# Apêndice - Códigos Python Utilizados

## coletar\_links\_inep.py

0001: # scripts/coleta\_dados/coletar\_links\_inep.py

0002: # Este script coleta os links dos microdados do Censo da Educação Superior do site do INEP (1995-2023).

0003:

0004: import requests

0005: from bs4 import BeautifulSoup

0006:

0007: def coletar\_links\_inep(url\_base, palavra\_chave):

0008: """

0009: Coleta os links dos microdados do Censo da Educação Superior do INEP.

0010:

0011: :param url\_base: URL da página do INEP com os dados.

0012: :param palavra\_chave: Palavra-chave que identifica os links desejados.

0013: :return: Dicionário com os anos e os links dos microdados.

0014: """

0015: resposta = requests.get(url\_base)

0016: if resposta.status\_code != 200:

0017: raise Exception(f"Erro ao acessar {url\_base}: {resposta.status\_code}")

0018:

0019: soup = BeautifulSoup(resposta.content, 'html.parser')

0020: links = soup.find\_all('a', href=True)

0021:

0022: urls = {}

0023: for link in links:

0024: href = link['href']

0025: if palavra\_chave in href:

0026: # Extrai o ano do link e cria a entrada no dicionário

0027: for ano in range(1995, 2024):

0028: if str(ano) in href:

0029: urls[f"INEP\_{ano}-MICRODADOS-CENSO"] = href

0030: break

0031: return urls

0032:

0033: def salvar\_links\_em\_arquivo(urls, caminho\_arquivo):

0034: """

0035: Salva os links em um arquivo de texto no formato especificado.

0036:

0037: :param urls: Dicionário com os links e descrições.

0038: :param caminho\_arquivo: Caminho do arquivo onde os links serão salvos.

0039: """

0040: with open(caminho\_arquivo, 'w') as arquivo:

0041: for chave, url in urls.items():

0042: arquivo.write(f"{chave}: {url}\n")

0043: print(f"Links salvos em {caminho\_arquivo}")

0044:

0045: def main():

0046: url\_base = "https://www.gov.br/inep/pt-br/acesso-a-informacao/dados-abertos/microdados/censo-da-educacao-superior"

0047: palavra\_chave = "microdados\_censo\_da\_educacao\_superior"

0048: caminho\_arquivo = "./dados/bruto/lista-links.txt"

0049:

0050: print("Coletando links do INEP...")

0051: urls = coletar\_links\_inep(url\_base, palavra\_chave)

0052: salvar\_links\_em\_arquivo(urls, caminho\_arquivo)

0053: print("Processo concluído.")

0054:

0055: if \_\_name\_\_ == '\_\_main\_\_':

0056: main()

## coleta\_dados\_oficiais.py

0001: # scripts/coleta\_dados/coleta\_dados\_oficiais.py

0002: # Este script realiza a coleta e processamento de microdados oficiais do INEP e MEC.

0003:

0004: import os

0005: import requests

0006: import pandas as pd

0007: from zipfile import ZipFile

0008:

0009: def baixar\_arquivo(url, destino):

0010: """

0011: Faz o download de um arquivo a partir de uma URL e salva no destino especificado.

0012: """

0013: try:

0014: resposta = requests.get(url, stream=True)

0015: resposta.raise\_for\_status() # Verifica se houve erro na resposta

0016: with open(destino, 'wb') as arquivo:

0017: for chunk in resposta.iter\_content(chunk\_size=8192):

0018: arquivo.write(chunk)

0019: print(f'Download concluído: {destino}')

0020: except requests.RequestException as e:

0021: print(f'Erro ao baixar {url}: {e}')

0022:

0023: def extrair\_arquivo(zip\_caminho, destino\_pasta):

0024: """

0025: Extrai um arquivo ZIP para o destino especificado e ajusta estrutura se necessário.

0026: """

0027: try:

0028: with ZipFile(zip\_caminho, 'r') as zip\_ref:

0029: zip\_ref.extractall(destino\_pasta)

0030: print(f'Arquivos extraídos para: {destino\_pasta}')

0031:

0032: # Verificar estrutura do diretório após extração

0033: for root, dirs, files in os.walk(destino\_pasta):

0034: print(f'Conteúdo de {root}: {dirs}, {files}')

0035: break # Apenas primeiro nível para não poluir o log

0036:

0037: except Exception as e:

0038: print(f'Erro ao extrair {zip\_caminho}: {e}')

0039:

0040: def normalizar\_nomes\_arquivos(caminho\_pasta, ano):

0041: """

0042: Renomeia os arquivos de IES para seguir o formato esperado: MICRODADOS\_ED\_SUP\_IES\_YEAR.CSV.

0043: """

0044: for root, \_, arquivos in os.walk(caminho\_pasta):

0045: for arquivo in arquivos:

0046: if "CADASTRO\_IES" in arquivo.upper() and arquivo.upper().endswith('.CSV'):

0047: caminho\_antigo = os.path.join(root, arquivo)

0048: caminho\_novo = os.path.join(root, f"MICRODADOS\_ED\_SUP\_IES\_{ano}.CSV")

0049: os.rename(caminho\_antigo, caminho\_novo)

0050: print(f'Renomeado: {caminho\_antigo} -> {caminho\_novo}')

0051:

0052: def processar\_microdados(caminho\_pasta):

0053: """

0054: Processa os arquivos de microdados e retorna um DataFrame consolidado.

0055: """

0056: # Normaliza os nomes dos arquivos de IES antes de processar

0057: # Extraindo o ano a partir do nome da pasta (assumindo que a pasta contenha o ano)

0058: ano = os.path.basename(caminho\_pasta).split('\_')[1] if '\_' in os.path.basename(caminho\_pasta) else 'NA'

0059: normalizar\_nomes\_arquivos(caminho\_pasta, ano)

0060: arquivos\_csv = [os.path.join(caminho\_pasta, f) for f in os.listdir(caminho\_pasta) if f.upper().endswith('.CSV')]

0061: df\_lista = []

0062: for arquivo in arquivos\_csv:

0063: try:

0064: df = pd.read\_csv(arquivo, sep=';', encoding='latin1')

0065: df\_lista.append(df)

0066: except Exception as e:

0067: print(f'Erro ao processar {arquivo}: {e}')

0068: if df\_lista:

0069: return pd.concat(df\_lista, ignore\_index=True)

0070: else:

0071: print('Nenhum dado processado.')

0072: return pd.DataFrame()

0073:

0074: def main():

0075: # URLs reais dos microdados

0076: urls = {

0077: 'INEP\_2023-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2023.zip',

0078: 'INEP\_2022-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2022.zip',

0079: 'INEP\_2021-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2021.zip',

0080: 'INEP\_2020-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2020.zip',

0081: 'INEP\_2019-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2019.zip',

0082: 'INEP\_2018-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2018.zip',

0083: 'INEP\_2017-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2017.zip',

0084: 'INEP\_2016-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2016.zip',

0085: 'INEP\_2015-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2015.zip',

0086: 'INEP\_2014-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2014.zip',

0087: 'INEP\_2013-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2013.zip',

0088: 'INEP\_2012-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2012.zip',

0089: 'INEP\_2011-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2011.zip',

0090: 'INEP\_2010-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2010.zip',

0091: 'INEP\_2009-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2009.zip',

0092: # 'INEP\_2008-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2008.zip',

0093: # 'INEP\_2007-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2007.zip',

0094: # 'INEP\_2006-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2006.zip',

0095: # 'INEP\_2005-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2005.zip',

0096: # 'INEP\_2004-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2004.zip',

0097: # 'INEP\_2003-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2003.zip',

0098: # 'INEP\_2002-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2002.zip',

0099: # 'INEP\_2001-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2001.zip',

0100: # 'INEP\_2000-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2000.zip',

0101: # 'INEP\_1999-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_1999.zip',

0102: # 'INEP\_1998-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_1998.zip',

0103: # 'INEP\_1997-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_1997.zip',

0104: # 'INEP\_1996-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_1996.zip',

0105: # 'INEP\_1995-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_1995.zip'

0106: }

0107:

0108: # Caminho para salvar os dados

0109: pasta\_dados = './dados/bruto'

0110: os.makedirs(pasta\_dados, exist\_ok=True)

0111:

0112: for fonte, url in urls.items():

0113: zip\_caminho = os.path.join(pasta\_dados, f'{fonte}.zip')

0114: pasta\_extracao = os.path.join(pasta\_dados, fonte)

0115: os.makedirs(pasta\_extracao, exist\_ok=True)

0116:

0117: # Baixar o arquivo

0118: print(f'Baixando dados da fonte: {fonte}')

0119: baixar\_arquivo(url, zip\_caminho)

0120:

0121: # Extrair o arquivo

0122: print(f'Extraindo dados da fonte: {fonte}')

0123: extrair\_arquivo(zip\_caminho, pasta\_extracao)

0124:

0125: # Processar os dados

0126: print(f'Processando dados da fonte: {fonte}')

0127: df = processar\_microdados(pasta\_extracao)

0128:

0129: # Salvar o DataFrame consolidado

0130: if not df.empty:

0131: df.to\_csv(os.path.join(pasta\_dados, f'{fonte}\_dados\_brutos.csv'), index=False)

0132: print(f'Dados processados e salvos em: {os.path.join(pasta\_dados, f"{fonte}\_dados\_brutos.csv")}')

0133: else:

0134: print(f'Nenhum dado válido processado para {fonte}.')

0135:

0136: if \_\_name\_\_ == '\_\_main\_\_':

0137: main()

## pre\_processamento.py

0001: # pre\_processamento.py - Script para carregar, padronizar e salvar os microdados do INEP/MEC.

0002: # Processa os dados brutos (IES e Cursos) por ano, realizando limpeza, renomeação e formatação padronizada.

0003:

0004: import os

0005: import pandas as pd

0006: import unicodedata

0007: import re

0008: import numpy as np

0009: from io import StringIO

0010: import glob

0011:

0012: # ==========================================================================

0013: # CONFIGURAÇÕES GERAIS

0014: # ==========================================================================

0015:

0016: # Diretórios de entrada e saída (ajuste conforme sua estrutura)

0017: PASTA\_BRUTO = "./dados/bruto"

0018: PASTA\_PROCESSADO = "./dados/processado"

0019:

0020: # ==========================================================================

0021: # COLUNAS DE INTERESSE E MAPEAMENTOS

0022: # ==========================================================================

0023:

0024: # Exemplo de colunas relevantes para a base de IES, conforme dicionário do Censo 2023.

0025: COLUNAS\_IES\_RELEVANTES = [

0026: "CO\_IES",

0027: "NO\_IES",

0028: # "TP\_REDE",

0029: "TP\_CATEGORIA\_ADMINISTRATIVA",

0030: "QT\_DOC\_TOTAL",

0031: "QT\_DOC\_EXE",

0032: "QT\_DOC\_EX\_FEMI",

0033: "QT\_DOC\_EX\_MASC"

0034: # Adicione outras colunas se precisar (p. ex. QT\_TEC\_TOTAL, etc.)

0035: ]

0036:

0037: MAPPING\_IES = {

0038: "CO\_IES": "id\_ies",

0039: "NO\_IES": "nome\_ies",

0040: # "TP\_REDE": "tipo\_rede", # 1 = pública, 2 = privada

0041: "TP\_CATEGORIA\_ADMINISTRATIVA": "cat\_adm", # 1 = Fed, 2 = Est, etc.

0042: "QT\_DOC\_TOTAL": "docentes\_total",

0043: "QT\_DOC\_EXE": "docentes\_exercicio",

0044: "QT\_DOC\_EX\_FEMI": "docentes\_feminino",

0045: "QT\_DOC\_EX\_MASC": "docentes\_masculino"

0046: }

0047:

0048: # Exemplo de colunas relevantes para a base de Cursos, conforme dicionário do Censo 2023.

0049: COLUNAS\_CURSOS\_RELEVANTES = [

0050: "CO\_IES",

0051: "CO\_CURSO",

0052: "NO\_CURSO",

0053: "TP\_MODALIDADE\_ENSINO",

0054: "QT\_CURSO",

0055: "QT\_VG\_TOTAL",

0056: "QT\_INSCRITO\_TOTAL",

0057: "QT\_ING",

0058: "QT\_MAT",

0059: "QT\_CONC",

0060: "QT\_DIPLOMADOS",

0061: "QT\_DIPLO"

0062: # de 1995 a 2008 os concluíntes eram chamados de diplomados. Adicione outras colunas se quiser (e.g. QT\_ING\_FEM, QT\_MAT\_18\_24, etc.)

