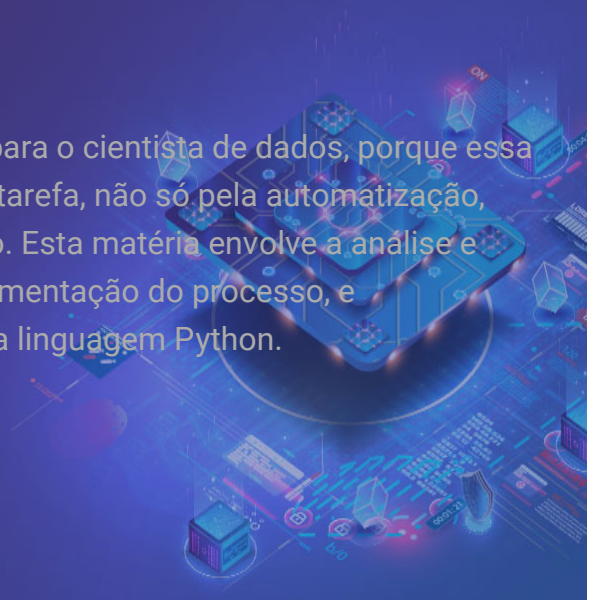
Tema 4

Análise de Dados em Python com Pandas

Entender a análise de dados em Python é fundamental para o cientista de dados, porque essa é uma das linguagens mais utilizadas para esse tipo de tarefa, não só pela automatização, mas também pela robustez de suas bibliotecas de apoio. Esta matéria envolve a análise e preparação de dados, incluindo pré-processamento e alimentação do processo, e apresentação visual de resultados usando bibliotecas da linguagem Python.



Tempo total de leitura
126 min.

Créditos

F
Professor (a)
FERNANDO DURIER

Introdução

Você sabia que a análise de dados é fundamental para a iniciar qualquer projeto de sistemas de informação?

Afinal, é esse tipo de análise que nos traz insights de como desenhar melhor o processo, além do que nos foi entregue no levantamento de requisitos.

Uma das linguagens mais utilizadas na automatização da análise de dados é o Python.

Aqui, você vai 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 suas análises.

Preparação

Antes de iniciar o conteúdo, você vai precisar criar uma conta no Gmail para acessar o Google Colab e executar o exemplo prático na linguagem Python.

Objetivos

Ao final desta aula, você será capaz de:

- Descrever a preparação de dados para análise no Python.
- Aplicar o tratamento de dados para análise no Python.
- Descrever a manipulação de dados no Python.
- Aplicar a visualização de dados em Python.

01. Análise de dados

Introdução aos componentes e sintaxe de Python

Histórico da linguagem Python

A linguagem Python é chamada de linguagem de alto nível devido a sua abstração com relação ao hardware e aos registros, diferentemente 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-fonte, poderia entendê-los facilmente, independentemente de ser especialista na área.

Curiosamente, por mais que o símbolo da linguagem seja o da serpente constritora não venenosa Python (Píton), a origem do nome da linguagem refere-se ao grupo humorístico britânico Monty Python, criador da série cômica *Monty Python's flying circus* (Circo voador de Monty Python).

Logo da linguagem Python.

Tipagem de dados

Diferentemente de linguagens de médio nível, como C e Java, Python não precisa declarar o tipo de suas variáveis, o que também se alinha 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, de 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, 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:

str, unicode

Cadeia de caracteres.

Ex.: “batata”, u“alface”

list

Lista.

Ex.: [1,2,3], ['a','b', 'c'], [1.0, 'a', True]

Tuple

Tupla.

Ex.: (“what”, ”who”, ”whom”, ”where”, ”when”)

set, frozenset

Conjunto não ordenado.

Ex.: set([1,2,3]), frozenset(['batata','alface','uva'])

Dict

Dicionário, conjunto chave-valor.

Ex.: {“a”:1,”b”:2,”c”:3}, {“k1”:”a”,”k2”:”b”,”k3”:”c”}

Int

Número inteiro (se muito grande, será convertido em Long).

Ex.: 42, 50, 100, 1, 2, 3, 78394024920L

float

Número de ponto flutuante ou racional.

Ex.: 3.7, 4.55, 9.012, 9.18293, 10.1

complex

Número complexo.

Ex.: 1e10, 3i, 7+4j

bool

Booleano.

Ex.: True, False

!=

Diferente.

Ex.: Diferente de, <>, !==

Controles de execução

Em Python, também há construções clássicas de controle de execução de comandos, como na maioria das linguagens de programação. Veja alguns exemplos:

Constructo condicional

O constructo de condição **if**, como em outras linguagens de programação, executa uma ação quando a condição é atendida. Para executar uma ação alternativa, caso a condição inicial não seja atendida, utilizamos o **else**, ou seja, **se** essa condição, então faça isso, **senão** faça isso. Podemos ter diversas condições no mesmo fluxo, mas diferente de outras linguagens que utilizam o switch, em Python utilizamos **elif.** Para entender melhor como essa dinâmica acontece na prática, veja os exemplos a seguir.

