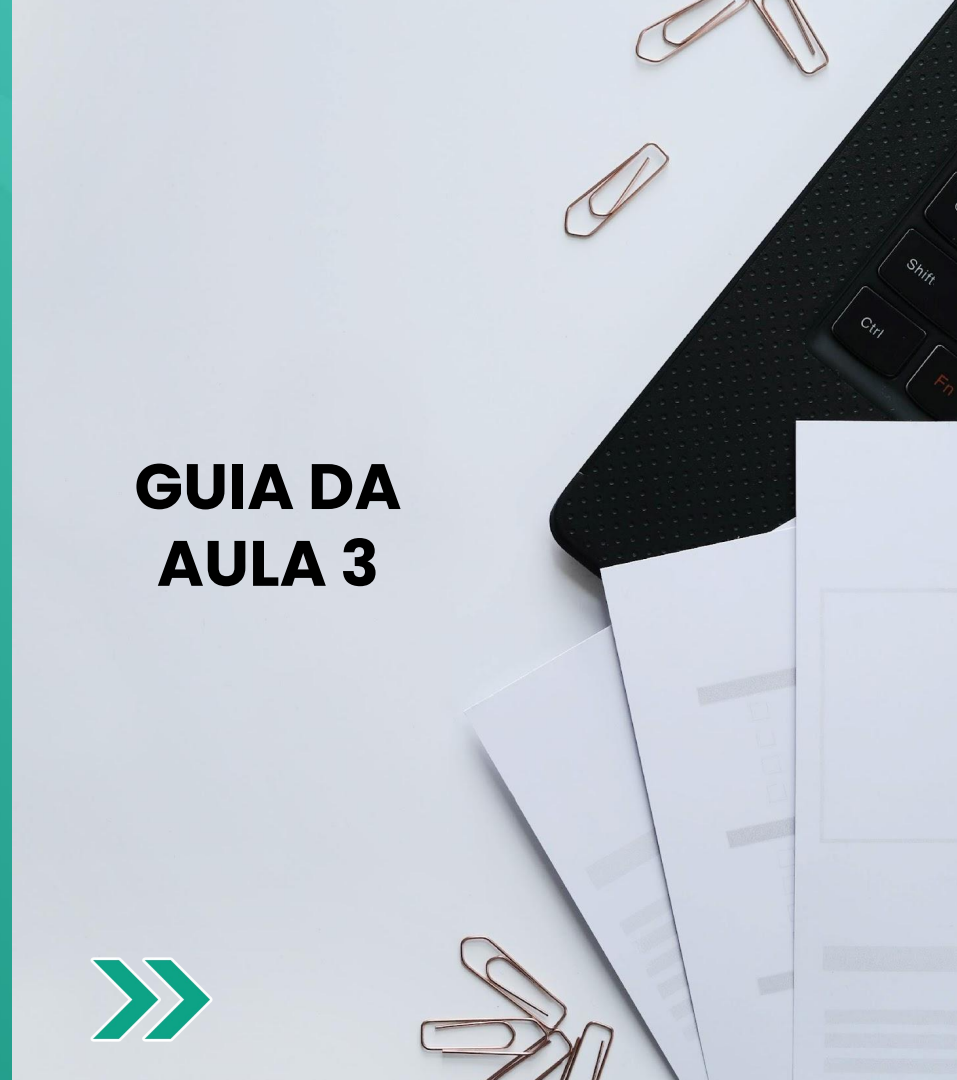
escola britânica de artes criativas & tecnologia

Profissão: Analista de dados

escola britânica de artes criativas & tecnologia

PROJETO: ANÁLISE EXPLORATÓRIA DE DADOS EM PYTHON

GUIA DA AULA 3



escola britânica de artes criativas & tecnologia

Explore dados

Acompanhe aqui os temas que serão tratados na videoaula

- **Coleta**
- **Wrangling**
- **Estrutura**
- **Schema**
- **Dados faltantes**

1. Coleta

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 }, ... ] } ] ...
```

O dado bruto está disponível para *download* no *link* https://github.com/andre-marcos-perez/ebac-course-utils/blob/main/dataset/deliveries.json

Vamos realizar o *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']
```

2. Wrangling

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

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

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

- **Coluna**: origin

Repare que a coluna `origin` contém 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 juntá-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` contém 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 normalizá-los para combiná-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 os dados em mãos, vamos conhecer um pouco melhor a **estrutura** do nosso conjunto de dados.

3. Estrutura

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

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

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

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

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()
```

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()
```