Relatório Final do Projeto – SankeyPy

Aluno: Jorge Duarte – Nº 2201671

Curso: Licenciatura em Engenharia Informática

Unidade Curricular: Projeto Final

Docente: Pedro Pestana

Ano Letivo: 2025

Resumo

O presente relatório descreve o desenvolvimento da **SankeyPy**, uma biblioteca Python dedicada à criação de diagramas de Sankey de forma simples, modular e extensível. Estes diagramas são amplamente utilizados em engenharia, ciência de dados e análise de sistemas para representar fluxos entre entidades, permitindo visualizar de forma intuitiva a magnitude e direção desses fluxos. Embora existam bibliotecas capazes de produzir este tipo de visualização — como Plotly, Matplotlib ou Seaborn — muitas exigem um elevado nível de configuração ou não disponibilizam mecanismos nativos suficientemente flexíveis para Sankey diagrams.

O principal objetivo deste projeto consistiu em criar uma solução que abstraísse essa complexidade, oferecendo uma interface clara e acessível para gerar diagramas de Sankey diretamente a partir de DataFrames do pandas. A biblioteca inclui funcionalidades essenciais como validação automática, agrupamento de fluxos pequenos, personalização visual e exportação de resultados. A arquitetura foi desenvolvida de forma modular, distribuindo responsabilidades por componentes independentes — validação, pré-processamento, estilos, geração do diagrama e criação de exemplos — garantindo assim uma solução simples de manter e facilmente extensível.

A implementação integrando o Plotly Graph Objects como motor gráfico permite gerar visualizações interativas de alta qualidade. A SankeyPy suporta exportação direta para HTML e, mediante a disponibilidade do motor de renderização **Kaleido**, possibilita também a exportação para formatos de imagem como **PNG, SVG e PDF**, ampliando a utilidade da biblioteca em relatórios e aplicações documentais. Testes realizados demonstraram a robustez da solução, mesmo perante dados incompletos ou fluxos residuais, validando o comportamento consistente da biblioteca em diferentes cenários.

Os resultados obtidos permitem concluir que a SankeyPy cumpre os objetivos operacionais, técnicos e académicos definidos, constituindo uma base sólida para desenvolvimentos futuros. Entre as possíveis extensões destacam-se a criação de temas visuais adicionais, suporte a animações, integração com Dash ou Streamlit, visualizações temporais e eventual publicação oficial no PyPI.

A SankeyPy demonstra que é possível conjugar simplicidade, flexibilidade e clareza visual através de uma implementação estruturada e orientada ao utilizador, contribuindo de forma relevante para o domínio da visualização de dados no contexto da Engenharia Informática.

1. Introdução

1.1 Enquadramento

A visualização de dados desempenha um papel fundamental na análise, interpretação e comunicação de informação em praticamente todas as áreas da ciência e engenharia. A crescente disponibilidade de dados — provenientes de sistemas computacionais, sensores, plataformas digitais e processos industriais — exige ferramentas eficazes que permitam transformar grandes volumes de informação num formato compreensível e visualmente apelativo.

Entre os diversos tipos de visualizações existentes, os **diagramas de Sankey** assumem particular importância quando o objetivo é representar **fluxos**, ou seja, movimentos de recursos entre entidades. Exemplos incluem fluxos de energia, transferências financeiras, cadeias logísticas, tráfego de utilizadores em websites, ou a decomposição de processos computacionais. Estes diagramas destacam-se pela sua capacidade de ilustrar proporcionalmente a magnitude dos fluxos, facilitando a comparação e a compreensão das relações entre elementos.

No entanto, apesar da sua utilidade, a criação de diagramas de Sankey em Python continua a ser uma tarefa pouco acessível para utilizadores que não dominam bibliotecas de visualização mais complexas. Ferramentas como *Matplotlib*, *Plotly* e *Seaborn* oferecem suporte parcial a Sankey diagrams, mas frequentemente exigem sintaxe extensa, regras específicas ou customização avançada. Essa barreira técnica motiva a procura por soluções mais simples e orientadas ao utilizador.

É neste contexto que surge o presente projeto: desenvolver uma biblioteca Python — *SankeyPy* — que simplifique a criação deste tipo de visualização, mantendo simultaneamente flexibilidade e potencial de expansão.

1.2 Motivação

A motivação principal para este projeto nasce da constatação de duas lacunas:

1. **Dificuldade em gerar Sankey diagrams com código simples**
   Utilizar diretamente bibliotecas como *Plotly Graph Objects* exige conhecimento prévio da sua estrutura interna, nomeadamente a forma como os índices são mapeados para nós, como os fluxos são representados, e como configurar manualmente os elementos gráficos.

2. **Ausência de uma biblioteca leve, modular e dedicada**
   A maioria das bibliotecas existentes fornece suporte a Sankey diagrams como funcionalidade adicional, não como foco principal. Isso limita:

- a facilidade de utilização,
- a capacidade de validação automática dos dados,
- e a extensibilidade para cenários mais avançados.

Assim, este projeto propõe a criação de uma ferramenta **simples, intuitiva e customizável**, construída especificamente para este tipo de visualização.
Um utilizador deve conseguir gerar um diagrama Sankey completo com poucas linhas de código, a partir de um DataFrame comum, sem necessidade de compreender os detalhes internos da biblioteca gráfica subjacente.

Para além disso, durante o desenvolvimento académico surgiram necessidades concretas que reforçaram esta motivação: a criação de diagramas de Sankey estava prevista em vários trabalhos práticos envolvendo visualização de fluxos, e a ausência de uma ferramenta simplificada representava um obstáculo real tanto para estudantes como para profissionais.

1.3 Objetivos do Projeto

O projeto *SankeyPy* foi desenvolvido com um conjunto claro de objetivos, agrupados em três categorias: **operacionais**, **técnicos** e **académicos**.

Objetivos Operacionais

- Criar uma biblioteca Python capaz de gerar diagramas de Sankey de forma simples.
- Permitir a integração direta com DataFrames do *pandas*, uma ferramenta amplamente utilizada na comunidade científica.
- Disponibilizar funcionalidades prontas a usar, incluindo:
  - validação automática dos dados,
  - agrupamento de fluxos pequenos (threshold),
  - personalização visual opcional,
  - exportação em múltiplos formatos (HTML, PNG,).

Objetivos Técnicos

- Construir uma solução modular, com responsabilidades bem definidas.
- Utilizar *Plotly Graph Objects* como motor gráfico, garantindo interatividade e boa qualidade visual.
- Minimizar dependências externas, mantendo o projeto leve.
- Garantir que o código é simples de manter, extender e documentar.

Objetivos Académicos

- Demonstrar capacidade de:
    - análise de requisitos;
    - desenho e implementação de software modular;
    - documentação técnica e científica;
    - criação de uma biblioteca reutilizável;
    - avaliação crítica das soluções existentes e desenvolvidas.
- Produzir um relatório final claro, rigoroso e alinhado com as exigências da unidade curricular Projeto Final.

1.4 Organização do Documento

Este relatório organiza-se da seguinte forma:

- **Capítulo 2 – Enquadramento Teórico e Estado da Arte:**
  Apresenta os conceitos fundamentais associados aos diagramas de Sankey e discute ferramentas e bibliotecas existentes.
- **Capítulo 3 – Análise de Requisitos:**
  Define os requisitos funcionais, não funcionais e casos de uso da biblioteca.

- **Capítulo 4 – Arquitetura e Desenho da Solução:**
  Descreve a estrutura interna da biblioteca, os seus módulos e as decisões de design.

- **Capítulo 5 – Implementação da Biblioteca:**
  Detalha o código desenvolvido, explicando a lógica principal e as funções mais relevantes.

- **Capítulo 6 – Testes, Demonstração e Avaliação:**
  Apresenta o processo de testes e exemplos de utilização.

- **Capítulo 7 – Discussão, Limitações e Trabalho Futuro:**
  Analisa os resultados, identifica limitações e propõe melhorias futuras.

- **Capítulo 8 – Conclusões:**
  Resume os resultados e reflexões finais do projeto.

2. Enquadramento Teórico e Estado da Arte

2.1 Visualização de Dados

A visualização de dados é uma área fundamental para a análise, interpretação e comunicação de informação em sistemas complexos. Num contexto onde o volume e a diversidade dos dados aumentam de forma exponencial, torna-se essencial recorrer a representações visuais capazes de transformar informação bruta em estruturas cognitivamente acessíveis. Este capítulo apresenta os principais fundamentos teóricos da visualização, com foco nos conceitos relevantes para o desenvolvimento da biblioteca **SankeyPy**, bem como uma revisão do estado da arte das ferramentas existentes para a construção de Sankey diagrams.

2.1.1 Importância da Visualização de Dados

A literatura reconhece amplamente que a visualização não é apenas uma ferramenta de apresentação, mas sobretudo um meio de raciocínio. Cairo (2020) defende que visualizar dados é uma forma de pensar visualmente, permitindo identificar padrões, relações e anomalias que dificilmente seriam detectadas através de tabelas numéricas. A sua função principal é promover clareza, rigor e honestidade comunicacional — atributos essenciais quando se pretende explicar sistemas, fluxos ou processos.

De forma complementar, Munzner (2014) define a visualização como uma transformação estruturada que converte dados num conjunto de elementos gráficos organizados segundo princípios perceptuais. A autora identifica quatro níveis fundamentais no processo de design:

1. **Entender o domínio e as tarefas**
2. **Caracterizar o tipo de dados**
3. **Selecionar a codificação visual e o layout**
4. **Refinar a interação e a apresentação**

A SankeyPy, ao transformar automaticamente DataFrames do Pandas em diagramas de Sankey, segue precisamente esta estrutura: identifica fluxos (dados), cria abstrações (nós e ligações), escolhe uma codificação visual (largura proporcional), e produz uma visualização interativa.

2.2.2.Princípios Fundamentais da Visualização

Edwards Tufte (1983) introduziu princípios que permanecem centrais na visualização moderna, especialmente:

- **Maximização da densidade informativa** — mostrar o máximo de informação com o mínimo de elementos gráficos redundantes;
- **Data-ink ratio** — privilegiar elementos que representam efetivamente dados;
- **Evitar ruído visual** — reduzir artefactos, decorações ou elementos que não acrescentem significado.

Estes princípios justificam diretamente funcionalidades como a **remoção de fluxos inválidos** e o **agrupamento de fluxos pequenos ("Outros")**, implementados na SankeyPy para reduzir elementos insignificantes que comprometeriam a clareza do diagrama.

Schwabish (2021) reforça esta perspetiva ao mostrar que escolhas visuais inadequadas — cores excessivas, ligações demasiado finas ou rótulos inconsistentes — influenciam negativamente a perceção do utilizador. Assim, funcionalidades como **cores personalizadas**, **orientação vertical** e **hover informativo** adquirem grande relevância para garantir que a visualização é compreendida de forma intuitiva.

2.2 Diagramas de Sankey

Os diagramas de Sankey são gráficos de fluxo onde:

- os nós representam entidades ou estados,
- as ligações representam fluxos entre essas entidades,
- e **a espessura das ligações é proporcional à quantidade do fluxo representado**.

Criados originalmente por Matthew Sankey (1898) para representar perdas energéticas em sistemas térmicos, rapidamente foram adotados em diversas áreas:

Aplicações modernas de Sankey diagrams

- **Energia e termodinâmica:** perdas, conversões, eficiências.
- **Economia:** transferências financeiras, distribuição de receitas ou impostos.
- **Ecologia:** fluxos de massa, energia em cadeias alimentares.
- **Engenharia de software:** logs de execução, tráfego interno de sistemas.
- **Web analytics:** caminhos de utilizadores em websites.
- **Ciência de dados:** decomposição de conjuntos, partições ou clusters.

