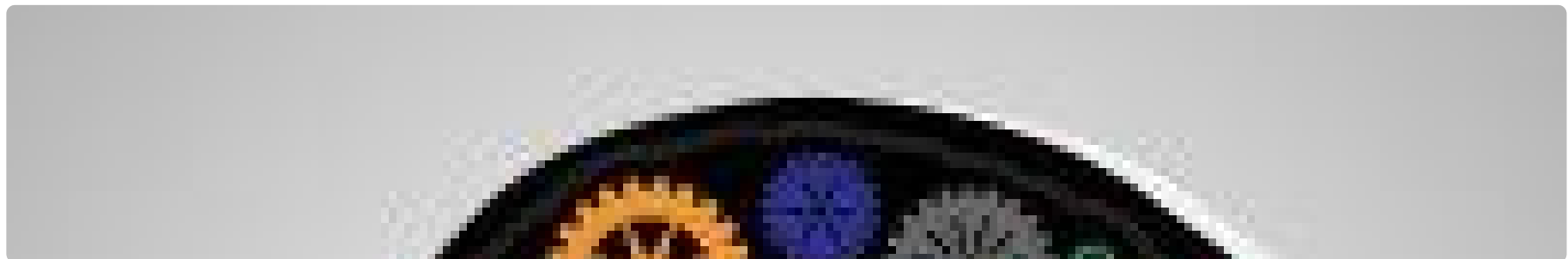


Mestrando em Ciência da Computação, interessado em Machine Learning, NLP e Data Science.

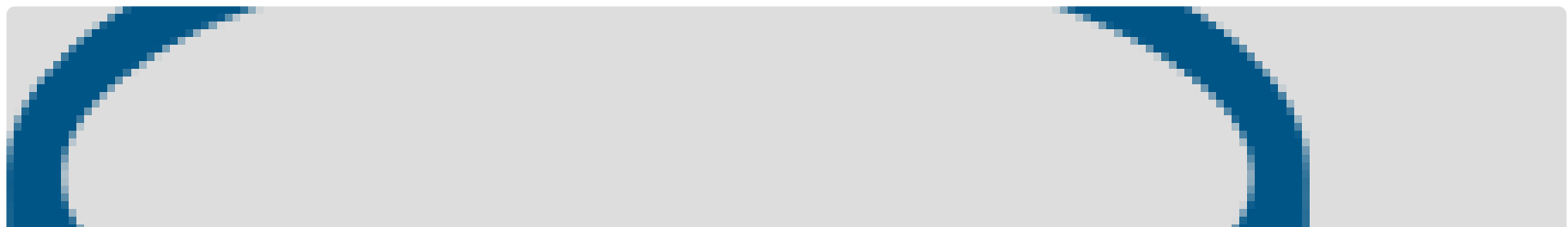
 Artigos relacionados



Exploratory Data Analysis (EDA): Aprenda Definitivamente como Extrair Valiosos Insights de Bases de Dados Reais



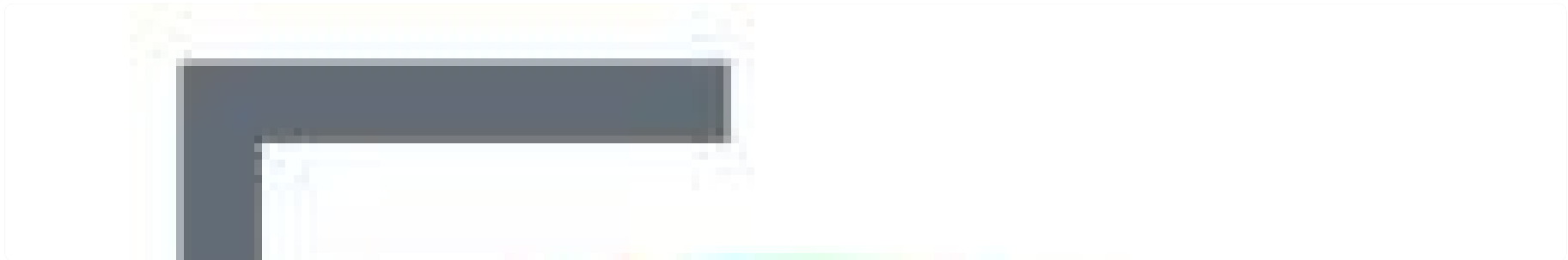
Café com Código #21 GridSearch: Melhore a eficiência dos seus algoritmos de Machine Learning



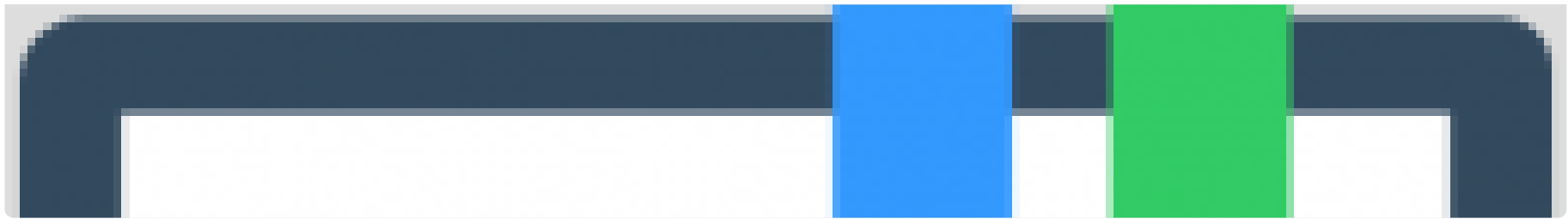
Café com Código #16: Split Dataset, Como separar conjuntos de treino e conjuntos de teste?



