Análise de dados de Python com Pandas

Análise e preparação de dados, incluindo pré-processamento e alimentação do processo, e apresentação visual de resultados usando bibliotecas da linguagem Python.

Prof. Fernando Cardoso Durier da Silva

1. Itens iniciais

Propósito

Compreender a análise de dados em Python é fundamental para o cientista de dados, por ser uma das linguagens mais utilizadas para esse tipo de tarefa, não só pela automatização, mas também pela robustez de suas bibliotecas de apoio.

Preparação

Para a compreensão e reprodução dos exemplos, é imprescindível ter instalado, em sua máquina de estudos, o Python na versão 3.8 ou superior, as bibliotecas Pandas, Numpy e Plotly. Será necessária também uma ferramenta para execução de notebooks, como Jupyter, que pode ser instalado como uma biblioteca do Python. Nesse contexto, notebooks são arquivos com extensão IPYNB que contêm documentos gerados pelo ambiente de desenvolvimento interativo, usados por cientistas de dados que trabalham com a linguagem Python.

Objetivos

- Reconhecer os componentes e a sintaxe do Python para análise de dados.
- Descrever a preparação de dados para análise no Python.
- Descrever a manipulação de dados no Python.
- Aplicar a visualização de dados no Python.

Introdução

A análise de dados é fundamental para a inicialização de qualquer projeto de sistemas de informação, uma vez que nos trará insights de como melhor desenhar o processo, além do que nos foi entregue no levantamento de requisitos.

Atualmente, com os avanços da tecnologia, a automatização da análise de dados se dá de forma menos flexível com sistemas de análise dedicados, principalmente graças a linguagens de programação que nos possibilitam escrever desde simples scripts até sistemas inteiros de modelos exploratórios de dados.

Uma das linguagens mais utilizadas para tal é o Python, linguagem plena de recursos graças às suas bibliotecas de ciência de dados, processamento de dados, análise de séries temporais etc.

Vamos aprender os principais componentes do Python, sua sintaxe para análise de dados e uma série de recursos e técnicas que serão instrumentais para o sucesso de nossas análises. Também entenderemos os tipos de dados e quais cuidados devemos tomar com eles para extrair o melhor dos conjuntos de dados a serem analisados, bem como saber onde procurá-los.

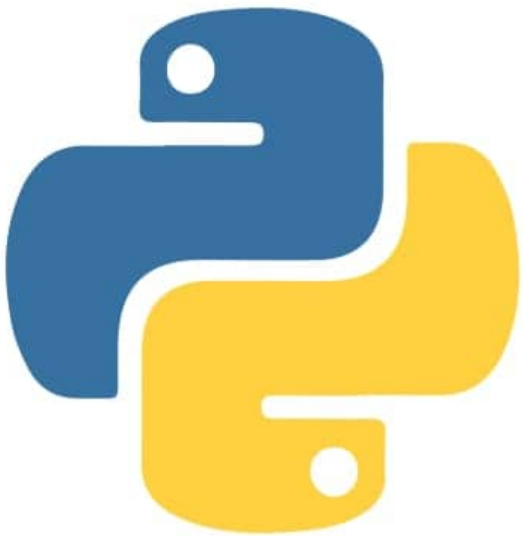
Por fim, aprenderemos como visualizar os dados para concretizar nossas análises e obter insights imediatos.

Introdução aos componentes e à sintaxe de Python

A linguagem Python é chamada de **linguagem de alto nível** por causa da sua abstração com relação ao hardware e aos registros, diferente de Assembly por exemplo.

> Linguagens de programação de alto nível têm mais distância do código de máquina e mais proximidade com a linguagem humana.

Esse é um dos principais objetivos do Python, que foi idealizado para ser parecido com o inglês de modo que, quando qualquer pessoa fosse ler os códigos fontes, poderia entender facilmente, independentemente de ser especialista.



Ícone da linguagem de programação Python.

Curiosidade

Por mais que o símbolo da linguagem seja o da serpente constritora não venenosa Python (Píton), a origem do nome da linguagem, na realidade, refere-se ao grupo humorístico britânico Monty Python, criadores da série cômica o Circo Voador de Monty Python (Monty Python's Flying Circus). Em contrapartida, os comediantes resolveram homenagear Lord Montgomery (Monty), oficial do exército britânico, e queriam uma palavra evasiva como Python, aqui sim se referindo à serpente.

Grupo Monty Python.

Atenção

Diferentemente de linguagens de médio nível como C e Java, Python não precisa declarar o tipo de suas variáveis, o que se alinha também com o objetivo de imitar o idioma inglês, mas, ainda assim, seus tipos são fortes, ou seja, não sofrem coerções.

Falando em sintaxe, a primeira regra que devemos observar é a de **indentação**, que serve para separar blocos lógicos, de loop, classes e funções.

A indentação é feita por recuo com espaços em branco após o bloco lógico em questão; para o interpretador da linguagem, isso se traduz em ler primeiro a linha de função e, dentro da indentação, a sua parametrização.

A declaração de dados não precisa do preâmbulo de tipos para dizer que a variável é do tipo inteiro ou string, mas os tipos de dados são associados às variáveis em Python, mesmo não sendo declarados explicitamente. Entre eles, temos:

Tipo	Descrição	Exemplo
str, unicode	Cadeia de Caracteres	“batata”, u“alface”
list	Lista	[1,2,3], ['a','b', 'c'], [1.0, 'a', True]
Tuple	Tupla	(“what”,”who”,”whom”,”where”,”when”)
set, frozenset	Conjunto não ordenado	set([1,2,3]), frozenset(['batata','alface','uva'])
Dict	Dicionário, Conjunto chave-valor	{“a”:1,”b”:2,”c”:3}, {“k1”:”a”,”k2”:”b”,”k3”:”c”}
Int	Número Inteiro (se muito grande será convertido em Long)	42, 50, 100, 1, 2, 3, 78394024920L
float	Número de Ponto Flutuante ou racional	3.7, 4.55, 9.012, 9.18293, 10.1
complex	Número Complexo	1e10, 3i, 7+4j
bool	Booleano	True, False
!=	Diferente	Diferente de, <>, !==

Quadro: Tipos de dados e variáveis em Python.Elaborado por: Fernando Durier.

Temos também as construções clássicas de controle de execução de comandos, como na maioria das linguagens de programação, por exemplo:

Tipo de Constructo	Constructo	Descrição	Exemplo
Condicional	If	Condicional se	if(x%2 == 0): print(“X é par”)
	Else	Condicional senão	if(x%2 == 0): print(“X é par”) else : print("X é ímpar")

	Elif	Condicional senão se	If(x==0) :
			print("X é zero")
			elif(x%2 == 0):
			print(“X é par”)
			else :
			print("X é ímpar")

Repetição	For	Laço Para todo	for c in df.columns: print(“c:”, c)
	While	Laço Enquanto condição	while(true): main()
Classe	Class	Classe	Class SGDRegression: def __init__(): pass()
Funções/Rotinas	Def	Definição de função	def soma(a,b): return a+b
Escopo	With	Dado que, ou no escopo de	with open(“./query.sql”) as file: sql_query=file.read()

Quadro: Tipos de comandos em Python.Elaborado por: Fernando Durier.