Os diagramas oferecem:

- leitura intuitiva,
- clareza visual,
- foco imediato nas magnitudes,
- comparação entre fluxos concorrentes.

Estas características transformam-nos numa ferramenta essencial para “storytelling” com dados — permitindo que decisões e fenómenos complexos sejam explicados de forma acessível.

2.3 Características Fundamentais

1. Proporcionalidade Visual

A característica mais importante: **a largura da ligação representa o valor do fluxo**.
Isto distingue Sankey diagrams de grafos normais.

2. Direcionalidade

Os fluxos têm sempre direção (origem → destino).
Permite entender sequências, níveis, perdas e bifurcações.

3. Possibilidade de Agrupamento

Numa visualização complexa, fluxos residuais são agrupados ("Outros"), preservando clareza.

4. Integração com Dados Tabulares

Estruturas como *pandas.DataFrame* encaixam naturalmente em Sankey diagrams:
cada linha representa um fluxo.

2.4 Estado da Arte (Matplotlib, Seaborn, Plotly, etc.)

Nesta secção analisam-se as principais ferramentas existentes para criar Sankey diagrams em Python ou no ecossistema científico, destacando vantagens e limitações.

2.4.1. Matplotlib

Embora seja uma das bibliotecas mais importantes e robustas de visualização em Python, a Matplotlib **não possui suporte nativo forte** para Sankey diagrams. A funcionalidade existente:

```
from matplotlib.sankey import Sankey
```

é:

- pouco flexível,
- de difícil personalização,
- com documentação extremamente limitada,

- e inadequada para fluxos complexos.

Além disso:

- não suporta interatividade,
- não permite hover com valores,
- e carece de exportações modernas.

Conclusão: Matplotlib **não é indicada** para Sankey moderno.

2.4.2. Seaborn

O Seaborn é construído sobre Matplotlib e é excelente para estatística, mas:

- não possui qualquer suporte para Sankey diagrams,
- não foi projetado para grafos ou fluxos,
- não oferece mecanismos de personalização adequados.

Logo, não constitui uma alternativa.

2.4.3. Plotly Graph Objects

O Plotly fornece suporte nativo a Sankey via:

```
go.Sankey(...)
```

Vantagens:

- interatividade avançada,
- hover com informações,
- visualização moderna,
- exportação para HTML,
- exportação para PNG via *Kaleido*,
- integração natural com *pandas*.

Limitações:

- exige construção manual de listas de índices,
- o utilizador deve mapear nós manualmente,
- necessita de configuração detalhada (labels, core, links),
- API extensa e por vezes complexa.
  Ainda assim:

  É a melhor solução existente e serve como base para a SankeyPy.

2.4.4. Plotly Express

Possui uma API simplificada, mas **não suporta Sankey diagrams**.
Depende do módulo Graph Objects.

2.4.5. Outras ferramentas relevantes

NetworkX

Útil para grafos, mas não para Sankey.
Sem suporte nativo à largura proporcional dos fluxos.

D3.js

Bilbioteca JavaScript poderosa, mas:

- difícil de integrar com Python,
- curva de aprendizagem elevada.

PowerBI, Tableau

Ferramentas comerciais com suporte a Sankey por extensões, mas fora do âmbito da programação Python e da liberdade académica.

2.5 Justificação da Criação da SankeyPy

Após a análise do estado da arte, a criação da *SankeyPy* justifica-se por cinco lacunas claras:

1. Simplicidade

Nenhuma ferramenta permite gerar Sankey diagrams a partir de um DataFrame com uma única função simples.

2. Validação Automática

Problemas comuns como:

- colunas incorretas,
- valores nulos,
- fluxos negativos,
- ausência de nós,
  não são tratados pelas bibliotecas tradicionais.

3. Agrupamento Inteligente de Fluxos Pequenos

Fluxos insignificantes poluem visualizações complexas.
A SankeyPy inclui:

```
threshold → agrupar fluxos pequenos em “Outros”
```

4. Arquitetura Modular

Importante para:

- manutenção,
- testes,
- possível distribuição futura no PyPI.

5. Extensibilidade

A SankeyPy foi desenhada de raiz para:

- aceitar novos tipos de visualização no futuro,
- integrar temas visuais,
- suportar interfaces web (Dash, Streamlit).

2.6. Conclusão do Estado da Arte

A análise das bibliotecas existentes revela que, apesar da existência de ferramentas capazes de gerar Sankey diagrams, nenhuma delas apresenta um enfoque claro na usabilidade, simplicidade e modularidade.

O *Plotly* fornece a base gráfica ideal, mas exige demasiado conhecimento prévio. Por outro lado, ferramentas como Matplotlib ou NetworkX não oferecem o suporte adequado.

A *SankeyPy* surge, assim, como uma proposta relevante e oportunamente posicionada: uma biblioteca especializada, minimalista, mas poderosa, capaz de produzir diagramas de Sankey interativos de forma acessível e extensível

3.Análise de Requisitos

A análise de requisitos constitui um dos pilares de qualquer projeto de software.

É nesta fase que se identificam, descrevem e organizam as funcionalidades que o sistema deve oferecer, bem como as suas restrições, características de desempenho e necessidades de qualidade.Para a biblioteca *SankeyPy*, a definição rigorosa e antecipada dos requisitos permitiu estabelecer uma visão clara do que teria de ser implementado, garantindo uma base sólida para o desenho da arquitetura e o desenvolvimento consequente.

Os requisitos identificados foram estruturados em **requisitos funcionais**, **requisitos não funcionais**, **casos de uso**, **perfis de utilizador** e **fluxos de dados**.

3.1 Requisitos Funcionais (RF)

Os requisitos funcionais descrevem **o que o sistema deve fazer**, ou seja, as funcionalidades observáveis pelos utilizadores.

RF1 — Leitura direta de dados a partir de Pandas DataFrames

O utilizador deve poder fornecer os dados sob a forma de um pandas.DataFrame, contendo as colunas necessárias para a criação de fluxos: source, target, value.

RF2 — Validação automática das colunas obrigatórias

Se faltar alguma das colunas necessárias, a biblioteca deve emitir um erro claro e evitar a criação de um gráfico inválido.

RF3 — Remoção de valores nulos, negativos ou inconsistentes

Linhas com valores invá-lidos devem ser excluídas.

RF4 — Agrupamento opcional de fluxos pequenos (threshold)

A biblioteca deve permitir agrupar fluxos cujo valor seja inferior a uma proporção definida do total (por exemplo, fluxos < 1%), agrupando-os no nó “Outros”.

RF5 — Geração automática do diagrama de Sankey

A biblioteca deve gerar um Sankey diagram completo através de uma única função pública: **plot(df)**

RF6 — Personalização visual

O utilizador deve poder personalizar:

- cores dos nós,
- orientação (horizontal ou vertical),
- tamanho da fonte,
- título do gráfico.

RF7 — Exportação para ficheiros HTML e imagem

A biblioteca deve suportar:

- HTML interativo (. html)
- imagens (. png, .svg, .jpeg, .pdf)
  através da integração com Kaleido.

RF8 — Mensagens de erro claras e informativas

Quando forem detetados problemas, a biblioteca deve emitir erros compreensíveis, facilitando o diagnóstico.

RF9 — Disponibilização de um DataFrame de exemplo

O módulo utils.py deve fornecer uma função gerar_df_exemplo() para facilitar testes e demonstrações.

3.2 Requisitos Não Funcionais

Os requisitos não funcionais descrevem características gerais do sistema, tais como desempenho, qualidade, usabilidade e extensibilidade.

RNF1 — Usabilidade

O sistema deve ser simples de usar.
Idealmente, um gráfico Sankey deve ser criado com **no máximo três linhas de código**, sem conhecimento profundo de Plotly.

RNF2 — Eficiência

O tempo de processamento deve ser suficientemente rápido para lidar com milhares de linhas, desde que o motor gráfico o permita.

RNF3 — Robustez

A biblioteca deve lidar corretamente com datasets incompletos, fluxos residuais, valores nulos e outras irregularidades.

RNF4 — Portabilidade

A biblioteca deve funcionar em qualquer plataforma que suporte Python 3.x.

RNF5 — Extensibilidade

O código deve ser modular, permitindo novas funcionalidades sem alterar a estrutura geral.

RNF6 — Documentação interna

Todas as funções devem incluir docstrings claras.

RNF7 — Ausência de dependências desnecessárias

Para manter leveza e compatibilidade, apenas as bibliotecas essenciais devem ser utilizadas:
pandas, plotly e kaleido.

3.3 Casos de Uso

UC1 — Criar um Sankey Diagram básico

Ator: Analista / utilizador comum
Objetivo: Gerar um gráfico simples a partir de um DataFrame
Fluxo:

1. Utilizador carrega um DataFrame com colunas source, target, value
2. Executa a função plot(df)
3. A biblioteca valida dados internalmente
4. É apresentado o gráfico

UC2 — Criar um Sankey Diagram com personalização

Objetivo: Alterar cores, orientação e título
Fluxo:

1. Utilizador prepara DataFrame
2. Chama:
   plot(df, colors=[...], orientation='v', title="Fluxo de Energia")
3. Gráfico é produzido com as opções personalizadas

UC3 — Exportar o gráfico

Objetivo: Guardar um ficheiro HTML ou PNG
Fluxo:

1. Utilizador chama:

   ```
   plot(df, save_as="grafico.html")
   ```

2. A biblioteca exporta diretamente o ficheiro

UC4 — Agrupar fluxos pequenos

Objetivo: Aumentar legibilidade do gráfico
Fluxo:

1. Utilizador define threshold=0.05
2. Função agrupa valores < 5% do total
3. Gráfico exibe “Outros” como nó agrupado

3.4 Perfis de Utilizador

Perfil 1 — Estudante / Investigador

- Conhecimentos básicos de Python e Pandas
- Precisa de uma ferramenta rápida e simples para análises científicas

Perfil 2 — Cientista de Dados

- Manipula grandes volumes de dados
- Procura rapidez e personalização visual

Perfil 3 — Engenheiro de Software

- Preocupa-se com modularidade e extensibilidade
- Eventualmente quer integrar SankeyPy noutros sistemas

Perfil 4 — Utilizador Casual

- Apenas deseja gerar um gráfico simples
- Precisa de bibliotecas fáceis de usar

3.5 Fluxos de Dados

O fluxo de dados dentro da biblioteca pode ser descrito da seguinte forma:

1. **Input:** DataFrame fornecido pelo utilizador
2. **Validação:** Verificação de colunas, remoção de nulos e valores inválidos
3. **Pré-processamento:** Agrupamento de fluxos pequenos
4. **Construção de Nós:** Mapeamento de labels para índices

5. **Construção de Links:** Conversão de colunas para arrays reconhecidos pelo Plotly
6. **Renderização:** Criação do objeto go.Sankey
7. **Configuração de Layout:** título, fonte, orientação
8. **Output:**
    - Visualização no navegador
    - Exportação para ficheiro

4. Arquitetura e Desenho da Solução

A arquitetura da biblioteca *SankeyPy* foi desenhada com base em princípios fundamentais da engenharia de software: modularidade, separação de responsabilidades, simplicidade, facilidade de manutenção e extensibilidade.

O objetivo é criar uma biblioteca que não só resolva o problema imediato (geração de Sankey diagrams), mas que também permita evoluções futuras, como integração com dashboards interativos, novos tipos de visualização ou expansão das capacidades gráficas.

