# Apêndice - Códigos Python Utilizados

## coletar\_links\_inep.py

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

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # scripts/coleta\_dados/coletar\_links\_inep.py

0006: # Este script coleta os links dos microdados do Censo da Educação Superior do site do INEP (1995-2023).

0007:

0008: import requests

0009: from bs4 import BeautifulSoup

0010:

0011: def coletar\_links\_inep(url\_base, palavra\_chave):

0012: """

0013: Coleta os links dos microdados do Censo da Educação Superior do INEP.

0014:

0015: :param url\_base: URL da página do INEP com os dados.

0016: :param palavra\_chave: Palavra-chave que identifica os links desejados.

0017: :return: Dicionário com os anos e os links dos microdados.

0018: """

0019: resposta = requests.get(url\_base)

0020: if resposta.status\_code != 200:

0021: raise Exception(f"Erro ao acessar {url\_base}: {resposta.status\_code}")

0022:

0023: soup = BeautifulSoup(resposta.content, 'html.parser')

0024: links = soup.find\_all('a', href=True)

0025:

0026: urls = {}

0027: for link in links:

0028: href = link['href']

0029: if palavra\_chave in href:

0030: # Extrai o ano do link e cria a entrada no dicionário

0031: for ano in range(1995, 2024):

0032: if str(ano) in href:

0033: urls[f"INEP\_{ano}-MICRODADOS-CENSO"] = href

0034: break

0035: return urls

0036:

0037: def salvar\_links\_em\_arquivo(urls, caminho\_arquivo):

0038: """

0039: Salva os links em um arquivo de texto no formato especificado.

0040:

0041: :param urls: Dicionário com os links e descrições.

0042: :param caminho\_arquivo: Caminho do arquivo onde os links serão salvos.

0043: """

0044: with open(caminho\_arquivo, 'w') as arquivo:

0045: for chave, url in urls.items():

0046: arquivo.write(f"{chave}: {url}\n")

0047: print(f"Links salvos em {caminho\_arquivo}")

0048:

0049: def main():

0050: url\_base = "https://www.gov.br/inep/pt-br/acesso-a-informacao/dados-abertos/microdados/censo-da-educacao-superior"

0051: palavra\_chave = "microdados\_censo\_da\_educacao\_superior"

0052: caminho\_arquivo = "./dados/bruto/lista-links.txt"

0053:

0054: print("Coletando links do INEP...")

0055: urls = coletar\_links\_inep(url\_base, palavra\_chave)

0056: salvar\_links\_em\_arquivo(urls, caminho\_arquivo)

0057: print("Processo concluído.")

0058:

0059: if \_\_name\_\_ == '\_\_main\_\_':

0060: main()

## coleta\_dados\_oficiais.py

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

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # scripts/coleta\_dados/coleta\_dados\_oficiais.py

0006: # Este script realiza a coleta e processamento de microdados oficiais do INEP e MEC.

0007:

0008: import os

0009: import requests

0010: import pandas as pd

0011: from zipfile import ZipFile

0012:

0013: def baixar\_arquivo(url, destino):

0014: """

0015: Faz o download de um arquivo a partir de uma URL e salva no destino especificado.

0016: """

0017: try:

0018: resposta = requests.get(url, stream=True)

0019: resposta.raise\_for\_status() # Verifica se houve erro na resposta

0020: with open(destino, 'wb') as arquivo:

0021: for chunk in resposta.iter\_content(chunk\_size=8192):

0022: arquivo.write(chunk)

0023: print(f'Download concluído: {destino}')

0024: except requests.RequestException as e:

0025: print(f'Erro ao baixar {url}: {e}')

0026:

0027: def extrair\_arquivo(zip\_caminho, destino\_pasta):

0028: """

0029: Extrai um arquivo ZIP para o destino especificado e ajusta estrutura se necessário.

0030: """

0031: try:

0032: with ZipFile(zip\_caminho, 'r') as zip\_ref:

0033: zip\_ref.extractall(destino\_pasta)

0034: print(f'Arquivos extraídos para: {destino\_pasta}')

0035:

0036: # Verificar estrutura do diretório após extração

0037: for root, dirs, files in os.walk(destino\_pasta):

0038: print(f'Conteúdo de {root}: {dirs}, {files}')

0039: break # Apenas primeiro nível para não poluir o log

0040:

0041: except Exception as e:

0042: print(f'Erro ao extrair {zip\_caminho}: {e}')

0043:

0044: def normalizar\_nomes\_arquivos(caminho\_pasta, ano):

0045: """

0046: Renomeia os arquivos de IES para seguir o formato esperado: MICRODADOS\_ED\_SUP\_IES\_YEAR.CSV.

0047: """

0048: for root, \_, arquivos in os.walk(caminho\_pasta):

0049: for arquivo in arquivos:

0050: if "CADASTRO\_IES" in arquivo.upper() and arquivo.upper().endswith('.CSV'):

0051: caminho\_antigo = os.path.join(root, arquivo)

0052: caminho\_novo = os.path.join(root, f"MICRODADOS\_ED\_SUP\_IES\_{ano}.CSV")

0053: os.rename(caminho\_antigo, caminho\_novo)

0054: print(f'Renomeado: {caminho\_antigo} -> {caminho\_novo}')

0055:

0056: def processar\_microdados(caminho\_pasta):

0057: """

0058: Processa os arquivos de microdados e retorna um DataFrame consolidado.

0059: """

0060: # Normaliza os nomes dos arquivos de IES antes de processar

0061: # Extraindo o ano a partir do nome da pasta (assumindo que a pasta contenha o ano)

0062: ano = os.path.basename(caminho\_pasta).split('\_')[1] if '\_' in os.path.basename(caminho\_pasta) else 'NA'

0063: normalizar\_nomes\_arquivos(caminho\_pasta, ano)

0064: arquivos\_csv = [os.path.join(caminho\_pasta, f) for f in os.listdir(caminho\_pasta) if f.upper().endswith('.CSV')]

0065: df\_lista = []

0066: for arquivo in arquivos\_csv:

0067: try:

0068: df = pd.read\_csv(arquivo, sep=';', encoding='latin1')

0069: df\_lista.append(df)

0070: except Exception as e:

0071: print(f'Erro ao processar {arquivo}: {e}')

0072: if df\_lista:

0073: return pd.concat(df\_lista, ignore\_index=True)

0074: else:

0075: print('Nenhum dado processado.')

0076: return pd.DataFrame()

0077:

0078: def main():

0079: # URLs reais dos microdados

0080: urls = {

0081: 'INEP\_2023-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2023.zip',

0082: 'INEP\_2022-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2022.zip',

0083: 'INEP\_2021-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2021.zip',

0084: 'INEP\_2020-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2020.zip',

0085: 'INEP\_2019-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2019.zip',

0086: 'INEP\_2018-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2018.zip',

0087: 'INEP\_2017-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2017.zip',

0088: 'INEP\_2016-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2016.zip',

0089: 'INEP\_2015-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2015.zip',

0090: 'INEP\_2014-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2014.zip',

0091: 'INEP\_2013-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2013.zip',

0092: 'INEP\_2012-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2012.zip',

0093: 'INEP\_2011-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2011.zip',

0094: 'INEP\_2010-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2010.zip',

0095: 'INEP\_2009-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2009.zip',

0096: # 'INEP\_2008-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2008.zip',

0097: # 'INEP\_2007-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2007.zip',

0098: # 'INEP\_2006-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2006.zip',

0099: # 'INEP\_2005-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2005.zip',

0100: # 'INEP\_2004-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2004.zip',

0101: # 'INEP\_2003-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2003.zip',

0102: # 'INEP\_2002-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2002.zip',

0103: # 'INEP\_2001-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2001.zip',

0104: # 'INEP\_2000-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_2000.zip',

0105: # 'INEP\_1999-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_1999.zip',

0106: # 'INEP\_1998-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_1998.zip',

0107: # 'INEP\_1997-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_1997.zip',

0108: # 'INEP\_1996-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_1996.zip',

0109: # 'INEP\_1995-MICRODADOS-CENSO': 'https://download.inep.gov.br/microdados/microdados\_censo\_da\_educacao\_superior\_1995.zip'

0110: }

0111:

0112: # Caminho para salvar os dados

0113: pasta\_dados = './dados/bruto'

0114: os.makedirs(pasta\_dados, exist\_ok=True)

0115:

0116: for fonte, url in urls.items():

0117: zip\_caminho = os.path.join(pasta\_dados, f'{fonte}.zip')

0118: pasta\_extracao = os.path.join(pasta\_dados, fonte)

0119: os.makedirs(pasta\_extracao, exist\_ok=True)

0120:

0121: # Baixar o arquivo

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

0123: baixar\_arquivo(url, zip\_caminho)

0124:

0125: # Extrair o arquivo

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

0127: extrair\_arquivo(zip\_caminho, pasta\_extracao)

0128:

0129: # Processar os dados

0130: print(f'Processando dados da fonte: {fonte}')

0131: df = processar\_microdados(pasta\_extracao)

0132:

0133: # Salvar o DataFrame consolidado

0134: if not df.empty:

0135: df.to\_csv(os.path.join(pasta\_dados, f'{fonte}\_dados\_brutos.csv'), index=False)

0136: print(f'Dados processados e salvos em: {os.path.join(pasta\_dados, f"{fonte}\_dados\_brutos.csv")}')

0137: else:

0138: print(f'Nenhum dado válido processado para {fonte}.')

0139:

0140: if \_\_name\_\_ == '\_\_main\_\_':

0141: main()

## pre\_processamento.py

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

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # pre\_processamento.py - Script para carregar, padronizar e salvar os microdados do INEP/MEC.

0006: # Processa os dados brutos (IES e Cursos) por ano, realizando limpeza, renomeação e formatação padronizada.

0007:

0008: import os

0009: import pandas as pd

0010: import unicodedata

0011: import re

0012: import numpy as np

0013: from io import StringIO

0014: import glob

0015:

0016: # ==========================================================================

0017: # CONFIGURAÇÕES GERAIS

0018: # ==========================================================================

0019:

0020: # Diretórios de entrada e saída (ajuste conforme sua estrutura)

0021: PASTA\_BRUTO = "./dados/bruto"

0022: PASTA\_PROCESSADO = "./dados/processado"

0023:

0024: # ==========================================================================

0025: # COLUNAS DE INTERESSE E MAPEAMENTOS

0026: # ==========================================================================

0027:

0028: # Exemplo de colunas relevantes para a base de IES, conforme dicionário do Censo 2023.

0029: COLUNAS\_IES\_RELEVANTES = [

0030: "CO\_IES",

0031: "NO\_IES",

0032: # "TP\_REDE",

0033: "TP\_CATEGORIA\_ADMINISTRATIVA",

0034: "QT\_DOC\_TOTAL",

0035: "QT\_DOC\_EXE",

0036: "QT\_DOC\_EX\_FEMI",

0037: "QT\_DOC\_EX\_MASC"

0038: # Adicione outras colunas se precisar (p. ex. QT\_TEC\_TOTAL, etc.)

0039: ]

0040:

0041: MAPPING\_IES = {

0042: "CO\_IES": "id\_ies",

0043: "NO\_IES": "nome\_ies",

0044: # "TP\_REDE": "tipo\_rede", # 1 = pública, 2 = privada

0045: "TP\_CATEGORIA\_ADMINISTRATIVA": "cat\_adm", # 1 = Fed, 2 = Est, etc.

0046: "QT\_DOC\_TOTAL": "docentes\_total",

0047: "QT\_DOC\_EXE": "docentes\_exercicio",

0048: "QT\_DOC\_EX\_FEMI": "docentes\_feminino",

0049: "QT\_DOC\_EX\_MASC": "docentes\_masculino"

0050: }

0051:

0052: # Exemplo de colunas relevantes para a base de Cursos, conforme dicionário do Censo 2023.

0053: COLUNAS\_CURSOS\_RELEVANTES = [

0054: "CO\_IES",

0055: "CO\_CURSO",

0056: "NO\_CURSO",

0057: "TP\_MODALIDADE\_ENSINO",

0058: "QT\_CURSO",

0059: "QT\_VG\_TOTAL",

0060: "QT\_INSCRITO\_TOTAL",

0061: "QT\_ING",

0062: "QT\_MAT",

0063: "QT\_CONC",

0064: "QT\_DIPLOMADOS",

0065: "QT\_DIPLO"

0066: # 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.)

0067: ]

0068:

0069: MAPPING\_CURSOS = {

0070: "CO\_IES": "id\_ies",

0071: "CO\_CURSO": "id\_curso",

0072: "NO\_CURSO": "nome\_curso",

0073: "TP\_MODALIDADE\_ENSINO": "modalidade\_ensino", # 1=Presencial, 2=EAD

0074: "QT\_CURSO": "numero\_cursos",

0075: "QT\_VG\_TOTAL": "vagas\_totais",

0076: "QT\_INSCRITO\_TOTAL": "inscritos\_totais",

0077: "QT\_ING": "ingressantes",

0078: "QT\_MAT": "matriculados",

0079: "QT\_CONC": "concluintes",

0080: "QT\_DIPLOMADOS": "concluintes",

0081: "QT\_DIPLO": "concluintes"

0082: }

0083:

0084: # # Os mapeamentos específicos para determinados anos (1995, 2000, 2008) são definidos

0085: # # como cópias do mapeamento geral para permitir, futuramente, ajustes pontuais nesses períodos.

