Dashboard de Risco (Streamlit) — `app4.py` + Pipeline de Atualização (Scrapers/ETL)

Este repositório contém um **dashboard em Streamlit** (`app4.py`) para análise e gestão de portfólio (VaR, stress, simulação de cota etc.) e um **pipeline de atualização de dados** via scripts auxiliares:

- `ScrapAF3.py` → coleta **PL diário** dos fundos no portal interno AF (Selenium) e salva histórico em parquet
- `ScrapB3_v2.py` → coleta **dados B3** (DI/DAP/WDO/TREASURY) e gera bases wide/long em parquet (+ CSV/JSON pt-BR)
- `TransformarRetornosParquet.py` → "ETL canivete suíço": converte CSV→parquet, gera `df_inicial` , `df_divone` , mescla NTNBs no parquet B3, exporta série LFT

Sumário

1) Visão geral do sistema

O projeto é composto por:

1. **App (Streamlit):** `app4.py`
   - Permite montar/editar portfólio, analisar risco (VaR, vol), rodar stress tests e simular cota.
   - Usa **dados locais (parquet/csv/xlsx)** e também persiste um "snapshot" do portfólio no **Supabase**.
2. **Atualização de dados (ETL):**
   - `ScrapAF3.py` atualiza PL histórico dos fundos no parquet `Dados/pl_fundos_teste.parquet` (ou `Dados/pl_fundos.parquet`)
   - `ScrapB3_v2.py` atualiza bases B3 em:
     - `Dados/df_preco_de_ajuste_atual_completo.parquet` (preços/ajustes)
     - `Dados/df_valor_ajuste_contrato.parquet` (variações/valores por contrato)
     - e uma base longa `Dados/df_ajustes_b3.parquet`
   - `TransformarRetornosParquet.py` normaliza e produz:
     - `Dados/df_inicial.parquet` (universo/histórico wide)
     - `Dados/df_divone.parquet` (DV01/DIV01)
     - merge de NTNBs no parquet de preços B3
     - `Dados/dados_lft.csv` (retorno LFT)

2) Estrutura de diretórios

Estrutura recomendada:

```
.
├─ app4.py
├─ ScrapAF3.py
├─ ScrapB3_v2.py
├─ TransformarRetornosParquet.py
├─ Dados/
│  ├─ config.json
│  ├─ pl_fundos.parquet                          (opcional)
│  ├─ pl_fundos_teste.parquet                    (principal na prática atual)
│  ├─ df_inicial.parquet
│  ├─ df_divone.parquet
│  ├─ df_preco_de_ajuste_atual_completo.parquet
│  ├─ df_valor_ajuste_contrato.parquet
│  ├─ df_ajustes_b3.parquet
│  ├─ df_preco_de_ajuste_atual_completo.csv
│  ├─ df_valor_ajuste_contrato.csv
│  ├─ df_preco_de_ajuste_atual_completo.json
│  ├─ BBG - ECO DASH.xlsx
│  ├─ FechamentoNTNBs.xlsx
│  └─ dados_lft.csv
└─ BaseFundos/
   ├─ <FUNDO_1>.parquet
   ├─ <FUNDO_2>.parquet
   └─ ...
```

3) Como rodar (ordem recomendada)

Ordem diária sugerida para atualizar tudo que o app4.py usa.

1. **Atualizar PL dos fundos (AF interno)**
   python ScrapAF3.py
2. **Atualizar dados B3 (DI/DAP/WDO/TREASURY)**
   python ScrapB3_v2.py
3. **Rodar normalizações/ETL (BBG → parquets, NTNB merge, LFT, conversões)**
   python TransformarRetornosParquet.py
4. **Abrir o app**
   streamlit run app4.py

3) Como rodar (ordem recomendada)

Ordem diária sugerida para atualizar tudo que o `app4.py` usa:

1. **Atualizar PL dos fundos (AF interno)**

```
python ScrapAF3.py
```

1. **Atualizar dados B3 (DI/DAP/WDO/TREASURY)**

```
python ScrapB3_v2.py
```

1. **Rodar normalizações/ETL (BBG → parquets, NTNB merge, LFT, conversões)**

```
python TransformarRetornosParquet.py
```

1. **Abrir o app**

```
streamlit run app4.py
```

4) `app4.py` --- o que faz e como funciona

O `app4.py` é um dashboard Streamlit que:

- Monta/edita um portfólio de ativos (ex.: DI, DAP, WDO1, TREASURY, IBOV, NTNBs etc.)
- Calcula métricas de risco (VaR, volatilidade, contribuições)
- Faz stress tests (choques em juros e FX usando DV01/"DIV01" e transformações auxiliares)
- Mantém um snapshot do portfólio atual em Supabase (`portfolio_posicoes`) e também salva um espelho local em parquet
- Mantém histórico por fundo em `BaseFundos/<FUNDO>.parquet`
- Possui tela de Simulação de Cota (NAV) usando série de PL + retornos (LFT + análises internas)

4.1) Login

O app implementa um login antes de liberar as telas:

- **Config:** `Dados/config.json`
- **Sessão:** expira após `session_timeout_min`
- **Hash:** usa `bcrypt` se o hash estiver no JSON

Formato esperado (exemplo):

```
{   "users": {     "usuario1": "$2b$12$...hash_bcrypt...",     "usuario2": "$2b$12$...hash_bcrypt..."   },   "auth": {     "session_timeout_min": 6
```

Observação: pode haver fallback para senha em texto puro (não recomendado).

4.2) Persistência (Supabase ↔ parquet local)

O app utiliza Supabase para armazenar o portfólio atual:

- **Tabela:** `portfolio_posicoes`
- **Ações típicas:**
  - `load` (lê tudo)
  - `add` (apaga tudo e reinsere --- replace total)
  - `update` (atualiza por chave: Ativo + Dia de Compra)
  - `delete` (remove por Ativo + Dia de Compra)
- **Espelho local:**
  - `Dados/portifolio_posições.parquet`
- **Fluxo comum do app:**
  - Atualizar base de fundos / sincronizar portfólio
  - Carregar dados locais (PL, `df_inicial` etc.)
  - Operar nas telas

4.3) Arquivos que o app consome

Principais arquivos/datasets usados pelo `app4.py` :

- **PL dos fundos**
  - `Dados/pl_fundos.parquet` **ou** `Dados/pl_fundos_teste.parquet`
- **Universo e histórico de preços**
  - `Dados/df_inicial.parquet`
- **DV01/DIV01 (stress)**
  - `Dados/df_divone.parquet`
- **Preços/ajustes B3**
  - `Dados/df_preco_de_ajuste_atual_completo.parquet`
  - `Dados/df_valor_ajuste_contrato.parquet`

- **Macro/Bloomberg**
  - `Dados/BBG - ECO DASH.xlsx` (quando usado no pipeline para gerar parquets)
- **Retorno LFT (simulação de cota)**
  - `Dados/dados_lft.csv`

4.4) Telas e fluxo de uso

O app organiza o uso por "telas" via sidebar (por exemplo):

4.4.1) Ver Portfólio

- Visualização do portfólio consolidado e outputs (dependendo da implementação local).

4.4.2) Adicionar Ativos

Fluxo típico:

1. Escolher ativos do universo (colunas de `df_inicial.parquet`)
2. Informar quantidades e preços (com suporte opcional a CSV de execuções)
3. Calcular retornos/vol/VaR do portfólio
4. Rodar stress tests usando DV01 e normalizações
5. Persistir portfólio (Supabase/parquet) e alimentar `BaseFundos` quando aplicável

4.4.3) Remover Ativos / Apagar Dados

- Remove entradas do portfólio atual (por ativo e data de compra) e/ou faz limpeza.

4.4.4) Simular Cota

- Simula o NAV (cota) com base em PL diário + séries de retorno (LFT + componentes do portfólio).

4.4.5) CSV de execuções (opcional)

O app permite subir um CSV de execuções para pré-preencher:

- quantidade por ativo
- preço médio/compra

Além disso, faz "normalização" de tickers (ex.: DI e DAP) dependendo do padrão.

4.5) Simulação de Cota

A simulação de cota normalmente:

1. Pede % do PL para investir
2. Carrega série de PL total e série de retorno LFT
3. Combina com retornos do portfólio/ativos e aplica custos (taxa adm/performance) se configurado
4. Gera uma série simulada de NAV/cota e métricas associadas

5) Pipeline de dados --- scripts

5.1) `ScrapAF3.py` (PL fundos)

5.1.1) Objetivo

Coletar PL/patrimônio dos fundos no portal interno AF via Selenium e manter histórico em parquet.