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")
```

Constructo repetição

O constructo de repetição ou laço é essencial em todas as linguagens de programação. Em Python, temos o **for** (para quantidade de repetições conhecida) e o **while** (para quantidade de repetições variada). Dessa forma, quando queremos repetir uma ação por um número previsto de vezes, como um número exato ou o tamanho de uma lista, utilizaremos o **for**, mas quando desejarmos repetir uma ação até que uma determinada condição seja atendida, enquanto um programa estiver ligado ou até uma entrada específica de dado, por exemplo, utilizaremos o **while**. Veja os exemplos.

for - para:

```
S = [1, 2, 3]
for i in range (len(S)):
  print(“Oi”)
```

while - enquanto:

```
i = 0
S = [1, 2, 3]
while (i + 1 < len(S)):
  print(“Oi”)
  i +=1
```

Constructo classe

Uma **class** (classe) pode ser entendida como uma representação de algo do mundo real, que pode ter características chamadas de atributos e desempenhar ações, chamadas de métodos. Veja os dois exemplos a seguir. O primeiro (SGDRegression) apresenta a estrutura básica de uma classe, já no segundo exemplo temos a declaração de uma classe (Pokemon) com métodos e atributos.

Class - classe:

```
Class SGDRegression:
  def __init__():
```

```
    pass()

Class Pokemon:
  def __init__(self, type1, type2):
    self.type1 = type1
    self.type2 = type2
  def attack(self, name, type, damage):
    print(name, “attack!!”)
    print(damage, “damage.”)
```

Constructo funções/rotinas

Uma função ou rotina é um conjunto de instruções que executam uma determinada tarefa. Uma mesma função pode ser chamada diversas vezes em diferentes pontos do código, propiciando uma melhor organização e reaproveitamento do código. No exemplo a seguir, temos uma função para somar dois números que será chamada dentro do laço. Dessa forma, ao ser chamada pela primeira vez, a função somará 1 e 2 imprimindo o resultado 3 na tela. Já ao ser chamada pela segunda vez, a função somará 3 e 4 imprimindo o resultado 7 na tela.

def - função ou rotina:

```
def soma(x, y):
  return x+y

lista = [(1, 2), (3, 4)]
for l in lista:
  print(soma(l[0], l[1])
```

Constructo escopo

O constructo de escopo limita uma tarefa ao escopo declarado. Isso permite uma redução de código, ao passo que, ao limitar a ação ao escopo específico,

descarta-se a necessidade do tratamento de exceções que seria necessário caso aquela ação não possa ser executada. No exemplo a seguir, armazenamos os dados contidos no arquivo query.sql em sql_query. Se os dados não puderem ser gravados, a execução do sistema continuará sem problemas, enquanto a não utilização do with exigiria um tratamento de exceções ou a execução seria interrompida.

With - dado que, ou no escopo de:

with open(“./query.sql”) as file:
  sql_query=file.read()

Modularização

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

Podemos importar o bloco lógico em outra parte do projeto. Para isso, podemos utilizar o caminho absoluto ou o caminho relativo. A seguir, temos um exemplo com o caminho absoluto:

Python

```
1  from module import *
```

Caminho absoluto.

Ao utilizarmos o caminho relativo precisamos observar se o bloco lógica está localizado na mesma pasta ou em pasta diferente do arquivo que estamos editando. Veja as duas formas a seguir:

Python

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

Caminho relativo na mesma pasta onde a parte é chamada.

Python

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

Caminho relativo nas duas pastas da chamada original.

Você pode 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__.

Quando importamos um módulo, é como se importássemos uma classe para outra parte do projeto. 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 entendê-lo como o construtor do pacote/módulo ali declarado.

Atividade

Questão 1

Imagine que você instanciou uma função auxiliar no mesmo projeto, no mesmo nível do arquivo que chamará essa função, mas na pasta utils, em um arquivo chamado preprocessing.py. Você deve chamar essa função por meio de qual comando de import do Python?

A Import preprocessing

B Import utils

C From preprocessing import utils

D from .utils import preprocessing

E from .utils

Parabéns! A alternativa D está correta.

Como o arquivo está em uma pasta no mesmo nível, começamos com ponto (.), estabelecendo, assim, um caminho relativo. Isso é sempre bom para reaproveitar o projeto. Em seguida, importamos (import) a pasta em que está o arquivo python (utils) e, depois, o arquivo que cedia a função (preprocessing). Agora, o arquivo que está chamando tem acesso aos métodos descritos em preprocessing.py, entre eles essa função.

Estrutura de projetos e boas práticas em Python

Importação de bibliotecas

Assim como em outras linguagens, podemos importar bibliotecas criadas por outras pessoas. Essa é uma das grandes vantagens da computação hoje em dia, pois a comunidade de desenvolvedores é grande, diversa e muito colaborativa..

Por mais que estejamos importando módulos externos, teremos acesso à documentação e a trilhas de discussões em plataformas como as seguintes:

Stack Overflow

GitHub

Reddit

Stack Overflow e GitHub. Mas como fazemos isso? É simples: por meio do gerenciador de pacotes do Python – o **pip.** Na maioria das vezes, basta digitarmos no terminal ou prompt de comando: pip install . Assim, importamos em nosso código como **import <nome da biblioteca>**, e teremos acesso a seus módulos e métodos.

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

Python

```
pip freeze > requirements.txt
```

Comando freeze.

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

Python