0063: ]

0064:

0065: MAPPING\_CURSOS = {

0066: "CO\_IES": "id\_ies",

0067: "CO\_CURSO": "id\_curso",

0068: "NO\_CURSO": "nome\_curso",

0069: "TP\_MODALIDADE\_ENSINO": "modalidade\_ensino", # 1=Presencial, 2=EAD

0070: "QT\_CURSO": "numero\_cursos",

0071: "QT\_VG\_TOTAL": "vagas\_totais",

0072: "QT\_INSCRITO\_TOTAL": "inscritos\_totais",

0073: "QT\_ING": "ingressantes",

0074: "QT\_MAT": "matriculados",

0075: "QT\_CONC": "concluintes",

0076: "QT\_DIPLOMADOS": "concluintes",

0077: "QT\_DIPLO": "concluintes"

0078: }

0079:

0080: # # Os mapeamentos específicos para determinados anos (1995, 2000, 2008) são definidos

0081: # # como cópias do mapeamento geral para permitir, futuramente, ajustes pontuais nesses períodos.

0082: # MAPPING\_IES\_1995 = MAPPING\_IES.copy()

0083: # COLUNAS\_IES\_RELEVANTES\_1995 = COLUNAS\_IES\_RELEVANTES.copy()

0084:

0085: # MAPPING\_CURSOS\_1995 = MAPPING\_CURSOS.copy()

0086: # COLUNAS\_CURSOS\_RELEVANTES\_1995 = COLUNAS\_CURSOS\_RELEVANTES.copy()

0087:

0088: # MAPPING\_IES\_2000 = MAPPING\_IES.copy()

0089: # COLUNAS\_IES\_RELEVANTES\_2000 = COLUNAS\_IES\_RELEVANTES.copy()

0090:

0091: # MAPPING\_CURSOS\_2000 = MAPPING\_CURSOS.copy()

0092: # COLUNAS\_CURSOS\_RELEVANTES\_2000 = COLUNAS\_CURSOS\_RELEVANTES.copy()

0093:

0094: # MAPPING\_IES\_2008 = MAPPING\_IES.copy()

0095: # COLUNAS\_IES\_RELEVANTES\_2008 = COLUNAS\_IES\_RELEVANTES.copy()

0096: # MAPPING\_CURSOS\_2008 = MAPPING\_CURSOS.copy()

0097: # COLUNAS\_CURSOS\_RELEVANTES\_2008 = COLUNAS\_CURSOS\_RELEVANTES.copy()

0098:

0099: # ==========================================================================

0100: # FUNÇÕES DE APOIO

0101: # ==========================================================================

0102:

0103: def normalizar\_conteudo\_pipe(conteudo):

0104: """

0105: Remove repetições de delimitadores, espaços desnecessários e caracteres indesejados

0106: para uniformizar o uso do pipe ("|") como delimitador.

0107: """

0108: conteudo = re.sub(r"\|{2,}", "|", conteudo)

0109: conteudo = re.sub(r'\s\*\|\s\*', '|', conteudo)

0110: conteudo = conteudo.replace('\r\n', '\n').replace('\r', '\n')

0111: conteudo = conteudo.replace('"', '')

0112: return conteudo

0113:

0114: def registrar\_problemas(arquivo, erro):

0115: """

0116: Registra problemas de leitura em um log.

0117: """

0118: with open("log\_erros.txt", "a", encoding="utf-8") as log:

0119: log.write(f"Arquivo: {arquivo}, Erro: {erro}\n")

0120:

0121: def carregar\_csv(caminho\_arquivo, sep=";", encoding="latin1", year=None):

0122: """

0123: Carrega um CSV, tratando parsing e erros.

0124: Carrega um arquivo CSV a partir de um caminho, tratando a normalização dos delimitadores.

0125:

0126: Se o ano for menor ou igual a 2008 e o conteúdo apresentar o delimitador "|", tenta

0127: normalizar repetições de delimitadores (por exemplo, '||' ou '|||') para que os dados sejam

0128: lidos corretamente. Se mesmo assim o DataFrame resultar em apenas uma coluna, poderá ser necessário

0129: separar manualmente essa coluna.

0130: """

0131: try:

0132: print(f"Lendo arquivo: {caminho\_arquivo}")

0133: with open(caminho\_arquivo, encoding=encoding) as f:

0134: conteudo = f.read()

0135:

0136: if year is not None and year <= 2008 and "|" in conteudo:

0137: print(f"⚠️ Detecção de separadores múltiplos para ano {year}. Normalizando...")

0138: conteudo = normalizar\_conteudo\_pipe(conteudo)

0139: df = pd.read\_csv(StringIO(conteudo), sep="|", header=0, engine="python", on\_bad\_lines='skip')

0140: if df.shape[1] == 1:

0141: # Se restar somente uma coluna, tenta separar manualmente essa coluna usando o delimitador "|"

0142: print("⚠️ Apenas uma coluna detectada após normalização. Tentando separar manualmente...")

0143: df = df.iloc[:, 0].str.split("|", expand=True)

0144: # Assume-se que a primeira linha são os cabeçalhos

0145: df.columns = df.iloc[0]

0146: df = df[1:]

0147: return df

0148: else:

0149: return pd.read\_csv(StringIO(conteudo), sep=sep, header=0, engine="python", on\_bad\_lines='skip')

0150: except Exception as e:

0151: print(f"Erro ao carregar {caminho\_arquivo}: {e}")

0152: registrar\_problemas(caminho\_arquivo, e)

0153: return pd.DataFrame()

0154:

0155: def filtrar\_renomear(df, colunas\_relevantes, mapping):

0156: """

0157: Seleciona apenas as colunas relevantes contidas no DataFrame e as renomeia conforme

0158: o dicionário de mapeamento fornecido.

0159: """

0160: # Identifica somente colunas que existam no df

0161: existentes = [c for c in colunas\_relevantes if c in df.columns]

0162: df\_filtrado = df[existentes].copy()

0163: # Renomeia

0164: return df\_filtrado.rename(columns=mapping)

0165:

0166: def corrigir\_nome\_pasta(caminho\_base, ano):

0167: """

0168: Corrige possíveis problemas com caracteres especiais nos nomes das pastas extraídas,

0169: retornando o caminho completo da pasta que contenha a string "microdados".

0170: """

0171: caminho\_esperado = os.path.join(caminho\_base, f"INEP\_{ano}-MICRODADOS-CENSO")

0172: if os.path.exists(caminho\_esperado):

0173: for pasta in os.listdir(caminho\_esperado):

0174: pasta\_corrigida = unicodedata.normalize("NFKD", pasta).encode("ASCII", "ignore").decode("ASCII")

0175: pasta\_corrigida = re.sub(r'[^a-zA-Z0-9\_\- ]', '', pasta\_corrigida)

0176: if "microdados" in pasta\_corrigida.lower():

0177: return os.path.join(caminho\_esperado, pasta)

0178: print(f"Aviso: Nenhuma pasta de microdados encontrada para {ano}")

0179: return None

0180:

0181: def corrigir\_nome\_arquivo(nome\_arquivo):

0182: """

0183: Corrige caracteres especiais no nome dos arquivos, removendo aqueles inválidos.

0184: """

0185: return re.sub(r'[^a-zA-Z0-9\_\-\. ]', '', nome\_arquivo)

0186:

0187: # ==========================================================================

0188: # PROCESSAMENTO PRINCIPAL

0189: # ==========================================================================

0190:

0191: def main(year: int = 2024):

0192: arquivos\_disponiveis = []

0193: caminho\_base\_ano = corrigir\_nome\_pasta(PASTA\_BRUTO, year)

0194: caminho\_dados = os.path.join(caminho\_base\_ano, "dados")

0195:

0196: if os.path.isdir(caminho\_dados):

0197: arquivos\_disponiveis = os.listdir(caminho\_dados)

0198:

0199: if year < 2009:

0200: # Renomeia arquivos com padrões diferentes de nomenclatura para padronizá-los

0201: for arq in arquivos\_disponiveis:

0202: if "INSTITUICAO" in arq.upper():

0203: os.rename(

0204: os.path.join(caminho\_dados, arq),

0205: os.path.join(caminho\_dados, f"MICRODADOS\_ED\_SUP\_IES\_{year}.CSV")

0206: )

0207: elif "GRADUACAO\_PRESENCIAL" in arq.upper():

0208: os.rename(

0209: os.path.join(caminho\_dados, arq),

0210: os.path.join(caminho\_dados, f"MICRODADOS\_CADASTRO\_CURSOS\_{year}.CSV")

0211: )

0212: arquivos\_disponiveis = os.listdir(caminho\_dados)

0213:

0214: ARQUIVO\_IES = f"MICRODADOS\_ED\_SUP\_IES\_{year}.CSV"

0215: ARQUIVO\_CURSOS = f"MICRODADOS\_CADASTRO\_CURSOS\_{year}.CSV"

0216:

0217: caminho\_ies = None

0218: for arquivo in arquivos\_disponiveis:

0219: nome\_normalizado = corrigir\_nome\_arquivo(arquivo).upper()

0220: if nome\_normalizado.startswith(f"MICRODADOS\_ED\_SUP\_IES\_{year}") and nome\_normalizado.endswith(".CSV"):

0221: caminho\_ies = os.path.join(caminho\_dados, arquivo)

0222: break

0223:

0224: # Fallback para nome padrão, se não encontrar com sufixo

0225: if caminho\_ies is None:

0226: caminho\_ies = os.path.join(caminho\_dados, f"MICRODADOS\_ED\_SUP\_IES\_{year}.CSV")

0227:

0228: print(f"Arquivos encontrados em {caminho\_dados}: {arquivos\_disponiveis}")

0229: print(f"Arquivo IES esperado: {ARQUIVO\_IES}")

0230: print(f"Arquivo IES identificado: {caminho\_ies if os.path.exists(caminho\_ies) else 'NÃO ENCONTRADO'}")

0231:

0232: df\_ies\_final = pd.DataFrame()

0233: df\_cursos\_final = pd.DataFrame()

0234:

0235: # ----------------------------------------------------------------------

0236: # Carregar e processar MICRODADOS\_ED\_SUP\_IES\_{year}.CSV

0237: # ----------------------------------------------------------------------

0238: if os.path.isfile(caminho\_ies):

0239: df\_ies = carregar\_csv(caminho\_ies, year=year)

0240: if not df\_ies.empty:

0241: df\_ies\_final = filtrar\_renomear(df\_ies, COLUNAS\_IES\_RELEVANTES, MAPPING\_IES)

0242: else:

0243: print(f"Aviso: {ARQUIVO\_IES} está vazio ou não pôde ser processado.")

0244: else:

0245: print(f"Aviso: Arquivo {ARQUIVO\_IES} não encontrado em {caminho\_ies}.")

0246:

0247: # ----------------------------------------------------------------------

0248: # Carregar e processar MICRODADOS\_CADASTRO\_CURSOS\_{year}.CSV

0249: # ----------------------------------------------------------------------

0250: caminho\_cursos = os.path.join(caminho\_dados, corrigir\_nome\_arquivo(ARQUIVO\_CURSOS))

0251: if os.path.isfile(caminho\_cursos):

0252: df\_cursos = carregar\_csv(caminho\_cursos, year=year)

0253: if not df\_cursos.empty:

0254: df\_cursos\_final = filtrar\_renomear(df\_cursos, COLUNAS\_CURSOS\_RELEVANTES, MAPPING\_CURSOS)

0255: else:

0256: print(f"Aviso: {ARQUIVO\_CURSOS} está vazio ou não pôde ser processado.")

0257: else:

0258: print(f"Aviso: Arquivo {ARQUIVO\_CURSOS} não encontrado em {caminho\_cursos}.")

0259:

0260: # ----------------------------------------------------------------------

0261: # Salvando resultados (IES e Cursos) na pasta "processado"

0262: # ----------------------------------------------------------------------

0263: if not df\_ies\_final.empty:

0264: saida\_ies = os.path.join(PASTA\_PROCESSADO, f"dados\_ies\_{year}.csv")

0265: os.makedirs(os.path.dirname(saida\_ies), exist\_ok=True)

0266: df\_ies\_final.to\_csv(saida\_ies, sep=";", index=False, encoding="utf-8")

0267: print(f"[OK] dados\_ies gerado em: {saida\_ies}")

0268: else:

0269: print("Nenhum dado de IES para salvar.")

