
Lendo e Manipulando Arquivos PDF

Asimov Academy

ASIMOV

Conteúdo

| | |
|--|-----------|
| 01. O que é um PDF? | 3 |
| 02. A estrutura de um arquivo PDF | 4 |
| 03. Abrindo o primeiro arquivo PDF | 5 |
| 04. Extraindo páginas de um PDF | 6 |
| 05. Combinando PDFs | 7 |
| Desafio | 7 |
| Resolução | 7 |
| 06. Rotação de páginas e conteúdo | 8 |
| Rotação de páginas | 8 |
| Rotação de conteúdo | 8 |
| 07. Mudando as dimensões de um PDF | 10 |
| Redimensionando as páginas | 10 |
| Redimensionando o conteúdo | 10 |
| 08. Cortando regiões de um PDF | 12 |
| 09. Adicionando carimbo e marca d'água | 14 |
| Adicionando carimbos | 14 |
| Adicionando marca d'água | 19 |
| 10. PDFs com senha | 21 |
| Protegendo PDFs com senha | 21 |
| Lidando com senhas em um PDF | 21 |
| Questões sobre segurança | 22 |
| 11. Anotações e anexos | 23 |
| O que são anotações? | 23 |
| Lendo anotações | 24 |
| Arquivos anexados | 24 |
| 12. Lendo o texto de um PDF | 25 |
| Exemplo de uso: contador de palavras | 25 |

| | |
|--|-----------|
| 13. Os desafios com extração de texto | 27 |
| PDFs complexos de extrair texto | 27 |
| 14. Extraíndo imagens de um PDF | 29 |
| Iterando sobre as imagens | 29 |
| Pegando os dados das imagens | 29 |
| 15. Convertendo PDFs em imagens | 32 |
| Salvando uma imagem em PDF | 32 |
| Salvando múltiplas imagens em um PDF | 32 |
| Manipulando o PDF com imagens | 33 |
| 16. Lendo tabelas de PDFs | 37 |
| Como vamos ler tabelas? | 37 |
| Instalando o pacote <code>tabula-py</code> | 37 |
| Lendo tabelas com <code>tabula-py</code> | 37 |
| Extraíndo a tabela | 38 |
| 17. Extraíndo todas as tabelas de um PDF | 41 |
| Sobre a extração de tabelas | 41 |
| Extraíndo todas as tabelas | 41 |
| Lidando com erros na extração | 42 |

01. O que é um PDF?

Arquivos PDF são muito utilizados para exibir uma variedade de informações: texto, imagens, gráficos e tabelas. Também são capazes de reproduzir mídia como vídeos e gifs, conter formulários, e até mesmo executar código - por isso, podem ser potencialmente perigosos para o computador!

O formato foi criado em 1991 pela Adobe. A partir de 2008, foi padronizado com uma especificação ISO. Com isso, o formato de arquivo PDF gradativamente deixou de ser vinculado à Adobe, para representar a forma “definitiva” de exibir documentos de forma padronizada.

Hoje em dia, há muitas opções de leitores de PDF, e também é possível gerar arquivos PDF a partir de outros programas como Word, Excel, ou navegadores web. PDF é um formato eficaz para garantir que a formatação do documento não será alterada, em função de ser aberto em um programa ou outro.

Essa flexibilidade tem certo preço: como são bastante flexíveis, nem sempre é possível trabalhar com PDFs de forma padronizada. O exemplo clássico é o arquivo PDF que contém texto, mas onde o texto está salvo como uma imagem. Neste caso, não é possível selecionar o texto diretamente, dificultando a criação de um processo padrão de trabalho!

Ao longo do curso, usaremos uma variedade de pacotes de Python para interagir e extrair valores de arquivos PDF. Cada pacote tem suas particularidades e vantagens: alguns são usados para manipular PDFs, modificando suas páginas, ou extraindo e combinando páginas específicas.

Outros são usados para extrair dados, como texto, imagens e tabelas. Mesmo assim, vale ressaltar dois pontos:

- 1) Não há garantia de que qualquer um dos pacotes irá funcionar em 100% dos casos. No curso, tentaremos ao máximo obter os valores de PDFs automaticamente, mas sempre há a possibilidade de você encontrar PDFs complexos e pouco padronizados, dificultando a extração de dados.
- 2) Há outros pacotes de Python disponíveis para trabalhar com PDFs. Se você não conseguir realizar sua automatização com os pacotes da aula, lembre-se de pesquisar e tentar usar outros pacotes que apliquem estratégias diferentes para ler os PDFs.

02. A estrutura de um arquivo PDF

Do ponto de vista da programação, arquivos PDF são um pouco diferentes de outros arquivos, como Excel e Word.

Pouca gente sabe, mas os arquivos do pacote Office são meramente um conjunto de arquivos XML dentro de uma pasta ZIP. Se quisermos, podemos extrair e acessar diretamente o XML destes arquivos. Os pacotes de Python que trabalham com arquivos do Office fazem exatamente isso!

Por outro lado, um PDF é um arquivo binário. Portanto, só é possível abri-lo com um programa específico (isto é, um leitor de PDF). Esse formato específico é mais complexo de ser lido por um programa de computador.

Em última análise, o formato binário de um arquivo PDF é responsável tanto pela flexibilidade de arquivos PDF (uma vez que virtualmente qualquer tipo de informação no computador pode ser representado por um formato binário), quanto pela dificuldade ou falta de padronização na sua leitura.

Mesmo assim, vamos aprender neste curso como trabalhar diretamente com PDFs e automatizar processos que muitos precisam fazer manualmente todos os dias. Vamos lá!

03. Abrindo o primeiro arquivo PDF

Para manipular arquivos PDF, vamos usar a biblioteca `pypdf`. Ela é uma biblioteca muito fácil de usar para fazer operações com arquivos PDF em si (ainda não vamos abordar como extrair dados).

Para instalá-la, use o comando abaixo no seu terminal ou CMD (já configurado com seu ambiente de Python):

```
pip install pypdf
```

Neste primeiro exemplo, vamos trabalhar com o arquivo PDF do livro “Dom Casmurro”, que está nos materiais de aula.

Dado um caminho para o arquivo PDF:

```
from pathlib import Path
```

```
caminho_pdf = Path('materiais de aula') / 'documentos' / 'dom_casmurro.pdf'
```

Podemos inicializar um **leitor de PDF** (classe `PdfReader`) da forma abaixo:

```
import pypdf
```

```
leitor_pdf = pypdf.PdfReader(caminho_pdf)
```

```
print(leitor_pdf)
```

Este objeto possui um atributo `pages` que representa uma lista de páginas do PDF. Cada página é um objeto próprio!

Através desse atributo, conseguimos pegar páginas específicas, e verificar o número de páginas do PDF:

```
print(leitor_pdf.pages) # Lista contendo as páginas
```

```
print(len(leitor_pdf.pages)) # Número de páginas
```

```
print(leitor_pdf.pages[0]) # Primeira página
```

Além disso, PDFs contêm um atributo `metadata`. Esse atributo contém os metadados do arquivo, como data de criação e nome do autor:

```
print(leitor_pdf.metadata) # Metadados
```

Por mais que pareça um dicionário, acessamos os valores com a sintaxe de atributo (usando pontos). Valores inexistentes retornam `None`:

```
print(leitor_pdf.metadata.creation_date) # Acessando metadado
```

```
print(leitor_pdf.metadata.author) # Acessando metadado inexistente
```

04. Extraíndo páginas de um PDF

Agora, vamos aprender a extrair páginas de um arquivo PDF. vamos usar como exemplo o PDF de um artigo científico escrito pela OpenAI, empresa que criou o ChatGPT.

Vamos extrair a página número 5 do artigo para um novo arquivo PDF (lembrando que a página 5 possui índice 4 no Python):

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'vpt_minecraft.pdf'

leitor_pdf = pypdf.PdfReader(caminho_pdf)
pagina = leitor_pdf.pages[4]
```

Para extrair páginas de um PDF, basta criarmos um novo **escritor de PDF** (classe PdfWriter) e adicionarmos a ele qualquer página que queremos copiar:

```
escritor_pdf = pypdf.PdfWriter()

escritor_pdf.add_page(pagina)

escritor_pdf.write('vpt_minecraft_p5.pdf')
```

Podemos também realizar esta ação com mais de uma página. Por exemplo, se quisermos copiar toda a seção de introdução, podemos fazer da seguinte forma:

```
escritor_pdf = pypdf.PdfWriter()

for pagina in leitor_pdf.pages[:3]:
    escritor_pdf.add_page(pagina)

escritor_pdf.write('vpt_minecraft_intro.pdf')
```

Alguns talvez já estejam vendo o poder destes métodos simples. Se combinarmos com loops, índices e slices, conseguimos manipular e extrair páginas de PDFs da forma que quisermos!

05. Combinando PDFs

Desafio

Combine as páginas de múltiplos PDFs.

Partindo dos 2 PDFs vistos até aqui, gere um novo arquivo PDF contendo as páginas 2, 4, e 9 dos arquivos originais.

As páginas devem ser extraídas de forma intercalada, isto é, devem seguir a ordem abaixo:

- Página 2 do PDF 1
- Página 2 do PDF 2
- Página 4 do PDF 1
- Página 4 do PDF 2
- Página 9 do PDF 1
- Página 9 do PDF 2

Resolução