```
pip install -r requirements.txt
```

Comando para instalar bibliotecas registradas em 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, 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: **venv**. Para sair, basta digitar: **Deactivate.**

Existem alternativas, a saber:

Pyenv e pipenv

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.

Jupyter Notebook (ambiente de scripts Python)

Pode ser instalado no nível global, fora de virtualenv, por meio do comando: pip install jupyter. Para abrir um servidor local digitamos no terminal jupyter notebook na pasta do projeto. O servidor será levantado, e 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

Qualquer sistema de informação tem como entrada dados que serão processados por 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 usados localmente.

> Para lidarmos com os dados, deveremos criar uma pasta no nível da raiz do projeto para armazenamento dos arquivos. Por convenção, podemos chamá-la de assets (ativos).

Essa pasta pode 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.

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

Biblioteca Pandas

Na maioria dos casos, a manipulação de dados em Python é feita com **DataFrames**, pois essas estruturas são muito mais cômodas e robustas de se trabalhar do que matrizes. Antes de falarmos dos DataFrames, precisamos explicar a **biblioteca Pandas.**

> Essa biblioteca em Python foi criada para manipulação e análise de dados. Ela

> oferece estruturas e operações para manipular tabelas numéricas e séries temporais.

Cada coluna de um DataFrame, que é a principal estrutura do Pandas, é um dado do tipo Series ou série temporal. Veja um exemplo:

	First Name	Gender	Start Date	Last Login Time	Salary
92	Linda	Female	5/25/2000	5:45 PM	119009
65	Steve	Male	11/11/2009	11:44 PM	61310
445	Chris	Male	12/12/2006	1:57 AM	71642
732	Henry	Male	5/12/1986	2:04 AM	59943
352	NaN	Male	10/9/2011	9:29 AM	69906
293	Jesse	Male	10/25/1999	3:35 PM	118733
456	Deborah	NaN	2/3/1983	11:38 PM	101457
171	Patrick	Male	8/17/2007	3:16 AM	143499
562	Sara	NaN	10/7/1983	1:35 PM	87713
320	NaN	Female	7/8/2008	11:40 PM	62960
568	Susan	Female	4/18/1986	9:31 AM	90829
775	Rose	Female	11/3/1999	9:06 AM	75181
32	NaN	Male	8/21/1998	2:27 PM	122340

Pandas DataFrame.

Os DataFrames são 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.

Curiosidade

Ao contrário do que possa parecer, o nome Pandas não vem do urso monocromático da China, mas sim do conceito de panel data sets (dados em painel). É um termo estatístico relativo a conjuntos que incluem diferentes unidades amostrais acompanhadas ao longo do tempo.

Atividade

Questão 1

Quando estamos desenvolvendo diversos projetos em Python, é comum utilizarmos diferentes versões de uma mesma biblioteca entre esses projetos. Essa tarefa é muito complexa para o sistema operacional gerenciar e pode gerar conflitos entre as versões.

O mais comum é utilizarmos diferentes ambientes virtuais, chamados de virtualenvs – um para cada projeto. Nesse contexto, quais são os comandos para levantar e sair de um ambiente virtual Python?

A start; quit

B startvenv; !venv

C venv; ~venv

D venv; deactivate

E activate; deactivate

Parabéns! A alternativa D está correta.

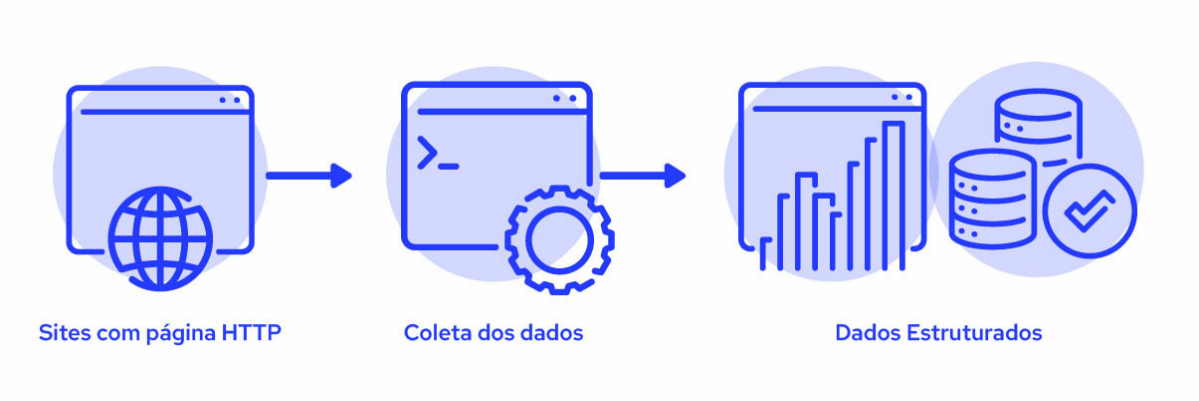
O comando venv levanta o ambiente virtual em Python. Quando não desejamos mais trabalhar ali dentro, basta digitarmos deactivate para sair. As demais alternativas possuem comandos não suportados pela linguagem Python.

Preparação de dados para análise no Python

Coleta e conjunto de dados

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

A ilustração a seguir representa esse processo:



Processo de coleta de dados.

As informações coletadas fazem parte de um conjunto de dados que podem ser:

Qualitativos

São, em sua maioria, dados não numéricos, normalmente descritivos, e apresentam-se em formato de textos. 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. É um bom jeito de mapear o funcionamento de um sistema ou a razão de um fenômeno. Um exemplo para obter dados qualitativos é a avaliação de sentimentos em relação a algum serviço ou produto como: muito bom, bom e ruim.

Quantitativos

São os dados numéricos, que podem ser matematicamente computados. Essa categoria de dados mede diferentes escalas: nominais, ordinais, intervalares e proporcionais. Na maioria dos casos, esses dados resultam da medição de algum aspecto ou fenômeno. Entre os métodos dessa categoria, temos os surveys, as queries, o consumo de relatórios e os escavadores (scrapers). Alguns exemplos de dados quantitativos incluem dados numéricos, receita bancária, valores depositados, quantidade de dados trocados entre servidores etc.

Primários

São os dados coletados de primeira mão, ou seja, aqueles que ainda não foram publicados, que são autênticos ou inéditos. Como são dados recém-coletados, não têm interferência humana. Por isso, são considerados mais puros do que os secundários. Entre as fontes de dados primários, temos surveys, experimentos, questionários e entrevistas. Por exemplo, imagine que você construiu um sistema de coleta de logs de operação em um hospital. Cada registro é uma atividade de um médico ou enfermeiro sobre determinado paciente. Isso é um conjunto de dados primários.

Secundários

