Usando Python para Gerenciar Dados

Marcos F. Silva Bruno M. S. S. Melo
João Paulo de F. Ramirez

23 de março de 2024

Índice

Apresentação do Curso

Objetivo

Usando bases de dados reais, o objetivo do curso é mostrar como utilizar a *linguagem* Python para fazer o gerenciamento dos dados. Esse processo também é conhecido por "data wrangling" e consiste em realizar a limpeza e pré-processamento de bases de dados de forma a deixá-las "prontas" para o trabalho de modelagem e obtenção das respostas que se deseja obter com a análise dos dados.

O curso tem foco nas bibliotecas Python como `{pandas}` e `{polars}` para manipulação de dados, além de `{matplotlib}` e `{plotly}` para visualização de dados.

Algumas bibliotecas-chave em Python para essas tarefas incluem: - `pandas` - `polars` - `matplotlib` - `plotly` - `numpy` - `scipy` - `scikit-learn`

Exemplo de código em Pandas

```
import pandas as pd

# Criando um DataFrame de exemplo
df = pd.DataFrame({
    'nome': ['Alice', 'Bob', 'Carol'],
    'idade': [25, 30, 35]
})

print(df)
```

```
    nome  idade
0  Alice     25
1    Bob     30
2  Carol     35
```

Exemplo de código em Polars

```python
import polars as pl

# Criando um DataFrame de exemplo
df = pl.DataFrame({
    'nome': ['Alice', 'Bob', 'Carol'],
    'idade': [25, 30, 35]
})

print(df)
```

```
shape: (3, 2)

 nome    idade
 ---     ---
 str     i64

 Alice   25
 Bob     30
 Carol   35
```

Algumas referências bibliográficas

Para quem tiver interesse em avançar no estudo do Python, seguem alguns materiais que podem ser úteis:

- Python Data Science Handbook
- Pandas Documentation
- Polars Documentation
- Matplotlib Documentation
- Seaborn Documentation

Conteúdo programático

Este curso foi pensado com o objetivo de mostrar os aspectos práticos envolvidos no processo de análise de dados tendo um enfoque nas operações que usualmente são realizadas nas bases de dados objeto de análise.

Este material está divido em cinco partes:

Parte 1 - Importação e exportação dos dados
Parte 2 - Operações típicas em bases de dados
Parte 3 - Limpeza e arrumação dos dados
Parte 4 - Introdução à visualização de dados
Parte 5 - Análise exploratória e descritiva de dados

Essa divisão tem cunho meramente didático.

Bases de dados utilizadas

Procuramos utilizar bases de dados semelhantes àquelas que o usuário possa lidar no seu dia a dia.

Especificamente serão utilizadas ao longo dessa apostila as seguintes bases de dados:

- `ISP2020.xlsx` - esta base de dados, em formato do Excel, é disponibilizada pelo Mi-nistério da Previdência Social e contém informações sobre o indicador de situação previdenciária de todos os Regimes Próprios de Previdência Social - RPPS do Brasil.
- `fundos_investimentos.xlsx` - esta base de dados contêm uma relação de fundos de investimentos e seu enquadramento conforme norma do Conselho Monetário Nacional.
- `UG123400_Jan18-Set18_Cta621201101.xls` - base de dados relativa ao razão contábil da conta 621201101 extraída da UG 23400 - RIOPREVIDENCIA.
- `dair_br2023.Rds` - esta base de dados contém informações sobre os investimentos realizados pelos RPPS de todo o Brasil. Contêm informações para o ano de 2023. Foi obtida a partir da API do CADPREV.

O processo de análise de dados

A análise de dados pode ser vista como um processo onde o analista, partindo de uma questão que quer responder a partir da análise dos dados, identifica os dados necessários e faz a **importação** desses dados para o Python (naturalmente pode ser utilizada qualquer outra linguagem de programação ou *softawre* para análise de dados).

