ETAPA 2a — Preparação do Painel CAGED + ILO Exposure Index

Dissertação: Inteligência Artificial Generativa e o Mercado de Trabalho Brasileiro: Uma Análise de Exposição Ocupacional e seus Efeitos Distributivos.

Aluno: Manoel Brasil Orlandi

0.1 Contextualização

A rápida difusão de modelos de IA generativa (LLMs, geradores de imagem/código) levanta questões centrais sobre seus impactos no mercado de trabalho. Para mensurar esse potencial de impacto, a Organização Internacional do Trabalho (OIT) criou um índice de exposição ocupacional à IA generativa, publicado como *Working Paper* 140 (WP140). O índice atribui scores de exposição a cada ocupação da classificação ISCO-08, com base na avaliação de suas tarefas constituintes por modelos de linguagem e validação humana.

Este notebook prepara uma base de dados que junta os dados do **Novo CAGED** (Cadastro Geral de Empregados e Desempregados) ao **índice de exposição à IA generativa da OIT**, para ser usado em um modelo de Diferenças-em-Diferenças (DiD) no Notebook 2b.

0.2 Objetivo

Construir o **painel mensal de ocupações formais brasileiras (2021–2025)** a partir do Novo CAGED, realizar o **crosswalk CBO 2002 → ISCO-08** (especificação dual: 2 dígitos como principal, 4 dígitos para robustez), e fazer o merge com o índice de exposição à IA generativa da OIT (Gmyrek, Berg & Cappelli, 2025). O output final é um dataset analítico pronto para a estimação DiD.

Estratégia de crosswalk: Análise principal a **2 dígitos** ISCO-08 (match por Sub-major Group com fallback hierárquico a Major Group), com robustez a **4 dígitos** via correspondência ISCO-88 ↔ ISCO-08 + fallback hierárquico em 6 níveis.

Inspiração metodológica: Hui, Reshef & Zhou (2024), “The Short-Term Effects of Generative Artificial Intelligence on Employment: Evidence from an Online Labor Market” — adaptado para dados administrativos brasileiros (CAGED) com o índice ILO de exposição ocupacional.

0.3 Ficha Técnica dos Dados

Item	Descrição
Fonte CAGED	Ministério do Trabalho e Emprego (MTE), via Base dos Dados (BigQuery)
Dataset BigQuery	`basedosdados.br_me_caged.microdados_movimentacao`
Período	Janeiro/2021 — Dezembro/2025 (60 meses)
Unidade	Movimentação individual (admissão ou desligamento)

Item	Descrição
Cobertura	Emprego formal (CLT) em todo o Brasil
Índice ILO	`ilo_exposure_clean.csv` — 427 ocupações ISCO-08 com exposure scores
Classificação	CBO 2002 (CAGED) → ISCO-08 (ILO) via crosswalk hierárquico

0.4 Referências principais

- Gmyrek, P., Berg, J. & Cappelli, D. (2025). *Generative AI and Jobs: An updated global assessment*. ILO Working Paper 140.
- Hui, X., Reshef, O. & Zhou, L. (2024). *The Short-Term Effects of Generative AI on Employment*. Organization Science, 35(6).
- Brynjolfsson, E., Chandar, P. & Chen, J. (2025). *Canaries in the Coal Mine? Six Facts about the Recent Employment Effects of AI*.
- Callaway, B. & Sant'Anna, P. (2021). *Difference-in-differences with multiple time periods*. Journal of Econometrics, 225(2).
- Muendler, M.-A. & Poole, J.P. (2004). *Job Concordances for Brazil: Mapping CBO to ISCO-88*. UC San Diego.

0.5 1. Configuração do ambiente

Definir caminhos, importar bibliotecas e configurar parâmetros do painel. Todos os caminhos são relativos ao diretório `notebook/`.

> **Nota sobre a janela temporal:** Excluímos 2020 para evitar os efeitos distorcivos da pandemia de COVID-19 sobre o mercado de trabalho formal. O ano de 2020 apresentou quedas e recuperações atípicas que contaminariam o período pré-tratamento do DiD. A janela Jan/2021–Dez/2025 oferece 23 meses pré-ChatGPT e 31 meses pós.

> **Nota sobre o Novo CAGED:** A partir de janeiro/2020, o CAGED foi substituído pelo sistema eSocial (Portaria SEPRT 1.127/2019). Usamos dados de 2021+ para consistência metodológica (eSocial já estabilizado).

```
# Verificar dependências e instalar apenas o que faltar (rode esta célula primeiro)
import importlib.util
import subprocess
import sys

# (nome para import, nome para pip install)
PACOTES = [
    ("pandas", "pandas"),
    ("numpy", "numpy"),
    ("pyarrow", "pyarrow"),
    ("matplotlib", "matplotlib"),
    ("seaborn", "seaborn"),
```

```
    ("scipy", "scipy"),
    ("pyfixest", "pyfixest"),
    ("bcb", "python-bcb"),  # IPCA Banco Central; import: from bcb import sgs
    ("xlrd", "xlrd"),  # Leitura de .xls (Crosswalk SOC/ISCO, Estrutura COD) — Anexo 1
    ("openpyxl", "openpyxl"),  # Leitura de .xlsx (Crosswalk SOC 2010 a 2018) — Anexo
    ("google.cloud.bigquery", "google-cloud-bigquery"),
]

def ja_instalado(nome_import):
    return importlib.util.find_spec(nome_import) is not None

faltando = [pip for imp, pip in PACOTES if not ja_instalado(imp)]
if faltando:
    subprocess.check_call([sys.executable, "-m", "pip", "install", "-q"] + faltando)
    print("Instalado:", ", ".join(faltando))
else:
    print("Todas as dependências já estão instaladas.")
```

```
Instalado: xlrd

[notice] A new release of pip is available: 24.2 -> 26.0.1

[notice] To update, run: python3.10 -m pip install --upgrade pip
```

```
# Etapa 2a.1 — Configuração do ambiente

import warnings
import pandas as pd
import numpy as np
from pathlib import Path

warnings.filterwarnings("ignore", category=FutureWarning)

# ------------------------------------------------------------------------------
# Caminhos (relativos ao diretório do notebook)
# ------------------------------------------------------------------------------
DATA_INPUT     = Path("data/input")
DATA_RAW       = Path("data/raw")
DATA_PROCESSED = Path("data/processed")
DATA_OUTPUT    = Path("data/output")

for d in [DATA_INPUT, DATA_RAW, DATA_PROCESSED, DATA_OUTPUT]:
    d.mkdir(parents=True, exist_ok=True)

# ------------------------------------------------------------------------------
# Parâmetros do Painel CAGED
# ------------------------------------------------------------------------------
GCP_PROJECT_ID = "mestrado-pnad-2026"

ANO_INICIO     = 2021
ANO_FIM        = 2025
ANO_TRATAMENTO = 2022
MES_TRATAMENTO = 12   # Dezembro/2022 como primeiro mês "pós"
```

```
SALARIO_MINIMO = {
    2021: 1100, 2022: 1212, 2023: 1320, 2024: 1412, 2025: 1518
}

# ------------------------------------------------------------------------
# Colunas a selecionar do CAGED (BigQuery)
# ------------------------------------------------------------------------
COLUNAS_CAGED = """
    ano, mes, sigla_uf, id_municipio, cbo_2002,
    categoria, tipo_movimentacao, saldo_movimentacao,
    salario_mensal, grau_instrucao, idade, sexo, raca_cor,
    cnae_2_secao, cnae_2_subclasse, tamanho_estabelecimento_janeiro
"""

# ------------------------------------------------------------------------
# Arquivos de referência
# ------------------------------------------------------------------------
ILO_FILE               = DATA_PROCESSED / "ilo_exposure_clean.csv"
ISCO_08_88_FILE        = DATA_INPUT / "Correspondência ISCO 08 a 88.xlsx"
ISCO_08_ESTRUTURA_FILE = DATA_INPUT / "ISCO 08 Estruturas e Definições.xlsx"
MUENDLER_FILE          = DATA_INPUT / "cbo-isco-conc.csv"

# ------------------------------------------------------------------------
# Checkpoints intermediários
# ------------------------------------------------------------------------
PAINEL_MENSAL_FILE     = DATA_PROCESSED / "painel_caged_mensal.parquet"
PAINEL_CROSSWALK_FILE  = DATA_PROCESSED / "painel_caged_crosswalk.parquet"
PAINEL_TRATAMENTO_FILE = DATA_PROCESSED / "painel_caged_tratamento.parquet"
PAINEL_FINAL_PARQUET   = DATA_OUTPUT / "painel_caged_did_ready.parquet"
PAINEL_FINAL_CSV       = DATA_OUTPUT / "painel_caged_did_ready.csv"

# ------------------------------------------------------------------------
# Cache: quando False, arquivos são deletados antes de (re)gerar (força reprocessament
# ------------------------------------------------------------------------
KEEP_CAGED_RAW         = True   # data/raw/caged_*.parquet
KEEP_PANEL_MENSAL      = True   # painel_caged_mensal.parquet
KEEP_PANEL_CROSSWALK   = True   # painel_caged_crosswalk.parquet
KEEP_PANEL_TRATAMENTO  = False   # painel_caged_tratamento.parquet
KEEP_PANEL_FINAL       = False   # painel_caged_did_ready.parquet/csv
KEEP_ANTHROPIC_INDEX   = True    # anthropic_automation_augmentation_cbo.parquet (Anex

# Cache do índice Anthropic (Automation vs Augmentation) — Anexo 1
ANTHROPIC_INDEX_CACHE  = DATA_PROCESSED / "anthropic_automation_augmentation_cbo.parqu

# ------------------------------------------------------------------------
# Deflator (salário real): índice base para IPCA
# ------------------------------------------------------------------------
INDICE_BASE_ANO = 2024
INDICE_BASE_MES = 12   # Dez/2024 = 100
INDICE_BASE     = 100.0
IPCA_MENSAL_FILE = DATA_PROCESSED / "ipca_mensal.parquet"

# ------------------------------------------------------------------------
# Setor tecnológico (CNAE 2.0 seção): J = Informação e comunicação; M = Atividades pro
```

```
# ----------------------------------------------------------------------------
CNAE_SECOES_TECNOLOGICO = ['J']   # ou ['J', 'M'] para incluir atividades profissionai
SETOR_TECNOLOGICO_LIMIAR = 0.5    # setor_tecnologico=1 quando pct_tecnologico_adm >=

# ----------------------------------------------------------------------------
# Grandes grupos CBO (para sanity checks)
# ----------------------------------------------------------------------------
GRANDES_GRUPOS_CBO = {
    '0': 'Forças Armadas',
    '1': 'Dirigentes',
    '2': 'Profissionais das ciências',
    '3': 'Técnicos nível médio',
    '4': 'Trabalhadores de serv. admin.',
    '5': 'Trabalhadores de serviços/comércio',
    '6': 'Agropecuária',
    '7': 'Produção industrial',
    '8': 'Operadores de máquinas',
    '9': 'Manutenção e reparação',
}

print("Configuração carregada.")
print(f"  Período: {ANO_INICIO}–{ANO_FIM} ({(ANO_FIM - ANO_INICIO + 1) * 12} meses)")
print(f"  Evento: ChatGPT — Nov/2022 (pós a partir de {MES_TRATAMENTO}/{ANO_TRATAMENTO
print(f"  Projeto GCP: {GCP_PROJECT_ID}")
print(f"  ILO file: {ILO_FILE} (existe: {ILO_FILE.exists()})")
```

```
Configuração carregada.
  Período: 2021–2025 (60 meses)
  Evento: ChatGPT — Nov/2022 (pós a partir de 12/2022)
  Projeto GCP: mestrado-pnad-2026
  ILO file: data/processed/ilo_exposure_clean.csv (existe: True)
```

0.6 2. Índice IPCA (deflator para salário real)