0086: # MAPPING\_IES\_1995 = MAPPING\_IES.copy()

0087: # COLUNAS\_IES\_RELEVANTES\_1995 = COLUNAS\_IES\_RELEVANTES.copy()

0088:

0089: # MAPPING\_CURSOS\_1995 = MAPPING\_CURSOS.copy()

0090: # COLUNAS\_CURSOS\_RELEVANTES\_1995 = COLUNAS\_CURSOS\_RELEVANTES.copy()

0091:

0092: # MAPPING\_IES\_2000 = MAPPING\_IES.copy()

0093: # COLUNAS\_IES\_RELEVANTES\_2000 = COLUNAS\_IES\_RELEVANTES.copy()

0094:

0095: # MAPPING\_CURSOS\_2000 = MAPPING\_CURSOS.copy()

0096: # COLUNAS\_CURSOS\_RELEVANTES\_2000 = COLUNAS\_CURSOS\_RELEVANTES.copy()

0097:

0098: # MAPPING\_IES\_2008 = MAPPING\_IES.copy()

0099: # COLUNAS\_IES\_RELEVANTES\_2008 = COLUNAS\_IES\_RELEVANTES.copy()

0100: # MAPPING\_CURSOS\_2008 = MAPPING\_CURSOS.copy()

0101: # COLUNAS\_CURSOS\_RELEVANTES\_2008 = COLUNAS\_CURSOS\_RELEVANTES.copy()

0102:

0103: # ==========================================================================

0104: # FUNÇÕES DE APOIO

0105: # ==========================================================================

0106:

0107: def normalizar\_conteudo\_pipe(conteudo):

0108: """

0109: Remove repetições de delimitadores, espaços desnecessários e caracteres indesejados

0110: para uniformizar o uso do pipe ("|") como delimitador.

0111: """

0112: conteudo = re.sub(r"\|{2,}", "|", conteudo)

0113: conteudo = re.sub(r'\s\*\|\s\*', '|', conteudo)

0114: conteudo = conteudo.replace('\r\n', '\n').replace('\r', '\n')

0115: conteudo = conteudo.replace('"', '')

0116: return conteudo

0117:

0118: def registrar\_problemas(arquivo, erro):

0119: """

0120: Registra problemas de leitura em um log.

0121: """

0122: with open("log\_erros.txt", "a", encoding="utf-8") as log:

0123: log.write(f"Arquivo: {arquivo}, Erro: {erro}\n")

0124:

0125: def carregar\_csv(caminho\_arquivo, sep=";", encoding="latin1", year=None):

0126: """

0127: Carrega um CSV, tratando parsing e erros.

0128: Carrega um arquivo CSV a partir de um caminho, tratando a normalização dos delimitadores.

0129:

0130: Se o ano for menor ou igual a 2008 e o conteúdo apresentar o delimitador "|", tenta

0131: normalizar repetições de delimitadores (por exemplo, '||' ou '|||') para que os dados sejam

0132: lidos corretamente. Se mesmo assim o DataFrame resultar em apenas uma coluna, poderá ser necessário

0133: separar manualmente essa coluna.

0134: """

0135: try:

0136: print(f"Lendo arquivo: {caminho\_arquivo}")

0137: with open(caminho\_arquivo, encoding=encoding) as f:

0138: conteudo = f.read()

0139:

0140: if year is not None and year <= 2008 and "|" in conteudo:

0141: print(f"⚠️ Detecção de separadores múltiplos para ano {year}. Normalizando...")

0142: conteudo = normalizar\_conteudo\_pipe(conteudo)

0143: df = pd.read\_csv(StringIO(conteudo), sep="|", header=0, engine="python", on\_bad\_lines='skip')

0144: if df.shape[1] == 1:

0145: # Se restar somente uma coluna, tenta separar manualmente essa coluna usando o delimitador "|"

0146: print("⚠️ Apenas uma coluna detectada após normalização. Tentando separar manualmente...")

0147: df = df.iloc[:, 0].str.split("|", expand=True)

0148: # Assume-se que a primeira linha são os cabeçalhos

0149: df.columns = df.iloc[0]

0150: df = df[1:]

0151: return df

0152: else:

0153: return pd.read\_csv(StringIO(conteudo), sep=sep, header=0, engine="python", on\_bad\_lines='skip')

0154: except Exception as e:

0155: print(f"Erro ao carregar {caminho\_arquivo}: {e}")

0156: registrar\_problemas(caminho\_arquivo, e)

0157: return pd.DataFrame()

0158:

0159: def filtrar\_renomear(df, colunas\_relevantes, mapping):

0160: """

0161: Seleciona apenas as colunas relevantes contidas no DataFrame e as renomeia conforme

0162: o dicionário de mapeamento fornecido.

0163: """

0164: # Identifica somente colunas que existam no df

0165: existentes = [c for c in colunas\_relevantes if c in df.columns]

0166: df\_filtrado = df[existentes].copy()

0167: # Renomeia

0168: return df\_filtrado.rename(columns=mapping)

0169:

0170: def corrigir\_nome\_pasta(caminho\_base, ano):

0171: """

0172: Corrige possíveis problemas com caracteres especiais nos nomes das pastas extraídas,

0173: retornando o caminho completo da pasta que contenha a string "microdados".

0174: """

0175: caminho\_esperado = os.path.join(caminho\_base, f"INEP\_{ano}-MICRODADOS-CENSO")

0176: if os.path.exists(caminho\_esperado):

0177: for pasta in os.listdir(caminho\_esperado):

0178: pasta\_corrigida = unicodedata.normalize("NFKD", pasta).encode("ASCII", "ignore").decode("ASCII")

0179: pasta\_corrigida = re.sub(r'[^a-zA-Z0-9\_\- ]', '', pasta\_corrigida)

0180: if "microdados" in pasta\_corrigida.lower():

0181: return os.path.join(caminho\_esperado, pasta)

0182: print(f"Aviso: Nenhuma pasta de microdados encontrada para {ano}")

0183: return None

0184:

0185: def corrigir\_nome\_arquivo(nome\_arquivo):

0186: """

0187: Corrige caracteres especiais no nome dos arquivos, removendo aqueles inválidos.

0188: """

0189: return re.sub(r'[^a-zA-Z0-9\_\-\. ]', '', nome\_arquivo)

0190:

0191: # ==========================================================================

0192: # PROCESSAMENTO PRINCIPAL

0193: # ==========================================================================

0194:

0195: def main(year: int = 2024):

0196: arquivos\_disponiveis = []

0197: caminho\_base\_ano = corrigir\_nome\_pasta(PASTA\_BRUTO, year)

0198: caminho\_dados = os.path.join(caminho\_base\_ano, "dados")

0199:

0200: if os.path.isdir(caminho\_dados):

0201: arquivos\_disponiveis = os.listdir(caminho\_dados)

0202:

0203: if year < 2009:

0204: # Renomeia arquivos com padrões diferentes de nomenclatura para padronizá-los

0205: for arq in arquivos\_disponiveis:

0206: if "INSTITUICAO" in arq.upper():

0207: os.rename(

0208: os.path.join(caminho\_dados, arq),

0209: os.path.join(caminho\_dados, f"MICRODADOS\_ED\_SUP\_IES\_{year}.CSV")

0210: )

0211: elif "GRADUACAO\_PRESENCIAL" in arq.upper():

0212: os.rename(

0213: os.path.join(caminho\_dados, arq),

0214: os.path.join(caminho\_dados, f"MICRODADOS\_CADASTRO\_CURSOS\_{year}.CSV")

0215: )

0216: arquivos\_disponiveis = os.listdir(caminho\_dados)

0217:

0218: ARQUIVO\_IES = f"MICRODADOS\_ED\_SUP\_IES\_{year}.CSV"

0219: ARQUIVO\_CURSOS = f"MICRODADOS\_CADASTRO\_CURSOS\_{year}.CSV"

0220:

0221: caminho\_ies = None

0222: for arquivo in arquivos\_disponiveis:

0223: nome\_normalizado = corrigir\_nome\_arquivo(arquivo).upper()

0224: if nome\_normalizado.startswith(f"MICRODADOS\_ED\_SUP\_IES\_{year}") and nome\_normalizado.endswith(".CSV"):

0225: caminho\_ies = os.path.join(caminho\_dados, arquivo)

0226: break

0227:

0228: # Fallback para nome padrão, se não encontrar com sufixo

0229: if caminho\_ies is None:

0230: caminho\_ies = os.path.join(caminho\_dados, f"MICRODADOS\_ED\_SUP\_IES\_{year}.CSV")

0231:

0232: print(f"Arquivos encontrados em {caminho\_dados}: {arquivos\_disponiveis}")

0233: print(f"Arquivo IES esperado: {ARQUIVO\_IES}")

0234: print(f"Arquivo IES identificado: {caminho\_ies if os.path.exists(caminho\_ies) else 'NÃO ENCONTRADO'}")

0235:

0236: df\_ies\_final = pd.DataFrame()

0237: df\_cursos\_final = pd.DataFrame()

0238:

0239: # ----------------------------------------------------------------------

0240: # Carregar e processar MICRODADOS\_ED\_SUP\_IES\_{year}.CSV

0241: # ----------------------------------------------------------------------

0242: if os.path.isfile(caminho\_ies):

0243: df\_ies = carregar\_csv(caminho\_ies, year=year)

0244: if not df\_ies.empty:

0245: df\_ies\_final = filtrar\_renomear(df\_ies, COLUNAS\_IES\_RELEVANTES, MAPPING\_IES)

0246: else:

0247: print(f"Aviso: {ARQUIVO\_IES} está vazio ou não pôde ser processado.")

0248: else:

0249: print(f"Aviso: Arquivo {ARQUIVO\_IES} não encontrado em {caminho\_ies}.")

0250:

0251: # ----------------------------------------------------------------------

0252: # Carregar e processar MICRODADOS\_CADASTRO\_CURSOS\_{year}.CSV

0253: # ----------------------------------------------------------------------

0254: caminho\_cursos = os.path.join(caminho\_dados, corrigir\_nome\_arquivo(ARQUIVO\_CURSOS))

0255: if os.path.isfile(caminho\_cursos):

0256: df\_cursos = carregar\_csv(caminho\_cursos, year=year)

0257: if not df\_cursos.empty:

0258: df\_cursos\_final = filtrar\_renomear(df\_cursos, COLUNAS\_CURSOS\_RELEVANTES, MAPPING\_CURSOS)

0259: else:

0260: print(f"Aviso: {ARQUIVO\_CURSOS} está vazio ou não pôde ser processado.")

0261: else:

0262: print(f"Aviso: Arquivo {ARQUIVO\_CURSOS} não encontrado em {caminho\_cursos}.")

0263:

0264: # ----------------------------------------------------------------------

0265: # Salvando resultados (IES e Cursos) na pasta "processado"

0266: # ----------------------------------------------------------------------

0267: if not df\_ies\_final.empty:

0268: saida\_ies = os.path.join(PASTA\_PROCESSADO, f"dados\_ies\_{year}.csv")

0269: os.makedirs(os.path.dirname(saida\_ies), exist\_ok=True)

0270: df\_ies\_final.to\_csv(saida\_ies, sep=";", index=False, encoding="utf-8")

0271: print(f"[OK] dados\_ies gerado em: {saida\_ies}")

0272: else:

0273: print("Nenhum dado de IES para salvar.")

0274:

0275: if not df\_cursos\_final.empty:

0276: saida\_cursos = os.path.join(PASTA\_PROCESSADO, f"dados\_cursos\_{year}.csv")

0277: os.makedirs(os.path.dirname(saida\_cursos), exist\_ok=True)

0278: df\_cursos\_final.to\_csv(saida\_cursos, sep=";", index=False, encoding="utf-8")

0279: print(f"[OK] dados\_cursos gerado em: {saida\_cursos}")

0280: else:

0281: print("Nenhum dado de Cursos para salvar.")

0282:

0283: if \_\_name\_\_ == "\_\_main\_\_":

0284: # Processa todos os anos de 2009 a 2023.

0285: for year in range(2009, 2024):

0286: print(f"\tProcessing year {year} ...")

0287: main(year)

## pre\_processamento\_Transfer\_Learn.py

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

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # scripts/processamento\_dados/pre\_processamento\_Transfer\_Learn.py

0006: # Este script prepara um CSV limpo e padronizado para uso nos modelos de Transfer Learning e Fine-tuning

0007:

0008: import pandas as pd

0009: import os

0010:

0011: def preprocessar\_transfer\_learning(caminho\_entrada, caminho\_saida):

0012: df = pd.read\_csv(caminho\_entrada, sep=None, engine='python')

0013:

0014: # Seleciona apenas as colunas de interesse e renomeia se necessário

0015: colunas\_validas = ['taxa\_ingresso', 'vagas\_totais', 'taxa\_evasao']

0016: df = df[[col for col in colunas\_validas if col in df.columns]]

0017:

0018: # Remove linhas com valores ausentes ou inválidos

0019: df = df.dropna()

0020:

0021: # Garante que todas as colunas estão em float

0022: df = df.astype(float)

0023:

0024: os.makedirs(os.path.dirname(caminho\_saida), exist\_ok=True)

0025: df.to\_csv(caminho\_saida, index=False)

0026: print(f"Base processada salva em: {caminho\_saida}")

0027:

0028:

0029: if \_\_name\_\_ == '\_\_main\_\_':

0030: entrada = './dados/processado/dados\_transfer\_learning.csv'

0031: saida = './dados/processado/dados\_transfer\_learning\_clean.csv'

0032: preprocessar\_transfer\_learning(entrada, saida)

## tratar\_dados.py

0001: # Path: scripts/processamento\_dados/processamento\_dados/tratar\_dados.py

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # tratar\_dados.py

0006: # Script responsável pelo carregamento, limpeza, tratamento, cálculo de taxas e consolidação de dados educacionais extraídos dos microdados do INEP/MEC.