4.1 Visão Geral da Arquitetura

sankeypy/

├── plot.py ← Núcleo da biblioteca (função principal plot())

├── parser.py ← Validação e préprocessamento dos dados

├── style.py ← Paletas de cores e estilos predefinidos

├── utils.py ← Geração de DataFrames de exemplo

├── __init__.py ← Exporta a API pública (from .plot import plot)

└── examples/

├── example_modular.py ← Script de demonstração modular

└── generate_sankey_examples.py ← Testes automatizados e geração de figuras

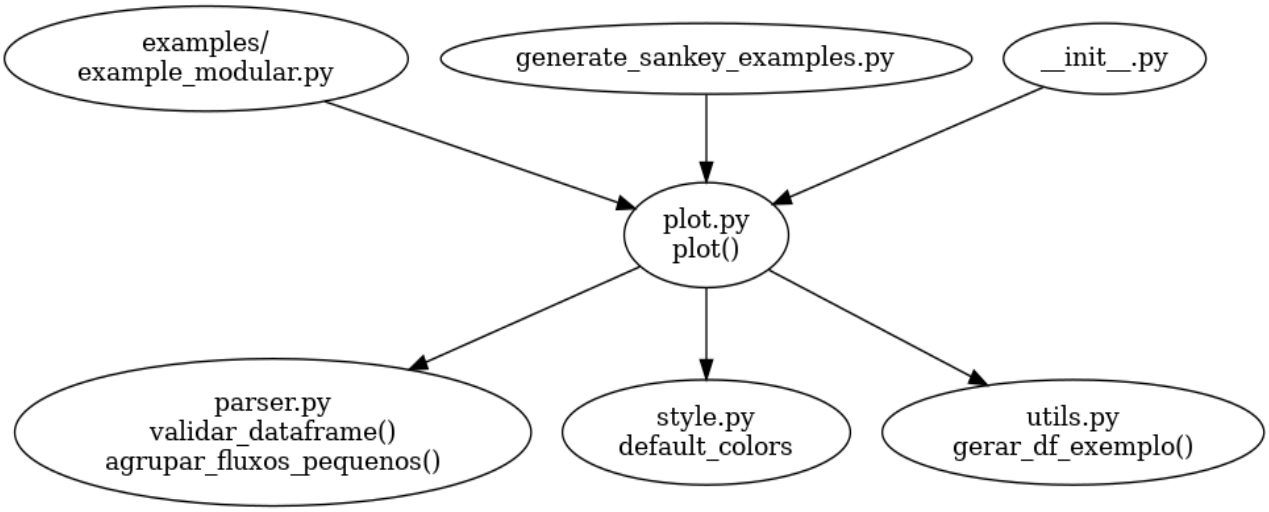
Cada módulo desempenha um papel específico:

- plot.py → núcleo da biblioteca, contém a função principal plot().
- parser.py → validação e pré-processamento de dados.
- utils.py → geração de dados de exemplo.
- style.py → definição de estilos predefinidos (cores).
- __init__.py → expõe a API pública do pacote.

Toda a arquitetura foi organizada no paradigma **processamento → transformação → visualização**, que garante clareza no fluxo interno dos dados.

Para além dos módulos internos que constituem a biblioteca SankeyPy, o projeto inclui um ficheiro adicional — example_modular.py — concebido como um módulo externo de demonstração e validação. Embora este ficheiro não faça parte do núcleo da biblioteca, ele desempenha um papel relevante no processo de teste e na demonstração prática de utilização.

4.2 Diagrama de Módulos



4.3 Justificação da Arquitetura

A estrutura modular permite:

Separação de responsabilidades

- Qualquer problema na validação é tratado no parser.py, nunca no plot.py.
- O estilo visual pode ser alterado sem interferir no código funcional.
- A função principal mantém-se limpa e com poucos argumentos visíveis.

Extensibilidade

Por exemplo:

- Para adicionar um tema escuro no futuro, basta alterar style.py.
- Para criar gráficos hierárquicos (Sankey multi-nível), basta expandir plot.py.
- Para integrar Dash/Streamlit, só o módulo plot.py precisará de ajustes.

Reutilização

Módulos como parser.py ou style.py podem ser usados por outras bibliotecas externas.

Testabilidade

Funções modulares permitem:

- testes unitários independentes,
- facilidade em detetar onde ocorre um erro,
- simulação de comportamentos específicos.

4.4 Descrição Detalhada dos Módulos

4.4.1. Módulo parser.py

O módulo parser.py é responsável por garantir que os dados fornecidos são válidos.
Inclui duas funções fundamentais:

a) validar_dataframe(df, source, target, value)

Função responsável por:

- verificar se as colunas existem,
- remover linhas nulas,
- excluir valores ≤ 0,
- lançar erros claros.

Código base:

```
for col in (source, target, value):
    if col not in df.columns:
        raise ValueError(...)
df = df.dropna(...)
df = df[df[value] > 0]
```

Esta funcão assegura que nenhum Sankey diagram é construído com dados inválidos, evitando erros posteriores no módulo de visualização.

b) **agrupar_fluxos_pequenos(df, source, target, value, threshold)**

Quando existem fluxos muito pequenos, o gráfico torna-se ilegível.
Esta função identifica fluxos cujo valor é menor que:

valor_fluxo < threshold × total_fluxos

Esses fluxos são substituídos por uma entrada com o nó "Outros".

Vantagens:

- melhora a legibilidade,
- evita poluição visual,
- mantém a proporção total dos dados.

4.4.2. Módulo style . py

Contém um conjunto de cores padrão em formato hexadecimal:

```
default_colors = [
    "#FF9999", "#99FF99", "#9999FF",
    "#FF66CC", "#66FFFF", "#AAAAAA", "#CCCCCC"
]
```

Objetivos:

- fornecer estilos consistentes,
- simplificar o uso por parte do utilizador,
- permitir futuras paletas temáticas.

4.4.3. Módulo utils.py

Disponibiliza a função gerar_df_exemplo(), que fornece um dataset pequeno:

source	target	value
A	B	10
A	C	5
B	D	15
C	D	5
E	D	0.5

Usado para:

- testes rápidos,
- verificação do funcionamento da biblioteca,
- demonstração na documentação.

4.4.4. Módulo plot.py (núcleo do sistema)

O módulo central da biblioteca.
Contém a função:

```
plot(df, source, target, value, title, threshold, colors,
orientation, font_size, save_as)
```

Responsável por:

- validar o DataFrame,
- aplicar agrupamentos,

- construir estruturas visuais do Plotly,
- definir nós e ligações,
- criar o Sankey diagram,
- exportar ou visualizar o gráfico.

O processo interno pode ser representado assim:

df -> validar_dataframe()

-> agrupar_fluxos_pequenos()

-> extrair nós únicos

-> mapear nodes para índices

-> definir cores

-> construir go.Sankey()

-> criar go.Figure()

-> exportar/mostrar

O fluxo é claramente sequencial e modular.

4.5 Decisões Arquiteturais Importantes

A) Utilização do Plotly Graph Objects

Escolha feita devido à:

- maturidade da biblioteca,
- documentação extensa,
- interatividade superior,
- capacidade de exportação.

B) Organização Modular

Evita monólitos difíceis de manter.
Permite que cada parte seja substituída no futuro sem afetar o resto.

C) Validação antes da visualização

Garante robustez e evita gráficos com dados inválidos.

D) Threshold configurável

Decisão tomada para garantir legibilidade e permitir personalização.

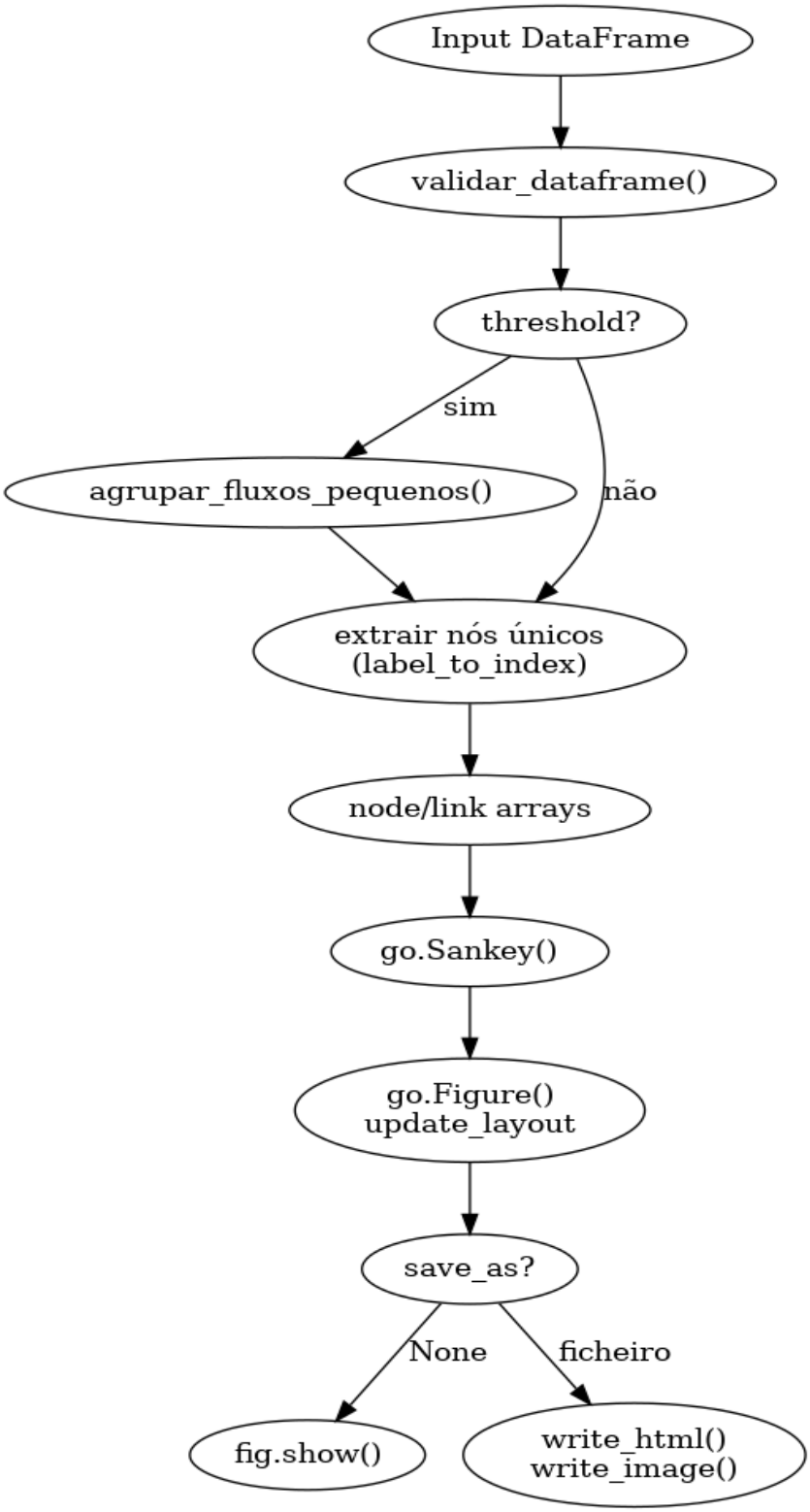
E) API amigável

Toda a biblioteca foi desenhada para:

```
plot(df)
```

sem complexidade adicional para o utilizador.

4.6 Pipeline Interno da Biblioteca



5. Implementação da Biblioteca

Este capítulo descreve detalhadamente a implementação da biblioteca **SankeyPy**, apresentando a arquitetura interna, os módulos que a compõem e as principais decisões técnicas tomadas ao longo do desenvolvimento. A abordagem seguida articula-se diretamente com os princípios teóricos apresentados no Capítulo 2 — nomeadamente os relacionados com clareza, legibilidade e redução de ruído visual — e prepara o enquadramento para a demonstração prática e avaliação formal no Capítulo 6.

A biblioteca foi concebida de forma modular, distribuindo responsabilidades por componentes independentes. Esta organização permite que cada parte seja desenvolvida, testada e evoluída de forma isolada, garantindo simultaneamente a coerência interna e a extensibilidade futura..