Carregar série mensal do IPCA (Banco Central SGS) e construir índice com base Dez/2024 = 100, para deflacionar salários nominais no enriquecimento do painel.

```
# Etapa 2a.2 — Índice IPCA mensal (base Dez/2024 = 100)
# Fonte: Banco Central SGS (série 433 = variação mensal IPCA). Índice construído por a

if IPCA_MENSAL_FILE.exists():
    df_ipca = pd.read_parquet(IPCA_MENSAL_FILE)
    print(f"Índice IPCA carregado do cache: {IPCA_MENSAL_FILE.name}")
    print(f"  Período: {df_ipca['ano'].min():.0f}–{df_ipca['mes'].min():.0f} a {df_ipc
else:
    from bcb import sgs
    # Série 433 = IPCA - Variação mensal (%)
    start = f"{ANO_INICIO}-01-01"
    end = f"{ANO_FIM}-12-31"
    raw = sgs.get({"IPCA_var": 433}, start=start, end=end)
    raw = raw.reset_index()
    raw.columns = ["data", "variacao"]
    raw["ano"] = raw["data"].dt.year
```

```
    raw["mes"] = raw["data"].dt.month
    # Ordenar por tempo e construir índice: base = 100 no mês INDICE_BASE_ANO/INDICE_B
    raw = raw.sort_values(["ano", "mes"]).reset_index(drop=True)
    raw["fator"] = 1 + raw["variacao"] / 100.0
    # Índice em cadeia: 100 no mês base; para trás divide pelo fator do mês seguinte;
    base_idx = raw[(raw["ano"] == INDICE_BASE_ANO) & (raw["mes"] == INDICE_BASE_MES)].
    if len(base_idx) == 0:
        raise ValueError(f"Mês base {INDICE_BASE_ANO}/{INDICE_BASE_MES} não encontrado
    base_idx = base_idx[0]
    indice = np.ones(len(raw)) * np.nan
    indice[base_idx] = INDICE_BASE
    for i in range(base_idx - 1, -1, -1):
        indice[i] = indice[i + 1] / raw.loc[i + 1, "fator"]
    for i in range(base_idx + 1, len(raw)):
        indice[i] = indice[i - 1] * raw.loc[i, "fator"]
    raw["indice"] = indice
    df_ipca = raw[["ano", "mes", "indice"]].copy()
    df_ipca.to_parquet(IPCA_MENSAL_FILE, index=False)
    print(f"Índice IPCA construído e salvo: {IPCA_MENSAL_FILE.name}")
    print(f"  Período: {df_ipca['ano'].min():.0f}-{df_ipca['mes'].min():.0f} a {df_ipc
```

```
Índice IPCA carregado do cache: ipca_mensal.parquet
  Período: 2021-1 a 2025-12, base 2024/12 = 100.0
```

0.7 2a. Download dos microdados CAGED

Extrair do Novo CAGED (BigQuery/Base dos Dados) todas as movimentações de emprego formal no período 2021–2025.

Item	Descrição
Tabela BigQuery	`basedosdados.br_me_caged.microdados_movimentacao`
Período	2021-01 a 2025-12
Filtros	`ano BETWEEN 2021 AND 2025`
Estratégia	Download ano a ano via `google-cloud-bigquery` (Storage API) com fallback para `basedosdados`

> **Nota metodológica — Volume de dados:** O CAGED registra ~20-25 milhões de movimentações/ano. Para 5 anos, esperamos ~100-125M de registros. O download é feito ano a ano para evitar OOM e timeout, com salvamento em parquets individuais (`caged_{ano}.parquet`).

> **Nota sobre otimização:** Usamos a BigQuery Storage API (`create_bqstorage_client=True`) que transfere dados via gRPC/Arrow, sendo 2-5x mais rápida que o método padrão REST. Se o arquivo parquet já existir, o download é pulado automaticamente.

```
# Etapa 2a.2a — Download dos microdados CAGED
# Estratégia: download ano a ano via BigQuery Storage API, com cache local em parquet.

if not KEEP_CAGED_RAW:
    for f in DATA_RAW.glob("caged_*.parquet"):
        f.unlink()
        print(f"Cache CAGED removido: {f.name}")

from google.cloud import bigquery

def download_caged_ano(ano):
    """Baixar microdados CAGED de um ano via BigQuery Storage API."""
    parquet_path = DATA_RAW / f"caged_{ano}.parquet"

    if parquet_path.exists():
        size_mb = parquet_path.stat().st_size / 1e6
        df = pd.read_parquet(parquet_path)
        print(f"  {ano}: Carregado do cache — {len(df):,} registros ({size_mb:.0f} MB)")
        return df

    print(f"  {ano}: Baixando do BigQuery...", end="", flush=True)
    query = f"""
    SELECT {COLUNAS_CAGED}
    FROM `basedosdados.br_me_caged.microdados_movimentacao`
    WHERE ano = {ano}
    """
    client = bigquery.Client(project=GCP_PROJECT_ID)
    df = client.query(query).to_dataframe(create_bqstorage_client=True)
    df.to_parquet(parquet_path, index=False)
    size_mb = parquet_path.stat().st_size / 1e6
    print(f" {len(df):,} registros ({size_mb:.0f} MB)")
    return df

# ------------------------------------------------------------------------------
# Download ano a ano
# ------------------------------------------------------------------------------
print("Download dos microdados CAGED:")
dfs_anuais = []
for ano in range(ANO_INICIO, ANO_FIM + 1):
    df_ano = download_caged_ano(ano)
    dfs_anuais.append(df_ano)

# Resumo (sem concatenar em memória para evitar OOM)
total = sum(len(df) for df in dfs_anuais)
print(f"\nTotal: {total:,} movimentações ({ANO_INICIO}–{ANO_FIM})")
print(f"Colunas: {list(dfs_anuais[0].columns)}")
```

```
Download dos microdados CAGED:
  2021: Carregado do cache — 36,554,795 registros (380 MB)
  2022: Carregado do cache — 42,475,516 registros (441 MB)
  2023: Carregado do cache — 44,485,982 registros (469 MB)
  2024: Carregado do cache — 48,996,040 registros (510 MB)
  2025: Carregado do cache — 26,312,103 registros (269 MB)
```

```
Total: 198,824,436 movimentações (2021–2025)
Colunas: ['ano', 'mes', 'sigla_uf', 'id_municipio', 'cbo_2002', 'categoria',
'tipo_movimentacao', 'saldo_movimentacao', 'salario_mensal', 'grau_instrucao',
'idade', 'sexo', 'raca_cor', 'cnae_2_secao', 'cnae_2_subclasse',
'tamanho_estabelecimento_janeiro']
```

0.8 2b. Verificar dados CAGED (CHECKPOINT)

Verificar integridade dos dados baixados: cobertura temporal (12 meses/ano), volume por ano, preenchimento de variáveis-chave, e formato dos códigos CBO.

Critérios de aceite: - Todos os meses cobertos (Jan–Dez) para cada ano - CBO com >95% de preenchimento - ~20-30M registros por ano

```
# Etapa 2a.2b — CHECKPOINT: Verificar dados CAGED
# Carrega cada parquet individualmente (para evitar OOM)

print("=" * 60)
print("CHECKPOINT — Microdados CAGED")
print("=" * 60)

total_registros = 0
for ano in range(ANO_INICIO, ANO_FIM + 1):
    parquet_path = DATA_RAW / f"caged_{ano}.parquet"
    df = pd.read_parquet(parquet_path)

    # Volume
    print(f"\n--- {ano}: {len(df):,} movimentações ---")
    total_registros += len(df)

    # Cobertura mensal
    meses = sorted(df['mes'].dropna().unique())
    status = "OK" if len(meses) == 12 else f"ALERTA: {len(meses)} meses"
    print(f"  Meses: {len(meses)} ({status})")

    # Preenchimento CBO
    cbo_pct = df['cbo_2002'].notna().mean()
    print(f"  CBO preenchido: {cbo_pct:.1%}")

    # CBOs únicos
    cbos = df['cbo_2002'].dropna().astype(str).str[:4].nunique()
    print(f"  Famílias CBO 4d únicas: {cbos}")

    # Admissões vs desligamentos
    if 'saldo_movimentacao' in df.columns:
        adm = (df['saldo_movimentacao'] == 1).sum()
        desl = (df['saldo_movimentacao'] == -1).sum()
        print(f"  Admissões: {adm:,} | Desligamentos: {desl:,} | Saldo: {adm-desl:+,}")

print(f"\n{'=' * 60}")
```

```
print(f"TOTAL: {total_registros:,} movimentações ({ANO_INICIO}–{ANO_FIM})")
print(f"{'=' * 60}")
```

```
============================================================
CHECKPOINT — Microdados CAGED
============================================================

--- 2021: 36,554,795 movimentações ---
  Meses: 12 (OK)
  CBO preenchido: 100.0%
  Famílias CBO 4d únicas: 626
  Admissões: 19,703,604 | Desligamentos: 16,851,191 | Saldo: +2,852,413

--- 2022: 42,475,516 movimentações ---
  Meses: 12 (OK)
  CBO preenchido: 100.0%
  Famílias CBO 4d únicas: 624
  Admissões: 22,243,441 | Desligamentos: 20,232,075 | Saldo: +2,011,366

--- 2023: 44,485,982 movimentações ---
  Meses: 12 (OK)
  CBO preenchido: 100.0%
  Famílias CBO 4d únicas: 626
  Admissões: 22,982,161 | Desligamentos: 21,503,821 | Saldo: +1,478,340

--- 2024: 48,996,040 movimentações ---
  Meses: 12 (OK)
  CBO preenchido: 100.0%
  Famílias CBO 4d únicas: 625
  Admissões: 25,336,277 | Desligamentos: 23,659,763 | Saldo: +1,676,514

--- 2025: 26,312,103 movimentações ---
  Meses: 6 (ALERTA: 6 meses)
  CBO preenchido: 100.0%
  Famílias CBO 4d únicas: 622
  Admissões: 13,763,059 | Desligamentos: 12,549,044 | Saldo: +1,214,015

============================================================
TOTAL: 198,824,436 movimentações (2021–2025)
============================================================
```

0.9 3a. Agregação: Microdados → Painel Mensal por Ocupação

Agregar os microdados de movimentação (nível individual) em um painel mensal por ocupação CBO (4 dígitos). Cada linha do painel representará uma ocupação-mês com métricas agregadas.

0.9.1 Estratégia de agregação

Seguindo a abordagem de Hui, Reshef & Zhou (2024), construímos um painel ao nível de **ocupação × mês** com as seguintes métricas:

Métrica	Cálculo	Descrição
`admissoes`	Contagem de `saldo_movimentacao == 1`	Fluxo de contratação
`desligamentos`	Contagem de `saldo_movimentacao == -1`	Fluxo de demissão
`saldo`	`admissoes - desligamentos`	Criação líquida de empregos
`salario_medio_adm`	Média do `salario_mensal` (admissões)	Nível salarial
`salario_mediano_adm`	Mediana do `salario_mensal` (admissões)	Robustez a outliers
`pct_mulher_adm`	% de `sexo == 3` nas admissões	Composição de gênero
`pct_superior_adm`	% com `grau_instrucao >= 9`	Composição educacional

> **Nota — CBO 4 dígitos:** A CBO tem 6 dígitos (XXXX-XX), onde os 4 primeiros definem a "família" ocupacional. Para o crosswalk com ISCO-08, usamos os 4 primeiros dígitos (família CBO ≈ unit group ISCO-08).

> **Nota — Otimização de memória:** O processamento é feito ano a ano para evitar OOM (Out-of-Memory) com ~100M+ registros. Flags booleanos são pré-computados como float para permitir agregação vetorizada (evitando lambdas lentas). A mediana é calculada separadamente por ser computacionalmente cara.