0007:

0008: import os

0009: import pandas as pd

0010: from pathlib import Path

0011: import re

0012: import glob

0013: import numpy as np

0014:

0015: # Defina a variável global para a pasta processada

0016: PASTA\_PROCESSADO = "./dados/processado"

0017:

0018: def carregar\_dados(caminho\_entrada):

0019: try:

0020: print(f"Carregando dados de: {caminho\_entrada}")

0021: df = pd.read\_csv(caminho\_entrada, sep=';', encoding='utf-8', low\_memory=False)

0022: print("Colunas disponíveis:", df.columns.tolist())

0023: return df

0024: except Exception as e:

0025: raise ValueError(f"Erro ao carregar os dados: {e}")

0026:

0027: def tratar\_dados(df, colunas\_numericas=None):

0028: df = df.drop\_duplicates()

0029: df = df.dropna()

0030:

0031: if colunas\_numericas:

0032: for col in colunas\_numericas:

0033: if col in df.columns:

0034: df[col] = pd.to\_numeric(df[col], errors='coerce')

0035:

0036: for col in ['ingressantes', 'concluintes', 'vagas\_totais', 'matriculados', 'numero\_cursos']:

0037: if col in df.columns:

0038: df = df[df[col] >= 0]

0039:

0040: if 'ingressantes' in df.columns:

0041: df = df[df['ingressantes'] > 0]

0042:

0043: df = df.dropna()

0044:

0045: return df

0046:

0047: def salvar\_dados\_tratados(df, caminho\_saida):

0048: try:

0049: os.makedirs(os.path.dirname(caminho\_saida), exist\_ok=True)

0050: df.to\_csv(caminho\_saida, index=False, sep=';', encoding='utf-8')

0051: print(f"Dados tratados salvos em: {caminho\_saida}")

0052: except Exception as e:

0053: raise ValueError(f"Erro ao salvar os dados: {e}")

0054:

0055: def pivotar\_dados\_cursos():

0056: arquivos = sorted(Path("./dados/processado").glob("dados\_cursos\_tratado\_\*.csv"))

0057: dfs = []

0058: for arq in arquivos:

0059: ano\_match = re.search(r"(\d{4})", arq.name)

0060: if not ano\_match:

0061: continue

0062: ano = int(ano\_match.group(1))

0063: df = pd.read\_csv(arq, sep=';', encoding='utf-8')

0064: df['ano'] = ano

0065: dfs.append(df)

0066:

0067: if not dfs:

0068: return

0069:

0070: df\_geral = pd.concat(dfs, ignore\_index=True)

0071:

0072: id\_cols = ["id\_curso", "nome\_curso", "modalidade\_ensino", "id\_ies"]

0073: id\_cols = [col for col in id\_cols if col in df\_geral.columns]

0074:

0075: df\_pivot = pd.DataFrame()

0076: for var in ["ingressantes", "concluintes", "matriculados", "vagas\_totais", "inscritos\_totais"]:

0077: if var in df\_geral.columns:

0078: tabela = df\_geral.pivot\_table(index=id\_cols, columns="ano", values=var)

0079: tabela.columns = [f"{var}\_{int(col)}" for col in tabela.columns]

0080: df\_pivot = pd.concat([df\_pivot, tabela], axis=1)

0081:

0082: df\_final = df\_geral[id\_cols].drop\_duplicates().set\_index(id\_cols)

0083: df\_final = df\_final.join(df\_pivot).reset\_index()

0084:

0085: salvar\_dados\_tratados(df\_final, "./dados/processado/dados\_cursos\_serie\_temporal.csv")

0086:

0087: def calcular\_taxas(df):

0088: df = df.copy()

0089: for coluna in ['ingressantes', 'concluintes', 'vagas\_totais']:

0090: if coluna in df.columns:

0091: df[coluna] = pd.to\_numeric(df[coluna], errors='coerce')

0092:

0093: df = df[(df['ingressantes'] > 0) & (df['vagas\_totais'] > 0)]

0094: df['taxa\_ingresso'] = df['ingressantes'] / df['vagas\_totais']

0095: df['taxa\_conclusao'] = df['concluintes'] / df['ingressantes']

0096: df['taxa\_evasao'] = 1 - df['taxa\_conclusao']

0097:

0098: for col in ['taxa\_ingresso', 'taxa\_conclusao', 'taxa\_evasao']:

0099: df = df[(df[col] >= 0) & (df[col] <= 1)]

0100:

0101: return df

0102:

0103: def salvar\_taxas\_consolidadas():

0104: pattern = os.path.join(PASTA\_PROCESSADO, "dados\_cursos\_tratado\_\*.csv")

0105: files = glob.glob(pattern)

0106: if not files:

0107: print("Nenhum arquivo de cursos tratado foi encontrado para consolidar.")

0108: return

0109: list\_df = []

0110: for f in files:

0111: try:

0112: df\_temp = pd.read\_csv(f, sep=";", encoding="utf-8")

0113: list\_df.append(df\_temp)

0114: except Exception as e:

0115: print(f"Erro ao ler o arquivo {f}: {e}")

0116: if not list\_df:

0117: print("Nenhum dado foi carregado para consolidação.")

0118: return

0119: df\_consolidado = pd.concat(list\_df, ignore\_index=True)

0120: df\_consolidado = calcular\_taxas(df\_consolidado)

0121: caminho\_saida = os.path.join(PASTA\_PROCESSADO, "dados\_ingresso\_evasao\_conclusao.csv")

0122: salvar\_dados\_tratados(df\_consolidado, caminho\_saida)

0123: print(f"[OK] Dados consolidados e taxas salvos em: {caminho\_saida}")

0124:

0125: def ler\_taxas\_consolidadas():

0126: caminho = os.path.join(PASTA\_PROCESSADO, "dados\_ingresso\_evasao\_conclusao.csv")

0127: try:

0128: df = pd.read\_csv(caminho, sep=";", encoding="utf-8")

0129: print("Dados consolidados lidos com sucesso.")

0130: return df

0131: except Exception as e:

0132: raise ValueError(f"Erro ao ler o arquivo de taxas consolidadas: {e}")

0133:

0134: def main(year: int = 2024):

0135: caminho\_ies = f'./dados/processado/dados\_ies\_{year}.csv'

0136: caminho\_cursos = f'./dados/processado/dados\_cursos\_{year}.csv'

0137:

0138: caminho\_ies\_tratado = f'./dados/intermediario/dados\_ies\_tratado\_{year}.csv'

0139: caminho\_cursos\_tratado = f'./dados/intermediario/dados\_cursos\_tratado\_{year}.csv'

0140:

0141: caminho\_ies\_final = f'./dados/processado/dados\_ies\_tratado\_{year}.csv'

0142: caminho\_cursos\_final = f'./dados/processado/dados\_cursos\_tratado\_{year}.csv'

0143:

0144: colunas\_numericas\_ies = [

0145: 'docentes\_total',

0146: 'docentes\_exercicio',

0147: 'docentes\_feminino',

0148: 'docentes\_masculino'

0149: ]

0150: colunas\_numericas\_cursos = [

0151: 'numero\_cursos',

0152: 'vagas\_totais',

0153: 'inscritos\_totais',

0154: 'ingressantes',

0155: 'matriculados',

0156: 'concluintes'

0157: ]

0158:

0159: try:

0160: df\_ies = carregar\_dados(caminho\_ies)

0161: except ValueError as e:

0162: print(e)

0163: df\_ies = pd.DataFrame()

0164:

0165: if not df\_ies.empty:

0166: try:

0167: df\_ies\_tratado = tratar\_dados(df\_ies, colunas\_numericas=colunas\_numericas\_ies)

0168: salvar\_dados\_tratados(df\_ies\_tratado, caminho\_ies\_tratado)

0169: salvar\_dados\_tratados(df\_ies\_tratado, caminho\_ies\_final)

0170: except ValueError as e:

0171: print(f"Erro ao processar dados de IES: {e}")

0172: else:

0173: print("Nenhum dado de IES disponível para tratar.")

0174:

0175: try:

0176: df\_cursos = carregar\_dados(caminho\_cursos)

0177: except ValueError as e:

0178: print(e)

0179: df\_cursos = pd.DataFrame()

0180:

0181: if not df\_cursos.empty:

0182: try:

0183: df\_cursos\_tratado = tratar\_dados(df\_cursos, colunas\_numericas=colunas\_numericas\_cursos)

0184: salvar\_dados\_tratados(df\_cursos\_tratado, caminho\_cursos\_tratado)

0185: salvar\_dados\_tratados(df\_cursos\_tratado, caminho\_cursos\_final)

0186: except ValueError as e:

0187: print(f"Ano de {year} Erro ao processar dados de Cursos: {e}")

0188: else:

0189: print("Nenhum dado de Cursos disponível para tratar.")

0190:

0191: if \_\_name\_\_ == '\_\_main\_\_':

0192: for year in range(2024):

0193: print(f"\tProcessing year {year} ...")

0194: main(year)

0195:

0196: pivotar\_dados\_cursos()

0197: salvar\_taxas\_consolidadas()

0198: df\_taxas = ler\_taxas\_consolidadas()

0199: print(df\_taxas.head())

0200:

## tratar\_dados\_Transfer\_Learn.py

0001: # Path: scripts/processamento\_dados/processamento\_dados/tratar\_dados\_Transfer\_Learn.py

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # Script: tratar\_dados\_Transfer\_Learn.py

0006: # Finalidade: Este script realiza o pré-processamento de dados para técnicas de Transfer Learning.

0007: # Ele lê a base limpa `dados\_transfer\_learning\_clean.csv`, verifica colunas essenciais, remove dados ausentes/duplicados

0008: # e gera dois arquivos com transformações diferentes: normalização MinMax e padronização Z-Score.

0009: # Saídas: `dados\_transfer\_normalizado.csv` e `dados\_transfer\_padronizado.csv` no diretório `dados/processado/`.

0010: # Etapa do pipeline: 2ª etapa (após pre\_processamento\_Transfer\_Learn.py e antes do treino dos modelos).

0011: import pandas as pd

0012: import os

0013: from sklearn.preprocessing import MinMaxScaler, StandardScaler

0014:

0015: # Caminho de entrada e saída

0016: caminho\_entrada = './dados/processado/dados\_transfer\_learning\_clean.csv'

0017: caminho\_saida = './dados/processado/'

0018:

0019: def main():

0020: print("🔍 Lendo arquivo de entrada...")

0021: df = pd.read\_csv(caminho\_entrada)

0022:

0023: print("🔧 Verificando colunas necessárias...")

0024: colunas\_esperadas = ['taxa\_ingresso', 'vagas\_totais', 'taxa\_evasao']

0025: for col in colunas\_esperadas:

0026: if col not in df.columns:

0027: raise ValueError(f"Coluna obrigatória ausente: {col}")

0028:

0029: print("🧼 Removendo linhas com valores ausentes ou duplicadas...")

0030: df = df[colunas\_esperadas].dropna().drop\_duplicates()

0031:

0032: # ===== Normalização Min-Max =====

0033: print("📉 Aplicando normalização MinMaxScaler...")

0034: scaler\_minmax = MinMaxScaler()

0035: df\_minmax = pd.DataFrame(scaler\_minmax.fit\_transform(df), columns=colunas\_esperadas)

0036: df\_minmax.to\_csv(os.path.join(caminho\_saida, 'dados\_transfer\_normalizado.csv'), index=False)

0037:

0038: # ===== Padronização Z-Score =====

0039: print("📊 Aplicando padronização StandardScaler...")

0040: scaler\_zscore = StandardScaler()

0041: df\_zscore = pd.DataFrame(scaler\_zscore.fit\_transform(df), columns=colunas\_esperadas)

0042: df\_zscore.to\_csv(os.path.join(caminho\_saida, 'dados\_transfer\_padronizado.csv'), index=False)

0043:

0044: print("✅ Arquivos gerados com sucesso:")

0045: print(f" - {os.path.join(caminho\_saida, 'dados\_transfer\_normalizado.csv')}")

0046: print(f" - {os.path.join(caminho\_saida, 'dados\_transfer\_padronizado.csv')}")

0047:

0048: if \_\_name\_\_ == '\_\_main\_\_':

0049: main()

## preparar\_entrada\_modelos.py

0001: # Path: scripts/processamento\_dados/preparar\_entrada\_modelos.py

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # Este script prepara os dados de entrada para o treinamento de modelos preditivos.

0006: # Ele lê a base tratada sem vazamento de dados, verifica se as colunas esperadas estão presentes,

0007: # seleciona as features relevantes ('taxa\_ingresso' e 'vagas\_totais') e, se disponível,

0008: # inclui a coluna 'taxa\_evasao' como target. O resultado é salvo como CSV pronto para modelagem.

0009: import pandas as pd

0010: import os

0011:

0012: # Caminho de entrada e saída

0013: caminho\_entrada = 'dados/processado/dados\_transfer\_learning\_clean.csv'

0014: caminho\_saida = 'dados/processado/entrada\_modelos.csv'

0015:

0016: print("📥 Lendo base tratada para preparar entrada dos modelos...")

0017: try:

0018: df = pd.read\_csv(caminho\_entrada)

0019: print(df.columns.tolist())

0020: print("📋 Colunas disponíveis:", df.columns.tolist())

0021:

0022: colunas\_entrada = ['taxa\_ingresso', 'vagas\_totais']

0023:

0024: if not all(col in df.columns for col in colunas\_entrada):

0025: raise ValueError(f"❌ Colunas necessárias não encontradas: {colunas\_entrada}")

0026:

0027: # Seleciona apenas features

0028: df\_modelo = df[colunas\_entrada].copy()

0029: # df\_modelo.columns = ['feature1', 'feature2']

0030:

0031: if 'taxa\_evasao' in df.columns:

0032: df\_modelo['target'] = df['taxa\_evasao']

0033: else:

0034: raise ValueError("❌ Coluna 'taxa\_evasao' não encontrada para definir como target.")

