SIMPÓSIO DE EXCELÊNCIA EM GESTÃO E TECNOLOGIA
XXIISEGeT
Economia Circular: Desafios para construir uma sociedade sustentável
ECONOMIA CIRCULAR
Realização:
UniDomBosco Centro Universitário Dom Bosco do Rio de Janeiro
unesp

Automação do Controle de Contas a Pagar com Python: Integração de Dados Financeiros ao Calendário Corporativo

Daniel Vitor da Silva Teodoro
daniel.vitor@aedb.br
AEDB

Anderson Fernandes de Barros - Orientador
anderson.barros@aedb.br
AEDB

Resumo: Este artigo apresenta um estudo de caso aplicado ao setor varejista, com foco na automação do controle de contas a pagar por meio da linguagem de programação Python. A pesquisa teve como objetivo principal desenvolver uma solução de baixo custo, baseada em código aberto, que integre dados financeiros extraídos do sistema interno da empresa com o calendário do Microsoft Outlook, otimizando a visualização e a gestão das obrigações financeiras. Para isso, foram utilizadas bibliotecas como pandas, datetime, win32com e pyautogui, possibilitando a criação automatizada de eventos financeiros distribuídos por data do pagamento. A metodologia adotada é de natureza aplicada e com métodos mistos, incluindo mapeamento de processos, desenvolvimento de scripts personalizados e integração sistêmica com foco na redução de tarefas manuais repetitivas. Como resultado, obteve-se um calendário financeiro dinâmico e centralizado, promovendo ganhos significativos em controle, organização e confiabilidade das informações. A pesquisa evidencia o potencial da automação como estratégia viável para a implementação digital no varejo, demonstrando como a tecnologia pode ser aplicada para resolver desafios operacionais cotidianos.

Palavras Chave: Automação - Python - Varejo - Microsoft Outlook - Financeiro

1. INTRODUÇÃO

Com o advento de tecnologias oriundas da Indústria 4.0, empresas de diversos setores vêm buscando se adaptar a rápida e flexível era digital, almejando o aprimoramento contínuo dos processos que envolvem o negócio. Com isso, surge a necessidade de automação dos processos, que atualmente são atividades manuais e repetitivas, objetivando a melhoria dos indicadores de qualidade, rapidez, eficiência, confiabilidade e custo.

No segmento do varejo, um mercado onde características como alto volume de informações, que se transformam em dados, e um efetivo controle de prazos de pagamentos das contas, a utilização padronizada de tecnologias promove, no âmbito estratégico, uma vantagem competitiva sustentável. A automação de tarefas administrativas, como por exemplo o controle de contas a pagar, contribui diretamente para a gestão de compras de uma empresa, oferecendo suporte na escolha do melhor dia de faturamento.

Neste sentido, a linguagem *Python* tem se mostrado uma solução viável e acessível para a automação de processos. Ela conta uma ampla comunidade, simplicidade de aprendizado e capacidade de integração com outras plataformas, como o Microsoft Outlook. O uso de bibliotecas específicas como pandas (tratamento de dados), datetime (formatação de datas) e win32com (conexão com APIs Microsoft) torna possível a leitura de planilhas, processamento de dados e a inserção de eventos diretamente no calendário da empresa.

Logo, o objetivo deste artigo é apresentar um estudo de caso que será desenvolvido por meio de um *script* em *Python* com o intuito de criar e automatizar um calendário financeiro da empresa. A proposta consiste na leitura dos dados lançados no sistema interno da constituição, sobre as contas a pagar, sejam elas efetivo ou em estado de previsão. Essas informações serão padronizadas e posteriormente integradas com o calendário do Microsoft Outlook. Como resultado, as obrigações financeiras da empresa irão passar a serem visualizadas facilmente, de forma clara, centralizada e atualizada em um calendário dia a dia no Outlook, otimizando a rotina das áreas financeira e de compras.

O estudo contribui para a discussão sobre transformação digital no segmento do varejo e sua aplicabilidade, ou seja, como as novas ferramentas que vêm surgindo com o avanço tecnológico podem ser utilizadas para resolver problemas operacionais cotidianos com baixo custo e alto impacto. Além disso, demonstra a relevância da automação de processos como caminho para auxiliar os diversos setores de uma instituição.