5.1 Implementação do plot.py

O módulo plot.py constitui o núcleo funcional da biblioteca SankeyPy. É aqui que ocorre a transformação de um DataFrame Pandas num diagrama Sankey interativo, seguindo um pipeline bem definido: validação → pré-processamento → mapeamento → construção visual → renderização.

5.1.1.Importações

As primeiras linhas importam bibliotecas externas e módulos internos:

```
import plotly.graph_objects as go
import pandas as pd
from .parser import validar_dataframe, agrupar_fluxos_pequenos
```

Explicação:

- **plotly.graph_objects** fornece o objeto go.Sankey, responsável pela renderização visual do diagrama.
- **pandas** é necessário para manipulação segura e eficiente do DataFrame ao longo do pipeline.
- As funções **validar_dataframe()** e **agrupar_fluxos_pequenos()** são importadas do módulo interno parser.py, garantindo separação clara entre:
    - pré-processamento dos dados, e
    - construção da visualização.

Ao contrário de muitas bibliotecas, a SankeyPy não importa diretamente o módulo style.py por defeito.
Em vez disso, a paleta de cores pode ser passada pelo utilizador através do argumento colors.
Isto dá flexibilidade e evita dependências implícitas no desenho do gráfico.

5.1.2.Assinatura da função principal

```
def plot(df, source='source', target='target', value='value',
        title=None, threshold=None,
        colors=None, orientation='h', font_size=10,
        save_as=None):
```

Esta assinatura foi pensada para:

- ser simples para iniciantes
- ser flexível para utilizadores avançados
- permitir personalização sem sobrecarregar a função

Ou seja: O objetivo foi criar uma API simples e intuitiva, coerente com a filosofia “o utilizador fornece o DataFrame, a biblioteca trata do resto”.

Parâmetros importantes:

- source, target, value: permitem adaptar a biblioteca a qualquer dataset, desde que tenha essas 3 colunas.
- threshold: ativa o agrupamento de fluxos pequenos,permite reduzir ruído visual,alinhando com os princípios de Tufte.
- colors/title : lista opcional de cores personalizadas/adiciona titulo ao gráfico-aumentando assim flexibilidade e personalização
- orientation: `'h'` para horizontal (padrão), `'v'` para vertical.
- save_as: ativa exportação automática.

5.1.3 Validação do DataFrame

```
df = validar_dataframe(df, source, target, value)
df = agrupar_fluxos_pequenos(df, source, target, value, threshold)
```

Esta etapa garante:

Problema	Ação
Falta de colunas	erro informativo
Valores nulos	remoção
Valores ≤ 0	remoção
Muitos fluxos pequenos	agrupamento "Outros"

Num diagrama Sankey, valores negativos não têm significado interpretável e entradas incompletas comprometem a estrutura do grafo.

Sem esta etapa, o utilizador teria gráficos incorretos ou erros difíceis de interpretar

5.1.4 Construção dos nós do gráfico

```
all_nodes = list(set(df[source]) | set(df[target]))
label_to_index = {label: i for i, label in enumerate(all_nodes)}
```

Explicação:

A união dos conjuntos source e target garante uma recolha completa e elimina duplicados. O mapeamento label_to_index converte rótulos textuais em índices inteiros, o formato esperado pelo Plotly.

Impacto na visualização:
Os nós definidos aqui determinam toda a estrutura do Sankey, influenciando o layout, as ligações e o hover.

5.1.5. Definição de cores dos nós

```
if colors:
    node_colors = colors[:len(all_nodes)]
    while len(node_colors) < len(all_nodes):
        node_colors.append('gray')
else:
    node_colors = default_colors[:len(all_nodes)]
```

Explicação:

- Se o utilizador passar colors, essa lista é usada e, se for mais curta que o número de nós, preenche-se com 'gray'.
- Se colors não for fornecido, node_colors fica None e o Plotly usará o comportamento por defeito para cores de nós.
- Esta solução mantém a biblioteca leve (sem dependência automática de um módulo de estilos) e dá controlo total ao utilizador sobre a paleta quando necessário.

5.1.6.Criação da estrutura Sankey

A criação da estrutura Sankey corresponde ao momento em que os dados previamente validados são transformados numa representação visual através do objeto `go.Sankey`. Nesta etapa, todos os elementos essenciais — nós, ligações, cores e orientação — são reunidos num único bloco de configuração.

A parte central da implementação:

```
sankey_data = go.Sankey(
    orientation=orientation,
    node=dict(
        label=all_nodes,
        pad=15,
        thickness=20,
        color=node_colors
    ),
    link=dict(
  source=df[source].map(label_to_index),
target=df[target].map(label_to_index),

        value=df[value],
        customdata=df[[source, target,
value]].values,
         hovertemplate='source:
%{customdata[0]}<br>target:
%{customdata[1]}<br>value:
%{customdata[2]}<extra></extra>'
    )
)
```

Explicação:

- **orientation**: define o layout horizontal/vertical.
- **node**: configura os nós do gráfico (nomes,espessuras,espaçamentos).
- **pad**: espaço entre nós.
- **thickness**: espessura visual dos nós.
- **color**: lista de cores.
- **link/source/target**: representam as ligações entre nós.
- **customdata + hovertemplate**: permite controlar totalmente a informação exibida quando o utilizador passa o cursor sobre cada ligação, mostrando os valores reais de *source*, *target* e *value*. Isto assegura que o hover é claro e informativo, em vez de apresentar apenas índices internos..

5.1.7.Criação da figura final

Depois de definida a estrutura do diagrama Sankey, a última etapa consiste na criação da figura Plotly e na aplicação das opções finais de apresentação. Esta fase é responsável por transformar os dados configurados no objeto go.Sankey numa visualização completa, pronta a ser exibida ou exportada.

O código essencial é:

```
fig = go.Figure(data=[sankey_data])
fig.update_layout(
    title_text=title or "Sankey Diagram",
    font_size=font_size,
    hoverlabel=dict(namelength=-1)
)
```

Explicação:

A função go.Figure recebe o objeto sankey_data e constrói a figura final, permitindo ajustar elementos como o título, o tamanho da fonte e o comportamento do *hover*. O parâmetro hoverlabel=dict(namelength=-1) garante que os nomes exibidos no hover não são truncados, tornando a interação mais informativa.

A função pode operar de duas formas:

- **Visualização interativa**, quando save_as é None, utilizando fig.show().
- **Exportação automática**, quando o utilizador fornece um ficheiro com extensão suportada, recorrendo a:

```
fig.write_html(...)
fig.write_image(...)
```

Esta abordagem torna a biblioteca flexível, permitindo tanto a inspeção visual imediata como a geração de ficheiros para relatórios e documentação.

5.2 Implementação do parser.py

O módulo parser.py desempenha um papel fundamental na biblioteca *SankeyPy*, pois garante que os dados fornecidos pelo utilizador possuem a estrutura e qualidade necessárias antes de serem utilizados pelo módulo plot.py. A separação explícita entre **validação**, **limpeza** e **pré-processamento** permite que o código seja mais robusto, modular e fácil de manter, evitando que a função principal plot() acumule responsabilidades excessivas.

O módulo é composto por duas funções principais:

- validar_dataframe(df, source, target, value)

- agrupar_fluxos_pequenos(df, source, target, value, threshold)

Cada uma delas contribui para uma etapa distinta do pipeline interno de execução, assumindo uma função clara e independent

5.2.1 Função validar_dataframe()

Esta função tem como objetivo garantir que o DataFrame fornecido como entrada contém todas as colunas necessárias (source, target, value) e que os dados não apresentam problemas estruturais.

Objetivos principais:

- Verificar se as colunas obrigatórias existem.
- Remover linhas com valores *nulos* (NaN) nessas colunas.
- Remover linhas com valores *não positivos* (≤ 0), uma vez que fluxos negativos ou nulos não fazem sentido num Sankey diagram.
- Garantir que, após a limpeza, os dados continuam válidos; caso contrário, é lançado um erro claro e informativo.

Código da função:

```
def validar_dataframe(df, source, target, value):
      for col in (source, target, value):
      if col not in df.columns:
            raise ValueError(f"Coluna '{col}' não existe no DataFrame.")
      df = df.dropna(subset=[source, target, value])
      df = df[df[value] > 0]
      if df.empty:
      raise ValueError("Não há dados válidos após remover fluxos
nulos/negativos.")
      return df
```

Explicação detalhada do funcionamento:

1. **Verificação das colunas**
   O código percorre as três colunas essenciais e verifica se existem no DataFrame.
   Caso falte alguma, é lançada uma exceção *ValueError*, informando exatamente qual a coluna em falta.

2. **Remoção de valores nulos**
   A função df.dropna() elimina todas as linhas que contenham dados em falta.
   Isto evita erros durante a criação dos fluxos.

3. **Remoção de fluxos inválidos**
   Linhas cujo valor (value) seja menor ou igual a zero são eliminadas:

   - Fluxos negativos não são permitidos.
   - Fluxos zero não contribuem para o gráfico.

4. **Verificação final de integridade**
   Caso todas as linhas tenham sido removidas, é lançado um erro indicando que o dataset não é utilizável.

Esta etapa garante que o módulo plot.py recebe dados **limpos, seguros e consistentes**, prevenindo falhas difíceis de interpretar no processo de visualização.

5.2.2.Função agrupar_fluxos_pequenos()

Esta função implementa a lógica de **agrupamento de fluxos residuais** — fluxos que, por serem muito pequenos, poderiam tornar o gráfico visualmente confuso.

O parâmetro threshold representa uma proporção (por exemplo, 0.05 = 5%) que permite classificar fluxos insignificantes.

Código da função:

```
def agrupar_fluxos_pequenos(df, source, target, value,
threshold):
    if threshold is not None:
        total = df[value].sum()
        mask = df[value] < (threshold * total)
        if mask.any():
            agrupados = pd.DataFrame({
                source: ['Outros'] * mask.sum(),
                target: df.loc[mask, target],
                value: df.loc[mask, value]
            })
            df = pd.concat([df.loc[~mask], agrupados],
ignore_index=True)
    return df
```

Explicação:

1. **Verificação da existência de threshold**
   Se o utilizador não especificar o parâmetro, a função não altera o dataset.
2. **Cálculo do total de fluxos**
   A soma dos valores (value) define o total sobre o qual os fluxos serão comparados.
3. **Identificação dos fluxos pequenos**
   É criada uma máscara para identificar linhas cujo valor seja inferior ao threshold, por exemplo: *value < 0.05 * total*
4. ***Agrupamento dos fluxos insignificantes*** *Estas entradas são substituídas num novo DataFrame, onde o nó de origem passa a ser "Outros".*
5. ***Concatenar dados filtrados com dados agrupados*** *Os fluxos que não foram considerados pequenos são mantidos intactos.*
6. ***Retorno do DataFrame modificado***
   O resultado final contém:

   1-os fluxos principais tal como estavam,

   2-uma categoria adicional "Outros" agregando valores residuais.

Importância desta função para a legibilidade do gráfico

A prática de agrupar fluxos é comum em visualizações profissionais, e evita problemas como:

- excesso de nós finos,
- linhas quase invisíveis,
- gráficos confusos e difíceis de interpretar.

A presença desta função na arquitetura reflete uma preocupação com a **qualidade visual** e **claridade** do Sankey Diagram resultante.

5.3 Implementação do utils.py

O módulo `utils.py` tem como objetivo fornecer funções auxiliares que são úteis durante o desenvolvimento, teste e demonstração da biblioteca *SankeyPy*. Embora não faça parte do núcleo funcional do pipeline interno da biblioteca, este módulo desempenha um papel relevante ao disponibilizar dados consistentes e de fácil utilização para verificar se os restantes componentes funcionam corretamente.

