



escola
britânica de
artes criativas
& tecnologia

Analista de Dados

Análise Exploratória de Dados em Python II

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

Caderno de Aula

Professor [André Perez](#)

Tópicos

1. Manipulação;
 2. Visualização;
 3. Storytelling.
-

Aulas

0. Projeto

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

1. Manipulação

1.1. Dados

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

Processamos o **dado bruto** e construímos o DataFrame Pandas `deliveries_df` através de operações como achatamento ( `flatten` ) e explosão ( `explode` ) de colunas:

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

```
In []: import json

import pandas as pd

dado bruto em um dict

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

dado bruto no pandas

deliveries_df = pd.DataFrame(data)

coluna origin

hub_origin_df = pd.json_normalize(deliveries_df["origin"])

deliveries_df = pd.merge(left=deliveries_df, right=hub_origin_df,
 how='inner', left_index=True, right_index=True)
deliveries_df = deliveries_df.drop("origin", axis=1)
deliveries_df = deliveries_df[
 ["name", "region", "lng", "lat", "vehicle_capacity", "deliveries"]
]
deliveries_df.rename(columns={"lng": "hub_lng", "lat": "hub_lat"},
 inplace=True)

coluna deliveries

deliveries_exploded_df = deliveries_df[["deliveries"]].explode("deliveries")

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_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)
```

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

## 1.2. Enriquecimento

- **Geocodificação reversa do hub**

A **geocodificação** é o processo que transforma uma localização descrita por um texto (endereço, nome do local, etc.) em sua respectiva coordenada geográfica (latitude e longitude). A **geocodificação reversa** faz o oposto, transforma uma coordenada geográfica de um local em suas respectivas descrições textuais.

```
In []: hub_df = deliveries_df[["region", "hub_lng", "hub_lat"]]
hub_df = hub_df.drop_duplicates().sort_values(by="region").reset_index(
 drop=True
)
hub_df.head()
```

Empresas como Google, Bing e Yahoo! fornecem **geocodificação** como serviço (e cobram por isso). Existe uma projeto *open source* chamado de [OpenStreetMap](#) que mantém um serviço gratuito de geocodificação chamado [Nominatim](#), serviço este que apresenta como limitação a quantia de **uma única consulta por segundo**. Vamos utilizá-lo através do pacote Python `geopy` para fazer a operação reversa e enriquecer o nosso DataFrame principal.

```
In []: import json

import geopy
from geopy.geocoders import Nominatim

geolocator = Nominatim(user_agent="ebac_geocoder")
location = geolocator.reverse("-15.657013854445248, -47.802664728268745")

print(json.dumps(location.raw, indent=2, ensure_ascii=False))
```

Vamos então aplicar a geocodificação nas coordenadas das três regiões e extrair informações de **cidade** e **bairro**.

```
In []: from geopy.extra.rate_limiter import RateLimiter

geocoder = RateLimiter(geolocator.reverse, min_delay_seconds=1)
```

```
In []: hub_df["coordinates"] = hub_df["hub_lat"].astype(str) + ", "
hub_df["coordinates"] = hub_df["coordinates"] + hub_df["hub_lng"].astype(str)
hub_df["geodata"] = hub_df["coordinates"].apply(geocoder)
hub_df.head()
```

```
In []: hub_geodata_df = pd.json_normalize(hub_df["geodata"].apply(
 lambda data: data.raw
))
hub_geodata_df.head()
```

```
In []: import numpy as np

hub_geodata_df = hub_geodata_df[
 ["address.town", "address.suburb", "address.city"]
]
```

```

hub_geodata_df.rename(
 columns={
 "address.town": "hub_town",
 "address.suburb": "hub_suburb",
 "address.city": "hub_city"
 }, inplace=True)
hub_geodata_df["hub_city"] = np.where(
 hub_geodata_df["hub_city"].notna(),
 hub_geodata_df["hub_city"],
 hub_geodata_df["hub_town"]
)
hub_geodata_df["hub_suburb"] = np.where(
 hub_geodata_df["hub_suburb"].notna(),
 hub_geodata_df["hub_suburb"],
 hub_geodata_df["hub_city"])
hub_geodata_df = hub_geodata_df.drop("hub_town", axis=1)
hub_geodata_df.head()

