escola britânica de artes criativas & tecnologia

Analista de Dados

Análise Exploratória de Dados em Python I

Módulo | Análise de Dados:
Análise Exploratória de Dados de Logística I

Caderno de Aula

Professor André Perez

Tópicos

1. Introdução ao Kaggle;
2. Introdução ao problema de negócios;
3. Exploração de dados.

Aulas

1. Projeto

- Análise exploratória de dados (artigo de referência) através das seguintes etapas:
  1. Exploração;
  2. Manipulação;
  3. Visualização;
  4. *Storytelling*.

2. Introdução ao Kaggle

Kaggle é a maior comunidade online de ciência de dados e aprendizado de máquina. A plataforma permite que usuários encontrem e publiquem conjuntos de **dados**, construam e compartilhem **notebooks** (como este do Google Colab) e participem de **competições** (que pagam muito dinheiro as vezes) e desafios de dados.

> Vamos publicar nosso notebook de exercícios na plataforma web do Kaggle para que você possa compartilhar tudo o que você aprendeu nesta primeira parte do curso e compor o seu portfólio.

2. Introdução ao problema de negócios

2.1. Loggi

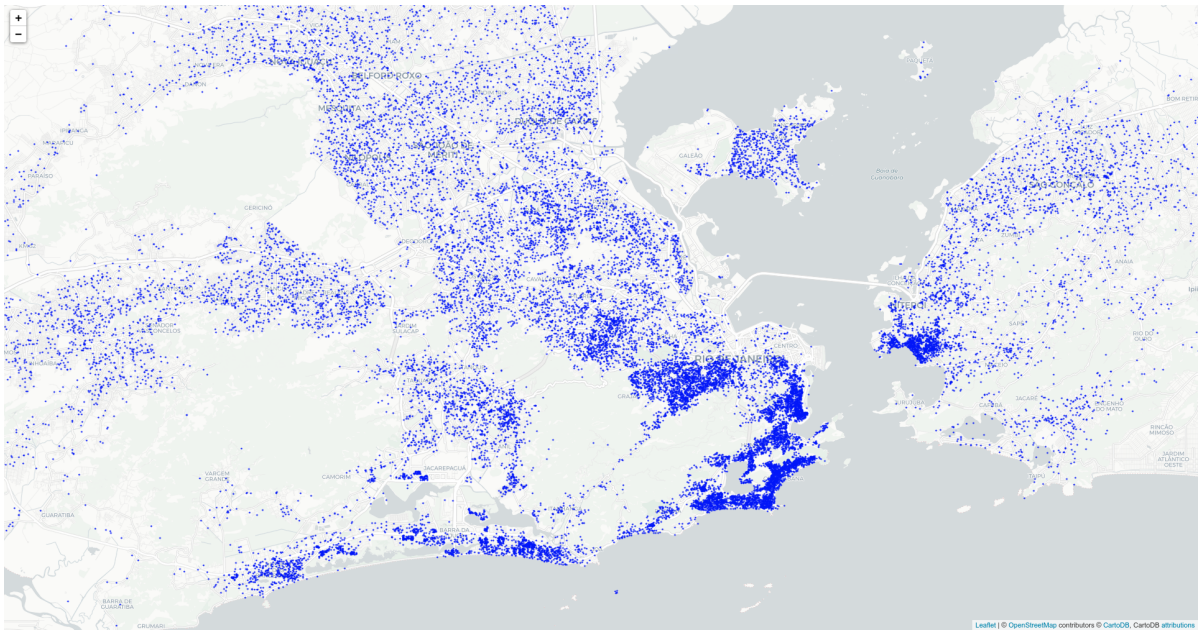
A Loggi é uma startup unicórnio brasileira de tecnologia focada em **logística**. A Loggi começou entregando apenas documentos entre 2013 e 2014. Dois anos depois, entrou no segmento de e-commerce. E, desde 2017, tem atuado nas entregas de alimentos também.

