Data Quality, Data Clean e Testes

Tratar dados exige um forte cuidado, visto que decisões são diretamente orientadas ao que se entende nos diversos indicadores, relatórios e painéis (dashboards). Por exemplo, quando um motorista observa que o indicador de combustível está abaixo da metade, pode se planejar em abastecer seu veículo… entretanto, caso o alerta de combustível baixo (reserva) acenda, ele deve se preocupar em abastecer no veículo mais próximo. Ou quando uma médica observa os resultados dos exames de um paciente, pode indicar o tratamento adequado para a cura. Semelhantemente, diretores de uma companhia, ao observar os painéis e relatórios, podem tomar decisões de expansão ou de economia de recursos visando o propósito da corporação.

Neste cenário, os especialistas em dados preocupam-se com a qualidade dos dados e, para tanto, técnicas de limpeza, validação e testes são aplicadas. Ao longo deste tópico, veremos algumas das principais técnicas para que os dados possam, refletindo a realidade, orientar o processo de tomada de decisão.

## Modelagem de tabelas especializadas com dados agrupados e/ou sumarizados

Modelagem de tabelas especializadas é uma técnica utilizada para organizar e estruturar os dados de maneira apropriada para uma análise específica. A técnica envolve a criação de tabelas com dados agrupados e/ou sumarizados, que fornecem uma visão mais clara e compreensível dos dados. Nesta aula, vamos aprender como criar tabelas especializadas com Python, usando a biblioteca Pandas.

## Introdução ao Pandas

Pandas é uma biblioteca Python de código aberto para análise de dados. Ela fornece uma estrutura de dados flexível e de alto desempenho para manipular e analisar dados em Python. O Pandas é uma ferramenta útil para criar tabelas especializadas e analisar grandes conjuntos de dados.

Para começar a trabalhar com o Pandas, precisamos importar a biblioteca em nosso script. Podemos usar a seguinte linha de código para fazer isso:

import pandas as pd

### Criando um DataFrame

O Pandas fornece um objeto chamado DataFrame, que é uma estrutura de dados tabular bidimensional com rótulos em linha e coluna. Um DataFrame é semelhante a uma tabela de banco de dados ou a uma planilha do Excel. Podemos criar um DataFrame a partir de um dicionário de Python ou de um arquivo CSV.

### Criando um DataFrame a partir de um dicionário

Vamos criar um DataFrame a partir de um dicionário Python. Suponha que temos dados sobre as vendas de três produtos em três regiões diferentes:

data = {

'Produto': ['Camisa', 'Calça', 'Sapato', 'Camisa', 'Calça', 'Sapato', 'Camisa', 'Calça', 'Sapato'],

'Região': ['Norte', 'Norte', 'Norte', 'Sul', 'Sul', 'Sul', 'Leste', 'Leste', 'Leste'],

'Vendas': [100, 200, 150, 120, 180, 200, 150, 250, 170]

}

df = pd.DataFrame(data)

print(df)

Este código irá criar um DataFrame com os dados agrupados por produto e região, com as vendas sumarizadas para cada combinação de produto e região. O resultado será uma tabela especializada que podemos usar para análises e decisões precisas.

### Criando um DataFrame a partir de um arquivo CSV

Também podemos criar um DataFrame a partir de um arquivo CSV. Suponha que temos um arquivo CSV chamado "vendas.csv" que contém os dados de vendas dos produtos:

Produto,Região,Vendas

Camisa,Norte,100

Calça,Norte,200

Sapato,Norte,150

Camisa,Sul,120

Calça,Sul,180

Sapato,Sul,200

Camisa,Leste,150

Calça,Leste,250

Sapato,Leste,170

Podemos ler o arquivo CSV e criar um DataFrame com os seguintes comandos:

df = pd.read\_csv('vendas.csv')

print(df)

Este código irá criar um DataFrame com os dados lidos do arquivo CSV. O resultado será o mesmo DataFrame que criamos a partir do dicionário.

## Agrupando e sumarizando dados

Agora que temos um DataFrame com os dados, podemos usar o Pandas para agrupar e sumarizar esses dados de maneira especializada. Vamos ver como isso pode ser feito.

### Agrupando dados por coluna

Podemos agrupar os dados por uma coluna específica usando o método groupby() do Pandas. Vamos agrupar os dados do DataFrame por produto e somar as vendas para cada produto. Para isso, podemos usar o seguinte código:

df\_produto = df.groupby(['Produto'])['Vendas'].sum()

print(df\_produto)

Este código irá criar uma tabela especializada com os dados agrupados por produto e com as vendas somadas para cada produto.