```

O DataFrame `hub_geodata_df` com as informações de **cidade** e **bairro** é então combinado ao DataFrame principal `deliveries_df`, enriquecendo assim o dado.

```

In []:
hub_df = pd.merge(
 left=hub_df, right=hub_geodata_df, left_index=True, right_index=True
)
hub_df = hub_df[["region", "hub_suburb", "hub_city"]]
hub_df.head()

```

```

In []:
deliveries_df = pd.merge(
 left=deliveries_df, right=hub_df, how="inner", on="region"
)
deliveries_df = deliveries_df[[
 "name",
 "region",
 "hub_lng",
 "hub_lat",
 "hub_city",
 "hub_suburb",
 "vehicle_capacity",
 "delivery_size",
 "delivery_lng",
 "delivery_lat"
]]
deliveries_df.head()

```

- **Geocodificação reversa da entrega**

Enquanto o **hub** contem apenas **3** geolocalizações distintas, as **entregas** somam o total de **636.149**, o que levaria em torno de 7 dias para serem consultadas no servidor do Nominatim, dada a restrição de uma consulta por segundo. Contudo, para cargas pesadas como esta, o software oferece uma instalação [local](#) (na sua própria máquina) que pode ser utilizada sem restrição.

**Atenção:** Como a instalação do servidor local envolve tecnologias que estão fora do escopo deste curso (como [Docker](#)), eu vou providenciar estes dados para você através deste [link](#).

```

In []:
!wget -q << EOF

```

```
https://raw.githubusercontent.com/andre-marcos-perez/ebac-course-utils/main/dataset/deliveries-geodata.csv
EOF \
-0 deliveries-geodata.csv
```

```
In []: deliveries_geodata_df = pd.read_csv("deliveries-geodata.csv")
 deliveries_geodata_df.head()
```

```
In []: deliveries_df = pd.merge(
 left=deliveries_df,
 right=deliveries_geodata_df[["delivery_city", "delivery_suburb"]],
 how="inner",
 left_index=True,
 right_index=True
)
 deliveries_df.head()
```

## 1.3. Qualidade

Qualidade do dados está relacionado a consistência do seu `schema`, valores faltantes, etc.

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

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

- Geocodificação reversa

```
In []: 100 * (
 deliveries_df["delivery_city"].isna().sum() / len(deliveries_df)
)
```

```
In []: 100 * (
 deliveries_df["delivery_suburb"].isna().sum() / len(deliveries_df)
)
```

```
In []: prop_df = deliveries_df[["delivery_city"]].value_counts(
) / len(deliveries_df)
 prop_df.sort_values(ascending=False).head(10)
```

```
In []: prop_df = deliveries_df[["delivery_suburb"]].value_counts(
) / len(deliveries_df)
 prop_df.sort_values(ascending=False).head(10)
```

## 2. Visualização

### 2.1. Mapa de entregas por região

Vamos utilizar o pacote Python Geopandas ([link da documentação](#)) para visualizar as coordenadas dos **hubs** e das **entregas** no mapa do Distrito Federal, segmentados pela região dos **hubs**. O pacote adiciona funcionalidades geoespaciais ao pacote Python Pandas.

```
In []: !pip3 install geopandas;
```

```
In []: import geopandas
```

- **Mapa do Distrito Federal**

Vamos fazer o download dos dados do mapa do Distrito Federal do site oficial do IBGE através do seguinte [link](#) para criar o DataFrame `mapa`. Note a coluna `geometry`.

```
In []: !wget -q << EOF
https://geoftp.ibge.gov.br/cartas_e_mapas/bases_cartograficas_continuas/
bc100/go_df/versao2016/shapefile/bc100_go_df_shp.zip
EOF \ -O distrito-federal.zip
!unzip -q distrito-federal.zip -d ./maps
!cp ./maps/LIM_Unidade_Federacao_A.shp ./distrito-federal.shp
!cp ./maps/LIM_Unidade_Federacao_A.shx ./distrito-federal.shx
```

```
In []: mapa = geopandas.read_file("distrito-federal.shp")
mapa = mapa.loc[[0]]
mapa.head()
```

- **Mapa dos Hubs**

Vamos criar o DataFrame `geo_hub_df` através do DataFrame `deliveries_df`. Note a nova coluna `geometry`.

```
In []: hub_df = deliveries_df[["region", "hub_lng", "hub_lat"]].drop_duplicates(
).reset_index(drop=True)
geo_hub_df = geopandas.GeoDataFrame(
 hub_df,
 geometry=geopandas.points_from_xy(
 hub_df["hub_lng"],
 hub_df["hub_lat"]
)
)
geo_hub_df.head()
```