2. PROBLEMA DE PESQUISA E OBJETIVO

Em se tratando do comportamento do mercado atual, especificamente no setor comercial, a gestão das contas a pagar representam grandes desafios para os departamentos financeiro e de compras de forma geral. A necessidade por processos manuais e repetitivos, como a geração de relatórios, verificação de planilhas, tratamento de dados e organização de vencimentos comprometem a produtividade do setor. Apesar da disponibilidade gratuita de tecnologias de fácil aprendizado e implementação, como a linguagem *Python*, a grande maioria das empresas desse setor ainda não exploram o potencial imenso que essa ferramenta oferece para automatizar tarefas rotineiras, e principalmente para integrar ela a sistemas internos ou sistemas amplamente utilizados como o Microsoft Outlook. Diante desse cenário, surge a seguinte questão problema:

Como a linguagem *Python* pode ser aplicada para facilitar o controle de contas a pagar, de modo a otimizar a organização financeira e distribuir os valores, buscando não sobrecarregar um único dia.

Todavia, este artigo tem como objetivo principal desenvolver uma solução, utilizando a linguagem de programação *Python*, para integrar e gerenciar informações financeiras em um calendário, promovendo a otimização do controle das contas a pagar e facilitando a tomada de decisões estratégicas.

Em consonância com o objetivo principal, faz-se necessário alguns objetivos específicos para um melhor entendimento:

- Mapear as etapas manuais dos processos de contas a pagar executados pela empresa;
- Implementar *scripts* em *Python* com uso de bibliotecas como pandas, datetime, pyautogui e win32com, para automação de tarefas rotineiras;
- Integrar os dados de planilhas Excel ao Outlook, criando eventos detalhados no calendário com base nos valores e vencimentos;
- Entregar um calendário de contas a pagar por dia dos próximos 8 meses.

3. FUNDAMENTAÇÃO TEÓRICA

Conforme a digitalização do setor varejista vem crescendo, surgiu também uma nova necessidade por profissionais com habilidades voltadas à automação de processos e análise de dados em tempo real. Como ressalta Carvalho (2024), ferramentas como *Python*, SQL e Power BI, vem se tornando cada vez mais relevantes no novo perfil dos profissionais de

finanças e varejo, contribuindo para uma atuação mais assertiva frente às oscilações do mercado. Assim reforçando que a combinação de habilidades técnicas (*hard skills*) com competências analíticas proporciona uma resiliência digital indispensável em um contexto competitivo e incerto.

Além disso, a integração de sistemas corporativos através de tecnologias como a automação via *scripts* em *Python* permite ganhos expressivos de desempenho. De acordo com Meirelles (2023), a integração de sistemas é hoje uma das principais prioridades estratégicas das empresas, especialmente no varejo, onde a necessidade de conectividade entre canais físicos e digitais é vital. Sistemas bem integrados garantem velocidade e precisão nos dados e otimizam a gestão operacional das organizações.

Segundo Rosa (2024), o uso de *Python* na automação de processos empresariais também tem se destacado por sua versatilidade e acessibilidade. Em um estudo aplicado à análise de dados de faturamento no varejo, foi demonstrado que a automatização com *Python* proporciona não apenas economia de tempo, mas também a redução de erros em registros críticos. A estrutura clara e orientada a dados dessa linguagem a torna ideal para operações que exigem precisão e atualização constante.

A linguagem *Python* tem se destacado como uma ferramenta eficaz para automatizar tarefas operacionais no varejo. Atividades como organização de planilhas, envio de e-mails e extração de dados podem ser realizadas com poucos comandos. Isso reduz o tempo gasto com tarefas manuais e minimiza falhas humanas. Sua simplicidade permite aplicação mesmo por profissionais sem formação técnica. (Sweigart, 2019).

A automação com *Python* tem se consolidado como ferramenta estratégica no varejo moderno, ao permitir maior agilidade e precisão na análise de dados. De acordo com McKinney (2022), bibliotecas como *pandas* e *NumPy* viabilizam a automatização de tarefas repetitivas, como geração de relatórios e monitoramento de vendas. Isso permite que empresas tomem decisões com base em dados em tempo real. A integração desses processos com sistemas internos reduz erros humanos e aumenta a produtividade. No contexto do varejo, essas aplicações favorecem uma atuação mais competitiva. Assim, *Python* se destaca como recurso acessível e poderoso para automação orientada a dados.

