

Risk Frontier para Carteiras de Crédito Consignado (INSS)

Estrutura metodológica, equações e script de referência

August 12, 2025

1 Visão geral

O objetivo é construir uma *fronteira eficiente* de risco-retorno para carteiras agregadas por grupos homogêneos de risco (GHRs), levando em conta:

- dinâmica de inadimplência via matriz de *roll-rates* por *buckets* (**current**, 30, 60, 90, **wo**);
- perdas esperadas (EL), perdas inesperadas (UL) no modelo de *fator único* (Vasicek/IRB);
- ajuste de maturidade (IRB) para capital econômico;
- *carry cost* (funding, suspensão de accrual, opex de cobrança);
- normalização de retorno por **prazo médio remanescente** (modo anual/12m/EAA).

O script agrega métricas por GHR, sorteia carteiras sintéticas (*weights* Dirichlet), estima o par (EC, Retorno) e identifica a envoltória superior (*efficient frontier*). Classificamos GHRs como eficientes (acima/na fronteira) ou ineficientes (abaixo), tanto na visão total quanto em *sem safra* e *só safra*.

2 Fluxos financeiros e saldo (Price)

Para um contrato com valor liberado V , taxa mensal i e prazo N , a prestação (Price) é:

$$\text{PMT} = V \cdot \frac{i(1+i)^N}{(1+i)^N - 1}. \quad (1)$$

Após n parcelas, o saldo (EAD contábil) é:

$$\text{Saldo}(n) = V(1+i)^n - \text{PMT} \frac{(1+i)^n - 1}{i}. \quad (2)$$

No script, quando **Saldo_Atual** não é informado, o **Saldo** é calculado por essa fórmula. Para **Na Safra** aplicamos $n = 0$.

3 Dinâmica de atraso: roll-rates e PD condicional

Defina os *buckets* $\mathcal{B} = \{\text{current}, 30, 60, 90, \text{wo}\}$ e uma matriz de transição mensal $\mathbf{P} \in \mathbb{R}^{|\mathcal{B}| \times |\mathcal{B}|}$, com a última linha/coluna absorvente em **wo**. Seja o estado inicial \mathbf{s}_0 (vetor padrão de uma posição), e H o horizonte (meses). A probabilidade acumulada de absorção em **wo** até H (aproximação usada no código) é:

$$p_{\text{def}}(H) = \sum_{t=1}^H \mathbf{s}_{t-1} \mathbf{P} \mathbf{e}_{\text{wo}}, \quad \text{com} \quad \mathbf{s}_t = \mathbf{s}_{t-1} \mathbf{P}, \quad (3)$$

$$\text{e anulamos } s_t(\text{wo}) \text{ após contabilizar o fluxo ao absorvedor.} \quad (4)$$

Aqui, e_{wo} extrai a coluna de entrada em `wo`. No código, `pd_condicional_from_roll` implementa essa recursão de forma robusta.

A **LGD efetiva por bucket** é obtida de uma tabela $LGD(bucket)$ (com limites de piso/teto), refletindo severidade crescente com o atraso. O **EL forward** por contrato usa $p_{def}(H)$ e a LGD do bucket vigente.

4 Carry cost gerencial

Supondo taxa de funding k_f a.a. e spread contratado s a.a., as taxas mensais são $k_f^{(m)} = (1 + k_f)^{1/12} - 1$ e $s^{(m)} = (1 + s)^{1/12} - 1$. Se houver suspensão de accrual em atraso, a receita perdida é $s^{(m)} \cdot EAD$. Incorporamos ainda *opex* de cobrança por bucket. Para H meses:

$$\text{Carry}(H) = H \cdot \left(k_f^{(m)} \cdot EAD + \mathbf{1}_{\text{atraso}} \cdot s^{(m)} \cdot EAD + \text{OPEX}(\text{bucket}) \right). \quad (5)$$

5 Perdas esperadas e inesperadas (Vasicek/IRB)

Dados PD , LGD , EAD , correlação de ativo ρ , e nível de confiança α :

$$EL = PD \cdot LGD \cdot EAD, \quad (6)$$

$$UL = LGD \cdot EAD \cdot \Phi \left(\frac{\Phi^{-1}(PD) + \sqrt{\rho} z_\alpha}{\sqrt{1 - \rho}} \right), \quad (7)$$

onde Φ é a CDF normal padrão e $z_\alpha = \Phi^{-1}(\alpha)$. O **capital econômico de 1 ano** é $EC_{1y} = \max(UL - EL, 0)$. Ajuste de maturidade (IRB) aplicado:

$$b(PD) = \frac{0.11852 - 0.05478 \ln(PD)}{1 - 0.11852 - 0.05478 \ln(PD)}, \quad MA(M) = \frac{1 + (M - 2.5)b}{1 - 1.5b}, \quad EC = \max\{EC_{1y} \cdot MA(M), 0\}. \quad (8)$$

Obs.: a expressão de $b(PD)$ acima segue a implementação do script.

6 Normalização por prazo: EAA

Para comparar GHRs com prazos médios remanescentes distintos M (em anos), usamos um proxy de valor “lifetime” simples, $NPV_{\text{proxy}} \approx \text{Retorno Anual} \times M$, e convertemos para uma renda anual uniforme equivalente (EAA) a uma taxa K :

$$EAA(K, M) = NPV_{\text{proxy}} \cdot \frac{K}{1 - (1 + K)^{-M}}. \quad (9)$$

No código, a métrica de retorno usada na fronteira (`Ret_Metrica`) pode ser:

- **anual/12m**: usa o retorno anual gerencial (coerente com o horizonte de carry/ECL);
- **eaa**: usa EAA com taxa K (parâmetro `K_EAA`).

7 Construção da Fronteira Eficiente

Considere n GHRs. Defina:

$$\mathbf{w} \in \Delta^{n-1} \text{ (pesos Dirichlet)}, \quad (10)$$

$$\mathbf{e} = \text{EAD por GHR}, \quad \boldsymbol{\ell} = \text{LGD por GHR}, \quad \mathbf{r} = \text{Retorno por GHR (métrica)}, \quad \mathbf{el} = \text{EL por GHR}. \quad (11)$$

Aproximamos o desvio-padrão agregado (capital econômico proxy) como:

$$EC(\mathbf{w}) \approx \sqrt{\mathbf{1}^\top \left[(\boldsymbol{\ell} \odot (\mathbf{w} \odot \mathbf{e})) (\boldsymbol{\ell} \odot (\mathbf{w} \odot \mathbf{e}))^\top \odot \boldsymbol{\rho} \right] \mathbf{1}}, \quad (12)$$

onde $\boldsymbol{\rho}$ é a matriz de correlação (ajustada pelo mix de buckets) e \odot denota produto Hadamard. O retorno agregado é $R(\mathbf{w}) = \sum_i w_i r_i$, e na prática usamos $R(\mathbf{w}) - \sum_i w_i el_i$ (equivalente ao que o código já computa na nuvem sintética). Amostramos muitos \mathbf{w} , obtemos o conjunto (EC, R) e, por faixas de EC , tomamos o R máximo; a curva resultante é a **fronteira eficiente**.

