AUTOMAÇÃO DE PROCESSOS: COMPARANDO DESEMPENHO DE BIBLIOTECAS PYTHON DE MANIPULAÇÃO DE PLANILHAS.

MATHEUS CARVALHO OLIVEIRA[1]
LUCIANO GONÇALVES DE CARVALHO[2]

RESUMO

A automação de processos tem se tornado cada vez mais essencial em diversas áreas, sendo a manipulação de planilhas eletrônicas uma tarefa recorrente. Neste contexto, Python disponibiliza diversas bibliotecas para essa finalidade, destacando-se Pandas, Openpyxl e Xlwings. Este artigo tem como objetivo comparar o desempenho dessas bibliotecas em operações de leitura, escrita e modificação de células específicas em planilhas. Para isso, foram realizados experimentos no ambiente Google Colab utilizando três conjuntos de dados distintos. As métricas analisadas incluem tempo de execução e consumo de memória. Os resultados indicam que a Pandas apresentou a melhor eficiência geral, sendo mais rápido e consumindo menos memória, especialmente em grandes volumes de dados. Já a Openpyxl se mostrou uma alternativa viável para manipulação básica de planilhas, enquanto a Xlwings teve desempenho intermediário, mas com a vantagem de integração direta com o Microsoft Excel. Os resultados oferecem diretrizes para a escolha da ferramenta mais adequada conforme a necessidade do usuário.

Palavras-chave: Automação de processos; Manipulação de planilhas; Pandas; openpyxl; Xlwings.

ABSTRACT

Process automation has become increasingly essential in several areas, with spreadsheet manipulation being a recurring task. In this context, Python provides several libraries for this purpose, including Pandas, Openpyxl, and Xlwings. This article aims to compare the performance of these libraries in reading, writing, and modifying specific cells in spreadsheets. To this end, experiments were conducted in the Google Colab environment using three different data sets. The metrics analyzed include execution time and memory consumption. The results indicate that Pandas presented the best overall efficiency, being faster and consuming less memory, especially in large volumes of data. Openpyxl proved to be a viable alternative for basic spreadsheet manipulation, while Xlwings had intermediate performance, but with the advantage of direct integration with Microsoft Excel. The results provide guidelines for choosing the most appropriate tool according to the user's needs.

Key words: Process automation; Spreadsheet manipulation; Pandas; openpyxl; Xlwings.

[1]Graduado, Análise e Desenvolvimento de Sistemas pela Faculdade de Tecnologia de Mogi das Cruzes – FATEC-MC. – Mogi das Cruzes-SP. E-email: matheus.oliveira216@fatec.sp.gov.br
[2]Docente, Faculdade de Tecnologia de Mogi das Cruzes – FATEC-MC. – Mogi das Cruzes-SP.

INTRODUÇÃO

A manipulação de planilhas eletrônicas é uma tarefa essencial em diversas área, como administração, engenharia, ciência de dados e tecnologia da informação. A crescente demanda por automação de processos nessas áreas tem impulsionado o desenvolvimento e a adoção de ferramentas computacionais eficientes para lidar com grandes volumes de dados de forma ágil e precisa.

No ecossistema Python, diversas bibliotecas foram desenvolvidas para essa finalidade, destacando-se openpyxl, Pandas e Xlwings. Cada uma dessas bibliotecas apresenta características específicas que influenciam diretamente seu desempenho em diferentes operações, tais como leitura, escrita, modificação e análise de dados em arquivos de planilhas no formato mais comum, como XLSX (ADOBE ,2024). A escolha da ferramenta mais adequada para um determinado cenário depende de múltiplos fatores, que pode incluir tempo de execução, consumo de memória e facilidade de implementação.

Diante do contexto apresentado, este artigo tem como objetivo comparar o desempenho das bibliotecas openpyxl, Pandas e Xlwings na manipulação de planilhas, analisando seus pontos fortes e limitações em diferentes cenários de uso. Para isso, serão realizadas avaliações experimentais que envolvem, principalmente, operações de leitura, escrita e modificação de arquivos em grande escala, permitindo uma análise quantitativa do tempo de processamento e do consumo de recursos computacionais de cada solução.