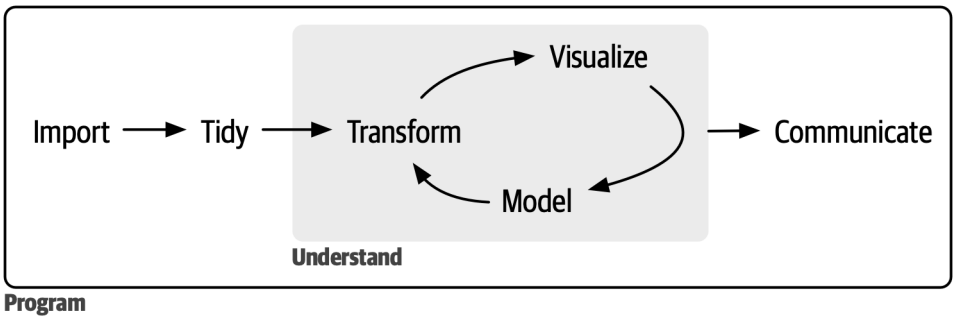
Usualmente esses dados não estarão organizados num formato no qual já seja possível realizar análises, sendo frequentemente necessário **arrumar**, **organizar** (do inglês, *tidy up the data*) esses dados.

Então o analista pode identificar a necessidade de **transformar** algumas variáveis da base de dados e **combinar** a base com outras, para gerar mais informações.

A identificação de relações entre as variáveis da base de dados, sua distribuição, a possível identificação de valores atípicos, a obtenção de novos *insights*, podem ser enormemente facilitados com a **visualização** dos dados.

Com isso o analista já está em condições de criar **modelos** que expliquem o comportamento observado das variáveis e certamente deverá **reportar**, **comunicar**, esses resultados a outras pessoas interessadas na resposta à questão que motivou toda a análise conduzida pelo analista.

Todo esse processo pode ser sintetizado na seguinte figura:[1]



[1]Fonte: R for Data Science, 2nd Ed.

Parte 1 - Importação e exportação de dados

Toda análise de dados vai ter início com a importação para Python dos dados necessários à análise a ser realizada. Para a importação de dados, existem diversas opções de bibliotecas a depender do formato do arquivo no qual os dados estão armazenados.

Uma opção muito comum é termos dados armazenados em planilhas eletrônicas, em particular o Excel. Mas diversas outras opções de arquivos para armazenamento de dados existem como, por exemplo, `csv` e `json`.

Para a importação de dados, o `{pandas}` é uma biblioteca muito utilizada, permitindo a leitura de dados em diferentes formatos, como texto, Excel, SQL, entre outros.

Para a importação de dados no formato do Excel, o `{pandas}` faz uso de bibliotecas específicas para este fim. Uma das mais utilizadas chama-se `{openpyxl}` (para o formato mais atual `xlsx`) e `{xlrd}` (para o formato mais legado `xls`).

Para instalar o `{pandas}` e `{openpyxl}`, você pode usar o gerenciador de pacotes pip. Observe que a sintaxe é bem simples. O comando -U no pip install é opcional e indica ao pip para atualizar os pacotes a serem instalados para a versão mais recente disponível. Se os pacotes já estiverem instalados, o pip irá verificar se há uma versão mais recente disponível no Python Package Index (PyPI). Se houver, o pip irá desinstalar as versões atualmente instaladas e instalar as versões mais recentes.

```
!pip install -U pandas openpyxl xlrd
```

Depois de instalar as bibliotecas, é necessário importá-las para que suas funções sejam disponibilizadas para uso.

```
# Importando a biblioteca Pandas
import pandas as pd
```

Observe que a menos que queiramos utilizar diretamente a biblioteca `{openpyxl}`, não é necessário importá-la diretamente uma vez que o Pandas, quando necessitar realizar uma operação de leitura ou escrita para arquivos do Excel irá procurar, automaticamente, uma biblioteca instalada no sistema que sirva a este fim. No caso em tela, ele encontrará a biblioteca `{openpyxl}` e importará suas funcionalidades sem que precisemos fazê-lo de forma explícita.

Importação de dados contidos no Excel

Para a importação de dados armazenados em arquivos Excel, o Pandas possui a função read_excel que permite a leitura de arquivos no formato xlsx.

Vamos iniciar fazendo a importação dos dados contidos no arquivo excel `ISP2020.xlsx`, que contém 9 abas (ou planilhas). Estamos interessados nos dados da planilha "`GRUPOS`".