0035:

0036: # Salva em CSV

0037: os.makedirs(os.path.dirname(caminho\_saida), exist\_ok=True)

0038: df\_modelo.to\_csv(caminho\_saida, index=False)

0039: print(f"✅ Arquivo de entrada para os modelos salvo em: {caminho\_saida}")

0040:

0041: except Exception as e:

0042: print(f"❌ Erro ao preparar entrada dos modelos: {e}")

## gerar\_base\_modelo.py

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

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # scripts/modelagem/gerar\_base\_modelo.py

0006: from tensorflow.keras.models import Sequential

0007: from tensorflow.keras.layers import Dense, Input

0008: from tensorflow.keras.losses import MeanSquaredError

0009: from tensorflow.keras.metrics import MeanAbsoluteError

0010:

0011: model = Sequential([

0012: Input(shape=(2,)),

0013: Dense(64, activation='relu'),

0014: Dense(32, activation='relu'),

0015: Dense(16, activation='relu'),

0016: Dense(1, activation='linear')

0017: ])

0018:

0019: model.compile(optimizer='adam', loss=MeanSquaredError(), metrics=[MeanAbsoluteError()])

0020: model.save('modelos/base\_modelo\_neural.h5')

## treinamento\_modelo\_original.py

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

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # /scripts/modelagem/treinamento\_modelo.py

0006: # Este script treina modelos de aprendizado de máquina para prever a taxa de evasão,

0007: # bem como explorar as relações entre taxa de ingresso, taxa de evasão e taxa de conclusão,

0008: # e realizar previsões para ingressantes e concluintes.

0009:

0010: import pandas as pd

0011: from sklearn.model\_selection import train\_test\_split

0012: from sklearn.ensemble import RandomForestRegressor

0013: from sklearn.linear\_model import LinearRegression

0014: from sklearn.metrics import mean\_squared\_error, r2\_score, confusion\_matrix

0015: import joblib

0016: import os

0017: import matplotlib.pyplot as plt

0018: import seaborn as sns

0019:

0020: def main():

0021: # Caminho para os dados processados

0022: caminho\_dados = './dados/processado/dados\_ingresso\_evasao\_conclusao.csv'

0023: df = pd.read\_csv(caminho\_dados, sep=';', encoding='utf-8', low\_memory=False)

0024:

0025: # Definir caminho\_modelo (com caminho relativo) antes de utilizá-lo em qualquer parte

0026: caminho\_modelo = './modelos/modelos\_salvos'

0027: os.makedirs(caminho\_modelo, exist\_ok=True)

0028:

0029: # Treinamento para prever ingressantes e concluintes

0030: if all(col in df.columns for col in ['numero\_cursos', 'vagas', 'inscritos', 'docentes', 'ingressantes', 'concluintes']):

0031: print("Treinando modelos para ingressantes e concluintes...")

0032:

0033: # Features e targets para previsões de ingressantes e concluintes

0034: X = df[['numero\_cursos', 'vagas', 'inscritos', 'docentes']]

0035: y\_ingressantes = df['ingressantes']

0036: y\_concluintes = df['concluintes']

0037:

0038: # Divisão dos dados para ingressantes

0039: 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)

0040:

0041: # Modelo Random Forest para ingressantes

0042: modelo\_ingressantes = RandomForestRegressor(random\_state=42)

0043: modelo\_ingressantes.fit(X\_train\_ing, y\_train\_ing)

0044: score\_ingressantes = modelo\_ingressantes.score(X\_test\_ing, y\_test\_ing)

0045: print(f"Score para ingressantes: {score\_ingressantes:.4f}")

0046:

0047: # Divisão dos dados para concluintes

0048: 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)

0049:

0050: # Modelo Random Forest para concluintes

0051: modelo\_concluintes = RandomForestRegressor(random\_state=42)

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

0053: score\_concluintes = modelo\_concluintes.score(X\_test\_conc, y\_test\_conc)

0054: print(f"Score para concluintes: {score\_concluintes:.4f}")

0055:

0056: # Salvar modelos de ingressantes e concluintes

0057: joblib.dump(modelo\_ingressantes, os.path.join(caminho\_modelo, 'modelo\_ingressantes.pkl'))

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

0059:

0060: # Treinamento para taxas de evasão

0061: print("Treinando modelos para taxa de evasão...")

0062:

0063: # Filtrar as variáveis relevantes

0064: colunas\_relevantes = ['taxa\_ingresso', 'taxa\_evasao', 'taxa\_conclusao']

0065: df = df[colunas\_relevantes].dropna()

0066:

0067: # Dividir as features e o target

0068: X = df[['taxa\_ingresso', 'taxa\_conclusao']]

0069: y = df['taxa\_evasao']

0070:

0071: # Dividir os dados em treino e teste

0072: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

0073:

0074: # Treinar dois modelos: Regressão Linear e Random Forest

0075: # Modelo 1: Regressão Linear

0076: modelo\_linear = LinearRegression()

0077: modelo\_linear.fit(X\_train, y\_train)

0078: y\_pred\_linear = modelo\_linear.predict(X\_test)

0079: mse\_linear = mean\_squared\_error(y\_test, y\_pred\_linear)

0080: r2\_linear = r2\_score(y\_test, y\_pred\_linear)

0081:

0082: # Modelo 2: Random Forest

0083: modelo\_rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

0084: modelo\_rf.fit(X\_train, y\_train)

0085: y\_pred\_rf = modelo\_rf.predict(X\_test)

0086: mse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf)

0087: r2\_rf = r2\_score(y\_test, y\_pred\_rf)

0088:

0089: # Escolher o melhor modelo baseado no R²

0090: melhor\_modelo = modelo\_rf if r2\_rf > r2\_linear else modelo\_linear

0091: nome\_melhor\_modelo = 'Random Forest' if r2\_rf > r2\_linear else 'Regressão Linear'

0092:

0093: # Salvar o modelo escolhido

0094: joblib.dump(melhor\_modelo, os.path.join(caminho\_modelo, 'modelo\_melhor\_evasao.pkl'))

0095:

0096: # Salvar as métricas dos modelos

0097: caminho\_metricas = './modelos/resultados\_modelos'

0098: os.makedirs(caminho\_metricas, exist\_ok=True)

0099: with open(os.path.join(caminho\_metricas, 'metricas\_modelos.txt'), 'w') as f:

0100: f.write(f"Modelo: {nome\_melhor\_modelo}\n")

0101: f.write(f"MSE - Regressão Linear: {mse\_linear:.4f}, R²: {r2\_linear:.4f}\n")

0102: f.write(f"MSE - Random Forest: {mse\_rf:.4f}, R²: {r2\_rf:.4f}\n")

0103: f.write(f"Melhor Modelo Selecionado: {nome\_melhor\_modelo}\n")

0104:

0105: # Geração da Matriz de Confusão (binarizando a taxa de evasão com threshold de 0.5)

0106: threshold = 0.5

0107: y\_test\_class = (y\_test >= threshold).astype(int)

0108: y\_pred\_class = (melhor\_modelo.predict(X\_test) >= threshold).astype(int)

0109: cm = confusion\_matrix(y\_test\_class, y\_pred\_class)

0110: plt.figure(figsize=(6, 4))

0111: sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

0112: plt.title(f"Matriz de Confusão - {nome\_melhor\_modelo}")

0113: plt.xlabel("Predito")

0114: plt.ylabel("Verdadeiro")

0115: caminho\_cm = os.path.join(caminho\_metricas, 'matriz\_confusao.png')

0116: plt.savefig(caminho\_cm)

0117: plt.close()

0118:

0119: print(f"Treinamento concluído. Melhor modelo para evasão: {nome\_melhor\_modelo}")

0120: print(f"Métricas salvas em: {caminho\_metricas}")

0121: print(f"Modelos salvos em: {caminho\_modelo}")

0122:

0123: if \_\_name\_\_ == '\_\_main\_\_':

0124: main()

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

0001: # Path: scripts/modelagem/treinamento\_modelo\_C4.5\_Tree\_J48.py

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # Script: treinamento\_modelo\_C4.5\_Tree\_J48.py

0006: # Descrição: Treina modelos de regressão para prever a taxa de evasão acadêmica

0007: # utilizando as variáveis 'taxa\_ingresso' e 'vagas\_totais' como preditoras.

0008: # Avalia o desempenho de Regressão Linear e Random Forest, seleciona o melhor

0009: # modelo com base no R², e salva o modelo e as métricas em arquivos apropriados.

0010:

0011: import pandas as pd

0012: from sklearn.model\_selection import train\_test\_split

0013: from sklearn.linear\_model import LinearRegression

0014: from sklearn.ensemble import RandomForestRegressor

0015: from sklearn.metrics import mean\_squared\_error, r2\_score

0016: import joblib

0017: import os

0018:

0019: print("Treinando modelos para taxa de evasão...")

0020:

0021: # Carregar dados tratados

0022: df = pd.read\_csv('dados/processado/entrada\_modelos.csv')

0023:

0024: X = df[['taxa\_ingresso', 'vagas\_totais']]

0025: y = df['target']

0026:

0027: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

0028:

0029: # Regressão Linear

0030: modelo\_lr = LinearRegression()

0031: modelo\_lr.fit(X\_train, y\_train)

0032: y\_pred\_lr = modelo\_lr.predict(X\_test)

0033: mse\_lr = mean\_squared\_error(y\_test, y\_pred\_lr)

0034: r2\_lr = r2\_score(y\_test, y\_pred\_lr)

0035:

0036: # Random Forest

0037: modelo\_rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

0038: modelo\_rf.fit(X\_train, y\_train)

0039: y\_pred\_rf = modelo\_rf.predict(X\_test)

0040: mse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf)

0041: r2\_rf = r2\_score(y\_test, y\_pred\_rf)

0042:

0043: # Escolher melhor modelo

0044: melhor\_modelo = modelo\_lr if r2\_lr > r2\_rf else modelo\_rf

0045: melhor\_nome = "Regressão Linear" if r2\_lr > r2\_rf else "Random Forest"

0046:

0047: # Criar pastas

0048: os.makedirs("modelos/modelos\_salvos", exist\_ok=True)

0049: os.makedirs("modelos/resultados\_modelos", exist\_ok=True)

0050:

0051: # Salvar melhor modelo

0052: joblib.dump(melhor\_modelo, 'modelos/modelos\_salvos/modelo\_melhor\_evasao.pkl')

0053:

0054: # Salvar métricas

0055: with open("modelos/resultados\_modelos/metricas\_modelos.txt", "w") as f:

0056: f.write("Modelo: " + melhor\_nome + "\n")

0057: f.write(f"MSE - Regressão Linear: {mse\_lr:.4f}, R²: {r2\_lr:.4f}\n")

0058: f.write(f"MSE - Random Forest: {mse\_rf:.4f}, R²: {r2\_rf:.4f}\n")

0059: f.write(f"Melhor Modelo Selecionado: {melhor\_nome}\n")

0060:

0061: print("Treinamento concluído. Melhor modelo para evasão:", melhor\_nome)

0062: print("Métricas salvas em: ./modelos/resultados\_modelos")

0063: print("Modelos salvos em: ./modelos/modelos\_salvos")

## treinamento\_modelo\_Feature-based.py

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

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # /scripts/modelagem/treinamento\_modelo\_Feature-based.py

0006: # Este script treina modelos de aprendizado de máquina combinado com Transfer Learning baseado em features para prever a taxa de evasão,

0007: # bem como explorar as relações entre taxa de ingresso, taxa de evasão e taxa de conclusão,

0008: # e realizar previsões para ingressantes e concluintes.

0009:

0010: import pandas as pd

0011: from sklearn.model\_selection import train\_test\_split

0012: from sklearn.ensemble import RandomForestRegressor

0013: from sklearn.linear\_model import LinearRegression

0014: from sklearn.metrics import mean\_squared\_error, r2\_score, confusion\_matrix

0015: import joblib

0016: import os

0017: import matplotlib.pyplot as plt

0018: import seaborn as sns

0019: from sklearn.preprocessing import StandardScaler

0020:

0021: def main():

0022: # Caminho para os dados processados

0023: caminho\_dados = './dados/processado/dados\_ingresso\_evasao\_conclusao.csv'

0024: df = pd.read\_csv(caminho\_dados, sep=';', encoding='utf-8', low\_memory=False)

0025:

0026: # Definir caminho\_modelo (com caminho relativo) antes de utilizá-lo em qualquer parte

0027: caminho\_modelo = './modelos/modelos\_salvos'

0028: os.makedirs(caminho\_modelo, exist\_ok=True)

0029:

0030: # Treinamento para prever ingressantes e concluintes

0031: if all(col in df.columns for col in ['numero\_cursos', 'vagas', 'inscritos', 'docentes', 'ingressantes', 'concluintes']):

0032: print("Treinando modelos para ingressantes e concluintes...")

0033:

0034: # Features e targets para previsões de ingressantes e concluintes

0035: X = df[['numero\_cursos', 'vagas', 'inscritos', 'docentes']]

0036: y\_ingressantes = df['ingressantes']

0037: y\_concluintes = df['concluintes']

0038:

0039: # Divisão dos dados para ingressantes

0040: 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)

0041:

0042: # Modelo Random Forest para ingressantes

0043: modelo\_ingressantes = RandomForestRegressor(random\_state=42)

0044: modelo\_ingressantes.fit(X\_train\_ing, y\_train\_ing)

0045: score\_ingressantes = modelo\_ingressantes.score(X\_test\_ing, y\_test\_ing)

0046: print(f"Score para ingressantes: {score\_ingressantes:.4f}")

0047:

0048: # Divisão dos dados para concluintes

0049: 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)

0050:

0051: # Modelo Random Forest para concluintes

0052: modelo\_concluintes = RandomForestRegressor(random\_state=42)

0053: modelo\_concluintes.fit(X\_train\_conc, y\_train\_conc)

0054: score\_concluintes = modelo\_concluintes.score(X\_test\_conc, y\_test\_conc)

0055: print(f"Score para concluintes: {score\_concluintes:.4f}")

0056:

0057: # Salvar modelos de ingressantes e concluintes

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

0059: joblib.dump(modelo\_concluintes, os.path.join(caminho\_modelo, 'modelo\_concluintes.pkl'))

0060:

0061: # Treinamento para taxas de evasão

0062: print("Treinando modelos para taxa de evasão...")