São dados que já foram publicados de alguma forma, ou seja, que sofreram alguma interferência humana. Por exemplo, ao fazermos a revisão de literatura em qualquer estudo, estamos revisando esses tipos de dados. Entre as fontes de dados secundários, temos livros, jornais, censos, arquivos de dados etc. Esses dados são muito úteis quando não conseguimos fazer coleta em primeira mão. Para entender melhor, imagine que você reproduziu um estudo de um trabalho da literatura. Nesse caso, você teria de utilizar os dados (secundários) fornecidos pelos autores desse trabalho.

Tratamento de dados nulos ou corrompidos

Um problema muito comum na atividade de pré-processamento de dados é a qualidade deles. É possível que os dados venham com atributos faltantes, registros nulos, mal escritos (desformatados) etc.

No caso de dados repetidos, a solução é 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. Para resolver isso, é sempre útil estudar os metadados do conjunto de dados, se estiverem disponíveis, ou estudar a origem deles.

Já quando o problema é dados faltantes ou nulos, a estratégia mais indicada para resolver o problema varia de acordo com o tamanho da base tratada, como veremos a seguir:

Grandes

No casos de dados faltantes ou nulos em bases grandes, podemos resolvê-los ignorando o registro todo, ou seja, removendo-o da base, se a proporção de nulos não for expressiva (até 10% de registros). Outra estratégia é utilizar técnicas de regressão para dados numéricos ou de classificação para dados categóricos, para o preenchimento automático. O fato de a base ser grande ajuda o algoritmo de preenchimento automático. É claro que dados com variância alta podem prejudicar esse processo.

Restritas

Nos casos de dados faltantes ou nulos em bases muito restritas ou pequenas (1.000 dados ou menos) podemos optar entre duas alternativas: a primeira é tentarmos preencher os dados de forma automática, já a segunda é voltarmos ao processo de coleta de dados e tentamos melhorá-lo, a fim de consertar o problema que causou a nulidade ou falta deles.

Precárias

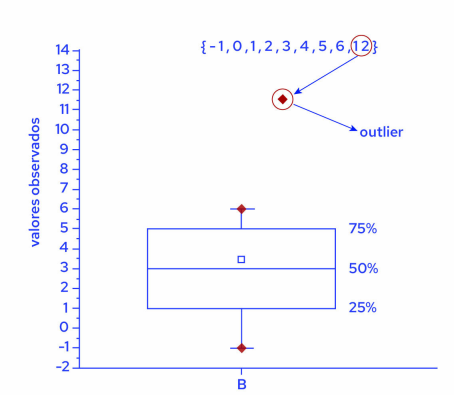
Nos casos de dados faltantes ou nulos em bases de dados precárias devemos retomar diretamente ao processo de coleta nos casos em que a exclusão do registro ou a interpolação é inviável, pois, claramente, os dados são insuficientes para o projeto.

Regularização de dados

Além de poderem ter sido corrompidos ou simplesmente faltarem, os dados coletados 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], dizemos que 50 e 100 são outliers, pois são muito distantes do restante. Essa perturbação pode causar problemas nos processamentos do sistema sobre esse conjunto, levando-o a demonstrar comportamentos indesejados.

Veja um exemplo:



Ponto fora da curva.

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

O método min max consiste em colocar os dados em escala com o mínimo valor do atributo no conjunto contra o máximo valor. Aplicando a fórmula a seguir, o dado observado é inserido em uma régua que começa no mínimo valor possível e vai até o máximo, transformando o d_observado em um valor entre 0 e 1. Veja:

$$\text{d\_observado (d\_max-d\_min)}$$

Atividade

Questão 1

Imagine que você está diante de uma planilha de vendas de uma loja de roupas. Na coluna Parcelas, com valores que variam entre [1,2,5,10,12], faltam alguns dados na proporção de 5/1000 linhas em branco. Considerando uma distribuição desbalanceada de dados e de inteiros na coluna, qual é a melhor estratégia para esse caso?

A Remover a coluna.

B Rodar um processo de autopreenchimento.

C Remover as linhas.

D Embaralhar o dataset.

E Preencher com “nada”.

Parabéns! A alternativa C está correta.

Quando a proporção de dados faltantes é pequena, quase insignificante, e quando a coluna é desbalanceada ou muito variada, a melhor ação é remover as linhas irrelevantes. A alternativa E só funcionaria se preenchesse os valores faltantes com um número, e não uma string.

02. Demonstração com Python e Pandas

Sintaxe de Python

Criando um projeto com fluxos de controle

1. Crie uma pasta vazia no computador para ser o projeto em Python.
2. Instale o Jupyter Notebook usando o comando pip install jupyter.
3. Com o terminal aberto nessa pasta nova, digite Jupyter Notebook e rode com o enter.
4. Ao abrir a ferramenta pelo browser, crie um novo arquivo Notebook ali, nomeando-o como pratica_1.
5. Dentro do Jupyter Notebook, crie variáveis para diferentes tipos de dados.
6. Implemente, ao longo do mesmo Jupyter Notebook, fluxos de controles variados para os diferentes tipos de dados.

Atividade

Questão 1

Considere a lista de números [1.0,2.0,3.0,4.0,5.0] a ser alocada na variável lista _nums. Execute um laço simples percorrendo essa lista e imprimindo o tipo de cada componente dela. Feito isso, crie uma nova variável lista_mista, que receberá o seguinte vetor [1,"gato",2,"cachorro",3,"peixe"]. Em seguida, faça o mesmo laço simples, agora imprimindo apenas os itens do vetor que forem string de caracteres apenas.

Abrir solução

Para a primeira parte desse exercício prático, o resultado deve ser uma sequência de repetições da palavra “float” cinco vezes, e, para a segunda parte, uma sequência de espaço vazio, gato, espaço vazio, cachorro, espaço vazio, peixe. Assim, você pode replicar o fluxo de controle mais comum da linguagem (laço), além de se ambientar com tipagem de dados e controle de tipos.

Veja um exemplo de laço:

```

for n in lista_nums:

 print(“----------”)

 print(n,“:”,type(n))

```

Modularização em Python

Criando um projeto para análise de dados com modularização e pandas

1. Crie uma pasta chamada projeto_modular.
2. Crie um arquivo app.py.
3. Crie uma pasta chamada preprocessing.
4. Crie um arquivo __init__.py na pasta preprocessing.
5. Crie um virtualenv e ativá-lo.
6. Crie uma pasta data.
7. Insira na pasta data o arquivo iris.csv.
8. Instale as bibliotecas Pandas, NumPy e Plotly.
9. Faça o pip freeze para um arquivo requirements.txt.
10. Crie o arquivo normalize.py na pasta preprocessing.
11. Crie chamadas para o dado csv e a função normalize na app.py.
12. Rode a app.py por meio do comando python app.py.

Atividade

Questão 1

Com base no projeto que acabamos de desenvolver, crie um novo arquivo dentro de preprocessing chamado one_hot_encode.py. Dentro dele, crie uma função que pegue os valores categóricos de um dataframe, chame a função pd.get_dummies() e cheque as novas colunas que serão retornadas ao arquivo app.py.

Abrir solução

O resultado dessa prática deve ser um dataframe de 0s e 1s apenas com colunas com nomes para cada espécie de íris no arquivo csv.

Tratamento de dados com a biblioteca Pandas

Tratando dados com Pandas

1. Crie uma pasta chamada pandas_experiment.
2. Crie uma pasta data dentro da pasta do projeto.
3. Insira um arquivo íris.csv dentro da pasta data.
4. Instale as bibliotecas Pandas, NumPy, Plotly e Sklearn na pasta pandas_experiment.
5. Execute o comando Jupyter Notebook no terminal, na pasta do projeto.
6. Crie um novo notebook assim que abrir o servidor do Jupyter no navegador.
7. Clique no novo notebook e o renomeie para teste.
8. Crie uma célula com os imports das bibliotecas baixadas.
9. Leia o arquivo de dados com o Pandas.

10. Execute o comando de fillna em uma próxima célula.
11. Execute a função de normalização em uma próxima célula.
12. Execute a função info em uma próxima célula.
13. Execute, em uma próxima célula, pd.to_csv() com o dataframe resultante, nomeando-o para preprocessed.csv.

Atividade

Questão 1

Com o mesmo projeto base criado, extraia o formato one_hot_encode da classe íris, junte-a ao dataframe novamente e remova a antiga classe de espécie íris original.

Abrir solução

O resultado dessa prática deve ser um dataframe com os mesmos dados da íris, mas sem a coluna classe, que se transformará em três novas colunas com os nomes das espécies preenchidas com 0 ou 1, caso o registro seja da dada espécie.

03. Manipulação de dados no Python

Estrutura de dados

DataFrames X tabelas

Na estrutura de DataFrames, os dados podem ser criados a partir de:

- Arquivos CSV;
- Excel;

- Listas de dicionários;
- Matrizes;
- Junções de listas.

As tabelas são muito similares às estruturas dos bancos de dados. Uma semelhança entre DataFrames e tabelas é o index, que serve para marcar cada linha do DataFrame e ordenar os dados, assim como nas tabelas dos bancos de dados SQL.

Podemos descrever o conteúdo de um DataFrame no Pandas de duas formas: info() e describe. Observe:

Python

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

Instrução que retorna informações sobre o DataFrame.

O DataFrame detalhado pelo info() resulta na descrição de cada coluna e seu tipo, com a contagem de valores não nulos.Veja:

Python

```
<class 'Pandas.core.frame.DataFrame'>
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


```

Instrução que retorna informações sobre o DataFrame.

A alternativa é o describe, executado pelo comando:

Python

```
df.describe()
```

Instrução que retorna informações sobre o DataFrame.

O describe resulta na descrição de estatísticas básicas, como:

- Contagem de cada coluna;
- Quantidade de valores únicos de cada variável;
- Quantidade de categorias;
- Primeiro registro;
- Frequência.

Veja um exemplo:

Python

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

Retorno da instrução df.describe().

Agora, vamos executar a 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 n
```

df.info() e df.describe().

Manipulação de dados em DataFrames

Projeção e seleção

Quando fazemos uma projeção ou uma seleção, queremos criar um subconjunto dos dados originais. Nos DataFrames, podemos usar os métodos **loc**, **iloc** e **query**.

Veremos mais detalhes sobre cada um a seguir.

Método loc

A projeção das colunas do DataFrame é feita pelos rótulos:

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']])
```

Método loc.

Nesse exemplo, criamos um DataFrame artificial e fizemos duas projeções: a primeira com apenas uma coluna ('viper') e a segunda com duas colunas ('viper', 'sidewinder').

Ao selecionarmos apenas uma coluna, teremos uma série (Pandas series). Caso escolhamos mais de uma coluna, além de precisarmos passar o conjunto como uma lista dentro dos colchetes do loc, o retorno será um DataFrame, pois é o coletivo de duas ou mais séries.

Método iloc

A projeção depende dos indexes numéricos das linhas e colunas de um DataFrame. Veja:

Python

```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])

```

Método iloc.

Aqui, observamos:

- O primeiro print nos trará a primeira coluna do conjunto.
- O segundo print nos trará a primeira linha do DataFrame.
- O terceiro print nos trará as primeira e segunda linhas do DataFrame.
- O quarto print nos trará as três primeiras linhas do DataFrame.
- O quinto print nos trará a projeção com a condição de que o valor das células seja par.
- E os dois últimos prints nos projetam uma submatriz (ou filtro) do DataFrame original – o primeiro de forma direta e o segundo por fatiamento.

Método query

Podemos fazer seleções e projeções por queries. Veja um exemplo:

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])

```

Método query.

No primeiro print, temos uma query que retorna dados cujo valor da dimensão A é maior do que o da dimensão B.

O segundo é uma igualdade, em que desejamos os dados cujo valor da dimensão B seja igual ao da dimensão 'C C'. O que for textual tem de ser declarado com o acento grave (`).

Um último jeito de filtrar é passando a referência da coluna no colchete do DataFrame.

Deleção

Para remover colunas, podemos optar por entre dois procedimentos utilizando o drop:

Python

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

Deleção de colunas - procedimento 1.

Python

Python

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

Deleção de colunas - procedimento 2.

Podemos deletar a célula associando o valor None à célula localizada com a ajuda do iloc ou outra forma de projeção.