```
from pathlib import Path

import pypdf

caminhos = [
    Path('materiais de aula') / 'documentos' / 'dom_casmurro.pdf',
    Path('materiais de aula') / 'documentos' / 'vpt_minecraft.pdf'
]

leitores = [pypdf.PdfReader(caminho) for caminho in caminhos]

indice_pgs = [1, 3, 8]

escritor_pdf = pypdf.PdfWriter()

for indice_pg in indice_pgs:
    for leitor in leitores:
        pagina = leitor.pages[indice_pg]
        escritor_pdf.add_page(pagina)

escritor_pdf.write('documento_intercalado.pdf')
```


06. Rotação de páginas e conteúdo

Rotação de páginas

Podemos girar uma página de um PDF simplesmente usando o método `rotate()` de uma página.

No código abaixo, para cada página do PDF inicial, geramos a página rotacionada em 90 graus (sentido horário) e a escrevemos no PDF de saída:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'vpt_minecraft.pdf'
leitor_pdf = pypdf.PdfReader(caminho_pdf)

escritor_pdf = pypdf.PdfWriter()

for page in leitor_pdf.pages:
    rotated_page = page.rotate(90)
    escritor_pdf.add_page(rotated_page)

escritor_pdf.write('vpt_minecraft_rotacionado_01.pdf')
```

Dica: podemos usar o argumento `clone_from` para passarmos páginas de um PDF para dentro do `PdfWriter`, poupando a repetição.

No exemplo abaixo, fazemos isso juntamente de uma rotação de -90 , ou seja, 90 graus no sentido horário:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'vpt_minecraft.pdf'
escritor_pdf = pypdf.PdfWriter(clone_from=caminho_pdf)

for page in escritor_pdf.pages:
    page.rotate(-90)

escritor_pdf.write('vpt_minecraft_rotacionado_02.pdf')
```

Rotação de conteúdo

Também podemos manter a página em sua configuração original, e rotacionar apenas o seu *conteúdo*. Para isso, precisamos usar o objeto `Transformation()` da seguinte forma:

```
from pathlib import Path
```

```
import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'vpt_minecraft.pdf'
escritor_pdf = pypdf.PdfWriter(clone_from=caminho_pdf)

transformation = pypdf.Transformation().rotate(45)

for page in escritor_pdf.pages:
    page.add_transformation(transformation)

escritor_pdf.write('vpt_minecraft_rotacionado_03.pdf')
```

Note que as páginas permanecem no sentido original, mas o conteúdo está girado em 45 graus! (No caso do `Transformation()`, 45 graus corresponde a um giro no sentido anti-horário). Apenas o conteúdo pode ser girado em valores diferentes de 90 graus.

Neste último exemplo, adicionamos também uma *translação* para ajustar a posição após a rotação. Note que é possível fazer operações em sequência no objeto `Transformation()`:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'vpt_minecraft.pdf'
escritor_pdf = pypdf.PdfWriter(clone_from=caminho_pdf)

# Transformação de 300 unidades no eixo X
transformation = pypdf.Transformation().rotate(30).translate(tx=300)

for page in escritor_pdf.pages:
    page.add_transformation(transformation)

escritor_pdf.write('vpt_minecraft_rotacionado_04.pdf')
```

As unidades do `pypdf` são mais ou menos arbitrárias, portanto a forma mais efetiva é testar valores diferentes até encontrar o que funcione para você!

07. Mudando as dimensões de um PDF

Redimensionando as páginas

No exemplo abaixo, redimensionamos o PDF inteiro para 10% do tamanho original:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'vpt_minecraft.pdf'
escritor_pdf = pypdf.PdfWriter(clone_from=caminho_pdf)

for page in escritor_pdf.pages:
    page.scale_by(0.1)

escritor_pdf.write('vpt_minecraft_redimensionado_01.pdf')
```

Parece não ter mudado nada, porém o tamanho das páginas se alterou! Dá para reparar isso se colocarmos a opção de 100% no programa de visualização de PDFs.

Possibilidade de uso: redimensionar páginas para outro tamanho de papel:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'vpt_minecraft.pdf'
escritor_pdf = pypdf.PdfWriter(clone_from=caminho_pdf)

for page in escritor_pdf.pages:
    page.scale_to(pypdf.PaperSize.A8.width, pypdf.PaperSize.A8.height)

escritor_pdf.write('vpt_minecraft_redimensionado_02.pdf')
```

Redimensionando o conteúdo

Da mesma forma como na rotação, podemos redimensionar apenas o *conteúdo* de cada página:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'vpt_minecraft.pdf'
escritor_pdf = pypdf.PdfWriter(clone_from=caminho_pdf)

transformation = pypdf.Transformation().scale(sx=1.5, sy=0.5)

for page in escritor_pdf.pages:
    page.add_transformation(transformation)
```

```
escritor_pdf.write('vpt_minecraft_redimensionado_03.pdf')
```

Pode parecer irreversível, mas sempre é possível fazer o caminho contrário:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('vpt_minecraft_redimensionado_03.pdf')
escritor_pdf = pypdf.PdfWriter(clone_from=caminho_pdf)

transformation = pypdf.Transformation().scale(sx=0.666, sy=2)

for page in escritor_pdf.pages:
    page.add_transformation(transformation)

escritor_pdf.write('vpt_minecraft_redimensionado_04.pdf')
```

O conteúdo continua dentro do arquivo PDF, mesmo quando não é possível vê-lo!

08. Cortando regiões de um PDF

As páginas de um PDF contém o atributo `mediabox`, que definem a posição do conteúdo visível de cada página:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'vpt_minecraft.pdf'
leitor_pdf = pypdf.PdfReader(caminho_pdf)

primeira_pagina = leitor_pdf.pages[0]
print('Mediabox original:', primeira_pagina.mediabox)
```

Podemos acessar e alterar os atributos do `mediabox` para configurar a região que será visível da página. Em outras palavras, “cortar” o conteúdo em páginas maiores ou menores:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'vpt_minecraft.pdf'
leitor_pdf = pypdf.PdfReader(caminho_pdf)

primeira_pagina = leitor_pdf.pages[0]
print('Mediabox original:', primeira_pagina.mediabox)

# Adicionando primeira página no output

escritor_pdf = pypdf.PdfWriter()
escritor_pdf.add_page(primeira_pagina)

# Modificando limites do mediabox diretamente
primeira_pagina.mediabox.left = -200
primeira_pagina.mediabox.right = 800
primeira_pagina.mediabox.top = 792
primeira_pagina.mediabox.bottom = -50
print('Mediabox modificado pelos limites:', primeira_pagina.mediabox)

escritor_pdf.add_page(primeira_pagina)

# Modificando limites do mediabox pelos "cantos"
primeira_pagina.mediabox.lower_left = (100, 220)
primeira_pagina.mediabox.upper_right = (500, 500)
print('Mediabox modificado pelos cantos:', primeira_pagina.mediabox)

escritor_pdf.add_page(primeira_pagina)
```

```
escritor_pdf.write('vpt_minecraft_cortado.pdf')
```

09. Adicionando carimbo e marca d'água

Podemos adicionar um carimbo ou marca d'água em páginas de um PDF. A forma de fazer isso é a mesma, a única diferença é conceitual: carimbos ficam por cima do conteúdo, enquanto marca d'água fica por baixo.

Adicionando carimbos

Vamos adicionar o carimbo abaixo a todas as páginas do PDF do livro Dom Casmurro:



Podemos adicionar um carimbo a uma página usando o método `merge_page`. Efetivamente, este método combina as duas páginas em uma. O argumento `over=True` garante que o carimbo será colado em cima do conteúdo da página:

```
from pathlib import Path
```

```
import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'dom_casmurro.pdf'
escritor_pdf = pypdf.PdfWriter(clone_from=caminho_pdf)

caminho_carimbo = Path('materiais de aula') / 'assets' / 'carimbo.pdf'
carimbo = pypdf.PdfReader(caminho_carimbo).pages[0]

for pagina in escritor_pdf.pages:
    pagina.merge_page(carimbo, over=True)

escritor_pdf.write('dom_casmurro_carimbo.pdf')
```

Note que, neste caso, o carimbo excedeu o tamanho da página. Isso aconteceu porque as dimensões do PDF do carimbo são maiores que a do livro:



Podemos consertar isso se usarmos uma transformação, como vimos na aula anterior. No código abaixo, o carimbo é redimensionado e movido para o canto superior esquerdo de cada página. Esta transformação é feita com o método `merge_transformed_page` de uma página:

```
from pathlib import Path
```

```
import pypdf
```

```
caminho_pdf = Path('materiais de aula') / 'documentos' / 'dom_casmurro.pdf'
```