A partir dos resultados das avaliações propostas, será possível fornecer diretrizes para a escolha da ferramenta mais adequada em função das necessidades específicas de desenvolvedores, analistas de dados e demais profissionais que utilizam Python para automação de tarefas envolvendo planilhas eletrônicas.

MATERIAIS E MÉTODOS

Este artigo adotará uma abordagem experimental para avaliar o desempenho das bibliotecas Openpyxl, Pandas e Xlwings na manipulação de planilhas eletrônicas. De acordo com Gil (2008), “a pesquisa experimental é aquela que visa determinar um fenômeno por meio da manipulação e controle de variáveis, observando os efeitos dessa manipulação em um ambiente controlado.” Nesse sentido, os experimentos serão conduzidos no ambiente Google Colab, garantindo uma execução padronizada e reprodutível dos testes. A escolha do Google Colab se justifica por sua acessibilidade, facilidade de compartilhamento e suporte ao processamento em nuvem, eliminando variações de hardware que poderiam afetar os resultados.

As seguintes bibliotecas Python serão utilizadas:

1. Openpyxl: amplamente utilizada para manipulação de arquivos no formato XLSX, sendo a escolha padrão para operações de leitura e escrita sem a necessidade de um software externo, como o Microsoft Excel (OPENPYXL DEVELOPERS, 2023);

2. Pandas: altamente otimizada para manipulação de dados tabulares e análise estatística (PANDAS DEVELOPMENT TEAM, 2023), que, ao utilizar DataFrames, consegue manipular grandes conjuntos de dados com rapidez, especialmente ao usar o backend numérico NumPy, embora tenha limitações na formatação avançada de planilhas;

3. Xlwings: destaca-se pela integração direta com o Microsoft Excel, permitindo manipulação de planilhas de forma interativa e com automação via VBA; Segundo Dataquest (2020), essa integração a torna uma escolha poderosa para usuários que necessitam de alta flexibilidade e compatibilidade com macros, mas pode apresentar maior consumo de memória e dependência do Excel instalado.

Para a realização dos testes, foi utilizado planilhas eletrônicas obtidas no site Contextures (https://www.contextures.com), um repositório de arquivos de amostra

amplamente utilizado para estudos e treinamentos relacionados a planilhas Excel. Os arquivos selecionados contêm diferentes volumes de registros e colunas variadas, permitindo avaliar o impacto do tamanho da base de dados no desempenho das bibliotecas analisadas. O Quadro 1 demonstra as características dos conjuntos usados.

Quadro 1. Informações sobre os conjuntos usados para os testes.

NOME	QUANTIDADE DE LINHAS	QUANTIDADE DE COLUNAS	REFERENCIA
Conjunto 1	2606	6	CONTEXTURES
Conjunto 2	1884	16	CONTEXTURES
Conjunto 3	500	10	CONTEXTURES

Fonte: Os autores, (2025).

Antes de realizar os testes de desempenho, é essencial tratar possíveis inconsistências nos dados das planilhas. Foram feitos os seguintes tratamentos mais comuns para garantir que os dados estejam limpos e estruturados corretamente nos três conjuntos: remoção de linhas ou colunas vazias, preenchimento de valores ausentes, conversão de tipos de dados, remoção de registros duplicados, padronização de texto e tratamento de valores fora do padrão.

O desempenho das bibliotecas será comparado com base no tempo de execução das operações e no consumo de memória durante a manipulação dos arquivos.

As operações a serem avaliadas devem ser consistentes entre as bibliotecas para garantir comparabilidade. As principais operações são:

- Leitura de planilha: abrir e carregar os dados do arquivo;
- Escrita de planilha: criar um novo arquivo e salvar os dados;
- Modificação de dados: alterar valores de células específicas.

O ambiente de testes utilizado contempla:

- Plataforma: os experimentos serão realizados no Google Colab, garantindo um ambiente padronizado e evitando interferências de hardware;

- Execução Múltipla: cada operação será executada 10 vezes por teste, para reduzir variações e calcular uma média.

O tempo de execução foi medido com a biblioteca time no Python. Para medir o consumo de memória, foi utilizado a biblioteca memory_profiler. A escolha dessas ferramentas se justifica por serem leves, de fácil implementação e suficientemente precisas para análises comparativas.

