



Transformando dados em ouro: A arte de manipular e
limpar dados com Python em ciência de dados

Autor: Jefferson Luiz da Costa Xaxá

Hello world!

O presente artigo foi criado com a finalidade de explicitar, de forma simples, a manipulação e limpeza de dados utilizando a linguagem Python na Ciência de Dados, para que o leitor possa melhorar seu entendimento a respeito desse assunto.

Sumário

Introdução	4
Contextualização	4
Objetivo do artigo	4
Entendendo a Manipulação e Limpeza de Dados	4
Definição	5
A importância da manipulação e limpeza para a análise de dados	5
O benefício de ter os dados limpos e bem manipulados	5
Ferramentas e Bibliotecas para Manipulação e Limpeza de Dados em Python	6
Visão geral das principais bibliotecas para manipulação de dados	6
Demonstração de algumas funcionalidades importantes dessas bibliotecas	6
Etapas de Manipulação e Limpeza de Dados com Python.....	8
Coleta e aquisição de dados.....	8
Análise preliminar dos dados	10
Identificação e tratamento de dados ausentes ou faltantes	11
Remoção de duplicatas	13
Transformação de dados para formatos adequados	13
Lidando com outliers e anomalias em dados.....	14
Tratamento de dados sensíveis e privacidade	15
Boas práticas de manipulação e limpeza de dados.....	16
Considerações finais.....	17
Referencial teórico	17

Introdução

Contextualização

Não é novidade que o conhecimento revoluciona o mundo a cada ano, tornando-se ouro na era da informação (Século XXI). Para obter conhecimento é necessária uma fonte de informação que possua dados relevantes a respeito de determinado assunto. Pois bem, tais dados não podem conter “Fake News” e muito menos erros. Neste ponto, nos deparamos com a manipulação e limpeza dos dados.

Surgindo dentro do contexto de ciência de dados, a manipulação e a limpeza de dados ganham visibilidade nos anos 2000, por decorrência da junção de conhecimentos como matemática, estatística e programação, tornando-se uma das principais etapas do processo de análise de dados, uma vez que a integridade e qualidade são fundamentais para se obter resultados precisos e confiáveis.

Entre alguns dos problemas a serem tratados em uma base de dados, podemos citar: Erros de digitação, outliers, inconsistências, valores ausentes, valores duplicados etc. Para os tratamentos desses problemas, os comunidade de dados (Engenheiro de Dados, Cientistas de Dados, Analistas etc.) utilizam-se de ferramentas como a linguagem de programação Python para manipular e limpar esses dados.

No geral, essa é uma jornada de contínua evolução de ferramentas, e métodos de tratamento de dados para atender a demandas do mercado e da sociedade.

Objetivo do artigo

Aprimorar o conhecimento do leitor a respeito da manipulação e limpeza de dados com Python.

Entendendo a Manipulação e Limpeza de Dados

Definição

A manipulação de dados refere-se à alteração, transformação, e operações com dados em um conjunto ou base de dados. Por outro lado, a limpeza diz respeito à correção de erros como valores ausentes, digitação, inconsistências, outliers etc.

Em resumo, ao fazer algum trabalho com dados realizaremos uma manipulação, já a limpeza é toda correção de problemas.

A importância da manipulação e limpeza para a análise de dados

Essa etapa é tão importante que a maioria dos devs que trabalham com dados gastam a maior parte do tempo no seu tratamento.

Caso hipotético: Você é um cientista de dados em uma empresa de pequeno porte, onde sua análise contribui para a tomada de decisão. Em uma dessas análises foi lançado o desafio apresentar ao cliente a melhor opção, em relação ao custo-benefício, para a escolha de um fornecedor. Você está apressado e pula a etapa de manipulação e limpeza dos dados. No momento de sua apresentação, verifica que os gráficos apresentaram erros como “NULL”, “None” ou valores discrepantes (outliers) e que mesmo assim foi possível extrair insights para aquele cliente. Meses depois, o cliente percebe que seu retorno financeiro está abaixo do que esperava e percebe a má decisão que tomou baseada em uma análise ruim realizada por você.

O caso hipotético acima demonstra uma das consequências que a ausência dessa etapa tão importante pode causar a uma empresa, projeto ou a um cliente. A manipulação e limpeza de dados são suas aliadas na análise de dados e fundamental que represente a realidade de forma significativa e consistente. Evitando, dessa forma, prejuízos e garantindo feedback que seu cliente deseja ter, afinal, é o que se espera ao contratar seu serviço.