Para definirmos um módulo, um bloco de código para ser reaproveitado em outras partes do projeto, basta criarmos uma pasta com o nome do módulo em questão, dentro dela criarmos um arquivo **__init__.py** e depois criarmos o arquivo do código fonte **module.py**.

Podemos importar o bloco lógico em outra parte do projeto da seguinte forma:

```
python


 ```
 from module import *
 ```
```

Ou

```
python


 ```
 from .module import function_x
 ```
```

Ou

```
python


 ```
 from ..module.submodule import function_y
 ```
```

Respectivamente, essas importações se referem a caminhos:

- Absoluto.

- Relativo na mesma pasta onde é chamada a parte.
- Relativo nas duas pastas da chamada original.

Você deve estar se perguntando: por que __init__?

O __init__ é o equivalente ao construtor de classe das linguagens orientadas a objetos; é a partir do método construtor que podemos instanciar objetos, seguindo os parâmetros declarados na função __init__.

Agora você deve estar ainda mais intrigado sobre a razão de um arquivo __init__.py vazio! Sabe o por quê?

> Quando importamos um módulo, é como se importássemos uma classe para outra parte do projeto, e o __init__.py é executado pelo motor do Python para atrelar os objetos pelo módulo declarado ao namespace (“domínio”) do módulo. Em outras palavras, podemos entender como o construtor do pacote/módulo ali declarado.

Estrutura de projetos e boas práticas

Assim como em outras linguagens, **podemos importar bibliotecas criadas por outras pessoas**, o que nos facilita muito. Essa é uma das grandes vantagens da computação nos tempos atuais, pois a comunidade de desenvolvedores é grande, diversa e muito colaborativa. Por mais que estejamos importando módulos externos, teremos acesso a documentação e trilhas de discussões em plataformas como o **StackOverflow** e o **Github**.

Mas como fazemos isso?

> É simples, por meio do gerenciador de pacotes do Python, o **pip**.
>
> Na maioria das vezes, o que basta é digitarmos no terminal ou prompt de comando, o comando “pip install < nome do módulo >” e pronto, assim importamos no nosso código como “import < nome da biblioteca >” e teremos acesso aos seus módulos e métodos.


Biblioteca referenciando a imensidão de módulos presentes no Python.

Atenção

É importante registrar a lista de bibliotecas para que outras pessoas que tiverem acesso ao projeto possam rodá-lo em suas máquinas. E isso pode ser feito por meio do arquivo requirements.txt, um arquivo de texto que contém as dependências do projeto.

Para isso, basta digitar o comando:

```
python

    pip freeze > requirements.txt
```

Para que outra pessoa possa importar as mesmas bibliotecas, ou para que você mesmo possa importar de novo em outro computador, basta digitar:

```
python

    pip install -r requirements.txt
```

Existem formas mais avançadas de construirmos o projeto e suas dependências por meio dos **ambientes virtuais**, que são espaços de trabalho do projeto, como se fossem máquinas virtuais leves para rodar os programas. Se os comandos pip forem rodados enquanto o ambiente estiver de pé, então as bibliotecas não serão instaladas globalmente, mas sim restritas ao ambiente virtual (**virtualenv**).

Para levantar o ambiente, criando o ambiente virtual para a pasta corrente, basta digitar o comando:

```
python

    venv
```

Para sair, basta digitar:

```
python

    Deactivate
```

Existem alternativas como o **Pyenv** e o **pipenv** que, além de criar o ambiente virtual, podem gerenciar versões diferentes de Python no mesmo computador, mas diferentemente do virtualenv (venv), **o Pyenv e o pipenv não vêm instalados junto ao Python**.

Por fim, falaremos do **Jupyter Notebook**, um ambiente de scripts Python, muito útil para análise exploratória, que pode ser instalado no nível global mesmo, fora de virtualenv, pois queremos que toda a máquina tenha acesso a ele.

Para instalá-lo, basta digitar o comando:

```
python

    pip install jupyter
```

Comentário

Para abrir um servidor local de Jupyter Notebook, basta digitarmos no terminal “jupyter notebook” na pasta do projeto em questão e pronto, o servidor será levantado e automaticamente o browser abrirá na IDE do Jupyter Notebook. Então, você poderá criar notebooks, que são scripts em células de Python, muito utilizados para prototipação e análises exploratórias.

Ativos de dados

Falaremos agora dos dados. Afinal, qualquer sistema de informação tem como entrada dados que serão processados pelos seus programas até saírem como informação relevante. No caso de projetos em Python, os dados podem vir externamente por meio de captação (scraping) ou de artefatos de dados que nada mais são do que planilhas, textos, entre outros arquivos que são usados localmente mesmo.

Ativos de dados.

A fim de lidarmos com os dados de forma elegante, deve ser criada uma pasta no nível da raiz do projeto, para armazenamento dos arquivos; por convenção, podemos chamá-la de assets (ativos). Tal pasta pode ainda ser subdividida por categorização preestabelecida dos ativos de dados. Essa subdivisão é muito comum em projetos de FrontEnd (projetos de interface gráfica, principalmente de websites), em que os desenvolvedores guardam ícones e outros dados estáticos.

Comentário

Os ativos de dados podem ser usados para servir de base de dados temporária ou de testes/ experimentação de certos sistemas, ou também como recursos que alimentam bibliotecas importadas, por exemplo, licenças de software.

Biblioteca Pandas

A manipulação de dados em Python, em sua esmagadora maioria, é feita com DataFrames, uma vez que são estruturas muito mais cômodas e robustas de se trabalhar do que matrizes. Antes de falarmos dos DataFrames em profundidade, precisamos explicar o Pandas.

> Pandas é uma biblioteca em Python, criada para manipulação e análise de dados, oferecendo estruturas e operações para manipular tabelas numéricas e séries temporais.

O nome Pandas, ao contrário do que possa parecer, não vem do urso fofo monocromático da China, mas sim do conceito panel data, ou dado em painel, no plural, panel data sets, e daí o nome Pandas.

O conceito em si é um termo estatístico que se refere a conjuntos que incluem diferentes unidades amostrais acompanhadas ao longo do tempo, o que faz muito sentido, pois cada coluna de um DataFrame, principal estrutura do Pandas, é um dado do tipo Series, ou série temporal.

```
Cool
1000 250
```

Out[19]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
92	Linda	Female	5/25/2000	5:45 PM	119009	12.506	True	Business Development
65	Steve	Male	11/11/2009	11:44 PM	61310	12.428	True	Distribution
445	Chris	Male	12/12/2006	1:57 AM	71642	1.496	False	NaN
732	Henry	Male	5/12/1986	2:04 AM	59943	1.432	False	Finance
352	NaN	Male	10/9/2011	9:29 AM	69906	4.844	NaN	Engineering
293	Jesse	Male	10/25/1999	3:35 PM	118733	9.653	False	Marketing
456	Deborah	NaN	2/3/1983	11:38 PM	101457	6.662	False	Engineering
171	Patrick	Male	8/17/2007	3:16 AM	143499	17.495	True	Engineering
562	Sara	NaN	10/7/1983	1:35 PM	87713	18.863	True	Legal
320	NaN	Female	7/8/2008	11:40 PM	62960	14.356	NaN	Sales
568	Susan	Female	4/18/1986	9:31 AM	90829	19.142	False	Marketing
775	Rose	Female	11/3/1999	9:06 AM	75181	6.060	True	Finance
32	NaN	Male	8/21/1998	2:27 PM	122340	6.417	NaN	NaN

Exemplo de Pandas DataFrame disponível no site geeksforgeeks.org no artigo Python Pandas Dataframe.sample() de Kartikaybhutanios.