0063:

0064: # Treinamento adicional com Transfer Learning baseado em features

0065: print("Treinando modelo com Transfer Learning baseado em features...")

0066:

0067: caminho\_dados\_transfer = './dados/processado/dados\_transfer\_learning\_clean.csv'

0068: if os.path.exists(caminho\_dados\_transfer):

0069: df\_transfer = pd.read\_csv(caminho\_dados\_transfer)

0070: if all(col in df\_transfer.columns for col in ['media\_geral', 'taxa\_ingresso', 'taxa\_conclusao', 'taxa\_evasao']):

0071: X\_transfer = df\_transfer[['media\_geral', 'taxa\_ingresso', 'taxa\_conclusao']]

0072: y\_transfer = df\_transfer['taxa\_evasao']

0073:

0074: scaler\_transfer = StandardScaler()

0075: X\_transfer\_scaled = scaler\_transfer.fit\_transform(X\_transfer)

0076:

0077: 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)

0078: modelo\_transfer\_rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

0079: modelo\_transfer\_rf.fit(X\_train\_tr, y\_train\_tr)

0080:

0081: caminho\_modelo\_transfer = os.path.join(caminho\_modelo, 'modelo\_transfer\_rf.pkl')

0082: joblib.dump(modelo\_transfer\_rf, caminho\_modelo\_transfer)

0083: print(f"Modelo de Transfer Learning salvo em: {caminho\_modelo\_transfer}")

0084:

0085: # Filtrar as variáveis relevantes

0086: colunas\_relevantes = ['taxa\_ingresso', 'taxa\_evasao', 'taxa\_conclusao']

0087: df = df[colunas\_relevantes].dropna()

0088:

0089: # Dividir as features e o target

0090: X = df[['taxa\_ingresso', 'taxa\_conclusao']]

0091: y = df['taxa\_evasao']

0092:

0093: # Dividir os dados em treino e teste

0094: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

0095:

0096: # Treinar dois modelos: Regressão Linear e Random Forest

0097: # Modelo 1: Regressão Linear

0098: modelo\_linear = LinearRegression()

0099: modelo\_linear.fit(X\_train, y\_train)

0100: y\_pred\_linear = modelo\_linear.predict(X\_test)

0101: mse\_linear = mean\_squared\_error(y\_test, y\_pred\_linear)

0102: r2\_linear = r2\_score(y\_test, y\_pred\_linear)

0103:

0104: # Modelo 2: Random Forest

0105: modelo\_rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

0106: modelo\_rf.fit(X\_train, y\_train)

0107: y\_pred\_rf = modelo\_rf.predict(X\_test)

0108: mse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf)

0109: r2\_rf = r2\_score(y\_test, y\_pred\_rf)

0110:

0111: # Escolher o melhor modelo baseado no R²

0112: melhor\_modelo = modelo\_rf if r2\_rf > r2\_linear else modelo\_linear

0113: nome\_melhor\_modelo = 'Random Forest' if r2\_rf > r2\_linear else 'Regressão Linear'

0114:

0115: # Salvar o modelo escolhido

0116: joblib.dump(melhor\_modelo, os.path.join(caminho\_modelo, 'modelo\_melhor\_evasao.pkl'))

0117:

0118: # Salvar as métricas dos modelos

0119: caminho\_metricas = './modelos/resultados\_modelos'

0120: os.makedirs(caminho\_metricas, exist\_ok=True)

0121: with open(os.path.join(caminho\_metricas, 'metricas\_modelos.txt'), 'w') as f:

0122: f.write(f"Modelo: {nome\_melhor\_modelo}\n")

0123: f.write(f"MSE - Regressão Linear: {mse\_linear:.4f}, R²: {r2\_linear:.4f}\n")

0124: f.write(f"MSE - Random Forest: {mse\_rf:.4f}, R²: {r2\_rf:.4f}\n")

0125: f.write(f"Melhor Modelo Selecionado: {nome\_melhor\_modelo}\n")

0126:

0127: # Geração da Matriz de Confusão (binarizando a taxa de evasão com threshold de 0.5)

0128: threshold = 0.5

0129: y\_test\_class = (y\_test >= threshold).astype(int)

0130: y\_pred\_class = (melhor\_modelo.predict(X\_test) >= threshold).astype(int)

0131: cm = confusion\_matrix(y\_test\_class, y\_pred\_class)

0132: plt.figure(figsize=(6, 4))

0133: sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

0134: plt.title(f"Matriz de Confusão - {nome\_melhor\_modelo}")

0135: plt.xlabel("Predito")

0136: plt.ylabel("Verdadeiro")

0137: caminho\_cm = os.path.join(caminho\_metricas, 'matriz\_confusao.png')

0138: plt.savefig(caminho\_cm)

0139: plt.close()

0140:

0141: print(f"Treinamento concluído. Melhor modelo para evasão: {nome\_melhor\_modelo}")

0142: print(f"Métricas salvas em: {caminho\_metricas}")

0143: print(f"Modelos salvos em: {caminho\_modelo}")

0144:

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

0146: main()

## treinamento\_modelo\_Fine-tuning.py

0001: # Path: scripts/modelagem/treinamento\_modelo\_Fine-tuning.py

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # /scripts/modelagem/treinamento\_modelo.py

0006: # Script unificado para treinamento de modelos tradicionais e fine-tuning com rede neural

0007:

0008: import os

0009: import pandas as pd

0010: import numpy as np

0011: import joblib

0012: import matplotlib.pyplot as plt

0013: import seaborn as sns

0014: from sklearn.model\_selection import train\_test\_split

0015: from sklearn.ensemble import RandomForestRegressor

0016: from sklearn.linear\_model import LinearRegression

0017: from sklearn.metrics import mean\_squared\_error, r2\_score, confusion\_matrix

0018: from tensorflow.keras.models import load\_model

0019: from tensorflow.keras.layers import Dense

0020: from tensorflow.keras import Model

0021:

0022:

0023: def treinar\_modelos\_basicos(df, caminho\_modelo, caminho\_metricas):

0024: # Treinamento para prever ingressantes e concluintes

0025: if all(col in df.columns for col in ['numero\_cursos', 'vagas', 'inscritos', 'docentes', 'ingressantes', 'concluintes']):

0026: print("Treinando modelos para ingressantes e concluintes...")

0027:

0028: X = df[['numero\_cursos', 'vagas', 'inscritos', 'docentes']]

0029: y\_ing = df['ingressantes']

0030: y\_conc = df['concluintes']

0031:

0032: 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)

0033: 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)

0034:

0035: modelo\_i = RandomForestRegressor(random\_state=42)

0036: modelo\_c = RandomForestRegressor(random\_state=42)

0037: modelo\_i.fit(X\_train\_i, y\_train\_i)

0038: modelo\_c.fit(X\_train\_c, y\_train\_c)

0039:

0040: joblib.dump(modelo\_i, os.path.join(caminho\_modelo, 'modelo\_ingressantes.pkl'))

0041: joblib.dump(modelo\_c, os.path.join(caminho\_modelo, 'modelo\_concluintes.pkl'))

0042:

0043: print(f"Score ingressantes: {modelo\_i.score(X\_test\_i, y\_test\_i):.4f}")

0044: print(f"Score concluintes: {modelo\_c.score(X\_test\_c, y\_test\_c):.4f}")

0045:

0046: # Treinamento para evasão com modelos tradicionais

0047: print("Treinando modelos para taxa de evasão...")

0048: df = df[['taxa\_ingresso', 'taxa\_conclusao', 'taxa\_evasao']].dropna()

0049: X = df[['taxa\_ingresso', 'taxa\_conclusao']]

0050: y = df['taxa\_evasao']

0051: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

0052:

0053: modelo\_lr = LinearRegression()

0054: modelo\_rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

0055: modelo\_lr.fit(X\_train, y\_train)

0056: modelo\_rf.fit(X\_train, y\_train)

0057:

0058: r2\_lr = r2\_score(y\_test, modelo\_lr.predict(X\_test))

0059: r2\_rf = r2\_score(y\_test, modelo\_rf.predict(X\_test))

0060: melhor\_modelo = modelo\_rf if r2\_rf > r2\_lr else modelo\_lr

0061: nome\_melhor = 'Random Forest' if r2\_rf > r2\_lr else 'Regressão Linear'

0062: joblib.dump(melhor\_modelo, os.path.join(caminho\_modelo, 'modelo\_melhor\_evasao.pkl'))

0063:

0064: with open(os.path.join(caminho\_metricas, 'metricas\_modelos.txt'), 'w') as f:

0065: f.write(f"Modelo: {nome\_melhor}\n")

0066: f.write(f"R² Linear: {r2\_lr:.4f}\n")

0067: f.write(f"R² RF: {r2\_rf:.4f}\n")

0068: f.write(f"Melhor modelo: {nome\_melhor}\n")

0069:

0070: y\_pred\_class = (melhor\_modelo.predict(X\_test) >= 0.5).astype(int)

0071: y\_test\_class = (y\_test >= 0.5).astype(int)

0072: cm = confusion\_matrix(y\_test\_class, y\_pred\_class)

0073: plt.figure(figsize=(6, 4))

0074: sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

0075: plt.title(f"Matriz de Confusão - {nome\_melhor}")

0076: plt.xlabel("Predito")

0077: plt.ylabel("Verdadeiro")

0078: plt.savefig(os.path.join(caminho\_metricas, 'matriz\_confusao.png'))

0079: plt.close()

0080:

0081:

0082: def fine\_tune\_modelo\_transfer(df, caminho\_modelo):

0083: print("Iniciando Fine-Tuning com modelo neural pré-treinado...")

0084: esperadas = ['taxa\_ingresso', 'taxa\_conclusao']

0085: faltando = [col for col in ['taxa\_ingresso', 'taxa\_conclusao'] if col not in df.columns]

0086: if faltando:

0087: raise ValueError(f"Colunas ausentes para fine-tuning: {faltando}")

0088: X = df[['taxa\_ingresso', 'taxa\_conclusao']].values

0089: y = df['taxa\_evasao'].values

0090:

0091: from tensorflow.keras import Input

0092:

0093: base\_model = load\_model('modelos/base\_modelo\_neural.h5')

0094:

0095: entrada = Input(shape=(X.shape[1],), name='entrada\_transfer')

0096: x = base\_model(entrada, training=False)

0097: saida = Dense(1, activation='linear', name='saida\_finetune')(x)

0098: model\_finetuned = Model(inputs=entrada, outputs=saida)

0099:

0100: model\_finetuned.compile(optimizer='adam', loss='mse', metrics=['mae'])

0101: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

0102: model\_finetuned.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.2)

0103:

0104: model\_finetuned.save(os.path.join(caminho\_modelo, 'modelo\_finetuned\_tcc.h5'))

0105: print("Fine-Tuning concluído e modelo salvo.")

0106:

0107:

0108: def main():

0109: caminho\_modelo = './modelos/modelos\_salvos'

0110: caminho\_metricas = './modelos/resultados\_modelos'

0111: os.makedirs(caminho\_modelo, exist\_ok=True)

0112: os.makedirs(caminho\_metricas, exist\_ok=True)

0113:

0114: df\_principal = pd.read\_csv('./dados/processado/dados\_ingresso\_evasao\_conclusao.csv', sep=';', encoding='utf-8')

0115: treinar\_modelos\_basicos(df\_principal, caminho\_modelo, caminho\_metricas)

0116:

0117: # Validação e limpeza prévia do CSV de transferência

0118: caminho\_transfer = './dados/processado/dados\_transfer\_learning.csv'

0119: try:

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

0121: except pd.errors.ParserError:

0122: print("Erro ao ler CSV com separador ';'. Tentando com ','...")

0123: try:

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

0125: except pd.errors.ParserError:

0126: print("Erro persistente na leitura do CSV. Tentando leitura manual e correção...")