8 Eficiência e ROE

Para cada GHR com par (EC_i, R_i) :

$$\text{Retorno máximo estimado na fronteira dado } EC_i \Rightarrow R_i^{\max} = \hat{R}(EC_i), \quad (13)$$

$$\text{Eficiência } \eta_i = \frac{R_i}{R_i^{\max}}, \quad (14)$$

$$\text{ROE ajustado } ROE_i = \frac{R_i}{EC_i}, \quad ROE_i^{\max} = \frac{R_i^{\max}}{EC_i}. \quad (15)$$

Classificamos como *eficiente* se $\eta_i \geq 1$ (no gráfico, marcadores verdes) e *ineficiente* se $\eta_i < 1$ (azul).

9 Entradas, saídas e visões

Entradas (planilha Excel):

- contratos, ghr_param, params (obrigatório o trio);
- opcionais: roll_rates, prov_reg, lgd_bucket, opex_cobranca, rho_mult_bucket.

Saídas:

- Excel: agregados por GHR (Total, Sem Safra, Só Safra), fronteiras e nuvens;
- PNGs: fronteira e ROE para as três visões.

Parâmetros-chave:

- Horizonte de carry/ECL: H_FWD_MESES (padrão 12);
- Modo de prazo: MODO_PRAZO {anual, 12m, eaa};
- Taxa EAA: K_EAA.

Referências (essenciais)

- Basel Committee on Banking Supervision (BCBS). *International Convergence of Capital Measurement and Capital Standards* (IRB), diversos documentos técnicos.
- Vasicek, O. (2002). *Loan Portfolio Value*. RISK.
- Princípios de roll-rates/Markov para análise de inadimplência em carteiras de crédito.

A Script Python completo

Abaixo, o script em Python utilizado neste relatório (*Delinquency-aware* + normalização por prazo e visões Total/Sem Safra/Só Safra). Ajuste os caminhos (INPUT_XLSX, OUTPUT_DIR) conforme seu ambiente.

```
1 # =====
2 # Risk Frontier      Consignado INSS (apenas contratos ATIVOS)
3 # Vers o "Delinquency-aware" + Normaliza o por Prazo (anual/12m/EAA
4 # Entradas obrigat rias: 'contratos', 'ghr_param', 'params'
5 # Opcionais: 'roll_rates', 'prov_reg', 'lgd_bucket', 'opex_cobranca', '
6 # Sa das: Excel consolidado + gr ficos PNG (Total, Sem Safra, S
7 # Requisitos: numpy, pandas, matplotlib, scipy, xlswriter
8 # =====
9
10 import numpy as np
11 import pandas as pd
12 import matplotlib.pyplot as plt
13 import matplotlib.patheffects as pe
14 from scipy.stats import norm
15 from scipy.interpolate import interp1d
16 from pathlib import Path
17
18 # ----- Config -----
19 INPUT_XLSX = r"C:\Users\Lenovo\Desktop\Desktop\Mestrado FGV\
20 RiskFrontier\exemplo_input_risk_frontier.xlsx"
21 OUTPUT_DIR = Path(r"C:\Users\Lenovo\Desktop\Desktop\Mestrado FGV\
22 RiskFrontier")
23 OUTPUT_DIR.mkdir(parents=True, exist_ok=True)
24 OUT_XLSX = OUTPUT_DIR / "risk_frontier_relatorio.xlsx"
25
26 SEED = 20250812
27 np.random.seed(SEED)
28 CONF_LEVEL = 0.999
29 Z_ALPHA = norm.ppf(CONF_LEVEL)
30
31 RHO_PADRAO = 0.12
32 PD_FLOOR, PD_CAP = 0.001, 0.30
33 LGD_FLOOR, LGD_CAP = 0.05, 0.50
34 ALPHA_OVERS, ALPHA_DELAY = 0.7, 0.3
35 FUNDING_AA_DEFAULT = 0.10
36 H_FWD_MESES_DEFAULT = 12
37
38 ESTADOS_ATIVOS = {"Em dia", "Em atraso", "Na Safra"}
39 BUCKETS = ["current", "30", "60", "90", "wo"] # tudo min sculo!
40
41 # ---- Normaliza o por prazo ----
42 MODO_PRAZO = "eaa" # "anual" | "12m" | "eaa"
43 K_EAA = 0.12 # taxa a.a. p/ EAA
44 M_PISO_ANOS = 0.25 # piso de prazo m dio
45
46 # ----- Fun es -----
47 def pd_dinamico(pd_base, overs, is_delayed, alpha_overs=ALPHA_OVERS,
48 alpha_delay=ALPHA_DELAY):
49     fator = 1 + alpha_overs * overs + alpha_delay * is_delayed
```

```

47     return float(np.clip(pd_base * fator, PD_FLOOR, PD_CAP))
48
49 def vasicek_ul(pd, lgd, ead, rho, z=Z_ALPHA):
50     inv_pd = norm.ppf(pd)
51     cond_pd = norm.cdf((inv_pd + np.sqrt(rho) * z) / np.sqrt(1 - rho))
52     return float(lgd * ead * cond_pd)
53
54 def basel_maturity_adjustment(pd, k_1y, m_years):
55     pd_clip = np.clip(pd, PD_FLOOR, 0.5)
56     b = (0.11852 - 0.05478 * np.log(pd_clip)) / (1 - 0.11852 - 0.05478
57         * np.log(pd_clip))
58     fator_m = (1 + (m_years - 2.5) * b) / (1 - 1.5 * b)
59     return float(max(k_1y * max(fator_m, 0.1), 0.0))
60
61 def pmt_price(valor, i_mensal, n_meses):
62     if i_mensal == 0:
63         return valor / max(n_meses, 1)
64     return valor * (i_mensal * (1 + i_mensal)**n_meses) / ((1 +
65         i_mensal)**n_meses - 1)
66
67 def saldo_price(valor, i_mensal, n_total, n_pag):
68     if i_mensal == 0:
69         return float(max(valor - (valor / max(n_total, 1)) * n_pag,
70             0.0))
71     pmt = pmt_price(valor, i_mensal, n_total)
72     saldo = valor * (1 + i_mensal)**n_pag - pmt * ((1 + i_mensal)**
73         n_pag - 1) / i_mensal
74     return float(max(saldo, 0.0))
75
76 def construir_frenteira(EAD, LGD, EL_1y, RetLiq_1y, rho_mat, n_points
77 =30000, seed=SEED):
78     rng = np.random.default_rng(seed)
79     n = len(EAD)
80     if n == 0:
81         return np.array([]), np.array([]), np.array([]), np.array([])
82     if n == 1:
83         ecs = np.array([np.sqrt((LGD[0]*EAD[0])**2 * rho_mat[0,0])])
84         rets = np.array([RetLiq_1y[0] - EL_1y[0]])
85         return ecs, rets, ecs, rets
86     weights = rng.dirichlet(np.ones(n), n_points)
87     ecs, rets = [], []
88     for w in weights:
89         ead_w = w * EAD
90         losses_scale = LGD * ead_w
91         cov = np.outer(losses_scale, losses_scale) * rho_mat
92         ec_total = np.sqrt(np.sum(cov))
93         el_total = np.sum(w * EL_1y)
94         ret_total = np.sum(w * (RetLiq_1y + EL_1y))
95         rets.append(ret_total - el_total)
96         ecs.append(ec_total)
97     ecs = np.array(ecs); rets = np.array(rets)
98     grid = np.linspace(ecs.min(), ecs.max(), 180)
99     fr_ec, fr_ret = [], []
100     tol = (ecs.max() - ecs.min()) / 400
101     for g in grid:
102         mask = (ecs >= g - tol) & (ecs <= g + tol)
103         if np.any(mask):
104             i = np.argmax(rets[mask])