A presença deste módulo simplifica o processo de experimentação por parte do utilizador, permitindo gerar rapidamente um dataset adequado para a construção de Sankey Diagrams sem necessidade de preparar dados manualmente. Além disso, facilita a criação de exemplos na documentação e foi essencial para a fase de testes da biblioteca.

5.3.1 Função gerar_df_exemplo()

Código da função:

```
def gerar_df_exemplo():
    return pd.DataFrame({
        'source': ['A', 'A', 'B', 'C', 'E'],
        'target': ['B', 'C', 'D', 'D', 'D'],
        'value':  [10, 5, 15, 5, 0.5]
    })
```

Explicação:

A função retorna um DataFrame de cinco entradas, organizado em três colunas essenciais:

- **source** — nó de origem do fluxo
- **target** — nó de destino
- **value** — magnitude do fluxo

Os valores foram escolhidos de forma a:

1. **Incluir múltiplas transições a partir de um mesmo nó**
   Ex.: `A → B` e `A → C`
   Isto permite testar se o algoritmo identifica corretamente todos os nós únicos e constrói o mapa de índices necessário para o gráfico.

2. **Gerar um caso típico de convergência**
   Vários nós enviam fluxos para o mesmo destino:
   B → D, C → D, E → D
   Isto reproduz um padrão comum em sistemas reais.

3. **Criar um fluxo pequeno para testar threshold**
   O valor 0.5 da linha E → D é propositalmente muito inferior aos restantes valores, permitindo observar:

   - como funciona o agrupamento de fluxos pequenos,
   - a criação automática do nó "Outros",
   - e o impacto visual antes/depois do threshold.

4. **Permitir testes rápidos**
   O pequeno tamanho do DataFrame torna-o ideal para execução em:

   - notebooks,
   - documentação,
   - scripts de demonstração (como example_modular.py).

5.4 Implementação do style.py

O módulo style.py é responsável por centralizar e disponibilizar as paletas de cores padrão utilizadas na biblioteca *SankeyPy*. Embora seja um módulo relativamente simples em termos de implementação, ele desempenha um papel importante na organização da biblioteca ao permitir separar a lógica funcional da lógica de apresentação visual.

5.4.1 Conteúdo do Módulo style.py

O módulo `style.py` tem um papel deliberadamente simples: centralizar definições de paletas de cores que podem ser reutilizadas pelo utilizador para personalizar a aparência dos diagramas. Na versão atual do projecto, o módulo disponibiliza uma paleta padrão, `default_colors`, definida como uma lista de strings em formato hexadecimal:

Código do módulo:

```
default_colors = [
    "#FF9999", "#99FF99", "#9999FF",
    "#FF66CC", "#66FFFF", "#AAAAAA", "#CCCCCC"
]
```

- A lista default_colors fornece uma sequência de cores com bom contraste e legibilidade para a maioria dos usos documentais e de apresentação.
- O formato em hexadecimais permite compatibilidade directa com o Plotly e outros motores gráficos.
- O módulo foi mantido intencionalmente minimalista para não impor um estilo único: a escolha de centralizar apenas uma paleta facilita a extensão futura (podem ser adicionadas paletas alternativas — ex.: dark_theme, high_contrast) sem afectar o núcleo da biblioteca.

5.4.2 Utilização pelo módulo plot.py

Na versão atual da biblioteca, o módulo plot.py **não importa automaticamente** a paleta definida em *style.py*.
A gestão das cores é feita apenas com base no parâmetro opcional colors passado pelo utilizador.
Isto torna o funcionamento mais simples e flexível: quem quiser personalizar as cores pode fazê-lo diretamente, e quem não quiser não é obrigado a usar qualquer estilo pré-definido.

O comportamento é o seguinte:

- **Se o utilizador fornecer uma lista de cores**, essa lista é aplicada aos nós.
  Caso existam mais nós do que cores, o código acrescenta a cor "gray" para os restantes, garantindo que não ocorre erro e mantendo o gráfico visualmente consistente.

```
if colors:
    node_colors = colors[:len(all_nodes)]
    while len(node_colors) < len(all_nodes):
        node_colors.append('gray')
```

- **Se o utilizador não fornecer nenhuma lista**, o parâmetro `node_colors` fica a `None`, e o Plotly utiliza as suas cores padrão.

Desta forma, a biblioteca funciona imediatamente sem necessidade de configuração adicional.

- **Se o utilizador quiser usar a paleta definida em `style.py`**, pode importá-la manualmente:

5.4.3 Importância deste módulo na arquitetura

O módulo style.py desempenha um papel simples mas útil dentro da arquitetura da SankeyPy.
Embora não seja obrigatório para o funcionamento do gráfico, ele permite organizar num único local as paletas de cores e estilos visuais que podem ser utilizados pelo utilizador.

Manter estas definições separadas do módulo plot.py traz várias vantagens:

- **Evita sobrecarregar o código principal** com opções visuais que nem sempre são necessárias.
- **Facilita a personalização**, pois o utilizador pode importar diretamente um conjunto de cores já definido ou criar o seu próprio.
- **Permite expansão futura**, como a criação de temas "dark mode", temas de alto contraste ou estilos empresariais personalizados.
- **Mantém o design modular**, uma característica importante para evoluir a biblioteca sem quebrar funcionalidades existentes.

Assim, mesmo não sendo usado por defeito no pipeline interno, o módulo style.py oferece uma base organizada para gerir estilos e torna mais clara a separação entre **lógica de visualização** (plot) e **definições estéticas** (style).

5.5 Scripts de Demonstração e Testes Automatizados

Durante o desenvolvimento da SankeyPy, foram criados dois scripts complementares que desempenham papéis distintos no processo de demonstração, validação e verificação da biblioteca: **example_modular.py** e **generate_sankey_examples.py**.
Embora ambos utilizem internamente a função plot(), cada um tem um propósito diferente dentro do ecossistema da biblioteca.

5.5.1 example_modular.py — Demonstração Modular Interativa

O script **example_modular.py** foi concebido como um exemplo simples e modular destinado ao utilizador final. O seu objetivo é demonstrar, de forma clara e direta, como a biblioteca SankeyPy pode ser utilizada com um conjunto de dados real, ilustrando as principais funcionalidades:

- geração básica do diagrama Sankey;
- personalização de cores;
- aplicação de threshold;
- exportação para HTML e PNG.

Este script é frequentemente usado como primeiro ponto de contacto com a biblioteca, permitindo ao utilizador compreender as capacidades essenciais da SankeyPy através de chamadas diretas e intuitivas da função plot().

5.5.2 generate_sankey_examples.py — Testes Automatizados e Geração de Exemplos Visuais

Com a evolução do projeto e a necessidade de produzir um conjunto consistente de exemplos para validação e documentação, tornou-se necessário criar um segundo script: **generate_sankey_examples.py**.

Este script é agora parte oficial do projeto, funcionando como:

a) Ferramenta de testes automatizados

- Gera todos os casos de teste essenciais:
  ✔ Sankey básico
  ✔ threshold
  ✔ cores personalizadas
  ✔ orientação vertical
  ✔ dados inválidos

- Permite verificar rapidamente se alterações ao código interno mantêm os comportamentos esperados.
- Funciona como uma forma prática de *regressão visual*:
  se um diagrama deixar de ter o aspeto previsto, o script deteta imediatamente.

b) Sistema de geração automática de figuras

- Produz automaticamente todas as imagens utilizadas na documentação e no relatório.
- Garante consistência visual entre diferentes execuções.
- Permite repetição fácil dos testes em qualquer ambiente.

c) Ferramenta de apoio à validação interna

O script chama diretamente as funções da biblioteca, sem atalhos, garantindo que:

- o pipeline completo parser → plot → exportação funciona como esperado;
- os comportamentos implementados (validação, threshold, cores, orientação) são reproduzidos corretamente;
- as figuras produzidas refletem fielmente o estado atual da biblioteca.

Exemplo do seu funcionamento:

```
df = gerar_df_exemplo()
fig = fig_basic()
fig.write_image("figs/ex1_basic.png")
```

A presença de dois scripts distintos permite separar claramente dois papéis:

Script	Papel	Utilização
example_modular.py	Demonstração simples	Utilizadores finais
generate_sankey_examples.py	Testes + documentação	Desenvolvedores

CAPÍTULO 6 — Testes, Demonstração e Avaliação

6.1 Objetivos dos Testes

O presente capítulo tem como objetivo avaliar o comportamento da biblioteca SankeyPy em diferentes contextos de utilização, validando a robustez da implementação, a coerência visual dos diagramas gerados e o correto funcionamento das funcionalidades principais: validação de dados, agrupamento por threshold, personalização de cores, orientação vertical e compatibilidade com diferentes entradas.

A abordagem adotada combina testes unitários, testes exploratórios e geração sistemática de exemplos visuais.

6.2 Metodologia de Testes

Para garantir consistência e reprodutibilidade, foram utilizados dois scripts oficiais da biblioteca:

- **example_modular.py** — demonstração modular orientada ao utilizador final.

- **generate_sankey_examples.py** — ferramenta de testes automatizados e geração de figuras utilizadas neste capítulo.

A metodologia baseou-se nos seguintes princípios:

1. **Testes incrementais** — análise funcional isolada de cada parâmetro.
2. **Comparação "antes/depois"** — especialmente relevante no threshold.
3. **Testes de dados inválidos** — para validar mecanismos de segurança.
4. **Geração automatizada de gráficos** — eliminando variações manuais.
5. **Avaliação visual** — análise qualitativa das figuras produzidas.

Todos os gráficos apresentados resultam diretamente da execução da biblioteca sobre DataFrames reais, sem manipulação externa.

6.3 Validação de Dados e Comportamentos Esperados

A função `validar_dataframe()`, presente em *parser.py*, implementa verificações essenciais para prevenir erros silenciosos. As regras aplicadas são:

- remoção de linhas com valores nulos;
- exclusão de fluxos com valores negativos;
- lançamento de erro sempre que o DataFrame fica vazio.

Este comportamento assegura que a biblioteca não produz diagramas incoerentes. A validação foi verificada com um DataFrame contendo valores inválidos, resultando no ficheiro ***ex5_validation_log.txt***, que confirma a correta remoção dos mesmos.

6.3.1 Sankey Básico

O primeiro teste consiste na geração de um diagrama Sankey sem quaisquer parâmetros adicionais.

Código utilizado utilzador:

```
df = gerar_df_exemplo()
plot(df)
```

Trechos da biblioteca envolvidos:

Validação dos dados (parser.py)

```
df = df.dropna(subset=[source, target, value])
df = df[df[value] > 0]
```

Construção da lista de nós (plot.py)

```
all_nodes = list(set(df[source]) | set(df[target]))
label_to_index = {label: i for i, label in
enumerate(all_nodes)}
```

Construção das ligações (plot.py)

```
link=dict(
   source=df[source].map(label_to_index),
   target=df[target].map(label_to_index),
   value=df[value]
)
```

Resultado visual

- A lista all_nodes define todos os nós exibidos (A, B, C, D, E).
- A função map(label_to_index) cria as ligações (A→B, A→C, B→D…).
- O Sankey resultante representa fielmente o DataFrame sem alterações.

