Projeto Hedge de Ativos %CDI — Guia Operacional (README)

Este repositório possui um **pipeline em 5 scripts** (executados nesta ordem) que:

1. consolida dados extraídos de PDFs (CETIP/B3) com características e fluxo dos ativos;
2. cruza isso com o relatório de posição (carteira) + exceções;
3. calcula/atribui **taxa efetiva** por ativo (via ISIN em carteiras BTG ou via matriz de curvas por rating/YMF);
4. baixa e interpola a **curva DI (B3)** por **DU**;
5. (opcional) faz scrap de debêntures (ANBIMA) para alimentar a “Calculadora de Debêntures”.

No final, o aplicativo `cota.py` (Streamlit) consome os artefatos gerados para:

- “Match de Ativos e DI” (tabela e gráficos);
- calculadoras (Bancários, Debêntures, Fundos).

0) Pré-requisitos

0.1) Python e ambiente

- Recomendado: **Python 3.10+**.
- Crie um ambiente virtual e instale as dependências.

Exemplo (Windows):

```
python -m venv .venv
.venv\Scripts\activate
pip install -U pip
```

0.2) Dependências (mínimo)

A lista exata pode variar conforme seus arquivos, mas o pipeline usa, de forma recorrente:

- `pandas`, `numpy`

- `openpyxl`, `xlsxwriter`
- `requests`
- `pandas_market_calendars`
- `unidecode`
- (para PDF) `pymupdf` (fitz), `pdfplumber`, `PyPDF2` (fallbacks possíveis)
- (para interpolação) `scipy` (opcional, melhora a interpolação; o script cai em `np.interp` se não existir)
- (para o app) `streamlit`, `altair`, `plotly`

Exemplo de instalação:

```
pip install pandas numpy openpyxl xlsxwriter requests pandas-market-calendars unidecode
pip install pymupdf pdfplumber pypdf2
pip install scipy
pip install streamlit altair plotly
```

0.3) Selenium / ChromeDriver (somente para o script 5)

O `5-ScrapDebentures.py` usa Selenium com Chrome. Garanta que:

- o **Google Chrome** esteja instalado;
- o **ChromeDriver** compatível esteja disponível no PATH (ou configure o `Service()` corretamente).

1) Estrutura esperada de pastas/arquivos

O pipeline usa (por padrão) a pasta **`Dados/`** (criada automaticamente quando necessário).

Arquivos típicos consumidos/gerados (resumo):

- **Saídas do passo 1**: `Dados/consolidado_pdfs_codativos.xlsx` e (opcional) `Dados/consolidado_pdfs_codativos.csv`
- **Saídas do passo 2**: `Dados/ativos_mapeados_para_controle.xlsx` (base para o passo 3)
- **Saídas do passo 3**: `Dados/ativos_mapeados_com_taxa_efetiva.xlsx` + `Dados/ativos_sem_taxa.xlsx` (se necessário)
- **Saídas do passo 4**: `Dados/b3_taxas_ref_di_raw.parquet` e `Dados/curva_di_interpolada_por_DU.parquet`
- **Saídas do passo 5 (opcional)**: `Dados/flux_deb.csv` (ou similar, dependendo do seu fluxo no app)

Além disso, o app menciona arquivos auxiliares como:

- `Dados/Tratamento Exceções.xlsx`
- `Dados/Base de Deb e Prêmios.xlsx`
- `feriados_nacionais.xls`
- `Dados/cdi_cached.csv` (fallback)

2) Ordem de execução (o “runbook”)

Passo 1 — `1-mapear_fluxo_telas_cetip.py`

Objetivo
Consolidar a extração dos PDFs (CETIP/B3) e gerar uma base tabular por ativo (`cod_Ativo`), incluindo campos como emissor, ISIN, datas, PU, forma de CDI, fluxo/agenda de juros, calls, e checks de consistência.