Para fazer a importação dos dados, vamos utilizar a função `read_excel()` do pacote `pandas` que já foi anteriormente carregado.

```
# Importando a planilha "GRUPOS"
grupos = pd.read_excel("./dados/ISP2020.xlsx", sheet_name="GRUPOS")
```

> **Aviso**
>
> Caso tentem executar esse código com a planilha aberta vocês obterão uma mensagem de erro!

Importado o conjunto de dados, é sempre recomendado dar uma conferida para ver se a importação ocorreu conforme esperado. Para fazer essa checagem, podemos utilizar as funções `info()`, `head()` e `tail()`.

```
# Verificando as informações básicas do DataFrame
print(grupos.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2154 entries, 0 to 2153
Data columns (total 8 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   ENTE                              2154 non-null   object
 1   GRUPO                             2154 non-null   object
 2   SUBGRUPO                          2154 non-null   object
 3   SEGURADOS ATIVOS                  2154 non-null   int64
 4   APOSENTADOS                       2154 non-null   int64
 5   PENSIONISTAS                      2154 non-null   int64
 6   ESTRUTURA DE MATURIDADE DE MASSA  2154 non-null   float64
 7   FONTE                             2154 non-null   object
dtypes: float64(1), int64(3), object(4)
memory usage: 134.8+ KB
None
```

```
# Inspeção dos registros iniciais da base de dados
print(grupos.head(2))
```

```
                  ENTE          GRUPO          SUBGRUPO  SEGURADOS ATIVOS  \
0  ABADIA DE GOIÁS - GO  PEQUENO PORTE  MENOR MATURIDADE               325
1        ABADIÂNIA - GO  PEQUENO PORTE  MENOR MATURIDADE               415

   APOSENTADOS  PENSIONISTAS  ESTRUTURA DE MATURIDADE DE MASSA     FONTE
0           36             7                          7.558140  DRAA2020
1           49            24                          5.684932  DRAA2016
```

```
# Inspeção dos registros finais da base de dados
print(grupos.tail(2))
```

```
                ENTE          GRUPO          SUBGRUPO  SEGURADOS ATIVOS  \
2152  XANGRI-LÁ - RS    MÉDIO PORTE  MENOR MATURIDADE               833
2153   ZACARIAS - SP  PEQUENO PORTE  MENOR MATURIDADE               242

      APOSENTADOS  PENSIONISTAS  ESTRUTURA DE MATURIDADE DE MASSA        FONTE
2152           75             7                         10.158537  DIPR06/2020
2153           46            10                          4.321429  DIPR04/2020
```

Aparentemente, os dados foram importados sem maiores problemas.

Exercício

1. Importar os dados contidos na planilha `RESULTADO`. Fazer a inspeção dos dados conforme mostrado. Coloque os dados no objeto `resultado`.
2. Importar os dados contidos no arquivo `fundos_investimentos.xlsx`. Fazer a inspeção dos dados.

Desafio

Importar os dados contidos no arquivo `UG123400_Jan18-Set18_Cta621201101.xls`. Coloque os dados no objeto `razao` e faça as checagens recomendadas.

Exportando dados no formato Excel

Para exportar dados que já estejam no Pyhton, não será necessário instalar nenhum pacote adicional uma vez que a biblioteca `{openpyxl}`, que utilizamos para importar dados do Excel, também possui a funcionalidade de exportação. De forma análoga à vista na seção anterior, o Pandas encapsula essa funcionalidade na função `write_excel()`

Vamos exportar para uma planilha do excel o arquivo do razão contábil que acabamos de importar.

```
pd.write_excel(razao, path="dados/razao.xlsx", index=False)
```

> Dica
>
> No Python, os objetos possuem algumas funções a eles associadas. Essas funções são chamadas de **métodos**. No contexto do Pandas, os objetos que armazenam tabelas possuem o método `to_excel()` que poderia ter sido chamado alternativamente à função `write_excel`.