> *Somos unicórnio! Com investimentos de SoftBank, Microsoft, GGV Capital, Monashees e Kaszek e outros, a Loggi está avaliada em US$ 1 bilhão.* (fonte)

2.2. Loggi BUD

O Loggi Benchmark for Urban Deliveries (BUD) é um repositório do GitHub (link) com dados e códigos para problemas típicos que empresas de logística enfrentam: otimização das rotas de entrega, alocação de entregas nos veículos da frota com capacidade limitada, etc. Os dados são sintetizados de fontes públicas (IBGE, IPEA, etc.) e são representativos dos desafios que a startup enfrenta no dia a dia, especialmente com relação a sua escala.

A figura abaixo ilustra a dimensão do problema para a cidade do Rio de Janeiro. Na figura, cada ponto azul representa um ponto de **entrega** que deve ser alocado a um **veículo** para que a entrega seja realizada. Veículos pertencem a **hubs** de distribuição regionais espalhados pela cidade.



2.3. Dados

> **Atenção**: Vamos trabalhar com um sub conjunto dos dados originais presentes

> neste link. Em especial, consolidei em um único arquivo `JSON` as instâncias de treino de `cvrp` da cidade de Brasília.

O dado bruto é um arquivo do tipo `JSON` com uma lista de instâncias de entregas. Cada instância representa um conjunto de **entregas** que devem ser realizadas pelos **veículos** do **hub** regional. Exemplo:

```json [ { "name": "cvrp-0-df-0", "region": "df-0", "origin": {"lng": -47.802664728268745, "lat": -15.657013854445248}, "vehicle_capacity": 180, "deliveries": [ { "id": "ed0993f8cc70d998342f38ee827176dc", "point": {"lng": -47.7496622016347, "lat": -15.65879313293694}, "size": 10 }, { "id": "c7220154adc7a3def8f0b2b8a42677a9", "point": {"lng": -47.75887552060412, "lat": -15.651440380492554}, "size": 10 }, ... ] } ] ...
```

Onde:

- **name**: uma `string` com o nome único da instância;
- **region**: uma `string` com o nome único da região do **hub**;
- **origin**: um `dict` com a latitude e longitude da região do **hub**;
- **vehicle_capacity**: um `int` com a soma da capacidade de carga dos **veículos** do **hub**;
- **deliveries**: uma `list` de `dict` com as **entregas** que devem ser realizadas.

Sendo que:

- **id**: uma `string` com o id único da **entrega**;
- **point**: um `dict` com a latitude e longitude da **entrega**;
- **size**: um `int` com o tamanho ou a carga que a **entrega** ocupa no **veículo**.

3. Exploração de Dados

3.1. Coleta

O dado bruto é um arquivo do tipo `JSON` com uma lista de instâncias de entregas. Cada instância representa um conjunto d**entregas** que devem ser realizadas pelos **veículos** do **hub** regional. Exemplo:

```json [ { "name": "cvrp-0-df-0", "region": "df-0", "origin": {"lng": -47.802664728268745, "lat": -15.657013854445248}, "vehicle_capacity": 180, "deliveries": [ { "id": "ed0993f8cc70d998342f38ee827176dc", "point": {"lng": -47.7496622016347, "lat": -15.65879313293694}, "size": 10 }, { "id": "c7220154adc7a3def8f0b2b8a42677a9", "point": {"lng": -47.75887552060412, "lat": -15.651440380492554}, "size": 10 }, ... ] } ] ...
```

O dado bruto está disponível para download neste link. Vamos realizar o seu download num arquivo `JSON` com o nome `deliveries.json`.

In [ ]:
```
!wget -q << EOF
https://raw.githubusercontent.com/andre-marcos-perez/ebac-course-utils/main/dataset/deliveries.json
EOF \
-O deliveries.json
```

Vamos carregar os dados do arquivo em um dicionário Python chamado `data`:

In [ ]:
```
import json
```

```
with open('deliveries.json', mode='r', encoding='utf8') as file:
  data = json.load(file)
```

In [ ]:
```
len(data)
```

Vamos então explorar um exemplo:

In [ ]:
```
example = data[0]
```

In [ ]:
```
print(example.keys())
```

In [ ]:
```
example['name']
```

In [ ]:
```
example['region']
```

In [ ]:
```
example['origin']['lat']
```

In [ ]:
```
example['origin']['lng']
```

In [ ]:
```
example['vehicle_capacity']
```

In [ ]:
```
example['deliveries'][0]['point']['lat']
```

3.2. Wrangling

In [ ]:
```
import pandas as pd
```

In [ ]:
```
deliveries_df = pd.DataFrame(data)
```

In [ ]:
```
deliveries_df.head()
```

- **Coluna**: origin

Repare que a coluna `origin` contem dados `nested` ou aninhados na estrutura do JSON. Vamos normalizar a coluna com uma operação conhecida como `flatten` ou achatamento que transforma cada chave do JSON em uma nova coluna:

In [ ]:
```
hub_origin_df = pd.json_normalize(deliveries_df["origin"])
hub_origin_df.head()
```

Com o dados achatados, vamos junta-los ao conjunto de dados principal:

In [ ]:
```
deliveries_df = pd.merge(left=deliveries_df, right=hub_origin_df,
```

```
                           how='inner', left_index=True, right_index=True)
deliveries_df.head()
```

In [ ]:
```
deliveries_df = deliveries_df.drop("origin", axis=1)
deliveries_df = deliveries_df[
    ["name", "region", "lng", "lat", "vehicle_capacity", "deliveries"]
]
deliveries_df.head()
```

In [ ]:
```
deliveries_df.rename(columns={"lng": "hub_lng", "lat": "hub_lat"},
                     inplace=True)
deliveries_df.head()
```

- **Coluna**: deliveries

Repare que a coluna `deliveries` contem dados uma lista de dados `nested` ou aninhados na estrutura do JSON. Vamos normalizar a coluna com uma operação conhecida como `explode` ou explosão que transforma cada elemento da lista em uma linha. Por fim, faremos os `flatten` ou achatamento do resultado coluna:

In [ ]:
```
deliveries_exploded_df = deliveries_df[["deliveries"]].explode("deliveries")
deliveries_exploded_df.head()
```

In [ ]:
```
deliveries_normalized_df = pd.concat([
  pd.DataFrame(deliveries_exploded_df["deliveries"].apply(
      lambda record: record["size"])
              ).rename(columns={"deliveries": "delivery_size"}),
  pd.DataFrame(deliveries_exploded_df["deliveries"].apply(
      lambda record: record["point"]["lng"])
              ).rename(columns={"deliveries": "delivery_lng"}),
  pd.DataFrame(deliveries_exploded_df["deliveries"].apply(
      lambda record: record["point"]["lat"])
              ).rename(columns={"deliveries": "delivery_lat"}),
], axis= 1)
deliveries_normalized_df.head()
```

Com o dados explodidos, vamos normaliza-los para combina-los ao conjunto de dados principal:

In [ ]:
```
len(deliveries_exploded_df)
```

In [ ]:
```
len(deliveries_df)
```

In [ ]:
```
deliveries_df = deliveries_df.drop("deliveries", axis=1)
deliveries_df = pd.merge(left=deliveries_df, right=deliveries_normalized_df,
                         how='right', left_index=True, right_index=True)
deliveries_df.reset_index(inplace=True, drop=True)
deliveries_df.head()
```

In [ ]:
```
len(deliveries_df)
```

Com o dados em mãos, vamos conhecer um pouco melhor a estrutura do nosso conjunto de

dados.

3.3. Estrutura

In [ ]:
```
deliveries_df.shape
```

In [ ]:
```
deliveries_df.columns
```

In [ ]:
```
deliveries_df.index
```

In [ ]:
```
deliveries_df.info()
```

3.4. Schema

In [ ]:
```
deliveries_df.head(n=5)
```

- Colunas e seus respectivos tipos de dados.

In [ ]:
```
deliveries_df.dtypes
```

- Atributos **categóricos**.

In [ ]:
```
deliveries_df.select_dtypes("object").describe().transpose()
```

- Atributos **numéricos**.

In [ ]:
```
deliveries_df.drop(
    ["name", "region"], axis=1
).select_dtypes('int64').describe().transpose()
```

3.5. Dados faltantes

Dados faltantes podem ser:

- Vazios (`""`);
- Nulos (`None`);
- Não disponíveis ou aplicaveis (`na` , `NA` , etc.);
- Não numérico (`nan` , `NaN` , `NAN` , etc).

Podemos verificar quais colunas possuem dados faltantes.

In [ ]:
```
deliveries_df.isna().any()
```