### Agrupando dados por colunas múltiplas

Também podemos agrupar os dados por múltiplas colunas usando o método groupby(). Vamos agrupar os dados do DataFrame por produto e região, e somar as vendas para cada combinação de produto e região. Para isso, podemos usar o seguinte código:

df\_produto\_regiao = df.groupby(['Produto', 'Região'])['Vendas'].sum()

print(df\_produto\_regiao)

Este código irá criar uma tabela especializada com os dados agrupados por produto e região, com as vendas somadas para cada combinação de produto e região.

### Filtrando dados com base em critérios

Podemos filtrar os dados do DataFrame com base em critérios específicos usando o método loc[] do Pandas. Vamos filtrar os dados do DataFrame para exibir apenas as vendas do produto "Camisa". Para isso, podemos usar o seguinte código:

df\_camisa = df.loc[df['Produto'] == 'Camisa']

print(df\_camisa)

Este código irá criar uma tabela especializada com os dados filtrados para exibir apenas as vendas do produto "Camisa".

### Exportando dados para um arquivo CSV

Podemos exportar os dados do DataFrame para um arquivo CSV usando o método to\_csv() do Pandas. Vamos exportar os dados do DataFrame para um arquivo CSV chamado "vendas\_produto\_regiao.csv". Para isso, podemos usar o seguinte código:

df\_produto\_regiao.to\_csv('vendas\_produto\_regiao.csv')

Este código irá exportar os dados do DataFrame para um arquivo CSV com os dados agrupados por produto e região, com as vendas sumarizadas para cada combinação de produto e região.

## Princípios de qualidade de dados e seus benefícios

A qualidade dos dados é fundamental para a tomada de decisão eficaz. Quando os dados são precisos, consistentes, confiáveis e relevantes, eles podem ser usados para orientar as decisões empresariais e melhorar o desempenho. Por outro lado, dados de baixa qualidade podem levar a decisões equivocadas e prejudicar o desempenho da empresa. Agora, discutiremos os princípios da qualidade de dados e seus benefícios.

### Princípios da qualidade de dados

Os princípios da qualidade de dados podem ser divididos em quatro categorias: precisão, consistência, confiabilidade e relevância.

* Precisão: Os dados precisos são dados que estão corretos e livres de erros. Eles são baseados em fontes confiáveis e são atualizados regularmente. A precisão dos dados pode ser afetada por erros de entrada manual, integração de dados de várias fontes, omissão de dados e duplicação de dados. Para garantir a precisão dos dados, é importante realizar verificações regulares dos dados para identificar e corrigir erros.
* Consistência: Os dados consistentes são dados que estão no mesmo formato e mantêm as mesmas definições em todos os sistemas e processos. A consistência dos dados pode ser afetada pela falta de padronização dos dados, pela variação dos nomes de campos e pela falta de integridade referencial. Para garantir a consistência dos dados, é importante estabelecer padrões para a entrada de dados e verificar regularmente se esses padrões estão sendo seguidos.
* Confiabilidade: Os dados confiáveis são dados que são precisos e consistentes, mas também são acessíveis e seguros. Eles são protegidos contra perda, roubo e corrupção, e podem ser acessados por usuários autorizados. A confiabilidade dos dados pode ser afetada pela falta de segurança de dados, pela falta de backups regulares e pela falta de controle de acesso aos dados. Para garantir a confiabilidade dos dados, é importante implementar práticas de segurança de dados, como backups regulares e controles de acesso.
* Relevância: Os dados relevantes são dados que são necessários para as decisões empresariais. Eles são coletados com base nos requisitos do negócio e são relevantes para os usuários finais. A relevância dos dados pode ser afetada pela coleta de dados desnecessários ou pela falta de coleta de dados importantes. Para garantir a relevância dos dados, é importante estabelecer um processo para identificar os requisitos de dados do negócio e coletar apenas os dados necessários.

### Benefícios da qualidade de dados

A qualidade dos dados tem vários benefícios para a tomada de decisão empresarial. Alguns desses benefícios incluem:

* Tomada de decisão mais informada: Dados precisos, consistentes, confiáveis e relevantes permitem que as empresas tomem decisões informadas com base em informações precisas e confiáveis.
* Redução de custos: Dados de baixa qualidade podem levar a decisões equivocadas, o que pode levar a custos adicionais para a empresa. Ao melhorar a qualidade dos dados, as empresas podem reduzir os custos desnecessários associados à tomada de decisão equivocada.
* Melhoria da eficiência: Dados precisos e consistentes podem ajudar a melhorar a eficiência dos processos empresariais, reduzindo o tempo gasto na correção de erros e retrabalho.
* Aumento da satisfação do cliente: Dados precisos e relevantes permitem que as empresas entendam melhor as necessidades de seus clientes e ofereçam soluções personalizadas para atender a essas necessidades. Isso pode levar a um aumento na satisfação do cliente e na fidelidade à marca.