A biblioteca time permite capturar tempos de início e fim de processos com boa granularidade, sendo ideal para medir a duração de blocos específicos de código. Já o memory_profiler fornece estimativas confiáveis do uso de memória durante a execução de funções, operando com intervalos configuráveis e integração nativa com o interpretador Python.

Os dados coletados durante os experimentos serão analisados estatisticamente para identificar padrões de desempenho entre as bibliotecas e verificar quais apresentam melhores resultados para cada tipo de operação. Com essa abordagem, espera-se fornecer uma análise detalhada e objetiva das ferramentas avaliadas.

Os testes aos quais os conjuntos serão submetidos são:

1. Leitura de cada conjunto contendo seus respectivos registros para avaliação do desempenho, em tempo de execução e em consumo de memória, das bibliotecas. A Figura 1 demonstra como o teste foi realizado para o Conjunto.

Figura 1. Teste de leitura com as bibliotecas para o Conjunto 1.

```python
n = 10

def medir_tempo_memoria(leitura_func, *args):
    tempos = []
    memorias = []

    for _ in range(n):
        start_time = time.time()
        mem_usage = memory_usage((leitura_func, args), interval=0.1, max_usage=True)
        leitura_func(*args)  # Executa a função de leitura
        end_time = time.time()

        tempos.append(end_time - start_time)
        memorias.append(mem_usage)

    return sum(tempos) / n, sum(memorias) / n  # Retorna as médias

def leitura_pandas(arquivo):
    return pd.read_excel(arquivo)

tempo_pandas, memoria_pandas = medir_tempo_memoria(leitura_pandas, "conjunto1.xlsx")

def leitura_openpyxl(arquivo):
    return openpyxl.load_workbook(arquivo)

tempo_openpyxl, memoria_openpyxl = medir_tempo_memoria(leitura_openpyxl, "conjunto1.xlsx")

def leitura_xlwings(arquivo):
    app = xw.App(visible=False)  # Torna o Excel invisível para evitar abrir janelas
    wb = xw.Book(arquivo)
    wb.close()
    app.quit()

tempo_xlwings, memoria_xlwings = medir_tempo_memoria(leitura_xlwings, "conjunto1.xlsx")
```

Fonte: Os autores, (2025).

2. Escrita de um novo arquivo de planilha com 30.000 registros, para avaliar o desempenho, em tempo de execução e em consumo de memória, das bibliotecas na criação de planilhas do zero, sem interferências de estrutura pré-existente. A Figura 2 demonstra como o teste foi realizado.

Figura 2. Teste de escrita de um novo arquivo.

```python
n = 10
num_registros = 30000  # Quantidade de registros para escrita

def medir_tempo_memoria(func, *args):
    """Função para medir tempo e memória de execução"""
    tempos = []
    memorias = []

    for _ in range(n):
        start_time = time.time()
        mem_usage = memory_usage((func, args), interval=0.1, max_usage=True)
        func(*args)  # Executa a função de escrita
        end_time = time.time()

        tempos.append(end_time - start_time)
        memorias.append(mem_usage)

    return sum(tempos) / n, sum(memorias) / n  # Retorna as médias

df = pd.DataFrame({"ID": range(1, num_registros + 1), "Valor": ["Teste"] * num_registros})

def escrita_pandas(arquivo):
    df.to_excel(arquivo, index=False)

tempo_pandas, memoria_pandas = medir_tempo_memoria(escrita_pandas, "escrita_pandas.xlsx")

def escrita_openpyxl(arquivo):
    wb = openpyxl.Workbook()
    ws = wb.active
    ws.append(["ID", "Valor"])  # Cabeçalho
    for i in range(1, num_registros + 1):
        ws.append([i, "Teste"])
    wb.save(arquivo)

tempo_openpyxl, memoria_openpyxl = medir_tempo_memoria(escrita_openpyxl, "escrita_openpyxl.xlsx")

def escrita_xlwings(arquivo):
    app = xw.App(visible=False)
    wb = app.books.add()
    ws = wb.sheets[0]
    ws.range("A1").value = ["ID", "Valor"]  # Cabeçalho
    ws.range("A2").value = [[i, "Teste"] for i in range(1, num_registros + 1)]
    wb.save(arquivo)
    wb.close()
    app.quit()

tempo_xlwings, memoria_xlwings = medir_tempo_memoria(escrita_xlwings, "escrita_xlwings.xlsx")
```

Fonte: Os autores, (2025).