As tecnologias digitais vêm transformando profundamente o setor varejista, promovendo experiências mais personalizadas e eficientes para o consumidor. Recursos como automação, inteligência artificial e análise de dados possibilitam decisões mais rápidas, gestão dinâmica de estoques e integração de canais físicos e digitais. Essas soluções aumentam a competitividade das empresas e atendem melhor às novas exigências do mercado. O varejo

inteligente conecta dados, comportamento do cliente e operações em tempo real. (Heinemann, 2021).

O conceito de realizar a análise e coleta de dados dentro da sua empresa se deve essencial devido ao fato de os dados estarem interligados, e serem considerados vitais para o gerenciamento de qualquer negócio, tendo em vista que um setor inteiro pode fazer uso de relatórios que mostram a situação atual da empresa, prevê uma situação futura e auxilia na tomada de decisões baseadas em informações em tempo real (Sharda, Delen & Turban, 2019).

De acordo com (Padilha, Soares, Alves, 2022), independentemente do tamanho dos dados envolvidos, a padronização e apresentação de dados é essencial para qualquer análise. Quando as informações são apresentadas de maneira padronizada, se torna mais simples e fácil para os usuários entenderem e analisar os dados, a consistência também é fundamental para realizar comparações. Além disso, a padronização permite que os responsáveis por tomar de decisão importantes identifiquem e ajam com base nas informações relevantes rapidamente.

Em conformidade com (Neto, Marques, 2020) a imensa quantidade de dados disponíveis atualmente representa um desafio e uma oportunidade para as organizações. Enquanto a quantidade de informações pode ser esmagadora, é a habilidade de discernir, analisar e aplicar esses dados que distingue as empresas bem-sucedidas. A tomada de decisão baseada em dados requer uma combinação de tecnologia, intuição e estratégia, o que garante uma vantagem competitiva no mercado dinâmico atual.

4. METODOLOGIA

A metodologia empregada neste artigo é classificada como pesquisa de natureza aplicada, com abordagem exploratória e utilização de métodos mistos, ou seja, combinação de análises qualitativas e quantitativas. A escolha metodológica está alinhada à proposta de desenvolver uma solução para automatizar o controle de contas a pagar, utilizando a linguagem de programação Python. Para isso, foi realizada uma investigação baseada em estudo de caso, com o mapeamento do processo, análise dos fluxos operacionais existentes e implementação de scripts personalizados. O foco esteve na criação de uma rotina automatizada capaz de processar informações financeiras e integrá-las ao calendário do Microsoft Outlook, melhorando a visualização e a organização das obrigações da empresa. Conforme demonstrado no fluxograma da Figura 1.

Figura 1: Fluxograma.
Fonte: Elaborada pelo autor (2025).

Na sequência, o editor Jupiter notebook juntamento com Visual Studio Code (VS Code) foi instalado como ambiente de desenvolvimento. Dentro dele, foram adicionadas as extensões "*Python*" e "*Jupyter*", permitindo a criação de notebooks com células de código. Essa estrutura facilitou o teste progressivo dos comandos e a depuração dos *scripts* durante o desenvolvimento da automação. A Figura 2 demonstra um SIPOC de todo o processo.

Figura 2: SIPOC.
Fonte: Elaborada pelo autor (2025).

Após configurar o ambiente, foram instaladas as bibliotecas necessárias via terminal com o comando “pip install”. A principal biblioteca utilizada foi a PyAutoGUI, responsável por simular interações humanas, como movimentar o mouse, clicar, pressionar teclas e digitar. Também foram usadas bibliotecas complementares, como time (controle de pausas), pandas (tratamento de dados) e win32com.client (conexão com o outlook).

Com as ferramentas preparadas, foi feito um mapeamento detalhado das tarefas a serem automatizadas. Isso incluiu a identificação de cada etapa da rotina manual, desde a abertura de sistemas e navegação em menus até o preenchimento de campos e exportação de arquivos. A lógica foi estruturada para simular essas ações na ordem correta, com o controle do tempo entre os comandos.