## Dados sujos

Dados sujos (ou "dirty data", em inglês) são dados que contêm erros, duplicatas, valores ausentes, inconsistências e outros problemas que dificultam a análise e a tomada de decisões. Os dados sujos podem ser provenientes de fontes diversas, como formulários preenchidos manualmente, sistemas de coleta de dados mal projetados, erros de digitação, entre outros.

### Como um dado sujo pode atrapalhar uma boa decisão?

Um dado sujo pode levar a decisões ruins ou imprecisas porque ele pode:

* Levar a conclusões erradas: dados inconsistentes ou incorretos podem levar a conclusões erradas sobre os dados, o que pode levar a decisões ineficazes ou prejudiciais.
* Aumentar o tempo de análise: dados sujos podem exigir mais tempo e esforço para serem limpos e preparados para análise, o que pode atrasar a tomada de decisões importantes.
* Reduzir a qualidade da análise: dados sujos podem dificultar a análise e a visualização dos dados, reduzindo a qualidade da análise.

## Qualidade de dados na prática em Python

### Verificação de dados precisos

Para verificar a precisão dos dados, podemos usar a biblioteca pandas para realizar algumas verificações. Vamos supor que temos um conjunto de dados contendo informações de vendas e queremos verificar se o valor total de vendas é consistente com a soma das vendas individuais. Podemos fazer isso da seguinte maneira:

import pandas as pd

# Ler os dados de um arquivo CSV

df = pd.read\_csv('dados\_vendas.csv')

# Verificar a soma das vendas individuais

soma\_vendas = df['Vendas'].sum()

# Verificar o valor total de vendas

valor\_total\_vendas = df['Valor Total'].sum()

# Verificar se a soma das vendas individuais é igual ao valor total de vendas

if soma\_vendas == valor\_total\_vendas:

print('Os dados de vendas são precisos')

else:

print('Os dados de vendas são imprecisos')

### Padronização de dados

Para garantir a consistência dos dados, é importante padronizar os dados para que eles estejam no mesmo formato em todos os sistemas e processos. Vamos supor que temos um conjunto de dados contendo informações de clientes e queremos padronizar o formato dos nomes dos clientes para que todos estejam em maiúsculas. Podemos fazer isso da seguinte maneira:

import pandas as pd

# Ler os dados de um arquivo CSV

df = pd.read\_csv('dados\_clientes.csv')

# Padronizar o formato dos nomes dos clientes para que todos estejam em maiúsculas

df['Nome'] = df['Nome'].str.upper()

# Salvar os dados padronizados em um novo arquivo CSV

df.to\_csv('dados\_clientes\_padronizados.csv', index=False)

### Verificação de segurança de dados

Para verificar a segurança dos dados, podemos usar a biblioteca os para verificar se um arquivo existe antes de abri-lo. Vamos supor que temos um conjunto de dados contendo informações financeiras e queremos verificar se o arquivo que contém essas informações existe antes de abri-lo. Podemos fazer isso da seguinte maneira:

import os

# Verificar se o arquivo existe

if os.path.exists('dados\_financeiros.csv'):

# Ler os dados do arquivo

with open('dados\_financeiros.csv', 'r') as f:

dados = f.read()

# Realizar as análises de dados aqui

else:

print('O arquivo não existe ou não é acessível')

## Garantindo a completude do dado

Garantir a completude do dado é uma etapa importante na análise de dados, pois dados incompletos podem prejudicar a precisão e a confiabilidade das análises.

Antes de começar, vamos criar um conjunto de dados para usarmos em nossos exemplos. Suponha que temos um conjunto de dados de funcionários que inclui informações sobre o nome, idade, salário e departamento. O conjunto de dados contém alguns valores nulos, para ilustrar os desafios de lidar com dados incompletos. Podemos criar esse conjunto de dados da seguinte maneira:

import pandas as pd

import numpy as np

data = {

'nome': ['João', 'Maria', 'José', 'Pedro', np.nan],

'idade': [25, 30, np.nan, 40, 35],

'salario': [5000, 6000, 4000, np.nan, 4500],

'departamento': ['RH', 'TI', 'Vendas', np.nan, 'Marketing']

}

df = pd.DataFrame(data)

print(df)

Isso nos dará a seguinte saída:

nome idade salario departamento

0 João 25.0 5000.0 RH

1 Maria 30.0 6000.0 TI

2 José NaN 4000.0 Vendas

3 Pedro 40.0 NaN Marketing

4 NaN 35.0 4500.0 NaN

Agora, vamos discutir algumas técnicas para lidar com dados incompletos.