O exemplo a seguir ilustra a exportação para o Excel por meio de uma chamada ao método `to_excel()` associado ao objeto `razao`.

```
razao.to_excel(razao, path="dados/razao.xlsx", index=False)
```

Parte 2 - Operações usuais em bases de dados

Objetivo do capítulo

Determinadas operações são realizadas corriqueiramente em bases de dados, como por exemplo, filtrar, ordenar, agregar, sumarizar, juntar bases de dados, criar novas variáveis na base de dados etc.

Neste capítulo nosso objetivo é mostrar como usar o `{pandas}` para realizar essas operações.

Métodos vs Funções

Conforme brevemente antecipado ao final da parte anterior, o Python implementa o paradigma de programação orientada a objetos (POO). Este paradigma se diferencia da programação procedural (tradicional) por encapsular dados e procedimentos (funções) em uma única entidade denominada *objeto*. No Python, é possível utilizar (e até mesmo mesclar) tanto o paradigma tradicional quanto a POO. Quando um procedimento não está associado a um objeto ele é denominado *função*, caso esteja, é denominado *método*. Os dados associados a um objeto são seus *atributos*.

Encadeamento de operações

O encadeamento de operações sobre um objeto do tipo tabela (`dataframe`) é muito comum em análise de dados.

Um exemplo típico de operações envolvendo um objeto do tipo `dataframe` seria: leitura de arquivo, seleção de colunas, filtragem de registros, junção com outros dados e sumarização.

Incialmente, faremos algo um pouco mais simples: leitura dos dados de um arquivo, seleção de algumas colunas de interesse e descrição estatística dos elementos das colunas selecionadas.

Para codificar a sequência de operações descrita, costuma-se mesclar os paradigmas (tradicional e POO) da seguinte maneira:

```
import pandas as pd

grupos = pd.read_excel("./dados/ISP2020.xlsx", sheet_name="GRUPOS")

grupos_selecionados = grupos.loc[:, ["SEGURADOS ATIVOS",
                                     "APOSENTADOS", "PENSIONISTAS"]]

grupos_selecionados.describe()
```

Podemos reescrever o código, explicitando o encadeamento de operações de modo a torná-lo mais organizado, facilitando sua leitura e compreensão, da seguinte forma:

```
import pandas as pd

grupos = pd.read_excel("./dados/ISP2020.xlsx", sheet_name="GRUPOS")

(grupos
    .loc[:, ["SEGURADOS ATIVOS", "APOSENTADOS", "PENSIONISTAS"]]
    .describe())
```

	SEGURADOS ATIVOS	APOSENTADOS	PENSIONISTAS
count	2154.000000	2154.000000	2154.000000
mean	2134.497679	1195.924791	267.805478
std	12556.498730	11320.704335	2772.251472
min	0.000000	0.000000	0.000000
25%	228.250000	43.000000	9.000000
50%	487.000000	104.500000	23.000000
75%	1092.750000	260.000000	60.000000
max	421955.000000	313091.000000	91285.000000

Observe que o fluxo de operações agora pode ser lido de forma mais clara e natural: partindo de um conjunto de dados tabular (`grupos`), seleciona-se um subconjunto de colunas (`.loc[]`) e obtém-se estatísticas que descrevem os dados das colunas selecionadas (`.describe()`).

Caso estivéssemos utilizando a biblioteca {polars} ao invés de {pandas}, o código seria praticamente idêntico:

```
import polars as pl

grupos = pl.read_excel("./dados/ISP2020.xlsx", sheet_name="GRUPOS")

(grupos
    .select(["SEGURADOS ATIVOS", "APOSENTADOS", "PENSIONISTAS"])
    .describe())
```

statistic str	SEGURADOS ATIVOS f64	APOSENTADOS f64	PENSIONISTAS f64
”count”	2154.0	2154.0	2154.0
”null_count”	0.0	0.0	0.0

statistic str	SEGURADOS ATIVOS f64	APOSENTADOS f64	PENSIONISTAS f64
”mean”	2134.497679	1195.924791	267.805478
”std”	12556.49873	11320.704335	2772.251472
”min”	0.0	0.0	0.0
”25%”	228.0	43.0	9.0
”50%”	487.0	105.0	23.0
”75%”	1093.0	260.0	60.0
”max”	421955.0	313091.0	91285.0