O desenvolvimento dos scripts foi realizado com o uso de notebooks do Jupyter, permitindo testar o código por blocos. A automação foi construída com comandos da PyAutoGUI para localizar áreas da tela, clicar em botões, preencher campos e navegar entre janelas. Com o uso do time.sleep(), foi possível garantir que os sistemas tivessem tempo de resposta suficiente entre as etapas.

5. RESULTADOS

O início do *script Python* desenvolvido para este estudo de caso reúne as importações indispensáveis das bibliotecas pandas, datetime, os e win32com, estabelecendo assim as bases para a leitura de planilhas do Excel. Esses arquivos contêm dados essenciais, que constituem para a automação de eventos no calendário do Outlook. Dessa forma, tais planilhas são cruciais para alimentar todo o processo, assegurando a integração e a organização necessárias das informações financeiras projetadas e realizadas.

Além da geração automatizada de eventos de pagamento, o código implementado obteve ganhos significativos em termos de visualização e distribuição das obrigações financeiras ao longo dos próximos 8 meses. A inclusão de eventos agregados por dia, com a soma total de pagamentos previstos e efetivos, facilitou o monitoramento das datas com maior concentração de despesas. Isso garante que a equipe do setor de compras consiga ajustar os prazos de faturamento de forma mais estratégica, evitando acúmulo de compromissos em dias críticos, tais como o 5º dia útil, reservado para a folha de pagamento.

Nas Figuras 3 e 4, apresenta-se a primeira metade do *script Python*, que foi elaborado para gerenciar, de maneira automatizada, as contas a pagar no calendário do Microsoft

Outlook. Este código é a principal ferramenta do projeto e será analisado em maior profundidade nos parágrafos seguintes, onde se destacará sua estrutura, suas funcionalidades e a relevância que assume para o andamento deste estudo de caso.

```python
import pandas as pd
from datetime import datetime
import os
import win32com.client as win32

# Caminho base
base_path = r"Z:\Daniel\Calendario_outlook_pag\Texte_Calendario.xlsx"
Prev = pd.read_excel(base_path, sheet_name="Previsão")
Efet = pd.read_excel(base_path, sheet_name="Efetivo")
Orig = pd.read_excel(base_path, sheet_name="Base_Original")
```

Figura 3: Código Parte 01.
Fonte: Elaborada pelo autor (2025).

```python
# Importar e ajeitar a Tabela original
Orig["Parcela"] = Orig["Parcela"].str.replace("_x000F_", "", regex=False)
Orig["Venc."] = pd.to_datetime(Orig["Venc."])

# Função para limpar strings e converter para float
def limpar_valor(valor):
    if isinstance(valor, str):
        valor = valor.replace("R$", "").replace(" ", "").replace(".",
"").replace(",", ".")
    try:
        return float(valor)
    except:
        return None

# Converter 'Valor' para float
Orig["Valor"] = Orig["Valor"].apply(limpar_valor)

# Garantir que a coluna Previsão existe
if "Previsão" not in Orig.columns:
    Orig["Previsão"] = ""

# Criar data formatada
Orig["Venc_fmt"] = Orig["Venc."].dt.strftime("%d/%m/%Y")

# Agrupamento simples (ainda usado por compatibilidade, mas não será usado no body)
agrupado = Orig.groupby("Venc.")["Fornecedor"].apply(list).reset_index()
agrupado["Venc_fmt"] = agrupado["Venc."].dt.strftime("%d/%m/%Y")
agrupado = agrupado.sort_values("Venc.")[["Venc_fmt", "Fornecedor"]]
fornecedores_por_data = dict(zip(agrupado["Venc_fmt"], agrupado["Fornecedor"]))

# Formatar Previsão e Efetivo
Prev["Previsão"] = Prev["Previsão"].apply(limpar_valor)
Efet["Efetivo"] = Efet["Efetivo"].apply(limpar_valor)

formata_moeda = lambda x: f"R$ {x:,.2f}".replace(",", "v").replace(".",
",").replace("v", ".") if pd.notnull(x) else ""
Prev["Previsão"] = Prev["Previsão"].apply(formata_moeda)
Efet["Efetivo"] = Efet["Efetivo"].apply(formata_moeda)

# Conversão robusta de datas com timezone
def converter_data(data):
    try:
        ts = pd.to_datetime(data, dayfirst=True, errors="coerce")
        if pd.notnull(ts) and ts.tzinfo is None:
            ts = ts.tz_localize("America/Sao_Paulo")
        return ts
    except Exception as e:
        print(f"Erro ao converter data: {data} -> {e}")
        return pd.NaT

Prev["Data"] = Prev["Data"].apply(converter_data)
Efet["Data"] = Efet["Data"].apply(converter_data)
```

Figura 4: Código Parte 02.
Fonte: Elaborada pelo autor (2025).