- **Saída:** `Dados/pl_fundos_teste.parquet` (configurável)

5.1.2) Principais parâmetros

- `PARQUET_PATH` : caminho do parquet final
- `TARGET_FUND` , `CUTOFF_DATE` : regra para zerar fundo a partir de uma data
- `LOOKBACK_DAYS` : janela para detectar valores repetidos e forçar re-scrape
- `INDICES_TOTAL` : índices de linhas para calcular o TOTAL (hardcoded)
- `IGNORE_FUND_DUPES` : fundos ignorados na regra de duplicados

5.1.3) Processual completo

1. Carrega parquet existente (ou cria base inicial `fundos_base`)
2. Descobre última data já salva (colunas `YYYY-MM-DD`)
3. Define:
   - `new_dates` : todas as datas do dia seguinte até hoje (inclui fds/feriados)
   - `dates_repeated` : dias úteis recentes com PL repetido para re-scrape
4. Selenium:
   - login no portal
   - abre relatório de patrimônios
   - seleciona data custom
5. Para cada data a coletar:
   - digita a data
   - lê a tabela
   - aplica regra de zerar fundo após cutoff
   - preenche coluna `YYYY-MM-DD`
   - recalcula TOTAL somando os fundos de `INDICES_TOTAL`
6. Reforça a regra do fundo zerado em TODAS as colunas `>= cutoff` e recalcula TOTAL
7. Finaliza:
   - calcula Último Valor
   - ordena colunas
   - forward-fill horizontal (substitui `"--"` pelo dia anterior)
   - salva parquet

5.1.4) Saídas

- `Dados/pl_fundos_teste.parquet` (ou `Dados/pl_fundos.parquet`)

5.1.5) Pontos de atenção

- Credenciais hardcoded (risco alto)
- TOTAL calculado por índice (quebra se ordem/lista mudar)
- Se aparecer fundo novo, precisa adicioná-lo em `fundos_base` para ter linha no histórico

5.2) `ScrapB3_v2.py` (B3 DI/DAP/WDO/TREASURY)

5.2.1) Objetivo

Consumir o endpoint da B3 `ConsolidatedTradesDerivatives` , filtrar e transformar em:

- `Dados/df_preco_de_ajuste_atual_completo.parquet` (Ajuste)
- `Dados/df_valor_ajuste_contrato.parquet` (Variação/Valor por contrato)

Além de CSV/JSON pt-BR e log de execução.

5.2.2) O que entra e o que sai

Entradas

- Parquets wide existentes (se já existirem)
- Endpoint B3 via POST

Saídas

- `Dados/df_ajustes_b3.parquet` (base longa histórica)
- `Dados/df_preco_de_ajuste_atual_completo.parquet` + `.csv` + `.json`
- `Dados/df_valor_ajuste_contrato.parquet` + `.csv`
- `atualizacao_b3_log.txt`
- `debug_b3_csv/` (se ativado)

5.2.3) Processual completo

1. Carrega `wide_preco` e `wide_valor` já existentes
2. Determina `start_dt` = última data presente
3. Define `end_dt_hist` = último dia útil B3 antes de hoje
4. Para cada dia útil B3 no range:
   - baixa CSV do endpoint
   - se o dia estiver vazio, faz backoff/backfill usando dias úteis anteriores
   - faz parse do CSV (normaliza colunas e converte pt-BR → float)
   - reduz universo aplicando regras de "vértices":
     - **DI:** apenas mês "F" por ano (`DI_yy`)
     - **DAP:** regra par/ímpar (`DAPyy` , mantendo K/Q conforme regra)
     - **WDO1:** escolhe contrato mais próximo (WDO preferencial, DOL fallback)
     - **TREASURY:** escolhe contrato T10 mais próximo (>= mês referência)
   - cria colunas wide:
     - preço = `PrecoAjusteAtual`
     - valor = `Pontos` , exceto DAP que usa `ValorAjusteR$`
5. Tenta "hoje" (no código atual, tenta `today = hoje - 1 dia`), sem backfill
6. Cria uma coluna duplicada para o próximo DU B3 como cópia da última (se ainda não existir)
7. Exporta tudo e registra logs