Lendo e Manipulando Arquivos PDF

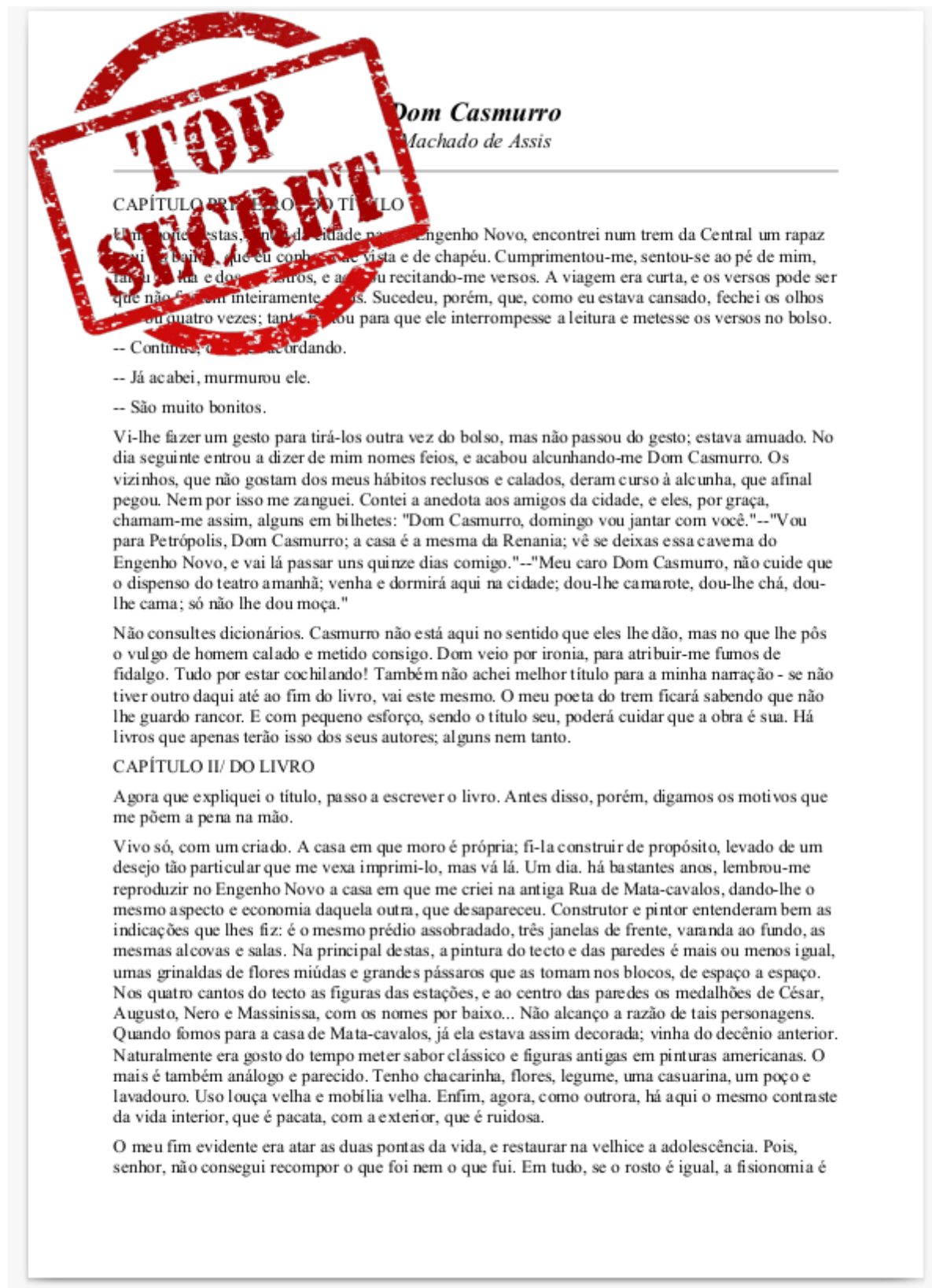
```
escritor_pdf = pypdf.PdfWriter(clone_from=caminho_pdf)

caminho_carimbo = Path('materiais de aula') / 'assets' / 'carimbo.pdf'
carimbo = pypdf.PdfReader(caminho_carimbo).pages[0]

for pagina in escritor_pdf.pages:
    fator_escalado = 4
    x = pagina.mediabox.left
    y = pagina.mediabox.top - carimbo.mediabox.height / fator_escalado
    transformation = pypdf.Transformation().scale(1 / fator_escalado).translate(x, y)
    pagina.merge_transformed_page(carimbo, transformation, over=True)

escritor_pdf.write('dom_casmurro_carimbo_ajustado.pdf')
```

E o resultado:



Adicionando marca d'água

Como mencionado, a única diferença entre um carimbo e uma marca d'água é que o carimbo fica sobre o texto original, e a marca fica por baixo. Este comportamento é controlado pelo argumento `over` do método `merge_page` / `merge_transformed_page`.

Vamos adicionar a marca d'água abaixo a todas as páginas do livro:



O código abaixo também usa uma transformação. Dessa vez, utiliza uma escala de transformação que resulta na marca d'água ficar totalmente ajustada à página:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'dom_casmurro.pdf'
escritor_pdf = pypdf.PdfWriter(clone_from=caminho_pdf)

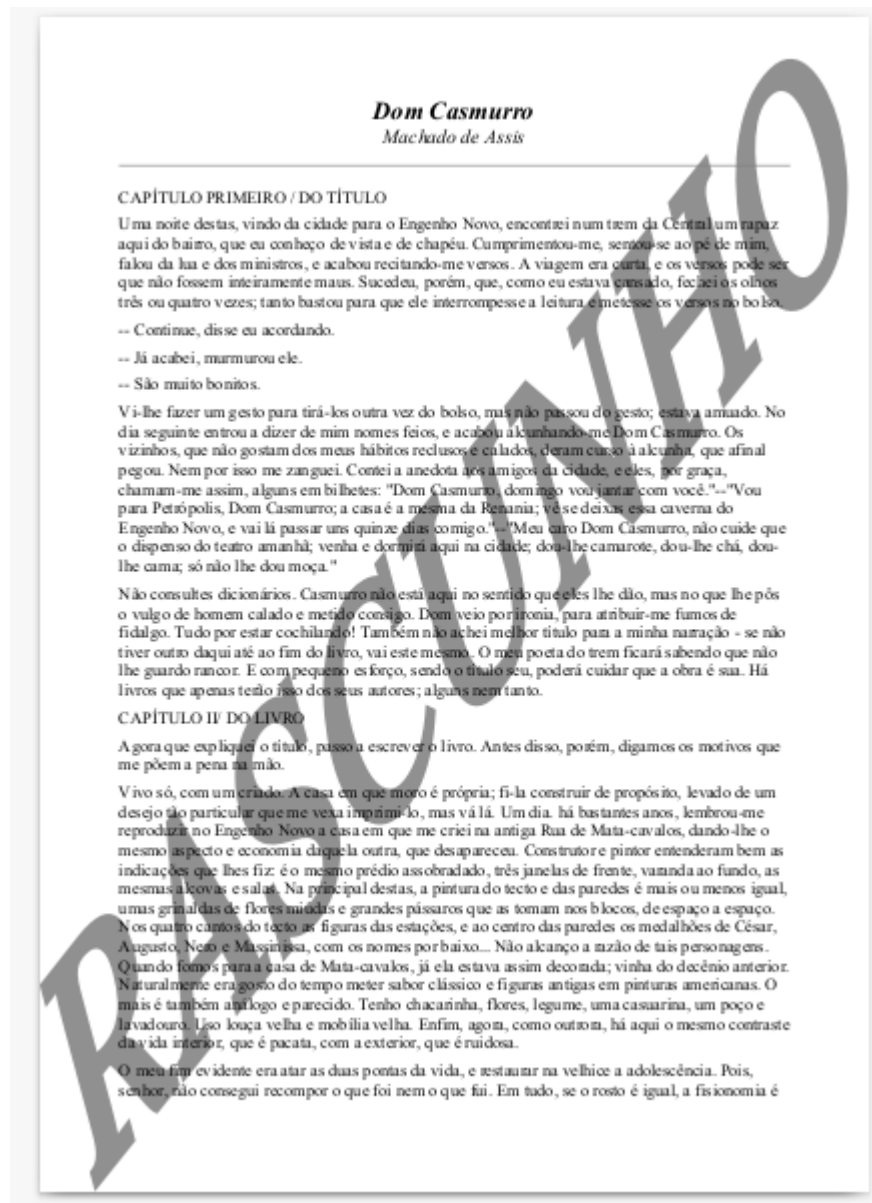
caminho_marca = Path('materiais de aula') / 'assets' / 'marca.pdf'
marca = pypdf.PdfReader(caminho_marca).pages[0]

for pagina in escritor_pdf.pages:
    escala_x = pagina.mediabox.width / marca.mediabox.width
    escala_y = pagina.mediabox.height / marca.mediabox.height
    transformation = pypdf.Transformation().scale(sx=escala_x, sy=escala_y)
```

```
pagina.merge_transformed_page(marca, transformation, over=False)
```

```
escritor_pdf.write('dom_casmurro_marca.pdf')
```

Resultado:



10. PDFs com senha

Protegendo PDFs com senha

É possível proteger um arquivo PDF com senha. Para fazer isso com Python, usamos o método `encrypt()` antes de salvar o PDF, e passamos a senha:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'dom_casmurro.pdf'
escritor_pdf = pypdf.PdfWriter(clone_from=caminho_pdf)

escritor_pdf.encrypt('senha-secreta')

escritor_pdf.write('dom_casmurro_com_senha.pdf')
```

Ao tentarmos abrir o arquivo `dom_casmurro_com_senha.pdf`, encontramos uma tela que pede pela senha do PDF!