Comentário

Os DataFrames são justamente esses painéis de dados, mas podemos encará-los como tabelas para fins de abstração. São muito utilizados em projetos de análise de dados como criação de dashboards, scripts de análise de dados e aplicações de aprendizado de máquina.

Introdução aos componentes e à sintaxe de Python

No vídeo a seguir, abordamos os principais componentes e a sintaxe básica da linguagem Python para a análise de dados.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Vem que eu te explico!

Os vídeos a seguir abordam os assuntos mais relevantes do conteúdo que você acabou de estudar.

Estrutura de projeto e boas práticas

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Ativos de dados

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

Quais os comandos para levantar e sair de um ambiente virtual Python (virtualenv)?

A

start; quit.

B

startvenv; !venv.

C

venv; ~venv.

D

venv; deactivate.

E

activate; deactivate.

A alternativa D está correta.

O comando venv levanta o ambiente virtual em Python, e quando não se deseja mais trabalhar ali dentro, basta digitar deactivate para sair. As demais alternativas possuem comandos não suportados pela linguagem Python.

Questão 2

O constructo with da linguagem Python se refere a que tipo de estrutura de controle?

A

Comparação

B

Atribuição

C

Laço

D

Condicional

E

Escopo

A alternativa E está correta.

O constructo with delimita o escopo das funções e variáveis abaixo dele, como na leitura de um arquivo, por exemplo. As demais alternativas referem-se a estruturas da linguagem, porém os constructos correspondentes são diferentes do with.

Obtenção do conjunto de dados

A coleta de dados é o processo de captura e medição de informações e variáveis de interesse, de forma sistemática que permita responder a perguntas de pesquisa, bem como testar hipóteses e avaliar resultados.

Coleta de dados.

Existe uma diferença entre os conjuntos de dados qualitativos e quantitativos. Vamos ver qual é?

Dados qualitativos

São dados não numéricos, em sua maioria, normalmente descritivos ou nominais. Apresentam-se, em geral, em formato de textos, sentenças ou rótulos. As perguntas que geram essa categoria de dados são abertas e os métodos envolvidos para seu tratamento são grupos de foco, de discussão e entrevistas. São um bom jeito de mapear o funcionamento de um sistema ou a razão de um fenômeno. Em contrapartida, são abordagens custosas que consomem muito tempo, e os resultados são restritos aos grupos de foco envolvidos.

Dados quantitativos

São os dados numéricos, que podem ser matematicamente computados. Essa categoria de dado mede diferentes escalas que podem ser nominais, ordinais, intervalares e proporcionais. Na maioria dos casos, esses dados resultam da medição de algum aspecto ou fenômeno. Normalmente, existe uma abordagem sistemática muito bem definida e mais barata de se implementar do que a coleta de dados qualitativa, uma vez que é possível construir processos automáticos ou simplesmente consumir relatórios gerados. Entre os métodos dessa categoria, temos os surveys, as queries, o consumo de relatórios e os escavadores (scrapers).

É importante ressaltar que **essas categorias de dados não são mutuamente exclusivas**, pois é muito comum encontrar relatórios ou fazer entrevistas cujo resultado contenha tanto dados quantitativos (renda, número de familiares, despesas etc.) como dados qualitativos (endereço, sobrenome, instituição de ensino, grau de instrução etc.).

Existe ainda uma diferença no que tange à obtenção dos dados em si, classificados como dados primários e secundários. Vejamos a seguir.

Dados primários

O que são:
São aqueles coletados de primeira mão, ou seja, dados que ainda não foram publicados, autênticos ou inéditos.

Como é um dado recém-coletado, não tem interferência humana e, por isso, é considerado mais puro do que os dados secundários.

Entre as fontes de dados primários, temos surveys, experimentos, questionários e entrevistas.

Vantagens:
São dados puros, coletados para a resolução de um problema específico e, se necessário, podem ser coletados novamente a qualquer momento para aumentar a quantidade.

Dados secundários

O que são:
São aqueles dados que já foram publicados de alguma forma, ou seja, sofreram alguma interferência humana. Por exemplo, ao fazermos a revisão de literatura em qualquer estudo, estamos revisando dados secundários.

Entre as fontes de dados secundários, temos: livros, registros, biografias, jornais, censos, arquivos de dados etc. Os dados secundários são muito úteis quando não conseguimos fazer coleta em primeira mão; por exemplo, ao reproduzirmos um estudo de um trabalho da literatura, temos que utilizar os dados providos pelos autores.

Vantagens:
Economia de tempo em não ter que desenvolver um sistema de coleta, menor custo e delegação de responsabilidade (em relação aos dados) do profissional para o dono dos dados originais.

Tratamento de dados nulos ou corrompidos

Um problema muito comum na atividade de pré-processamento de dados é a qualidade dos dados. Os dados podem vir certinhos, consistentes, talvez muito variados, mas, pelo menos, completos. Entretanto, é possível que os dados venham com atributos faltantes, registros nulos, registros mal escritos (desformatados) etc. Para esses casos, existem diversas formas de resolver o problema, dependendo do tamanho da base, do tipo de processo de extração e da importância do atributo prejudicado.

Comentário

Dados faltantes ou nulos em bases grandes (por volta da ordem de grandeza de 10.000 registros ou mais) podem ser resolvidos ignorando o registro todo, ou seja, removendo-o da base, se a proporção de nulos não for expressiva (não passar de 10% da quantidade de registros). Essa estratégia é comum para bases grandes, pois a remoção desses registros nulos não será tão danosa ao processo de treinamento.

Outra estratégia para esse caso específico, de base de dados grande, é utilizar técnicas de regressão para dados numéricos ou de classificação para dados categóricos, para o preenchimento automático desses dados. O fato de a base ser grande ajuda o algoritmo de preenchimento automático, sendo claro que dados com variância alta podem prejudicar esse processo.

Para dados faltantes ou nulos em bases de dados **muito restritas** ou **pequenas** (por volta da ordem de grandeza de 1.000 ou menos), temos duas alternativas: ou tentamos preencher de forma automática como vimos anteriormente, ou voltamos ao processo de coleta e tentamos melhorá-lo a fim de consertar o problema que causou a nulidade ou a falta.

E, finalmente, para dados faltantes ou nulos em bases de dados **precárias** em que a exclusão do registro ou a interpolação é inviável, o correto é retomar diretamente ao processo de coleta, pois claramente os dados são insuficientes para o projeto.

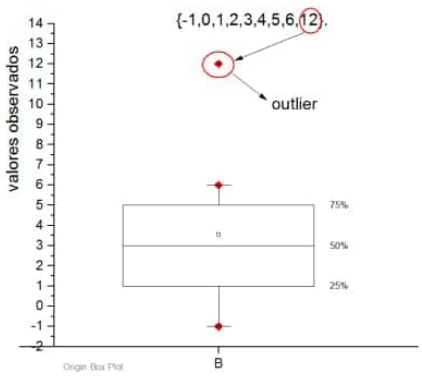
Para dados repetidos, normalmente a solução é simples, pois basta eliminá-los, a não ser que a repetição seja apenas para um subconjunto de atributos. Se for o caso, provavelmente a repetição é uma decomposição de uma agregação ou é uma repetição de um evento em diferentes períodos de tempo.