Principais saídas

- `Dados/consolidado_pdfs_codativos.xlsx`
- `Dados/consolidado_pdfs_codativos.csv` (se habilitado no script)

Como rodar

1. Garanta que os PDFs de origem e o diretório de saída estejam configurados (variáveis do script).
2. Execute:

```
python 1-mapear_fluxo_telas_cetip.py
```

Checklist de sucesso

- O script imprime no console o caminho do XLSX/CSV gerado e estatísticas (FormaCDI, sanity check etc.).
- Abra o XLSX e valide especialmente:
  - aba `sanity_issues` (linhas com sanity != OK)
  - aba `cods_unicos` (um por `cod_Ativo`)

Passo 2 — `2-Gera_Base.py`

Objetivo

Montar uma “base de controle” para o pipeline de hedge:

- lê a base do passo 1 (PDF consolidado);
- lê um **Relatório de Posição** (carteira/posição) e aplica filtros/normalizações;
- aplica **Tratamento de Exceções** (planilha);
- resolve/mantém um **mapa de códigos** (por chave estrita) para preservar identificadores e evitar duplicidades.

O script usa uma **chave estrita**:

`Sub Classe | Ativo | Emissor | Fundo | Vencimento_final`

Principais saídas

- `Dados/ativos_mapeados_para_controle.xlsx`
  (arquivo que será consumido no passo 3)
- Atualização/persistência do “mapa de códigos” utilizado pelo processo

Como rodar

1. Coloque o relatório de posição no diretório esperado (o script procura por glob).
2. Garanta que `Dados/Tratamento Exceções.xlsx` esteja presente (se aplicável).
3. Execute:

```
python 2-Gera_Base.py
```

Checklist de sucesso

- No console, o script mostra contagens por chave estrita e imprime onde salvou:
  - as “entradas sem código (chave estrita)”
  - o mapa de códigos preservando `Fundo`

Passo 3 — `3-Scrapy_Taxas_MTM.py`

Objetivo (Fluxo unificado de taxas)

Ler `Dados/ativos_mapeados_para_controle.xlsx` e atribuir **TAXA_EFETIVA** por ativo, seguindo a prioridade:

1. Para fundos BTG, tenta obter **TAXA_ISIN** (último dia disponível do fundo nas carteiras BTG);
2. Para todos, calcula fallback por **Matriz de Curvas** (rating/YMF) → `Taxa_matriz` ;
3. Define `TAXA_EFETIVA = TAXA_ISIN` se existir, senão `Taxa_matriz` ;
4. Propaga taxa entre fundos para o mesmo ativo (mesma chave sem `Fundo`).

Principais entradas

- `Dados/ativos_mapeados_para_controle.xlsx`
- `Dados/Matriz de Curvas 09012026.xlsx` (ou o arquivo configurado no script)
- Acesso ao diretório de carteiras BTG (pasta em `Z:\Asset Management` , se aplicável)

Principais saídas

- `Dados/ativos_mapeados_com_taxa_efetiva.xlsx`
- `Dados/ativos_sem_taxa.xlsx` (se restarem itens sem taxa)

Como rodar

1. Ajuste os paths de `BASE_DIR` , `SPECIFIC_DATE` , e o arquivo da matriz (se necessário).
2. Execute:

```
python 3-Scrapy_Taxas_MTM.py
```

Checklist de sucesso

- O script imprime um RESUMO com:
  - quantos vieram por ISIN
  - quantos vieram por matriz
  - quantas taxas foram propagadas
  - quantos ficaram sem taxa

Passo 4 — `4-ScrapTaxasReferencias.py`

Objetivo

Baixar a **Taxa Referência da B3** (ex.: PRE/DI), ler o CSV, e gerar uma curva interpolada por **DU** usando **log-DF** com **PCHIP** (SciPy) ou fallback linear.

Além de atualizar um histórico `RAW` , ele gera um parquet “interp” que o app consome.

Principais entradas/configurações

- `USE_SPECIFIC_DATE` e `SPECIFIC_DATE` para fixar a data (ou rodar via CSV local)
- `feriados_nacionais.xls` (opcional, para ajustar calendário)
- `RATE_ID = "PRE"` (pode ser alterado conforme o tipo de curva desejada)

Principais saídas

- `Dados/b3_taxas_ref_di_raw.parquet` (histórico por `ref_date` e nós)
- `Dados/curva_di_interpolada_por_DU.parquet` (curva por DU para a `ref_date`)

Como rodar

1. Ajuste a data em `SPECIFIC_DATE` (se for usar data fixa).
2. Execute:

```
python 4-ScrapTaxasReferencias.py
```

Checklist de sucesso

- O script imprime: `RAW=... | INTERP=...` e as primeiras linhas (`head()`).
- Se der erro de parse, ele orienta verificar `DEBUG_parse_csv_*.txt` dentro de `Dados/`.