Um PDF com senha não pode ser lido antes que a senha seja entregue:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('dom_casmurro_com_senha.pdf')
leitor_pdf = pypdf.PdfReader(caminho_pdf)

# A linha abaixo gera erro: FileNotDecryptedError: File has not been decrypted
print(leitor_pdf.pages)
```

Lidando com senhas em um PDF

Podemos usar o método `decrypt()` para desbloquear o PDF.

Note que o método `decrypt()` retorna `0` se a senha estiver incorreta. Além disso, um `PdfReader` possui o atributo `is_encrypted`, que retorna se ele requer senha ou não.

Com esses métodos, é possível criar um controle de fluxo para lidar com a senha (até mesmo considerando a situação onde a senha passada não está certa):

```
from pathlib import Path

import pypdf

caminho_pdf = Path('dom_casmurro_com_senha.pdf')
leitor_pdf = pypdf.PdfReader(caminho_pdf)
```

```
if leitor_pdf.is_encrypted:
    senha = input('PDF requer senha. Digite a senha do PDF: ')
    result = leitor_pdf.decrypt(senha)
    if result == 0:
        print('Senha incorreta.')
    else:
        print('Senha correta.')
        print(f'O PDF tem {len(leitor_pdf.pages)} páginas.')
else:
    print(f'O PDF tem {len(leitor_pdf.pages)} páginas.')
```

Questões sobre segurança

O algoritmo usado por padrão para proteger o PDF com senha se chama RC4. Contudo, segundo [informações na documentação](#) do pypdf, esse algoritmo não é 100% seguro.

Por isso, se você tiver documentos realmente sensíveis, o ideal é utilizar um outro pacote de Python chamado cryptography para proteger um PDF com senha.

O link da documentação do pypdf mostra instruções de instalação e uso dos algoritmos do cryptography.

11. Anotações e anexos

O que são anotações?

PDFs podem conter uma gama de conteúdo variado, que são conjuntamente chamados de “anotações”.

Anotações podem ser:

- Textos
- Links
- Formas geométricas (linhas, quadrados, círculos, polígonos)
- Mídia (vídeos, gifs, música)
- Arquivos em anexo
- Muitos outros!

Podemos criá-las e acessá-las via Python. O código abaixo cria algumas anotações no PDF:

```
from pathlib import Path

import pypdf
from pypdf.annotations import FreeText, Text, Line, Rectangle

caminho_pdf = Path('materiais de aula') / 'documentos' / 'dom_casmurro.pdf'
escritor_pdf = pypdf.PdfWriter(clone_from=caminho_pdf)

# Texto livre
texto_livre = FreeText(text="Olá Mundo!\nEsta é uma anotação flutuante!", rect=(400, 550, 550,
↪ 600))
escritor_pdf.add_annotation(0, texto_livre)

# Texto em uma nota
texto = Text(text="Texto da anotação!", rect=(50, 600, 200, 650))
escritor_pdf.add_annotation(1, texto)

# Linha
linha = Rectangle(rect=(50, 550, 200, 650))
escritor_pdf.add_annotation(2, linha)

# Arquivo anexo qualquer
# (deve ser lido como bytes, modo de leitura "rb")
caminho_imagem = Path('materiais de aula') / 'assets' / 'dog.jpg'
with open(caminho_imagem, 'rb') as arquivo:
    dados = arquivo.read()
escritor_pdf.add_attachment('cachorro.jpg', dados)

# Gera um PDF com todas as anotações
escritor_pdf.write('dom_casmurro_annotado.pdf')
```


Nota: para acessar o arquivo em anexo, pode ser necessário clicar em algum botão no seu leitor de PDF!

Lendo anotações

Lembrando: dentro do pypdf, cada página funciona de forma semelhante a um dicionário. A presença da chave `"/Annots"` é o que indica que há uma anotação naquela página.

Sendo assim, podemos iterar sobre todas as páginas do PDF e encontrar as anotações existentes:

```
leitor_pdf = pypdf.PdfReader("dom_casmurro_annotado.pdf")

for pagina in leitor_pdf.pages:
    if "/Annots" not in pagina:
        continue
    for annot in pagina["/Annots"]:
        print(annot.get_object())
```

Cada anotação também funciona como um dicionário, e possui uma chave `"\Subtype"` que descreve o tipo de anotação que é. A partir dessa chave, podemos extrair o conteúdo (no caso de textos) ou a posição (no caso de figuras geométricas), por exemplo.

Arquivos anexados

Arquivos em anexo não ficam em uma página específica, mas sim dentro do leitor em si. Podemos checá-los a partir do atributo `attachments`.

O código abaixo pega os bytes do arquivo anexo `dog.jpg` e os escreve de volta em um arquivo no computador:

```
leitor_pdf = pypdf.PdfReader("dom_casmurro_annotado.pdf")

print(leitor_pdf.attachments)

bytes = leitor_pdf.attachments['cachorro.jpg'][0]
with open('nova_imagem.jpg', 'wb') as imagem:
    imagem.write(bytes)
```

12. Lendo o texto de um PDF

Podemos usar o método `extract_text()` de uma página para obter todo o seu texto como um string.

O código abaixo roda o método `extract_text()` em cada uma das páginas, com o número da página exibido no topo do texto (através do `enumerate`):

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'vpt_minecraft.pdf'
leitor_pdf = pypdf.PdfReader(caminho_pdf)

for i, pagina in enumerate(leitor_pdf.pages, 1):
    print(f' ----- Página {i:2d}')
    print(pagina.extract_text())
    input()
```

Neste exemplo, usamos `input()` apenas para pausar o código temporariamente e lermos o conteúdo de uma página antes de seguir para a próxima (apertando a tecla Enter).

Exemplo de uso: contador de palavras

O exemplo abaixo usa o objeto `Counter`, do módulo da biblioteca padrão `collections`, para contar a ocorrência de cada palavra no texto:

```
from collections import Counter
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'vpt_minecraft.pdf'
leitor_pdf = pypdf.PdfReader(caminho_pdf)

texto_por_pagina = [pagina.extract_text() for pagina in leitor_pdf.pages]

palavras = []
for texto in texto_por_pagina:
    palavras_pagina = texto.lower().replace('\n', ' ').replace(',', ' ').replace('.', ' ')
    ↪ palavras_pagina = palavras_pagina.split(' ')
    palavras.extend(palavras_pagina)

contagem = Counter(palavras)

print(contagem)

print(contagem.most_common(20))
```

Ou seja, para cada página obtida na lista `texto_por_pagina`, são feitas as seguintes ações:

- Transformar todos os caracteres em minúsculo.
- Remover todas as quebras de linhas e caracteres de vírgula e ponto.
- Dar um `.split()` no string resultante, gerando uma lista de palavras.
- Adicionar todas as palavras dessa lista à lista `palavras` com o método `.extend()`.

O objeto `Counter` funciona como um dicionário. Podemos exibir as palavras de 5 ou mais caracteres que aparecem 5 ou mais vezes no texto da seguinte forma:

```
for palavra, cont in sorted(contagem.items(), key=lambda tupla: tupla[1]):  
    if len(palavra) >= 5 and cont >= 5:  
        print(f'{palavra} -> {cont}')
```

13. Os desafios com extração de texto

A extração de texto funcionou razoavelmente bem para o arquivo da aula anterior, mas será que é sempre assim tão fácil?

Vamos testar com o PDF do Dom Casmurro:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'dom_casmurro.pdf'
leitor_pdf = pypdf.PdfReader(caminho_pdf)

for i, pagina in enumerate(leitor_pdf.pages, 1):
    print(f' ----- Página {i:2d}')
    print(pagina.extract_text())
    input()
```

Infelizmente, nem sempre a extração de texto é perfeita. Neste caso, há muitas palavras que foram quebradas de forma errada. E o pior é que não fica claro quando uma extração de texto funcionará ou não — só testando para saber!

O método `extract_text()` até possui alguns parâmetros de configuração, mas mesmo quando os modificamos, não é garantia de que a extração seja possível. A própria [documentação do pypdf](#) possui uma seção apenas discutindo as dificuldades de extrair texto de PDFs.

PDFs complexos de extrair texto

Em geral, quanto mais complexa for a formatação do PDF, menos provável será que a extração de texto funcione sem erros.

Vamos tentar ler o PDF com dados para investidores da Ambev:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'RI Ambev - 1T23.pdf'
leitor_pdf = pypdf.PdfReader(caminho_pdf)

for i, pagina in enumerate(leitor_pdf.pages, 1):
    print(f' ----- Página {i:2d}')
    print(pagina.extract_text())
    input()
```

Mesmo quando o texto é lido corretamente, em alguns casos os blocos de texto podem vir fora de ordem. Textos como rodapés, linhas de tabelas, ou texto flutuante também podem vir em ordem

arbitrária.