Atenção

Para resolver o problema de dados repetidos, é sempre útil estudar os metadados do conjunto de dados, se estiverem disponíveis, ou estudar a origem deles.

Regularização de dados

Os dados coletados, além de poderem ter sido corrompidos ou simplesmente estar faltando, podem estar com ruídos, ou pelo menos com pontos fora da curva, os chamados outliers. Por exemplo, na série [1,1,2,3,4,5,6,100,7,8,9,50], podemos dizer que 50 e 100 são outliers, uma vez que são muito distantes do restante. Tal perturbação pode causar problemas nos processamentos do sistema sobre esse conjunto, levando-o a demonstrar comportamentos indesejados.



Ponto fora da curva. Podemos ver que o ponto se distancia demais da média e das variâncias esperadas do boxplot.

Existem várias formas de contornar tal situação. Podemos remover os dados destoantes do conjunto, assim limpando o sistema. Entretanto, se quisermos levá-los em consideração mesmo assim, o que pode ser feito? Para isso, podemos usar a **regularização**.

> Regularizar um dado significa colocar seus atributos em escala; assim, o conjunto não sofrerá tanto com um ponto fora da curva, bem como se adaptará a novos valores que venham a ser incorporados.

Dentre as opções de regularização, temos o método **min max**, que consiste em colocar os dados em escala com o mínimo valor do atributo no conjunto contra o máximo valor.

A fórmula é:

$$d\_observado(d\_max - d\_min)$$

A fórmula do método min max coloca o dado observado em uma régua que começa no mínimo valor possível e vai até o máximo, transformando assim o d_observado em um valor entre 0 e 1.

Demonstração com Python e Pandas

Existem várias formas de incorporarmos os dados nos nossos scripts Python. Podemos fazer a coleta dos dados a partir de bibliotecas, que fazem varreduras em sites, ou em dados semiestruturados como notícias e tweets por meio do **BeautifulSoup** e **Scrapy**.

Para fins de estudo e demonstração, vamos tratar como se nossos dados recuperados fossem arquivos, como os separados por vírgula, CSV. Com o Pandas, basta utilizarmos o seguinte comando de importação da biblioteca inicialmente:

````
python


 ```
 import Pandas as pd
 df = pd.read_csv('./DIRETÓRIO_DE_DADOS/ARQUIVO.csv')
 ```
````

Atenção

Os delimitadores de strings nos comandos Python são aspas simples (') no início e no final.

Como esperado, o Pandas lê o arquivo separado por vírgulas convertendo-o em estrutura tabelar e, por padrão, a primeira linha do arquivo é considerada o **cabeçalho da tabela**. Também por padrão, para esse tipo de operação, o leitor de arquivo do Pandas considera a vírgula como separador de colunas e o parágrafo como separador de registros ou linhas.

> Esse padrão pode ser alterado passando o marcador desejado para o parâmetro sep.

Existe o leitor de Excel (read_excel), que funciona exatamente da mesma forma, com o cabeçalho também na primeira linha por padrão, mas o separador padrão é o **tab** (espaço tabular), o que também pode ser alterado. A maior diferença é que no Excel, diferentemente do CSV, pode haver abas (sheets); por padrão, considera-se a primeira aba na leitura.

A função de leitura do Pandas é tão poderosa que pode ler arquivos não convencionais como lista de dicionários, bastando utilizar o método **read_json**, por exemplo, que tem um atributo específico chamado **orient**, que vem de orientação, uma vez que a lista de dicionários pode ser passada como **record**, ou seja, por vetor de registros, ou, ao contrário, **split**, que é vetor de valores de coluna variável.

```
python


 ```
 import Pandas as pd
 df = pd.read_json('./DIRETÓRIO_DE_DADOS/ARQUIVO.json', orient='records')
 ```
```

A conversão de dados mais comum é **transformar listas de dicionários em DataFrames**, dentro do próprio código, ou seja, por atribuição de variável, o que é mais simples ainda e pode ser feito da seguinte forma:

```
python


 ```
 import Pandas as pd
 json_array=[{'a':1,'b':2}, {'a':3,'b':4}, {'a':5,'b':6}]
 df = pd.DataFrame(json_array)
 ```
```

Vale ressaltar que nem todos os arquivos **JSON** já vêm no formato de lista de dicionários, mas sim como **strings convertidas**. Logo, é considerada uma boa prática tentar fazer a decodificação caso haja a dúvida:

```
python


 import json
 json_array=”[ {'a':1,'b':2}, {'a':3,'b':4}, {'a':5,'b':6} ]”
 df = pd.DataFrame(json.loads(json_array) )
```

Assim como é possível ler os arquivos, também é possível persistirmos com eles; o método usado para isso é **to_csv, to_excel, to_json** etc. Deve-se dar especial atenção à **persistência**, pois em casos como a exportação dos dados no formato de lista de dicionários, ou **JSON Array**, o padrão de orientação é importantíssimo porque interfere na retrocompatibilidade com outros sistemas que compartilhem os dados.

```
python


 ```
 import Pandas as pd
 json_array=[{'a':1,'b':2}, {'a':3,'b':4}, {'a':5,'b':6}]
 df = pd.DataFrame(json_array)
 df.to_json('./DIRETÓRIO_DE_DADOS/ARQUIVO.json', orient='records')
 ```
```

Existem formas ainda mais extravagantes de se ler dados com o Pandas como o **read_sql**, que lê os dados a partir de conexões **ODBC** ou **JDBC** com bancos de dados compatíveis, sendo que cada tipo de banco tem suas especificidades.

Comentário

Ainda há a possibilidade de leitura por URL, em que, no lugar de passarmos o caminho de diretórios local da máquina, passamos o caminho do arquivo no sistema on-line, que será transferido para o projeto Python seguindo um protocolo FTP.

Preparação de dados para análise no Python

As técnicas de preparação de dados para análise no Python são abordadas no vídeo a seguir.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Vem que eu te explico!

Os vídeos a seguir abordam os assuntos mais relevantes do conteúdo que você acabou de estudar.

Tratamento de dados nulos ou corrompidos

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Regularização de dados

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

Na fase de pré-processamento, o objetivo da regularização de dados é

A

deletar os dados.

B

embaralhar os dados.

C

remover pontos fora da curva e colocar os dados em escala.

D

atualizar os dados.

E

remover pontos fora da curva e tirar os dados de escala.

A alternativa C está correta.

O objetivo da regularização dos dados, como já diz o nome, é fazer com que eles fiquem na mesma escala, evitando ruídos de diferença de tamanho, e também suavizar e até mesmo remover pontos fora da curva.

Questão 2

Dados nulos e corrompidos são um problema para a análise de dados. Qual das técnicas abaixo é viável para lidar com esse tipo de problema?

A

Remoção do registro com dados faltantes ou nulos do conjunto.

B

Embaralhamento do registro com dados faltantes ou nulos.

C

Fusão dos dados nulos ou faltantes.

D

Transformação linear dos dados nulos ou faltantes.

E

Rotação dos dados nulos ou faltantes.