Validação da codificação `sexo` (CAGED/Base dos Dados): Confirmar que os valores são 1 = Masculino e 3 = Feminino (não há código 2). O agregado `pct_mulher_adm` usa `sexo == 3`.

```
# Conferir codificação de sexo nos microdados CAGED (um ano como amostra)
_ano_amostra = 2024
_path_amostra = DATA_RAW / f"caged_{_ano_amostra}.parquet"
if not _path_amostra.exists():
    _ano_amostra = ANO_INICIO
    _path_amostra = DATA_RAW / f"caged_{_ano_amostra}.parquet"
if _path_amostra.exists():
    _df_sexo = pd.read_parquet(_path_amostra, columns=["sexo"])
    print(f"sexo — value_counts (ano {_ano_amostra}):")
    print(_df_sexo["sexo"].value_counts().sort_index())
    print("  Esperado: 1 = Masculino, 3 = Feminino (Base dos Dados/CAGED).")
else:
    print("Nenhum parquet de microdados encontrado para verificar sexo.")
```

```
sexo — value_counts (ano 2024):
sexo
1    28494836
3    20500968
9         236
Name: count, dtype: int64
  Esperado: 1 = Masculino, 3 = Feminino (Base dos Dados/CAGED).
```

```
# Etapa 2a.3a — Agregação: Microdados → Painel Mensal por Ocupação
# Checkpoint: se o painel já existe, carrega direto.

if not KEEP_PANEL_MENSAL and PAINEL_MENSAL_FILE.exists():
    PAINEL_MENSAL_FILE.unlink()
    print(f"Cache removido: {PAINEL_MENSAL_FILE.name}")

if PAINEL_MENSAL_FILE.exists():
    painel = pd.read_parquet(PAINEL_MENSAL_FILE)
    print(f"Painel carregado do checkpoint: {PAINEL_MENSAL_FILE.name}")
    print(f"  {len(painel):,} linhas, {painel['cbo_4d'].nunique()} ocupações, "
          f"{painel['periodo'].nunique()} períodos")
else:
    print("Construindo painel a partir dos microdados (ano a ano)...")
    paineis_anuais = []

    for ano in range(ANO_INICIO, ANO_FIM + 1):
        print(f"\n  Processando {ano}...", flush=True)
        df = pd.read_parquet(DATA_RAW / f"caged_{ano}.parquet")

        # CBO 4 dígitos
        df['cbo_2002'] = df['cbo_2002'].astype(str).str.strip()
        df['cbo_4d'] = df['cbo_2002'].str[:4]
        df = df[df['cbo_4d'].str.len() == 4]
        df = df[df['cbo_4d'].str.isdigit()]
        df = df[~df['cbo_4d'].isin(['0000'])]

        # Variáveis temporais
        df['periodo'] = df['ano'].astype(str) + '-' + df['mes'].astype(str).str.zfill(
        df['periodo_num'] = df['ano'].astype(int) * 100 + df['mes'].astype(int)
        df['post'] = (df['periodo_num'] >= ANO_TRATAMENTO * 100 + MES_TRATAMENTO).asty

        # Flags booleanos (CAGED: sexo 1=Masc, 3=Fem; alinhado à Etapa 1a)
        CODIGO_SEXO_MULHER = 3
        CODIGOS_RACA_BRANCA, CODIGOS_RACA_NEGRA = [1], [2, 4]
        IDADE_CORTE_JOVEM = 29
        CODIGOS_ESCOLARIDADE_SUPERIOR = ['9', '10', '11', '12', '13']
        df['is_mulher'] = (df['sexo'].astype(str) == str(CODIGO_SEXO_MULHER)).astype(f
        raca_str = df['raca_cor'].astype(str)
        df['is_branco'] = raca_str.isin([str(c) for c in CODIGOS_RACA_BRANCA]).astype(
        df['is_negro'] = raca_str.isin([str(c) for c in CODIGOS_RACA_NEGRA]).astype(fl
        df['is_jovem'] = (df['idade'] <= IDADE_CORTE_JOVEM).astype(float)
        df['is_superior'] = df['grau_instrucao'].astype(str).isin(CODIGOS_ESCOLARIDADE
        # Setor tecnológico (CNAE 2.0 seção: J = Informação e comunicação, etc.)
        df['is_setor_tech'] = df['cnae_2_secao'].astype(str).str.strip().isin(CNAE_SEC
```

```
        # Separar admissões e desligamentos
        df_adm = df[df['saldo_movimentacao'] == 1].copy()
        df_desl = df[df['saldo_movimentacao'] == -1]
        print(f"    {ano}: {len(df_adm):,} admissões, {len(df_desl):,} desligamentos",

        # Colunas de salário mascaradas (média condicional por grupo)
        df_adm['sal_mulher'] = np.where(df_adm['is_mulher'] == 1, df_adm['salario_mens
        df_adm['sal_homem'] = np.where(df_adm['is_mulher'] == 0, df_adm['salario_mensa
        df_adm['sal_branco'] = np.where(df_adm['is_branco'] == 1, df_adm['salario_mens
        df_adm['sal_negro'] = np.where(df_adm['is_negro'] == 1, df_adm['salario_mensal
        df_adm['sal_jovem'] = np.where(df_adm['is_jovem'] == 1, df_adm['salario_mensal
        df_adm['sal_naojovem'] = np.where(df_adm['is_jovem'] == 0, df_adm['salario_men
        df_adm['sal_sup'] = np.where(df_adm['is_superior'] == 1, df_adm['salario_mensa
        df_adm['sal_med'] = np.where(df_adm['is_superior'] == 0, df_adm['salario_mensa
        df_adm['is_homem'] = 1 - df_adm['is_mulher']

        # Agregar admissões + heterogeneidade demográfica
        painel_adm = df_adm.groupby(['cbo_4d', 'ano', 'mes']).agg(
            admissoes=('saldo_movimentacao', 'count'),
            salario_medio_adm=('salario_mensal', 'mean'),
            idade_media_adm=('idade', 'mean'),
            pct_mulher_adm=('is_mulher', 'mean'),
            pct_superior_adm=('is_superior', 'mean'),
            pct_branco_adm=('is_branco', 'mean'),
            pct_negro_adm=('is_negro', 'mean'),
            pct_jovem_adm=('is_jovem', 'mean'),
            pct_tecnologico_adm=('is_setor_tech', 'mean'),
            salario_medio_mulher=('sal_mulher', 'mean'),
            salario_medio_homem=('sal_homem', 'mean'),
            salario_medio_branco=('sal_branco', 'mean'),
            salario_medio_negro=('sal_negro', 'mean'),
            salario_medio_jovem=('sal_jovem', 'mean'),
            salario_medio_naojovem=('sal_naojovem', 'mean'),
            salario_medio_superior=('sal_sup', 'mean'),
            salario_medio_medio=('sal_med', 'mean'),
            admissoes_mulher=('is_mulher', 'sum'),
            admissoes_homem=('is_homem', 'sum'),
            admissoes_jovem=('is_jovem', 'sum'),
            admissoes_negro=('is_negro', 'sum'),
        ).reset_index()

        # Mediana separada (performance)
        mediana = df_adm.groupby(['cbo_4d', 'ano', 'mes'])['salario_mensal'].median().
        mediana.columns = ['cbo_4d', 'ano', 'mes', 'salario_mediano_adm']
        painel_adm = painel_adm.merge(mediana, on=['cbo_4d', 'ano', 'mes'], how='left'

        # Agregar desligamentos
        painel_desl = df_desl.groupby(['cbo_4d', 'ano', 'mes']).agg(
            desligamentos=('saldo_movimentacao', 'count'),
            salario_medio_desl=('salario_mensal', 'mean'),
        ).reset_index()

        # Merge
        p = painel_adm.merge(painel_desl, on=['cbo_4d', 'ano', 'mes'], how='outer').fi
```

```
        p['saldo'] = p['admissoes'] - p['desligamentos']
        p['n_movimentacoes'] = p['admissoes'] + p['desligamentos']
        p['setor_tecnologico'] = (p['pct_tecnologico_adm'] >= SETOR_TECNOLOGICO_LIMIAR
        p['periodo'] = p['ano'].astype(int).astype(str) + '-' + p['mes'].astype(int).a
        p['periodo_num'] = p['ano'].astype(int) * 100 + p['mes'].astype(int)
        p['post'] = (p['periodo_num'] >= ANO_TRATAMENTO * 100 + MES_TRATAMENTO).astype
        p['ln_admissoes'] = np.log(p['admissoes'] + 1)
        p['ln_desligamentos'] = np.log(p['desligamentos'] + 1)
        p['ln_salario_adm'] = np.log(p['salario_medio_adm'].clip(lower=1))
        p['cbo_2d'] = p['cbo_4d'].str[:2]
        # Logs de heterogeneidade (salários e admissões por grupo)
        for grp in ['mulher', 'homem', 'branco', 'negro', 'jovem', 'naojovem', 'superi
            col = f'salario_medio_{grp}'
            if col in p.columns:
                p[f'ln_salario_{grp}'] = np.log(p[col].clip(lower=1))
        for grp in ['mulher', 'homem', 'jovem', 'negro']:
            col = f'admissoes_{grp}'
            if col in p.columns:
                p[f'ln_admissoes_{grp}'] = np.log(p[col].astype(float) + 1)

        paineis_anuais.append(p)
        print(f"    → {len(p):,} linhas no painel", flush=True)

    painel = pd.concat(paineis_anuais, ignore_index=True)
    painel.to_parquet(PAINEL_MENSAL_FILE, index=False)
    print(f"\nPainel salvo: {PAINEL_MENSAL_FILE.name}")

print(f"\nPainel final: {len(painel):,} linhas")
print(f"  Ocupações: {painel['cbo_4d'].nunique()}, Períodos: {painel['periodo'].nuniqu
print(f"  Shape: {painel.shape}")
```

```
Painel carregado do checkpoint: painel_caged_mensal.parquet
  32,988 linhas, 629 ocupações, 54 períodos

Painel final: 32,988 linhas
  Ocupações: 629, Períodos: 54
  Shape: (32988, 49)
```

0.10 3b. Verificar painel agregado (CHECKPOINT)

Verificar integridade do painel: dimensões, balanceamento (ocupações × períodos), cobertura temporal, distribuição de variáveis-chave e série temporal agregada.

```
# Etapa 2a.3b — CHECKPOINT: Verificar painel agregado

print("=" * 60)
print("CHECKPOINT — Painel Ocupação × Mês")
print("=" * 60)

# 1. Dimensões
n_ocup = painel['cbo_4d'].nunique()
n_periodos = painel['periodo'].nunique()
print(f"\nOcupações: {n_ocup}")
```

```
print(f"Períodos: {n_periodos}")
print(f"Painel teórico (balanceado): {n_ocup * n_periodos:,}")
print(f"Painel real: {len(painel):,}")
print(f"Balanceamento: {len(painel) / (n_ocup * n_periodos):.1%}")

# 2. Ocupações com poucos meses
ocup_meses = painel.groupby('cbo_4d')['periodo'].nunique()
print(f"\nMeses por ocupação:")
print(f"  Min: {ocup_meses.min()}, Max: {ocup_meses.max()}, Média: {ocup_meses.mean():
print(f"  Ocupações com < 12 meses: {(ocup_meses < 12).sum()}")
print(f"  Ocupações com todos os {n_periodos} meses: {(ocup_meses == n_periodos).sum()

# 3. Estatísticas descritivas
print("\nEstatísticas descritivas:")
print(painel[['admissoes', 'desligamentos', 'saldo', 'salario_medio_adm']].describe().

# 4. Série temporal agregada
ts = painel.groupby('periodo_num').agg(
    total_adm=('admissoes', 'sum'),
    total_desl=('desligamentos', 'sum'),
    sal_medio=('salario_medio_adm', 'mean'),
).reset_index()
print("\nSérie temporal (primeiros e últimos 3 meses):")
print(ts.head(3).to_string(index=False))
print("...")
print(ts.tail(3).to_string(index=False))
```

```
============================================================
CHECKPOINT — Painel Ocupação × Mês
============================================================

Ocupações: 629
Períodos: 54
Painel teórico (balanceado): 33,966
Painel real: 32,988
Balanceamento: 97.1%

Meses por ocupação:
  Min: 2, Max: 54, Média: 52.4
  Ocupações com < 12 meses: 9
  Ocupações com todos os 54 meses: 589

Estatísticas descritivas:
       admissoes  desligamentos    saldo  salario_medio_adm
count    32988.0        32988.0  32988.0            32988.0
mean      3153.5         2873.6    279.9             6131.1
std      13880.0        12450.5   2323.0           158475.4
min          0.0            0.0 -46650.0                0.0
25%         60.0           60.0    -16.0             1782.8
50%        293.0          289.0      7.0             2371.3
75%       1338.0         1264.0    107.0             3811.2
max     289900.0       284338.0  90556.0         18799538.2

Série temporal (primeiros e últimos 3 meses):
```

```
 periodo_num  total_adm  total_desl   sal_medio
      202101    1550075     1293016 3697.157086
      202102    1715425     1317510 3723.743109
      202103    1626885     1450555 3252.934263
...
 periodo_num  total_adm  total_desl    sal_medio
      202504    2282187     2024659  4011.260301
      202505    2256225     2107233  4441.730675
      202506    2139182     1972561 11519.380585
```

