Cálculo de distância entre duas geolocalizações

None

Richard Wallan

ScientifiCloud

Table of contents

1. Cálculo de distância entre duas geolocalizações

O projeto foi feito para calcular a distância entre duas geolocalizações.

As etapas são construídas desde a limpeza e formatação de CEPs, utilização da API Google para buscar as geolocalizações dos CEPs e o cálculo da distância entre as geolocalizações obtidas.

1.1 Estrutura do projeto

```
mkdocs.yml          # Configuração do mkdocs.
requirements.txt    # Bibliotecas python necessárias.
Makefile            # Arquivo make com comandos úteis.
.gitignore          # Configuração git.
README.md           # Descrição do projeto.

docs/
    index.md                    # Documentação da estrutura do projeto.
    01_cep_cleaning.md          # Documentação do script de limpeza de CEP.
    02_geo_search.md            # Documentação do script de busca das geolocalizações.
    03_distance_calculate.md    # Documentação do script para cálculo das distâncias.
    funs_cep.md                 # Documentação do script com funções úteis.

config/
    faixacep.txt    # Contém as faixas de CEPs e sua respectiva localidade. Construído a partir dos sites do correios.
    key_google.txt  # Chave de acesso para a API do google. Solicitar ao autor.

data_cep/ # Diretório com os **data frames** necessários para a execução do projeto. Solicitar ao autor.

sample/
    01_cep_cleaning.py          # Limpeza de CEP.
    02_geo_search.py            # Busca de geolocalizações.
    03_distance_calculate.py    # Cálculo de distâncias.
    funs_cep.py                 # Funções úteis.
```

1.2 Para o projeto

Para poder atender aos requisitos necessários do projeto, escolha um dos tópicos a seguir que encaixe com as suas ferramentas:

Execute o arquivo Makefile com o GNU Make. Nele será criado um ambiente virtual e instalará todos os pacotes necessários do projeto. **É necessário ter o virtualenv instalado.**

Para executar via GNU Make:

```
make install
```

Caso não tenha o GNU Make. É aconselhável criar um ambiente virtual a partir da sua ferramente de escolha e execute o requirements.txt através do PIP.

Algumas ferramentas para a criação de ambientes virtuais:

```
virtualenv
venv
poetry
anaconda
```

Para executar via PIP:

```
pip install -r requirements.txt
```

1.3 Links úteis

Site de consulta de faixas de CEPs: https://buscacepinter.correios.com.br/app/faixa_cep_uf_localidade/index.php

GNU Make: https://www.gnu.org/software/make/

Anaconda: https://www.anaconda.com

2. Script - Limpeza e formatação de CEPs

O script tem como objetivo limpar e formatar os CEPs para um padrão. O padrão adotado é para que o script de busca de geolocalizações possa ser o mais preciso possível.

A seguir será descrito as seções presentes no script:

2.1 Importando dados

Importa os dados com as informações de CEP a serem formatados.

2.2 Aplicando tratamento nos dados de CEP

Aplica a classe: `FaixaCep`, e as funções: `limpar_cep()`, `formatar_cep()`. Para mais informações, clique aqui.

2.2.1 Dicionário de faixas de CEP

Transforma o arquivo faixacep.txt em um dicionário para ser consultado.

```
dict_cep = eval(open("../config/faixacep.txt", "r").read())
faixacep = FaixaCep(dict_cep)
```

2.2.2 Tratamento

Limpa as strings de CEP, removendo qualquer caracter que não seja número e sinaliza como `None` os CEPs inválidos.

```
df["cepaluno"] = df["cepaluno"].astype("str")
df["cepaluno_limpo"] = df["cepaluno"].apply(limpar_cep)
df["cepunidade_limpo"] = df["cepunidade"].apply(limpar_cep)
```

O arquivo é filtrado para somente aqueles que contém CEP.

```
df_notnull = df.dropna(inplace=False)
df_notnull[["cepaluno_limpo", "cepunidade_limpo"]] = df_notnull[
    ["cepaluno_limpo", "cepunidade_limpo"]
].astype("int")
```

O processo é repetido para os alunos e para a unidade de estudo.

2.2.3 Formatação

Constrói a formatação a ser utilizada para a busca de geolocalização tanto para os alunos quanto para as unidades de estudo.

2.3 Exportando CEPs limpos

Exportando dados limpos para data_cep/

2.3.1 Para mais informações

Para mais informações, consulte o texto Funções úteis.

3. Script - Busca de geolocalização

O script tem como objetivo buscar a geolocalização dos dados de Estado - CEP.