5.2.4) Pontos de atenção

- Mudança no layout/colunas do CSV da B3 pode quebrar o parse
- A duplicação do próximo DU não é "dado real" (cuidado em análises)
- Exclusões (`ASSETS_EXCLUIR`) precisam acompanhar o universo do app

5.3) `TransformarRetornosParquet.py` (ETL/normalizações)

5.3.1) Objetivo

Script de normalização e geração de bases para o app:

- Converter CSVs (root e `BaseFundos/`) → parquet
- Gerar `df_inicial.parquet` e `df_divone.parquet` a partir do Excel `BBG - ECO DASH.xlsx`
- Mesclar NTNBs (`FechamentoNTNBs.xlsx`) dentro do parquet de preços B3 e fazer forward-fill por data
- Exportar série LFT (`dados_lft.csv`) a partir de planilha interna em rede

5.3.2) Processual completo

1. Converte `*.csv` do diretório atual para `.parquet` e remove os CSVs
2. Converte `BaseFundos/*.csv` para `.parquet` e remove os CSVs
3. Lê `Dados/BBG - ECO DASH.xlsx` :
   - aba **BZ RATES** → limpa/renomeia colunas → salva `Dados/df_inicial.parquet`
   - aba **DIV01** → transpõe e renomeia → salva `Dados/df_divone.parquet`
4. Lê `Dados/FechamentoNTNBs.xlsx` :
   - ajusta formato wide
   - concatena com `Dados/df_preco_de_ajuste_atual_completo.parquet`
   - ordena colunas por data e aplica `ffill(axis=1)`
   - salva de volta `Dados/df_preco_de_ajuste_atual_completo.parquet`
5. Lê planilha interna de LFT (caminho `Z:\...dashboard LFT.xlsx`):

- extrai coluna `BLFT 0 06/01/30`
- salva `Dados/dados_lft.csv`

5.3.3) Pontos de atenção

- Depende do layout do Excel Bloomberg (`BBG - ECO DASH.xlsx`) estar conforme esperado (abas/colunas)
- Depende de acesso ao caminho de rede `Z:\...` para LFT
- Converte ponto↔vírgula (texto pt-BR) em alguns trechos; se o app precisar de float, é necessário reconverter

6) Como adicionar/remover fundos

6.1) Onde um fundo "existe"

Você precisa alinhar fundo em três camadas:

1. **PL histórico (Scraper AF)**

- `ScrapAF3.py` : adicionar fundo em `fundos_base`
- Revisar:
  - `INDICES_TOTAL` (quais fundos entram no TOTAL)
  - `IGNORE_FUND_DUPES` (se o fundo deve ser ignorado no check de duplicados)

1. **UI e pesos (app4.py)**

- Adicionar no dicionário/lista de pesos exibida no sidebar
- Garantir que cálculos que dependem de fundos não estejam hardcoded por posição

1. **Histórico por fundo (BaseFundos)**

- Se o fundo participa do histórico/simulação:
  - Garantir que `BaseFundos/<FUNDO>.parquet` exista (ou será criado)
  - Garantir que o app "enxerga" esse arquivo no carregamento

6.2) Checklist para adicionar fundo

- Incluir no `fundos_base` em `ScrapAF3.py`
- Revisar `INDICES_TOTAL` e `IGNORE_FUND_DUPES`
- Garantir que o parquet de PL (em `Dados/`) mantém o fundo como linha
- Incluir fundo nos pesos do `app4.py`
- Criar/validar `BaseFundos/<FUNDO>.parquet` se necessário

6.3) Checklist para remover fundo

- Remover do `fundos_base` (ou manter e zerar se quiser histórico)
- Revisar `INDICES_TOTAL`
- Remover dos pesos no `app4.py`
- Arquivar/remover `BaseFundos/<FUNDO>.parquet` se não usado

7) Como adicionar/remover ativos

7.1) Onde um ativo "existe"

1. **Universo do app (`Dados/df_inicial.parquet`)**

- Se o ativo não existir como coluna no `df_inicial`, ele não aparece no multiselect

1. **DV01/DIV01 (`Dados/df_divone.parquet`)**

- Stress depende do ativo ter DV01 (ou regra de tratamento)