0270:

0271: if not df\_cursos\_final.empty:

0272: saida\_cursos = os.path.join(PASTA\_PROCESSADO, f"dados\_cursos\_{year}.csv")

0273: os.makedirs(os.path.dirname(saida\_cursos), exist\_ok=True)

0274: df\_cursos\_final.to\_csv(saida\_cursos, sep=";", index=False, encoding="utf-8")

0275: print(f"[OK] dados\_cursos gerado em: {saida\_cursos}")

0276: else:

0277: print("Nenhum dado de Cursos para salvar.")

0278:

0279: if \_\_name\_\_ == "\_\_main\_\_":

0280: # Processa todos os anos de 2009 a 2023.

0281: for year in range(2009, 2024):

0282: print(f"\tProcessing year {year} ...")

0283: main(year)

## pre\_processamento\_Transfer\_Learn.py

0001: # scripts/processamento\_dados/pre\_processamento\_Transfer\_Learn.py

0002: # Este script prepara um CSV limpo e padronizado para uso nos modelos de Transfer Learning e Fine-tuning

0003:

0004: import pandas as pd

0005: import os

0006:

0007: def preprocessar\_transfer\_learning(caminho\_entrada, caminho\_saida):

0008: df = pd.read\_csv(caminho\_entrada, sep=None, engine='python')

0009:

0010: # Seleciona apenas as colunas de interesse e renomeia se necessário

0011: colunas\_validas = ['taxa\_ingresso', 'vagas\_totais', 'taxa\_evasao']

0012: df = df[[col for col in colunas\_validas if col in df.columns]]

0013:

0014: # Remove linhas com valores ausentes ou inválidos

0015: df = df.dropna()

0016:

0017: # Garante que todas as colunas estão em float

0018: df = df.astype(float)

0019:

0020: os.makedirs(os.path.dirname(caminho\_saida), exist\_ok=True)

0021: df.to\_csv(caminho\_saida, index=False)

0022: print(f"Base processada salva em: {caminho\_saida}")

0023:

0024:

0025: if \_\_name\_\_ == '\_\_main\_\_':

0026: entrada = './dados/processado/dados\_transfer\_learning.csv'

0027: saida = './dados/processado/dados\_transfer\_learning\_clean.csv'

0028: preprocessar\_transfer\_learning(entrada, saida)

## tratar\_dados.py

0001: # tratar\_dados.py

0002: # Script responsável pelo carregamento, limpeza, tratamento, cálculo de taxas e consolidação de dados educacionais extraídos dos microdados do INEP/MEC.

0003:

0004: import os

0005: import pandas as pd

0006: from pathlib import Path

0007: import re

0008: import glob

0009: import numpy as np

0010:

0011: # Defina a variável global para a pasta processada

0012: PASTA\_PROCESSADO = "./dados/processado"

0013:

0014: def carregar\_dados(caminho\_entrada):

0015: try:

0016: print(f"Carregando dados de: {caminho\_entrada}")

0017: df = pd.read\_csv(caminho\_entrada, sep=';', encoding='utf-8', low\_memory=False)

0018: print("Colunas disponíveis:", df.columns.tolist())

0019: return df

0020: except Exception as e:

0021: raise ValueError(f"Erro ao carregar os dados: {e}")

0022:

0023: def tratar\_dados(df, colunas\_numericas=None):

0024: df = df.drop\_duplicates()

0025: df = df.dropna()

0026:

0027: if colunas\_numericas:

0028: for col in colunas\_numericas:

0029: if col in df.columns:

0030: df[col] = pd.to\_numeric(df[col], errors='coerce')

0031:

0032: for col in ['ingressantes', 'concluintes', 'vagas\_totais', 'matriculados', 'numero\_cursos']:

0033: if col in df.columns:

0034: df = df[df[col] >= 0]

0035:

0036: if 'ingressantes' in df.columns:

0037: df = df[df['ingressantes'] > 0]

0038:

0039: df = df.dropna()

0040:

0041: return df

0042:

0043: def salvar\_dados\_tratados(df, caminho\_saida):

0044: try:

0045: os.makedirs(os.path.dirname(caminho\_saida), exist\_ok=True)

0046: df.to\_csv(caminho\_saida, index=False, sep=';', encoding='utf-8')

0047: print(f"Dados tratados salvos em: {caminho\_saida}")

0048: except Exception as e:

0049: raise ValueError(f"Erro ao salvar os dados: {e}")

0050:

0051: def pivotar\_dados\_cursos():

0052: arquivos = sorted(Path("./dados/processado").glob("dados\_cursos\_tratado\_\*.csv"))

0053: dfs = []

0054: for arq in arquivos:

0055: ano\_match = re.search(r"(\d{4})", arq.name)

0056: if not ano\_match:

0057: continue

0058: ano = int(ano\_match.group(1))

0059: df = pd.read\_csv(arq, sep=';', encoding='utf-8')

0060: df['ano'] = ano

0061: dfs.append(df)

0062:

0063: if not dfs:

0064: return

0065:

0066: df\_geral = pd.concat(dfs, ignore\_index=True)

0067:

0068: id\_cols = ["id\_curso", "nome\_curso", "modalidade\_ensino", "id\_ies"]

0069: id\_cols = [col for col in id\_cols if col in df\_geral.columns]

0070:

0071: df\_pivot = pd.DataFrame()

0072: for var in ["ingressantes", "concluintes", "matriculados", "vagas\_totais", "inscritos\_totais"]:

0073: if var in df\_geral.columns:

0074: tabela = df\_geral.pivot\_table(index=id\_cols, columns="ano", values=var)

0075: tabela.columns = [f"{var}\_{int(col)}" for col in tabela.columns]

0076: df\_pivot = pd.concat([df\_pivot, tabela], axis=1)

0077:

0078: df\_final = df\_geral[id\_cols].drop\_duplicates().set\_index(id\_cols)

0079: df\_final = df\_final.join(df\_pivot).reset\_index()

0080:

0081: salvar\_dados\_tratados(df\_final, "./dados/processado/dados\_cursos\_serie\_temporal.csv")

0082:

0083: def calcular\_taxas(df):

0084: df = df.copy()

0085: for coluna in ['ingressantes', 'concluintes', 'vagas\_totais']:

0086: if coluna in df.columns:

0087: df[coluna] = pd.to\_numeric(df[coluna], errors='coerce')

0088:

0089: df = df[(df['ingressantes'] > 0) & (df['vagas\_totais'] > 0)]

0090: df['taxa\_ingresso'] = df['ingressantes'] / df['vagas\_totais']

0091: df['taxa\_conclusao'] = df['concluintes'] / df['ingressantes']

0092: df['taxa\_evasao'] = 1 - df['taxa\_conclusao']

0093:

0094: for col in ['taxa\_ingresso', 'taxa\_conclusao', 'taxa\_evasao']:

0095: df = df[(df[col] >= 0) & (df[col] <= 1)]

0096:

0097: return df

0098:

0099: def salvar\_taxas\_consolidadas():

0100: pattern = os.path.join(PASTA\_PROCESSADO, "dados\_cursos\_tratado\_\*.csv")

0101: files = glob.glob(pattern)

0102: if not files:

0103: print("Nenhum arquivo de cursos tratado foi encontrado para consolidar.")

0104: return

0105: list\_df = []

0106: for f in files:

0107: try:

0108: df\_temp = pd.read\_csv(f, sep=";", encoding="utf-8")

0109: list\_df.append(df\_temp)

0110: except Exception as e:

0111: print(f"Erro ao ler o arquivo {f}: {e}")

0112: if not list\_df:

0113: print("Nenhum dado foi carregado para consolidação.")

0114: return

0115: df\_consolidado = pd.concat(list\_df, ignore\_index=True)

0116: df\_consolidado = calcular\_taxas(df\_consolidado)

0117: caminho\_saida = os.path.join(PASTA\_PROCESSADO, "dados\_ingresso\_evasao\_conclusao.csv")

0118: salvar\_dados\_tratados(df\_consolidado, caminho\_saida)

0119: print(f"[OK] Dados consolidados e taxas salvos em: {caminho\_saida}")

0120:

0121: def ler\_taxas\_consolidadas():

0122: caminho = os.path.join(PASTA\_PROCESSADO, "dados\_ingresso\_evasao\_conclusao.csv")

0123: try:

0124: df = pd.read\_csv(caminho, sep=";", encoding="utf-8")

0125: print("Dados consolidados lidos com sucesso.")

0126: return df

0127: except Exception as e:

0128: raise ValueError(f"Erro ao ler o arquivo de taxas consolidadas: {e}")

0129:

0130: def main(year: int = 2024):

0131: caminho\_ies = f'./dados/processado/dados\_ies\_{year}.csv'

0132: caminho\_cursos = f'./dados/processado/dados\_cursos\_{year}.csv'

0133:

0134: caminho\_ies\_tratado = f'./dados/intermediario/dados\_ies\_tratado\_{year}.csv'

0135: caminho\_cursos\_tratado = f'./dados/intermediario/dados\_cursos\_tratado\_{year}.csv'

0136:

0137: caminho\_ies\_final = f'./dados/processado/dados\_ies\_tratado\_{year}.csv'

0138: caminho\_cursos\_final = f'./dados/processado/dados\_cursos\_tratado\_{year}.csv'

0139:

0140: colunas\_numericas\_ies = [

0141: 'docentes\_total',

0142: 'docentes\_exercicio',

0143: 'docentes\_feminino',

0144: 'docentes\_masculino'

0145: ]

0146: colunas\_numericas\_cursos = [

0147: 'numero\_cursos',

0148: 'vagas\_totais',

0149: 'inscritos\_totais',

0150: 'ingressantes',

0151: 'matriculados',

0152: 'concluintes'

0153: ]

0154:

0155: try:

0156: df\_ies = carregar\_dados(caminho\_ies)

0157: except ValueError as e:

0158: print(e)

0159: df\_ies = pd.DataFrame()

0160:

0161: if not df\_ies.empty:

0162: try:

0163: df\_ies\_tratado = tratar\_dados(df\_ies, colunas\_numericas=colunas\_numericas\_ies)

0164: salvar\_dados\_tratados(df\_ies\_tratado, caminho\_ies\_tratado)

0165: salvar\_dados\_tratados(df\_ies\_tratado, caminho\_ies\_final)

0166: except ValueError as e:

0167: print(f"Erro ao processar dados de IES: {e}")

0168: else:

0169: print("Nenhum dado de IES disponível para tratar.")

0170:

0171: try:

0172: df\_cursos = carregar\_dados(caminho\_cursos)

0173: except ValueError as e:

0174: print(e)

0175: df\_cursos = pd.DataFrame()

0176:

0177: if not df\_cursos.empty:

0178: try:

0179: df\_cursos\_tratado = tratar\_dados(df\_cursos, colunas\_numericas=colunas\_numericas\_cursos)

0180: salvar\_dados\_tratados(df\_cursos\_tratado, caminho\_cursos\_tratado)

0181: salvar\_dados\_tratados(df\_cursos\_tratado, caminho\_cursos\_final)

0182: except ValueError as e:

0183: print(f"Ano de {year} Erro ao processar dados de Cursos: {e}")

0184: else:

0185: print("Nenhum dado de Cursos disponível para tratar.")

0186:

0187: if \_\_name\_\_ == '\_\_main\_\_':

0188: for year in range(2024):

0189: print(f"\tProcessing year {year} ...")

0190: main(year)

0191:

0192: pivotar\_dados\_cursos()

0193: salvar\_taxas\_consolidadas()

0194: df\_taxas = ler\_taxas\_consolidadas()

0195: print(df\_taxas.head())

0196:

## tratar\_dados\_Transfer\_Learn.py

0001: # Script: tratar\_dados\_Transfer\_Learn.py

0002: # Finalidade: Este script realiza o pré-processamento de dados para técnicas de Transfer Learning.

0003: # Ele lê a base limpa `dados\_transfer\_learning\_clean.csv`, verifica colunas essenciais, remove dados ausentes/duplicados

0004: # e gera dois arquivos com transformações diferentes: normalização MinMax e padronização Z-Score.