A seguir está descrito as seções presentes no script.

3.1 Importando dados limpos

Import os dados limpos obtidos do script de limpeza de CEPs.

3.2 Criando array apenas com CEPs distintos

Cria uma lista de CEPs únicos tanto de alunos quanto das unidades.

3.3 Buscando geolocalização

Primeiramente faz uma verificação para encontrar a base geocode.csv existente de geolocalizações para CEPs conhecidos.

```
if os.path.isfile(PATH):
    df_verify = pd.read_csv(PATH)
else:
    df_verify = pd.DataFrame(columns=column_name)
```

Faz um filtro para executar a API para os CEPs que não constam na base de dados. Caso não haja esse arquivo, é criado uma lista vazia e é feito a busca para todos os CEPs.

```
cep_distinct = [cep for cep in cep_distinct if cep not in df_verify["cep"].values]
```

Feito a lista de CEPs definitiva, uma lógica de lotes é executada. A API é executada em uma sequência de 100 CEPs. E após finalizado a sequência é salvo no arquivo geocode.csv as informações coletadas. Esse passo é executado até percorrer a lista de todos os CEPs.

```
BUFFER_SIZE = 100
for i in range(0, n_distinct, BUFFER_SIZE):
    start_at = i
    end_at = min(start_at + BUFFER_SIZE, n_distinct)
    temp = []
    print(f"Buscando CEPs {start_at} - {end_at}")
    for cep in cep_distinct[start_at:end_at]:
        geocode = geo.locate_geocode(cep)
        temp.append([cep, geocode])
    df_geocode = pd.DataFrame(temp, columns=column_name)

    if not os.path.isfile(PATH):
        df_geocode.to_csv(PATH, index=False)
    else:
        df_geocode.to_csv(PATH, mode="a", header=False, index=False)
```

3.3.1 Para mais informações

Para mais informações, consulte o texto Funções úteis.

4. Script - Calcular distância das geolocalizações

O script tem como objetivo executar o cálculo de distância entre duas geolocalizações.

A seguir será descrito as seções presentes no script:

4.1 Importando dados

Importa os dados contendo as informações das geolocalizações e os dados contendo os CEPs dos alunos e unidades.

4.2 Dando merge com os CEPs conhecidos

A partir dos dados contendo as geolocalizações e os CEPs únicos, é dado um merge com os CEPs de alunos.

```
df_aluno_geocode = df_notnull[["DR_CD_MATRICULA", "cepaluno_f"]].merge(
    df_unique_geocode, left_on="cepaluno_f", right_on="cep"
)
```

O mesmo é feito para os CEP de unidade:

```
df_unidade_geocode = df_notnull[["DR_CD_MATRICULA", "cepunidade_f"]].merge(
    df_unique_geocode, left_on="cepunidade_f", right_on="cep"
)
```

Contendo um banco de dados com as geolocalizações dos alunos e outro com as das unidades, é feito um merge com os dois bancos.

```
df_merged = df_aluno_geocode.merge(
    df_unidade_geocode,
    left_on="DR_CD_MATRICULA",
    right_on="DR_CD_MATRICULA",
    how="inner",
)
df_merged = df_merged.merge(
    df_notnull[["DR", "DR_CD_MATRICULA"]],
    left_on="DR_CD_MATRICULA",
    right_on="DR_CD_MATRICULA",
)
```

4.3 Cálculo de distância

Seção destinada para calcular a distância entre as geolocalizações

4.3.1 Filtrando para geolocalizações existentes

Remove as linhas que não contém informação da geolocalização do aluno ou da unidade.

4.3.2 Distância

É feito o cálculo da distância para a métrica de Haversine e para o GeoDesic. Feito os cálculos, um novo data frame é salvo com as informações:

- df_distance_full.csv - Possui todas as colunas existentes;
- df_distance.csv - Possui as colunas de DR_CD_MATRICULA, distance_hav e distance_dg.

4.3.3 Para mais informações

Para mais informações, consulte o texto Funções úteis.

5. Script - Funções úteis

5.1 `FaixaCep`

Bases: `dict`

Consulta de faixas de CEP de acordo com a sua localidade.