### Eliminação de valores nulos

A técnica mais simples para lidar com dados incompletos é eliminar as linhas ou colunas que contêm valores nulos. No entanto, essa técnica pode levar à perda de informações e não é recomendada em muitos casos. Vamos ver como isso é feito em Python:

# eliminar linhas com valores nulos

df.dropna(inplace=True)

print(df)

# eliminar colunas com valores nulos

df.dropna(axis=1, inplace=True)

print(df)

A primeira linha elimina todas as linhas que contêm valores nulos, enquanto a segunda linha elimina todas as colunas que contêm valores nulos. O parâmetro inplace=True é usado para modificar o próprio DataFrame em vez de criar um novo.

### Preenchimento com valores fixos

Outra abordagem para lidar com valores nulos é preencher os valores com um valor fixo. Isso pode ser útil em alguns casos, por exemplo, quando a média ou a mediana dos valores é um bom substituto para os valores nulos. Vamos ver como isso é feito em Python:

# preencher valores nulos com a média da coluna

df.fillna(df.mean(), inplace=True)

print(df)

# preencher valores nulos com um valor fixo (por exemplo, 0)

df.fillna(0, inplace=True)

print(df)

Na primeira linha, preenchemos os valores nulos de cada coluna com a média dos valores da coluna. Na segunda linha, preenchemos os valores nulos com 0.

### Interpolação

A interpolação é outra técnica para lidar com valores nulos, que envolve o preenchimento dos valores faltantes com base nos valores existentes. Por exemplo, podemos usar a interpolação linear para preencher os valores faltantes em uma série temporal. Vamos ver como isso é feito em Python:

# preencher valores nulos usando interpolação linear

df.interpolate(method='linear', inplace=True)

print(df)

A interpolação linear calcula os valores faltantes com base em uma linha reta entre os valores existentes mais próximos. Existem outras opções de método de interpolação que podem ser usados, como 'polynomial' e 'spline'.

### Imputação com modelos de aprendizado de máquina

Outra abordagem para lidar com valores nulos é usar modelos de aprendizado de máquina para imputar os valores faltantes. Essa abordagem pode ser mais precisa do que preencher com valores fixos ou interpolação, pois leva em consideração a relação entre as variáveis. Vamos ver como isso é feito em Python:

from sklearn.impute import KNNImputer

# criar um imputador KNN com 3 vizinhos

imputer = KNNImputer(n\_neighbors=3)

# imputar os valores faltantes com base nos valores existentes

df\_imputed = pd.DataFrame(imputer.fit\_transform(df), columns=df.columns)

print(df\_imputed)

O KNNImputer é um modelo que imputa os valores faltantes com base nos valores existentes e na proximidade dos vizinhos. Neste exemplo, usamos 3 vizinhos para imputar os valores faltantes.

Essas são algumas das técnicas que podemos usar para garantir a completude do dado em Python. É importante lembrar que cada conjunto de dados pode ter suas próprias particularidades e que a escolha da técnica adequada depende do contexto e dos objetivos da análise.

## Técnicas de tratamento de dados

A validação da integridade e qualidade dos dados é uma etapa crítica na análise de dados. Ela é essencial para garantir que os dados usados em análises sejam confiáveis e precisos, e que as conclusões baseadas nesses dados sejam válidas. Agora, vamos explorar várias técnicas para validação da integridade e qualidade dos dados, incluindo tratamento de duplicidade, nulidade, ausência de registros ou colunas, bem como divergência de tipagem, usando Python.

### Tratamento de Duplicidade

O tratamento de duplicidade é uma técnica que visa encontrar e remover observações duplicadas em um conjunto de dados. Duplicatas podem ocorrer em vários cenários, como em bases de dados coletadas de formulários ou transações. O uso de técnicas para lidar com a duplicidade é importante para evitar distorções na análise de dados.