O código baseia-se fortemente em bibliotecas consolidadas como pandas para manipulação de dados, datetime para formatação de datas e win32com para interação com a API do Outlook. Inicialmente, os dados são carregados a partir de três planilhas distintas contidas em um arquivo Excel: a planilha “Previsão”, que traz dados projetados; a planilha “Efetivo”, que contém registros que muito provavelmente serão realizados; e a planilha “Base_Original”, que serve como base principal para o processamento dos eventos.

A primeira etapa do código dedica-se à leitura e tratamento das colunas. É feita a limpeza das strings de valores monetários para convertê-las em floats, utilizando uma função personalizada. Além disso, o código garante a consistência das datas, realizando conversões robustas que incluem verificação de timezone, garantindo que os eventos sejam corretamente inseridos no fuso horário de São Paulo, evitando discrepâncias temporais.

Outro ponto relevante do desenvolvimento foi o agrupamento dos dados por fornecedor e por data de vencimento. Essa estratégia de agrupamento simplifica a visualização dos dados e permite a geração de eventos agrupados, sempre respeitando a integridade dos dados originais. O código também inclui uma etapa de formatação dos valores monetários em moeda, apresentando os dados de maneira mais clara e compreensível ao usuário final. Conforme demonstrado na Figura em anexo.

A parte central do código dedica-se à criação e inserção dos eventos no calendário do Outlook. Através da API win32com, o *script* interage diretamente com a aplicação, criando eventos com descrições detalhadas e categorias visuais que facilitam o entendimento dos compromissos. É possível, por exemplo, distinguir entre previsões e efetivos, bem como sinalizar categorias de acordo com o valor financeiro do dia, destacando os dias com valores maiores do que o limite definido do dia.

Um ponto de atenção especial foi o desenvolvimento de uma função para limpar o calendário antes da atualização. Tal medida garante que o calendário reflita apenas os dados mais recentes, evitando sobreposição de eventos antigos ou criação de inúmeros eventos repetidos no mesmo dia, devido ao ritmo de atualização dos dados e preservando a clareza visual das agendas compartilhadas.

Além disso, o código cria eventos resumidos por data, incluindo um evento agregado que soma todos os valores do dia, apresentando um panorama consolidado das movimentações previstas e efetivas. Esta funcionalidade foi implementada para facilitar o monitoramento da responsável de compras, para que ela possa distribuir melhor as datas de faturamento de compra de mercadoria, com o objetivo de não sobrecarregar o setor financeiro

em determinados dias que já possuem valores de pagamento elevado, tais como o 5º dia útil do mês que já conta com a folha de pagamento.

Por fim, vale destacar que o código foi desenvolvido de maneira modular e clara, com funções bem definidas para cada etapa do processo. Isso não só garante a legibilidade do *script*, mas também facilita futuras manutenções ou expansões da ferramenta. A robustez do tratamento de dados e a integração com o Outlook demonstram o potencial do *Python* como ferramenta para automação corporativa, proporcionando ganhos de produtividade e confiabilidade para as equipes de trabalho.

Em relação ao resultado no Outlook, apresentado na Figura 5, ela representa o resultado do processo automatizado desenvolvido em *Python*. Cada evento diário no calendário exibe, de forma resumida, o somatório das contas a pagar divididas entre previstas e efetivas para aquele dia. Além disso, é incluído um evento consolidado, identificado como "Total", que mostra o valor geral a ser pago naquela data específica. Essa organização diária garante uma visualização clara e direta dos compromissos financeiros futuros.