14. Extraíndo imagens de um PDF

Podemos usar o `pypdf` mais uma vez para extrair imagens. Contudo, ele utiliza internamente a biblioteca `Pillow` (a mesma que é usada no curso de edição de imagens).

Portanto, temos que instalá-la (atenção com o `P` maiúsculo):

```
pip install Pillow
```

Iterando sobre as imagens

As imagens ficam armazenadas em cada página, em uma lista no atributo `.images`. Assim, podemos facilmente acessar esta lista e iterar sobre as imagens de cada página:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'RI Ambev - 1T23.pdf'
leitor_pdf = pypdf.PdfReader(caminho_pdf)

for page in leitor_pdf.pages:
    for obj_imagem in page.images:
        print(f'{obj_imagem} -> {obj_imagem.name}')
```

Como é possível ver, as imagens ficam armazenadas dentro de um objeto específico chamado `ImageFile`.

Pegando os dados das imagens

Além do atributo `.name`, cada `ImageFile` possui também o atributo `.data`. Este atributo contém a informação da imagem, armazenada no formato de bytes:

```
from pathlib import Path

import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'RI Ambev - 1T23.pdf'
leitor_pdf = pypdf.PdfReader(caminho_pdf)

for page in leitor_pdf.pages:
    for obj_imagem in page.images:
        print(f'{obj_imagem}\n{obj_imagem.data}')
        input()
```

O código acima exibe os dados no console, mas isso tem pouca utilidade para nós.

Se quisermos transformar os bytes da imagem em um arquivo de imagem de fato, vamos precisar escrevê-los para dentro de um arquivo, usando o modo de escrita "wb" (de escrita binária, ou *write binary* do inglês).

O código abaixo faz essa escrita para todas as imagens encontradas no PDF, além de salvá-las todas em ordem sequencial dentro de uma pasta específica:

```
from pathlib import Path

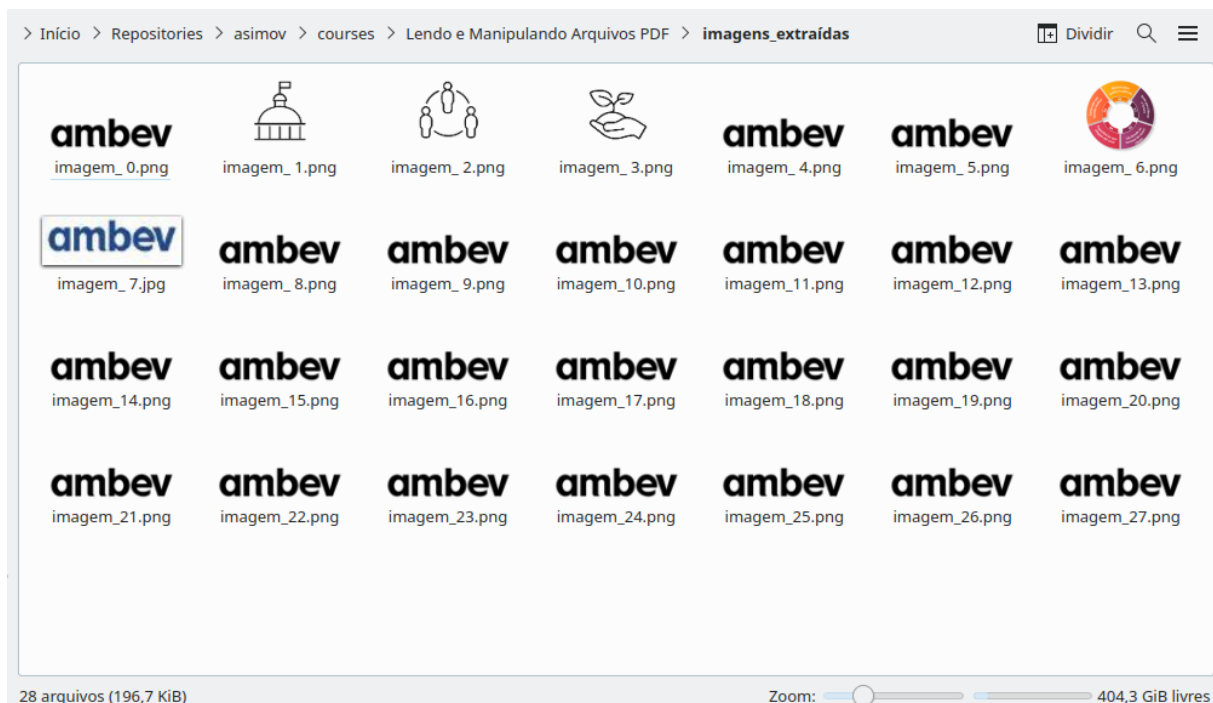
import pypdf

caminho_pdf = Path('materiais de aula') / 'documentos' / 'RI Ambev - 1T23.pdf'
leitor_pdf = pypdf.PdfReader(caminho_pdf)

pasta_output = Path('imagens_extraídas')
pasta_output.mkdir(exist_ok=True)

i = 0
for page in leitor_pdf.pages:
    for obj_imagem in page.images:
        extensao = obj_imagem.name.split('.')[-1]
        caminho_output = pasta_output / f'imagem_{i:2d}.{extensao}'
        with open(caminho_output, 'wb') as arquivo_imagem:
            arquivo_imagem.write(obj_imagem.data)
        i += 1
```

O resultado é o seguinte:



Veja que muitas imagens repetidas foram extraídas, provavelmente do cabeçalho de cada página.

Dentro do PDF, essa imagem aparece com uma única referência ("Image11.png"). Mas ao iterarmos sobre o PDF, o código detecta cada vez em que a imagem aparece repetida.

15. Convertendo PDFs em imagens

Podemos usar o Pillow também para fazer o caminho contrário: transformar uma imagem (ou diversas imagens) em um arquivo PDF.

Salvando uma imagem em PDF

Este processo é bastante simples: basta usarmos o objeto Image do Pillow. Lemos a imagem com `Image.open()`, passando o caminho da imagem, e salvamos em PDF com `Image.save()`:

```
from pathlib import Path

from PIL import Image

caminho = Path('materiais de aula') / 'assets' / 'cachorro.jpg'
imagem = Image.open(caminho)
imagem.save('cachorro.pdf')
```

O Pillow é “inteligente” e sabe que queremos salvar um PDF apenas com base na extensão do nome usado!

Salvando múltiplas imagens em um PDF

Podemos salvar múltiplas imagens em um mesmo arquivo PDF, uma imagem por página. Para isso, basta modificar alguns argumentos em `Image.save()`.

Usamos aqui o método `iterdir()` e o atributo `.suffix` de objetos Path (módulo `pathlib` da biblioteca padrão de Python) para iterar sobre todas as imagens de uma pasta, e para encontrar arquivos com extensão correspondente a alguma imagem:

```
from pathlib import Path

from PIL import Image

pasta_imagens = Path('materiais de aula') / 'assets'
extensoes = ['.png', '.jpg', '.jpeg']

imagens = []
for caminho in sorted(pasta_imagens.iterdir()):
    if caminho.suffix in extensoes:
        imagem = Image.open(caminho)
        imagens.append(imagem)

primeira_imagem = imagens[0]
demais_imagens = imagens[1:]