Para deletarmos linhas, passamos os indexes:

Python

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

Deleção de linhas.

Nesse caso, deletamos as duas primeiras linhas do DataFrame.

Para regularizar a indexação depois de um drop, ou seja, reindexar, temos duas opções:

Python

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

Regularização da indexação - procedimento 1.

Python

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

Regularização da indexação - procedimento 2.

O index tem o papel de enumerar o conjunto e facilitar a manipulação dos dados.

Junção

As operações de junção são feitas pelos métodos **concat**, **merge** e **join**.

Veremos mais detalhes sobre cada um a seguir.

Método concat

Veja um exemplo que realiza a junção entre as séries pelos seus indexes:

Python

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

```

Método concat.

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, é possível fazer um reset_index também, mas podemos encurtar esse passo declarando dentro do pd.concat() o parâmetro ignore_index=True.

Método merge

Veja um exemplo:

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”
```

Método merge.

Declaramos as chaves estrangeiras, que serão a base da junção, e os sufixos, em caso de repetição de nome de colunas, como a coluna value, na qual teremos values da esquerda e da direita.

Método join

O join é feito a partir de um dos DataFrames, mas depende do index. Veja um exemplo:

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')
```

Método join.

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”).

Atividade

Questão 1

Um DataFrame é semelhante a uma matriz, mas suas colunas têm nomes e podem conter dados de diferentes tipos. Ele pode ser visto como uma tabela de uma base de

dados, em que cada linha corresponde a um registro. Nesse contexto, qual é a função do index de um DataFrame?

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

B — Ordenar apenas as linhas da tabela por meio de índices e servir como orientação visual.

C — É meramente estética, sem nenhuma outra atribuição.

D — Indexar o DataFrame, facilitar o acesso aos registros e otimizar consultas.

E — Indicar a performance do conjunto de dados analisado a partir de índice.

Parabéns! A alternativa D está correta.

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

Agregações em DataFrames

Groupby

Você sabe o que são e como fazer agregações em DataFrames?

As agregações são as operações mais utilizadas para a análise dos dados e para a extração de estatísticas básicas. Elas possibilitam a compreensão holística do conjunto de dados. Ao agregá-los, reduzimos o conjunto, 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())
```

Método groupby.

Aqui, podemos perceber como ocorre a agregação dos dados. O método groupby aceita uma lista de colunas, que serão os agrupadores, mas só chamá-lo não fará as agregações desejadas.

O que precisamos para fazer as 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. Basta passar o método agg(['mean']), que recebe uma lista de agregações cobertas na documentação do método no Pandas.

Atenção!

Nos bancos de dados tradicionais, 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.

Apply

A operação apply, que é um método do DataFrame e das séries, nada mais é do que o equivalente à função map de outras linguagens, como Java e Javascript. Nela, 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. Basta mudar o atributo axis na chamada. Veja um exemplo:

Python

```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)
```

Método apply.

Aqui, criamos um DataFrame sintético de três linhas: cada linha com 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.

Por fim, criamos uma nova coluna para ser a soma das colunas A e B.

Atividade

Questão 1

A função apply( ) é definida internamente na linguagem Python, que tem o objetivo de aplicar algo a algum objeto anteriormente estabelecido. Já as funções lambda permitem a escrita de um código Python mais organizado. Com base nesse entendimento, considere a coluna C no comando “df['C'].apply(lambda x: x*x)”. Você sabe que ela não tem valores nulos e é toda composta por números de ponto flutuante. Nesse caso, o que acontecerá com essa coluna?

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.

Parabéns! 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. Nesse caso, essa função é a multiplicação de x por ele mesmo, ou seja, a potência de 2.

Visualização de dados

Tipos de dados

No mundo de Big Data, há os mais variados dados possíveis, que, isoladamente, podem não significar muito. Mas, com o devido processamento, podemos extrair deles informação útil e conhecimento para tomada de decisões.

Para visualizar os dados, primeiro, precisamos conhecer seus tipos. Os dados podem ser classificados como:

Dados numéricos

São aqueles que expressam quantidades, proporções, valores monetários e métricas, tipicamente passíveis de operações e cálculos matemáticos, cujos valores são números:

- Reais – como os pontos de uma reta numérica infinita;
- Racionais – como 7/9 = 0,777...;
- Naturais – como N = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...}.

Dados categóricos

São aqueles normalmente expressos por texto, que representam rótulos, símbolos, nomes e identificadores. Podemos ter dados categóricos expressos por:

- Números – como notas de provas significando conceitos, que podem ser substituídos pelo sistema de letras A, B, C, D, E;
- Ordinais – que representam categorias de posição ou ordem, como primeiro (1º), segundo (2º) etc.

Podem ocorrer dados numéricos que, quando estratificados, passam a ser categóricos, como idade, temperatura etc.

Dados temporais

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

Tipos de visualizações

Para cada tipo de dado, temos visualizações mais apropriadas ou que indicam melhor a definição do conjunto de dados. Para dados numéricos, por exemplo, é comum demonstrar suas distribuições, correlações e proporções.

A seguir, veja como os dados podem ser visualizados a partir de gráficos.