Para trabalhar com Data Science tenho que saber programar?



Café com Código #13: Visualizando Mapas Interativos com Python



7 Tipos de Gráficos que Todo Cientista de Dados Deve Conhecer



ARTIGO ANTERIOR

[Machine Learning de forma visual, rápida e fácil](#)

PRÓXIMO ARTIGO



[Named Entity Recognition: Aprenda como implementar usando Python e o Framework de NLP Spacy.](#)

4 COMENTÁRIOS

www.minerandodados.com.br

 1 Iniciar sessão ▾

 Recomendar

 Partilhar

Mostrar primeiro os mais votados ▾



Escreva o seu comentário...

INICIE SESSÃO COM O

OU REGISTE-SE NO DISQUS

Nome



renata c • há 2 meses

Oi Rodrigo! E se eu precisar fazer um cleansing com vários termos diferentes? Tipo: doença cardiovascular, diabetes, fratura.... como posso proceder?

^ | v • Responder • Partilhar ›



Rodrigo Santana Ferreira Moderador ➔ renata c • há 2 meses

Oi Renata, tudo bem?

Nesse caso você poderia fazer interar sobre esses termos diferentes e aplicar o fuzzywuzzy em cada termo, por exemplo:

```
termos = ['cardiovar','diabetes', 'diabet', 'fratura']  
termos_corretos = ['cardiovascular','diabetes','fratura']
```

```
for a,b in zip(termos,termos_corretos):  
    process.extract(a,b)
```

Pensei em algo simples assim, talvez lhe ajude.

Um Abraço.

^ | v • Responder • Partilhar ›



renata c ➔ Rodrigo Santana Ferreira • há 2 meses

Muito obrigada :D

^ | v • Responder • Partilhar ›



Rodrigo Santana Ferreira Moderador ➔ renata c • há 2 meses

Por nada. :)

^ | v • Responder • Partilhar ›

TAMBÉM NO WWW.MINERANDODADOS.COM.BR

Entenda o Algoritmo K-means e Saiba como Aplicar essa Técnica.

6 COMENTÁRIOS • há 9 meses

Felipe Santana — Obrigado Marcos!

Análise de Sentimentos – Aprenda de uma vez por todas como funciona utilizando dados do ...

28 COMENTÁRIOS • há um ano

Leonardo Vilarinho — Muito legal a postagem, só achei o dataset meio difícil por conta do assunto tratado, é difícil achar mensagens classificadas ...

Café com Código #17: Manipulando Arrays com Numpy



2 COMENTÁRIOS • há 9 meses

Rodrigo Santana Ferreira — Opa, que bom que gostou Victor.Forte Abraço :)

Exploratory Data Analysis (EDA): Aprenda Definitivamente como Extrair Valiosos Insights ...

15 COMENTÁRIOS • há um ano

Rodrigo Santana Ferreira — Oi Renata, que bom que gostou do artigo.Obrigado pela visita.Um abraço.

✉ Subscriver  Acerca do DisqusAdicionar o DisqusAdicionar  Disqus' Privacy PolicyPolítica de privacidadePrivacidade



MINERADOR

Eu sou o **Minerador** e vou te ensinar tudo sobre o universo **Data Science** e muito mais...prepare suas ferramentas e mãos ao código!

SAIBA MAIS +

▶ Arquivo

- ▶ Café com Código #01: Tratando Valores Faltantes com Pandas e Python
- ▶ Café com Código #02: Scatter Plot – Visualizando a Dispersão dos Dados
- ▶ Café com Código #03: Normalização de Dados com Weka
- ▶ Café com Código #04: Nuvem de Tags com Python
- ▶ Café com Código #05: Processamento de Linguagem Natural com NLTK
- ▶ Café com Código #06: Introdução a Machine Learning com Scikit-Learn
- ▶ Café com Código #07: RapidMiner: Data Science sem escrever uma linha de código
- ▶ Café Com Código #08: Weka – Consultando um Banco de Dados MySQL
- ▶ Café Com Código #09: Entendendo Métricas de Avaliação de Modelos
- ▶ Café com Código #10: Consultando dados do MongoDB com Python



► Data Science do Zero



► Café com Código





Machine Learning



Data Analysis



Ferramentas

🔍 Buscar por:



Minerando Dados · 2018 © Todos os direitos reservados