primeira_imagem.save('arquivo_saida.pdf', save_all=True, append_images=demais_imagens)
```

No método `Image.save()`, usamos os argumentos `save_all` e `append_images` para salvar todas as demais imagens da lista no mesmo PDF.

Manipulando o PDF com imagens

Cada imagem ficou com um tamanho diferente. Isso acontece porque o Pillow gera cada página do PDF resultante usando a resolução de cada imagem.

Se quisermos consertar isso, podemos voltar para a aula de carimbos e combinar cada página como se fosse um “carimbo” sobre uma página em branco.

Aqui, vamos partir sempre do arquivo de saída do passo anterior. Usamos o método `PdfWriter.add_blank_page` para criarmos páginas em branco, com tamanho de uma página A4:

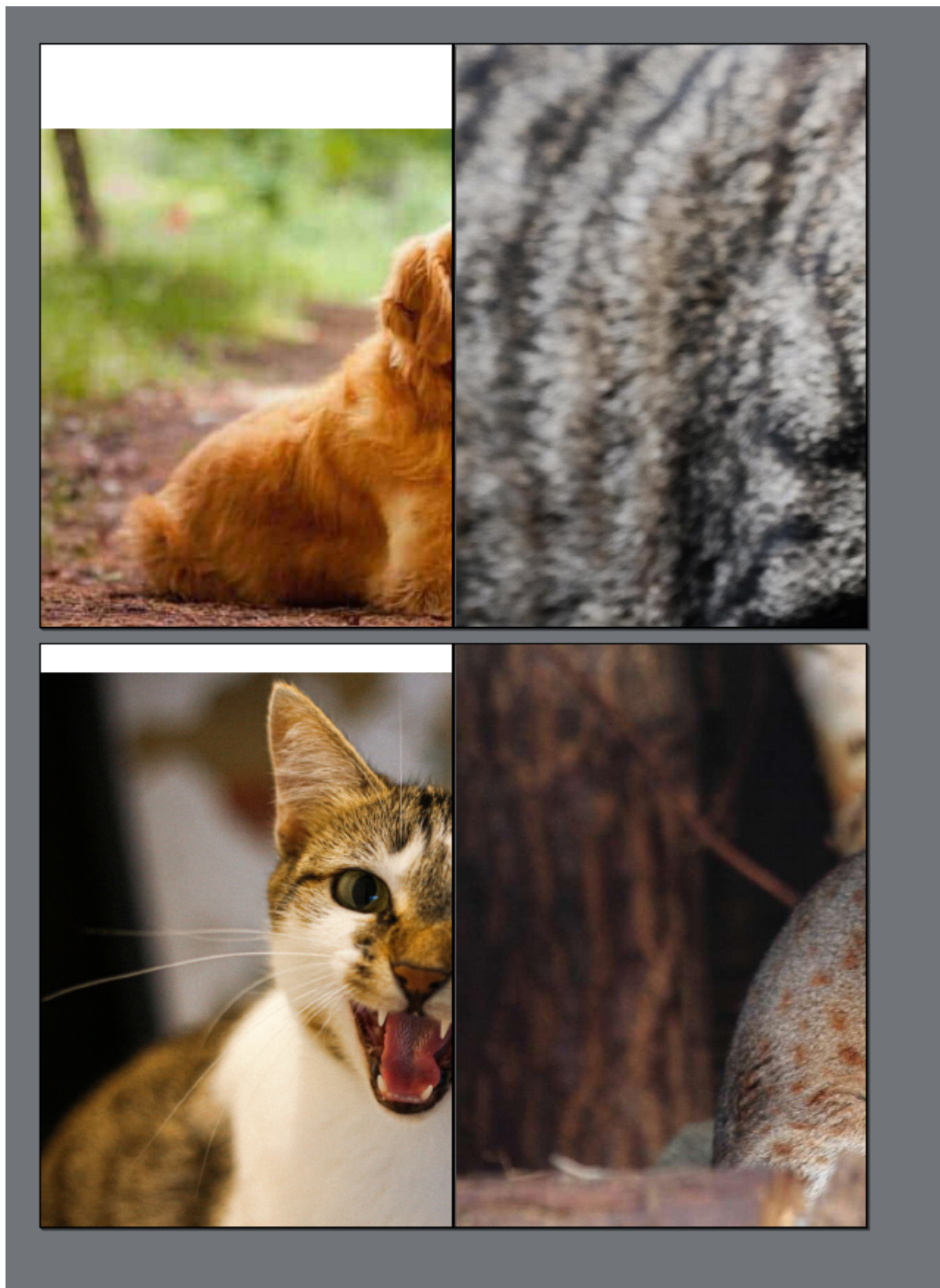
```
import pypdf

pdf_imagens = pypdf.PdfReader('arquivo_saida.pdf')
escritor_pdf = pypdf.PdfWriter()

for pagina in pdf_imagens.pages:
    pagina_em_branco = escritor_pdf.add_blank_page(
        width=pypdf.PaperSize.A4.width,
        height=pypdf.PaperSize.A4.height,
    )
    pagina_em_branco.merge_page(pagina, over=True)

escritor_pdf.write('arquivo_saida_A4.pdf')
```

O arquivo de saída ficou com páginas do mesmo formato, porém as imagens ficaram maiores que as páginas. Podemos corrigir isso com uma operação de scaling:



Lendo e Manipulando Arquivos PDF

```
import pypdf

pdf_imagens = pypdf.PdfReader('arquivo_saida.pdf')
escritor_pdf = pypdf.PdfWriter()

for pagina in pdf_imagens.pages:
    pagina_em_branco = escritor_pdf.add_blank_page(
        width=pypdf.PaperSize.A4.width,
        height=pypdf.PaperSize.A4.height,
    )
    if pagina.mediabox.top > pagina.mediabox.right: # Imagem vertical
        scale = pagina_em_branco.mediabox.top / pagina.mediabox.top * 0.9
    else: # Imagem horizontal (ou quadrada)
        scale = pagina_em_branco.mediabox.right / pagina.mediabox.right * 0.9
    transformation = pypdf.Transformation().scale(scale)
    pagina_em_branco.merge_transformed_page(pagina, transformation, over=True)

escritor_pdf.write('arquivo_saida_margem.pdf')
```

E por fim, um translate para centralizar:

```
import pypdf

pdf_imagens = pypdf.PdfReader('arquivo_saida.pdf')
escritor_pdf = pypdf.PdfWriter()

for pagina in pdf_imagens.pages:
    pagina_em_branco = escritor_pdf.add_blank_page(
        width=pypdf.PaperSize.A4.width,
        height=pypdf.PaperSize.A4.height,
    )
    if pagina.mediabox.top > pagina.mediabox.right: # Imagem vertical
        scale = pagina_em_branco.mediabox.top / pagina.mediabox.top * 0.9
    else: # Imagem horizontal (ou quadrada)
        scale = pagina_em_branco.mediabox.right / pagina.mediabox.right * 0.9
    tx = (pagina_em_branco.mediabox.right - pagina.mediabox.right * scale) / 2
    ty = (pagina_em_branco.mediabox.top - pagina.mediabox.top * scale) / 2
    transformation = pypdf.Transformation().scale(scale).translate(tx=tx, ty=ty)
    pagina_em_branco.merge_transformed_page(pagina, transformation, over=True)

escritor_pdf.write('arquivo_saida_centralizado.pdf')
```

Sucesso!



16. Lendo tabelas de PDFs

Como vamos ler tabelas?

Como vimos na aula anterior, se simplesmente usarmos o método `.extract_text()`, as tabelas muitas vezes serão lidas como texto desordenado. Até é possível tentar ler as linhas individuais e ir “adivinhandando” onde cada coluna da tabela está, mas é um trabalho bastante minucioso e sujeito a erros!

No lugar disso, vamos utilizar um pacote específico para ler tabelas em PDF chamado `tabula-py`. Esse pacote vai justamente lidar com as dificuldades de ler tabelas em arquivos PDF.

Instalando o pacote `tabula-py`

Podemos instalar o pacote com o comando abaixo no terminal:

```
pip install tabula-py
```

Importante: o pacote `tabula-py` utiliza Java por debaixo dos panos, portanto **você deve Java instalado no seu computador para utilizá-lo**. Caso contrário, seu código dará um erro na execução, acusando que não encontrou o Java no seu computador.

Em muitos computadores, Java já está instalado. Mas caso você não tenha, [você pode baixá-lo e instalá-lo gratuitamente a partir do site oficial](#).

Adicionalmente, vamos também instalar o pacote `openpyxl`:

```
pip install openpyxl
```

O pacote `openpyxl` é usado para interagir com planilhas de Excel via Python. Ele não é estritamente necessário para ler as tabelas do PDF, mas vamos utilizá-lo para salvar as tabelas em arquivos Excel.

Lendo tabelas com `tabula-py`

A função `tabula.read_pdf()` aceita um caminho para um arquivo PDF e retorna uma lista de todas as tabelas que encontrou. As tabelas são retornadas como DataFrames do pandas.

É necessário passar uma lista com o número das páginas (parâmetro `pages`) para selecionar de que páginas as tabelas serão extraídas. É possível passar o valor `"all"` para extrair todas as tabelas encontradas, mas dependendo do tamanho do PDF e do número de tabelas, a extração pode demorar!

Extraindo a tabela

Vamos tentar extrair a tabela da página 3 do relatório da Ambev:

Página | 3

ambev

Destaques financeiros - consolidado

| R\$ milhões | 1T22 | 1T23 | % Reportado | % Orgânico |
|-------------------------------|----------------|----------------|--------------|--------------|
| Volume ('000 hl) | 45.082,3 | 44.921,2 | -0,4% | -0,4% |
| Receita líquida | 18.439,2 | 20.531,7 | 11,3% | 26,5% |
| Lucro bruto | 9.024,7 | 10.400,1 | 15,2% | 33,8% |
| % Margem bruta | 48,9% | 50,7% | 180pb | 290pb |
| EBITDA ajustado | 5.522,9 | 6.444,4 | 16,7% | 39,9% |
| % Margem EBITDA ajustado | 30,0% | 31,4% | 140pb | 310pb |
| Lucro líquido | 3.528,8 | 3.819,2 | 8,2% | |
| Lucro líquido ajustado | 3.551,6 | 3.839,8 | 8,1% | |
| LPA (R\$/ação) | 0,22 | 0,23 | 8,4% | |
| LPA ajustado | 0,22 | 0,24 | 8,3% | |

Apenas para comparação: se tentarmos usar Ctrl+C, Ctrl+V para copiar e colar a tabela em arquivo Excel, não vai funcionar...

Ambev como plataforma

À medida que nossos negócios continuaram a evoluir, mantivemos o foco na execução e na entrega de resultados em cada um dos cinco pilares do nosso *framework*:

1. No Brasil, nossas marcas adicionaram mais de 700 mil fãs em relação ao 1T22, de acordo com nossas estimativas, e a saúde das nossas marcas foca *super premium* e *premium* aumentou em relação ao ano passado. Brahma e 24 Delivery foram as duas marcas mais citadas nas redes sociais pelos brasileiros durante as festividades de Carnaval.

2. No Brasil, o volume de Spaten cresceu dois dígitos volume da Budweiser Zero cresceu mais de sequencialmente, e sua cobertura mais do que dobrou, relação ao último trimestre.

5. Durante as festividades de Carnaval, focamos na logística reversa, fortalecendo nosso ecossistema de catadores com a distribuição de equipamentos de proteção individual (EPIs) adequados, apoio salarial e outros incentivos, resultando em mais de 175 toneladas de resíduos corretamente destinados

O código para a extração via Python segue abaixo:

```
from pathlib import Path
```

```
import tabula
```

```
caminho_pdf = Path('materiais de aula') / 'documentos' / 'RI Ambev - 1T23.pdf'
tabelas = tabula.read_pdf(caminho_pdf, pages=3)
```



```
print(f'Foram encontradas {len(tabelas)} tabelas no arquivo PDF.')
for i, df in enumerate(tabelas, 1):
    print(f'\n\n----- Tabela {i} ----- \n')
    print(df)
    df.to_excel(f'Tabela {i}.xlsx')
```

Neste for loop, estamos ao mesmo tempo exibindo cada uma das tabelas, e salvando-as em arquivos Excel (através do pacote `openpyxl`). Ao executar esse código, você deverá ver o arquivo `Tabela 1.xlsx` aparecer no seu sistema operacional.

Abaixo, o resultado do código no console:

```
----- Tabela 1 -----

      R$ milhões      1T22      1T23 % Reportado % Orgânico
0      Volume ('000 hl) 45.082,3 44.921,2      -0,4%      -0,4%
1      Receita líquida 18.439,2 20.531,7      11,3%      26,5%
2      Lucro bruto     9.024,7 10.400,1      15,2%      33,8%
3      % Margem bruta   48,9%   50,7%      180pb     290pb
4      EBITDA ajustado  5.522,9 6.444,4      16,7%      39,9%
5      % Margem EBITDA ajustado 30,0%   31,4%      140pb     310pb
6      NaN             NaN      NaN      NaN      NaN
7      Lucro líquido   3.528,8 3.819,2      8,2%      NaN
8      Lucro líquido ajustado 3.551,6 3.839,8      8,1%      NaN
9      LPA (R$/ação)    0,22    0,23      8,4%      NaN
10     LPA ajustado     0,22    0,24      8,3%      NaN

----- Tabela 2 -----

1. No Brasil, nossas marcas adicionaram mais de 700 mil fãs
0 em relação ao 1T22, de acordo com nossas estim...
1 saúde das nossas marcas foco super premium e p...
2 aumentou em relação ao ano passado. Brahma e ...
```

E no arquivo Excel gerado:

| A | B | C | D | E | F |
|-----------|--------------------------|-------------|-------------|--------------------|-------------------|
| | R\$ milhões | 1T22 | 1T23 | % Reportado | % Orgânico |
| 0 | Volume ('000 hl) | 45.082,3 | 44.921,2 | -0,4% | -0,4% |
| 1 | Receita líquida | 18.439,2 | 20.531,7 | 11,3% | 26,5% |
| 2 | Lucro bruto | 9.024,7 | 10.400,1 | 15,2% | 33,8% |
| 3 | % Margem bruta | 48,9% | 50,7% | 180pb | 290pb |
| 4 | EBITDA ajustado | 5.522,9 | 6.444,4 | 16,7% | 39,9% |
| 5 | % Margem EBITDA ajustado | 30,0% | 31,4% | 140pb | 310pb |
| 6 | | | | | |
| 7 | Lucro líquido | 3.528,8 | 3.819,2 | 8,2% | |
| 8 | Lucro líquido ajustado | 3.551,6 | 3.839,8 | 8,1% | |
| 9 | LPA (R\$/ação) | 0,22 | 0,23 | 8,4% | |
| 10 | LPA ajustado | 0,22 | 0,24 | 8,3% | |

Sucesso! Contudo, você deve ter notado que apareceu também uma segunda tabela. Vamos falar mais dela na próxima aula!

17. Extraíndo todas as tabelas de um PDF

Sobre a extração de tabelas

Você deve ter notado que a extração da aula anterior acabou gerando uma segunda tabela.

Se pararmos para analisá-la, vamos chegar à conclusão de que é apenas um erro do `tabula-py`. Afinal, é uma tabela com uma única coluna, cujas entradas são o texto de um parágrafo!

The screenshot shows a PDF document with a table and a paragraph of text. The table has the following data:

| | 1T22 | 1T21 | Variação |
|------------------------|---------|---------|----------|
| Lucro líquido | 3.528,8 | 3.819,2 | 8,2% |
| Lucro líquido ajustado | 3.551,6 | 3.839,8 | 8,1% |
| LPA (R\$/ação) | 0,22 | 0,23 | 8,4% |
| LPA ajustado | 0,22 | 0,24 | 8,3% |

The paragraph of text is:

1. No Brasil, nossas marcas adicionaram mais de 700 mil fãs em relação ao 1T22, de acordo com nossas estimativas, e a saúde das nossas marcas foco super premium e premium aumentou em relação ao ano passado. Brahma e Zé Delivery foram as duas marcas mais citadas nas redes sociais pelos brasileiros durante as festividades de Carnaval.

Essas tabelas lidas “a mais” são facilmente identificáveis: possuem uma única coluna, cujo conteúdo são linhas do texto do arquivo.

Extraíndo todas as tabelas

Vamos agora para a extração de todas as tabelas! Seguiremos do código da aula anterior, mas com as seguintes alterações:

- Usaremos `"all"` como parâmetro para `pages`, para pegar tabelas de todas as páginas.
- Usaremos um `ExcelWriter` do `pandas` para extrair todas as tabelas para um mesmo arquivo Excel.
- Ignoraremos tabelas vazias ou com uma única coluna, pois vamos partir do princípio que são erros de leitura do `tabula-py`.

Para usarmos o `ExcelWriter`, precisamos instalar `pandas` explicitamente:

```
pip install pandas
```

Com as alterações propostas, o código fica:

```
from pathlib import Path
```

```
import pandas as pd
```

Lendo e Manipulando Arquivos PDF

```
import tabula

caminho_pdf = Path('materiais de aula') / 'documentos' / 'RI Ambev - 1T23.pdf'
tabelas = tabula.read_pdf(caminho_pdf, pages='all')

escritor_excel = pd.ExcelWriter('Tabelas Ambev.xlsx')

print(f'Foram encontradas {len(tabelas)} tabelas no arquivo PDF.')
for i, df in enumerate(tabelas, 1):
    print(f'\n\n----- Tabela {i} ----- \n')
    print(df)
    if df.empty or len(df.columns) < 2:
        continue
    df.to_excel(excel_writer=escritor_excel, sheet_name=f'Tabela {i}')