```

```

100         fr_ec.append(ecs[mask][i]); fr_ret.append(rets[mask][i])
101     return np.array(fr_ec), np.array(fr_ret), ecs, rets
102
103 # ---- Defaults de delinquency ----
104 ROLL_DEFAULT = pd.DataFrame({
105     "current": [0.92, 0.06, 0.01, 0.00, 0.01],
106     "30"      : [0.35, 0.45, 0.15, 0.02, 0.03],
107     "60"      : [0.10, 0.25, 0.45, 0.15, 0.05],
108     "90"      : [0.02, 0.05, 0.28, 0.45, 0.20],
109     "wo"      : [0.00, 0.00, 0.00, 0.00, 1.00],
110 }, index=BUCKETS)
111 PROV_REG_DEFAULT = pd.DataFrame({"bucket": BUCKETS, "pct"
112     : [0.02, 0.10, 0.30, 0.50, 1.00]})
112 LGD_BUCKET_DEFAULT = pd.DataFrame({"bucket": BUCKETS, "lgd"
113     : [0.25, 0.30, 0.35, 0.40, 0.45]})
113 OPEX_DEFAULT = pd.DataFrame({"bucket": BUCKETS, "opex"
114     : [0.0, 15.0, 22.0, 35.0, 80.0]})
114 RHO_MULT_DEFAULT = pd.DataFrame({"bucket": BUCKETS, "rho_mult"
115     : [1.00, 1.05, 1.10, 1.15, 1.20]})
115
116 # ---- Helpers ----
117 def _lower_cols(df): df.columns = df.columns.str.strip().str.lower();
118     return df
118 def _lower_index(df): df.index = df.index.map(lambda x: str(x).strip().
119     lower()); return df
119
120 def ler_tabela_ou_default(xls, sheet, default_df):
121     try:
122         df = pd.read_excel(xls, sheet)
123         return _lower_cols(df)
124     except Exception:
125         return default_df.copy()
126
127 def load_roll_rates_strict(xls, sheet_name, default_df, buckets):
128     try:
129         df = pd.read_excel(xls, sheet_name)
130         df = _lower_cols(df)
131     except Exception:
132         df = default_df.copy()
133     df = df.loc[:, ~df.columns.astype(str).str.contains("^unnamed",
134         case=False, regex=True)]
134     cand_idx = None
135     for c in df.columns:
136         if str(c).strip().lower() in ["bucket", "bkt", "estado", "faixa", "
137             bucket_atraso"]:
138             cand_idx = c; break
137     if cand_idx is not None:
138         df = df.set_index(cand_idx)
139     df = _lower_index(df)
140     if set(buckets).issubset(set(df.columns)):
141         df = df[buckets]
142     elif set(buckets).issubset(set(df.index)):
143         df = df.loc[buckets].T
144     df = df.reindex(index=buckets, columns=buckets)
145     df = df.apply(pd.to_numeric, errors="coerce").fillna(0.0)
146     for i in df.index:
147         if i == "wo":
148             df.loc[i] = 0.0; df.loc[i, "wo"] = 1.0
149

```

```

150         else:
151             s = df.loc[i].sum()
152             df.loc[i] = (df.loc[i] / s) if s > 0 else 0.0
153             if s == 0: df.loc[i, i] = 1.0
154         return df
155
156     def pd_condicional_from_roll(start_bucket, horizon_m, roll_df):
157         idx = {b:i for i,b in enumerate(roll_df.index)}
158         state = np.zeros(len(roll_df)); state[idx[start_bucket]] = 1.0
159         p_default = 0.0
160         P = roll_df.values
161         wo_col = roll_df.columns.get_loc("wo")
162         for _ in range(horizon_m):
163             p_default += state @ P[:, wo_col]
164             state = state @ P
165             state[idx["wo"]] = 0.0
166         return float(np.clip(p_default, 0.0, 1.0))
167
168     def expected_carry_cost(EAD, bucket, funding_aa, spread_aa, months,
169                             opex_map, suspende_accrual=True):
170         funding_am = (1 + funding_aa)**(1/12) - 1
171         spread_am = (1 + spread_aa)**(1/12) - 1
172         lost_rev = spread_am * EAD if suspende_accrual else 0.0
173         carry_m = funding_am * EAD + lost_rev + opex_map.get(bucket,
174                                                                0.0)
175         return carry_m * months
176
177     # ----- 1) Ler entrada -----
178     xls = pd.ExcelFile(INPUT_XLSX)
179     contratos = pd.read_excel(xls, "contratos")
180     ghr_param = pd.read_excel(xls, "ghr_param")
181     try:
182         params = pd.read_excel(xls, "params")
183     except Exception:
184         params = None
185
186     # opcionais
187     roll_df = load_roll_rates_strict(xls, "roll_rates", ROLL_DEFAULT,
188                                     BUCKETS)
189     prov_reg_df = ler_tabela_ou_default(xls, "prov_reg", PROV_REG_DEFAULT)
190     lgd_bkt_df = ler_tabela_ou_default(xls, "lgd_bucket",
191                                       LGD_BUCKET_DEFAULT)
192     opex_df = ler_tabela_ou_default(xls, "opex_cobranca", OPEX_DEFAULT)
193     rho_mult_df = ler_tabela_ou_default(xls, "rho_mult_bucket",
194                                         RHO_MULT_DEFAULT)
195
196     prov_reg_df = _lower_cols(prov_reg_df); lgd_bkt_df = _lower_cols(
197         lgd_bkt_df)
198     opex_df = _lower_cols(opex_df); rho_mult_df = _lower_cols(
199         rho_mult_df)
200     prov_map = dict(zip(prov_reg_df["bucket"], prov_reg_df["pct"]))
201     lgd_map = dict(zip(lgd_bkt_df["bucket"], lgd_bkt_df["lgd"]))
202     opex_map = dict(zip(opex_df["bucket"], opex_df["opex"]))
203     rho_mult = dict(zip(rho_mult_df["bucket"], rho_mult_df["rho_mult"]))
204
205     funding_aa = FUNDING_AA_DEFAULT
206     H_FWD_MESES = H_FWD_MESES_DEFAULT
207     if params is not None and {"Parametro", "Valor"}.issubset(params.columns