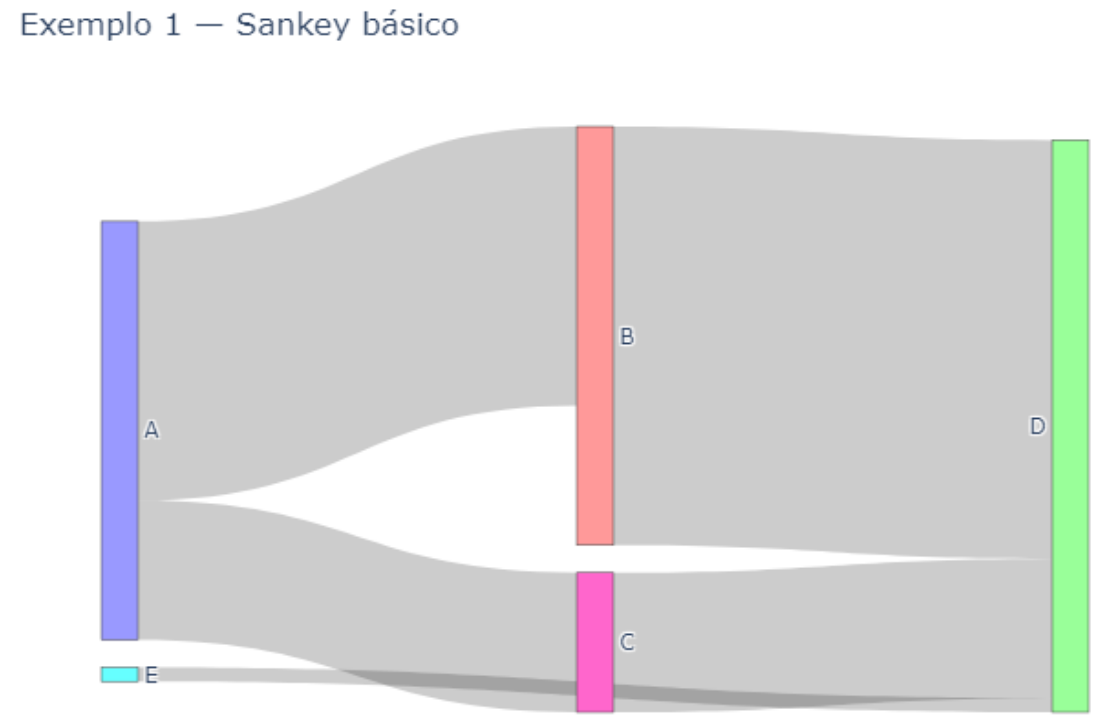


Figura 1 — Sankey básico gerado pela SankeyPy a partir do DataFrame de exemplo (sem threshold, paleta por defeito).

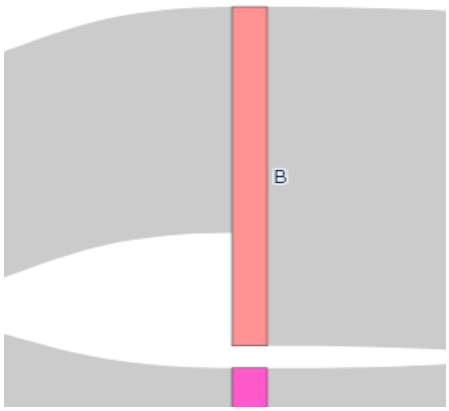


Figura 1.a — detalhe do link mostrando `source/target/value` no hover.

6.3.2 Threshold (Antes/Depois)

Este teste analisa o efeito do parâmetro threshold, responsável por consolidar fluxos pequenos num nó auxiliar.

```
plot(df, threshold=0.05)
```

Código utilizado pelo utilizador:

Trecho de códigol (parser.py)

```
if threshold is not None:
    total = df[value].sum()
    mask = df[value] < (threshold * total)
    if mask.any():
        agrupados = pd.DataFrame({
            source: ['Outros'] * mask.sum(),
            target: df.loc[mask, target],
            value: df.loc[mask, value]
        })
        df = pd.concat([df.loc[~mask],
agrupados], ignore_index=True)
```

O que este código faz:

- Calcula o total dos fluxos.
- Identifica valores abaixo de 5%.
- Substitui-os por um único nó "Outros".
- Reescreve o DataFrame, consolidando fluxos pequenos.

Resultado visual

- O fluxo E→D (0.5) desaparece do gráfico como linha individual.
- Surge um novo nó **“Outros”** ligado a D.
- O Sankey fica mais legível e menos “ruidoso”.

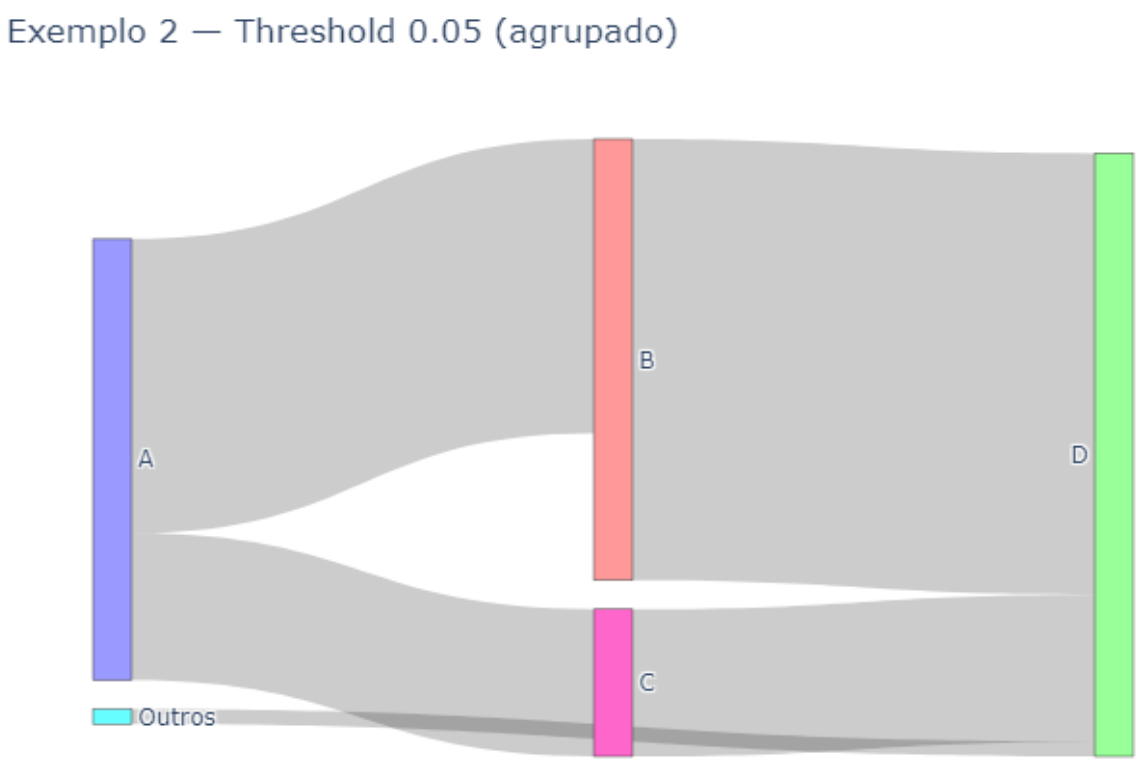


Figura 3 — Após threshold=0.05: fluxos residuais agrupados no nó "Outros", melhorando a legibilidade

6.3.3 Cores Personalizadas

Código usado pelo utilizador:

```
plot(df, colors=['#2E8B57', '#FFD700', '#FF6347'])
```

```
if colors:
    node_colors = colors[:len(all_nodes)]
    while len(node_colors) < len(all_nodes):
        node_colors.append('gray')
```

Trecho biblioteca (plot.py):

O que este código faz

- Aplica as cores fornecidas pelo utilizador aos nós.
- Garante que não há erro se o utilizador fornecer menos cores do que nós (preenche com “gray”).
- Mantém consistência visual.

Resultado visual

- Os nós A, B, C recebem as cores personalizadas.
- Nós adicionais (ex.: D, E) ficam a cinzento.
- Reforça a hierarquia visual do diagrama.

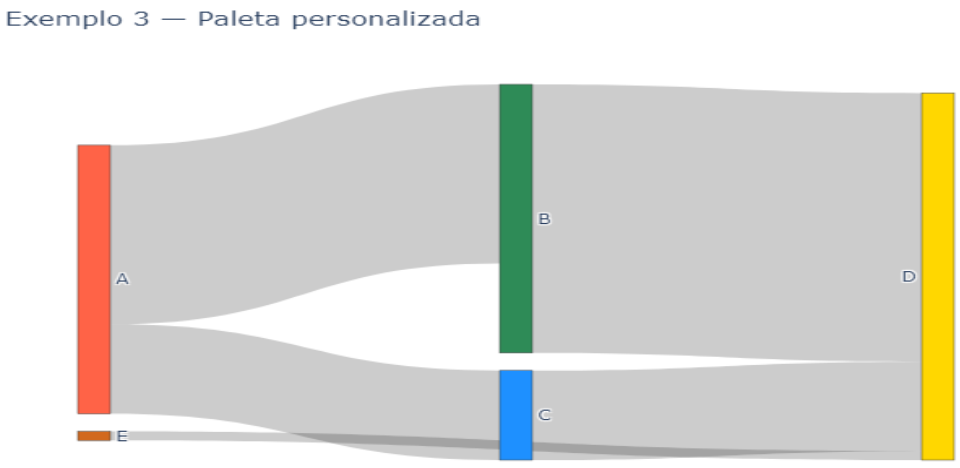


Figura 4 — Sankey com paleta de cores personalizada aplicada aos nós.

6.3.4 Orientação Vertical

Código usado pelo utilizador:

```
plot(df, orientation='v')
```

Trecho biblioteca (plot.py):

```
sankey_data = go.Sankey(
    orientation=orientation,
    node=dict(
        label=all_nodes,
        pad=15,
        thickness=20,
        color=node_colors
    ),
    link=dict(...)
)
```

O que este código faz

- Passa a orientação 'v' diretamente para o objeto Plotly.
- Muda o eixo principal: os nós passam de **horizontal** para **vertical**.
- O layout reorganiza-se automaticamente.

Resultado visual

- O Sankey é desenhado de cima para baixo.
- Fluxos seguem trajetórias verticais.
- Ideal para relatórios A4 ou painéis estreitos.

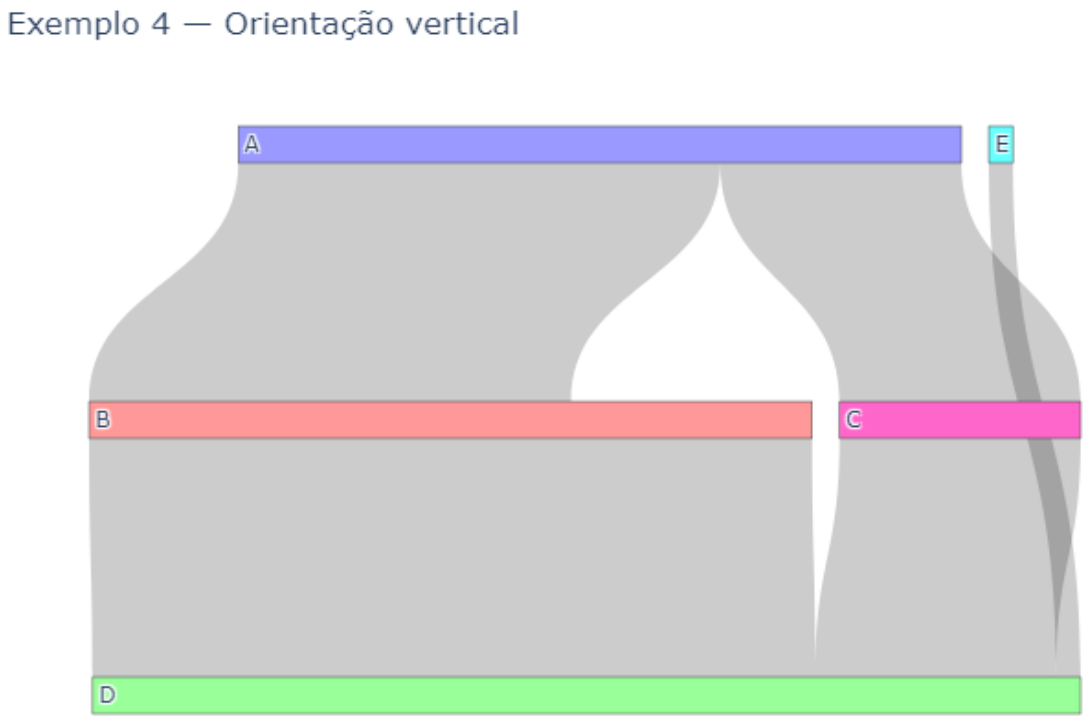


Figura 5 — Diagrama de Sankey com orientação vertical (orientation='v').