Parameters:

Name	Type	Description	Default
`dict`	`dict`	Objeto contendo o dicionário com as faixas de CEP e sua localidade correspondente. O dict deve ser no formato: {range(start,end): "localidade"}	required

Source code in `sample/funs_cep.py`

```
44  class FaixaCep(dict):
45      """Consulta de faixas de CEP de acordo com a sua localidade.
46
47      Args:
48          dict (dict): Objeto contendo o dicionário com as faixas de CEP e sua localidade correspondente. O dict deve ser no formato: {range(start,end):
49  "localidade"}
50      """
51
52      def __getitem__(self, item: int) -> str:
53          """Busca localidade a partir de uma consulta em faixas de valores.
54
55          Args:
56              item (int): CEP a ser consultado.
57
58          Returns:
59              str: Localidade correspondete ao CEP inserido.
60          """
61
62          if not isinstance(item, range):
63              for key in self:
64                  if item in key:
65                      return self[key]
66              return None
67          else:
              return super().__getitem__(item)
```

5.1.1 `__getitem__(item)`

Busca localidade a partir de uma consulta em faixas de valores.

Parameters:

Name	Type	Description	Default
`item`	`int`	CEP a ser consultado.	required

Returns:

Name	Type	Description
`str`	`str`	Localidade correspondete ao CEP inserido.

Source code in `sample/funs_cep.py`

```
def __getitem__(self, item: int) -> str:
    """Busca localidade a partir de uma consulta em faixas de valores.

    Args:
        item (int): CEP a ser consultado.

    Returns:
        str: Localidade correspondete ao CEP inserido.
    """

    if not isinstance(item, range):
        for key in self:
            if item in key:
                return self[key]
        return None
    else:
        return super().__getitem__(item)
```

5.2 `GeoCode`

Acessa a API do GoogleMaps para recuperar informações de latitude e longitude.

Source code in `sample/funs_cep.py`

```
class GeoCode:
    """Acessa a API do GoogleMaps para recuperar informações de latitude e longitude."""

    def __init__(self, path="../config/key_google.txt"):
        """Utiliza a client key no arquivo key_google.txt para estabelecer conexão com a API.

        Args:
            path (str): Diretório e nome do arquivo que está a key para a API google.
        """
        f = open(path, "r")
        f = f.read()
        self.key = eval(f)
        self.gmaps_key = googlemaps.Client(key=self.key)

    def locate_geocode(self, cep: str) -> tuple:
        """Consulta a latitude e longitude do CEP informado.

        Args:
            cep (str): CEP formatado, por exemplo: Distrito Federal - XXXXX-XXX. A função formatar_cep() formata o CEP da forma correta a ser utilizado.

        Returns:
            tuple: Latitude e Longitude a partir do CEP.
        """
        print(f"Buscando geolocalização para {cep}")
        g = self.gmaps_key.geocode(cep)
        try:
            self.lat = g[0]["geometry"]["location"]["lat"]
            self.lon = g[0]["geometry"]["location"]["lng"]
            return self.lat, self.lon
        except:
            return None
```

5.2.1 `__init__(path='../config/key_google.txt')`

Utiliza a client key no arquivo key_google.txt para estabelecer conexão com a API.

Parameters:

Name	Type	Description	Default
`path`	`str`	Diretório e nome do arquivo que está a key para a API google.	`'../config/key_google.txt'`

Source code in `sample/funs_cep.py`

```python
def __init__(self, path="../config/key_google.txt"):
    """Utiliza a client key no arquivo key_google.txt para estabelecer conexão com a API.

    Args:
        path (str): Diretório e nome do arquivo que está a key para a API google.
    """
    f = open(path, "r")
    f = f.read()
    self.key = eval(f)
    self.gmaps_key = googlemaps.Client(key=self.key)
```

5.2.2 `locate_geocode(cep)`

Consulta a latitude e longitude do CEP informado.

Parameters:

Name	Type	Description	Default
`cep`	`str`	CEP formatado, por exemplo: Distrito Federal - XXXXX-XXX. A função formatar_cep() formata o CEP da forma correta a ser utilizado.	required

Returns:

Name	Type	Description
`tuple`	`tuple`	Latitude e Longitude a partir do CEP.

Source code in `sample/funs_cep.py`

```python
def locate_geocode(self, cep: str) -> tuple:
    """Consulta a latitude e longitude do CEP informado.

    Args:
        cep (str): CEP formatado, por exemplo: Distrito Federal - XXXXX-XXX. A função formatar_cep() formata o CEP da forma correta a ser utilizado.

    Returns:
        tuple: Latitude e Longitude a partir do CEP.
    """
    print(f"Buscando geolocalização para {cep}")
    g = self.gmaps_key.geocode(cep)
    try:
        self.lat = g[0]["geometry"]["location"]["lat"]
        self.lon = g[0]["geometry"]["location"]["lng"]
        return self.lat, self.lon
    except:
        return None
```

5.3 `calcular_distancia(origin, destination, metric='hav')`

Cálculo de distância entre dois pontos a partir das suas geolocalizações.