0.10.1 Verificação: outlier salarial em Jun/2025

A série temporal agregada no checkpoint acima pode mostrar salário médio elevado em Jun/2025 (~3× os meses anteriores). Abaixo verificamos se isso reflete **(i)** poucas células ocupação×mês com salário muito alto e/ou poucas movimentações, **(ii)** possível publicação parcial do mês, ou **(iii)** padrão real dos dados. Conforme o diagnóstico, pode-se documentar, filtrar células com poucas movimentações ou truncar a janela em Mai/2025.

```
# Diagnóstico: Jun/2025 — distribuição de salario_medio_adm e células extremas
jun = painel[painel['periodo_num'] == 202506].copy()
print("Jun/2025 — Distribuição de salario_medio_adm (células ocupação×mês):")
print(jun['salario_medio_adm'].describe(percentiles=[0.5, 0.9, 0.95, 0.99]).round(2))
print()

# Células com poucas movimentações e salário alto
limiar_mov = 50
limiar_sal = 10_000
mask_extremo = (jun['n_movimentacoes'] < limiar_mov) & (jun['salario_medio_adm'] > lim
n_extremo = mask_extremo.sum()
adm_jun_total = jun['admissoes'].sum()
adm_extremo = jun.loc[mask_extremo, 'admissoes'].sum()
print(f"Células com n_movimentacoes < {limiar_mov} e salario_medio_adm > R$ {limiar_sa
print(f"  Admissões nessas células: {adm_extremo:,} ({100*adm_extremo/adm_jun_total:.2
if n_extremo > 0:
    print(jun.loc[mask_extremo, ['cbo_4d', 'admissoes', 'n_movimentacoes', 'salario_me
print()

# Comparação com Mai/2025 e média 2025
mai = painel[painel['periodo_num'] == 202505]
ano2025 = painel[painel['periodo_num'] // 100 == 2025]
print("Comparativo 2025:")
print(f"  Mai/2025: células={len(mai)}, total_adm={mai['admissoes'].sum():,.0f}, sal_m
print(f"  Jun/2025: células={len(jun)}, total_adm={adm_jun_total:,.0f}, sal_medio(agg)
print(f"  Média mensal 2025: células~{len(ano2025)//12:.0f}, total_adm~{ano2025.groupb
```

```
Jun/2025 — Distribuição de salario_medio_adm (células ocupação×mês):
count        615.00
mean       11519.38
std       170295.09
min            0.00
50%         2647.30
90%         7652.32
```

```
95%        10449.24
99%        38462.94
max      4216672.39
Name: salario_medio_adm, dtype: float64

Células com n_movimentacoes < 50 e salario_medio_adm > R$ 10,000: 9
  Admissões nessas células: 125 (0.01% do total de Jun/2025)
cbo_4d  admissoes  n_movimentacoes  salario_medio_adm
  2423          1                1       80466.670000
  1237         22               48       43743.163182
  1222         19               49       41853.112105
  1234         15               36       38843.381333
  1221          6               15       35815.645000
  1223         19               36       14490.656842
  2422          1                1       14097.220000
  2622         24               48       12752.264583
  1031         18               20       10953.792222

Comparativo 2025:
  Mai/2025: células=614, total_adm=2,256,225, sal_medio(agg)=4,442
  Jun/2025: células=615, total_adm=2,139,182, sal_medio(agg)=11,519
  Média mensal 2025: células~306, total_adm~2,293,843
```

0.11 4a. Crosswalk CBO 2002 → ISCO-08

Mapear os códigos CBO 2002 (usados no CAGED) para ISCO-08 (usados no índice ILO). **Esta é a etapa metodologicamente mais delicada do pipeline.**

0.11.1 Contexto

A CBO 2002 foi construída com base na ISCO-88/ISCO-08, compartilhando a mesma estrutura hierárquica:

Nível	CBO 2002	ISCO-08	Alinhamento
1 dígito	Grande Grupo (10)	Major Group (10)	Perfeito
2 dígitos	Subgrupo Principal (~46)	Sub-major Group (43)	Bom (14 CBOs sem match direto)
3 dígitos	Subgrupo (~194)	Minor Group (130)	Parcial (~45% direto)
4 dígitos	Família (~629)	Unit Group (427)	Divergente (~28% direto)

0.11.2 Estratégia adotada: Dual (2d principal + 4d robustez)

PARTE A — Especificação PRINCIPAL (2 dígitos): - CBO 2d → ISCO-08 Sub-major Group (match direto) - Fallback: CBO 1d → ISCO-08 Major Group (média) - Cobertura esperada: **100%**

PARTE B — Especificação de ROBUSTEZ (4 dígitos, fallback hierárquico em 6 níveis):

Nível	Estratégia	Cobertura esperada
N1	CBO 4d = ISCO-08 4d (match direto)	~28%
N2	CBO 4d = ISCO-88 4d → ISCO-08 via correspondência oficial	~+9%
N3	CBO 3d = ISCO-08 3d (média Minor Group)	~+20%
N4	CBO 3d = ISCO-88 3d → ISCO-08 via correspondência	~+1%
N5	CBO 2d = ISCO-08 2d (= especificação principal)	~+18%
N6	CBO 1d = ISCO-08 1d (média Major Group)	~+24%

> **Nota — Muendler (CBO 1994):** O arquivo `cbo-isco-conc.csv` de Muendler & Poole (2004) mapeia CBO **1994** → ISCO-88, NÃO a CBO 2002 usada no CAGED. Por isso, o match 4d via Muendler é limitado. A estratégia principal utiliza a similaridade estrutural entre CBO 2002 e ISCO-08/88 com fallback hierárquico.

> **Nota sobre atenuação:** Se o crosswalk a 4 dígitos introduz erro de medição, o efeito típico é **atenuação** (viés em direção a zero). Encontrar efeito significativo mesmo com erro de medição sugere que o efeito real é provavelmente maior.

Validação do crosswalk: O notebook não usa Muendler para o match principal; utiliza a correspondência oficial ISCO-08↔88 e fallback hierárquico. Na especificação 2d: match direto CBO 2d → ISCO-08 Sub-major Group, com fallback a 1 dígito (Major Group). Na 4d: fallback em 6 níveis (N1→N6). Resultado verificado: cobertura 100% em 2d e 4d, correlação entre exposure_score_2d e exposure_score_4d ~0,915 — consistente com o planejamento e pronto para o DiD no Notebook 2b.

```
# Etapa 2a.4a — Crosswalk CBO 2002 → ISCO-08 (Dual: 2d principal + 4d robustez)
# Checkpoint: se o painel com crosswalk já existe, carrega direto.

if not KEEP_PANEL_CROSSWALK and PAINEL_CROSSWALK_FILE.exists():
    PAINEL_CROSSWALK_FILE.unlink()
    print(f"Cache removido: {PAINEL_CROSSWALK_FILE.name}")

if PAINEL_CROSSWALK_FILE.exists():
    painel = pd.read_parquet(PAINEL_CROSSWALK_FILE)
    print(f"Crosswalk carregado do checkpoint: {PAINEL_CROSSWALK_FILE.name}")
    print(f"  {len(painel):,} linhas, cobertura 2d: {painel['exposure_score_2d'].notna
          f"4d: {painel['exposure_score_4d'].notna().mean():.1%}")
```

```
else:
    # ═══════════════════════════════════════════════════════════════════
    # Carregar dados de referência
    # ═══════════════════════════════════════════════════════════════════
    df_ilo = pd.read_csv(ILO_FILE)
    df_ilo['isco_08_str'] = df_ilo['isco_08'].astype(str).str.zfill(4)
    print(f"Índice ILO carregado: {len(df_ilo)} ocupações ISCO-08")
    print(f"  Score range: [{df_ilo['exposure_score'].min():.3f}, {df_ilo['exposure_sc

    # Dicts ILO em múltiplos níveis
    codes = df_ilo['isco_08_str']
    ilo_4d = df_ilo.groupby('isco_08_str')['exposure_score'].mean().to_dict()
    ilo_3d = df_ilo.assign(g=codes.str[:3]).groupby('g')['exposure_score'].mean().to_d
    ilo_2d = df_ilo.assign(g=codes.str[:2]).groupby('g')['exposure_score'].mean().to_d
    ilo_1d = df_ilo.assign(g=codes.str[:1]).groupby('g')['exposure_score'].mean().to_d

    # Correspondência ISCO-08 ↔ ISCO-88 (arquivo local)
    isco88_to_08 = {}
    isco88_3d_to_08_3d = {}
    if ISCO_08_88_FILE.exists():
        df_corr = pd.read_excel(ISCO_08_88_FILE, sheet_name='ISCO-08 to 88')
        df_corr['isco08_4d'] = df_corr['ISCO-08 code'].astype(str).str.strip().str.zfi
        df_corr['isco88_4d'] = df_corr['ISCO-88 code'].astype(str).str.strip().str.zfi
        isco88_to_08 = df_corr.groupby('isco88_4d')['isco08_4d'].apply(list).to_dict()
        df_corr['isco88_3d'] = df_corr['isco88_4d'].str[:3]
        df_corr['isco08_3d'] = df_corr['isco08_4d'].str[:3]
        isco88_3d_to_08_3d = df_corr.groupby('isco88_3d')['isco08_3d'].apply(
            lambda x: list(set(x))).to_dict()
        print(f"  Correspondência ISCO-08↔88: {len(df_corr)} mapeamentos")

    # ═══════════════════════════════════════════════════════════════════
    # PARTE A: 2 dígitos (PRINCIPAL)
    # ═══════════════════════════════════════════════════════════════════
    print(f"\n{'='*60}\nPARTE A: Crosswalk 2 dígitos (PRINCIPAL)\n{'='*60}")

    painel['exposure_score_2d'] = painel['cbo_2d'].map(ilo_2d)
    painel['match_level_2d'] = np.where(painel['exposure_score_2d'].notna(), '2-digit'

    # Fallback a 1 dígito
    mask_na = painel['exposure_score_2d'].isna()
    if mask_na.any():
        cbo_1d = painel.loc[mask_na, 'cbo_4d'].str[:1]
        painel.loc[mask_na, 'exposure_score_2d'] = cbo_1d.map(ilo_1d).values
        painel.loc[mask_na, 'match_level_2d'] = '1-digit (fallback)'

    painel['exposure_score'] = painel['exposure_score_2d']
    cov_2d = painel['exposure_score_2d'].notna().mean()
    print(f"  COBERTURA 2d: {cov_2d:.1%}")
    for lvl, cnt in painel['match_level_2d'].value_counts().items():
        print(f"    {lvl}: {cnt:,} ({cnt/len(painel):.1%})")

    # ═══════════════════════════════════════════════════════════════════
    # PARTE B: 4 dígitos (ROBUSTEZ) — fallback hierárquico 6 níveis
    # ═══════════════════════════════════════════════════════════════════
```

```
print(f"\n{'='*60}\nPARTE B: Crosswalk 4 dígitos (ROBUSTEZ)\n{'='*60}")

cbos_unicos = sorted(painel['cbo_4d'].unique())
cbo_score_4d = {}
cbo_match_level = {}
counts = {'N1': 0, 'N2': 0, 'N3': 0, 'N4': 0, 'N5': 0, 'N6': 0, 'sem': 0}

for cbo in cbos_unicos:
    score, level = None, None

    # N1: CBO 4d = ISCO-08 4d
    if cbo in ilo_4d:
        score, level = ilo_4d[cbo], 'N1: ISCO-08 4d direto'
        counts['N1'] += 1
    # N2: CBO 4d = ISCO-88 4d → ISCO-08
    if score is None and cbo in isco88_to_08:
        scores_c = [ilo_4d[c] for c in isco88_to_08[cbo] if c in ilo_4d]
        if scores_c:
            score, level = np.mean(scores_c), 'N2: via ISCO-88→08 4d'
            counts['N2'] += 1
    # N3: CBO 3d = ISCO-08 3d
    if score is None and cbo[:3] in ilo_3d:
        score, level = ilo_3d[cbo[:3]], 'N3: ISCO-08 3d'
        counts['N3'] += 1
    # N4: CBO 3d = ISCO-88 3d → ISCO-08 3d
    if score is None and cbo[:3] in isco88_3d_to_08_3d:
        scores_c = [ilo_3d[c] for c in isco88_3d_to_08_3d[cbo[:3]] if c in ilo_3d]
        if scores_c:
            score, level = np.mean(scores_c), 'N4: via ISCO-88→08 3d'
            counts['N4'] += 1
    # N5: CBO 2d = ISCO-08 2d
    if score is None and cbo[:2] in ilo_2d:
        score, level = ilo_2d[cbo[:2]], 'N5: ISCO-08 2d'
        counts['N5'] += 1
    # N6: CBO 1d = ISCO-08 1d
    if score is None and cbo[:1] in ilo_1d:
        score, level = ilo_1d[cbo[:1]], 'N6: ISCO-08 1d'
        counts['N6'] += 1

    if score is not None:
        cbo_score_4d[cbo] = score
        cbo_match_level[cbo] = level
    else:
        counts['sem'] += 1

painel['exposure_score_4d'] = painel['cbo_4d'].map(cbo_score_4d)
painel['match_level_4d'] = painel['cbo_4d'].map(cbo_match_level)

total = len(cbos_unicos)
print(f"  CBOs 4d únicos: {total}")
for k, v in counts.items():
    if v > 0:
        print(f"    {k}: {v} ({v/total:.1%})")
print(f"  COBERTURA 4d: {painel['exposure_score_4d'].notna().mean():.1%}")
```

```
    # Correlação 2d vs 4d
    df_check = painel[['cbo_4d', 'exposure_score_2d', 'exposure_score_4d']].drop_dupli
    corr = df_check['exposure_score_2d'].corr(df_check['exposure_score_4d'])
    print(f"\n  Correlação Pearson (2d vs 4d): {corr:.4f}")

    painel.to_parquet(PAINEL_CROSSWALK_FILE, index=False)
    print(f"\nSalvo: {PAINEL_CROSSWALK_FILE.name}")
```

```
Crosswalk carregado do checkpoint: painel_caged_crosswalk.parquet
  32,988 linhas, cobertura 2d: 100.0%, 4d: 100.0%
```