O benefício de ter os dados limpos e bem manipulados

Quando o tratamento dos dados é muito bem-feito, seus gráficos não apresentam erros, suas análises ganham relevância dentro de um projeto e gera lucro e satisfação por parte do contratante. Afinal, ninguém gosta de pagar e não receber o que foi combinado.

Ferramentas e Bibliotecas para Manipulação e Limpeza de Dados em Python

Visão geral das principais bibliotecas para manipulação de dados

Em Python, utilizamos as bibliotecas para facilitar o nosso trabalho e reduzir a quantidade de códigos em um programa ou aplicação. Bibliotecas nada mais são que pacotes ou módulos que possuem classes e métodos, criados pela comunidade dev, para realizar tarefas específicas em projetos no Python.

Relacionado à manipulação e limpeza de dados, as bibliotecas mais conhecidas são:

- Pandas: Pacote utilizado para a manipulação e análise de dados, sendo convencionalizado, o seu chamamento, por “pd”;
- Numpy: Pacote utilizado para realizar manipulação de números, sendo também convencionalizado, o seu chamamento, por “np”.

Demonstração de algumas funcionalidades importantes dessas bibliotecas

Podemos importar a biblioteca Pandas por meio do comando “pip install pandas” dentro da IDE utilizada pelo usuário (seja o Visual Studio Code, Jupyter Notebook, PyCharm etc.) e atribuindo, ao final, seu apelido “pd”, ficando então da seguinte forma:

```
# Importando o Pandas
import pandas as pd
```

Código 1

Com o Pandas importado podemos carregar um dataframe e atribuí-lo à variável “df” (abreviação de dataframe). É possível ler diversos formatos de dataframes, dentre alguns podemos exemplificar:

```
# Exemplo 1
df = pd.read_excel("Caminho do dataframe + nome + o formato do arquivo")

# Exemplo 2
df = pd.read_csv("Caminho do dataframe + nome + o formato do arquivo")

# Exemplo 3
df = pd.read_json("Caminho do dataframe + nome + o formato do arquivo")

# Exemplo 4
df = pd.read_html("Caminho do dataframe + nome + o formato do arquivo")

# Exemplo 5
df = pd.read_xml("Caminho do dataframe + nome + o formato do arquivo")

# Exemplo 6
df = pd.read_sql("Caminho do dataframe + nome + o formato do arquivo")
```

Código 2

Utilizando um dicionário, por exemplo, podemos criar um dataframe com o seguinte comando:

```
# Transformando um dicionário em um dataframe
df = pd.DataFrame('Caminho do dataframe + o formato do arquivo')
```

Código 3

Para exibir seu dataframe na tela podemos utilizar o comando a seguir:

```
# Exibindo o dataframe
print(df)
```

Código 4

Podemos ainda, importar a biblioteca NumPy por meio do comando “pip install numpy” dentro da IDE utilizada pelo usuário e adicionar, ao final, o seu apelido “np”, ficando então da seguinte forma:

```
# Importando a biblioteca NumPy
pip install numpy as np
```

Código 5

Com o NumPy podemos trabalhar com arrays e datatypes, entre outros, por exemplo, dentro de um dataframe. Contudo, neste artigo, daremos foco à biblioteca Pandas.

Passado por esta etapa, seria realizado uma exploração dos dados para conhecer o dataframe que seria manipulados os dados.

Etapas de Manipulação e Limpeza de Dados com Python

Com base no que já foi explicitado e considerando que o Pandas e o NumPy já tenham sido importados, vamos trabalhar com um dicionário chamado “Dados”, que é uma base de cadastro de clientes fictícios, para melhor desenvolver o entendimento.