0005: # Saídas: `dados\_transfer\_normalizado.csv` e `dados\_transfer\_padronizado.csv` no diretório `dados/processado/`.

0006: # Etapa do pipeline: 2ª etapa (após pre\_processamento\_Transfer\_Learn.py e antes do treino dos modelos).

0007: import pandas as pd

0008: import os

0009: from sklearn.preprocessing import MinMaxScaler, StandardScaler

0010:

0011: # Caminho de entrada e saída

0012: caminho\_entrada = './dados/processado/dados\_transfer\_learning\_clean.csv'

0013: caminho\_saida = './dados/processado/'

0014:

0015: def main():

0016: print("🔍 Lendo arquivo de entrada...")

0017: df = pd.read\_csv(caminho\_entrada)

0018:

0019: print("🔧 Verificando colunas necessárias...")

0020: colunas\_esperadas = ['taxa\_ingresso', 'vagas\_totais', 'taxa\_evasao']

0021: for col in colunas\_esperadas:

0022: if col not in df.columns:

0023: raise ValueError(f"Coluna obrigatória ausente: {col}")

0024:

0025: print("🧼 Removendo linhas com valores ausentes ou duplicadas...")

0026: df = df[colunas\_esperadas].dropna().drop\_duplicates()

0027:

0028: # ===== Normalização Min-Max =====

0029: print("📉 Aplicando normalização MinMaxScaler...")

0030: scaler\_minmax = MinMaxScaler()

0031: df\_minmax = pd.DataFrame(scaler\_minmax.fit\_transform(df), columns=colunas\_esperadas)

0032: df\_minmax.to\_csv(os.path.join(caminho\_saida, 'dados\_transfer\_normalizado.csv'), index=False)

0033:

0034: # ===== Padronização Z-Score =====

0035: print("📊 Aplicando padronização StandardScaler...")

0036: scaler\_zscore = StandardScaler()

0037: df\_zscore = pd.DataFrame(scaler\_zscore.fit\_transform(df), columns=colunas\_esperadas)

0038: df\_zscore.to\_csv(os.path.join(caminho\_saida, 'dados\_transfer\_padronizado.csv'), index=False)

0039:

0040: print("✅ Arquivos gerados com sucesso:")

0041: print(f" - {os.path.join(caminho\_saida, 'dados\_transfer\_normalizado.csv')}")

0042: print(f" - {os.path.join(caminho\_saida, 'dados\_transfer\_padronizado.csv')}")

0043:

0044: if \_\_name\_\_ == '\_\_main\_\_':

0045: main()

## preparar\_entrada\_modelos.py

0001: # Este script prepara os dados de entrada para o treinamento de modelos preditivos.

0002: # Ele lê a base tratada sem vazamento de dados, verifica se as colunas esperadas estão presentes,

0003: # seleciona as features relevantes ('taxa\_ingresso' e 'vagas\_totais') e, se disponível,

0004: # inclui a coluna 'taxa\_evasao' como target. O resultado é salvo como CSV pronto para modelagem.

0005: import pandas as pd

0006: import os

0007:

0008: # Caminho de entrada e saída

0009: caminho\_entrada = 'dados/processado/dados\_transfer\_learning\_clean.csv'

0010: caminho\_saida = 'dados/processado/entrada\_modelos.csv'

0011:

0012: print("📥 Lendo base tratada para preparar entrada dos modelos...")

0013: try:

0014: df = pd.read\_csv(caminho\_entrada)

0015: print(df.columns.tolist())

0016: print("📋 Colunas disponíveis:", df.columns.tolist())

0017:

0018: colunas\_entrada = ['taxa\_ingresso', 'vagas\_totais']

0019:

0020: if not all(col in df.columns for col in colunas\_entrada):

0021: raise ValueError(f"❌ Colunas necessárias não encontradas: {colunas\_entrada}")

0022:

0023: # Seleciona apenas features

0024: df\_modelo = df[colunas\_entrada].copy()

0025: # df\_modelo.columns = ['feature1', 'feature2']

0026:

0027: if 'taxa\_evasao' in df.columns:

0028: df\_modelo['target'] = df['taxa\_evasao']

0029: else:

0030: raise ValueError("❌ Coluna 'taxa\_evasao' não encontrada para definir como target.")

0031:

0032: # Salva em CSV

0033: os.makedirs(os.path.dirname(caminho\_saida), exist\_ok=True)

0034: df\_modelo.to\_csv(caminho\_saida, index=False)

0035: print(f"✅ Arquivo de entrada para os modelos salvo em: {caminho\_saida}")

0036:

0037: except Exception as e:

0038: print(f"❌ Erro ao preparar entrada dos modelos: {e}")

## gerar\_base\_modelo.py

0001: # scripts/modelagem/gerar\_base\_modelo.py

0002: from tensorflow.keras.models import Sequential

0003: from tensorflow.keras.layers import Dense, Input

0004: from tensorflow.keras.losses import MeanSquaredError

0005: from tensorflow.keras.metrics import MeanAbsoluteError

0006:

0007: model = Sequential([

0008: Input(shape=(2,)),

0009: Dense(64, activation='relu'),

0010: Dense(32, activation='relu'),

0011: Dense(16, activation='relu'),

0012: Dense(1, activation='linear')

0013: ])

0014:

0015: model.compile(optimizer='adam', loss=MeanSquaredError(), metrics=[MeanAbsoluteError()])

0016: model.save('modelos/base\_modelo\_neural.h5')

## treinamento\_modelo\_original.py

0001: # /scripts/modelagem/treinamento\_modelo.py

0002: # Este script treina modelos de aprendizado de máquina para prever a taxa de evasão,

0003: # bem como explorar as relações entre taxa de ingresso, taxa de evasão e taxa de conclusão,

0004: # e realizar previsões para ingressantes e concluintes.

0005:

0006: import pandas as pd

0007: from sklearn.model\_selection import train\_test\_split

0008: from sklearn.ensemble import RandomForestRegressor

0009: from sklearn.linear\_model import LinearRegression

0010: from sklearn.metrics import mean\_squared\_error, r2\_score, confusion\_matrix

0011: import joblib

0012: import os

0013: import matplotlib.pyplot as plt

0014: import seaborn as sns

0015:

0016: def main():

0017: # Caminho para os dados processados

0018: caminho\_dados = './dados/processado/dados\_ingresso\_evasao\_conclusao.csv'

0019: df = pd.read\_csv(caminho\_dados, sep=';', encoding='utf-8', low\_memory=False)

0020:

0021: # Definir caminho\_modelo (com caminho relativo) antes de utilizá-lo em qualquer parte

0022: caminho\_modelo = './modelos/modelos\_salvos'

0023: os.makedirs(caminho\_modelo, exist\_ok=True)

0024:

0025: # Treinamento para prever ingressantes e concluintes

0026: if all(col in df.columns for col in ['numero\_cursos', 'vagas', 'inscritos', 'docentes', 'ingressantes', 'concluintes']):

0027: print("Treinando modelos para ingressantes e concluintes...")

0028:

0029: # Features e targets para previsões de ingressantes e concluintes

0030: X = df[['numero\_cursos', 'vagas', 'inscritos', 'docentes']]

0031: y\_ingressantes = df['ingressantes']

0032: y\_concluintes = df['concluintes']

0033:

0034: # Divisão dos dados para ingressantes

0035: X\_train\_ing, X\_test\_ing, y\_train\_ing, y\_test\_ing = train\_test\_split(X, y\_ingressantes, test\_size=0.2, random\_state=42)

0036:

0037: # Modelo Random Forest para ingressantes

0038: modelo\_ingressantes = RandomForestRegressor(random\_state=42)

0039: modelo\_ingressantes.fit(X\_train\_ing, y\_train\_ing)

0040: score\_ingressantes = modelo\_ingressantes.score(X\_test\_ing, y\_test\_ing)

0041: print(f"Score para ingressantes: {score\_ingressantes:.4f}")

0042:

0043: # Divisão dos dados para concluintes

0044: X\_train\_conc, X\_test\_conc, y\_train\_conc, y\_test\_conc = train\_test\_split(X, y\_concluintes, test\_size=0.2, random\_state=42)

0045:

0046: # Modelo Random Forest para concluintes

0047: modelo\_concluintes = RandomForestRegressor(random\_state=42)

0048: modelo\_concluintes.fit(X\_train\_conc, y\_train\_conc)

0049: score\_concluintes = modelo\_concluintes.score(X\_test\_conc, y\_test\_conc)

0050: print(f"Score para concluintes: {score\_concluintes:.4f}")

0051:

0052: # Salvar modelos de ingressantes e concluintes

0053: joblib.dump(modelo\_ingressantes, os.path.join(caminho\_modelo, 'modelo\_ingressantes.pkl'))

0054: joblib.dump(modelo\_concluintes, os.path.join(caminho\_modelo, 'modelo\_concluintes.pkl'))

0055:

0056: # Treinamento para taxas de evasão

0057: print("Treinando modelos para taxa de evasão...")

0058:

0059: # Filtrar as variáveis relevantes

0060: colunas\_relevantes = ['taxa\_ingresso', 'taxa\_evasao', 'taxa\_conclusao']

0061: df = df[colunas\_relevantes].dropna()

0062:

0063: # Dividir as features e o target

0064: X = df[['taxa\_ingresso', 'taxa\_conclusao']]

0065: y = df['taxa\_evasao']

0066:

0067: # Dividir os dados em treino e teste

0068: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

0069:

0070: # Treinar dois modelos: Regressão Linear e Random Forest

0071: # Modelo 1: Regressão Linear

0072: modelo\_linear = LinearRegression()

0073: modelo\_linear.fit(X\_train, y\_train)

0074: y\_pred\_linear = modelo\_linear.predict(X\_test)

0075: mse\_linear = mean\_squared\_error(y\_test, y\_pred\_linear)

0076: r2\_linear = r2\_score(y\_test, y\_pred\_linear)

0077:

0078: # Modelo 2: Random Forest

0079: modelo\_rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

0080: modelo\_rf.fit(X\_train, y\_train)

0081: y\_pred\_rf = modelo\_rf.predict(X\_test)

0082: mse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf)

0083: r2\_rf = r2\_score(y\_test, y\_pred\_rf)

0084:

0085: # Escolher o melhor modelo baseado no R²

0086: melhor\_modelo = modelo\_rf if r2\_rf > r2\_linear else modelo\_linear

0087: nome\_melhor\_modelo = 'Random Forest' if r2\_rf > r2\_linear else 'Regressão Linear'

0088:

0089: # Salvar o modelo escolhido

0090: joblib.dump(melhor\_modelo, os.path.join(caminho\_modelo, 'modelo\_melhor\_evasao.pkl'))

0091:

0092: # Salvar as métricas dos modelos

0093: caminho\_metricas = './modelos/resultados\_modelos'

0094: os.makedirs(caminho\_metricas, exist\_ok=True)

0095: with open(os.path.join(caminho\_metricas, 'metricas\_modelos.txt'), 'w') as f:

0096: f.write(f"Modelo: {nome\_melhor\_modelo}\n")

0097: f.write(f"MSE - Regressão Linear: {mse\_linear:.4f}, R²: {r2\_linear:.4f}\n")

0098: f.write(f"MSE - Random Forest: {mse\_rf:.4f}, R²: {r2\_rf:.4f}\n")

0099: f.write(f"Melhor Modelo Selecionado: {nome\_melhor\_modelo}\n")

0100:

0101: # Geração da Matriz de Confusão (binarizando a taxa de evasão com threshold de 0.5)

0102: threshold = 0.5

0103: y\_test\_class = (y\_test >= threshold).astype(int)

0104: y\_pred\_class = (melhor\_modelo.predict(X\_test) >= threshold).astype(int)

0105: cm = confusion\_matrix(y\_test\_class, y\_pred\_class)

0106: plt.figure(figsize=(6, 4))

0107: sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

0108: plt.title(f"Matriz de Confusão - {nome\_melhor\_modelo}")

0109: plt.xlabel("Predito")

0110: plt.ylabel("Verdadeiro")

0111: caminho\_cm = os.path.join(caminho\_metricas, 'matriz\_confusao.png')

0112: plt.savefig(caminho\_cm)

0113: plt.close()

0114:

0115: print(f"Treinamento concluído. Melhor modelo para evasão: {nome\_melhor\_modelo}")

0116: print(f"Métricas salvas em: {caminho\_metricas}")

0117: print(f"Modelos salvos em: {caminho\_modelo}")

0118:

0119: if \_\_name\_\_ == '\_\_main\_\_':

0120: main()

## treinamento\_modelo\_C4.5\_Tree\_J48.py

0001: # Script: treinamento\_modelo\_C4.5\_Tree\_J48.py

0002: # Descrição: Treina modelos de regressão para prever a taxa de evasão acadêmica

0003: # utilizando as variáveis 'taxa\_ingresso' e 'vagas\_totais' como preditoras.