CBOs 2 dígitos sem match direto em ISCO-08: Subgrupos principais CBO que não possuem equivalente direto em Sub-major Group ISCO-08; recebem score via fallback a 1 dígito (Major Group). Lista abaixo (a partir do painel com crosswalk).

```
# Listar CBO 2d que caem no fallback a 1 dígito (sem match direto em ISCO-08 Sub-major
if 'match_level_2d' in painel.columns:
    fallback_2d = painel[painel['match_level_2d'] == '1-digit (fallback)']['cbo_2d'].u
    fallback_2d = sorted(fallback_2d)
    print(f"CBOs 2d sem match direto (fallback a Major Group): {len(fallback_2d)}")
    print("Códigos:", fallback_2d)
    # Opcional: exemplos de cbo_4d por um desses 2d
    if len(fallback_2d) > 0:
        ex = painel[painel['cbo_2d'] == fallback_2d[0]][['cbo_2d', 'cbo_4d']].drop_dup
        print(f"\nExemplo (cbo_2d={fallback_2d[0]}): cbo_4d presentes: {ex['cbo_4d'].t
else:
    print("Coluna match_level_2d não encontrada (crosswalk pode ter sido carregado sem
```

```
CBOs 2d sem match direto (fallback a Major Group): 14
Códigos: ['10', '20', '27', '30', '37', '39', '64', '76', '77', '78', '79', '84',
'86', '99']

Exemplo (cbo_2d=10): cbo_4d presentes: ['1010', '1011', '1020', '1021', '1030',
'1031']...
```

0.12 4b. Verificar crosswalk (CHECKPOINT)

Validar qualidade do crosswalk nas DUAS especificações: principal (2 dígitos) e robustez (4 dígitos). Verificar cobertura, distribuição de scores, correlação entre especificações e sanity check por grande grupo CBO.

Critérios de aceite: - Cobertura 2d ≥ 95% (esperado ~100%) - Cobertura 4d ≥ 80% (esperado ~100% com fallback) - Correlação 2d vs 4d > 0.8 (consistência entre especificações)

```
# Etapa 2a.4b — CHECKPOINT: Verificar crosswalk CBO → ISCO-08

print("=" * 60)
print("CHECKPOINT — Crosswalk CBO → ISCO-08 (Dual)")
print("=" * 60)

# 1. Cobertura
```

```
coverage_2d = painel['exposure_score_2d'].notna().mean()
coverage_4d = painel['exposure_score_4d'].notna().mean()
print(f"\n--- Cobertura ---")
print(f"  2 dígitos (PRINCIPAL): {coverage_2d:.1%}")
print(f"  4 dígitos (ROBUSTEZ):  {coverage_4d:.1%}")
if coverage_2d < 0.95:
    print(f"  ALERTA: Cobertura 2d abaixo de 95%!")
if coverage_4d < 0.80:
    print(f"  AVISO: Cobertura 4d abaixo de 80%.")

# 2. Estatísticas dos scores
print(f"\n--- Estatísticas dos scores ---")
print(f"\nexposure_score_2d (PRINCIPAL):")
print(painel['exposure_score_2d'].describe().round(4))
print(f"\nexposure_score_4d (ROBUSTEZ):")
print(painel['exposure_score_4d'].describe().round(4))

# 3. Correlação
mask_both = painel['exposure_score_2d'].notna() & painel['exposure_score_4d'].notna()
if mask_both.any():
    corr = painel.loc[mask_both, 'exposure_score_2d'].corr(
        painel.loc[mask_both, 'exposure_score_4d'])
    print(f"\n--- Correlação 2d vs 4d ---")
    print(f"  Pearson: {corr:.4f}")
    print(f"  {'Alta correlação — bom sinal de consistência.' if corr > 0.8 else 'Corr

# 4. Sanity check por grande grupo CBO
painel['grande_grupo_cbo'] = painel['cbo_4d'].str[0]
print(f"\n--- Exposição por grande grupo CBO ---")
print(f"{'Grande Grupo':<40} {'Score 2d':>10} {'Score 4d':>10}")
print("-" * 62)
for gg, nome in sorted(GRANDES_GRUPOS_CBO.items()):
    mask = painel['grande_grupo_cbo'] == gg
    if mask.any():
        s2d = painel.loc[mask, 'exposure_score_2d'].mean()
        s4d = painel.loc[mask, 'exposure_score_4d'].mean()
        s4d_str = f"{s4d:.3f}" if not np.isnan(s4d) else "N/A"
        flag = " (!)" if not np.isnan(s4d) and abs(s2d - s4d) > 0.1 else ""
        print(f"  {nome:<38} {s2d:>10.3f} {s4d_str:>10}{flag}")
print(f"  (!) = diferença > 0.1 entre 2d e 4d")
```

```
============================================================
CHECKPOINT — Crosswalk CBO → ISCO-08 (Dual)
============================================================

--- Cobertura ---
  2 dígitos (PRINCIPAL): 100.0%
  4 dígitos (ROBUSTEZ):  100.0%
```

```
--- Estatísticas dos scores ---

exposure_score_2d (PRINCIPAL):
count    32988.0000
mean         0.2778
```

```
std          0.1243
min          0.1167
25%          0.1658
50%          0.2459
75%          0.3725
max          0.6325
Name: exposure_score_2d, dtype: float64

exposure_score_4d (ROBUSTEZ):
count    32988.0000
mean         0.2830
std          0.1315
min          0.0900
25%          0.1658
50%          0.2500
75%          0.3650
max          0.7000
Name: exposure_score_4d, dtype: float64

--- Correlação 2d vs 4d ---
  Pearson: 0.9150
  Alta correlação — bom sinal de consistência.

--- Exposição por grande grupo CBO ---
Grande Grupo                                  Score 2d   Score 4d
------------------------------------------------------------------
  Dirigentes                                     0.367      0.375
  Profissionais das ciências                     0.393      0.396
  Técnicos nível médio                           0.334      0.338
  Trabalhadores de serv. admin.                  0.580      0.557
  Trabalhadores de serviços/comércio             0.243      0.247
  Agropecuária                                   0.157      0.161
  Produção industrial                            0.161      0.165
  Operadores de máquinas                         0.213      0.217
  Manutenção e reparação                         0.143      0.187
  (!) = diferença > 0.1 entre 2d e 4d
```

0.13 5a. Definição de tratamento

Definir as variáveis de tratamento para a análise DiD. O tratamento é baseado na **exposição ocupacional à IA generativa**: ocupações com alta exposição (top 20%) vs. baixa exposição.

0.13.1 Variáveis criadas

Variável	Definição	Uso
`alta_exp`	1 se `exposure_score_2d >= percentil 80`	**Especificação principal**
`alta_exp_10`	1 se `exposure_score_2d >= percentil 90`	Robustez (cutoff)

Variável	Definição	Uso
`alta_exp_25`	1 se `exposure_score_2d >= percentil 75`	Robustez (cutoff)
`alta_exp_mediana`	1 se `exposure_score_2d >= mediana`	Alternativa binária
`quintil_exp`	Quintil de exposição (Q1–Q5)	Análise por quantil
`alta_exp_4d`	1 se `exposure_score_4d >= percentil 80`	Robustez (crosswalk 4d)
`did`	`post × alta_exp`	Interação DiD principal
`did_4d`	`post × alta_exp_4d`	Interação DiD robustez

> **Nota:** Os thresholds são calculados sobre a distribuição de **ocupações** (uma obs por CBO), não ponderada por volume de movimentações. Cada ocupação tem peso igual na definição do tratamento.

> **Nota — Tratamento contínuo:** Além das dummies, `exposure_score_2d` e `exposure_score_4d` podem ser usados diretamente como tratamento contínuo em especificações alternativas, conforme Hui et al. (2024).

```
# Etapa 2a.5a — Definição de tratamento

# ── Thresholds sobre a distribuição de ocupações (2d) ──
ocup_scores_2d = painel.groupby('cbo_4d')['exposure_score_2d'].first().dropna()

thresholds_2d = {
    'alta_exp_10':      ocup_scores_2d.quantile(0.90),
    'alta_exp':         ocup_scores_2d.quantile(0.80),  # PRINCIPAL
    'alta_exp_25':      ocup_scores_2d.quantile(0.75),
    'alta_exp_mediana':  ocup_scores_2d.quantile(0.50),
}

print("Thresholds de exposição (2d, PRINCIPAL):")
for name, val in thresholds_2d.items():
    n_above = (ocup_scores_2d >= val).sum()
    pct = n_above / len(ocup_scores_2d) * 100
    print(f"  {name}: {val:.4f} ({n_above} ocupações, {pct:.0f}%)")

# ── Dummies de tratamento 2d ──
for name, threshold in thresholds_2d.items():
    painel[name] = (painel['exposure_score_2d'] >= threshold).astype(int)
```

```
# Quintis
painel['quintil_exp'] = pd.qcut(
    painel['exposure_score_2d'].rank(method='first'),
    q=5,
    labels=['Q1 (Baixa)', 'Q2', 'Q3', 'Q4', 'Q5 (Alta)']
)

# ── Dummies 4d (ROBUSTEZ) ──
ocup_scores_4d = painel.groupby('cbo_4d')['exposure_score_4d'].first().dropna()
threshold_4d_80 = ocup_scores_4d.quantile(0.80)
painel['alta_exp_4d'] = (painel['exposure_score_4d'] >= threshold_4d_80).astype(int)
print(f"\nThreshold 4d (p80): {threshold_4d_80:.4f} ({(ocup_scores_4d >= threshold_4d_

# ── Interações DiD ──
painel['did'] = painel['post'] * painel['alta_exp']
painel['did_4d'] = painel['post'] * painel['alta_exp_4d']

# ── Resumo ──
print(f"\n--- Distribuição de tratamento ---")
print(f"  Alta exp 2d (top 20%): {painel['alta_exp'].mean():.1%} das obs")
print(f"  Alta exp 4d (top 20%): {painel['alta_exp_4d'].mean():.1%} das obs")
print(f"  Períodos pré:  {painel[painel['post']==0].shape[0]:,}")
print(f"  Períodos pós:  {painel[painel['post']==1].shape[0]:,}")

concordancia = (painel['alta_exp'] == painel['alta_exp_4d']).mean()
print(f"  Concordância 2d vs 4d: {concordancia:.1%}")

# Tabela de contingência
ct = pd.crosstab(
    painel['post'].map({0: 'Pré', 1: 'Pós'}),
    painel['alta_exp'].map({0: 'Controle', 1: 'Tratamento'}),
    margins=True
)
print(f"\nTabela de contingência (2d, principal):")
print(ct)
```

```
Thresholds de exposição (2d, PRINCIPAL):
  alta_exp_10: 0.4433 (78 ocupações, 12%)
  alta_exp: 0.3854 (131 ocupações, 21%)
  alta_exp_25: 0.3725 (165 ocupações, 26%)
  alta_exp_mediana: 0.2459 (332 ocupações, 53%)

Threshold 4d (p80): 0.3863 (137 ocupações)

--- Distribuição de tratamento ---
  Alta exp 2d (top 20%): 20.3% das obs
  Alta exp 4d (top 20%): 21.3% das obs
  Períodos pré:  14,058
  Períodos pós:  18,930
  Concordância 2d vs 4d: 93.1%

Tabela de contingência (2d, principal):
alta_exp  Controle  Tratamento    All
post
```

```
Pré          11195        2863  14058
Pós          15092        3838  18930
All          26287        6701  32988
```

0.14 5b. Verificar tratamento (CHECKPOINT)