```

```

):
201     mparams = params.set_index("Parametro")["Valor"].to_dict()
202     try: funding_aa = float(mparams.get("Funding_aa",
        FUNDING_AA_DEFAULT))
203     except: pass
204     try: H_FWD_MESES = int(mparams.get("H_fwd_mesess",
        H_FWD_MESES_DEFAULT))
205     except: pass
206     if "Modo_prazo" in mparams:
207         MODO_PRAZO = str(mparams["Modo_prazo"]).strip().lower()
208     if "K_EAA" in mparams:
209         try: K_EAA = float(mparams["K_EAA"])
210         except: pass
211
212 # ----- PATCH Spread_aa -----
213 def _norm_cols_keepcase(df):
214     df.columns = df.columns.str.strip()
215     return df
216 contratos = _norm_cols_keepcase(contratos)
217 ghr_param = _norm_cols_keepcase(ghr_param)
218 alt_names = {"spread": "Spread_aa", "Spread aa": "Spread_aa", "spread_aa": "
    Spread_aa", "SPREAD_AA": "Spread_aa"}
219 contratos.rename(columns={k:v for k,v in alt_names.items() if k in
    contratos.columns}, inplace=True)
220 ghr_param.rename(columns={k:v for k,v in alt_names.items() if k in
    ghr_param.columns}, inplace=True)
221
222 if "GHR" not in contratos.columns or "GHR" not in ghr_param.columns:
223     raise ValueError("A coluna 'GHR' deve existir em 'contratos' e '
        ghr_param'.")
224 contratos["GHR"] = contratos["GHR"].astype(str).str.strip()
225 ghr_param["GHR"] = ghr_param["GHR"].astype(str).str.strip()
226
227 need_cols = {"PD_Base", "LGD_Base", "Spread_aa"}
228 if not need_cols.issubset(contratos.columns):
229     cols_to_bring = ["GHR"] + [c for c in ["PD_Base", "LGD_Base", "
        Spread_aa"] if c in ghr_param.columns]
230     contratos = contratos.merge(ghr_param[cols_to_bring], on="GHR", how
        ="left", validate="many_to_one")
231
232 if "Spread_aa" not in contratos.columns or contratos["Spread_aa"].isna
    ().all():
233     if "Spread_aa" in ghr_param.columns and not ghr_param["Spread_aa"].
        isna().all():
234         contratos["Spread_aa"] = contratos["GHR"].map(dict(zip(
            ghr_param["GHR"], ghr_param["Spread_aa"])))
235     if "Spread_aa" not in contratos.columns:
236         contratos["Spread_aa"] = np.nan
237     if contratos["Spread_aa"].isna().any():
238         ghrs = sorted(contratos["GHR"].unique())
239         defaults = [0.14, 0.16, 0.18, 0.20, 0.22]
240         while len(defaults) < len(ghrs): defaults.append(defaults[-1])
241         contratos["Spread_aa"] = contratos["Spread_aa"].fillna(
            contratos["GHR"].map({g:s for g,s in zip(ghrs, defaults)}))
242         contratos["Spread_aa"] = contratos["Spread_aa"].fillna(0.18)
243 if contratos["Spread_aa"].isna().any():
244     raise ValueError("N o foi poss vel determinar 'Spread_aa' para
        alguns contratos.")

```



```

245
246 # ----- 2) Pr -processamento -----
247 contratos = contratos[contratos["Estado"].isin(ESTADOS_ATIVOS)].copy()
248 req_cols = {"ContratoID", "GHR", "Estado", "Valor_Liberado", "Prazo_Meses",
249             "Meses_Pagos"}
249 missing = req_cols - set(contratos.columns)
250 if missing:
251     raise ValueError(f"Faltam colunas em 'contratos': {missing}")
252
253 contratos["Meses_Pagos"] = contratos.apply(
254     lambda r: 0 if r["Estado"]=="Na Safra" else min(max(int(r["
255     Meses_Pagos"]), 0), int(r["Prazo_Meses"])-1), axis=1
256 )
257 contratos["Meses_Restantes"] = contratos["Prazo_Meses"] - contratos["
258     Meses_Pagos"]
259 contratos = contratos[contratos["Meses_Restantes"] > 0].copy()
260
261 contratos["Juros_aa"] = funding_aa + contratos["Spread_aa"]
262 contratos["Juros_am"] = (1 + contratos["Juros_aa"])*(1/12) - 1
263
264 if "Saldo_Atual" in contratos.columns:
265     contratos["Saldo"] = contratos["Saldo_Atual"].astype(float)
266 else:
267     contratos["Saldo"] = [
268         saldo_price(v, i, n, p)
269         for v, i, n, p in zip(contratos["Valor_Liberado"], contratos["
270         Juros_am"], contratos["Prazo_Meses"], contratos["Meses_Pagos
271         "])
272     ]
273
274 def infer_bucket(row):
275     if "Bucket_Atraso" in contratos.columns and pd.notnull(row.get("
276     Bucket_Atraso")):
277         b = str(row["Bucket_Atraso"]).strip().lower()
278         return "current" if b in ["0", "cur", "current"] else b
279     if "DPD" in contratos.columns and pd.notnull(row.get("DPD")):
280         d = int(row["DPD"])
281         if d <= 0: return "current"
282         if d <= 30: return "30"
283         if d <= 60: return "60"
284         if d <= 90: return "90"
285         return "wo"
286     if row["Estado"] in ["Em dia", "Na Safra"]: return "current"
287     return "30"
288
289 contratos["bucket"] = contratos.apply(infer_bucket, axis=1)
290
291 if "Overs_Flag" not in contratos.columns or "IsDelayed_Flag" not in
292     contratos.columns:
293     rng = np.random.default_rng(SEED)
294     overs_prob = np.where(
295         contratos["Estado"].eq("Em atraso"), rng.uniform(0.20, 0.50, len
296         (contratos)),
297         np.where(contratos["Estado"].eq("Na Safra"), rng.uniform
298         (0.05, 0.15, len(contratos)),
299         rng.uniform(0.00, 0.05, len(contratos)))
300     )
301     isdel_prob = np.where(
302         contratos["Estado"].eq("Em atraso"), rng.uniform(0.10, 0.30, len