- **Mapa das Entregas**

Vamos criar o DataFrame `geo_deliveries_df` através do DataFrame `deliveries_df`. Note a nova coluna `geometry`.

```
In []: geo_deliveries_df = geopandas.GeoDataFrame(
 deliveries_df,
 geometry=geopandas.points_from_xy(
 deliveries_df["delivery_lng"],
 deliveries_df["delivery_lat"]
)
)
geo_deliveries_df.head()
```

- **Visualização**

```
In []: import matplotlib.pyplot as plt

cria o plot vazio
fig, ax = plt.subplots(figsize = (50/2.54, 50/2.54))

plot mapa do distrito federal
mapa.plot(ax=ax, alpha=0.4, color="lightgrey")

plot das entregas
geo_deliveries_df.query("region == 'df-0']").plot(
 ax=ax, markersize=1, color="red", label="df-0"
)
geo_deliveries_df.query("region == 'df-1']").plot(
 ax=ax, markersize=1, color="blue", label="df-1"
)
geo_deliveries_df.query("region == 'df-2']").plot(
 ax=ax, markersize=1, color="seagreen", label="df-2"
)

plot dos hubs
geo_hub_df.plot(
 ax=ax, markersize=30, marker="x", color="black", label="hub"
)

plot da legenda
plt.title(
 "Entregas no Distrito Federal por Região",
 fontdict={"fontsize": 16}
)
lgnd = plt.legend(prop={"size": 15})
for handle in lgnd.legendHandles:
 handle.set_sizes([50])
```

- **Insights:**

1. As **entregas** estão corretamente alocadas aos seus respectivos **hubs**;
2. Os **hubs** das regiões 0 e 2 fazem **entregas** em locais distantes do centro e entre si, o que pode gerar um tempo e preço de entrega maior.

## 2.2. Gráfico de entregas por região

- **Agregação:**

```
In []: data = pd.DataFrame(
 deliveries_df[['region', 'vehicle_capacity']].value_counts(normalize=True)
).reset_index()
data.rename(columns={0: "region_percent"}, inplace=True)
data.head()
```

- **Visualização:**

```
In []: import seaborn as sns

with sns.axes_style('whitegrid'):
 grafico = sns.barplot(
 data=data,
 x="region",
 y="region_percent",
```



```
ci=None,
palette="pastel"
)
grafico.set(
 title='Proporção de entregas por região',
 xlabel='Região',
 ylabel='Proporção'
);
```

- **Insights:**

1. A distribuição das **entregas** está muito concentrada nos **hubs** das regiões 1 e 2, mas pouco no da região 0. Contudo a capacidade dos veículos é mesma para todos os **hubs**, logo os **veículos** poderiam ser deslocados para as regiões de maior tráfego.

### 3. Storytelling

O *storytelling* no contexto de dados é um técnica de apresentação de resultados orientado a dados, ou seja, contar uma história baseada nos *insights* que foram gerados através da análise dos dados. Notebooks como este do Google Colab e os do Kaggle são excelentes ferramentas para conduzir e compartilhar *storytelling* de dados devido a natureza texto-código de suas células.

Para você montar o seu portfólio, eu sugiro a construção de um notebook com a seguinte estrutura (vou disponibiliza-la nos exercícios):

1. Título;
2. Breve descrição do problema;
3. Código de importação de bibliotecas;
4. Código o download/carregamento/geração de dados;
5. Etapa de exploração;
6. Etapa de limpeza e transformação;
7. Etapa de análise (com visualizações);
8. Resumo dos *insights* gerados.

Busquei organizar este notebook desta forma. Ademais, os notebooks presentes na plataforma do Kaggle são excelentes exemplos a serem seguidos, em especial os primeiros colocados em competições.

Para finalizar, algumas dicas:

1. Estruture seu código sempre de acordo com as boas práticas PEP8, assim ele será mais legível para o leitor;
2. Sempre se preocupe com a aparência dos seus gráficos, todos devem ter (no mínimo) título no topo e nos eixos e legendas;
3. Use e abuso das células de texto para estruturar seu notebook, siga as mesmas técnicas que eu utilizo nos notebooks do curso para estruturar seu texto.