Coleta e aquisição de dados

Para a aquisição de um dataframe foi importado a biblioteca Pandas. Em seguida criamos um dicionário para ser chamado pela função `pd.DataFrame()` do Pandas (além de chamar o dicionário, o converte de "chave" : "valor" para um dataframe de linhas e colunas, atribui "dados" a variável `df` e exibe o dataframe ao final com a função `print()`):

```
# Importando o Pandas
import pandas as pd

# Estruturando o dicionário
dados = {
    "Nome": [
```



```

        "João",
        "Maria",
        "José",
        "Ana",
        "Maria"
    ],

    "Idade": [
        '25',
        21,
        30,
        27,
        21
    ],

    "Sexo": [
        "Masc",
        "Fem",
        None,
        "Fem",
        "Fem"
    ],

    "Altura": [
        1.75,
        1.90,
        1.80,
        16.0,
        1.90
    ],

    "Peso": [
        75,
        67,
        90,
        60,
        67
    ]
}

# Atribuindo o dataframe do dicionário para a variável df
df = pd.DataFrame(dados)

print(df)

```

Código 6

O dicionário seria exibido da seguinte forma:

```
{'Nome': ['João', 'Maria', 'José', 'Ana', 'Maria'], 'Idade': [25, 21, 30, 27, 21], 'Sexo': ['Masc', 'Fem', None, 'Fem', 'Fem'], 'Altura': [1.75, 1.9, 1.8, 16.0, 1.9], 'Peso': [75, 67, 90, 60, 67]}
```

Imagem 1

Contudo, transformamos o dicionário em um dataframe, e o resultado do código é o seguinte:

	Nome	Idade	Sexo	Altura	Peso
0	João	25	Masc	1.75	75
1	Maria	21	Fem	1.90	67
2	José	30	None	1.80	90
3	Ana	27	Fem	16.00	60
4	Maria	21	Fem	1.90	67

Imagem 2

Este é um pequeno exemplo de manipulação de dados com a biblioteca Pandas do Python.

Análise preliminar dos dados

Vamos verificar as colunas que esse dataset possui e o tipo de dado que cada uma possui por meio da função `info()`:

```
# Exibindo informações do dataframe  
df.info()
```

Código 7

Essa função apresenta o seguinte retorno:

```

RangeIndex: 5 entries, 0 to 4
Data columns (total 5 columns):
#   Column   Non-Null Count  Dtype
---  -
0   Nome     5 non-null      object
1   Idade    5 non-null      object
2   Sexo     4 non-null      object
3   Altura  5 non-null      float64
4   Peso     5 non-null      int64
dtypes: float64(1), int64(1), object(3)
memory usage: 328.0+ bytes

```

Imagem 3

É possível verificar que o dataframe possui 5 colunas (Nome, Idade, Sexo, Altura e Peso), e que respectivamente os tipos de dados são object, object, object, float64 e int64 (onde object são dados de string, float64 são dados com números decimais e int64 são número inteiros). Ainda, podemos verificar que cada uma das colunas possui 5 valores, exceto a de “Sexo”, o que quer dizer que há valores ausentes nessa coluna, pois possui apenas 4 valores não nulos.

Identificação e tratamento de dados ausentes ou faltantes

Para verificar se há dados ausentes podemos usar também o seguinte comando:

```

# Atribuindo os valores ausentes à variável valores_ausentes
valores_ausentes = df.isnull()

# Exibindo o total de valores_ausentes
print(valores_ausentes.sum())

```

Código 8

A primeira linha de código atribui os valores ausentes à variável "valores_ausentes". Já na segunda linha, é pedido para exibir a soma total de valores ausentes por coluna, obtendo o seguinte resultado:

```
Nome      0
Idade     0
Sexo      1
Altura    0
Peso      0
dtype: int64
```

Imagem 4

Perceba que há um valor ausente na coluna "Sexo", o que só confirma a conclusão que tiramos da imagem 3. Para tratar esse erro podemos remover a linha que possui o erro, aplicar métodos estatísticos ou consultar o setor da empresa que nos forneceu os dados para que possamos alterá-lo de forma correta.

Pudemos verifica na imagem 2 que se trata de um valor ausente da informação sobre o sexo de José. Para alterá-la para "Masc" (masculino) podemos acessar a linha e a coluna do valor ausente e modificá-lo individualmente da seguinte forma:

```
# Localizando o valor nulo na linha 2 e coluna 2 e atribuindo o valor
df.iloc[2, 2] = "Masc"

# Exibindo o dataframe
print(df)
```

Código 9

O retorno no dataset seria o seguinte:

```
   Nome  Idade  Sexo  Altura  Peso
0  João    25  Masc    1.75    75
1  Maria   21  Fem    1.90    67
2  José    30  Masc    1.80    90
3   Ana    27  Fem    1.60    60
4  Maria   21  Fem    1.90    67
```