Validar a definição de tratamento: top/bottom ocupações por exposição, distribuição por quintil, e concordância entre especificações 2d e 4d.

```
# Etapa 2a.5b — CHECKPOINT: Verificar definição de tratamento

print("=" * 60)
print("CHECKPOINT — Definição de Tratamento")
print("=" * 60)

# 1. Top 10 ocupações mais expostas
print("\n--- Top 10 ocupações MAIS expostas ---")
top10 = painel.groupby('cbo_4d').agg(
    exposure=('exposure_score', 'first'),
    admissoes_total=('admissoes', 'sum'),
).nlargest(10, 'exposure')
for cbo, row in top10.iterrows():
    nome = GRANDES_GRUPOS_CBO.get(cbo[0], '')
    print(f"  CBO {cbo}: score={row['exposure']:.3f}, admissões={row['admissoes_total'

# 2. Bottom 10 ocupações menos expostas
print(f"\n--- 10 ocupações MENOS expostas ---")
bot10 = painel.groupby('cbo_4d').agg(
    exposure=('exposure_score', 'first'),
    admissoes_total=('admissoes', 'sum'),
).nsmallest(10, 'exposure')
for cbo, row in bot10.iterrows():
    nome = GRANDES_GRUPOS_CBO.get(cbo[0], '')
    print(f"  CBO {cbo}: score={row['exposure']:.3f}, admissões={row['admissoes_total'

# 3. Distribuição por quintil
print(f"\n--- Estatísticas por quintil de exposição ---")
for q in ['Q1 (Baixa)', 'Q2', 'Q3', 'Q4', 'Q5 (Alta)']:
    sub = painel[painel['quintil_exp'] == q]
    if len(sub) > 0:
        print(f"  {q}: n={len(sub):,}, "
              f"exposure={sub['exposure_score'].mean():.3f}, "
              f"adm_mean={sub['admissoes'].mean():.0f}, "
              f"sal_medio={sub['salario_medio_adm'].mean():,.0f}")

# 4. Concordância
concordancia = (painel['alta_exp'] == painel['alta_exp_4d']).mean()
print(f"\n--- Concordância 2d vs 4d: {concordancia:.1%} ---")

print(f"\n{'=' * 60}")
print(f"CHECKPOINT CONCLUÍDO")
print(f"{'=' * 60}")
```

```
============================================================
CHECKPOINT — Definição de Tratamento
============================================================

--- Top 10 ocupações MAIS expostas ---
  CBO 4101: score=0.632, admissões=343,660  (Trabalhadores de serv. admin.)
  CBO 4102: score=0.632, admissões=109,397  (Trabalhadores de serv. admin.)
  CBO 4110: score=0.632, admissões=7,169,112  (Trabalhadores de serv. admin.)
  CBO 4121: score=0.632, admissões=36,099  (Trabalhadores de serv. admin.)
  CBO 4122: score=0.632, admissões=217,727  (Trabalhadores de serv. admin.)
  CBO 4131: score=0.632, admissões=521,115  (Trabalhadores de serv. admin.)
  CBO 4132: score=0.632, admissões=178,605  (Trabalhadores de serv. admin.)
  CBO 4141: score=0.632, admissões=4,019,302  (Trabalhadores de serv. admin.)
  CBO 4142: score=0.632, admissões=406,552  (Trabalhadores de serv. admin.)
  CBO 4151: score=0.632, admissões=36,816  (Trabalhadores de serv. admin.)

--- 10 ocupações MENOS expostas ---
  CBO 9101: score=0.117, admissões=44,866  (Manutenção e reparação)
  CBO 9102: score=0.117, admissões=13,108  (Manutenção e reparação)
  CBO 9109: score=0.117, admissões=1,394  (Manutenção e reparação)
  CBO 9111: score=0.117, admissões=38,881  (Manutenção e reparação)
  CBO 9112: score=0.117, admissões=83,226  (Manutenção e reparação)
  CBO 9113: score=0.117, admissões=489,020  (Manutenção e reparação)
  CBO 9131: score=0.117, admissões=85,145  (Manutenção e reparação)
  CBO 9141: score=0.117, admissões=9,542  (Manutenção e reparação)
  CBO 9142: score=0.117, admissões=1,889  (Manutenção e reparação)
  CBO 9143: score=0.117, admissões=7,158  (Manutenção e reparação)

--- Estatísticas por quintil de exposição ---
  Q1 (Baixa): n=6,598, exposure=0.144, adm_mean=3200, sal_medio=4,469
  Q2: n=6,597, exposure=0.180, adm_mean=2207, sal_medio=2,599
  Q3: n=6,598, exposure=0.252, adm_mean=3676, sal_medio=7,018
  Q4: n=6,597, exposure=0.348, adm_mean=2735, sal_medio=8,072
  Q5 (Alta): n=6,598, exposure=0.466, adm_mean=3950, sal_medio=8,497

--- Concordância 2d vs 4d: 93.1% ---

============================================================
CHECKPOINT CONCLUÍDO
============================================================
```

0.15 6a. Enriquecimento do painel (variáveis adicionais)

Adicionar variáveis de controle e contexto temporal ao painel para a análise DiD.

Variável	Cálculo	Uso
`tempo_relativo_meses`	Meses desde Dez/2022 (t=0)	Event study
`trend`	Tendência linear (0, 1, 2, ...)	Controle de tendência

Variável	Cálculo	Uso
`mes_do_ano`	Mês do ano (1-12)	Dummies de sazonalidade
`salario_sm`	`salario_medio_adm / SM do ano`	Normalização salarial
`grande_grupo_nome`	Nome do grande grupo CBO	Efeitos fixos

```
# Etapa 2a.6a — Enriquecimento do painel

def periodo_num_to_months(pn):
    """Converter periodo_num (YYYYMM) para contagem absoluta de meses."""
    return (pn // 100) * 12 + (pn % 100)

# ── Tempo relativo ao tratamento ──
ref_periodo = ANO_TRATAMENTO * 100 + MES_TRATAMENTO
painel['meses_abs'] = painel['periodo_num'].apply(periodo_num_to_months)
ref_meses = periodo_num_to_months(ref_periodo)
painel['tempo_relativo_meses'] = painel['meses_abs'] - ref_meses

print(f"Tempo relativo: [{painel['tempo_relativo_meses'].min()}, "
      f"{painel['tempo_relativo_meses'].max()}] meses")
print(f"Referência (t=0): {MES_TRATAMENTO}/{ANO_TRATAMENTO}")

# ── Tendência temporal e sazonalidade ──
painel['trend'] = painel['meses_abs'] - painel['meses_abs'].min()
painel['mes_do_ano'] = painel['mes'].astype(int)

# ── Normalização salarial ──
painel['sm_ano'] = painel['ano'].astype(int).map(SALARIO_MINIMO)
painel['salario_sm'] = painel['salario_medio_adm'] / painel['sm_ano']
painel['ln_salario_sm'] = np.log(painel['salario_sm'].clip(lower=0.1))

# ── Salário real (deflacionado pelo IPCA, base Dez/2024 = 100) ──
painel = painel.merge(df_ipca[['ano', 'mes', 'indice']], on=['ano', 'mes'], how='left'
painel['salario_real_adm'] = painel['salario_medio_adm'] * (INDICE_BASE / painel['indi
painel['ln_salario_real_adm'] = np.log(painel['salario_real_adm'].clip(lower=1))

# ── Grande grupo ocupacional ──
painel['grande_grupo_cbo'] = painel['cbo_4d'].str[0]
painel['grande_grupo_nome'] = painel['grande_grupo_cbo'].map(GRANDES_GRUPOS_CBO)

print(f"\nVariáveis adicionadas: tempo_relativo_meses, trend, mes_do_ano, "
      f"salario_sm, ln_salario_sm, salario_real_adm, ln_salario_real_adm, grande_grupo
print(f"Colunas totais: {painel.shape[1]}")
```

```
Tempo relativo: [-23, 30] meses
Referência (t=0): 12/2022

Variáveis adicionadas: tempo_relativo_meses, trend, mes_do_ano, salario_sm,
```

```
ln_salario_sm, salario_real_adm, ln_salario_real_adm, grande_grupo_nome
Colunas totais: 74
```

0.16 Anexo 1: Integração Anthropic (Automation vs Augmentation)

Integração do **Anthropic Economic Index** (Brynjolfsson et al., 2025 — "Canaries in the Coal Mine") para diferenciar ocupações onde a IA atua como **Automação** (substitui trabalho) vs **Augmentação** (complementa). O índice é construído a partir dos modos de colaboração (directive, feedback loop = automação; learning, task iteration, validation = augmentação) e mapeado para CBO 4d via SOC → ISCO-08 → COD, com imputação hierárquica para cobertura total.

Arquivos necessários em `data/input` (copiar das pastas do repositório se não existirem):

Arquivo	Origem no repositório
`aei_raw_claude_ai_2025-11-13_to_2025-11-20.csv`	`EconomicIndex/release_2026_01_15/data/intermediate/` ou `etapa2_anthropic_index/`
`onet_task_statements.csv`	`EconomicIndex/release_2025_09_15/data/intermediate/`
`Crosswalk SOC 2010 a 2018.xlsx`	`etapa3_crosswalk_onet_isco08/data_input/`
`Crosswalk SOC 2010 ISCO-08.xls`	`etapa3_crosswalk_onet_isco08/data_input/`
`Estrutura Ocupação COD.xls`	`etapa1_ia_generativa/data/raw/` ou `etapa3_crosswalk_onet_isco08/data_input/`