A alternativa A está correta.

Dentre as alternativas, a remoção dos dados faltantes é a única viável a solucionar o problema. Existem outros métodos para tratamento de dados nulos, como a interpolação do valor pela média da coluna ou algo do gênero, que não estão entre as opções de respostas.

DataFrames e tabelas

Uma característica interessante na estrutura de dados de **DataFrame** ou dados em painel é o fato de poderem ser criados a partir de praticamente qualquer outra estrutura. Podemos criar Pandas DataFrames a partir de arquivos CSV, Excel, listas de dicionários, matrizes, junções de listas etc.

Comentário

As tabelas, por mais que tenham essa característica de DataFrames, são muito similares às estruturas que conhecemos nos bancos de dados, como colunas, por exemplo, sendo que cada coluna tem um tipo próprio, por exemplo object, que é o padrão de tipo de dados para strings, ou o equivalente a varchar. A outra semelhança é o index, que serve para marcar cada linha do DataFrame e ordenar os dados, assim como a estrutura de mesmo nome nas tabelas dos bancos de dados SQL.

Para descrever o conteúdo de um DataFrame no Pandas, podemos fazer de duas formas:

````
python

 ```
 df.info()
 ```
````

O DataFrame descrito pelo info() resulta na descrição de cada coluna e seu tipo, com a contagem de valores não nulos, como mostrado a seguir:

```
python


 RangeIndex: 551 entries, 0 to 550
 Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
 ---  ------  --------------  -----
 0   date    551 non-null    object
 1   text    551 non-null    object
 2   class   551 non-null    object
 dtypes: object(3)
 memory usage: 13.0+ KB
```

A alternativa é o describe, que é menos detalhista do que o info(), executado pelo comando:

```
python


 ```
 df.describe()
 ```
```

O describe resulta na descrição de estatísticas bem básicas, por exemplo, a contagem de cada coluna (count), quantos valores únicos de cada variável (unique) e quantas categorias, bem como o primeiro registro (top) e a frequência (freq.).

```
python


                        date      text      class
     count                   551   551        551
     unique                  344       551              1
     top            21/06/02       abc       False
     freq                     6         1             551
```

Tente executar, como exercício, o exemplo anterior da lista de dicionários, com o df.info() e com o df.describe().

```
python


 import Pandas as pd
 json_array=[ {'a':1,'b':2}, {'a':3,'b':4}, {'a':5,'b':6} ]
 df = pd.DataFrame(json_array)
 df.info()
 # df.describe() – note que pode ser necessário usar print para exibir no console
```

Essas informações são básicas e nos dão uma previsão de como lidarmos com o DataFrame. Existem outras funções que são mais parecidas com as que estamos acostumados nas tabelas de SQL, e as veremos a seguir.

Manipulação de dados em DataFrames

Como mencionamos, os DataFrames são similares a tabelas, como as dos bancos de dados convencionais **PostgreSQL, MySQL, DB2** etc. Tais estruturas contam com operações muito parecidas com as de projeção, seleção, deleção e junção.

Comentário

Quando fazemos uma projeção ou uma seleção, o que queremos é criar um subconjunto dos dados originais. Tal operação é realizada na linguagem SQL pelo comando SELECT, enquanto para os DataFrames podemos fazer o mesmo com os métodos loc, iloc e query.

O loc é o método que projeta as colunas do DataFrame pelos rótulos e funciona da seguinte forma:

````
python

```
df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
      index=['cobra', 'viper', 'sidewinder'],
      columns=['max_speed', 'shield'])
print(df.loc['viper'])
print(df.loc[['viper', 'sidewinder']])

```
````

Nesse exemplo, criamos um DataFrame artificial e fizemos duas projeções:

‘viper’

A primeira com apenas uma coluna.

‘viper’, ‘sidewinder’

A segunda com duas colunas.

Além do tamanho da projeção, podemos reparar que, ao selecionarmos apenas uma coluna, teremos de volta uma série (Pandas series) e, caso escolhamos mais de uma coluna, além de termos que passar o conjunto como uma lista dentro dos colchetes do loc, o retorno será um DataFrame, uma vez que é o coletivo de duas ou mais séries.

Exemplo com iloc:

```
python


 ```
 mydict = [{'a': 1, 'b': 2, 'c': 3, 'd': 4},
           {'a': 100, 'b': 200, 'c': 300, 'd': 400},
           {'a': 1000, 'b': 2000, 'c': 3000, 'd': 4000 }]
 df = pd.DataFrame(mydict)

 print(df.iloc[0])
 print(df.iloc[[0]])
 print(df.iloc[[0, 1]])
 print(df.iloc[:3])
 print(df.iloc[lambda x: x.index % 2 == 0])
 print(df.iloc[[0, 2], [1, 3]])
 print(df.iloc[1:3, 0:3])
 ```
```

No caso do iloc, a projeção depende dos indexes numéricos das linhas e colunas de um DataFrame.
Veja o detalhamento a seguir.

Print 1
O primeiro print nos trará a primeira coluna do conjunto.

Print 2
O segundo print nos trará a primeira linha do DataFrame.

Print 3
O terceiro print nos trará a primeira e segunda linhas do DataFrame.

Print 4
O quarto print nos trará as três primeiras linhas do DataFrame.

Print 5
O quinto print nos trará a projeção com a condição de que o valor das células seja par.

Outros prints
Os dois últimos prints nos projetam uma submatriz (ou filtro) do DataFrame original, o primeiro sendo feito de forma direta e o segundo por fatiamento.

Exemplo com query:

```
python


 ```
 df = pd.DataFrame({'A': range(1, 6),
                    'B': range(10, 0, -2),
                    'C': range(10, 5, -1)})
 print(df.query('A > B'))
 print(df.query('B == `C C`'))
 print(df[df.A > df.B])
 ```
```

Por fim, temos o jeito de fazer seleções e projeções por queries. Assim como nos bancos de dados SQL, isso é bastante intuitivo. No primeiro print, temos uma query que retorna dados cujo valor da dimensão A é maior do que o da dimensão de B. Já a segunda query é uma igualdade, em que desejamos os dados cujo valor da dimensão B seja igual ao da dimensão 'C C' (repare que, dentro da query, o que for textual, seja valor ou dimensão, tem de ser declarado com o acento crase (`)). E, por fim, um último jeito de filtrar é passando a referência da coluna no colchete do DataFrame.

É possível fazermos deleções também, e para isso utilizamos o drop para remover colunas, por exemplo, mediante o comando:

```
python


 ```
 df = df.drop(columns=['col1', 'col2', 'col3'])
 ```
```

Ou

```
python


 ```
 df.drop(columns=['col1', 'col2', 'col3'], inplace=True)
 ```
```

Também podemos deletar a célula e para isso basta associar o valor None à célula localizada com a ajuda do iloc, ou qualquer outra forma de projeção. Já para deletarmos linhas, basta passarmos os indexes:

```
python


 ```
 df.drop([0,1], inplace=True)
 ```
```

Nesse caso, deletamos as duas primeiras linhas do DataFrame.