0004: # Avalia o desempenho de Regressão Linear e Random Forest, seleciona o melhor

0005: # modelo com base no R², e salva o modelo e as métricas em arquivos apropriados.

0006:

0007: import pandas as pd

0008: from sklearn.model\_selection import train\_test\_split

0009: from sklearn.linear\_model import LinearRegression

0010: from sklearn.ensemble import RandomForestRegressor

0011: from sklearn.metrics import mean\_squared\_error, r2\_score

0012: import joblib

0013: import os

0014:

0015: print("Treinando modelos para taxa de evasão...")

0016:

0017: # Carregar dados tratados

0018: df = pd.read\_csv('dados/processado/entrada\_modelos.csv')

0019:

0020: X = df[['taxa\_ingresso', 'vagas\_totais']]

0021: y = df['target']

0022:

0023: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

0024:

0025: # Regressão Linear

0026: modelo\_lr = LinearRegression()

0027: modelo\_lr.fit(X\_train, y\_train)

0028: y\_pred\_lr = modelo\_lr.predict(X\_test)

0029: mse\_lr = mean\_squared\_error(y\_test, y\_pred\_lr)

0030: r2\_lr = r2\_score(y\_test, y\_pred\_lr)

0031:

0032: # Random Forest

0033: modelo\_rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

0034: modelo\_rf.fit(X\_train, y\_train)

0035: y\_pred\_rf = modelo\_rf.predict(X\_test)

0036: mse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf)

0037: r2\_rf = r2\_score(y\_test, y\_pred\_rf)

0038:

0039: # Escolher melhor modelo

0040: melhor\_modelo = modelo\_lr if r2\_lr > r2\_rf else modelo\_rf

0041: melhor\_nome = "Regressão Linear" if r2\_lr > r2\_rf else "Random Forest"

0042:

0043: # Criar pastas

0044: os.makedirs("modelos/modelos\_salvos", exist\_ok=True)

0045: os.makedirs("modelos/resultados\_modelos", exist\_ok=True)

0046:

0047: # Salvar melhor modelo

0048: joblib.dump(melhor\_modelo, 'modelos/modelos\_salvos/modelo\_melhor\_evasao.pkl')

0049:

0050: # Salvar métricas

0051: with open("modelos/resultados\_modelos/metricas\_modelos.txt", "w") as f:

0052: f.write("Modelo: " + melhor\_nome + "\n")

0053: f.write(f"MSE - Regressão Linear: {mse\_lr:.4f}, R²: {r2\_lr:.4f}\n")

0054: f.write(f"MSE - Random Forest: {mse\_rf:.4f}, R²: {r2\_rf:.4f}\n")

0055: f.write(f"Melhor Modelo Selecionado: {melhor\_nome}\n")

0056:

0057: print("Treinamento concluído. Melhor modelo para evasão:", melhor\_nome)

0058: print("Métricas salvas em: ./modelos/resultados\_modelos")

0059: print("Modelos salvos em: ./modelos/modelos\_salvos")

## treinamento\_modelo\_Feature-based.py

0001: # /scripts/modelagem/treinamento\_modelo\_Feature-based.py

0002: # Este script treina modelos de aprendizado de máquina combinado com Transfer Learning baseado em features para prever a taxa de evasão,

0003: # bem como explorar as relações entre taxa de ingresso, taxa de evasão e taxa de conclusão,

0004: # e realizar previsões para ingressantes e concluintes.

0005:

0006: import pandas as pd

0007: from sklearn.model\_selection import train\_test\_split

0008: from sklearn.ensemble import RandomForestRegressor

0009: from sklearn.linear\_model import LinearRegression

0010: from sklearn.metrics import mean\_squared\_error, r2\_score, confusion\_matrix

0011: import joblib

0012: import os

0013: import matplotlib.pyplot as plt

0014: import seaborn as sns

0015: from sklearn.preprocessing import StandardScaler

0016:

0017: def main():

0018: # Caminho para os dados processados

0019: caminho\_dados = './dados/processado/dados\_ingresso\_evasao\_conclusao.csv'

0020: df = pd.read\_csv(caminho\_dados, sep=';', encoding='utf-8', low\_memory=False)

0021:

0022: # Definir caminho\_modelo (com caminho relativo) antes de utilizá-lo em qualquer parte

0023: caminho\_modelo = './modelos/modelos\_salvos'

0024: os.makedirs(caminho\_modelo, exist\_ok=True)

0025:

0026: # Treinamento para prever ingressantes e concluintes

0027: if all(col in df.columns for col in ['numero\_cursos', 'vagas', 'inscritos', 'docentes', 'ingressantes', 'concluintes']):

0028: print("Treinando modelos para ingressantes e concluintes...")

0029:

0030: # Features e targets para previsões de ingressantes e concluintes

0031: X = df[['numero\_cursos', 'vagas', 'inscritos', 'docentes']]

0032: y\_ingressantes = df['ingressantes']

0033: y\_concluintes = df['concluintes']

0034:

0035: # Divisão dos dados para ingressantes

0036: X\_train\_ing, X\_test\_ing, y\_train\_ing, y\_test\_ing = train\_test\_split(X, y\_ingressantes, test\_size=0.2, random\_state=42)

0037:

0038: # Modelo Random Forest para ingressantes

0039: modelo\_ingressantes = RandomForestRegressor(random\_state=42)

0040: modelo\_ingressantes.fit(X\_train\_ing, y\_train\_ing)

0041: score\_ingressantes = modelo\_ingressantes.score(X\_test\_ing, y\_test\_ing)

0042: print(f"Score para ingressantes: {score\_ingressantes:.4f}")

0043:

0044: # Divisão dos dados para concluintes

0045: X\_train\_conc, X\_test\_conc, y\_train\_conc, y\_test\_conc = train\_test\_split(X, y\_concluintes, test\_size=0.2, random\_state=42)

0046:

0047: # Modelo Random Forest para concluintes

0048: modelo\_concluintes = RandomForestRegressor(random\_state=42)

0049: modelo\_concluintes.fit(X\_train\_conc, y\_train\_conc)

0050: score\_concluintes = modelo\_concluintes.score(X\_test\_conc, y\_test\_conc)

0051: print(f"Score para concluintes: {score\_concluintes:.4f}")

0052:

0053: # Salvar modelos de ingressantes e concluintes

0054: joblib.dump(modelo\_ingressantes, os.path.join(caminho\_modelo, 'modelo\_ingressantes.pkl'))

0055: joblib.dump(modelo\_concluintes, os.path.join(caminho\_modelo, 'modelo\_concluintes.pkl'))

0056:

0057: # Treinamento para taxas de evasão

0058: print("Treinando modelos para taxa de evasão...")

0059:

0060: # Treinamento adicional com Transfer Learning baseado em features

0061: print("Treinando modelo com Transfer Learning baseado em features...")

0062:

0063: caminho\_dados\_transfer = './dados/processado/dados\_transfer\_learning\_clean.csv'

0064: if os.path.exists(caminho\_dados\_transfer):

0065: df\_transfer = pd.read\_csv(caminho\_dados\_transfer)

0066: if all(col in df\_transfer.columns for col in ['media\_geral', 'taxa\_ingresso', 'taxa\_conclusao', 'taxa\_evasao']):

0067: X\_transfer = df\_transfer[['media\_geral', 'taxa\_ingresso', 'taxa\_conclusao']]

0068: y\_transfer = df\_transfer['taxa\_evasao']

0069:

0070: scaler\_transfer = StandardScaler()

0071: X\_transfer\_scaled = scaler\_transfer.fit\_transform(X\_transfer)

0072:

0073: X\_train\_tr, X\_test\_tr, y\_train\_tr, y\_test\_tr = train\_test\_split(X\_transfer\_scaled, y\_transfer, test\_size=0.3, random\_state=42)

0074: modelo\_transfer\_rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

0075: modelo\_transfer\_rf.fit(X\_train\_tr, y\_train\_tr)

0076:

0077: caminho\_modelo\_transfer = os.path.join(caminho\_modelo, 'modelo\_transfer\_rf.pkl')

0078: joblib.dump(modelo\_transfer\_rf, caminho\_modelo\_transfer)

0079: print(f"Modelo de Transfer Learning salvo em: {caminho\_modelo\_transfer}")

0080:

0081: # Filtrar as variáveis relevantes

0082: colunas\_relevantes = ['taxa\_ingresso', 'taxa\_evasao', 'taxa\_conclusao']

0083: df = df[colunas\_relevantes].dropna()

0084:

0085: # Dividir as features e o target

0086: X = df[['taxa\_ingresso', 'taxa\_conclusao']]

0087: y = df['taxa\_evasao']

0088:

0089: # Dividir os dados em treino e teste

0090: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

0091:

0092: # Treinar dois modelos: Regressão Linear e Random Forest

0093: # Modelo 1: Regressão Linear

0094: modelo\_linear = LinearRegression()

0095: modelo\_linear.fit(X\_train, y\_train)

0096: y\_pred\_linear = modelo\_linear.predict(X\_test)

0097: mse\_linear = mean\_squared\_error(y\_test, y\_pred\_linear)

0098: r2\_linear = r2\_score(y\_test, y\_pred\_linear)

0099:

0100: # Modelo 2: Random Forest

0101: modelo\_rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

0102: modelo\_rf.fit(X\_train, y\_train)

0103: y\_pred\_rf = modelo\_rf.predict(X\_test)

0104: mse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf)

0105: r2\_rf = r2\_score(y\_test, y\_pred\_rf)

0106:

0107: # Escolher o melhor modelo baseado no R²

0108: melhor\_modelo = modelo\_rf if r2\_rf > r2\_linear else modelo\_linear

0109: nome\_melhor\_modelo = 'Random Forest' if r2\_rf > r2\_linear else 'Regressão Linear'

0110:

0111: # Salvar o modelo escolhido

0112: joblib.dump(melhor\_modelo, os.path.join(caminho\_modelo, 'modelo\_melhor\_evasao.pkl'))

0113:

0114: # Salvar as métricas dos modelos

0115: caminho\_metricas = './modelos/resultados\_modelos'

0116: os.makedirs(caminho\_metricas, exist\_ok=True)

0117: with open(os.path.join(caminho\_metricas, 'metricas\_modelos.txt'), 'w') as f:

0118: f.write(f"Modelo: {nome\_melhor\_modelo}\n")

0119: f.write(f"MSE - Regressão Linear: {mse\_linear:.4f}, R²: {r2\_linear:.4f}\n")

0120: f.write(f"MSE - Random Forest: {mse\_rf:.4f}, R²: {r2\_rf:.4f}\n")

0121: f.write(f"Melhor Modelo Selecionado: {nome\_melhor\_modelo}\n")

0122:

0123: # Geração da Matriz de Confusão (binarizando a taxa de evasão com threshold de 0.5)

0124: threshold = 0.5

0125: y\_test\_class = (y\_test >= threshold).astype(int)

0126: y\_pred\_class = (melhor\_modelo.predict(X\_test) >= threshold).astype(int)

0127: cm = confusion\_matrix(y\_test\_class, y\_pred\_class)

0128: plt.figure(figsize=(6, 4))

0129: sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

0130: plt.title(f"Matriz de Confusão - {nome\_melhor\_modelo}")

0131: plt.xlabel("Predito")

0132: plt.ylabel("Verdadeiro")

0133: caminho\_cm = os.path.join(caminho\_metricas, 'matriz\_confusao.png')

0134: plt.savefig(caminho\_cm)

0135: plt.close()

0136:

0137: print(f"Treinamento concluído. Melhor modelo para evasão: {nome\_melhor\_modelo}")

0138: print(f"Métricas salvas em: {caminho\_metricas}")

0139: print(f"Modelos salvos em: {caminho\_modelo}")

0140:

0141: if \_\_name\_\_ == '\_\_main\_\_':

0142: main()

## treinamento\_modelo\_Fine-tuning.py

0001: # /scripts/modelagem/treinamento\_modelo.py

0002: # Script unificado para treinamento de modelos tradicionais e fine-tuning com rede neural

0003:

0004: import os

0005: import pandas as pd

0006: import numpy as np

0007: import joblib

0008: import matplotlib.pyplot as plt

0009: import seaborn as sns

0010: from sklearn.model\_selection import train\_test\_split

0011: from sklearn.ensemble import RandomForestRegressor

0012: from sklearn.linear\_model import LinearRegression

0013: from sklearn.metrics import mean\_squared\_error, r2\_score, confusion\_matrix

0014: from tensorflow.keras.models import load\_model

0015: from tensorflow.keras.layers import Dense

0016: from tensorflow.keras import Model

0017:

0018:

0019: def treinar\_modelos\_basicos(df, caminho\_modelo, caminho\_metricas):

0020: # Treinamento para prever ingressantes e concluintes

0021: if all(col in df.columns for col in ['numero\_cursos', 'vagas', 'inscritos', 'docentes', 'ingressantes', 'concluintes']):

0022: print("Treinando modelos para ingressantes e concluintes...")