Gráficos de histograma

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(). Veja:

Python

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

```

```
7
8
```

Criação de Gráfico Histograma.

Como resultado gráfico da execução do código, temos:

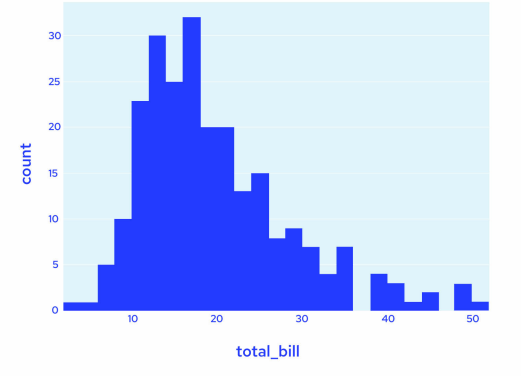


Gráfico de histograma da distribuição de dados de comandas de restaurante (data.tips).

Gráficos de dispersão

Também podemos querer saber a correlação entre **dados numéricos**. Para isso, utilizamos o **gráfico de dispersão** (scatterplot). Veja um exemplo:

Python

```python
import plotly.express as px
df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species"
fig.show()
5
6
7
8
```

Criação de Gráfico de dispersão.

Como resultado gráfico da execução do código, temos:

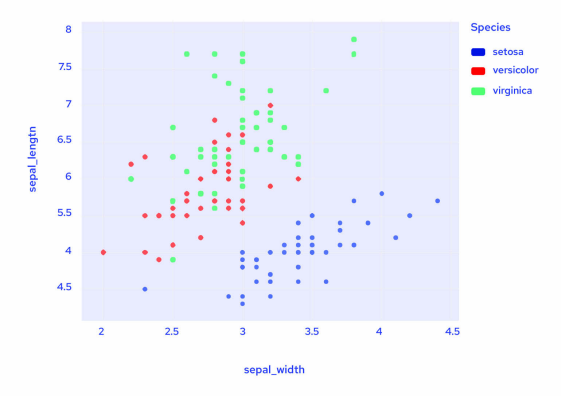


Gráfico de dispersão das larguras e comprimento de sépala de flores (data.iris).

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

Gráficos de barras

Para **dados categóricos**, queremos mostrar as categorias e suas proporções ou colorir outros gráficos, como fizemos no scatterplot. Podemos fazer isso por meio de **gráficos de barras**. Veja um 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
fig.show()

```

Criação de Gráfico de barra.

Como resultado gráfico da execução do código, temos:

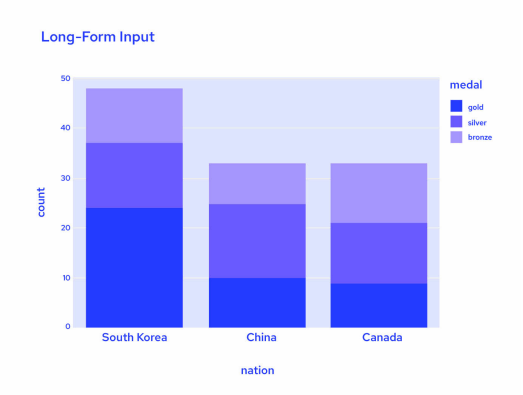


Gráfico de barras com quadro hipotético de medalhas da China, da Coreia do Sul e do Canadá (data.medals_long).

Como podemos ver, o gráfico de barras é muito parecido com o histograma. A diferença é que esse gráfico mostra a contagem e as proporções de dados categóricos. O eixo X será de dados categóricos, e o eixo Y será a contagem. É possível inverter os eixos, quando estamos olhando gráficos de barras orientados horizontalmente.

Gráficos de pizza

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

Python

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

```
4 fig.show()
5
6
```

Criação de Gráfico de pizza.

Como resultado gráfico da execução do código, temos:

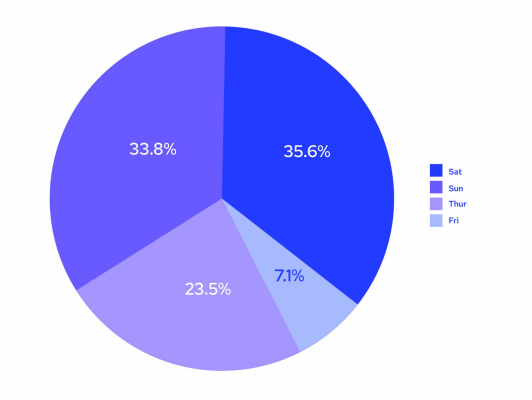


Gráfico de pizza com proporção dos valores distintos de dias da semana no conjunto de comandas (data.tips).

Nesse gráfico, podemos ver que o conjunto de dados só tem quatro dias distintos. Além disso, também visualizamos 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 há poucas categorias. Se ele tiver muitas fatias a exibir, ou seja, muitos valores únicos, passa a ficar ilegível. Nesse caso, é melhor dar preferência ao gráfico de barras.