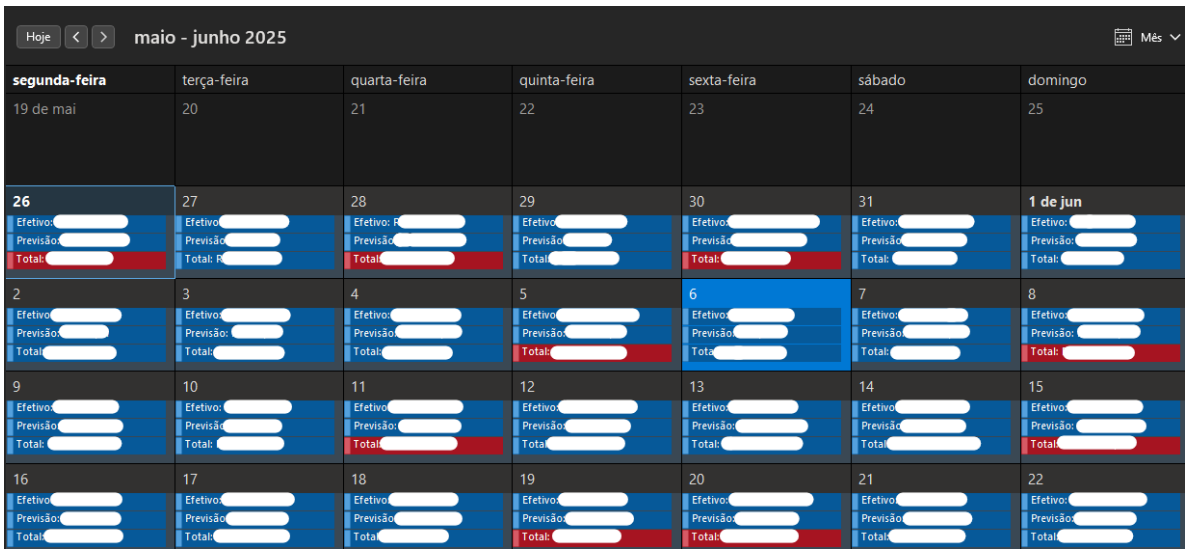

Figura 5: Tela Outlook.
Fonte: Elaborada pelo autor (2025)

A funcionalidade vai além de um simples resumo: ao abrir o evento de "Total" no Outlook, o usuário tem acesso imediato a um detalhamento completo de todas os gastos que compõem aquele valor. Essa tela exibe informações essenciais como o nome do fornecedor, o número da parcela e o valor específico de cada pagamento, permitindo uma análise minuciosa de cada compromisso registrado. Esse detalhamento elimina a necessidade de consultar

planilhas ou outros registros adicionais, centralizando todas as informações em um único lugar. É possível visualizar essa tela na Figura 6.

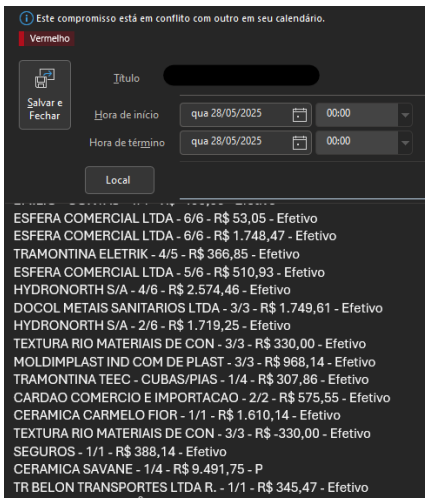

Figura 6: Detalhamento de Contas a Pagar.
Fonte: Elaborada pelo autor (2025)

Para o setor de compras e financeiro, essa solução traz benefícios diretos em termos de praticidade e facilidade. O calendário no Outlook oferece uma visão clara e detalhada dos compromissos financeiros diários, com informações sobre fornecedores, parcelas e valores de pagamento, organizadas de forma automática e centralizada. Essa abordagem facilita o acompanhamento das despesas futuras e permite um planejamento mais preciso, garantindo que as equipes possam focar em decisões estratégicas em vez de tarefas operacionais.

Além disso, a integração visual no Outlook promove uma comunicação mais ágil entre diferentes áreas da empresa, fortalecendo a gestão financeira e aprimorando o controle sobre as contas a pagar. Essa automação elimina o risco de esquecimentos e falhas na organização, consolidando todas as informações essenciais em um único local acessível e confiável.