Como você pôde observar, utilizamos muito os indexes para nossas manipulações. Para que possamos regularizar a indexação depois de um drop, ou seja, reindexar, basta fazer:

```
python


 ```
 df = df.reset_index(drop=True)
 ```
```

Ou

```
python


 ```
 df.reset_index(drop=True, inplace=True)
 ```
```

O index, assim como nos bancos de dados, tem os papéis de enumerar o conjunto e facilitar a manipulação dos dados em questão. Ao resetar o DataFrame, é como se estivéssemos desfragmentando a memória do DataFrame.

Assim como nos bancos de dados, temos também as operações de junção, feitas pelo concat, merge e join.

Exemplo com concat:

```
python


 ```
 s1 = pd.Series(['a', 'b'])
 s2 = pd.Series(['c', 'd'])
 pd.concat([s1, s2], axis=1)
 ```
```

Esse exemplo realiza a junção entre as séries pelos seus indexes. Basta definir o axis=1, pois normalmente o concat, quando não declarada a axis ou usando axis=0, fará uma união entre os dados. Após um concat, pode ser feito um reset_index também, mas podemos encurtar esse passo declarando dentro do pd.concat() o parâmetro ignore_index=True.

Exemplo com merge:

```
python


 ```
 df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'],
                     'value': [1, 2, 3, 5]})
 df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'],
                     'value': [5, 6, 7, 8]})
 print(df1.merge(df2, left_on='lkey', right_on='rkey', suffixes=(“_left”,”_right”)))
 ```
```

Nesse exemplo, podemos ver o jeito bem tradicional de junção, em que declaramos as chaves estrangeiras que serão a base da junção e declaramos os sufixos em caso de repetição de nome de colunas, que é o caso da coluna value, na qual teremos values da esquerda e da direita.

Exemplo com join:

```
python


 ```
 df = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3', 'K4', 'K5'],
                    'A': ['A0', 'A1', 'A2', 'A3', 'A4', 'A5']})
 other = pd.DataFrame({'key': ['K0', 'K1', 'K2'],
                       'B': ['B0', 'B1', 'B2']})
 df.join(other, lsuffix='_caller', rsuffix='_other')
 ```
```

Comentário

O join é feito a partir de um dos DataFrames, assim como o merge, mas ele depende do index, e a junção é feita no index padrão.

É possível reindexar ou alterar o index de um DataFrame para ser uma das outras colunas. Basta fazer o df.set_index(“nome da coluna”).

Groupby e apply

Agora veremos como fazer agregações em DataFrames, que são as operações mais utilizadas para a análise dos dados, bem como para a extração de estatísticas básicas, possibilitando uma compreensão holística do conjunto de dados. Ao agregar dados, estamos reduzindo o conjunto e resumindo-o a métricas, estatísticas e agrupamentos.

Para agruparmos, é simples. Basta chamarmos o método **groupby** a partir do DataFrame a ser analisado. Fazemos isso da seguinte forma:

```
python


 ```
 df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',
                               'Parrot', 'Parrot'],
                    'Max Speed': [380., 370., 24., 26.]})
 grouped = df.groupby(['Animal'])
 print(grouped.mean())
 ```
```

Aqui, podemos perceber como ocorre a agregação dos dados. O método groupby aceita uma lista de colunas, que serão os agrupadores, mas só chamar o método de groupby não fará as agregações desejadas. O que precisamos para fazer a agregações é chamar o método direto, como no exemplo mean(), que calcula a média de todas as colunas numéricas agrupadas pelo tipo de animal.

> Podemos fazer mais de uma agregação ao mesmo tempo, bastando passar o método agg(['mean']), que recebe uma lista de agregações cobertas na documentação do método no Pandas.

Uma coisa interessante, que não acontece nos bancos de dados tradicionais, é que o index das agregações passa a ser o agrupamento. Para que isso não ocorra, basta usar o reset_index() no resultado da agregação, com drop=False ou não declarado, para que os indexes não sejam deletados e você perca colunas no DataFrame da agregação.

Comentário

A operação apply, que é um método do DataFrame e das series, nada mais é do que o equivalente à função map de outras linguagens como Java e Javascript, em que definimos uma função e a passamos nos componentes de uma lista a fim de modificá-la.

Nos DataFrames, isso pode ser feito pelas colunas individualmente ou pelas linhas, bastando mudar o atributo axis na chamada. Por exemplo:

```
python


 ```
 df = pd.DataFrame([[4, 9]] * 3, columns=['A', 'B'])
 df.apply(np.sqrt)
 df['A'] = df['A'].apply(lambda x: x['A']+1)
 df['C'] = df.apply(lambda x: x['A']+x['B'], axis=1)
 ```
```

Nesse exemplo, criamos um DataFrame sintético de três linhas, com cada linha tendo os valores 4 e 9. No primeiro apply, queremos a raiz quadrada do DataFrame inteiro, mas apenas para visualização, não salvando na variável. No segundo apply, estamos modificando a coluna A para que seja acrescida de uma unidade. E, por fim, criamos uma nova coluna para ser a soma das colunas A e B.

Manipulação de dados com Pandas

No vídeo a seguir, abordamos as técnicas de manipulação de dados para análise no Python.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Vem que eu te explico!

Os vídeos a seguir abordam os assuntos mais relevantes do conteúdo que você acabou de estudar.

Manipulação de dados em DataFrames

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Groupby e apply

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

Qual a função do index de um DataFrame?

A

Servir de base de somatórios dos dados para checar a consistência por meio de checksum.

B

Apenas ordenar as linhas da tabela por meio de índices e servir como orientação visual.

C

Sua função é meramente estética, não tendo nenhuma outra atribuição.

D

Indexador do DataFrame, facilita o acesso aos registros, otimizando consultas.

E

Indicador de performance do conjunto de dados analisado a partir de índice.

A alternativa D está correta.

O index é como se fosse a chave primária e índice de um DataFrame, permitindo fazer junções e agregações a partir dele.

Questão 2

O que acontecerá com a coluna C no comando “df['C'].apply(lambda x: x*x)”, uma vez que ela não tem valores nulos e é toda composta por números de ponto flutuante?

A

O resultado será dividido por ele mesmo.

B

O resultado obtido será a raiz quadrada dos valores.

C

Os valores da coluna serão transformados em strings.

D

Os valores da coluna serão deletados.

E

Os valores da coluna serão elevados à potência de 2.

A alternativa E está correta.

O método apply funciona como a função map de outras linguagens, passando por cada valor e executando a função mapeada e declarada no apply, que neste caso é a multiplicação de x por ele mesmo, ou seja, potência de 2.

Tipos de dados

No mundo de Big Data, temos dados os mais variados possíveis, que isoladamente podem não significar muito, mas com o devido processamento, podemos extrair informação útil e conhecimento para tomada de decisões. Neste módulo, para fins de visualização de dados, vamos nos ater à seguinte classificação: dados numéricos, dados categóricos e dados temporais.

Dados numéricos

São aqueles cujos valores são em números reais, racionais ou naturais, que expressam quantidades, proporções, valores monetários, métricas. Esses números são tipicamente passíveis de operações e cálculos matemáticos.

Dados categóricos

São aqueles normalmente expressos por texto, que representam rótulos, símbolos, nomes, identificadores.

Dados temporais

São aqueles que passam a ideia de série, cronologia, fluxo de tempo. Exemplos de dados temporais são aqueles associados a datas, aos dias da semana, índices ordinais, séculos, meses, anos, horas etc.