Passo 5 (Opcional) — `5-ScrapDebentures.py`

Objetivo

Extrair da ANBIMA (via Selenium) informações e fluxo de pagamento de debêntures, incluindo:

- tabela do fluxo,
- `VNE` (derivado do VNA) e
- `TaxaEmissao` (remuneração em decimal).

Principais entradas

- Lista `ativos = [...]` no próprio script (tickers de debêntures).

Principais saídas

- `Dados/flux_deb.csv` (tabela consolidada do fluxo + colunas auxiliares)

Como rodar

1. Ajuste `ativos = ["HAPV21", "KLBNA2", ...]`.

2. Garanta Selenium/ChromeDriver ok.
3. Execute:

```
python 5-ScrapDebentures.py
```

Checklist de sucesso

- O script salva `Dados/flux_deb.csv` .
- Se a “Taxa ANBIMA do ativo” não aparecer, ele tenta buscar a taxa em `/caracteristicas` e preenche no input da calculadora.

3) Como rodar o app `cota.py` (Streamlit)

O `cota.py` é um aplicativo Streamlit que consolida as visões e calculadoras. Ele espera que o pipeline acima já tenha sido executado (especialmente o passo 3 e o passo 4), porque consome:

- `Dados/ativos_mapeados_com_taxa_efetiva.xlsx`
- `Dados/curva_di_interpolada_por_DU.parquet`
- e, dependendo da visão, arquivos auxiliares como `Dados/Tratamento Exceções.xlsx` , `Dados/Base de Deb e Prêmios.xlsx` , `Dados/flux_deb*.csv` , `Dados/cdi_cached.csv` .

Como iniciar

```
streamlit run cota.py
```

Navegação (Sidebar)

O app possui um seletor de “Visão” com opções como:

- **Match de Ativos e DI**
- **Calculadora de Bancarios**
- **Calculadora de Debentures**
- **Calculadora de Fundos**

Visão: “Match de Ativos e DI”

O foco é analisar o hedge em DI e o “casamento” entre:

- o que está na carteira/relatório
- e o que veio do PDF consolidado (passo 1/2),
  gerando:
- tabela de match (com download XLSX),
- gráfico da curva DI,
- comparação %CDI (PDF × Relatório).

Visão: “Calculadora de Debentures”

Essa visão carrega:

- `Dados/Base de Deb e Prêmios.xlsx` (aba `BaseAtivos&Premios`)
- um CSV de fluxo (ex.: `Dados/flux_deb2.csv` no seu código)
  e disponibiliza uma calculadora de “BE/precificação” com parsing robusto de taxas em formato PT-BR.

Observações importantes

- O app implementa fallback de CDI via `cdi_cached.csv` (se existir) e caches em `st.session_state` / `@st.cache_data`.
- O app também tem um bloco de comparação/calendário (B3 × arquivo) para diagnosticar divergência de dias úteis.

4) Problemas comuns (Troubleshooting)

1. **Arquivos não encontrados**

- Verifique se você rodou os passos na ordem e se a pasta `Dados/` contém os arquivos esperados.

2. **Sem acesso ao caminho `Z:\Asset Management`**

- Ajuste `BASE_DIR` no passo 3 (ou desabilite a parte de leitura de carteiras BTG, se aplicável).

3. **Erro no download da B3 (passo 4)**

- O script salva arquivos `DEBUG_*` em `Dados/` para inspeção.
- Confira se o endpoint retornou CSV direto, base64, ou ponteiro/URL.

4. **Selenium não abre / driver incompatível (passo 5)**

- Atualize o ChromeDriver e confirme compatibilidade com a versão do Chrome.

5) TL;DR (Executar do zero)

```
python 1-mapear_fluxo_telas_cetip.py
python 2-Gera_Base.py
python 3-Scrapy_Taxas_MTM.py
python 4-ScrapTaxasReferencias.py
# opcional:
python 5-ScrapDebentures.py

streamlit run cota.py
```