6. CONCLUSÃO

Com isso, esse estudo de caso teve como principal objetivo desenvolver uma solução prática e eficiente para o gerenciamento de informações e rotinas financeiras, utilizando *Python* como ferramenta central. O desafio consistiu em automatizar processos manuais e repetitivos, integrando-os ao Outlook para criar um calendário de contas a pagar que apoiasse a organização e o controle das finanças de forma mais estratégica.

Durante o desenvolvimento, o *Python* foi utilizado para criar *scripts* capazes de coletar, processar e organizar dados essenciais de contas a pagar. Essa etapa incluiu a leitura e o tratamento de dados provenientes de diferentes fontes, garantindo que as informações estivessem sempre atualizadas e corretas. A automação proporcionada pelo *Python* permitiu a padronização de rotinas que, anteriormente, eram executadas manualmente, reduzindo consideravelmente o risco de erros e aumentando a confiabilidade do processo.

Além disso, a integração com o Outlook foi um ponto central para a criação do calendário automatizado. Por meio do uso de bibliotecas específicas, foi possível automatizar o agendamento de eventos no calendário, permitindo que cada compromisso fosse adicionado com informações detalhadas, como a descrição da conta, o valor, a data de vencimento e outras observações relevantes. Essa integração resultou em um calendário de contas a pagar funcional, dinâmico e de fácil acesso, que se atualiza automaticamente sempre que necessário.

Os resultados obtidos reforçam o potencial do *Python* como uma ferramenta poderosa para automação e gerenciamento de informações. O trabalho não apenas demonstrou a aplicação prática da linguagem em um contexto real, mas também evidenciou a sua capacidade de transformar rotinas financeiras em processos mais ágeis, transparentes e confiáveis.

Outro ponto importante foi a possibilidade de expandir essa solução para outros contextos. A automação realizada, apesar de inicialmente voltada para contas a pagar, mostrou-se como uma base sólida que pode ser adaptada e ampliada para outros tipos de rotinas administrativas e financeiras. Assim, este projeto serviu como um primeiro passo para futuras integrações, onde o *Python* pode ser cada vez mais explorado como ferramenta estratégica para automatizar tarefas, melhorar a gestão de informações e aumentar a produtividade.

Em suma, o trabalho alcançou plenamente os objetivos propostos: demonstrar como o *Python* pode ser aplicado para gerenciar informações de forma automatizada, resolver problemas reais de forma eficiente e preparar o terreno para novas soluções integradas que podem agregar ainda mais valor aos processos organizacionais.

7. REFERÊNCIAS

CARVALHO, Nadson Lúcio de Sousa. **Resiliência Digital: como hard e soft skills estão redefinindo os profissionais de finanças.** ResearchGate, 2024. Disponível em:https://www.researchgate.net/publication/391657659. Acesso em: 17 jun. 2025.

HEINEMANN, Gerrit. **Intelligent retail: digital technologies and consumer behaviour**. Cham: Springer, 2021.

MCKINNEY, Wes. ***Python* for data analysis: data wrangling with pandas, NumPy, and Jupyter**. 3. ed. Sebastopol: O'Reilly Media, 2022.

MEIRELLES, Fernando S**. 35ª Pesquisa Anual do Uso de TI nas Empresas**. São Paulo: FGV, 2023. Disponível em: https://www.researchgate.net/publication/341917736. Acesso em: 17 jun. 2025.

NETO, Jocildo Figueiredo C.; MARQUES, Erico V**. Tomada de decisões gerenciais com analítica de dados**. [Digite o Local da Editora]: Editora Alta Books, 2020. E-book. ISBN 9788550817101. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788550817101/. Acesso em: 09 abr. 2025.

PADILHA, Juliana; SOARES, Juliane A.; ALVES, Nicolli S R.; et al. **Analytics para big data**. [Digite o Local da Editora]: Grupo A, 2022. E-book. ISBN 9786556903477. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786556903477/. Acesso em: 02 abr. 2025.

ROSA, Gabriel C. N. **Sistema de análise de dados e faturamento**. São Paulo: UNESP, 2024. Disponível em: https://repositorio.unesp.br/handle/11449/259814. Acesso em: 17 jun. 2025.