1. **Preços/ajustes (`Dados/df_preco_de_ajuste_atual_completo.parquet`)**

- Para histórico/precificação

1. **ScrapB3 (se derivativo B3)**

- Atualizar `mapear_asset()` e filtros de instrumentos se for necessário capturar um novo tipo

7.2) Checklist para adicionar ativo

- Atualizar `TransformarRetornosParquet.py` (mapeamento/colunas do `BBG - ECO DASH.xlsx`) para incluir o ativo no `df_inicial.parquet`
- Atualizar `df_divone.parquet` (colunas) para incluir DV01 do novo ativo
- Garantir presença do ativo no parquet de preços/ajustes (B3/NTNB etc.)
- Se for derivativo B3 novo: revisar `ScrapB3_v2.py` (parse + mapeamento)

7.3) Checklist para remover ativo

- Remover coluna do `df_inicial.parquet` (ou deixar, mas não usar)
- Remover coluna do `df_divone.parquet` (se necessário)
- Se estiver sendo excluído do output B3: adicionar em `ASSETS_EXCLUIR` no `ScrapB3_v2.py`

8) Manutenção e boas práticas

8.1) Segurança (recomendado)

- Nunca deixar credenciais hardcoded em `ScrapAF3.py`
- Nunca deixar `SUPABASE_KEY` sensível no `app4.py`

Preferir:

- variáveis de ambiente
- `st.secrets` no Streamlit
- `.env` (não commitado)

8.2) Evitar hardcode por índice

- `ScrapAF3.py`: `INDICES_TOTAL` depende da ordem da tabela/DF
- Ideal: calcular TOTAL por nome de fundo (lista de nomes) e não por índice

8.3) Logs e auditoria

- `ScrapB3_v2.py` já salva `atualizacao_b3_log.txt`
- Considere criar também log para `ScrapAF3.py` (datas atualizadas, fundos faltantes etc.)

8.4) Versionamento das bases

Parquets são ótimos, mas é importante:

- manter backups (ex.: por data)
- versionar "dicionários" (lista de fundos, colunas do universo)

9) Checklist de verificação (sanity check)

Após rodar todos os scripts:

- `Dados/pl_fundos_teste.parquet` atualizado com a(s) última(s) coluna(s) de data
- `Dados/df_preco_de_ajuste_atual_completo.parquet` atualizado (colunas recentes e ativos chave)
- `Dados/df_valor_ajuste_contrato.parquet` atualizado
- `Dados/df_inicial.parquet` existe e contém as colunas/ativos esperados
- `Dados/df_divone.parquet` existe e contém DV01 para os ativos do stress
- `Dados/dados_lft.csv` atualizado (se a simulação de cota depende dele)

Abrir o app e validar:

- login ok
- multiselect de ativos ok
- VaR/Stress não quebra por missing de colunas

10) Troubleshooting (problemas comuns)

10.1) "TOTAL" do PL está errado

- **Causa comum:** mudou ordem/quantidade de fundos e `INDICES_TOTAL` ficou desatualizado.
- **Solução:** atualizar `INDICES_TOTAL` ou (recomendado) refatorar para somar por nomes.

10.2) Fundo novo não aparece no histórico

- **Causa:** fundo não está em `fundos_base` no `ScrapAF3.py`.
- **Solução:** adicionar o fundo na lista `fundos_base` e rodar novamente.

10.3) Ativo do CSV de execuções "não existe no universo"

- **Causa:** o ativo não está como coluna no `Dados/df_inicial.parquet`.
- **Solução:** atualizar `TransformarRetornosParquet.py` / `BBG - ECO DASH.xlsx` para incluir o ativo no `df_inicial`.

10.4) `ScrapB3_v2.py` parou de parsear

- **Causa:** a B3 mudou a estrutura do CSV/colunas esperadas.
- **Solução:** ajustar o `parse_consolidated_trades()` (mapeamento de colunas) e validar com `debug_b3_csv/`.

10.5) Simulação de cota falha por LFT

- **Causa:** falta `Dados/dados_lft.csv` ou caminho `Z:\...` inacessível.
- **Solução:** rodar `TransformarRetornosParquet.py` em ambiente com acesso ao drive, ou mover a base LFT para um caminho local.

Rodar rápido (resumo) - No terminal

```
python ScrapAF3.py

python ScrapB3_v2.py

python TransformarRetornosParquet.py

streamlit run app4.py
```