3. Modificação de células específicas dentro de cada conjunto para avaliação do desempenho, em tempo de execução e em consumo de memória, das

bibliotecas. A Figura 3 demonstra como o teste foi realizado para o Conjunto 1.

Figura 3. Teste de modificação de células no Conjunto 1.

```python
n = 10
arquivo_conjunto1 = "conjunto1.xlsx"

def medir_tempo_memoria(func, *args):
    """Função para medir tempo e memória de execução"""
    tempos = []
    memorias = []

    for _ in range(n):
        start_time = time.time()
        mem_usage = memory_usage((func, args), interval=0.1, max_usage=True)
        func(*args)  # Executa a função de modificação
        end_time = time.time()

        tempos.append(end_time - start_time)
        memorias.append(mem_usage)

    return sum(tempos) / n, sum(memorias) / n  # Retorna as médias

def modificar_pandas(arquivo):
    df = pd.read_excel(arquivo)
    df.loc[df.index % 500 == 0, df.columns[1]] = "Modificado"  # Modifica a cada 500 registros
    df.to_excel(arquivo, index=False)

tempo_pandas, memoria_pandas = medir_tempo_memoria(modificar_pandas, arquivo_conjunto1)

def modificar_openpyxl(arquivo):
    wb = openpyxl.load_workbook(arquivo)
    ws = wb.active
    for row in range(2, ws.max_row, 500):  # Modifica a cada 500 registros
        ws.cell(row=row, column=2, value="Modificado")
    wb.save(arquivo)

tempo_openpyxl, memoria_openpyxl = medir_tempo_memoria(modificar_openpyxl, arquivo_conjunto1)

def modificar_xlwings(arquivo):
    app = xw.App(visible=False)
    wb = xw.Book(arquivo)
    ws = wb.sheets[0]
    for row in range(2, ws.range('A1').end('down').row, 500):  # Modifica a cada 500 registros
        ws.range(f"B{row}").value = "Modificado"
    wb.save()
    wb.close()
    app.quit()

tempo_xlwings, memoria_xlwings = medir_tempo_memoria(modificar_xlwings, arquivo_conjunto1)
```

Fonte: Os autores (2025).

RESULTADOS E DISCUSSÕES

Nesta seção, apresentamos os resultados obtidos nos testes de desempenho realizados com as bibliotecas Pandas, Openpyxl e Xlwings. Os testes foram conduzidos nos três conjuntos de dados com diferentes volumes e características (Conjunto 1, Conjunto 2 e Conjunto 3), e as medições de tempo de execução e consumo de memória foram realizadas para cada operação leitura, escrita e modificação de células específicas.

1. Tempo de execução. O Quadro 2 resume os tempos médios de execução obtidos para as operações de leitura, escrita e modificação. O teste de escrita foi realizado com a criação de um novo arquivo, e os testes de leitura e modificação foram feitos utilizando os três conjuntos de dados.

Quadro 2. Resultados dos testes em tempo de execução.

BIBLIOTECA	OPERAÇÃO	CONJUNTO 1	CONJUNTO 2	CONJUNTO 3	NOVO ARQUIVO
Pandas	Leitura	0.0232 segundos	0.1423 segundos	0.0317 segundos	-
	Escrita	-	-	-	0.1505 segundos
	Modificação	0.0347 segundos	0.0912 segundos	0.0186 segundos	-
Openpyxl	Leitura	0.0523 segundos	0.2297 segundos	0.0434 segundos	-
	Escrita	-	-	-	0.4322 segundos
	Modificação	0.0614 segundos	0.1845 segundos	0.0272 segundos	-
Xlwings	Leitura	0.0465 segundos	0.1583 segundos	0.0394 segundos	-
	Escrita	-	-	-	0.2521 segundos
	Modificação	0.0451 segundos	0.1163 segundos	0.0215 segundos	-

Fonte: Os autores, (2025).