```

```

294         (contratos)),
        np.where(contratos["Estado"].eq("Na Safra"), rng.uniform
        (0.05,0.20, len(contratos)),
295         rng.uniform(0.01,0.08, len(contratos)))
296     )
297     contratos["Overs_Flag"] = rng.random(len(contratos)) <
        overs_prob
298     contratos["IsDelayed_Flag"] = rng.random(len(contratos)) <
        isdel_prob
299
300 # ----- 3) Agregado por GHR -----
301 agg = []
302 for ghr, sub in contratos.groupby("GHR"):
303     ead = sub["Saldo"].sum()
304     vol = len(sub)
305     pd_base = float(sub["PD_Base"].iloc[0])
306     lgd_base = float(sub["LGD_Base"].iloc[0])
307     spread = float(sub["Spread_aa"].iloc[0])
308     overs = float(sub["Overs_Flag"].mean())
309     isdel = float(sub["IsDelayed_Flag"].mean())
310     m_rem_anos = float(np.average(sub["Meses_Restantes"], weights=sub["
        Saldo"]) / 12.0) if ead > 0 else 0.0
311     pd_adj = pd_dinamico(pd_base, overs, isdel)
312
313     el_mes_total = 0.0
314     lgd_eff_weighted = 0.0
315     rho_mult_w = 0.0
316     for _, r in sub.iterrows():
317         EADi = float(r["Saldo"]); bkt = str(r["bucket"]).strip().lower
            ()
318         prov_inc = float(prov_map.get(bkt, 0.0)) * EADi
319         H_i = int(min(H_FWD_MESES, r["Meses_Restantes"]))
320         pd_fwd = pd_condicional_from_roll(bkt, H_i, roll_df)
321         lgd_b = float(np.clip(lgd_map.get(bkt, lgd_base), LGD_FLOOR,
            LGD_CAP))
322         ecl_fwd = pd_fwd * lgd_b * EADi
323         susp = (bkt != "current")
324         carry = expected_carry_cost(EADi, bkt, funding_aa, spread,
            months=H_i, opex_map=opex_map, suspende_accrual=susp)
325         el_mes_total += (prov_inc + ecl_fwd + carry)
326         lgd_eff_weighted += lgd_b * EADi
327         rho_mult_w += rho_mult.get(bkt, 1.0) * EADi
328
329     lgd_eff = (lgd_eff_weighted / ead) if ead > 0 else lgd_base
330     rho_eff = (rho_mult_w / ead) * RHO_PADRAO if ead > 0 else
        RHO_PADRAO
331
332     el_1y = pd_adj * lgd_eff * ead
333     ul_1y = vasicek_ul(pd_adj, lgd_eff, ead, rho_eff, Z_ALPHA)
334     ec_1y = max(ul_1y - el_1y, 0.0)
335     ec_adj = basel_maturity_adjustment(pd_adj, ec_1y, m_rem_anos)
336
337     ret_liq_anual = spread * ead - el_mes_total
338
339     agg.append({
340         "GHR": ghr, "EAD": ead, "Volume_Contratos": vol,
341         "Overs": overs, "Is_Delayed": isdel, "M_Rem_anos": m_rem_anos,
342         "PD_Base": pd_base, "PD_Ajustado": pd_adj,

```

```

343         "LGD_Base": lgd_base, "LGD_Efetivo": lgd_eff,
344         "Spread": spread, "Rho_Efetivo": rho_eff,
345         "EL_gerencia_mes": el_mes_total,
346         "UL_1y": ul_1y, "EC_1y": ec_1y, "EC_Ajustado": ec_adj,
347         "Ret_Liq_Anual": ret_liq_anual
348     })
349 df_ghr = pd.DataFrame(agg).set_index("GHR")
350
351 # ----- 3.1) Métrica de retorno (prazo) -----
352 npv_proxy = df_ghr["Ret_Liq_Anual"] * df_ghr["M_Rem_anos"]
353 if MODDO_PRAZO.lower() in ["anual", "12m"]:
354     df_ghr["Ret_Metrica"] = df_ghr["Ret_Liq_Anual"]
355     Y_LABEL_RET = "Retorno Líquido Anual (R$ milhões)"
356 elif MODDO_PRAZO.lower() == "eaa":
357     anos = np.clip(df_ghr["M_Rem_anos"], M_PISO_ANOS, None)
358     fator_eaa = K_EAA / (1 - (1 + K_EAA)**(-anos))
359     df_ghr["Ret_Metrica"] = npv_proxy * fator_eaa
360     Y_LABEL_RET = "Retorno Normalizado (EAA, R$ milhões/ano)"
361 else:
362     raise ValueError("MODDO_PRAZO inválido. Use 'anual', '12m' ou 'eaa'")
363
364 # ----- 4) Fronteira e Eficiência -----
365 rho_mat = np.full((len(df_ghr), len(df_ghr)), RHO_PADRAO)
366 np.fill_diagonal(rho_mat, 1.0)
367 rho_scaler = np.clip(df_ghr["Rho_Efetivo"].mean() / RHO_PADRAO, 0.8,
368                     1.5)
369 rho_mat *= rho_scaler
370 np.fill_diagonal(rho_mat, 1.0)
371
372 fr_ec, fr_ret, ecs_cloud, rets_cloud = construir_fronteira(
373     EAD=df_ghr["EAD"].values,
374     LGD=df_ghr["LGD_Efetivo"].values,
375     EL_1y=(df_ghr["PD_Ajustado"]*df_ghr["LGD_Efetivo"]*df_ghr["EAD"]).
376         values,
377     RetLiq_1y=df_ghr["Ret_Metrica"].values,
378     rho_mat=rho_mat,
379     n_points=30000,
380     seed=SEED,
381 )
382 interp_ret_max = interp1d(fr_ec, fr_ret, bounds_error=False, fill_value
383                          ="extrapolate")
384 interp_roe_max = interp1d(fr_ec, (fr_ret / fr_ec), bounds_error=False,
385                          fill_value="extrapolate")
386
387 df_ghr["Ret_Fronteira_Ideal"] = interp_ret_max(df_ghr["EC_Ajustado"].
388         values)
389 df_ghr["Eficiencia"] = df_ghr["Ret_Metrica"] / df_ghr["
390     Ret_Fronteira_Ideal"]
391 df_ghr["ROE_Ajustado"] = df_ghr["Ret_Metrica"] / df_ghr["
392     EC_Ajustado"].replace(0, np.nan)
393 df_ghr["ROE_Max_Fronteira"] = interp_roe_max(df_ghr["EC_Ajustado"].
394         values)
395 df_ghr["Eficiencia_ROE"] = df_ghr["ROE_Ajustado"] / df_ghr["
396     ROE_Max_Fronteira"]
397
398 # ----- 5) Gráficos -----
399 mask_ok = df_ghr["Eficiencia"] >= 1.0