Parameters:

Name	Type	Description	Default
`origin`	`tuple`	Latitude e longitude da localização de origem.	required
`destination`	`tuple`	Latitude e longitude da localização de destino.	required
`metric`	`str, opcional`	Métrica a ser utilizada para o cálculo da distância. As métricas disponíveis são: hav - Haversine e gd - GeoDesic. Padrão "hav".	`'hav'`

Raises:

Type	Description
`KeyError`	Métrica de cálculo errado. As opções válidas são: hav - Haversine ou gd - GeoDesic

Returns:

Name	Type	Description
`float`	`float`	Distância em KM.

Source code in `sample/funs_cep.py`

```
126  def calcular_distancia(
127      origin: tuple, destination: tuple, metric: metrics = "hav"
128  ) -> float:
129      """Cálculo de distância entre dois pontos a partir das suas geolocalizações.
130
131      Args:
132          origin (tuple): Latitude e longitude da localização de origem.
133          destination (tuple): Latitude e longitude da localização de destino.
134
135          metric (str, opcional): Métrica a ser utilizada para o cálculo da distância. As métricas disponíveis são: hav - Haversine e gd - GeoDesic. Padrão
136  "hav".
137
138      Raises:
139          KeyError: Métrica de cálculo errado. As opções válidas são: hav - Haversine ou gd - GeoDesic
140
141      Returns:
142          float: Distância em KM.
143      """
144      if metric == "hav":
145          distance = haversine(origin, destination)
146      elif metric == "gd":
147          distance = GD(origin, destination).km
148      else:
149          raise KeyError("Metric incorrect!")
150
         return distance
```

5.4 `formatar_cep(cep, faixacep)`

Formatação do CEP com a localidade.

Concatena a informação da localidade (UF) e o CEP.

Parameters:

Name	Type	Description	Default
`cep`	`str or int`	CEP.	required
`faixacep`	`FaixaCep`	Objeto FaixaCep contendo as faixas de CEP e sua localidade correspondente.	required

Returns:

Name	Type	Description
`str`	`str`	CEP formatado, exemplo: Distrito Federal - XXXXXXXX.

Source code in `sample/funs_cep.py`

```python
def formatar_cep(cep, faixacep: FaixaCep) -> str:
    """Formatação do CEP com a localidade.

    Concatena a informação da localidade (UF) e o CEP.

    Args:
        cep (str or int): CEP.
        faixacep (FaixaCep): Objeto FaixaCep contendo as faixas de CEP e sua localidade correspondente.

    Returns:
        str: CEP formatado, exemplo: Distrito Federal - XXXXXXXX.
    """

    estado = faixacep[cep]
    cep_formatado = f"{estado} - {str(cep)}"
    return cep_formatado
```

5.5 `limpar_cep(cep)`

Limpa strings de CEP.

Retira caracteres que não sejam números. Retira CEPs com tamanhos incorretos, com 1 dígito em toda a string e retira aqueles que iniciam em uma faixa inexistente.

As faixas consultadas estão no arquivo faixas_cep.txt. As remoções existentes são aquelas que iniciam com:

remove_tuple = ("789", "76", "75")

Parameters:

Name	Type	Description	Default
`cep`	`str`	CEP	required

Returns:

Name	Type	Description
`int`	`int`	CEP em formato de int para que possa ser consultado no dicionário de faixas de CEP. Consultar documentação de FaixaCep().

Source code in `sample/funs_cep.py`

```
def limpar_cep(cep: str) -> int:
    """Limpa strings de CEP.

    Retira caracteres que não sejam números. Retira CEPs com tamanhos incorretos, com 1 dígito em toda a string e retira aqueles que iniciam em uma faixa inexistente.

    As faixas consultadas estão no arquivo faixas_cep.txt. As remoções existentes são aquelas que iniciam com:

    remove_tuple = ("789", "76", "75")

    Args:
        cep (str): CEP

    Returns:
        int: CEP em formato de int para que possa ser consultado no dicionário de faixas de CEP. Consultar documentação de FaixaCep().

    """

    remove_tuple = ("789", "76", "75")
    cep = re.sub("[^0-9]", "", cep)
    if len(cep) < 7 or len(cep) > 8:
        cep = None
    elif cep.count(cep[0]) == len(cep):
        cep = None
    elif cep.startswith(remove_tuple):
        cep = None
    elif cep.startswith("76"):
        cep = None
    if cep:
        return int(cep)
    return cep
```