Imagem 5

Remoção de duplicatas

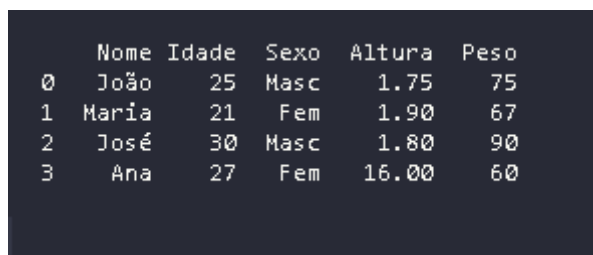
Na imagem 5 vemos claramente que “Maria” é um valor duplicado. Para realizar a remoção de duplicatas no Python, podemos utilizar o comando:

```
# Remove duplicatas e, um dataframe
df = df.drop_duplicates()

# Exibindo o dataframe
print(df)
```

Código 10

Essas linhas de código retornam o seguinte dataframe sem duplicatas:



	Nome	Idade	Sexo	Altura	Peso
0	João	25	Masc	1.75	75
1	Maria	21	Fem	1.90	67
2	José	30	Masc	1.80	90
3	Ana	27	Fem	16.00	60

Imagem 6

A partir desse momento, o dataset não possui mais 5 valores, conforme imagem 3, e sim 4, pois um valor duplicado foi removido.

Transformação de dados para formatos adequados

Na imagem 3, percebe-se que a coluna de idade, que é de valores inteiros, está como object (ou sejam string). Isso acontece porque no código 6 a idade de João está entre aspas, o que torna o objeto em string. Portanto, deveria ter o formato int32 ou int64.

Para transformarmos o tipo de objeto para inteiro, codamos o seguinte:

```
# Transformando string em inteiro
df["Idade"] = df["Idade"].astype(int)
```

```
# Exibindo informações do dataframe
df.info()
```

Código 11

Assim, podemos obter o resultado:

```
Int64Index: 4 entries, 0 to 3
Data columns (total 5 columns):
#   Column  Non-Null Count  Dtype
---  -
0   Nome    4 non-null      object
1   Idade    4 non-null      int32
2   Sexo     4 non-null      object
3   Altura  4 non-null      float64
4   Peso     4 non-null      int64
dtypes: float64(1), int32(1), int64(1), object(2)
```

Imagem 7

Nesse momento, podemos confirmar que idade agora é do tipo inteiro. Ainda, vemos o que foi informado no tópico anterior (remoção de duplicatas) que o dataframe só possui 4 valores.

Lidando com outliers e anomalias em dados

Outlier é qualquer coisa que fuja da normalidade, em outras palavras, é o que é drasticamente diferenciado.

Ao analisar dados, muitos podem ser os fatores que criaram um outlier, por exemplo: Erro de digitação, a o dado pode ter sido colhido com erro, pode ter havido má fé de quem forneceu os dados, pode também ocorrer que seja lago verídico e mesmo assim não deixa de ser algo fora da curva.

Para tratar outliers primeiro precisamos identificá-los, com isso, dependendo do caso concreto, podemos tratá-lo removendo-o da base de dados, transformação os dados, tratando os outliers como uma categoria separada ou até mesmo utilizar métodos robustos como estatística. “Cada caso é um caso”.

Em nosso dataframe possuímos um outlier. Se você prestar atenção na coluna de altura da imagem 6, perceberá que a medida está em metros, e que

Ana possui 16 metros, o que é impossível, pois a pessoa mais alta do mundo possui 2,5 metros de altura (G1, 2023).

Para tratar esse outlier, iremos considerá-lo como um erro de digitação, poderíamos utilizar também métodos estatísticos, como a média das idades, logaritmos ou função quadrática.

Para tal solução, codamos o seguinte:

```
# Tratando outliers da linha 3 e coluna 3, de 16.0m para 1.60m
df.iloc[3, 3] = 1.60
```

Código 12

O nosso dataset agora está correto da seguinte forma:

	Nome	Idade	Sexo	Altura	Peso
0	João	25	Masc	1.75	75
1	Maria	21	Fem	1.90	67
2	José	30	Masc	1.80	90
3	Ana	27	Fem	1.60	60

Imagem 8

Nesse momento, nosso dataframe está totalmente livre de erros. Finalizando assim, a etapa de manipulação e limpeza do dataset “dados”.