Se `anthropic_automation_augmentation_cbo.parquet` existir em `data/processed` e `KEEP_ANTHROPIC_INDEX = True`, o processamento pesado é pulado e o cache é carregado.

```
# Anexo 1 — Copiar inputs para data/input (se ainda não estiverem) para notebook autoc
import shutil
# Raiz do projeto: se estamos em notebook/, sobe um nível
_cwd = Path(".").resolve()
REPO_ROOT = _cwd.parent if (_cwd / "etapa3_crosswalk_onet_isco08").exists() == False e
origins = [
    (REPO_ROOT / "EconomicIndex" / "release_2026_01_15" / "data" / "intermediate" / "a
    (REPO_ROOT / "EconomicIndex" / "release_2025_09_15" / "data" / "intermediate" / "o
    (REPO_ROOT / "etapa3_crosswalk_onet_isco08" / "data_input" / "Crosswalk SOC 2010 a
    (REPO_ROOT / "etapa3_crosswalk_onet_isco08" / "data_input" / "Crosswalk SOC 2010 I
    (REPO_ROOT / "etapa1_ia_generativa" / "data" / "raw" / "Estrutura Ocupação COD.xls
]
for src, dst in origins:
    if src.exists() and (not dst.exists() or dst.stat().st_size == 0):
        shutil.copy2(src, dst)
        print(f"Copiado: {dst.name}")
    elif not src.exists() and dst.exists():
        print(f"Já em data/input: {dst.name}")
    elif not src.exists():
```

```
        print(f"AVISO: não encontrado no repo: {src.relative_to(REPO_ROOT) if src.is_r
print("Inputs em data/input conferidos.")
```

```
Copiado: aei_raw_claude_ai_2025-11-13_to_2025-11-20.csv
Copiado: onet_task_statements.csv
Copiado: Crosswalk SOC 2010 a 2018.xlsx
Copiado: Crosswalk SOC 2010 ISCO-08.xls
Inputs em data/input conferidos.
```

```
# Anexo 1 — Carregar cache ou rodar pipeline Anthropic (Automation vs Augmentation)
def _weighted_mean(values, weights):
    mask = ~(pd.isna(values) | pd.isna(weights))
    if mask.sum() == 0: return np.nan
    v, w = values[mask], weights[mask]
    return np.average(v, weights=w) if w.sum() > 0 else v.mean()

def _calculate_indices(df_task):
    required_modes = ["directive", "feedback loop", "learning", "task iteration", "val
    for m in required_modes:
        if m not in df_task.columns: df_task[m] = 0
    df_task["total_collab"] = df_task[required_modes].sum(axis=1)
    mask = df_task["total_collab"] > 0
    df_task.loc[mask, "automation_share"] = (df_task.loc[mask, "directive"] + df_task.
    df_task.loc[mask, "augmentation_share"] = (df_task.loc[mask, "learning"] + df_task
    df_task["automation_share"] = df_task["automation_share"].fillna(0)
    df_task["augmentation_share"] = df_task["augmentation_share"].fillna(0)
    df_task["automation_index"] = df_task["automation_share"] - df_task["augmentation_
    df_task["dominant_mode"] = np.where(df_task["automation_index"] > 0, "automation",
    return df_task

df_anthropic_cbo = None
if KEEP_ANTHROPIC_INDEX and ANTHROPIC_INDEX_CACHE.exists():
    df_anthropic_cbo = pd.read_parquet(ANTHROPIC_INDEX_CACHE)
    print(f"Índice Anthropic carregado do cache: {ANTHROPIC_INDEX_CACHE.name} ({len(df
else:
    # Pipeline completo
    AEI_RAW = DATA_INPUT / "aei_raw_claude_ai_2025-11-13_to_2025-11-20.csv"
    ONET_TASKS = DATA_INPUT / "onet_task_statements.csv"
    CROSSWALK_10_18 = DATA_INPUT / "Crosswalk SOC 2010 a 2018.xlsx"
    CROSSWALK_ISCO = DATA_INPUT / "Crosswalk SOC 2010 ISCO-08.xls"
    ESTRUTURA_COD = DATA_INPUT / "Estrutura Ocupação COD.xls"
    if not AEI_RAW.exists() or not ONET_TASKS.exists():
        raise FileNotFoundError("Coloque aei_raw_claude_ai_*.csv e onet_task_statement
    # 1) Carregar Anthropic + O*NET
    df_raw = pd.read_csv(AEI_RAW)
    mask = (df_raw["facet"] == "onet_task::collaboration") & (df_raw["geo_id"] == "GLO
    df_collab = df_raw[mask].copy()
    df_collab[["task_desc", "mode"]] = df_collab["cluster_name"].str.rsplit("::", n=1,
    df_collab["task_desc_clean"] = df_collab["task_desc"].str.strip().str.lower().str.
    df_pivot = df_collab.pivot_table(index="task_desc_clean", columns="mode", values="
    df_onet = pd.read_csv(ONET_TASKS)
    df_onet["task_desc_clean"] = df_onet["Task"].str.strip().str.lower().str.rstrip(".
    task_mapping = df_onet[["task_desc_clean", "O*NET-SOC Code"]].drop_duplicates()
```

```
    task_mapping.columns = ["task_desc_clean", "onet_soc_code"]
    df_pivot = df_pivot.merge(task_mapping, on="task_desc_clean", how="inner")
    mask_count = (df_raw["facet"] == "onet_task") & (df_raw["variable"] == "onet_task_
    df_counts = df_raw[mask_count][["cluster_name", "value"]].rename(columns={"cluster
    df_counts["task_desc_clean"] = df_counts["task_desc"].str.strip().str.lower().str.
    df_counts = df_counts.groupby("task_desc_clean")["usage_volume"].sum().reset_index
    df_task = df_pivot.merge(df_counts, on="task_desc_clean", how="inner")
    df_task = _calculate_indices(df_task)
    df_onet["soc_6d"] = df_onet["O*NET-SOC Code"].str.split(".").str[0]
    soc_mapping = df_onet[["O*NET-SOC Code", "soc_6d", "Title"]].drop_duplicates()
    soc_mapping.columns = ["onet_soc_code", "soc_6d", "occupation_title"]
    df_merged = df_task.merge(soc_mapping, on="onet_soc_code", how="inner")
    def agg_wm(x): return _weighted_mean(x, df_merged.loc[x.index, "usage_volume"])
    df_soc = df_merged.groupby(["soc_6d", "occupation_title"]).agg(
        automation_share=("automation_share", agg_wm), augmentation_share=("augmentati
    ).reset_index()
    df_soc["automation_index_cai"] = df_soc["automation_share"] - df_soc["augmentation
    df_soc["automation_share_cai"] = df_soc["automation_share"]
    df_soc["augmentation_share_cai"] = df_soc["augmentation_share"]
    df_soc.to_csv(DATA_PROCESSED / "onet_automation_augmentation_index.csv", index=Fal
    # 2) Crosswalk SOC -> ISCO-08 (fallback SOC 2018 -> 2010 -> ISCO)
    df_10_18 = pd.read_excel(CROSSWALK_10_18, skiprows=7).iloc[:, :4]
    df_10_18.columns = ["soc_2010_code", "soc_2010_title", "soc_2018_code", "soc_2018_
    df_10_18["soc_2018_code"] = df_10_18["soc_2018_code"].astype(str).str.replace(".00
    df_10_18 = df_10_18.dropna(subset=["soc_2010_code", "soc_2018_code"])
    df_soc_isco = pd.read_excel(CROSSWALK_ISCO, sheet_name="2010 SOC to ISCO-08", skip
    df_soc_isco.columns = ["soc_2010_code", "soc_2010_title", "part", "isco_08_code",
    df_soc_isco = df_soc_isco.dropna(subset=["soc_2010_code", "isco_08_code"])
    df_soc_isco["isco_08_code"] = df_soc_isco["isco_08_code"].astype(str).str.replace(
    df_merged = df_soc.merge(df_10_18, left_on="soc_6d", right_on="soc_2018_code", how
    def agg_isco(x): return _weighted_mean(x, df_merged.loc[x.index, "usage_volume"])
    df_isco = df_merged.groupby("isco_08_code").agg(
        automation_share_cai=("automation_share_cai", agg_isco), augmentation_share_ca
        automation_index_cai=("automation_index_cai", agg_isco), usage_volume=("usage_
    ).reset_index()
    df_isco["dominant_mode_cai"] = np.where(df_isco["automation_index_cai"] > 0, "auto
    df_isco.to_csv(DATA_PROCESSED / "isco_automation_augmentation_index.csv", index=Fa
```

```
# Anexo 1 — Continuação: ISCO -> COD, imputação hierárquica, salvar cache (só roda se
if df_anthropic_cbo is None:
    try:
        df_esco = pd.read_csv(DATA_PROCESSED / "isco_automation_augmentation_index.csv
    except Exception:
        raise RuntimeError("Execute a célula anterior do Anexo 1 primeiro.")
    ESTRUTURA_COD = DATA_INPUT / "Estrutura Ocupação COD.xls"
    if not ESTRUTURA_COD.exists():
        raise FileNotFoundError("Coloque Estrutura Ocupação COD.xls em data/input.")
    try:
        df_cod = pd.read_excel(ESTRUTURA_COD, sheet_name="Estrutura COD", engine="xlrd
    except Exception:
        df_cod = pd.read_excel(ESTRUTURA_COD, sheet_name="Estrutura COD", engine="open
    df_cod.columns = ["grande_grupo", "subgrupo_principal", "subgrupo", "grupo_base",
    df_gb = df_cod.dropna(subset=["grupo_base"])[["grupo_base", "denominacao"]].copy()
```

```
    df_gb["cod_cod"] = df_gb["grupo_base"].astype(str).str.replace(".0", "", regex=Fal
    df_esco["isco_08_code"] = df_esco["isco_08_code"].astype(str).str.zfill(4)
    df_cod_merge = df_gb[["cod_cod", "denominacao"]].merge(df_esco, left_on="cod_cod",
    metrics_cols = ["automation_share_cai", "augmentation_share_cai", "automation_inde
    df_cod_merge["imputation_method"] = np.where(df_cod_merge["automation_index_cai"].
    df_isco = df_esco.copy()
    df_isco["isco_3d"] = df_isco["isco_08_code"].astype(str).str[:3]
    df_isco["isco_2d"] = df_isco["isco_08_code"].astype(str).str[:2]
    if "usage_volume" not in df_isco.columns: df_isco["usage_volume"] = 1.0
    avg_3d = df_isco.groupby("isco_3d").apply(lambda g: pd.Series({c: _weighted_mean(g
    avg_2d = df_isco.groupby("isco_2d").apply(lambda g: pd.Series({c: _weighted_mean(g
    df_cod_merge["cod_3d"] = df_cod_merge["cod_cod"].astype(str).str[:3]
    df_cod_merge["cod_2d"] = df_cod_merge["cod_cod"].astype(str).str[:2]
    mask_miss = df_cod_merge["automation_index_cai"].isna()
    for idx in df_cod_merge[mask_miss].index:
        c3 = df_cod_merge.loc[idx, "cod_3d"]
        row3 = avg_3d[avg_3d["isco_3d"] == c3]
        if len(row3) > 0:
            for c in metrics_cols:
                if c in row3.columns: df_cod_merge.loc[idx, c] = row3[c].iloc[0]
            df_cod_merge.loc[idx, "imputation_method"] = "hierarchical_3d_mean"
    mask_miss = df_cod_merge["automation_index_cai"].isna()
    for idx in df_cod_merge[mask_miss].index:
        c2 = df_cod_merge.loc[idx, "cod_2d"]
        row2 = avg_2d[avg_2d["isco_2d"] == c2]
        if len(row2) > 0:
            for c in metrics_cols:
                if c in row2.columns: df_cod_merge.loc[idx, c] = row2[c].iloc[0]
            df_cod_merge.loc[idx, "imputation_method"] = "hierarchical_2d_mean"
    mask_still = df_cod_merge["automation_index_cai"].isna()
    df_cod_merge.loc[mask_still, metrics_cols] = 0.0
    df_cod_merge.loc[mask_still, "imputation_method"] = "zero_imputation_no_data"
    df_cod_merge["dominant_mode_cai"] = np.where(df_cod_merge["automation_index_cai"]
    df_cod_merge.to_csv(DATA_PROCESSED / "cod_automation_augmentation_index_final.csv"
    df_anthropic_cbo = df_cod_merge[["cod_cod", "automation_share_cai", "augmentation_
    df_anthropic_cbo = df_anthropic_cbo.rename(columns={"cod_cod": "cbo_4d", "automati
    df_anthropic_cbo["cbo_4d"] = df_anthropic_cbo["cbo_4d"].astype(str).str.zfill(4)
    df_anthropic_cbo.to_parquet(ANTHROPIC_INDEX_CACHE, index=False)
    print(f"Pipeline Anthropic concluído. Cache salvo: {ANTHROPIC_INDEX_CACHE}. Ocupaç
```

```
Pipeline Anthropic concluído. Cache salvo:
data/processed/anthropic_automation_augmentation_cbo.parquet. Ocupações: 435.
```

```
# Anexo 1 — Merge do índice Anthropic ao painel e criação das dummies (automation vs a
# Garantir formato cbo_4d string 4 dígitos para o merge
_cbo = painel["cbo_4d"].astype(str).str.zfill(4)
df_anthropic_cbo["cbo_4d"] = df_anthropic_cbo["cbo_4d"].astype(str).str.zfill(4)
cols_merge = [c for c in ["anthropic_automation_index", "automation_share_cai", "augme
idx_cbo = df_anthropic_cbo.set_index("cbo_4d")[cols_merge]
painel = painel.copy()
for c in cols_merge:
    painel[c] = _cbo.map(idx_cbo[c] if c in idx_cbo.columns else idx_cbo.squeeze()).va
# Score contínuo: preencher NaN com 0 (imputação hierárquica já cobriu todas as CBOs n
```

```
painel["anthropic_automation_index"] = painel["anthropic_automation_index"].fillna(0.0
# Dummies: Automação = 1 quando automation > augmentation (índice > 0); Augmentação =
painel["is_automation"] = (painel["anthropic_automation_index"] > 0).astype(int)
painel["is_augmentation"] = (painel["anthropic_automation_index"] <= 0).astype(int)
print(f"Anexo 1 — Merge concluído. anthropic_automation_index: cobertura {painel['anth
      f"is_automation={painel['is_automation'].mean():.1%}, is_augmentation={painel['i
```

```
Anexo 1 — Merge concluído. anthropic_automation_index: cobertura 100.0%.
is_automation=5.4%, is_augmentation=94.6%
```

0.17 7. Salvar dataset analítico final

Selecionar colunas finais, remover ocupações sem score de exposição e salvar o dataset pronto para a análise DiD (Notebook 2b).