0023:

0024: X = df[['numero\_cursos', 'vagas', 'inscritos', 'docentes']]

0025: y\_ing = df['ingressantes']

0026: y\_conc = df['concluintes']

0027:

0028: X\_train\_i, X\_test\_i, y\_train\_i, y\_test\_i = train\_test\_split(X, y\_ing, test\_size=0.2, random\_state=42)

0029: X\_train\_c, X\_test\_c, y\_train\_c, y\_test\_c = train\_test\_split(X, y\_conc, test\_size=0.2, random\_state=42)

0030:

0031: modelo\_i = RandomForestRegressor(random\_state=42)

0032: modelo\_c = RandomForestRegressor(random\_state=42)

0033: modelo\_i.fit(X\_train\_i, y\_train\_i)

0034: modelo\_c.fit(X\_train\_c, y\_train\_c)

0035:

0036: joblib.dump(modelo\_i, os.path.join(caminho\_modelo, 'modelo\_ingressantes.pkl'))

0037: joblib.dump(modelo\_c, os.path.join(caminho\_modelo, 'modelo\_concluintes.pkl'))

0038:

0039: print(f"Score ingressantes: {modelo\_i.score(X\_test\_i, y\_test\_i):.4f}")

0040: print(f"Score concluintes: {modelo\_c.score(X\_test\_c, y\_test\_c):.4f}")

0041:

0042: # Treinamento para evasão com modelos tradicionais

0043: print("Treinando modelos para taxa de evasão...")

0044: df = df[['taxa\_ingresso', 'taxa\_conclusao', 'taxa\_evasao']].dropna()

0045: X = df[['taxa\_ingresso', 'taxa\_conclusao']]

0046: y = df['taxa\_evasao']

0047: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

0048:

0049: modelo\_lr = LinearRegression()

0050: modelo\_rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

0051: modelo\_lr.fit(X\_train, y\_train)

0052: modelo\_rf.fit(X\_train, y\_train)

0053:

0054: r2\_lr = r2\_score(y\_test, modelo\_lr.predict(X\_test))

0055: r2\_rf = r2\_score(y\_test, modelo\_rf.predict(X\_test))

0056: melhor\_modelo = modelo\_rf if r2\_rf > r2\_lr else modelo\_lr

0057: nome\_melhor = 'Random Forest' if r2\_rf > r2\_lr else 'Regressão Linear'

0058: joblib.dump(melhor\_modelo, os.path.join(caminho\_modelo, 'modelo\_melhor\_evasao.pkl'))

0059:

0060: with open(os.path.join(caminho\_metricas, 'metricas\_modelos.txt'), 'w') as f:

0061: f.write(f"Modelo: {nome\_melhor}\n")

0062: f.write(f"R² Linear: {r2\_lr:.4f}\n")

0063: f.write(f"R² RF: {r2\_rf:.4f}\n")

0064: f.write(f"Melhor modelo: {nome\_melhor}\n")

0065:

0066: y\_pred\_class = (melhor\_modelo.predict(X\_test) >= 0.5).astype(int)

0067: y\_test\_class = (y\_test >= 0.5).astype(int)

0068: cm = confusion\_matrix(y\_test\_class, y\_pred\_class)

0069: plt.figure(figsize=(6, 4))

0070: sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

0071: plt.title(f"Matriz de Confusão - {nome\_melhor}")

0072: plt.xlabel("Predito")

0073: plt.ylabel("Verdadeiro")

0074: plt.savefig(os.path.join(caminho\_metricas, 'matriz\_confusao.png'))

0075: plt.close()

0076:

0077:

0078: def fine\_tune\_modelo\_transfer(df, caminho\_modelo):

0079: print("Iniciando Fine-Tuning com modelo neural pré-treinado...")

0080: esperadas = ['taxa\_ingresso', 'taxa\_conclusao']

0081: faltando = [col for col in ['taxa\_ingresso', 'taxa\_conclusao'] if col not in df.columns]

0082: if faltando:

0083: raise ValueError(f"Colunas ausentes para fine-tuning: {faltando}")

0084: X = df[['taxa\_ingresso', 'taxa\_conclusao']].values

0085: y = df['taxa\_evasao'].values

0086:

0087: from tensorflow.keras import Input

0088:

0089: base\_model = load\_model('modelos/base\_modelo\_neural.h5')

0090:

0091: entrada = Input(shape=(X.shape[1],), name='entrada\_transfer')

0092: x = base\_model(entrada, training=False)

0093: saida = Dense(1, activation='linear', name='saida\_finetune')(x)

0094: model\_finetuned = Model(inputs=entrada, outputs=saida)

0095:

0096: model\_finetuned.compile(optimizer='adam', loss='mse', metrics=['mae'])

0097: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

0098: model\_finetuned.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.2)

0099:

0100: model\_finetuned.save(os.path.join(caminho\_modelo, 'modelo\_finetuned\_tcc.h5'))

0101: print("Fine-Tuning concluído e modelo salvo.")

0102:

0103:

0104: def main():

0105: caminho\_modelo = './modelos/modelos\_salvos'

0106: caminho\_metricas = './modelos/resultados\_modelos'

0107: os.makedirs(caminho\_modelo, exist\_ok=True)

0108: os.makedirs(caminho\_metricas, exist\_ok=True)

0109:

0110: df\_principal = pd.read\_csv('./dados/processado/dados\_ingresso\_evasao\_conclusao.csv', sep=';', encoding='utf-8')

0111: treinar\_modelos\_basicos(df\_principal, caminho\_modelo, caminho\_metricas)

0112:

0113: # Validação e limpeza prévia do CSV de transferência

0114: caminho\_transfer = './dados/processado/dados\_transfer\_learning.csv'

0115: try:

0116: df\_transfer = pd.read\_csv(caminho\_transfer, sep=';', encoding='utf-8')

0117: except pd.errors.ParserError:

0118: print("Erro ao ler CSV com separador ';'. Tentando com ','...")

0119: try:

0120: df\_transfer = pd.read\_csv(caminho\_transfer, sep=',', encoding='utf-8')

0121: except pd.errors.ParserError:

0122: print("Erro persistente na leitura do CSV. Tentando leitura manual e correção...")

0123:

0124: # Correção linha a linha

0125: with open(caminho\_transfer, 'r', encoding='utf-8') as f:

0126: linhas\_validas = []

0127: for linha in f:

0128: if linha.count(',') == 3 or linha.count(';') == 3: # Espera-se 4 colunas

0129: linhas\_validas.append(linha)

0130:

0131: # Salva um novo CSV limpo

0132: caminho\_corrigido = './dados/processado/dados\_transfer\_learning\_corrigido.csv'

0133: with open(caminho\_corrigido, 'w', encoding='utf-8') as f\_out:

0134: f\_out.writelines(linhas\_validas)

0135: print(f"Arquivo corrigido salvo em: {caminho\_corrigido}")

0136: df\_transfer = pd.read\_csv(caminho\_corrigido, sep=',', encoding='utf-8', on\_bad\_lines='skip')

0137:

0138: # Já carregado acima após correção automatizada

0139:

0140: print("Colunas disponíveis em df\_transfer:", df\_transfer.columns.tolist())

0141:

0142: fine\_tune\_modelo\_transfer(df\_transfer, caminho\_modelo)

0143:

0144:

0145: if \_\_name\_\_ == '\_\_main\_\_':

0146: main()

## analises.py

0001: # Script: analises.py

0002: # Objetivo: Analisar dados de cursos e calcular/visualizar as taxas de conclusão e evasão por curso e ano

0003:

0004: import os

0005: import pandas as pd

0006: import numpy as np

0007: import plotly.express as px

0008: import plotly.graph\_objects as go

0009: import re

0010: from glob import glob

0011:

0012: # Define o mapeamento de cores: verde para "Taxa de Conclusão" e vermelho para "Taxa de Evasão"

0013: color\_map = {"Taxa de Conclusão": "green", "Taxa de Evasão": "red"}

0014:

0015: # =============================================================================

0016: # Leitura dos dados de cursos para os anos de 2009 a 2023

0017: # =============================================================================

0018: anos = range(2009, 2024)

0019: lista\_cursos = []

0020:

0021: # Lista de colunas que devem ser numéricas

0022: numeric\_cols = ["numero\_cursos", "vagas\_totais", "ingressantes", "concluintes", "inscritos\_totais"]

0023:

0024: for ano in anos:

0025: caminho = f"dados/processado/dados\_cursos\_tratado\_{ano}.csv"

0026: if os.path.exists(caminho):

0027: df = pd.read\_csv(caminho, sep=";", dtype=str) # Ler tudo inicialmente como string

0028: for col in numeric\_cols:

0029: if col in df.columns:

0030: df[col] = pd.to\_numeric(df[col], errors='coerce')

0031: df["ano"] = ano

0032: lista\_cursos.append(df)

0033: else:

0034: print(f"Arquivo não encontrado: {caminho}")

0035:

0036: dados\_cursos\_tratado = pd.concat(lista\_cursos, ignore\_index=True)

0037:

0038: # =============================================================================

0039: # Leitura dos dados de universidades (IES) – utilizando a base do ano 2023

0040: # =============================================================================

0041: ies\_path = "dados/processado/dados\_ies\_tratado\_2023.csv"

0042: dados\_ies\_tratado = pd.read\_csv(ies\_path, sep=";", dtype=str)

0043:

0044: # =============================================================================

0045: # Unindo os dados de cursos e universidades

0046: # =============================================================================

0047: all\_data = pd.merge(dados\_cursos\_tratado, dados\_ies\_tratado, on="id\_ies", how="left")

0048: all\_data = all\_data[(all\_data["numero\_cursos"].astype(float) > 0) & (all\_data["nome\_ies"].notnull())]

0049: all\_data["nome\_curso"] = all\_data["nome\_curso"].str.upper()

0050: all\_data["nome\_ies"] = all\_data["nome\_ies"].str.upper()

0051:

0052: for col in ["concluintes", "ingressantes", "inscritos\_totais", "vagas\_totais"]:

0053: if col in all\_data.columns:

0054: all\_data[col] = all\_data[col].astype("Int64")

0055:

0056: # Removemos a variável 'tipo\_rede' pois só aparece explicitamente a partir de 2023

0057: all\_data["cat\_adm"] = all\_data["cat\_adm"].replace({"1": "Federal", "2": "Estadual"}).fillna("Outro")

0058: all\_data["modalidade\_ensino"] = all\_data["modalidade\_ensino"].replace({"1": "Presencial", "2": "EAD"}).fillna("Outro")

0059:

0060: # =============================================================================

0061: # Agrupamento e pivot dos dados para calcular taxas de conclusão e evasão

0062: # =============================================================================

0063: ingress = (all\_data[["nome\_curso", "modalidade\_ensino", "ingressantes", "ano", "concluintes"]]

0064: .groupby(["nome\_curso", "modalidade\_ensino", "ano"], as\_index=False)

0065: .agg({"ingressantes": "sum", "concluintes": "sum"}))