0127:

0128: # Correção linha a linha

0129: with open(caminho\_transfer, 'r', encoding='utf-8') as f:

0130: linhas\_validas = []

0131: for linha in f:

0132: if linha.count(',') == 3 or linha.count(';') == 3: # Espera-se 4 colunas

0133: linhas\_validas.append(linha)

0134:

0135: # Salva um novo CSV limpo

0136: caminho\_corrigido = './dados/processado/dados\_transfer\_learning\_corrigido.csv'

0137: with open(caminho\_corrigido, 'w', encoding='utf-8') as f\_out:

0138: f\_out.writelines(linhas\_validas)

0139: print(f"Arquivo corrigido salvo em: {caminho\_corrigido}")

0140: df\_transfer = pd.read\_csv(caminho\_corrigido, sep=',', encoding='utf-8', on\_bad\_lines='skip')

0141:

0142: # Já carregado acima após correção automatizada

0143:

0144: print("Colunas disponíveis em df\_transfer:", df\_transfer.columns.tolist())

0145:

0146: fine\_tune\_modelo\_transfer(df\_transfer, caminho\_modelo)

0147:

0148:

0149: if \_\_name\_\_ == '\_\_main\_\_':

0150: main()

## analises.py

0001: # Path: scripts/analises/analises.py

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # Script: analises.py

0006: # Objetivo: Analisar dados de cursos e calcular/visualizar as taxas de conclusão e evasão por curso e ano

0007:

0008: import os

0009: import pandas as pd

0010: import numpy as np

0011: import plotly.express as px

0012: import plotly.graph\_objects as go

0013: import re

0014: from glob import glob

0015:

0016: # Define o mapeamento de cores: verde para "Taxa de Conclusão" e vermelho para "Taxa de Evasão"

0017: color\_map = {"Taxa de Conclusão": "green", "Taxa de Evasão": "red"}

0018:

0019: # =============================================================================

0020: # Leitura dos dados de cursos para os anos de 2009 a 2023

0021: # =============================================================================

0022: anos = range(2009, 2024)

0023: lista\_cursos = []

0024:

0025: # Lista de colunas que devem ser numéricas

0026: numeric\_cols = ["numero\_cursos", "vagas\_totais", "ingressantes", "concluintes", "inscritos\_totais"]

0027:

0028: for ano in anos:

0029: caminho = f"dados/processado/dados\_cursos\_tratado\_{ano}.csv"

0030: if os.path.exists(caminho):

0031: df = pd.read\_csv(caminho, sep=";", dtype=str) # Ler tudo inicialmente como string

0032: for col in numeric\_cols:

0033: if col in df.columns:

0034: df[col] = pd.to\_numeric(df[col], errors='coerce')

0035: df["ano"] = ano

0036: lista\_cursos.append(df)

0037: else:

0038: print(f"Arquivo não encontrado: {caminho}")

0039:

0040: dados\_cursos\_tratado = pd.concat(lista\_cursos, ignore\_index=True)

0041:

0042: # =============================================================================

0043: # Leitura dos dados de universidades (IES) – utilizando a base do ano 2023

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

0045: ies\_path = "dados/processado/dados\_ies\_tratado\_2023.csv"

0046: dados\_ies\_tratado = pd.read\_csv(ies\_path, sep=";", dtype=str)

0047:

0048: # =============================================================================

0049: # Unindo os dados de cursos e universidades

0050: # =============================================================================

0051: all\_data = pd.merge(dados\_cursos\_tratado, dados\_ies\_tratado, on="id\_ies", how="left")

0052: all\_data = all\_data[(all\_data["numero\_cursos"].astype(float) > 0) & (all\_data["nome\_ies"].notnull())]

0053: all\_data["nome\_curso"] = all\_data["nome\_curso"].str.upper()

0054: all\_data["nome\_ies"] = all\_data["nome\_ies"].str.upper()

0055:

0056: for col in ["concluintes", "ingressantes", "inscritos\_totais", "vagas\_totais"]:

0057: if col in all\_data.columns:

0058: all\_data[col] = all\_data[col].astype("Int64")

0059:

0060: # Removemos a variável 'tipo\_rede' pois só aparece explicitamente a partir de 2023

0061: all\_data["cat\_adm"] = all\_data["cat\_adm"].replace({"1": "Federal", "2": "Estadual"}).fillna("Outro")

0062: all\_data["modalidade\_ensino"] = all\_data["modalidade\_ensino"].replace({"1": "Presencial", "2": "EAD"}).fillna("Outro")

0063:

0064: # =============================================================================

0065: # Agrupamento e pivot dos dados para calcular taxas de conclusão e evasão

0066: # =============================================================================

0067: ingress = (all\_data[["nome\_curso", "modalidade\_ensino", "ingressantes", "ano", "concluintes"]]

0068: .groupby(["nome\_curso", "modalidade\_ensino", "ano"], as\_index=False)

0069: .agg({"ingressantes": "sum", "concluintes": "sum"}))

0070:

0071: # Pivotar os dados para 'ingressantes' e 'concluintes'

0072: pivot\_ing = ingress.pivot\_table(index=["nome\_curso", "modalidade\_ensino"], columns="ano", values="ingressantes")

0073: pivot\_con = ingress.pivot\_table(index=["nome\_curso", "modalidade\_ensino"], columns="ano", values="concluintes")

0074:

0075: # Renomeia as colunas com o padrão "ingressantes\_{ano}" e "concluintes\_{ano}"

0076: pivot\_ing.columns = [f"ingressantes\_{ano}" for ano in pivot\_ing.columns]

0077: pivot\_con.columns = [f"concluintes\_{ano}" for ano in pivot\_con.columns]

0078:

0079: ingress\_wide = pivot\_ing.join(pivot\_con).reset\_index()

0080:

0081: # Filtra somente os cursos de interesse

0082: cursos\_interesse = ["ENGENHARIA CIVIL", "MEDICINA", "DIREITO", "ADMINISTRAÇÃO"]

0083: ingress\_wide = ingress\_wide[ingress\_wide["nome\_curso"].isin(cursos\_interesse)]

0084:

0085: # =============================================================================

0086: # Cálculo das taxas de conclusão e evasão por curso

0087: # =============================================================================

0088: # Fórmulas:

0089: # Taxa de Conclusão = concluintes\_{ano} / ingressantes\_{ano - D}

0090: # Taxa de Evasão = 1 - Taxa de Conclusão

0091: duracoes = {

0092: "ENGENHARIA CIVIL": 5,

0093: "MEDICINA": 6,

0094: "DIREITO": 5,

0095: "ADMINISTRAÇÃO": 4

0096: }

0097:

0098: def calcular\_metricas(df, curso, duracao):

0099: df\_course = df[df["nome\_curso"] == curso].copy()

0100: for ano in range(2009 + duracao, 2024):

0101: col\_ing = f"ingressantes\_{ano - duracao}"

0102: col\_con = f"concluintes\_{ano}"

0103: col\_conclusao = f"taxa\_conclusao\_{ano}"

0104: col\_evasao = f"taxa\_evasao\_{ano}"

0105: if col\_ing in df\_course.columns and col\_con in df\_course.columns:

0106: # Evitar divisão por zero e valores não definidos:

0107: df\_course[col\_conclusao] = df\_course[col\_con] / df\_course[col\_ing]

0108: df\_course[col\_evasao] = 1 - df\_course[col\_conclusao]

0109: else:

0110: df\_course[col\_conclusao] = np.nan

0111: df\_course[col\_evasao] = np.nan

0112: return df\_course

0113:

0114: df\_eng = calcular\_metricas(ingress\_wide, "ENGENHARIA CIVIL", duracoes["ENGENHARIA CIVIL"])

0115: df\_dir = calcular\_metricas(ingress\_wide, "DIREITO", duracoes["DIREITO"])

0116: df\_med = calcular\_metricas(ingress\_wide, "MEDICINA", duracoes["MEDICINA"])

0117: df\_adm = calcular\_metricas(ingress\_wide, "ADMINISTRAÇÃO", duracoes["ADMINISTRAÇÃO"])

0118:

0119: # Função para converter data frame wide em long para um dado prefixo e rótulo

0120: def pivot\_long\_metric(df, prefix, label):

0121: metric\_cols = [col for col in df.columns if col.startswith(prefix)]

0122: df\_long = df.melt(id\_vars=["nome\_curso", "modalidade\_ensino"],

0123: value\_vars=metric\_cols,

0124: var\_name="ano", value\_name="taxa")

0125: df\_long["ano"] = df\_long["ano"].str.extract(r"(\d{4})")

0126: df\_long["tipo"] = label

0127: return df\_long

0128:

0129: # Criando data frames long para cada curso e para cada métrica

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

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

0132: eng\_civil\_long = pd.concat([eng\_long\_conclusao, eng\_long\_evasao])

0133:

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

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

0136: direito\_long = pd.concat([dir\_long\_conclusao, dir\_long\_evasao])

0137:

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

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

0140: medicina\_long = pd.concat([med\_long\_conclusao, med\_long\_evasao])

0141:

0142: adm\_long\_conclusao = pivot\_long\_metric(df\_adm, "taxa\_conclusao\_", "Taxa de Conclusão")

0143: adm\_long\_evasao = pivot\_long\_metric(df\_adm, "taxa\_evasao\_", "Taxa de Evasão")

0144: administracao\_long = pd.concat([adm\_long\_conclusao, adm\_long\_evasao])

0145:

0146: # =============================================================================

0147: # Plotagem com Plotly – duas linhas: uma para "Taxa de Conclusão" (verde) e outra para "Taxa de Evasão" (vermelha)

0148: # =============================================================================

0149: def plot\_taxa(df\_long, curso):

0150: fig = px.line(

0151: df\_long,

0152: x="ano",

0153: y="taxa",

0154: color="tipo",

0155: markers=True,

0156: title=f"Taxa de Evasão e Conclusão ao longo dos anos - Curso de {curso}",

0157: labels={"ano": "Ano de Ingresso", "taxa": "Taxa"},

0158: color\_discrete\_map=color\_map

0159: )

0160: fig.update\_yaxes(tickformat=".0%")

0161: fig.update\_layout(legend=dict(orientation="v", title=dict(text="Métrica")))

0162: fig.show()

0163:

0164: plot\_taxa(eng\_civil\_long, "Engenharia Civil")

0165: plot\_taxa(direito\_long, "Direito")

0166: plot\_taxa(medicina\_long, "Medicina")

0167: plot\_taxa(administracao\_long, "Administração")

0168:

0169: # =============================================================================

0170: # Exportação dos dados processados

0171: # =============================================================================

0172: ingress\_wide.to\_csv("dados/processado/final\_ingressantes.csv", index=False, sep=";")

0173: df\_eng.to\_csv("dados/processado/final\_eng\_civil.csv", index=False, sep=";")

0174: df\_dir.to\_csv("dados/processado/final\_direito.csv", index=False, sep=";")

0175: df\_med.to\_csv("dados/processado/final\_medicina.csv", index=False, sep=";")

0176: df\_adm.to\_csv("dados/processado/final\_administracao.csv", index=False, sep=";")

## comparar\_modelos.py

0001: # Path: scripts/analises/comparar\_modelos.py

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # Script: comparar\_modelos.py

0006: # Objetivo: Comparar dois modelos preditivos (.pkl e .h5) usando o dataset de entrada

0007: # 'entrada\_modelos.csv', que deve conter as colunas ['taxa\_ingresso', 'vagas\_totais', 'target'].

0008: # O script gera métricas (MSE e R²) e gráficos comparativos de linha e dispersão,

0009: # que são salvos na pasta 'resultados/'.

0010:

0011: import pandas as pd

0012: import os

0013: os.makedirs("resultados", exist\_ok=True)

0014:

0015: # Carregar o arquivo de entrada

0016: df = pd.read\_csv('dados/processado/entrada\_modelos.csv')

0017:

0018: # Verificar e exibir as colunas disponíveis

0019: colunas\_disponiveis = df.columns.tolist()

0020: print(f"📋 Colunas disponíveis: {colunas\_disponiveis}")

0021:

0022: # Verificar se todas as colunas necessárias estão presentes

0023: colunas\_necessarias = ['taxa\_ingresso', 'vagas\_totais', 'target']

0024: if all(col in colunas\_disponiveis for col in colunas\_necessarias):

0025: df\_modelo = df[colunas\_necessarias].copy()

0026: print("✅ Dados carregados e prontos para comparação de modelos.")

0027:

0028: import matplotlib.pyplot as plt

0029: import joblib

0030: from tensorflow.keras.models import load\_model

0031: from sklearn.metrics import mean\_squared\_error, r2\_score

0032:

0033: # Separar variáveis

0034: X = df\_modelo[['taxa\_ingresso', 'vagas\_totais']]

0035: y\_true = df\_modelo['target']

0036:

0037: # Carregar modelos

0038: modelo\_pkl = joblib.load('modelos/modelos\_salvos/modelo\_melhor\_evasao.pkl')

0039: modelo\_h5 = load\_model('modelos/modelos\_salvos/modelo\_finetuned\_tcc.h5', compile=False)

0040:

0041: # Fazer previsões

0042: y\_pred\_pkl = modelo\_pkl.predict(X)

0043: y\_pred\_h5 = modelo\_h5.predict(X).flatten()

0044:

0045: # Avaliar modelos

0046: mse\_pkl = mean\_squared\_error(y\_true, y\_pred\_pkl)

0047: r2\_pkl = r2\_score(y\_true, y\_pred\_pkl)

0048:

0049: mse\_h5 = mean\_squared\_error(y\_true, y\_pred\_h5)

0050: r2\_h5 = r2\_score(y\_true, y\_pred\_h5)

0051:

0052: print("\nModelo .pkl")

0053: print(f"MSE: {mse\_pkl:.4f}")

0054: print(f"R²: {r2\_pkl:.4f}")

0055:

0056: print("\nModelo .h5")

0057: print(f"MSE: {mse\_h5:.4f}")

0058: print(f"R²: {r2\_h5:.4f}")

0059:

0060: # Gráfico comparativo

0061: plt.figure(figsize=(10, 5))

0062: plt.plot(y\_true.values, label='Real', marker='o')

0063: plt.plot(y\_pred\_pkl, label='Predição .pkl', marker='x')

0064: plt.plot(y\_pred\_h5, label='Predição .h5', marker='s')

0065: plt.title('Comparação entre valores reais e predições dos modelos')

0066: plt.xlabel('Amostra')

0067: plt.ylabel('Taxa de evasão')

0068: plt.legend()

0069: plt.grid(True)

0070: plt.tight\_layout()

0071: # A pasta 'resultados/' deve existir manualmente para evitar erros ao salvar os arquivos.