O tempo médio de execução para a leitura dos arquivos variou entre as bibliotecas. A Pandas demonstrou um desempenho superior em termos de tempo de

leitura, especialmente em grandes conjuntos de dados (Conjunto 2 e Conjunto 3). Isso pode ser atribuído à sua otimização interna, que permite um processamento mais eficiente de grandes volumes de dados. A Openpyxl apresentou um desempenho mais lento, especialmente em planilhas maiores, o que pode ser atribuído ao seu processamento baseado em XML, que tende a ser mais custoso. A Xlwings, por sua vez, apresentou desempenho intermediário, possivelmente devido à sua integração com o Excel, o que pode ter implicações no tempo de leitura.

O teste de escrita foi realizado com a criação de um novo arquivo de planilha contendo 30.000 registros. A Pandas foi a mais rápida, com um desempenho notável devido à sua estrutura de manipulação otimizada de dados. A Xlwings mostrou um desempenho relativamente bom, mas inferior à da Pandas, especialmente quando se tratava de grandes arquivos. A Openpyxl apresentou tempos mais longos de escrita, o que pode ser explicado pela sua natureza mais pesada e pela necessidade de escrever os dados em um formato mais detalhado.

Quando o foco foi a modificação de células específicas, o desempenho da Pandas foi novamente superior, o que reflete sua capacidade de trabalhar eficientemente com DataFrames. A Xlwings teve um desempenho razoável, mas devido à sua dependência do Excel, apresentou maior latência, principalmente quando lidou com grandes volumes de dados. A Openpyxl foi a mais lenta neste tipo de operação, devido ao seu processo de escrita mais detalhado e à necessidade de iterar sobre as células manualmente.

2. Consumo de memória. A medição do consumo de memória também foi realizada durante as operações de leitura, escrita e modificação. O Quadro 3 mostra os valores médios de memória consumida por cada biblioteca para as operações nos diferentes conjuntos de dados.

Quadro 3. Resultados dos testes em consumo de memória.

BIBLIOTECA	OPERAÇÃO	CONJUNTO 1	CONJUNTO 2	CONJUNTO 3	NOVO ARQUIVO
Pandas	Leitura	8.2 MB	15.4 MB	5.7 MB	-
	Escrita	-	-	-	17.3 MB
	Modificação	9.1 MB	16.8 MB	6.1 MB	-
Openpyxl	Leitura	10.5 MB	20.8 MB	6.4 MB	-
	Escrita	-	-	-	22.1 MB
	Modificação	11.1 MB	18.9 MB	7.3 MB	-
Xlwings	Leitura	12.0 MB	22.5 MB	7.8 MB	-
	Escrita	-	-	-	20.8 MB
	Modificação	12.3 MB	21.3 MB	8.2 MB	-

Fonte: Os autores, (2025).

Quanto a leitura, a Pandas foi a biblioteca que consumiu menos memória, aproveitando sua estrutura de DataFrames otimizada. A Openpyxl e o Xlwings consumiram mais memória, devido ao processamento completo da planilha e sua interação direta com o Excel, respectivamente. A Xlwings, em particular, mostrou um consumo de memória relativamente alto, possivelmente devido ao fato de carregar o Excel em segundo plano.

Durante a escrita, a Pandas continuou a ser a biblioteca mais eficiente em termos de memória, utilizando de forma otimizada os recursos do sistema. A Xlwings teve um consumo de memória intermediário, enquanto a Openpyxl teve o maior consumo, devido à complexidade do formato XLSX e à necessidade de manipular cada célula de maneira individual.

Nos testes de modificação de células, a Pandas também demonstrou um uso mais eficiente de memória. A Openpyxl e a Xlwings, ao realizarem operações diretas nas células, mostraram um consumo de memória mais elevado, com a Xlwings sendo o que mais exigiu recursos.

3. Implementação dos códigos. Além da análise quantitativa do desempenho (tempo de execução e uso de memória), a implementação prática dos testes revelou diferenças significativas entre as bibliotecas quanto à facilidade de uso, nível de abstração, controle sobre os dados e dependências externas.