SHARDA, Ramesh; DELEN, Dursun; TURBAN, Efraim. **Business intelligence e análise de dados para gestão do negócio**. [Digite o Local da Editora]: Grupo A, 2019. E-book. ISBN 9788582605202. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788582605202/. Acesso em: 02 abr. 2025.

SWEIGART, Al. **Automate the boring stuff with *Python*: practical programming for total beginners**. 2. ed. San Francisco: No Starch Press, 2019.

ANEXO

```python
# Criar mapeamento para o evento "Total"
def formatar_linha_completa(row):
    valor_formatado = f"R$ {row['Valor']:,.2f}".replace(",", "v").replace(".",
",").replace("v", ".")
    previsao = row["Previsão"]
    if pd.isnull(previsao) or previsao == "":
        previsao = "Efetivo"
    return f"{row['Fornecedor']} - {row['Parcela']} - {valor_formatado} -
{previsao}"

linhas_completas_por_data = Orig.groupby("Venc_fmt", group_keys=False)[
    ["Fornecedor", "Parcela", "Valor", "Previsão"]
].apply(
    lambda df: [formatar_linha_completa(row) for _, row in df.iterrows()]
).to_dict()

# Inicializa o Outlook
outlook = win32.Dispatch("Outlook.Application")
namespace = outlook.GetNamespace("MAPI")
calendar = namespace.GetDefaultFolder(9)  # 9 = olFolderCalendar

# Apagar todos os eventos existentes (cuidado ao rodar em produção)
try:
    items = calendar.Items
    items.IncludeRecurrences = False
    for item in list(items):
        try:
            item.Delete()
        except Exception as del_err:
            print(f"Erro ao deletar item: {del_err}")
except Exception as e:
    print(f"Erro ao acessar itens do calendário: {e}")

# Função para criar evento no Outlook
def criar_evento(data, nome_evento, descricao_extra=None, categoria=None):
    try:
        if isinstance(data, pd.Timestamp):
            data = data.to_pydatetime()
        if data.tzinfo is not None:
            data = data.replace(tzinfo=None)

        appointment = outlook.CreateItem(1)  # olAppointmentItem
        appointment.Subject = nome_evento
        appointment.Start = data
        appointment.AllDayEvent = True
        appointment.ReminderSet = False
        appointment.BusyStatus = 2  # Ocupado

        if categoria:
            appointment.Categories = categoria
        if descricao_extra:
            appointment.Body = descricao_extra

        appointment.Save()

    except Exception as e:
        print(f"Erro ao criar evento '{nome_evento}' em {data}: {e}")

# Criar eventos da aba Previsão (sem descrição)
for _, row in Prev.iterrows():
    if pd.notnull(row["Data"]):
        previsao_valor = limpar_valor(row["Previsão"])
        categoria = "Vermelho" if previsao_valor and previsao_valor > 45000 else
None
        criar_evento(row["Data"], f"Previsão: {row['Previsão']}", categoria)

# Criar eventos da aba Efetivo (sem descrição)
for _, row in Efet.iterrows():
    if pd.notnull(row["Data"]):
        efetivo_valor = limpar_valor(row["Efetivo"])
        categoria = "Vermelho" if efetivo_valor and efetivo_valor > 45000 else None
        criar_evento(row["Data"], f"Efetivo: {row['Efetivo']}", categoria)

# Criar eventos "Total" com descrição detalhada e valor somado no título
for data_fmt, linhas in linhas_completas_por_data.items():
    data = pd.to_datetime(data_fmt, dayfirst=True)

    # Calcular o total dos valores neste dia
    total_valor = Orig.loc[Orig["Venc_fmt"] == data_fmt, "Valor"].sum()
    total_formatado = f"R$ {total_valor:,.2f}".replace(",", "v").replace(".",
",").replace("v", ".")

    descricao = "Resumo do dia:\n" + "\n".join(linhas)
    nome_evento = f"Total: {total_formatado}"

    categoria = "Vermelho" if total_valor > 45000 else None
    criar_evento(data, nome_evento, descricao_extra=descricao, categoria=categoria)

print("✅ Eventos atualizados no calendário do Outlook.")
```