Atenção

Podemos ter dados categóricos expressos por números, por exemplo, notas de provas significando conceitos (que podem ser substituídos pelo sistema de letras A, B, C, D, F) e ordinais representando categorias de posição ou ordem. Podem ocorrer dados numéricos que, quando estratificados, passam a ser categóricos como idade, temperatura etc.

Tipos de visualizações

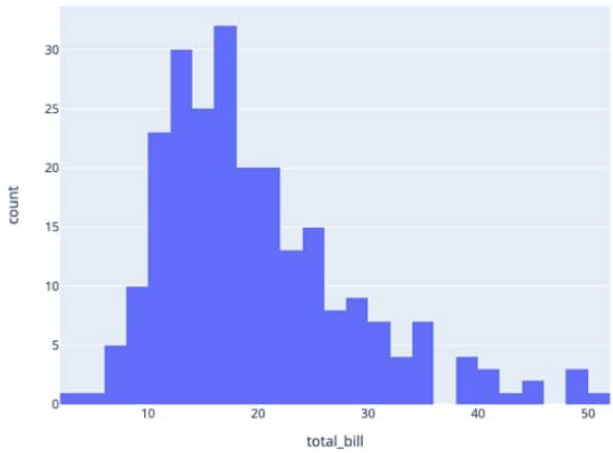
Para cada tipo de dado, temos visualizações mais adequadas ou que ressaltam melhor o significado por trás do conjunto de dados.

Para dados numéricos, usualmente queremos que sejam demonstradas suas proporções, distribuições e correlações. Quando queremos descobrir como é o formato de distribuições para entendermos possíveis padrões implícitos em dados numéricos, utilizamos os gráficos de histograma.

O código a seguir gera um histograma usando o **plotly.express**, uma API para criação de gráficos no Python que contém diversos datasets de testes, identificados por data.xxx().

```
python


 ```
 import plotly.express as px
 df = px.data.tips()
 fig = px.histogram(df, x='total_bill')
 fig.show()
 ```
```



Histograma da distribuição de dados de comandas de restaurante (data.tips) - resultado da execução do código anterior.

Também podemos querer saber a correlação entre dados numéricos e, para isso, podemos utilizar o gráfico de dispersão, scatterplot, como no exemplo a seguir:

```
python


 ```
 import plotly.express as px
 df = px.data.iris()
 fig = px.scatter(df, x='sepal_width', y='sepal_length', color='species',
symbol='species')
 fig.show()
 ```
```

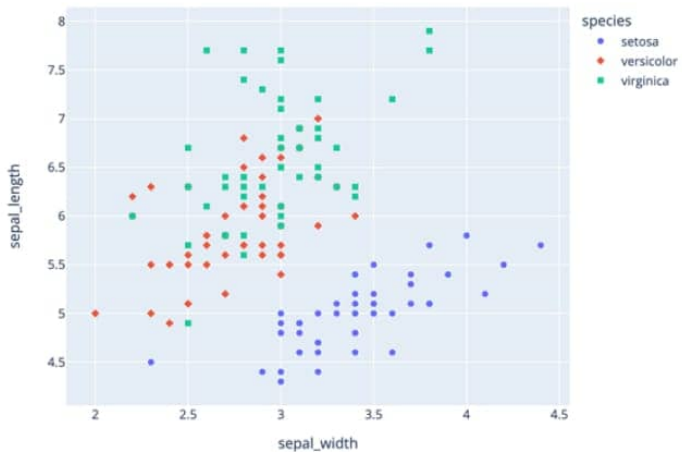


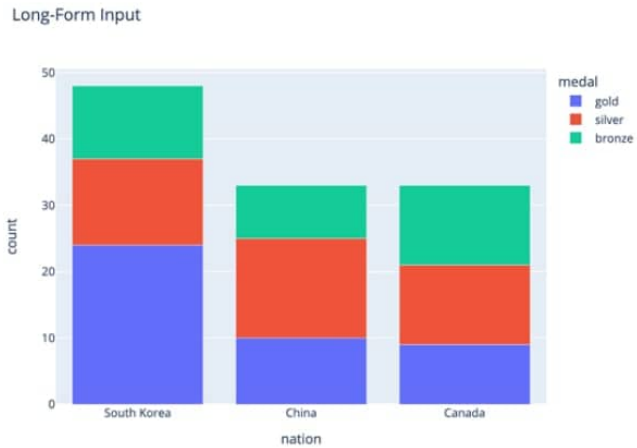
Gráfico de dispersão de larguras e comprimentos de sépala de flores (data.iris) - resultado da execução do código anterior.

No gráfico, podemos ver a relação entre as medidas, agregadas pela espécie.

Para dados categóricos, queremos mostrar as categorias e suas proporções, ou colorir outros gráficos como fizemos no scatterplot anteriormente. Por exemplo:

````
python


```
import plotly.express as px
long_df = px.data.medals_long()
fig = px.bar(long_df, x='nation', y='count', color='medal', title='Long-Form Input')
fig.show()
```
````



Exemplo de quadro hipotético de medalhas da China, Coreia do Sul e Canadá em um gráfico de barras (data.medals_long) - resultado da execução do código anterior.

Comentário

Como podemos ver, o gráfico de barras é muito parecido com o histograma; a diferença é que o gráfico de barras mostra a contagem e proporções de dados categóricos, ou seja, o eixo X será de dados categóricos e o Y representará a contagem (podendo inverter os eixos quando estamos olhando gráficos de barras orientados horizontalmente).

Outro exemplo para entendermos proporções das categorias é o gráfico de pizza ou torta (pie).

```
python


 ```
 import plotly.express as px
 df = px.data.tips()
 fig = px.pie(df, values='tip', names='day')
 fig.show()
 ```
```

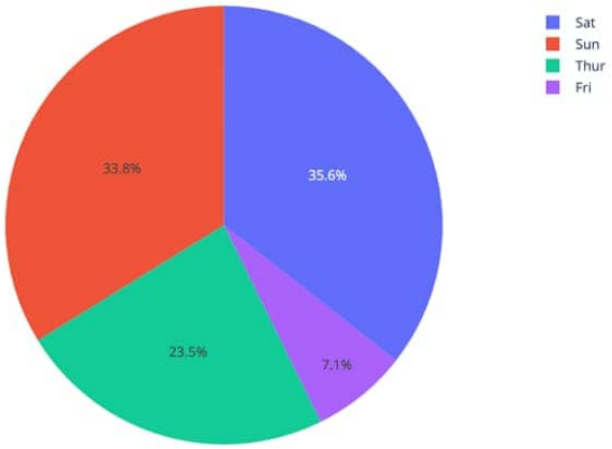


Gráfico de pizza mostrando a proporção dos valores distintos de dias da semana no conjunto de comandas (data.tips) - resultado da execução do código anterior.