Gráficos de linhas

Vamos falar, agora, dos dados temporais: aqueles que passam a ideia de cronologia. Uma solução natural é representarmos esses dados a partir dos gráficos de linhas. Veja um 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()


```

Criação de Gráfico de linhas.

Como resultado gráfico da execução do código, temos:

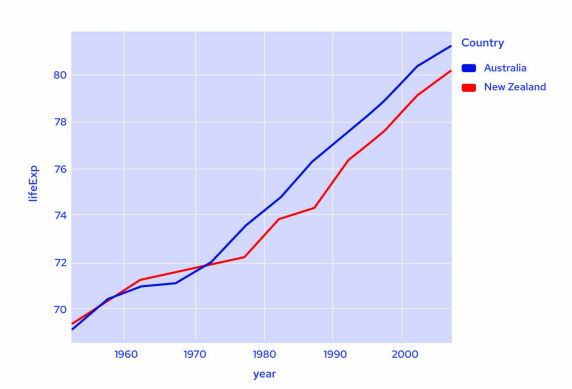


Gráfico de linhas com expectativa de vida ao longo dos anos para Austrália e Nova Zelândia (data.gapminder).

Aqui, é possível perceber bem a ideia de cronologia: o eixo X apresenta o dado temporal (anos, meses, datas completas, horas ou mesmo o index do DataFrame).

Assim, é 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. Precisamos tomar cuidado para os valores únicos dessas agregações não prejudicarem a visualização.

Atividade

Questão 1

Pandas é a biblioteca Python mais completa, que fornece ferramentas de análise e estruturas de dados de alta performance e fáceis de usar. Nesse contexto, 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.

Parabéns! A alternativa D está correta.

A classificação tradicional dos dados na hora da elaboração de visualizações inclui:

- Dados numéricos – que nos levam a histogramas e scatterplots;
- Dados categóricos – que nos levam aos gráficos de pizza ou de barras;
- Dados temporais – que nos levam aos gráficos de linha.

04. Geração de gráficos em Python

Gráfico de dispersão

Criando um gráfico de dispersão

1. Crie uma pasta vazia de projeto chamada scatter_plot_project.
2. Crie venv na mesma pasta de projeto, chamado venv mesmo.
3. Ative o venv.
4. Instale com pip as bibliotecas jupyter, pandas, NumPy, Sklearn e plotly.
5. Salve dependências no requirements.txt através de pip freeze.
6. Dentro da pasta vazia, no terminal, digite jupyter notebook.
7. Ao abrir o servidor Jupyter no browser, crie o arquivo jupyter notebook teste.
8. Na primeira célula, importe nossas bibliotecas (pandas as pd, NumPy as np, Sklearn, e plotly.express).
9. Importe o conjunto de dados Iris pela própria biblioteca do plotly express (e.g.: df = px.data.iris()).
10. Faça uma análise prévia do dataframe com describe.
11. Faça uma análise com info.
12. Monte a visualização do gráfico de dispersão scatterplot de duas das colunas do conjunto de dados.

Atividade

Questão 1

Agora que você já sabe visualizar dados, com base no exercício prático apresentado, crie em uma nova célula a visualização de matriz de dispersão. Todas as colunas do DF devem ser analisadas e coloridas pela espécie de íris.

Abrir solução

Você deve ter uma visualização similar àquela relativa ao gráfico de dispersão apresentado no exercício prático, mas para todas as colunas (par a par).

Gráfico de barras

Criando um gráfico de barras

1. Crie uma pasta vazia de projeto chamada histogram_plot_project.
2. Crie venv na mesma pasta de projeto, chamado venv mesmo.
3. Ative o venv.
4. Instale com pip as bibliotecas jupyter, pandas, NumPy, Sklearn e plotly.
5. Salve dependências no requirements.txt através de pip freeze.
6. Dentro da pasta vazia, no terminal, digite jupyter notebook.
7. Ao abrir o servidor Jupyter no browser, crie o arquivo jupyter notebook teste_hist.
8. Na primeira célula, importe nossas bibliotecas (Pandas as pd, NumPy as np, Sklearn, e plotly.express).
9. Importe o conjunto de dados Iris pela própria biblioteca do plotly express (e.g.: df =

px.data.iris()).

10. Faça uma análise prévia do dataframe com describe.
11. Faça uma análise com info.
12. Monte a visualização do gráfico de barras da coluna do comprimento de pétala colorida pela espécie de íris.

Atividade

Questão 1

Agora que você já sabe visualizar um histograma de colunas numéricas, imprima a visualização de gráfico de barras para contar a distribuição de espécies de cada flor. Sugerimos que comece com um groupby e imprima a partir dessa agregação.

Abrir solução

A resposta deve conter um dataframe agregado por groupby, utilizando a operação count, por espécie de planta. Na visualização, a coluna de espécie será passada como x, e a coluna de contagem escolhida, como y.

Gráfico de pizza

Criando um gráfico de pizza

1. Crie uma pasta vazia de projeto chamada pie_plot_project.
2. Crie venv na mesma pasta de projeto, chamado venv mesmo.
3. Ative o venv.
4. Instale com pip as bibliotecas jupyter, pandas, NumPy, Sklearn e plotly.
5. Salve dependências no requirements.txt através de pip freeze.
6. Dentro da pasta vazia, no terminal, digite jupyter notebook.
7. Ao abrir o servidor Jupyter no browser, crie o arquivo jupyter notebook teste_pie.
8. Na primeira célula, importe nossas bibliotecas (Pandas as pd, NumPy as np, Sklearn, e plotly.express).
9. Importe o conjunto de dados Iris pela própria biblioteca do plotly express (e.g.: df = px.data.iris()).
10. Faça uma análise prévia do dataframe com describe.
11. Faça uma análise com info.
12. Monte a visualização do gráfico de pizza pie de duas das distribuições de espécies do conjunto de dados.

Atividade

Questão 1

Agora que você já sabe fazer o gráfico de pizza pie chart, você pode criar um novo para contar as variações do comprimento de sépala no dataset. Para isso, categorize a coluna em questão, conte-a e imprima o resultado dessa transformação no gráfico de pizza.

Abrir solução

O gráfico de pizza conterá a distribuição de valores da coluna em questão com fatias diferentes para cada valor possível e a proporção de quantas vezes o valor aparece no conjunto.

Conclusão

O que você aprendeu neste conteúdo?

- O funcionamento da linguagem Python, algumas de suas peculiaridades, sua sintaxe e seus constructos;
- Os métodos de manipulação de dados disponíveis nos DataFrames, como projeção, deleção e junção;
- A coleta e o pré-processamento de dados para fins de visualização, ou seja, de concretização do processo de análise ou de recomeço, dependendo dos achados.

Explore +

- 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 usada no mercado.
- No **Pandas**, tente explorar outros métodos, pois a biblioteca é muito vasta e permite integrações diretas até mesmo com bancos de dados.
- Além do plotly, pesquise sobre as seguintes bibliotecas de visualização de dados: **Matplotlib** e **Seaborn**.

Referências bibliográficas

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.

KABIR, S. M. S.. **Methods of data collection.** In: KABIR, S. M. S. Basic guidelines for research: an introductory approach for all disciplines. 1. ed. Bangladesh: Book Zone Publication, 2016. p. 201-275.

SILVA, F. C. D.; GARCIA, A. C. B.. **Judice verum, a methodology for automatically classify fake, sarcastic and true portuguese news.** 2019. Dissertação (Mestrado em Informática) - Universidade Federal do Estado do Rio de Janeiro, 2019.

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: Springer-Verlag, 2000.