Saída: - `data/output/painel_caged_did_ready.parquet` — formato eficiente para análise - `data/output/painel_caged_did_ready.csv` — backup legível

```
# Etapa 2a.7 — Salvar dataset analítico final

if not KEEP_PANEL_FINAL:
    if PAINEL_FINAL_PARQUET.exists():
        PAINEL_FINAL_PARQUET.unlink()
        print(f"Cache removido: {PAINEL_FINAL_PARQUET.name}")
    if PAINEL_FINAL_CSV.exists():
        PAINEL_FINAL_CSV.unlink()
        print(f"Cache removido: {PAINEL_FINAL_CSV.name}")

# ── Selecionar colunas finais ──
cols_finais = [
    # Identificação
    'cbo_4d', 'cbo_2d', 'ano', 'mes', 'periodo', 'periodo_num',
    # Outcomes
    'admissoes', 'desligamentos', 'saldo', 'n_movimentacoes',
    'ln_admissoes', 'ln_desligamentos',
    'salario_medio_adm', 'salario_mediano_adm', 'salario_medio_desl',
    'ln_salario_adm', 'salario_sm', 'ln_salario_sm',
    'salario_real_adm', 'ln_salario_real_adm',
    # Demografia das admissões (proporções)
    'idade_media_adm', 'pct_mulher_adm', 'pct_superior_adm', 'pct_branco_adm', 'pct_ne
    'pct_tecnologico_adm', 'setor_tecnologico',
    # Heterogeneidade: salários (log)
    'ln_salario_homem', 'ln_salario_mulher', 'ln_salario_jovem', 'ln_salario_naojovem'
    'ln_salario_branco', 'ln_salario_negro', 'ln_salario_superior', 'ln_salario_medio'
    # Heterogeneidade: volumes (log)
    'ln_admissoes_homem', 'ln_admissoes_mulher', 'ln_admissoes_jovem', 'ln_admissoes_n
    # Exposição IA — DUAL
    'exposure_score_2d',   # PRINCIPAL
    'exposure_score_4d',   # ROBUSTEZ
    # Tratamento — DUAL
    'alta_exp',            # Top 20% score 2d (PRINCIPAL)
    'alta_exp_10', 'alta_exp_25', 'alta_exp_mediana', 'quintil_exp',
    'alta_exp_4d',         # Top 20% score 4d (ROBUSTEZ)
```

```
    # Temporal
    'post', 'did', 'did_4d', 'tempo_relativo_meses', 'trend', 'mes_do_ano',
    # Classificação
    'grande_grupo_cbo', 'grande_grupo_nome',
    # Anthropic (Automation vs Augmentation) — Anexo 1
    'anthropic_automation_index', 'is_automation', 'is_augmentation',
]

cols_existentes = [c for c in cols_finais if c in painel.columns]
cols_faltantes = [c for c in cols_finais if c not in painel.columns]
if cols_faltantes:
    print(f"AVISO: Colunas não encontradas: {cols_faltantes}")

painel_final = painel[cols_existentes].copy()

# ── Remover ocupações sem score principal (2d) ──
n_antes = len(painel_final)
painel_final = painel_final[painel_final['exposure_score_2d'].notna()]
n_depois = len(painel_final)
if n_antes > n_depois:
    print(f"Removidas {n_antes - n_depois:,} linhas sem exposure_score_2d")

# ── Verificação para o Notebook 2b ──
cols_obrigatorias = ['exposure_score_2d', 'exposure_score_4d', 'alta_exp', 'did', 'tem
for c in cols_obrigatorias:
    if c not in painel_final.columns:
        raise ValueError(f"Coluna obrigatória ausente para o DiD (Notebook 2b): {c}")
if painel_final[cols_obrigatorias].isna().any().any():
    raise ValueError("NA em coluna obrigatória para o DiD (Notebook 2b).")
print("  Verificação: colunas obrigatórias para o 2b presentes e sem NA.")

# ── Salvar ──
painel_final.to_parquet(PAINEL_FINAL_PARQUET, index=False)
painel_final.to_csv(PAINEL_FINAL_CSV, index=False)

# ═══════════════════════════════════════════════════════════════════
# RESUMO FINAL
# ═══════════════════════════════════════════════════════════════════
print(f"\n{'=' * 60}")
print("DATASET ANALÍTICO FINAL — ETAPA 2a")
print(f"{'=' * 60}")
print(f"  Observações:        {len(painel_final):,}")
print(f"  Ocupações (CBO 4d): {painel_final['cbo_4d'].nunique()}")
print(f"  Períodos:           {painel_final['periodo'].nunique()} meses")
print(f"    Pré-tratamento:   {painel_final[painel_final['post']==0]['periodo'].nuniqu
print(f"    Pós-tratamento:   {painel_final[painel_final['post']==1]['periodo'].nuniqu
print(f"  Cobertura 2d:       {painel_final['exposure_score_2d'].notna().mean():.1%}")
print(f"  Cobertura 4d:       {painel_final['exposure_score_4d'].notna().mean():.1%}")
print(f"  Tratamento 2d:      {painel_final['alta_exp'].mean():.1%} das obs")
print(f"  Tratamento 4d:      {painel_final['alta_exp_4d'].mean():.1%} das obs")
print(f"  Colunas:            {painel_final.shape[1]}")
print(f"\n  Salvo em:")
print(f"    {PAINEL_FINAL_PARQUET}")
print(f"    {PAINEL_FINAL_CSV}")
```

```
pq_mb = PAINEL_FINAL_PARQUET.stat().st_size / 1e6
csv_mb = PAINEL_FINAL_CSV.stat().st_size / 1e6
print(f"    Tamanho: {pq_mb:.1f} MB (parquet), {csv_mb:.1f} MB (csv)")

print(f"\n  Info:")
painel_final.info()
```

```
Cache removido: painel_caged_did_ready.parquet
Cache removido: painel_caged_did_ready.csv
  Verificação: colunas obrigatórias para o 2b presentes e sem NA.

============================================================
DATASET ANALÍTICO FINAL — ETAPA 2a
============================================================
  Observações:        32,988
  Ocupações (CBO 4d): 629
  Períodos:           54 meses
    Pré-tratamento:   23
    Pós-tratamento:   31
  Cobertura 2d:       100.0%
  Cobertura 4d:       100.0%
  Tratamento 2d:      20.3% das obs
  Tratamento 4d:      21.3% das obs
  Colunas:            59

  Salvo em:
    data/output/painel_caged_did_ready.parquet
    data/output/painel_caged_did_ready.csv
    Tamanho: 7.2 MB (parquet), 21.0 MB (csv)

  Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32988 entries, 0 to 32987
Data columns (total 59 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   cbo_4d                      32988 non-null  object
 1   cbo_2d                      32988 non-null  object
 2   ano                         32988 non-null  Int64
 3   mes                         32988 non-null  Int64
 4   periodo                     32988 non-null  object
 5   periodo_num                 32988 non-null  int64
 6   admissoes                   32988 non-null  Int64
 7   desligamentos               32988 non-null  Int64
 8   saldo                       32988 non-null  Int64
 9   n_movimentacoes             32988 non-null  Int64
 10  ln_admissoes                32988 non-null  Float64
 11  ln_desligamentos            32988 non-null  Float64
 12  salario_medio_adm           32988 non-null  float64
 13  salario_mediano_adm         32988 non-null  float64
 14  salario_medio_desl          32988 non-null  float64
 15  ln_salario_adm              32988 non-null  float64
 16  salario_sm                  32988 non-null  float64
 17  ln_salario_sm               32988 non-null  float64
```

```
 18  salario_real_adm            32988 non-null  float64
 19  ln_salario_real_adm         32988 non-null  float64
 20  idade_media_adm             32988 non-null  Float64
 21  pct_mulher_adm              32988 non-null  float64
 22  pct_superior_adm            32988 non-null  float64
 23  pct_branco_adm              32988 non-null  float64
 24  pct_negro_adm               32988 non-null  float64
 25  pct_jovem_adm               32988 non-null  float64
 26  pct_tecnologico_adm         32988 non-null  float64
 27  setor_tecnologico           32988 non-null  int64
 28  ln_salario_homem            32988 non-null  float64
 29  ln_salario_mulher           32988 non-null  float64
 30  ln_salario_jovem            32988 non-null  float64
 31  ln_salario_naojovem         32988 non-null  float64
 32  ln_salario_branco           32988 non-null  float64
 33  ln_salario_negro            32988 non-null  float64
 34  ln_salario_superior         32988 non-null  float64
 35  ln_salario_medio            32988 non-null  float64
 36  ln_admissoes_homem          32988 non-null  float64
 37  ln_admissoes_mulher         32988 non-null  float64
 38  ln_admissoes_jovem          32988 non-null  float64
 39  ln_admissoes_negro          32988 non-null  float64
 40  exposure_score_2d           32988 non-null  float64
 41  exposure_score_4d           32988 non-null  float64
 42  alta_exp                    32988 non-null  int64
 43  alta_exp_10                 32988 non-null  int64
 44  alta_exp_25                 32988 non-null  int64
 45  alta_exp_mediana            32988 non-null  int64
 46  quintil_exp                 32988 non-null  category
 47  alta_exp_4d                 32988 non-null  int64
 48  post                        32988 non-null  int64
 49  did                         32988 non-null  int64
 50  did_4d                      32988 non-null  int64
 51  tempo_relativo_meses        32988 non-null  int64
 52  trend                       32988 non-null  int64
 53  mes_do_ano                  32988 non-null  int64
 54  grande_grupo_cbo            32988 non-null  object
 55  grande_grupo_nome           32988 non-null  object
 56  anthropic_automation_index  32988 non-null  float64
 57  is_automation               32988 non-null  int64
 58  is_augmentation             32988 non-null  int64
dtypes: Float64(3), Int64(6), category(1), float64(29), int64(15), object(5)
memory usage: 14.9+ MB
```

0.18 Limitações desta etapa

1. **Novo CAGED (descontinuidade 2020):** A transição para o eSocial (2020) pode afetar a comparabilidade. Mitigamos ao iniciar em 2021 (eSocial já estabilizado, sem efeitos COVID).

2. **Fluxos vs. estoques:** O CAGED mede movimentações (admissões/desligamentos), não o estoque de empregados. Quedas em admissões não significam necessariamente queda no emprego total — podem refletir menor rotatividade. Esta é a mesma lógica usada por Hui et al. (2024) com dados do Upwork.

3. **Crosswalk CBO → ISCO-08 (especificação principal, 2 dígitos):** Ao agregar por Sub-major Group com fallback a Major Group, perdemos variação intragrupo. Ocupações diferentes dentro do mesmo grupo recebem o mesmo score. A especificação de robustez a 4 dígitos (com fallback hierárquico em 6 níveis) ajuda a avaliar se essa agregação afeta os resultados.

4. **Crosswalk CBO → ISCO-08 (robustez, 4 dígitos):** O match direto CBO 4d = ISCO-08 4d cobre apenas ~28% das ocupações. Para o restante, usamos fallback hierárquico (via correspondência ISCO-88→ISCO-08, médias a 3d, 2d e 1d). Quanto mais granular o match, mais preciso o score — mas mesmo com fallback, a correlação entre as especificações 2d e 4d é >0.91, indicando consistência. Erro de medição no tratamento tipicamente atenua os coeficientes (viés em direção a zero).

5. **Muendler: CBO 1994, não CBO 2002:** O arquivo de concordância Muendler & Poole (2004) mapeia a CBO *1994* (formato X-XX.XX), não a CBO 2002 (XXXX) usada no CAGED. A utilidade do Muendler para match 4d direto é limitada. A estratégia adotada usa a similaridade estrutural entre CBO 2002 e ISCO-08/88 (ambas baseadas na ISCO), combinada com a tabela oficial de correspondência ISCO-08↔ISCO-88.

6. **Emprego formal apenas:** O CAGED cobre apenas o mercado formal (CLT). A informalidade (~40% da força de trabalho brasileira) não é capturada. Efeitos da IA sobre o setor informal requerem fontes alternativas (PNAD).

7. **Índice global aplicado ao Brasil:** Mesma limitação da Etapa 1 — o índice ILO foi desenvolvido com foco global e pode não capturar especificidades do mercado de trabalho brasileiro.

0.19 Checklist de entregáveis

- [x] `data/raw/caged_{ano}.parquet` — Microdados CAGED por ano (2021–2025)
- [x] `data/input/cbo-isco-conc.csv` — Concordância Muendler CBO 1994→ISCO-88
- [x] `data/input/Correspondência ISCO 08 a 88.xlsx` — Tabela oficial ISCO-08↔ISCO-88
- [x] `data/processed/ilo_exposure_clean.csv` — Índice ILO processado (reusado da Etapa 1)
- [x] `data/output/painel_caged_did_ready.parquet` — Dataset analítico final (com scores 2d e 4d)
- [x] `data/output/painel_caged_did_ready.csv` — Backup CSV
- [x] Todos os CHECKPOINTs passando sem warnings críticos
- [x] Cobertura crosswalk 2d = 100%
- [x] Cobertura crosswalk 4d = 100% (com fallback hierárquico)
- [x] Correlação entre scores 2d e 4d: 0.9147
- [x] Sanity check por grande grupo coerente com a literatura