Vamos criar um conjunto de dados de exemplo com duplicatas:

import pandas as pd

data = {'Nome': ['João', 'Maria', 'Pedro', 'Maria', 'Luiz'],

'Idade': [25, 30, 35, 30, 27],

'Cidade': ['São Paulo', 'Rio de Janeiro', 'Belo Horizonte', 'Rio de Janeiro', 'São Paulo']}

df = pd.DataFrame(data)

print(df)

A saída deve ser:

Nome Idade Cidade

0 João 25 São Paulo

1 Maria 30 Rio de Janeiro

2 Pedro 35 Belo Horizonte

3 Maria 30 Rio de Janeiro

4 Luiz 27 São Paulo

Para remover as duplicatas do DataFrame, podemos usar o método drop\_duplicates() do pandas:

# remover as duplicatas

df = df.drop\_duplicates()

print(df)

A saída deve ser:

Nome Idade Cidade

0 João 25 São Paulo

1 Maria 30 Rio de Janeiro

2 Pedro 35 Belo Horizonte

4 Luiz 27 São Paulo

### Divergência de Tipagem

Outra questão que pode surgir em conjuntos de dados é a divergência de tipagem de dados. Por exemplo, uma coluna que deve conter valores inteiros pode conter valores de ponto flutuante ou strings. Para lidar com isso, podemos usar o método astype() para converter a tipagem das colunas do DataFrame:

data = {'Nome': ['João', 'Maria', 'Pedro', 'Luiz'],

'Idade': ['25', '30', '35', '27'],

'Cidade': ['São Paulo', 'Rio de Janeiro', 'Belo Horizonte', 'Salvador']}

df = pd.DataFrame(data)

# converter a coluna 'Idade' para inteiro

df['Idade'] = df['Idade'].astype(int)

print(df)

A saída deve ser:

Nome Idade Cidade

0 João 25 São Paulo

1 Maria 30 Rio de Janeiro

2 Pedro 35 Belo Horizonte

3 Luiz 27 Salvador

### Tratando valores discrepantes

Valores discrepantes, ou outliers, são valores extremos que podem afetar a precisão da análise. Podemos usar o NumPy para identificar e tratar valores discrepantes. Vamos usar o seguinte código para identificar valores discrepantes em um DataFrame:

import numpy as np

# criar um array com valores aleatórios

arr = np.array([1, 2, 3, 100, 200, 300])

# identificar valores discrepantes

desv\_padrao = np.std(arr)

media = np.mean(arr)

limite = media + (2 \* desv\_padrao)

outliers = arr[arr > limite]

print(outliers)

Este código irá criar um array com valores aleatórios e, em seguida, irá identificar os valores discrepantes usando o desvio padrão e a média do array. O limite é calculado como a média mais duas vezes o desvio padrão. Os valores discrepantes são aqueles que estão acima do limite.

Podemos tratar os valores discrepantes de várias maneiras, como removê-los do conjunto de dados ou substituí-los por um valor específico. Vamos usar o seguinte código para substituir os valores discrepantes em um array pelo valor médio do array:

# substituir valores discrepantes pelo valor médio do array

arr\_sem\_outliers = arr.copy()

arr\_sem\_outliers[outliers] = np.mean(arr)

print(arr\_sem\_outliers)

Este código irá substituir os valores discrepantes no array pelo valor médio do array usando o método copy() e a indexação booleana do NumPy.

## Conclusão

Para finalizar, pudemos constatar que a validação da integridade e qualidade dos dados é fundamental para garantir que os dados sejam confiáveis e precisos. Discutimos os princípios da qualidade dos dados e seus benefícios, bem como alguns exemplos em Python e técnicas para lidar com duplicidade, nulidade, ausência de registros ou colunas e a divergência de tipagem. Além disso, vimos que o Python e as bibliotecas pandas e numpy fornecem várias ferramentas úteis para lidar com esses problemas de integridade e qualidade dos dados. Com essas práticas, podemos garantir a qualidade dos dados e tomar decisões mais adequadas para aprimoramento de nossos negócios.

# Referências

* Livro: "Data Cleaning with Python", por Hadrien Jean
* Livro: "The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling", por Ralph Kimball e Margy Ross
* Livro: "Principles of Data Management: Facilitating Information Sharing", por Keith Gordon
* Documentação oficial do pandas: <https://pandas.pydata.org/docs/>
* "Python for Data Analysis" por Wes McKinney: <https://www.oreilly.com/library/view/python-for-data/9781491957653/>
* "Data Wrangling with Python" por Jacqueline Kazil e Katharine Jarmul: [https://www.oreilly.com/library/view/data-wrangling-with/9781491948811/](https://www.oreilly.com/library/view/data-wrangling-with/9781491948804/)
* "Python Data Science Handbook" por Jake VanderPlas: <https://jakevdp.github.io/PythonDataScienceHandbook/>
* "Python Machine Learning" por Sebastian Raschka e Vahid Mirjalili: <https://www.oreilly.com/library/view/python-machine-learning/9781787125933/>