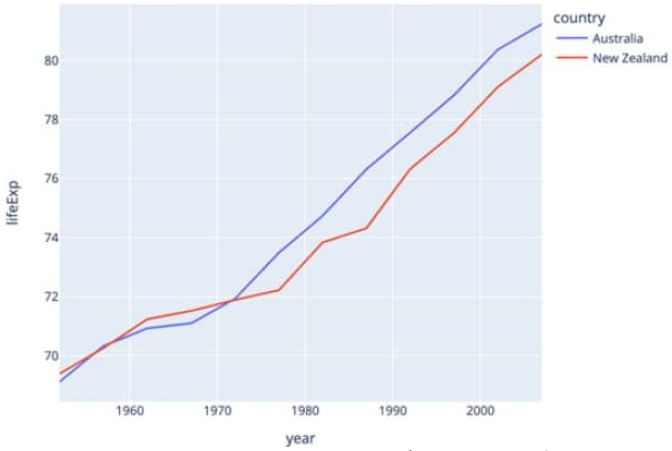
No gráfico de pizza, podemos tanto ver que o conjunto de dados só tem quatro dias distintos como também suas proporções, mostrando que o restaurante/lanchonete do conjunto de dados de comandas tem maior funcionamento nos fins de semana.

> O gráfico de pizza é adequado quando temos poucas categorias, pois, se ele tiver muitas fatias a exibir, ou seja, muitos valores únicos, passa a ficar ilegível, sendo preferível migrar para o gráfico de barras.

Finalmente, vamos falar dos dados temporais, aqueles que passam ideia de cronologia. Uma solução natural é representarmos tais dados a partir do gráfico de linhas. Por exemplo:

```
python


 ```
 import plotly.express as px
 df = px.data.gapminder().query('continent=='Oceania'')
 fig = px.line(df, x='year', y='lifeExp', color='country')
 fig.show()
 ```
```



Expectativa de vida ao longo dos anos para Austrália e Nova Zelândia (data.gapminder) - resultado da execução do código anterior.

Aqui podemos perceber bem a ideia de cronologia, com o eixo X apresentando o dado temporal (anos, meses, datas completas, horas ou mesmo o index do DataFrame). Assim, será possível entender a evolução da coluna analisada. Como podemos ver, foi feita uma coloração diferente por país, um dado categórico, isto é, uma agregação/agrupamento.

> ⚠
>
> Atenção
>
> Temos que tomar cuidado para os valores únicos dessas agregações não prejudicarem a visualização.

Biblioteca Plotly

Para elaborarmos todas essas visualizações escolhemos trabalhar com o Plotly, uma biblioteca de visualização do Python muito adequada ao ambiente do Jupyter Notebook, uma vez que suas visualizações são interativas, podendo passar com o mouse por cima dos gráficos e enxergar dados com mais detalhes, bem como operações de zoom, pane, export etc.

Sua instalação é bem simples, basta utilizarmos o comando pip:

````
python

 ```
 pip install plotly
 ```
````

E para usarmos os métodos nos nossos códigos, precisamos importar a biblioteca:

```
python


 ```
 Import plotly.express as px
 ```
```

Depois escolhemos uma visualização que mais se adeque aos dados a serem observados, sendo que o catálogo de visualizações da biblioteca é bem vasto e disponível em sua documentação.

Por exemplo, caso você queira algo diferente, use um bubble chart:

```
python


 ```
 import plotly.express as px
 df = px.data.gapminder()
 fig = px.scatter(df.query('year==2007'), x='gdpPercap', y='lifeExp',
                  size='pop', color='continent',
                   hover_name='country', log_x=True, size_max=60)
 fig.show()
 ```
```

Gráfico de bolhas da expectativa de vida pela renda per capita agregado por país, em que o tamanho da bolha é dado pela população do país - resultado da execução do código anterior.

Visualização de dados no Python

No vídeo a seguir, abordamos as técnicas de visualização de dados para análise no Python.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Vem que eu te explico!

Os vídeos a seguir abordam os assuntos mais relevantes do conteúdo que você acabou de estudar.

Tipos de visualizações

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Biblioteca Plotly

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

Para que serve o Plotly?

A

É uma biblioteca para gestão de arquivos do Python.

B

É uma biblioteca para criação de modelos de aprendizado de máquina do Python.

C

É uma biblioteca para manipulação de dados do Python em DataFrames.

D

É uma biblioteca para gerar visualização de dados interativa do Python.

E

É uma biblioteca para o split de dados do Python.

A alternativa D está correta.

O Plotly é a biblioteca do Python alternativa ao matplotlib e seaborn, capaz de gerar visualizações de dados de forma interativa e muito adequada para jupyter notebooks, que têm esse caráter mais exploratório. As demais alternativas se referem a outras bibliotecas do Python.

Questão 2

Quais são as categorias de dados que pensamos na hora de fazer visualizações na análise de dados?

A

Irracionais, complexos e temporais.

B

Numéricos, rotulados e cronológicos.

C

Numéricos, categóricos e quânticos.

D

Numéricos, categóricos e temporais.

E

Imaginários, categóricos e temporais.

A alternativa D está correta.

A classificação tradicional dos dados na hora da elaboração de visualizações é: dados numéricos, que nos levam a histogramas e scatterplots; dados categóricos, que nos levam ao gráfico de pizza ou de barras; e dados temporais, que nos levam aos gráficos de linha.

Considerações finais

Como vimos, a linguagem Python pode ser uma ferramenta fundamental para a análise de dados. Esta, por sua vez, é essencial para iniciarmos bem qualquer projeto de um sistema de informação na área de aprendizado de máquina e ciência de dados.

Neste conteúdo, entendemos como funciona a linguagem Python, algumas de suas peculiaridades, sua sintaxe e seus constructos. Vimos que podemos utilizar bibliotecas poderosas de análise e manipulação de dados como Pandas.

Fomos capazes de compreender o conceito de dado de painel, bem como entendermos os DataFrames do Pandas. Compreendemos e aplicamos os métodos de manipulação de dados disponíveis nos DataFrames como projeção, deleção e junção.

Aprendemos como é feita a coleta e o pré-processamento de dados, culminando na visualização, que é a concretização do processo de análise ou um recomeço, dependendo dos achados.

Podcast

Ouça no podcast a entrevista sobre análise de dados no Python.

Conteúdo interativo

Acesse a versão digital para ouvir o áudio.

Explore +

Para dar sequência ao seu estudo, teste mais formas de gerenciar o Python e suas versões com o Pyenv.

Estude o pipenv, pois é uma excelente maneira de gerenciar projetos Python, muito usado no mercado.

No Pandas, tente explorar outros métodos, uma vez que a biblioteca é muito vasta, permitindo integrações diretas até mesmo com bancos de dados.

Além do Plotly, temos o matplotlib e o seaborn, sendo recomendada a pesquisa dessas bibliotecas de visualização de dados.

Referências

AMARAL, F. **Aprenda mineração de dados:** teoria e prática. Rio de Janeiro: Alta Books, 2016.

AZEVEDO, A. I. R. L.; SANTOS, M. F. **KDD, SEMMA and CRISP-DM:** a parallel overview. IADS-DM, 2008.

DA SILVA, F. C. D.; GARCIA, A. C. B. **Judice Verum, a Methodology for Automatically Classify Fake, Sarcastic and True Portuguese News.** Dissertação (Mestrado em Informática) – Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2019.

KABIR, S. M. S. **Methods Of Data Collection.** *In:* **_____. Basic Guidelines for Research:** An Introductory Approach for All Disciplines. 1. ed. Bangladesh: Book Zone Publication, 2016, p. 201-275.

WIRTH, R.; HIPP, J. **CRISP-DM: Towards a standard process model for data mining.** *In:* Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining. London, UK: Springer-Verlag, 2000.