0072: plt.savefig('resultados/comparacao\_modelos\_linha.png')

0073: plt.savefig("resultados/comparacao\_modelos\_linha.png")

0074: plt.close()

0075:

0076: # Gráfico de dispersão

0077: plt.figure(figsize=(6, 6))

0078: plt.scatter(y\_true, y\_pred\_pkl, label='Modelo .pkl', alpha=0.7)

0079: plt.scatter(y\_true, y\_pred\_h5, label='Modelo .h5', alpha=0.7)

0080: plt.plot([y\_true.min(), y\_true.max()], [y\_true.min(), y\_true.max()], 'k--', lw=2)

0081: plt.title('Dispersão: Real vs Predito')

0082: plt.xlabel('Real')

0083: plt.ylabel('Predito')

0084: plt.legend()

0085: plt.grid(True)

0086: plt.tight\_layout()

0087: # A pasta 'resultados/' deve existir manualmente para evitar erros ao salvar os arquivos.

0088: plt.savefig('resultados/comparacao\_modelos\_dispersao.png')

0089: plt.savefig("resultados/comparacao\_modelos\_dispersao.png")

0090: plt.close()

0091:

0092: else:

0093: faltando = list(set(colunas\_necessarias) - set(colunas\_disponiveis))

0094: raise ValueError(f"❌ Colunas necessárias não encontradas: {faltando}")

## validar\_modelos\_temporais.py

0001: # Path: scripts/analises/validar\_modelos\_temporais.py

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # Script para validação temporal simples de modelo preditivo.

0006: # Realiza análise de correlação, separa treino/teste com shuffle,

0007: # treina modelo LinearRegression e exibe avaliação via MSE e R².

0008: # Também plota gráfico de linha com comparação entre valores reais e preditos.

0009:

0010: import pandas as pd

0011: from sklearn.linear\_model import LinearRegression

0012: from sklearn.metrics import mean\_squared\_error, r2\_score

0013:

0014: # Carrega os dados

0015: df = pd.read\_csv('dados/processado/entrada\_modelos.csv')

0016:

0017: print("\n📌 Correlação das features com o target:")

0018: print(df.corr(numeric\_only=True))

0019:

0020: # Adiciona coluna 'ano' sequencialmente, assumindo que cada linha representa um ano de 2009 a 2023

0021: if 'ano' not in df.columns:

0022: anos = list(range(2009, 2024))

0023: df['ano'] = [anos[i % len(anos)] for i in range(len(df))]

0024:

0025: from sklearn.model\_selection import train\_test\_split

0026: # Embaralha e divide o conjunto entre treino e teste

0027: X = df[['taxa\_ingresso', 'vagas\_totais']]

0028: y = df['target']

0029: X\_treino, X\_teste, y\_treino, y\_teste = train\_test\_split(X, y, test\_size=0.3, random\_state=42, shuffle=True)

0030:

0031: # Treina modelo

0032: modelo = LinearRegression()

0033: modelo.fit(X\_treino, y\_treino)

0034:

0035: # Avalia

0036: y\_pred = modelo.predict(X\_teste)

0037: print("\n📊 Avaliação no conjunto de TESTE:")

0038: print("MSE:", round(mean\_squared\_error(y\_teste, y\_pred), 4))

0039: print("R²:", round(r2\_score(y\_teste, y\_pred), 4))

0040:

0041: import matplotlib.pyplot as plt

0042:

0043: # Gráfico de linha: valores reais vs preditos

0044: plt.figure(figsize=(10, 5))

0045: plt.plot(range(len(y\_teste)), y\_teste.values, label='Real', marker='o')

0046: plt.plot(range(len(y\_pred)), y\_pred, label='Previsto', marker='x')

0047: plt.title('Comparação entre valores reais e previstos (teste)')

0048: plt.xlabel('Amostras')

0049: plt.ylabel('Taxa de Evasão (normalizada)')

0050: plt.legend()

0051: plt.grid(True)

0052: plt.tight\_layout()

0053: plt.savefig('resultados/validacao\_temporal\_comparacao.png')

0054: plt.show()

## prever\_com\_modelos.py

0001: # Path: scripts/analises/prever\_com\_modelos.py

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: import joblib

0006: import numpy as np

0007: import pandas as pd

0008: from tensorflow.keras.models import load\_model

0009: from sklearn.preprocessing import StandardScaler

0010:

0011: # Carregar modelo .pkl (Scikit-Learn)

0012: modelo\_pkl = joblib.load('modelos/modelos\_salvos/modelo\_melhor\_evasao.pkl')

0013:

0014: # Entrada de exemplo: taxa\_ingresso, taxa\_conclusao

0015: entrada = np.array([[0.45, 0.52]])

0016:

0017: # Previsão com modelo .pkl

0018: pred\_pkl = modelo\_pkl.predict(entrada)

0019: print(f"Predição modelo .pkl: {pred\_pkl[0]:.4f}")

0020:

0021: # Carregar modelo .h5 (Keras)

0022: modelo\_h5 = load\_model('modelos/modelo\_finetuned\_tcc.h5')

0023:

0024: # Ajustar entrada para 3 colunas esperadas: media\_geral (dummy), taxa\_ingresso, taxa\_conclusao

0025: entrada\_nn = np.array([[0.0, entrada[0][0], entrada[0][1]]])

0026:

0027: # Previsão com modelo .h5

0028: pred\_h5 = modelo\_h5.predict(entrada\_nn, verbose=0)

0029: print(f"Predição modelo .h5: {pred\_h5[0][0]:.4f}")

## resumir\_modelo\_h5.py

0001: # Path: scripts/analises/resumir\_modelo\_h5.py

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: from tensorflow.keras.models import load\_model

0006:

0007: # Carregar modelo

0008: modelo = load\_model('modelos/modelo\_finetuned\_tcc.h5')

0009:

0010: # Exibir arquitetura

0011: modelo.summary()

## gerar\_graficos.py

0001: # Path: scripts/visualizacao/gerar\_graficos.py

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # scripts/visualizacao/gerar\_graficos.py

0006: # Este script gera gráficos para visualizar os resultados da análise.

0007:

0008: import pandas as pd

0009: import matplotlib.pyplot as plt

0010: import seaborn as sns

0011: import unicodedata

0012:

0013: def main():

0014: df = pd.read\_csv('dados/processado/dados\_ingresso\_evasao\_conclusao.csv', sep=';', on\_bad\_lines='skip')

0015:

0016: # Gráfico de distribuição da taxa de evasão

0017: plt.figure(figsize=(10,6))

0018: sns.histplot(df['taxa\_evasao'], bins=20, kde=True, color='blue')

0019: plt.title('Distribuição da Taxa de Evasão')

0020: plt.xlabel('Taxa de Evasão (%)')

0021: plt.ylabel('Frequência')

0022: plt.savefig('relatorios/figuras/distribuicao\_taxa\_evasao.png')

0023: plt.close()

0024:

0025: # Gráfico de distribuição da taxa de conclusão

0026: plt.figure(figsize=(10,6))

0027: sns.histplot(df['taxa\_conclusao'], bins=20, kde=True, color='green')

0028: plt.title('Distribuição da Taxa de Conclusão')

0029: plt.xlabel('Taxa de Conclusão (%)')

0030: plt.ylabel('Frequência')

0031: plt.savefig('relatorios/figuras/grafico\_taxa\_conclusao.png')

0032: plt.close()

0033:

0034: # Gráfico de distribuição da taxa de ingresso

0035: plt.figure(figsize=(10,6))

0036: sns.histplot(df['taxa\_ingresso'], bins=20, kde=True, color='orange')

0037: plt.title('Distribuição da Taxa de Ingresso')

0038: plt.xlabel('Taxa de Ingresso (%)')

0039: plt.ylabel('Frequência')

0040: plt.savefig('relatorios/figuras/grafico\_taxa\_ingresso.png')

0041: plt.close()

0042:

0043: # Gráfico de dispersão entre ingressantes e concluintes

0044: plt.figure(figsize=(10,6))

0045: sns.scatterplot(x='ingressantes', y='concluintes', data=df)

0046: plt.title('Correlação entre Ingressantes e Concluintes')

0047: plt.xlabel('Ingressantes')

0048: plt.ylabel('Concluintes')

0049: plt.savefig('relatorios/figuras/grafico\_ingressantes\_vs\_concluintes.png')

0050: plt.close()

0051:

0052: # Gráfico de barras por modalidade de ensino (taxa de evasão média)

0053: if 'modalidade\_ensino' in df.columns:

0054: plt.figure(figsize=(10,6))

0055: sns.barplot(x='modalidade\_ensino', y='taxa\_evasao', data=df)

0056: plt.title('Taxa de Evasão Média por Modalidade de Ensino')

0057: plt.xlabel('Modalidade de Ensino')

0058: plt.ylabel('Taxa de Evasão (%)')

0059: plt.savefig('relatorios/figuras/grafico\_taxa\_evasao\_modalidade.png')

0060: plt.close()

0061:

0062: # Heatmap de correlação

0063: plt.figure(figsize=(12,10))

0064: sns.heatmap(df[['taxa\_ingresso', 'taxa\_conclusao', 'taxa\_evasao', 'ingressantes', 'concluintes']].corr(), annot=True, cmap='coolwarm')

0065: plt.title('Mapa de Calor das Correlações')

0066: plt.savefig('relatorios/figuras/grafico\_mapa\_calor\_correlacoes.png')

0067: plt.close()

0068:

0069: print('Gráficos gerados e salvos em relatorios/figuras/')

0070:

0071: if \_\_name\_\_ == '\_\_main\_\_':

0072: main()

## comparar\_predicoes\_cursos.py

0001: # Path: scripts/visualizacao/comparar\_predicoes\_cursos.py

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: import pandas as pd

0006: import matplotlib.pyplot as plt

0007: import seaborn as sns

0008: import csv

0009:

0010: caminhos = {

0011: 'Administração': 'dados/processado/final\_administracao.csv',

0012: 'Direito': 'dados/processado/final\_direito.csv',

0013: 'Engenharia Civil': 'dados/processado/final\_eng\_civil.csv',

0014: 'Medicina': 'dados/processado/final\_medicina.csv'

0015: }

0016:

0017: for curso, caminho in caminhos.items():

0018: try:

0019: df = pd.read\_csv(caminho, delimiter=';')

0020: # Se 'ano' e 'evasao\_real' não estiverem no DataFrame, recriá-los a partir das colunas 'taxa\_evasao\_20XX'

0021: if 'ano' not in df.columns or 'evasao\_real' not in df.columns:

0022: colunas\_evasao = [col for col in df.columns if col.startswith('taxa\_evasao\_')]

0023: if colunas\_evasao:

0024: anos = [int(col.split('\_')[-1]) for col in colunas\_evasao]

0025: evasoes = [df[col].values[0] for col in colunas\_evasao]

0026: df = pd.DataFrame({'ano': anos, 'evasao\_real': evasoes})

0027: df['evasao\_predita'] = df['evasao\_real'] # Inicialmente copia, pode ser sobrescrito se existir coluna real

0028: print(f"📂 Verificando cabeçalho do arquivo {caminho}...")

0029: with open(caminho, 'r') as f:

0030: for \_ in range(3):

0031: print(f.readline().strip())

0032: print(f"📄 Colunas encontradas: {df.columns.tolist()}")

0033: except Exception as e:

0034: print(f"❌ Erro ao ler {caminho}: {e}")

0035: continue

0036: if 'ano' not in df.columns:

0037: print(f"⚠️ Coluna 'ano' não encontrada no arquivo {caminho}.")

0038: print(f"\n📄 Colunas no arquivo {caminho}:\n{df.columns.tolist()}\n")

0039: plt.figure(figsize=(10, 5))

0040: sns.lineplot(x='ano', y='evasao\_real', data=df, label='INEP (real)', marker='o')

0041: sns.lineplot(x='ano', y='evasao\_predita', data=df, label='Projeto TCC (predita)', marker='s')

0042: plt.title(f'Taxa de Evasão - {curso}')

0043: plt.xlabel('Ano')

0044: plt.ylabel('Taxa de Evasão')

0045: plt.ylim(0, 1)

0046: plt.legend()

0047: plt.grid(True)

0048: plt.tight\_layout()

0049: plt.savefig(f'documentos/graficos/taxa\_evasao\_{curso.lower().replace(" ", "\_")}.png')

0050: plt.close()

0051:

0052: print("✅ Gráficos gerados para os cursos: Administração, Direito, Engenharia Civil e Medicina.")