escritor_excel.close()
```

O resultado são 20 tabelas extraídas em um piscar de olhos! Segundo o output no terminal, mais de 270 tabelas foram identificadas, mas nosso filtro ajudou bastante a reduzir os falsos positivos.

Lidando com erros na extração

Se olharmos o arquivo Excel gerado, veremos que algumas tabelas possuem erros na sua extração.

Na aba Tabela 9, por exemplo, podemos ver que o cabeçalho ficou parcialmente misturado na tabela:

| A | B | C | D | E | F | G | H | I |
|----|---|------------|------------|-----------|-------------|------------|-----------|------------|
| | NAB Brasil | Unnamed: 0 | Unnamed: 1 | Conversão | Crescimento | Unnamed: 2 | % | Unnamed: 3 |
| 0 | R\$ milhões | 1T22 | Escopo | de Moeda | Orgânico | 1T23 | Reportado | % Orgânico |
| 1 | Volume ('000 hl) | 7.575,0 | - | - | 550,1 | 8.125,1 | 7,3% | 7,3% |
| 2 | | | | | | | | |
| 3 | Receita líquida | 1.498,0 | - | - | 278,5 | 1.776,6 | 18,6% | 18,6% |
| 4 | Receita líquida/hi (R\$) | 197,8 | - | - | 20,9 | 218,7 | 10,6% | 10,6% |
| 5 | CPV | (905,9) | - | - | (94,9) | (1.000,8) | 10,5% | 10,5% |
| 6 | CPV/hi (R\$) | (119,6) | - | - | (3,6) | (123,2) | 3,0% | 3,0% |
| 7 | CPV excl. deprec. & amort. | (851,3) | - | - | (89,3) | (940,5) | 10,5% | 10,5% |
| 8 | CPV/hi excl. deprec. & amort. (R\$) | (112,4) | - | - | (3,4) | (115,8) | 3,0% | 3,0% |
| 9 | Lucro bruto | 592,1 | - | - | 183,6 | 775,8 | 31,0% | 31,0% |
| 10 | % Margem bruta | 39,5% | - | - | 43,7% | 420pb | 420pb | |
| 11 | SG&A excl. deprec. & amort. | (356,5) | - | - | (90,0) | (446,5) | 25,2% | 25,2% |
| 12 | SG&A deprec. & amort. | (38,8) | - | - | (25,7) | (64,6) | 66,3% | 66,3% |
| 13 | SG&A total | (395,3) | - | - | (115,8) | (511,1) | 29,3% | 29,3% |
| 14 | Outras receitas/(despesas) operacionais | 54,7 | (13,7) | - | 56,8 | 78,7% | 138,5% | |
| 15 | Lucro operacional ajustado | 251,5 | (13,7) | - | 124,7 | 362,5 | 44,1% | 52,4% |
| 16 | % Margem de Lucro operacional ajustado | 16,8% | - | - | 20,4% | 360pb | 450pb | |
| 17 | EBITDA ajustado | 345,0 | (13,7) | - | 156,0 | 487,3 | 41,3% | 47,1% |
| 18 | % Margem EBITDA ajustado | 23,0% | - | - | 27,4% | 440pb | 530pb | |

• Destaques comerciais: marcas premium, de energeticos e health & wellness superaram o crescimento de volume total mais uma vez, impulsionadas principalmente por H2OH1, Gatorade e nosso portfólio diet/light/zero. O Guarani Antarctica apresentou crescimento de volume de um dígito médio, enquanto o volume da família de cola da Pepsi cresceu cerca de 15% (mid-teens), com a Pepsi Black mais do que triplicando seu volume em relação ao 1T22.

| NAB Brasil | 1T22 | Escopo | Conversão | Crescimento | 1T23 | % |
|---|---------|--------|-----------|-------------|-----------|--------|
| R\$ milhões | 7.575,0 | | de Moeda | Orgânico | 8.125,1 | 7,3% |
| Volume ('000 hl) | 7.575,0 | | | | 8.125,1 | 7,3% |
| Receita líquida | 1.498,0 | | | 278,5 | 1.776,6 | 18,6% |
| Receita líquida/hi (R\$) | 197,8 | | | 20,9 | 218,7 | 10,6% |
| CPV | (905,9) | | | (94,9) | (1.000,8) | 10,5% |
| CPV/hi (R\$) | (119,6) | | | (3,6) | (123,2) | 3,0% |
| CPV excl. deprec. & amort. | (851,3) | | | (89,3) | (940,5) | 10,5% |
| CPV/hi excl. deprec. & amort. (R\$) | (112,4) | | | (3,4) | (115,8) | 3,0% |
| Lucro bruto | 592,1 | | | 183,6 | 775,8 | 31,0% |
| % Margem bruta | 39,5% | | | 43,7% | 420pb | 420pb |
| SG&A excl. deprec. & amort. | (356,5) | | | (90,0) | (446,5) | 25,2% |
| SG&A deprec. & amort. | (38,8) | | | (25,7) | (64,6) | 66,3% |
| SG&A total | (395,3) | | | (115,8) | (511,1) | 29,3% |
| Outras receitas/(despesas) operacionais | 54,7 | (13,7) | | 56,8 | 78,7% | 138,5% |
| Lucro operacional ajustado | 251,5 | (13,7) | | 124,7 | 362,5 | 44,1% |
| % Margem de Lucro operacional ajustado | 16,8% | | | 20,4% | 360pb | 450pb |
| EBITDA ajustado | 345,0 | (13,7) | | 156,0 | 487,3 | 41,3% |
| % Margem EBITDA ajustado | 23,0% | | | 27,4% | 440pb | 530pb |

Vamos criar uma função para combinar o texto do cabeçalho com o da primeira linha, e aplicá-la sempre que houver uma coluna chamada Unnamed:

```
def ajusta_header(df):
    novo_header = []
    for header_label, first_row_label in zip(df.columns, df.iloc[0]):
        if header_label.startswith('Unnamed:'):
            novo_header.append(first_row_label)
        else:
            novo_header.append(f'{header_label} {first_row_label}')
    df.columns = novo_header
    return df.iloc[1:].reset_index(drop=True)
```

Lendo e Manipulando Arquivos PDF

Agora, podemos modificar nosso código para consertar os DataFrames com colunas Unnamed:

```
from pathlib import Path

import pandas as pd
import tabula

caminho_pdf = Path('materiais de aula') / 'documentos' / 'RI Ambev - 1T23.pdf'
tabelas = tabula.read_pdf(caminho_pdf, pages='all')

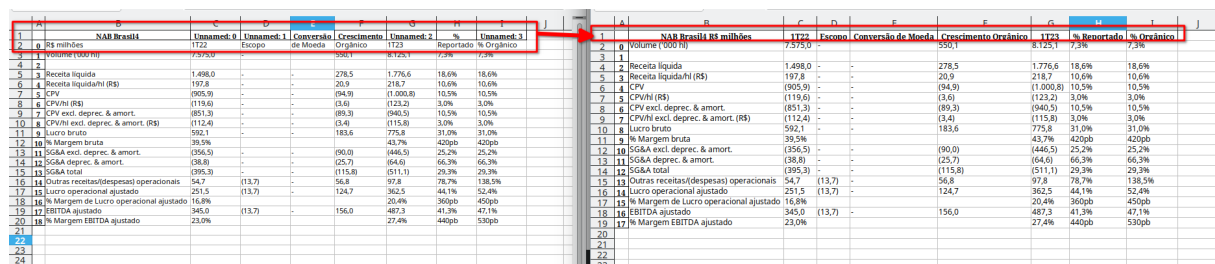
escritor_excel = pd.ExcelWriter('Tabelas Ambev.xlsx')

def ajusta_header(df):
    novo_header = []
    for header_label, first_row_label in zip(df.columns, df.iloc[0]):
        if header_label.startswith('Unnamed'):
            novo_header.append(first_row_label)
        else:
            novo_header.append(f'{header_label} {first_row_label}')
    df.columns = novo_header
    return df.iloc[1:].reset_index(drop=True)

print(f'Foram encontradas {len(tabelas)} tabelas no arquivo PDF.')
for i, df in enumerate(tabelas, 1):
    print(f'\n\n----- Tabela {i} ----- \n')
    print(df)
    if df.empty or len(df.columns) < 2:
        continue
    for col_name in df:
        if 'Unnamed' in col_name:
            df = ajusta_header(df=df)
            break
    df.to_excel(excel_writer=escritor_excel, sheet_name=f'Tabela {i}')

escritor_excel.close()
```

E conferindo o resultado:



| | A | B | C | D | E | F | G | H | I | J |
|----|----|-----------------|----------|----------|----------|----------|---------|---------|---------|---------|
| 1 | | NAB Brasil | Unamed:0 | Unamed:1 | Unamed:2 | Unamed:3 | | | | |
| 2 | 1 | RS milhões | 1122 | 1122 | 1122 | 1122 | 1122 | 1122 | 1122 | 1122 |
| 3 | 2 | Volume (000 It) | 7.575,0 | 7.575,0 | 7.575,0 | 7.575,0 | 7.575,0 | 7.575,0 | 7.575,0 | 7.575,0 |
| 4 | 3 | 1 | | | | | | | | |
| 5 | 4 | 2 | | | | | | | | |
| 6 | 5 | 3 | | | | | | | | |
| 7 | 6 | 4 | | | | | | | | |
| 8 | 7 | 5 | | | | | | | | |
| 9 | 8 | 6 | | | | | | | | |
| 10 | 9 | 7 | | | | | | | | |
| 11 | 10 | 8 | | | | | | | | |
| 12 | 11 | 9 | | | | | | | | |
| 13 | 12 | 10 | | | | | | | | |
| 14 | 13 | 11 | | | | | | | | |
| 15 | 14 | 12 | | | | | | | | |
| 16 | 15 | 13 | | | | | | | | |
| 17 | 16 | 14 | | | | | | | | |
| 18 | 17 | 15 | | | | | | | | |
| 19 | 18 | 16 | | | | | | | | |
| 20 | 19 | 17 | | | | | | | | |
| 21 | 20 | 18 | | | | | | | | |
| 22 | 21 | 19 | | | | | | | | |
| 23 | 22 | 20 | | | | | | | | |
| 24 | 23 | 21 | | | | | | | | |
| 25 | 24 | 22 | | | | | | | | |
| 26 | 25 | 23 | | | | | | | | |

Sucesso! O cabeçalho de diversas tabelas parece ter sido consertado!