6.3.5 Tratamento de Dados Inválidos

Código usado pelo utilizador:

```
df_bad = pd.DataFrame({
   "source": ["A", None, "B"],
   "target": ["B", "C", "D"],
   "value": [10, -5, 7]
})
plot(df_bad)
```

Trecho biblioteca (parser.py):

```
df = df.dropna(subset=[source, target, value])
df = df[df[value] > 0]

if df.empty:
   raise ValueError("Não há dados válidos após
remover fluxos nulos/negativos.")
```

O que este código faz

- Remove linhas com None nas colunas essenciais.
- Remove fluxos com valor ≤ 0.
- Garante que o gráfico nunca é gerado com dados incorretos.
- Lança um erro claro se todos os dados forem inválidos.

Resultado visual

- Apenas os fluxos válidos são desenhados.
- O gráfico resultante mostra um conjunto reduzido de ligações — comprovando que a validação foi aplicada.

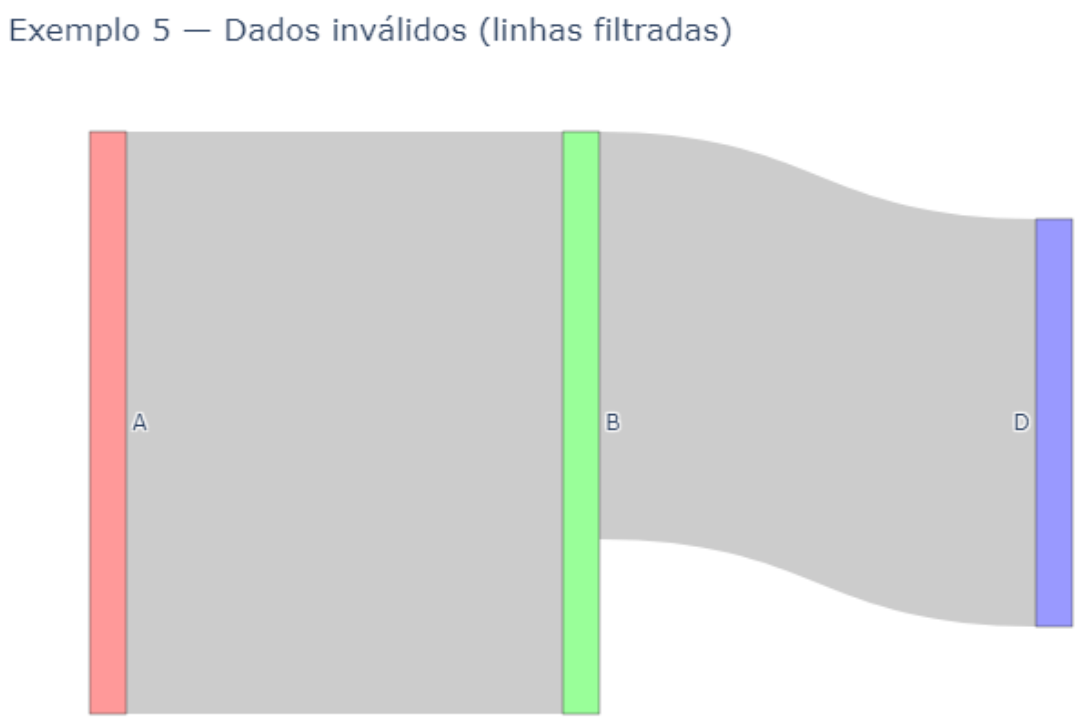


Figura 6 — Fluxos válidos após remoção automática de entradas inválidas pelo validador.

6.4 Aplicação Web de Demonstração (Streamlit)

De modo a facilitar a utilização prática da biblioteca desenvolvida e permitir a validação visual dos resultados produzidos, foi criada uma aplicação web simples utilizando **Streamlit**. Esta aplicação funciona como uma camada de demonstração interativa, permitindo ao utilizador gerar diagramas de Sankey sem necessidade de escrever qualquer código.

A interface disponibiliza um conjunto de funcionalidades básicas mas suficientes para testar o comportamento da biblioteca em múltiplos cenários. O utilizador pode:

- carregar um ficheiro CSV contendo os campos source, target e value;
- visualizar imediatamente os primeiros registos do dataset carregado;
- selecionar, através de menus suspensos, as colunas que representam origem, destino e valor dos fluxos;
- ajustar o parâmetro *threshold*, que controla o agrupamento automático dos fluxos de baixa expressão no nó "Outros";
- gerar dinamicamente o diagrama de Sankey;
- exportar o resultado final para **HTML**, mantendo a interatividade original do gráfico.

A aplicação não implementa lógica de processamento própria. Em vez disso, todas as operações de validação, pré-processamento e construção do diagrama são realizadas pela biblioteca SankeyPy. O Streamlit é utilizado apenas como camada de apresentação, com o objetivo de facilitar testes, demonstrações e a exploração dos dados de forma intuitiva.

As Figuras 6.1, 6.2 e 6.3 ilustram as principais fases de utilização da aplicação: a interface inicial, a pré-visualização do dataset e o diagrama de Sankey gerado a partir dos dados carregados.

Fig.6.1--Interface inicial da aplicação de demonstração desenvolvida em Streamlit.

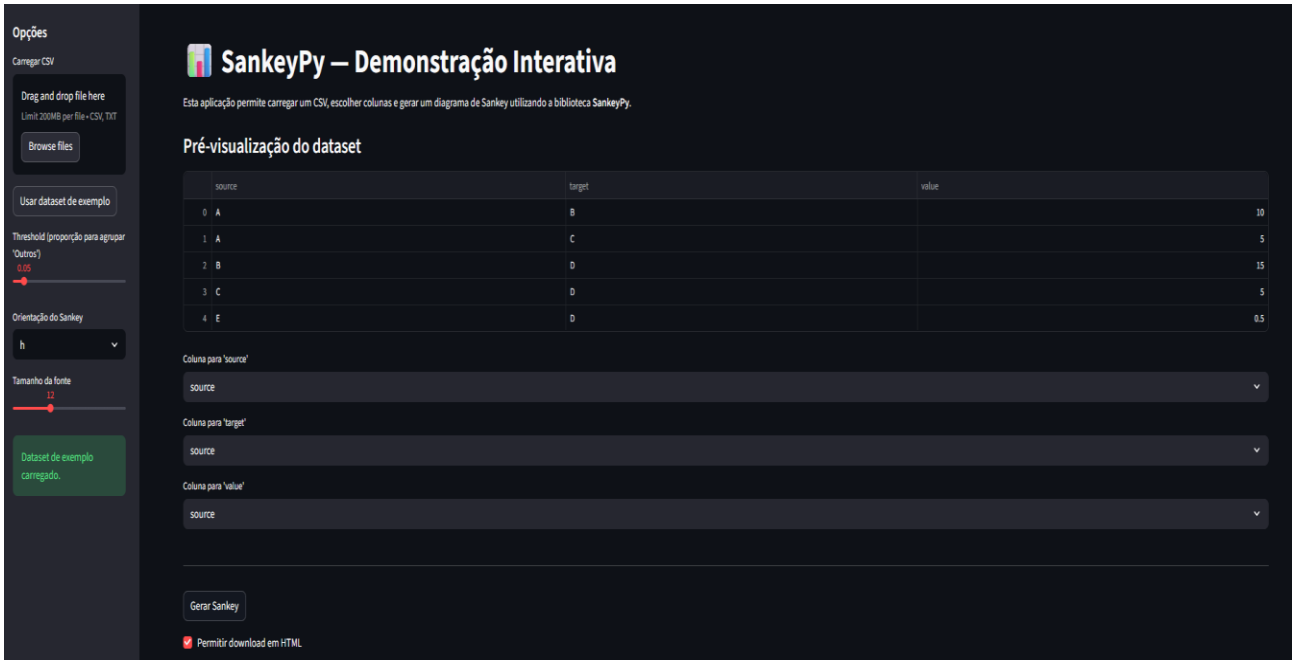


Figura.6.2 -- Pré-visualização do dataset carregado na aplicação Streamlit.

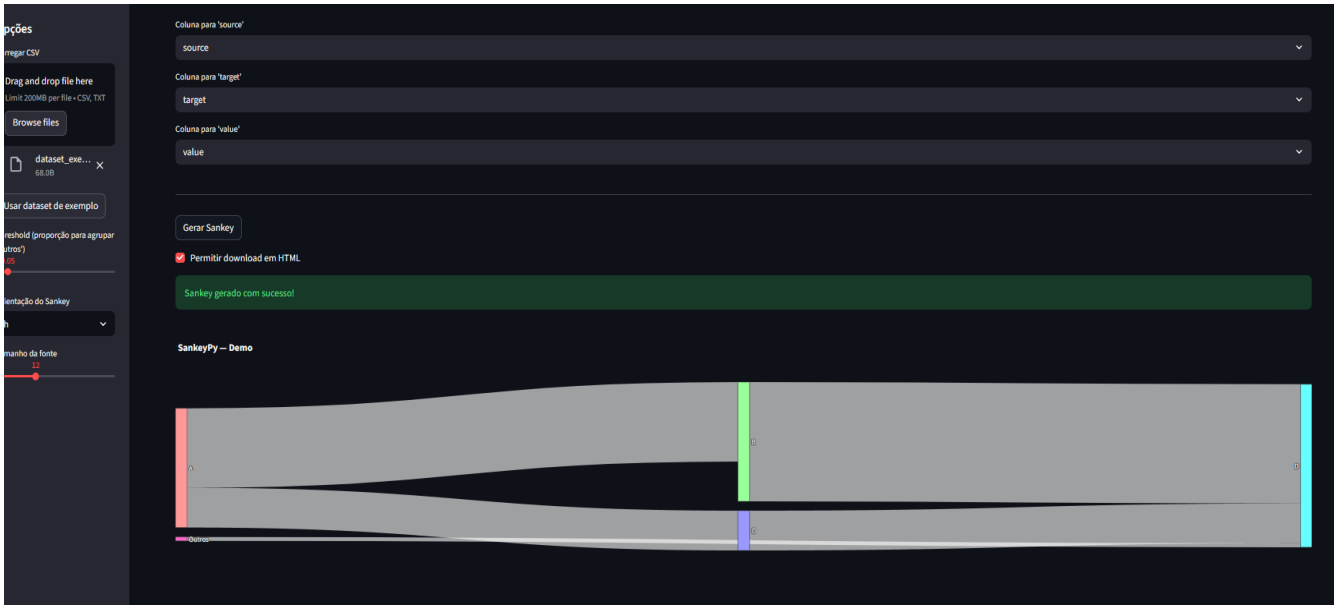


Fig.6.3--Exemplo de diagrama de Sankey produzido pela aplicação de demonstração baseada em Streamlit.

7. Discussão, Limitações e Trabalho Futuro

O presente capítulo apresenta uma análise crítica da solução desenvolvida, avaliando o grau de cumprimento dos objetivos definidos, as principais decisões técnicas tomadas, as limitações identificadas e as oportunidades de evolução futura. O trabalho desenvolvido permitiu validar a relevância da SankeyPy no contexto da visualização de dados em Python, demonstrando que é possível conciliar simplicidade de utilização, modularidade e flexibilidade através de uma arquitetura leve e extensível.