Como a nossa base de dados é pequena, percebemos facilmente alguns erros ao printar o dataframe e conseguir visualizá-lo por completo. Contudo, em uma base maior, como o Big Data, não seria tão fácil identificar os erros, necessitando então de uma análise exploratória de dados mais aprofundada, o que demandaria tempo e conhecimento.

Tratamento de dados sensíveis e privacidade

Quando se trata de dados sensíveis, a primeira coisa que vem à mente é a segurança da informação. Pois bem, para tratar brevemente deste assunto, citaremos a LGPD – Lei Geral de Proteção de Dados Pessoais (lei 13.709 / 2018).

A LGPD é uma lei que dispõe sobre o tratamento de dados pessoais, incluindo-se nesse rol os dados do meio digital, dados públicos ou privados, de pessoa física ou jurídica.

O profissional de dados deve ser uma pessoa responsável para ter acesso e poder manipular e limpar tais dados. Podendo vir a ser responsabilizado penal e civilmente por atos ilícitos cometidos diante dos dados de outrem.

Caso queira se aprofundar no assunto, no referencial teórico possui o link direto para a página da LGPD no site do Planalto.

Boas práticas de manipulação e limpeza de dados

Tratando-se de Python, a PEP 8 convencionou as boas práticas do uso dessa linguagem de programação. Contudo, não trata da manipulação e limpeza de dados. Porém, há do que se falar quando se trata de tal assunto.

Vamos abordar dois assuntos relacionados à manipulação e limpeza de dados e suas boas práticas brevemente.

- Documentação:

A documentação de uma biblioteca em Python, seja o Pandas, o NumPy etc., é passagem obrigatória para todos analista, cientista e engenheiro de dados que precise aprender, relembrar ou tirar dúvidas sobre como utilizar um pacote. Essa atitude te fará ganhar tempo no desenvolvimento de sua solução de negócio e até mesmo deixá-lo mais apto para tratamentos mais profundos nos dados.

- Registro de transformações aplicadas aos dados:

O registro nada mais é do que comentar o que está sendo alterado ou transformado em uma base de dados, aplicação ou algoritmo.

Em nosso código é possível verificar que acima de cada linha há um caractere “#” que determina que tudo o que vier após ele, na mesma linha, será apenas um comentário. Além da hashtag, podemos comentar também por meio de aspas simples (‘ ’), aspas duplas (“ ”) e aspas triplas (‘’ ’’). Neste último caso, o utilizamos para comentar um bloco ou conjunto de linhas que estiver dentro de seu corpo.

É uma boa prática realizar comentários, pois nem sempre será apenas quem os criou a utilizá-los, permitindo assim que outras pessoas que tenham acesso a ele, possam saber o que cada linha faz sem precisar perder tempo tentando entendê-la.

Considerações finais

Por meio desse artigo, pudemos pincelar o assunto no tocante à manipulação e limpeza de dados com Python para ciência de dados, e assim entender o porquê de transformar dados em ouro, pois sua utilização correta permite aos gestores tomarem decisões corretas baseadas em dados, ações como essa são conhecidamente como Data Driven.

A arte de manipular e limpar dados com Python em ciência de dados é para todos que estejam dispostos a aprender cada dia mais a gerar riqueza com os dados.

Referencial teórico

<https://ilumeo.com.br/todos-posts/2021/08/17/um-pouco-da-historia-da-ciencia-de-dados>

<https://pandas.pydata.org/docs/>

<https://www.jstatsoft.org/article/view/v059i10>

https://www.hashtagtreinamentos.com/bibliotecas-mais-importantes-do-python?gad=1&gclid=Cj0KCQjw1_SkBhDwARIsANbGpFsV7-5lsgmt1jv6W5nxmUZ6AcCiWg1h4mp9grEqZq5-MqWVt80o06AaAgifEALw_wcB

https://pandas.pydata.org/docs/user_guide/indexing.html

<https://numpy.org/doc/1.25/user/index.html>

<https://www.aquare.la/o-que-sao-outliers-e-como-trata-los-em-uma-analise-de-dados/>

<https://g1.globo.com/mundo/noticia/2023/01/04/o-homem-que-nao-para-de-crescer-e-sonha-em-se-tornar-o-mais-alto-do-mundo.ghtml>

https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/L13709compilado.htm

<https://peps.python.org/pep-0008/>

Repositório completo do artigo:

https://github.com/JeffersonLCXaxa/Data_Manipulation_and_Cleaning_with_Python_for_Simple_Algorithm_Data_Science