0066:

0067: # Pivotar os dados para 'ingressantes' e 'concluintes'

0068: pivot\_ing = ingress.pivot\_table(index=["nome\_curso", "modalidade\_ensino"], columns="ano", values="ingressantes")

0069: pivot\_con = ingress.pivot\_table(index=["nome\_curso", "modalidade\_ensino"], columns="ano", values="concluintes")

0070:

0071: # Renomeia as colunas com o padrão "ingressantes\_{ano}" e "concluintes\_{ano}"

0072: pivot\_ing.columns = [f"ingressantes\_{ano}" for ano in pivot\_ing.columns]

0073: pivot\_con.columns = [f"concluintes\_{ano}" for ano in pivot\_con.columns]

0074:

0075: ingress\_wide = pivot\_ing.join(pivot\_con).reset\_index()

0076:

0077: # Filtra somente os cursos de interesse

0078: cursos\_interesse = ["ENGENHARIA CIVIL", "MEDICINA", "DIREITO", "ADMINISTRAÇÃO"]

0079: ingress\_wide = ingress\_wide[ingress\_wide["nome\_curso"].isin(cursos\_interesse)]

0080:

0081: # =============================================================================

0082: # Cálculo das taxas de conclusão e evasão por curso

0083: # =============================================================================

0084: # Fórmulas:

0085: # Taxa de Conclusão = concluintes\_{ano} / ingressantes\_{ano - D}

0086: # Taxa de Evasão = 1 - Taxa de Conclusão

0087: duracoes = {

0088: "ENGENHARIA CIVIL": 5,

0089: "MEDICINA": 6,

0090: "DIREITO": 5,

0091: "ADMINISTRAÇÃO": 4

0092: }

0093:

0094: def calcular\_metricas(df, curso, duracao):

0095: df\_course = df[df["nome\_curso"] == curso].copy()

0096: for ano in range(2009 + duracao, 2024):

0097: col\_ing = f"ingressantes\_{ano - duracao}"

0098: col\_con = f"concluintes\_{ano}"

0099: col\_conclusao = f"taxa\_conclusao\_{ano}"

0100: col\_evasao = f"taxa\_evasao\_{ano}"

0101: if col\_ing in df\_course.columns and col\_con in df\_course.columns:

0102: # Evitar divisão por zero e valores não definidos:

0103: df\_course[col\_conclusao] = df\_course[col\_con] / df\_course[col\_ing]

0104: df\_course[col\_evasao] = 1 - df\_course[col\_conclusao]

0105: else:

0106: df\_course[col\_conclusao] = np.nan

0107: df\_course[col\_evasao] = np.nan

0108: return df\_course

0109:

0110: df\_eng = calcular\_metricas(ingress\_wide, "ENGENHARIA CIVIL", duracoes["ENGENHARIA CIVIL"])

0111: df\_dir = calcular\_metricas(ingress\_wide, "DIREITO", duracoes["DIREITO"])

0112: df\_med = calcular\_metricas(ingress\_wide, "MEDICINA", duracoes["MEDICINA"])

0113: df\_adm = calcular\_metricas(ingress\_wide, "ADMINISTRAÇÃO", duracoes["ADMINISTRAÇÃO"])

0114:

0115: # Função para converter data frame wide em long para um dado prefixo e rótulo

0116: def pivot\_long\_metric(df, prefix, label):

0117: metric\_cols = [col for col in df.columns if col.startswith(prefix)]

0118: df\_long = df.melt(id\_vars=["nome\_curso", "modalidade\_ensino"],

0119: value\_vars=metric\_cols,

0120: var\_name="ano", value\_name="taxa")

0121: df\_long["ano"] = df\_long["ano"].str.extract(r"(\d{4})")

0122: df\_long["tipo"] = label

0123: return df\_long

0124:

0125: # Criando data frames long para cada curso e para cada métrica

0126: eng\_long\_conclusao = pivot\_long\_metric(df\_eng, "taxa\_conclusao\_", "Taxa de Conclusão")

0127: eng\_long\_evasao = pivot\_long\_metric(df\_eng, "taxa\_evasao\_", "Taxa de Evasão")

0128: eng\_civil\_long = pd.concat([eng\_long\_conclusao, eng\_long\_evasao])

0129:

0130: dir\_long\_conclusao = pivot\_long\_metric(df\_dir, "taxa\_conclusao\_", "Taxa de Conclusão")

0131: dir\_long\_evasao = pivot\_long\_metric(df\_dir, "taxa\_evasao\_", "Taxa de Evasão")

0132: direito\_long = pd.concat([dir\_long\_conclusao, dir\_long\_evasao])

0133:

0134: med\_long\_conclusao = pivot\_long\_metric(df\_med, "taxa\_conclusao\_", "Taxa de Conclusão")

0135: med\_long\_evasao = pivot\_long\_metric(df\_med, "taxa\_evasao\_", "Taxa de Evasão")

0136: medicina\_long = pd.concat([med\_long\_conclusao, med\_long\_evasao])

0137:

0138: adm\_long\_conclusao = pivot\_long\_metric(df\_adm, "taxa\_conclusao\_", "Taxa de Conclusão")

0139: adm\_long\_evasao = pivot\_long\_metric(df\_adm, "taxa\_evasao\_", "Taxa de Evasão")

0140: administracao\_long = pd.concat([adm\_long\_conclusao, adm\_long\_evasao])

0141:

0142: # =============================================================================

0143: # Plotagem com Plotly – duas linhas: uma para "Taxa de Conclusão" (verde) e outra para "Taxa de Evasão" (vermelha)

0144: # =============================================================================

0145: def plot\_taxa(df\_long, curso):

0146: fig = px.line(

0147: df\_long,

0148: x="ano",

0149: y="taxa",

0150: color="tipo",

0151: markers=True,

0152: title=f"Taxa de Evasão e Conclusão ao longo dos anos - Curso de {curso}",

0153: labels={"ano": "Ano de Ingresso", "taxa": "Taxa"},

0154: color\_discrete\_map=color\_map

0155: )

0156: fig.update\_yaxes(tickformat=".0%")

0157: fig.update\_layout(legend=dict(orientation="v", title=dict(text="Métrica")))

0158: fig.show()

0159:

0160: plot\_taxa(eng\_civil\_long, "Engenharia Civil")

0161: plot\_taxa(direito\_long, "Direito")

0162: plot\_taxa(medicina\_long, "Medicina")

0163: plot\_taxa(administracao\_long, "Administração")

0164:

0165: # =============================================================================

0166: # Exportação dos dados processados

0167: # =============================================================================

0168: ingress\_wide.to\_csv("dados/processado/final\_ingressantes.csv", index=False, sep=";")

0169: df\_eng.to\_csv("dados/processado/final\_eng\_civil.csv", index=False, sep=";")

0170: df\_dir.to\_csv("dados/processado/final\_direito.csv", index=False, sep=";")

0171: df\_med.to\_csv("dados/processado/final\_medicina.csv", index=False, sep=";")

0172: df\_adm.to\_csv("dados/processado/final\_administracao.csv", index=False, sep=";")

## comparar\_modelos.py

0001: # Script: comparar\_modelos.py

0002: # Objetivo: Comparar dois modelos preditivos (.pkl e .h5) usando o dataset de entrada

0003: # 'entrada\_modelos.csv', que deve conter as colunas ['taxa\_ingresso', 'vagas\_totais', 'target'].

0004: # O script gera métricas (MSE e R²) e gráficos comparativos de linha e dispersão,

0005: # que são salvos na pasta 'resultados/'.

0006:

0007: import pandas as pd

0008: import os

0009: os.makedirs("resultados", exist\_ok=True)

0010:

0011: # Carregar o arquivo de entrada

0012: df = pd.read\_csv('dados/processado/entrada\_modelos.csv')

0013:

0014: # Verificar e exibir as colunas disponíveis

0015: colunas\_disponiveis = df.columns.tolist()

0016: print(f"📋 Colunas disponíveis: {colunas\_disponiveis}")

0017:

0018: # Verificar se todas as colunas necessárias estão presentes

0019: colunas\_necessarias = ['taxa\_ingresso', 'vagas\_totais', 'target']

0020: if all(col in colunas\_disponiveis for col in colunas\_necessarias):

0021: df\_modelo = df[colunas\_necessarias].copy()

0022: print("✅ Dados carregados e prontos para comparação de modelos.")

0023:

0024: import matplotlib.pyplot as plt

0025: import joblib

0026: from tensorflow.keras.models import load\_model

0027: from sklearn.metrics import mean\_squared\_error, r2\_score

0028:

0029: # Separar variáveis

0030: X = df\_modelo[['taxa\_ingresso', 'vagas\_totais']]

0031: y\_true = df\_modelo['target']

0032:

0033: # Carregar modelos

0034: modelo\_pkl = joblib.load('modelos/modelos\_salvos/modelo\_melhor\_evasao.pkl')

0035: modelo\_h5 = load\_model('modelos/modelos\_salvos/modelo\_finetuned\_tcc.h5', compile=False)

0036:

0037: # Fazer previsões

0038: y\_pred\_pkl = modelo\_pkl.predict(X)

0039: y\_pred\_h5 = modelo\_h5.predict(X).flatten()

0040:

0041: # Avaliar modelos

0042: mse\_pkl = mean\_squared\_error(y\_true, y\_pred\_pkl)

0043: r2\_pkl = r2\_score(y\_true, y\_pred\_pkl)

0044:

0045: mse\_h5 = mean\_squared\_error(y\_true, y\_pred\_h5)

0046: r2\_h5 = r2\_score(y\_true, y\_pred\_h5)

0047:

0048: print("\nModelo .pkl")

0049: print(f"MSE: {mse\_pkl:.4f}")

0050: print(f"R²: {r2\_pkl:.4f}")

0051:

0052: print("\nModelo .h5")

0053: print(f"MSE: {mse\_h5:.4f}")

0054: print(f"R²: {r2\_h5:.4f}")

0055:

0056: # Gráfico comparativo

0057: plt.figure(figsize=(10, 5))

0058: plt.plot(y\_true.values, label='Real', marker='o')

0059: plt.plot(y\_pred\_pkl, label='Predição .pkl', marker='x')

0060: plt.plot(y\_pred\_h5, label='Predição .h5', marker='s')

0061: plt.title('Comparação entre valores reais e predições dos modelos')

0062: plt.xlabel('Amostra')

0063: plt.ylabel('Taxa de evasão')

0064: plt.legend()

0065: plt.grid(True)

0066: plt.tight\_layout()

0067: # A pasta 'resultados/' deve existir manualmente para evitar erros ao salvar os arquivos.

0068: plt.savefig('resultados/comparacao\_modelos\_linha.png')

0069: plt.savefig("resultados/comparacao\_modelos\_linha.png")

0070: plt.close()

0071:

0072: # Gráfico de dispersão

0073: plt.figure(figsize=(6, 6))

0074: plt.scatter(y\_true, y\_pred\_pkl, label='Modelo .pkl', alpha=0.7)

0075: plt.scatter(y\_true, y\_pred\_h5, label='Modelo .h5', alpha=0.7)

0076: plt.plot([y\_true.min(), y\_true.max()], [y\_true.min(), y\_true.max()], 'k--', lw=2)

0077: plt.title('Dispersão: Real vs Predito')

0078: plt.xlabel('Real')

0079: plt.ylabel('Predito')

0080: plt.legend()

0081: plt.grid(True)

0082: plt.tight\_layout()

0083: # A pasta 'resultados/' deve existir manualmente para evitar erros ao salvar os arquivos.

0084: plt.savefig('resultados/comparacao\_modelos\_dispersao.png')

0085: plt.savefig("resultados/comparacao\_modelos\_dispersao.png")

0086: plt.close()

0087:

0088: else:

0089: faltando = list(set(colunas\_necessarias) - set(colunas\_disponiveis))

0090: raise ValueError(f"❌ Colunas necessárias não encontradas: {faltando}")

## validar\_modelos\_temporais.py

0001: # Script para validação temporal simples de modelo preditivo.

0002: # Realiza análise de correlação, separa treino/teste com shuffle,

0003: # treina modelo LinearRegression e exibe avaliação via MSE e R².

0004: # Também plota gráfico de linha com comparação entre valores reais e preditos.