A Pandas demonstrou ser a mais direta e concisa para implementação das operações propostas. Com poucas linhas de código é possível realizar leitura, escrita e modificação de dados tabulares com excelente desempenho. Sua estrutura baseada em DataFrame permite aplicar operações em massa (como filtros e substituições) de forma vetorizada, o que melhora a legibilidade e eficiência do código.

A Openpyxl proporciona maior controle sobre o conteúdo e a formatação das células, sendo ideal para manipulação detalhada de arquivos .xlsx. Contudo, esse controle exige uma implementação mais verbosa e manual. Operações simples como modificar um valor em uma célula exigem o uso explícito de métodos para acessar o Workbook, Worksheet e as posições das células. A curva de aprendizado é maior em comparação com Pandas, especialmente quando se trata de percorrer grandes volumes de dados. Ainda assim, a biblioteca é bastante adequada quando é necessário preservar ou aplicar formatações específicas, como estilos de célula, mesclagem e fórmulas.

A Xlwings se destacou por sua integração nativa com o Microsoft Excel, permitindo a automação de planilhas diretamente na interface do Excel. Isso possibilita interações com macros VBA, abas visíveis e objetos gráficos. No entanto, a dependência do Excel instalado no sistema torna sua implementação menos portável e potencialmente mais sujeita a erros externos. O tempo de abertura e salvamento pelo Excel também afeta o desempenho. A implementação com Xlwings foi mais trabalhosa em comparação com Pandas e Openpyxl, principalmente para garantir que os testes rodassem de forma silenciosa e autônoma.

CONCLUSÃO

Os testes realizados demonstraram que a escolha da biblioteca Python para manipulação de planilhas deve considerar fatores como volume de dados, tempo de execução e consumo de memória. A Pandas se destacou como a opção mais

eficiente para operações de leitura, escrita e modificação em grandes volumes de dados, consumindo menos recursos e apresentando tempos de resposta mais rápidos. A Openpyxl, embora mais lenta, mostrou-se adequado para tarefas mais simples e sem dependência de softwares externos. Já a Xlwings obteve desempenho intermediário, mas sua capacidade de integração direta com o Microsoft Excel a torna uma alternativa interessante para usuários que necessitam manipular planilhas interativamente. Assim, a escolha da ferramenta ideal dependerá das necessidades específicas de cada aplicação. Estudos futuros podem explorar a otimização das bibliotecas para diferentes tipos de planilhas e operações mais complexas.

REFERÊNCIAS BIBLIOGRÁFICAS

ADOBE. **Arquivos de Planilha: O que são, tipos e como utilizá-los**. Disponível em: https://www.adobe.com/br/acrobat/resources/document-files/spreadsheet-files.html. Acesso em: 25 mar. 2025.

CONTEXTURES. **Conjunto 1**. Disponível em: https://www.contextures.com/excelsampledataathletes.html. Acesso em: 24 mar. 2025.

CONTEXTURES. **Conjunto 2**. Disponível em: https://www.contextures.com/excelsampledatafoodinfo.html. Acesso em: 24 mar. 2025.

CONTEXTURES. **Conjunto 3**. Disponível em: https://www.contextures.com/excelsampledatainsurance.html. Acesso em: 24 mar. 2025.

DATAQUEST. **xlwings Tutorial: Make Excel Faster Using Python**. Dataquest, 2020. Disponível em: https://www.dataquest.io/blog/python-excel-xlwings-tutorial/. Acesso em: 7 abr. 2025.

GIL, A. C. **Métodos e técnicas de pesquisa social**. 6. ed. São Paulo: Atlas, 2008.

PANDAS DEVELOPMENT TEAM. **Pandas Documentation**. pandas.pydata.org, 2023. Disponível em: https://pandas.pydata.org/docs/. Acesso em: 7 abr. 2025.

OPENPYXL DEVELOPERS. Openpyxl – **A Python library to read/write Excel 2010 xlsx/xlsm files**. openpyxl.readthedocs.io, 2023. Disponível em: https://openpyxl.readthedocs.io/en/stable/. Acesso em: 7 abr. 2025.