7.1 Discussão Geral dos Resultados

A biblioteca SankeyPy cumpriu plenamente os objetivos estabelecidos no início do projeto. A implementação de uma interface simples — construída em torno da função principal `plot()` — permitiu abstrair a complexidade inerente à utilização direta de bibliotecas como o Plotly Graph Objects, reduzindo significativamente a quantidade de código que o utilizador necessita de escrever para gerar um Sankey diagram funcional.

A modularidade foi um dos principais fatores de sucesso. A separação clara entre os módulos da biblioteca:

- **parser.py** (validação e pré-processamento de dados),
- **plot.py** (geração e configuração do diagrama),
- **style.py** (gestão de estilos visuais),
- **utils.py** (apoio a testes e demonstrações),
- **generate_sankey_examples.py** (validação automática e criação de figuras),

permitiu não apenas organizar o código de forma limpa e coerente, mas também facilitar a identificação de responsabilidades e a evolução individual de cada componente. Esta abordagem segue os princípios fundamentais da engenharia de software, nomeadamente separação de responsabilidades, reutilização e extensibilidade.

Do ponto de vista funcional, os resultados obtidos foram consistentes em diferentes cenários de teste. A biblioteca demonstrou:

- **robustez na validação de dados**, removendo automaticamente entradas nulas ou inválidas;
- **capacidade de lidar com fluxos residuais**, através do mecanismo de threshold que agrupa fluxos de pequena magnitude;
- **flexibilidade visual**, permitindo personalização de cores, orientação e layout;
- **compatibilidade com diversas formas de exportação**, incluindo HTML e formatos de imagem;
- **interatividade**, tirando partido das capacidades nativas do Plotly.

Em todos os testes realizados, os diagramas gerados apresentaram boa legibilidade, clareza das relações entre entidades e comportamento previsível — elementos essenciais para garantir a utilidade da biblioteca em contextos académicos e profissionais.

Em síntese, os resultados demonstram que a SankeyPy oferece uma solução eficiente, intuitiva e tecnicamente sólida para a geração de diagramas de Sankey, posicionando-se como uma alternativa simples a soluções mais complexas existentes no ecossistema Python.

Para além dos resultados obtidos ao nível da biblioteca, destaca-se ainda a implementação de uma aplicação web de demonstração baseada em Streamlit (Capítulo 6.4). Esta interface permitiu validar visualmente o comportamento da biblioteca em diferentes cenários, facilitando a exploração dos dados e a análise dos diagramas gerados. Embora simples, esta aplicação reforça a utilidade prática da SankeyPy e demonstra a sua capacidade de integração em ferramentas orientadas ao utilizador final.

7.2 Limitações da Solução Atual

Apesar do cumprimento dos objetivos e da eficácia demonstrada, a versão atual da SankeyPy apresenta algumas limitações que importa identificar e compreender no contexto do projeto.

1. Paleta de cores limitada

A biblioteca inclui apenas uma paleta de cores predefinida no módulo style.py. Embora suficiente para casos simples, esta abordagem:

- não inclui geração dinâmica de cores,
- não suporta paletas temáticas (dark mode, high-contrast),
- pode ser insuficiente para diagramas com muitos nós.

2. Dependência de Plotly e Kaleido

A SankeyPy depende do Plotly para renderização e do Kaleido para exportação de imagens. Embora seja a solução mais estável atualmente, esta dependência:

- pode causar incompatibilidades em ambientes sem suporte gráfico,
- pode dificultar utilização em plataformas remotas (ex.: Google Colab),
- limita alternativas de renderização estática.

3. Funcionalidades avançadas ainda não suportadas

A biblioteca foca-se numa API simples e limpa, mas não inclui:

- otimização automática do layout (posicionamento inteligente de nós),
- animações,
- múltiplos níveis hierárquicos (Sankey multilayer),
- clusters de nós ou agrupamentos semânticos,
- controlo avançado sobre espaçamento, curvatura ou caminhos das ligações.

Estes elementos, embora não essenciais para a versão atual, seriam relevantes em aplicações mais exigentes.

4. Documentação externa reduzida

A documentação encontra-se limitada ao relatório e aos exemplos incluídos no projeto. Para distribuição pública (ex.: PyPI), seria necessária documentação formal, incluindo:

- manual do utilizador,
- tutoriais,
- exemplos avançados,
- documentação automática via Sphinx ou MkDocs.

5.Limitações ao nível de workflows interativos

Apesar da aplicação Streamlit desenvolvida permitir uma interação básica com a biblioteca, esta solução funciona apenas como uma demonstração simples e não como um sistema completo de workflows interativos. A interface não disponibiliza mecanismos avançados, como edição visual dos fluxos, manipulação dinâmica dos nós, personalização detalhada do diagrama ou processamento de múltiplos datasets em sequência. Estas funcionalidades exigiriam abstrações adicionais e uma camada de interação mais sofisticada, que ultrapassam o objetivo desta fase inicial do projeto. Assim, a capacidade de criar workflows totalmente interativos permanece limitada, constituindo uma área clara de evolução futura.

7.3 Trabalho Futuro

O desenvolvimento da SankeyPy constituiu um primeiro passo sólido para a criação de uma biblioteca modular e extensível para a geração de diagramas de Sankey. No entanto, existem diversas possibilidades de evolução que poderão enriquecer significativamente a solução e torná-la mais versátil para diferentes cenários de utilização. As linhas de trabalho futuro apresentadas de seguida refletem não só as limitações identificadas, mas também oportunidades de expansão natural da biblioteca e da aplicação de demonstração.

7.3.1 Expansão da API Visual

A biblioteca oferece atualmente uma interface visual simples através da aplicação Streamlit desenvolvida no Capítulo 6.4. No futuro, esta interface poderá ser expandida de forma a incluir um conjunto mais amplo de funcionalidades, tais como:

- personalização mais fina dos nós (cor, formato, tamanho, agrupamento manual);
- manipulação visual das ligações diretamente na interface;
- suporte para múltiplos datasets e comparação lado a lado;
- apresentação de estatísticas complementares associadas aos fluxos (por exemplo, percentagens e distribuições).

Esta expansão permitiria transformar a interface atual, de natureza demonstrativa, numa ferramenta mais completa de exploração interativa de fluxos.

7.3.2 Funcionalidades Avançadas de Layout

A biblioteca utiliza neste momento o layout padrão fornecido pelo motor do Plotly. Uma linha de trabalho futuro consiste em integrar algoritmos de layout mais avançados, com o objetivo de:

- reduzir sobreposições entre links;
- otimizar a disposição espacial dos nós;
- suportar diagramas multi-nível ou hierárquicos;
- permitir layouts verticais e horizontais híbridos.

Estas melhorias possibilitariam a representação de fluxos mais complexos e de datasets de maior dimensão.

7.3.3 Integração com Dashboards Interativos

A arquitetura modular da SankeyPy torna-a naturalmente adequada para integração com frameworks de dashboards, como Dash, Streamlit ou aplicações web baseadas em frameworks Python ou JavaScript. A aplicação Streamlit desenvolvida neste projeto constitui já uma primeira demonstração desta integração. No entanto, versões futuras poderão:

- disponibilizar endpoints para utilização da biblioteca em dashboards dinâmicos;
- permitir filtragem, seleção e atualização do diagrama em tempo real;
- suportar interações avançadas, como drill-down ou colapso/expansão de nós.

Esta linha de evolução é particularmente relevante para cenários de visualização contínua ou integração em plataformas de Business Intelligence.

7.3.4 Publicação no PyPI

Uma evolução natural da biblioteca consiste na sua disponibilização pública através do repositório PyPI. Para tal, seria necessário:

- estruturar a biblioteca segundo as boas práticas de empacotamento Python,
- criar documentação oficial detalhada (API Reference),
- definir testes automatizados que assegurem a estabilidade das versões publicadas,
- preparar exemplos simples de utilização.

A publicação no PyPI aumentaria a visibilidade e a acessibilidade da SankeyPy, possibilitando o seu uso por terceiros.

8. Conclusões

O desenvolvimento da biblioteca SankeyPy permitiu alcançar um conjunto significativo de objetivos técnicos, científicos e académicos, posicionando este projeto como um contributo relevante para o ecossistema de ferramentas de visualização de dados em Python. O foco esteve, desde o início, em criar uma solução simples, modular e robusta, capaz de gerar diagramas de Sankey sem exigir ao utilizador conhecimento aprofundado das estruturas internas do Plotly ou de outras bibliotecas de baixo nível.

A SankeyPy demonstrou que é possível abstrair grande parte da verbosidade associada à construção deste tipo de visualização, oferecendo uma API intuitiva centrada na função plot(), que encapsula todo o pipeline de geração do diagrama — validação de dados, pré-processamento, mapeamento de nós e ligações, configuração visual e exportação. Esta abordagem permitiu garantir uma experiência de utilização fluida, coerente e orientada à prática.

A análise do estado da arte evidenciou que, apesar de existirem várias bibliotecas com suporte parcial a Sankey diagrams, poucas oferecem simultaneamente uma abordagem leve, especializada e com mecanismos explícitos de validação. A SankeyPy distingue-se, assim, por incluir funcionalidades não comuns em ferramentas generalistas, como a validação automática de DataFrames e o agrupamento inteligente de fluxos pequenos, integrando estes processos de forma natural no pipeline de construção do gráfico.

A demonstração prática apresentada no Capítulo 6, complementada pela aplicação web desenvolvida em Streamlit, permitiu validar a utilidade real da biblioteca em diferentes cenários. A interface simples disponibilizada através do Streamlit revelou-se particularmente útil para exploração visual, permitindo testar rapidamente parâmetros, ajustar thresholds e gerar diagramas sem escrever código. Esta componente reforça a aplicabilidade prática da SankeyPy e evidencia a sua capacidade de integração em ferramentas orientadas ao utilizador final.

Os testes realizados confirmaram a eficácia e robustez da biblioteca perante diferentes tipos de dados, incluindo casos com entradas inválidas ou fluxos residuais. Os diagramas gerados mostraram-se consistentes, visualmente claros e facilmente interpretáveis, beneficiando das capacidades interativas do Plotly e das funcionalidades adicionais de exportação para HTML e formatos de imagem.

Do ponto de vista académico, o projeto permitiu aplicar de forma integrada conhecimentos de análise de requisitos, arquitetura de software, programação modular, validação de dados e documentação técnica. A solução desenvolvida constitui uma base sólida para futuras extensões, que poderão incluir temas visuais adicionais, algoritmos avançados de layout, integração com dashboards de dados, animações e eventual publicação oficial da biblioteca no PyPI.

Em síntese, a SankeyPy demonstra que é possível combinar simplicidade, extensibilidade e clareza visual através de uma implementação bem estruturada e orientada ao utilizador. O trabalho realizado evidencia a capacidade de conceber, implementar e avaliar software especializado para visualização de dados, contribuindo de forma significativa para o domínio da Engenharia Informática.

Bibliografia

Plotly Technologies Inc. (2024). *Plotly Python Graphing Library*. Disponível em https://plotly.com/python

Kaleido Project. (2024). *Kaleido: Static Image Export for Plotly*. GitHub Repository: https://github.com/plotly/Kaleido

Google Charts. (2024). *Sankey Diagram Documentation*. https://developers.google.com/chart/interactive/docs/gallery/sankey

Cairo, A. (2020). How Charts Lie: Getting Smarter about VIsual Information. WW Norton.

Kirk, A. (2019). Data Visualization: A Handbook for Data Driven Design. SAGE Publications.

McCandless, D. (2012). Information is beautiful. London: Collins

Munzner, T. (2014). Visualization Analysis and Design. CRC Press

Schwabish, J. (2021). Better Data Visualizations. Columbia University Press.

Tufte, E. R., & Graves-Morris, P. R. (1983). The visual display of quantitative information. Cheshire, CT: Graphics press.