```

```

391 mask_bad = ~mask_ok
392
393 def _label_points(ax, xs, ys, labels, color):
394     for x, y, lab in zip(xs, ys, labels):
395         ax.text(x, y, str(lab),
396                 fontsize=9, ha="left", va="bottom", color=color,
397                 path_effects=[pe.withStroke(linewidth=2, foreground="
398                                     white"))])
399
400 # Fronteira
401 plt.figure(figsize=(12,7))
402 ax = plt.gca()
403 if len(ecs_cloud) > 0:
404     ax.scatter(ecs_cloud/1e6, rets_cloud/1e6, alpha=0.06, s=4, label="
405                 Carteiras Simuladas")
406 ax.plot(fr_ec/1e6, fr_ret/1e6, linewidth=2, label="Fronteira Eficiente"
407 )
408 ax.scatter(df_ghr.loc[mask_ok,"EC_Ajustado"]/1e6, df_ghr.loc[mask_ok,"
409                 Ret_Metrica"]/1e6,
410             s=80, marker='o', label="GHRs Eficientes ( 1 )", color="#2
411                 ca02c")
412 _label_points(ax, df_ghr.loc[mask_ok,"EC_Ajustado"]/1e6, df_ghr.loc[
413                 mask_ok,"Ret_Metrica"]/1e6,
414                 df_ghr.loc[mask_ok].index, "#2ca02c")
415 ax.scatter(df_ghr.loc[mask_bad,"EC_Ajustado"]/1e6, df_ghr.loc[mask_bad,
416                 "Ret_Metrica"]/1e6,
417             s=90, marker='x', label="GHRs Ineficientes (<1)", color="#1
418                 f77b4")
419 _label_points(ax, df_ghr.loc[mask_bad,"EC_Ajustado"]/1e6, df_ghr.loc[
420                 mask_bad,"Ret_Metrica"]/1e6,
421                 df_ghr.loc[mask_bad].index, "#1f77b4")
422 ax.set_xlabel("Capital Econ mico Ajustado (R$ milh es)")
423 ax.set_ylabel(Y_LABEL_RET)
424 ax.set_title("Consignado INSS      GHRs vs. Fronteira Eficiente")
425 ax.legend(); ax.grid(True); plt.tight_layout()
426 plt.savefig(OUTPUT_DIR / "fronteira.png", dpi=200)
427 plt.show()
428
429 # ROE
430 plt.figure(figsize=(12,7))
431 ax = plt.gca()
432 ax.plot(fr_ec/1e6, (fr_ret/fr_ec)*100, linewidth=2, label="ROE M ximo
433                 (Fronteira)")
434 ax.scatter(df_ghr["EC_Ajustado"]/1e6, (df_ghr["ROE_Ajustado"]*100),
435             s=80, marker='o', label="GHRs (ROE Ajustado)")
436 _label_points(ax, df_ghr["EC_Ajustado"]/1e6, (df_ghr["ROE_Ajustado"
437                 ]*100),
438                 df_ghr.index, "#333333")
439 ax.set_xlabel("Capital Econ mico Ajustado (R$ milh es)")
440 ax.set_ylabel("ROE Ajustado ao Risco (% a.a.)")
441 ax.set_title("Consignado INSS      ROE Ajustado vs. ROE M ximo")
442 ax.legend(); ax.grid(True); plt.tight_layout()
443 plt.savefig(OUTPUT_DIR / "roe_maximo.png", dpi=200)
444 plt.show()
445
446 # ===== Vis es extra: sem safra / s safra =====
447 def agrega_e_frenteira_subset(subset, label_subset, seed=SEED):
448     if subset.empty:

```

```

438     return {"label": label_subset, "df_ghr": pd.DataFrame(),
439            "fr_ec": np.array([]), "fr_ret": np.array([]),
440            "ecs_cloud": np.array([]), "rets_cloud": np.array([])}
441 agg_loc = []
442 for ghr, sub in subset.groupby("GHR"):
443     ead = sub["Saldo"].sum(); vol = len(sub)
444     pd_base = float(sub["PD_Base"].iloc[0]); lgd_base = float(sub[
445         "LGD_Base"].iloc[0])
446     spread = float(sub["Spread_aa"].iloc[0])
447     overs = float(sub["Overs_Flag"].mean()); isdel = float(sub["
448         IsDelayed_Flag"].mean())
449     m_rem_anos = float(np.average(sub["Meses_Restantes"], weights=
450         sub["Saldo"]) / 12.0) if ead > 0 else 0.0
451     pd_adj = pd_dinamico(pd_base, overs, isdel)
452     el_mes_total = 0.0; lgd_eff_weighted = 0.0; rho_mult_w = 0.0
453     for _, r in sub.iterrows():
454         EADi = float(r["Saldo"]); bkt = str(r["bucket"]).strip().
455             lower()
456         prov_inc = float(prov_map.get(bkt, 0.0)) * EADi
457         H_i = int(min(H_FWD_MESES, r["Meses_Restantes"]))
458         pd_fwd = pd_condicional_from_roll(bkt, H_i, roll_df)
459         lgd_b = float(np.clip(lgd_map.get(bkt, lgd_base), LGD_FLOOR
460             , LGD_CAP))
461         ecl_fwd = pd_fwd * lgd_b * EADi
462         susp = (bkt != "current")
463         carry = expected_carry_cost(EADi, bkt, funding_aa, spread,
464             months=H_i, opex_map=opex_map, suspende_accrual=susp)
465         el_mes_total += (prov_inc + ecl_fwd + carry)
466         lgd_eff_weighted += lgd_b * EADi
467         rho_mult_w += rho_mult.get(bkt, 1.0) * EADi
468     lgd_eff = (lgd_eff_weighted / ead) if ead > 0 else lgd_base
469     rho_eff = (rho_mult_w / ead) * RHO_PADRAO if ead > 0 else
470         RHO_PADRAO
471     el_1y = pd_adj * lgd_eff * ead; ul_1y = vasicek_ul(pd_adj,
472         lgd_eff, ead, rho_eff, Z_ALPHA)
473     ec_1y = max(ul_1y - el_1y, 0.0); ec_adj =
474         basel_maturity_adjustment(pd_adj, ec_1y, m_rem_anos)
475     ret_liq_anual = spread * ead - el_mes_total
476     agg_loc.append({
477         "GHR": ghr, "EAD": ead, "Volume_Contratos": vol,
478         "Overs": overs, "Is_Delayed": isdel, "M_Rem_anos":
479             m_rem_anos,
480         "PD_Base": pd_base, "PD_Ajustado": pd_adj,
481         "LGD_Base": lgd_base, "LGD_Efetivo": lgd_eff,
482         "Spread": spread, "Rho_Efetivo": rho_eff,
483         "EL_gerencia_mes": el_mes_total,
484         "UL_1y": ul_1y, "EC_1y": ec_1y, "EC_Ajustado": ec_adj,
485         "Ret_Liq_Anual": ret_liq_anual
486     })
487 df_loc = pd.DataFrame(agg_loc).set_index("GHR")
488 if df_loc.empty:
489     return {"label": label_subset, "df_ghr": df_loc,
490            "fr_ec": np.array([]), "fr_ret": np.array([]),
491            "ecs_cloud": np.array([]), "rets_cloud": np.array([])}
492 npv_proxy_loc = df_loc["Ret_Liq_Anual"] * df_loc["M_Rem_anos"]
493 if MODO_PRAZO.lower() in ["anual", "12m"]:
494     df_loc["Ret_Metrica"] = df_loc["Ret_Liq_Anual"]
495 elif MODO_PRAZO.lower() == "eaa":

```

```

486     anos = np.clip(df_loc["M_Rem_anos"], M_PISO_ANOS, None)
487     fator_eaa = K_EAA / (1 - (1 + K_EAA)**(-anos))
488     df_loc["Ret_Metrica"] = npv_proxy_loc * fator_eaa
489 else:
490     raise ValueError("MODO_PRAZO inv lido no subset.")
491 rho_mat_loc = np.full((len(df_loc), len(df_loc)), RHO_PADRAO); np.
    fill_diagonal(rho_mat_loc, 1.0)
492 rho_scaler_loc = np.clip(df_loc["Rho_Efetivo"].mean() / RHO_PADRAO,
    0.8, 1.5)
493 rho_mat_loc *= rho_scaler_loc; np.fill_diagonal(rho_mat_loc, 1.0)
494 fr_ec_loc, fr_ret_loc, ecs_cloud_loc, rets_cloud_loc =
    construir_frenteira(
495     EAD=df_loc["EAD"].values, LGD=df_loc["LGD_Efetivo"].values,
496     EL_1y=(df_loc["PD_Ajustado"]*df_loc["LGD_Efetivo"]*df_loc["EAD"
    ]).values,
497     RetLiq_1y=df_loc["Ret_Metrica"].values,
498     rho_mat=rho_mat_loc, n_points=30000, seed=seed,
499 )
500 if len(fr_ec_loc):
501     interp_ret_max_loc = interp1d(fr_ec_loc, fr_ret_loc,
    bounds_error=False, fill_value="extrapolate")
502     interp_roe_max_loc = interp1d(fr_ec_loc, (fr_ret_loc /
    fr_ec_loc), bounds_error=False, fill_value="extrapolate")
503     df_loc["Ret_Frenteira_Ideal"] = interp_ret_max_loc(df_loc["
    EC_Ajustado"].values)
504     df_loc["Efici ncia"] = df_loc["Ret_Metrica"] / df_loc[
    "Ret_Frenteira_Ideal"]
505     df_loc["ROE_Ajustado"] = df_loc["Ret_Metrica"] / df_loc[
    "EC_Ajustado"].replace(0, np.nan)
506     df_loc["ROE_Max_Frenteira"] = interp_roe_max_loc(df_loc["
    EC_Ajustado"].values)
507     df_loc["Efici ncia_ROE"] = df_loc["ROE_Ajustado"] /
    df_loc["ROE_Max_Frenteira"]
508 else:
509     for c in ["Ret_Frenteira_Ideal", "Efici ncia", "ROE_Ajustado", "
    ROE_Max_Frenteira", "Efici ncia_ROE"]:
510         df_loc[c] = np.nan
511 return {"label": label_subset, "df_ghr": df_loc,
512        "fr_ec": fr_ec_loc, "fr_ret": fr_ret_loc,
513        "ecs_cloud": ecs_cloud_loc, "rets_cloud": rets_cloud_loc}
514
515 def plot_subset(res, fname_suffix, y_label_ret):
516     df_loc = res["df_ghr"]; fr_ec_loc, fr_ret_loc = res["fr_ec"], res["
    fr_ret"]
517     ecs_cloud_loc, rets_cloud_loc = res["ecs_cloud"], res["rets_cloud"]
518     if df_loc.empty:
519         print(f"[{res['label']}] Subconjunto vazio; gr ficos n o
    gerados."); return
520     mask_ok = df_loc["Efici ncia"] >= 1.0; mask_bad = ~mask_ok
521     def _label_points(ax, xs, ys, labels, color):
522         for x, y, lab in zip(xs, ys, labels):
523             ax.text(x, y, str(lab), fontsize=9, ha="left", va="bottom",
    color=color,
524                     path_effects=[pe.withStroke(linewidth=2, foreground
    ="white")])
525     plt.figure(figsize=(12,7)); ax = plt.gca()
526     if len(ecs_cloud_loc) > 0:
527         ax.scatter(ecs_cloud_loc/1e6, rets_cloud_loc/1e6, alpha=0.06, s