0005:

0006: import pandas as pd

0007: from sklearn.linear\_model import LinearRegression

0008: from sklearn.metrics import mean\_squared\_error, r2\_score

0009:

0010: # Carrega os dados

0011: df = pd.read\_csv('dados/processado/entrada\_modelos.csv')

0012:

0013: print("\n📌 Correlação das features com o target:")

0014: print(df.corr(numeric\_only=True))

0015:

0016: # Adiciona coluna 'ano' sequencialmente, assumindo que cada linha representa um ano de 2009 a 2023

0017: if 'ano' not in df.columns:

0018: anos = list(range(2009, 2024))

0019: df['ano'] = [anos[i % len(anos)] for i in range(len(df))]

0020:

0021: from sklearn.model\_selection import train\_test\_split

0022: # Embaralha e divide o conjunto entre treino e teste

0023: X = df[['taxa\_ingresso', 'vagas\_totais']]

0024: y = df['target']

0025: X\_treino, X\_teste, y\_treino, y\_teste = train\_test\_split(X, y, test\_size=0.3, random\_state=42, shuffle=True)

0026:

0027: # Treina modelo

0028: modelo = LinearRegression()

0029: modelo.fit(X\_treino, y\_treino)

0030:

0031: # Avalia

0032: y\_pred = modelo.predict(X\_teste)

0033: print("\n📊 Avaliação no conjunto de TESTE:")

0034: print("MSE:", round(mean\_squared\_error(y\_teste, y\_pred), 4))

0035: print("R²:", round(r2\_score(y\_teste, y\_pred), 4))

0036:

0037: import matplotlib.pyplot as plt

0038:

0039: # Gráfico de linha: valores reais vs preditos

0040: plt.figure(figsize=(10, 5))

0041: plt.plot(range(len(y\_teste)), y\_teste.values, label='Real', marker='o')

0042: plt.plot(range(len(y\_pred)), y\_pred, label='Previsto', marker='x')

0043: plt.title('Comparação entre valores reais e previstos (teste)')

0044: plt.xlabel('Amostras')

0045: plt.ylabel('Taxa de Evasão (normalizada)')

0046: plt.legend()

0047: plt.grid(True)

0048: plt.tight\_layout()

0049: plt.savefig('resultados/validacao\_temporal\_comparacao.png')

0050: plt.show()

## prever\_com\_modelos.py

0001: import joblib

0002: import numpy as np

0003: import pandas as pd

0004: from tensorflow.keras.models import load\_model

0005: from sklearn.preprocessing import StandardScaler

0006:

0007: # Carregar modelo .pkl (Scikit-Learn)

0008: modelo\_pkl = joblib.load('modelos/modelos\_salvos/modelo\_melhor\_evasao.pkl')

0009:

0010: # Entrada de exemplo: taxa\_ingresso, taxa\_conclusao

0011: entrada = np.array([[0.45, 0.52]])

0012:

0013: # Previsão com modelo .pkl

0014: pred\_pkl = modelo\_pkl.predict(entrada)

0015: print(f"Predição modelo .pkl: {pred\_pkl[0]:.4f}")

0016:

0017: # Carregar modelo .h5 (Keras)

0018: modelo\_h5 = load\_model('modelos/modelo\_finetuned\_tcc.h5')

0019:

0020: # Ajustar entrada para 3 colunas esperadas: media\_geral (dummy), taxa\_ingresso, taxa\_conclusao

0021: entrada\_nn = np.array([[0.0, entrada[0][0], entrada[0][1]]])

0022:

0023: # Previsão com modelo .h5

0024: pred\_h5 = modelo\_h5.predict(entrada\_nn, verbose=0)

0025: print(f"Predição modelo .h5: {pred\_h5[0][0]:.4f}")

## resumir\_modelo\_h5.py

0001: from tensorflow.keras.models import load\_model

0002:

0003: # Carregar modelo

0004: modelo = load\_model('modelos/modelo\_finetuned\_tcc.h5')

0005:

0006: # Exibir arquitetura

0007: modelo.summary()

## gerar\_graficos.py

0001: # scripts/visualizacao/gerar\_graficos.py

0002: # Este script gera gráficos para visualizar os resultados da análise.

0003:

0004: import pandas as pd

0005: import matplotlib.pyplot as plt

0006: import seaborn as sns

0007: import unicodedata

0008:

0009: def main():

0010: df = pd.read\_csv('dados/processado/dados\_ingresso\_evasao\_conclusao.csv', sep=';', on\_bad\_lines='skip')

0011:

0012: # Gráfico de distribuição da taxa de evasão

0013: plt.figure(figsize=(10,6))

0014: sns.histplot(df['taxa\_evasao'], bins=20, kde=True, color='blue')

0015: plt.title('Distribuição da Taxa de Evasão')

0016: plt.xlabel('Taxa de Evasão (%)')

0017: plt.ylabel('Frequência')

0018: plt.savefig('relatorios/figuras/distribuicao\_taxa\_evasao.png')

0019: plt.close()

0020:

0021: # Gráfico de distribuição da taxa de conclusão

0022: plt.figure(figsize=(10,6))

0023: sns.histplot(df['taxa\_conclusao'], bins=20, kde=True, color='green')

0024: plt.title('Distribuição da Taxa de Conclusão')

0025: plt.xlabel('Taxa de Conclusão (%)')

0026: plt.ylabel('Frequência')

0027: plt.savefig('relatorios/figuras/grafico\_taxa\_conclusao.png')

0028: plt.close()

0029:

0030: # Gráfico de distribuição da taxa de ingresso

0031: plt.figure(figsize=(10,6))

0032: sns.histplot(df['taxa\_ingresso'], bins=20, kde=True, color='orange')

0033: plt.title('Distribuição da Taxa de Ingresso')

0034: plt.xlabel('Taxa de Ingresso (%)')

0035: plt.ylabel('Frequência')

0036: plt.savefig('relatorios/figuras/grafico\_taxa\_ingresso.png')

0037: plt.close()

0038:

0039: # Gráfico de dispersão entre ingressantes e concluintes

0040: plt.figure(figsize=(10,6))

0041: sns.scatterplot(x='ingressantes', y='concluintes', data=df)

0042: plt.title('Correlação entre Ingressantes e Concluintes')

0043: plt.xlabel('Ingressantes')

0044: plt.ylabel('Concluintes')

0045: plt.savefig('relatorios/figuras/grafico\_ingressantes\_vs\_concluintes.png')

0046: plt.close()

0047:

0048: # Gráfico de barras por modalidade de ensino (taxa de evasão média)

0049: if 'modalidade\_ensino' in df.columns:

0050: plt.figure(figsize=(10,6))

0051: sns.barplot(x='modalidade\_ensino', y='taxa\_evasao', data=df)

0052: plt.title('Taxa de Evasão Média por Modalidade de Ensino')

0053: plt.xlabel('Modalidade de Ensino')

0054: plt.ylabel('Taxa de Evasão (%)')

0055: plt.savefig('relatorios/figuras/grafico\_taxa\_evasao\_modalidade.png')

0056: plt.close()

0057:

0058: # Heatmap de correlação

0059: plt.figure(figsize=(12,10))

0060: sns.heatmap(df[['taxa\_ingresso', 'taxa\_conclusao', 'taxa\_evasao', 'ingressantes', 'concluintes']].corr(), annot=True, cmap='coolwarm')

0061: plt.title('Mapa de Calor das Correlações')

0062: plt.savefig('relatorios/figuras/grafico\_mapa\_calor\_correlacoes.png')

0063: plt.close()

0064:

0065: print('Gráficos gerados e salvos em relatorios/figuras/')

0066:

0067: if \_\_name\_\_ == '\_\_main\_\_':

0068: main()

## comparar\_predicoes\_cursos.py

0001: import pandas as pd

0002: import matplotlib.pyplot as plt

0003: import seaborn as sns

0004: import csv

0005:

0006: caminhos = {

0007: 'Administração': 'dados/processado/final\_administracao.csv',

0008: 'Direito': 'dados/processado/final\_direito.csv',

0009: 'Engenharia Civil': 'dados/processado/final\_eng\_civil.csv',

0010: 'Medicina': 'dados/processado/final\_medicina.csv'

0011: }

0012:

0013: for curso, caminho in caminhos.items():

0014: try:

0015: df = pd.read\_csv(caminho, delimiter=';')

0016: # Se 'ano' e 'evasao\_real' não estiverem no DataFrame, recriá-los a partir das colunas 'taxa\_evasao\_20XX'

0017: if 'ano' not in df.columns or 'evasao\_real' not in df.columns:

0018: colunas\_evasao = [col for col in df.columns if col.startswith('taxa\_evasao\_')]

0019: if colunas\_evasao:

0020: anos = [int(col.split('\_')[-1]) for col in colunas\_evasao]

0021: evasoes = [df[col].values[0] for col in colunas\_evasao]

0022: df = pd.DataFrame({'ano': anos, 'evasao\_real': evasoes})

0023: df['evasao\_predita'] = df['evasao\_real'] # Inicialmente copia, pode ser sobrescrito se existir coluna real

0024: print(f"📂 Verificando cabeçalho do arquivo {caminho}...")

0025: with open(caminho, 'r') as f:

0026: for \_ in range(3):

0027: print(f.readline().strip())

0028: print(f"📄 Colunas encontradas: {df.columns.tolist()}")

0029: except Exception as e:

0030: print(f"❌ Erro ao ler {caminho}: {e}")

0031: continue

0032: if 'ano' not in df.columns:

0033: print(f"⚠️ Coluna 'ano' não encontrada no arquivo {caminho}.")

0034: print(f"\n📄 Colunas no arquivo {caminho}:\n{df.columns.tolist()}\n")

0035: plt.figure(figsize=(10, 5))

0036: sns.lineplot(x='ano', y='evasao\_real', data=df, label='INEP (real)', marker='o')

0037: sns.lineplot(x='ano', y='evasao\_predita', data=df, label='Projeto TCC (predita)', marker='s')

0038: plt.title(f'Taxa de Evasão - {curso}')

0039: plt.xlabel('Ano')

0040: plt.ylabel('Taxa de Evasão')

0041: plt.ylim(0, 1)

0042: plt.legend()

0043: plt.grid(True)

0044: plt.tight\_layout()

0045: plt.savefig(f'documentos/graficos/taxa\_evasao\_{curso.lower().replace(" ", "\_")}.png')

0046: plt.close()

0047:

0048: print("✅ Gráficos gerados para os cursos: Administração, Direito, Engenharia Civil e Medicina.")

## app\_evasao.py

0001: # app\_evasao.py

0002: # Dashboard interativo com análise de evasão usando Streamlit

0003:

0004: import streamlit as st

0005: import pandas as pd

0006: import matplotlib.pyplot as plt

0007: import seaborn as sns

0008: import os

0009:

0010: BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

0011: csv\_path = os.path.join(BASE\_DIR, '..', '..', 'dados', 'processado', 'dados\_ingresso\_evasao\_conclusao.csv')

0012: df = pd.read\_csv(csv\_path, sep=';')

0013:

0014: st.set\_page\_config(page\_title="Dashboard Evasão IES", layout="wide")

0015:

0016: # Título

0017: st.title("📊 Dashboard - Taxas de Ingresso, Conclusão e Evasão")

0018:

0019: # Filtro por curso

0020: cursos = df['nome\_curso'].unique()

0021: curso\_selecionado = st.selectbox("Selecione um curso:", sorted(cursos))

0022:

0023: # Filtrar

0024: df\_filtrado = df[df['nome\_curso'] == curso\_selecionado]

0025:

0026: # Métricas rápidas

0027: col1, col2, col3 = st.columns(3)

0028: col1.metric("Taxa de Ingresso (média)", f"{df\_filtrado['taxa\_ingresso'].mean():.2f}")

0029: col2.metric("Taxa de Conclusão (média)", f"{df\_filtrado['taxa\_conclusao'].mean():.2f}")

0030: col3.metric("Taxa de Evasão (média)", f"{df\_filtrado['taxa\_evasao'].mean():.2f}")

0031:

0032: # Gráfico de linha

0033: st.subheader("📈 Evolução das Taxas")

0034: fig, ax = plt.subplots(figsize=(10, 5))

0035: sns.lineplot(data=df\_filtrado[['taxa\_ingresso', 'taxa\_conclusao', 'taxa\_evasao']])

0036: st.pyplot(fig)