```

```

        =4, label="Carteiras Simuladas")
528 if len(fr_ec_loc) > 0:
529     ax.plot(fr_ec_loc/1e6, fr_ret_loc/1e6, linewidth=2, label="
        Fronteira Eficiente")
530 ax.scatter(df_loc.loc[mask_ok,"EC_Ajustado"]/1e6, df_loc.loc[
        mask_ok,"Ret_Metrica"]/1e6,
531             s=80, marker='o', label="GHRs Eficientes ( 1 )", color=
        "#2ca02c")
532 _label_points(ax, df_loc.loc[mask_ok,"EC_Ajustado"]/1e6, df_loc.loc
        [mask_ok,"Ret_Metrica"]/1e6,
533                df_loc.loc[mask_ok].index, "#2ca02c")
534 ax.scatter(df_loc.loc[mask_bad,"EC_Ajustado"]/1e6, df_loc.loc[
        mask_bad,"Ret_Metrica"]/1e6,
535             s=90, marker='x', label="GHRs Ineficientes (<1)", color=
        "#1f77b4")
536 _label_points(ax, df_loc.loc[mask_bad,"EC_Ajustado"]/1e6, df_loc.
        loc[mask_bad,"Ret_Metrica"]/1e6,
537                df_loc.loc[mask_bad].index, "#1f77b4")
538 ax.set_xlabel("Capital Econ mico Ajustado (R$ milh es)"); ax.
        set_ylabel(y_label_ret)
539 ax.set_title(f"Risk Frontier {res['label']}"); ax.legend(); ax.
        grid(True); plt.tight_layout()
540 plt.savefig(OUTPUT_DIR / f"fronteira_{fname_suffix}.png", dpi=200);
        plt.show()
541 plt.figure(figsize=(12,7)); ax = plt.gca()
542 if len(fr_ec_loc) > 0:
543     ax.plot(fr_ec_loc/1e6, (fr_ret_loc/fr_ec_loc)*100, linewidth=2,
        label="ROE M ximo (Fronteira)")
544 ax.scatter(df_loc["EC_Ajustado"]/1e6, (df_loc["ROE_Ajustado"]*100),
545             s=80, marker='o', label="GHRs (ROE Ajustado)")
546 _label_points(ax, df_loc["EC_Ajustado"]/1e6, (df_loc["ROE_Ajustado"
        ]*100),
547                df_loc.index, "#333333")
548 ax.set_xlabel("Capital Econ mico Ajustado (R$ milh es)");
549 ax.set_ylabel("ROE Ajustado ao Risco (% a.a.)");
550 ax.set_title(f"ROE Ajustado vs ROE M ximo {res['label']}"); ax
        .legend(); ax.grid(True); plt.tight_layout()
551 plt.savefig(OUTPUT_DIR / f"roe_maximo_{fname_suffix}.png", dpi=200)
        ; plt.show()
552
553 contratos_sem_safra = contratos[contratos["Estado"] != "Na Safra"].copy
        ()
554 contratos_safra = contratos[contratos["Estado"] == "Na Safra"].copy
        ()
555 res_sem_safra = agrega_e_frenteira_subset(contratos_sem_safra, "
        Carteira (sem Safra)")
556 res_safra = agrega_e_frenteira_subset(contratos_safra, "Somente
        Safra")
557 plot_subset(res_sem_safra, "sem_safra", Y_LABEL_RET)
558 plot_subset(res_safra, "safra", Y_LABEL_RET)
559
560 # ----- 6) Salvar Excel -----
561 frenteira_df = pd.DataFrame({"EC": fr_ec, "Retorno": fr_ret})
562 if len(ecs_cloud):
563     sample_n = min(15000, len(ecs_cloud))
564     sample_idx = np.random.choice(len(ecs_cloud), size=sample_n,
        replace=False)
565     nuvem_df = pd.DataFrame({"EC": ecs_cloud[sample_idx], "Retorno":

```



```

        rets_cloud[sample_idx]))
566 else:
567     nuvem_df = pd.DataFrame(columns=["EC", "Retorno"])
568
569     frenteira_sem_safr_df = pd.DataFrame({"EC": res_sem_safr["fr_ec"], "
        Retorno": res_sem_safr["fr_ret"]})
570     frenteira_safr_df = pd.DataFrame({"EC": res_safr["fr_ec"], "
        Retorno": res_safr["fr_ret"]})
571     nuvem_sem_safr_df = (pd.DataFrame({"EC": res_sem_safr["ecs_cloud"], "
        Retorno": res_sem_safr["rets_cloud"]})
572         if len(res_sem_safr["ecs_cloud"]) else pd.
            DataFrame(columns=["EC", "Retorno"]))
573     nuvem_safr_df = (pd.DataFrame({"EC": res_safr["ecs_cloud"], "
        Retorno": res_safr["rets_cloud"]})
574         if len(res_safr["ecs_cloud"]) else pd.DataFrame(
            columns=["EC", "Retorno"]))
575
576     with pd.ExcelWriter(OUT_XLSX, engine="xlsxwriter") as writer:
577         contratos.to_excel(writer, sheet_name="contratos_ativos_pp", index=
            False)
578         df_ghr.reset_index().to_excel(writer, sheet_name="
            ghr_agregado_total", index=False)
579         frenteira_df.to_excel(writer, sheet_name="frenteira_total", index=
            False)
580         nuvem_df.to_excel(writer, sheet_name="nuvem_carteiras_total", index
            =False)
581
582         res_sem_safr["df_ghr"].reset_index().to_excel(writer, sheet_name="
            ghr_agregado_sem_safr", index=False)
583         frenteira_sem_safr_df.to_excel(writer, sheet_name="
            frenteira_sem_safr", index=False)
584         nuvem_sem_safr_df.to_excel(writer, sheet_name="nuvem_sem_safr",
            index=False)
585
586         res_safr["df_ghr"].reset_index().to_excel(writer, sheet_name="
            ghr_agregado_safr", index=False)
587         frenteira_safr_df.to_excel(writer, sheet_name="frenteira_safr",
            index=False)
588         nuvem_safr_df.to_excel(writer, sheet_name="nuvem_safr", index=
            False)
589
590         roll_df.to_excel(writer, sheet_name="roll_rates_usado")
591         pd.DataFrame(list(prov_map.items()), columns=["bucket", "pct"]).
            to_excel(writer, sheet_name="prov_reg_usado", index=False)
592         pd.DataFrame(list(lgd_map.items()), columns=["bucket", "lgd"]).
            to_excel(writer, sheet_name="lgd_bucket_usado", index=False)
593         pd.DataFrame(list(opex_map.items()), columns=["bucket", "opex"]).
            to_excel(writer, sheet_name="opex_usado", index=False)
594         pd.DataFrame(list(rho_mult.items()), columns=["bucket", "rho_mult"])
            .to_excel(writer, sheet_name="rho_mult_usado", index=False)
595
596     print(f"\nRelat rio salvo em: {OUT_XLSX.resolve()}")
597     print(f"PNGs em: {OUTPUT_DIR.resolve()} (inclui *_sem_safr e *_safr)"
        )

```

Listing 1: riskfrontier.py