## app\_evasao.py

0001: # Path: scripts/dashboard/app\_evasao.py

0002: # Purpose (en): <write English purpose here>

0003: # Propósito (pt-BR): <escreva em Português aqui>

0004:

0005: # app\_evasao.py

0006: # Dashboard interativo com análise de evasão usando Streamlit

0007:

0008: import streamlit as st

0009: import pandas as pd

0010: import matplotlib.pyplot as plt

0011: import seaborn as sns

0012: import os

0013:

0014: BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

0015: csv\_path = os.path.join(BASE\_DIR, '..', '..', 'dados', 'processado', 'dados\_ingresso\_evasao\_conclusao.csv')

0016: df = pd.read\_csv(csv\_path, sep=';')

0017:

0018: st.set\_page\_config(page\_title="Dashboard Evasão IES", layout="wide")

0019:

0020: # Título

0021: st.title("📊 Dashboard - Taxas de Ingresso, Conclusão e Evasão")

0022:

0023: # Filtro por curso

0024: cursos = df['nome\_curso'].unique()

0025: curso\_selecionado = st.selectbox("Selecione um curso:", sorted(cursos))

0026:

0027: # Filtrar

0028: df\_filtrado = df[df['nome\_curso'] == curso\_selecionado]

0029:

0030: # Métricas rápidas

0031: col1, col2, col3 = st.columns(3)

0032: col1.metric("Taxa de Ingresso (média)", f"{df\_filtrado['taxa\_ingresso'].mean():.2f}")

0033: col2.metric("Taxa de Conclusão (média)", f"{df\_filtrado['taxa\_conclusao'].mean():.2f}")

0034: col3.metric("Taxa de Evasão (média)", f"{df\_filtrado['taxa\_evasao'].mean():.2f}")

0035:

0036: # Gráfico de linha

0037: st.subheader("📈 Evolução das Taxas")

0038: fig, ax = plt.subplots(figsize=(10, 5))

0039: sns.lineplot(data=df\_filtrado[['taxa\_ingresso', 'taxa\_conclusao', 'taxa\_evasao']])

0040: st.pyplot(fig)

0041:

0042: # Salvar gráfico como imagem PNG

0043: output\_dir = os.path.join(BASE\_DIR, '..', '..', 'acessibilidade\_web', 'graficos')

0044: os.makedirs(output\_dir, exist\_ok=True)

0045: fig.savefig(os.path.join(output\_dir, 'grafico\_taxas.png'))

## index.html

0001: <!DOCTYPE html>

0002: <html lang="pt-br" data-theme="auto">

0003: <head>

0004: <meta charset="UTF-8" />

0005: <meta name="viewport" content="width=device-width, initial-scale=1.0" />

0006: <meta name="description" content="Dashboard acessível para análise de evasão em IES públicas.">

0007: <title>Dashboard Acessível - Evasão IES</title>

0008: <link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=SF+Pro+Display:wght@400;600&display=swap">

0009: <style>

0010: :root {

0011: --background-light: #ffffff;

0012: --text-light: #1d1d1f;

0013: --background-dark: #000000;

0014: --text-dark: #f5f5f7;

0015: --highlight: #0071e3;

0016: --button-bg: #0071e3;

0017: --button-text: #ffffff;

0018: --button-hover: #005bb5;

0019: --border-radius: 12px;

0020: --font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial, sans-serif;

0021: }

0022: html[data-theme='dark'] {

0023: background-color: var(--background-dark);

0024: color: var(--text-dark);

0025: }

0026: html[data-theme='light'] {

0027: background-color: var(--background-light);

0028: color: var(--text-light);

0029: }

0030: body {

0031: font-family: var(--font-family);

0032: margin: 0;

0033: padding: 1rem;

0034: line-height: 1.6;

0035: background-color: inherit;

0036: color: inherit;

0037: }

0038: header {

0039: background-color: var(--highlight);

0040: color: white;

0041: padding: 1rem;

0042: text-align: center;

0043: border-radius: var(--border-radius);

0044: }

0045: h1 {

0046: margin: 0;

0047: font-size: 2rem;

0048: }

0049: main {

0050: max-width: 960px;

0051: margin: 0 auto;

0052: }

0053: section {

0054: margin-top: 2rem;

0055: }

0056: label, select, button {

0057: font-size: 1rem;

0058: margin-top: 1rem;

0059: display: block;

0060: }

0061: .tooltip {

0062: position: relative;

0063: cursor: help;

0064: }

0065: .tooltip::after {

0066: content: attr(data-tooltip);

0067: position: absolute;

0068: bottom: 125%;

0069: background-color: #333;

0070: color: white;

0071: padding: 5px;

0072: border-radius: 5px;

0073: font-size: 0.875rem;

0074: opacity: 0;

0075: transition: opacity 0.3s;

0076: white-space: nowrap;

0077: pointer-events: none;

0078: }

0079: .tooltip:hover::after {

0080: opacity: 1;

0081: }

0082: .controls {

0083: display: flex;

0084: gap: 1rem;

0085: }

0086: .visually-hidden {

0087: position: absolute;

0088: width: 1px;

0089: height: 1px;

0090: padding: 0;

0091: margin: -1px;

0092: overflow: hidden;

0093: clip: rect(0, 0, 0, 0);

0094: white-space: nowrap;

0095: border: 0;

0096: }

0097: button {

0098: background-color: var(--button-bg);

0099: color: var(--button-text);

0100: border: none;

0101: padding: 0.2rem 3rem;

0102: border-radius: var(--border-radius);

0103: cursor: pointer;

0104: transition: background-color 0.3s;

0105: }

0106:

0107: button:hover {

0108: background-color: var(--button-hover);

0109: }

0110:

0111: select {

0112: padding: 0.4rem;

0113: border-radius: var(--border-radius);

0114: border: 1px solid #ccc;

0115: font-size: 1rem;

0116: }

0117:

0118: img {

0119: max-width: 100%;

0120: height: auto;

0121: border-radius: var(--border-radius);

0122: box-shadow: 0 4px 16px rgba(0, 0, 0, 0.15);

0123: }

0124: </style>

0125: <script>

0126: function toggleTheme() {

0127: const current = document.documentElement.getAttribute('data-theme');

0128: const newTheme = current === 'dark' ? 'light' : 'dark';

0129: document.documentElement.setAttribute('data-theme', newTheme);

0130: }

0131: </script>

0132: </head>

0133: <body>

0134: <header role="banner">

0135: <h1 id="dashboard-title">Dashboard Acessível - Taxas em IES Públicas</h1>

0136: <button onclick="toggleTheme()" aria-label="Alternar tema claro/escuro">🌙 Alternar Tema</button>

0137: </header>

0138: <main role="main">

0139: <section aria-labelledby="filtros">

0140: <h2 id="filtros">Filtros e Controle</h2>

0141: <label for="curso-select">Curso <span class="tooltip" data-tooltip="Nome do curso conforme base INEP">❔</span></label>

0142: <select id="curso-select" aria-describedby="curso-help">

0143: <option value="direito">Direito</option>

0144: <option value="administracao">Administração</option>

0145: <option value="engenharia">Engenharia Civil</option>

0146: <option value="medicina">Medicina</option>

0147: </select>

0148: <p id="curso-help" class="visually-hidden">Selecione o curso para visualizar os dados</p>

0149: <div class="controls">

0150: <button aria-label="Exportar gráficos como imagem">📤 Exportar</button>

0151: <button aria-label="Limpar filtros">🧹 Limpar Filtros</button>

0152: </div>

0153: </section>

0154: <section aria-labelledby="graficos">

0155: <h2 id="graficos">Visualização de Dados</h2>

0156: <img src="graficos/grafico\_taxas.png" alt="Gráfico com taxas de ingresso, conclusão e evasão" role="img" aria-label="Gráfico de linhas com estilo Apple, representando a evolução das taxas educacionais (ingresso, conclusão e evasão) por curso ao longo do tempo."> <p class="tooltip" data-tooltip="Use o leitor de tela para ouvir a descrição dos gráficos">Os gráficos representam evolução anual</p>

0157: </section>

0158: <section aria-labelledby="ajuda">

0159: <h2 id="ajuda">Ajuda e Acessibilidade</h2>

0160: <p>Este painel foi desenvolvido com base em princípios de usabilidade e acessibilidade: tema escuro, contrastes adequados, suporte a leitores de tela, tooltips e navegação clara por teclado.</p>

0161: </section>

0162: </main>

0163: </body>

0164: </html>

## ajuda.html

0001: <!-- Path: acessibilidade\_web/templates/ajuda/ajuda.html

0002: Purpose (en): Help and Accessibility instructions page for Dashboard users.

0003: Propósito (pt-BR): Página de instruções de Ajuda e Acessibilidade para usuários do Dashboard. -->

0004:

0005: <!DOCTYPE html>

0006: <html lang="pt-br" data-theme="auto">

0007: <head>

0008: <meta charset="UTF-8" />

0009: <title>Central de Ajuda & Acessibilidade</title>

0010: <!-- (pt-BR) WCAG: Definir viewport e escalabilidade para permitir zoom até 200% -->

0011: <!-- (en) WCAG: Define viewport for responsive scaling and 200% zoom support -->

0012: <meta name="viewport" content="width=device-width, initial-scale=1.0" />

0013: <meta name="description" content="Dashboard acessível para análise de evasão em IES públicas.">

0014: <link rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}">

0015: <script src="{{ url\_for('static', filename='script.js') }}"></script>

0016: <style>

0017: /\* (pt-BR) WCAG: Contraste mínimo e espaçamento de texto \*/

0018: /\* (en) WCAG: Minimum contrast and text spacing \*/

0019: body {

0020: color: #000;

0021: background-color: #fff;

0022: font-size: 1.125em; /\* Base ~18px para boa legibilidade \*/

0023: line-height: 1.5; /\* Altura de linha de 1,5x \*/

0024: max-width: 60ch; /\* Limita largura para ~80 caracteres \*/

0025: margin: 0 auto;

0026: padding: 1rem;

0027: font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial, sans-serif;

0028: font-size: 17px;

0029: color: var(--text-light);

0030: background-color: var(--background-light);

0031: line-height: 1.6;

0032: }

0033: a {

0034: color: #0645AD; /\* Azul com contraste adequado \*/

0035: }

0036: a:focus, a:hover {

0037: outline: 2px solid #000; /\* Indicador de foco visível \*/

0038: }

0039: :root {

0040: /\* WCAG 1.4.3: Contraste mínimo de cores (Port BR) \*/

0041: /\* WCAG 1.4.3: Minimum color contrast (En) \*/

0042: --background-light: #ffffff;

0043: --text-light: #1a1a1a;

0044: --background-dark: #121212;

0045: --text-dark: #e0e0e0;

0046: --highlight: #005bb5;

0047: --button-bg: #005bb5;

0048: --button-text: #ffffff;

0049: --button-hover: #003f7d;

0050: --border-radius: 12px;

0051: --font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial, sans-serif;

0052: }

0053: /\* (pt-BR) WCAG 1.4.3: Contraste adequado no tema escuro para títulos \*/

0054: /\* (en) WCAG 1.4.3: Ensure proper contrast for headings in dark mode \*/

0055: html[data-theme="dark"] h1,

0056: html[data-theme="dark"] h2,

0057: html[data-theme="dark"] label {

0058: color: var(--text-dark);

0059: }

0060: h1, h2 {

0061: font-weight: 600;

0062: letter-spacing: -0.01em;

0063: margin-bottom: 0.5em;

0064: }

0065: button {

0066: font-family: var(--font-family);

0067: font-size: 17px;

0068: padding: 12px 24px;

0069: border-radius: 980px;

0070: /\* WCAG 2.5.8: Tamanho mínimo do alvo clicável (Port BR) \*/

0071: /\* WCAG 2.5.8: Minimum target size (En) \*/

0072: background-color: var(--button-bg);

0073: color: var(--button-text);

0074: border: none;

0075: cursor: pointer;

0076: transition: background-color 0.3s;

0077: margin-top: 1rem;

0078: }

0079: button:hover {

0080: background-color: var(--button-hover);

0081: }

0082: /\* WCAG 2.4.7: Foco visível para navegação por teclado (Port BR) \*/

0083: /\* WCAG 2.4.7: Visible focus for keyboard navigation (En) \*/

0084: button:focus {

0085: outline: 3px solid var(--highlight);

0086: outline-offset: 2px;

0087: }

0088: header {

0089: background-color: transparent;

0090: color: var(--text-light);

0091: padding: 1rem 0;

0092: text-align: center;

0093: }

0094: /\* Liquid Glass Card style, matching index.html \*/

0095: .liquid-glass-card {

0096: background: rgba(255, 255, 255, 0.70);

0097: box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.18);

0098: backdrop-filter: blur(9px);

0099: -webkit-backdrop-filter: blur(9px);

0100: border-radius: 18px;

0101: border: 1px solid rgba(255, 255, 255, 0.25);

0102: padding: 2rem 2.5rem;

0103: margin-bottom: 2rem;

0104: transition: background 0.3s;

0105: max-width: 960px;

0106: width: 100%;

0107: }

0108: html[data-theme="dark"] .liquid-glass-card {

0109: background: rgba(25, 25, 25, 0.70);

0110: border: 1px solid rgba(64, 64, 64, 0.22);

0111: }

0112: main {

0113: max-width: 960px;

0114: margin: 0 auto;

0115: }

0116: section {

0117: margin-top: 2rem;

0118: }

0119: label, select, button {

0120: font-size: 17px;

0121: margin-top: 1rem;

0122: display: block;

0123: font-family: var(--font-family);

0124: }

0125: select {

0126: padding: 12px 16px;

0127: border-radius: 12px;

0128: border: 1px solid #ccc;

0129: font-size: 17px;

0130: font-family: var(--font-family);

0131: margin-top: 0.5rem;

0132: width: 100%;

0133: box-sizing: border-box;

0134: }

0135: input[type="number"], input[type="text"] {

0136: padding: 12px 16px;

0137: border-radius: 12px;

0138: border: 1px solid #ccc;

0139: font-size: 17px;

0140: font-family: var(--font-family);

0141: width: 100%;

0142: box-sizing: border-box;

0143: margin-top: 0.5rem;

0144: }

0145: .tooltip {

0146: position: relative;

0147: cursor: help;

0148: font-size: 17px;

0149: font-family: var(--font-family);

0150: }

0151: .tooltip::after {

0152: content: attr(data-tooltip);

0153: position: absolute;

0154: bottom: 125%;

0155: background-color: #333;

0156: color: white;

0157: padding: 8px 12px;

0158: border-radius: 12px;

0159: font-size: 14px;

0160: opacity: 0;

0161: transition: opacity 0.3s;

0162: white-space: nowrap;

0163: pointer-events: none;

0164: z-index: 10;

0165: }

0166: .tooltip:hover::after {

0167: opacity: 1;

0168: }

0169: .controls {

0170: display: flex;

0171: gap: 1rem;

0172: flex-wrap: wrap;

0173: margin-top: 1rem;

0174: }

0175: .visually-hidden {

0176: position: absolute;

0177: width: 1px;

0178: height: 1px;

0179: padding: 0;

0180: margin: -1px;

0181: overflow: hidden;

0182: clip: rect(0, 0, 0, 0);

0183: white-space: nowrap;

0184: border: 0;

0185: }

0186: img {

0187: max-width: 100%;

0188: height: auto;

0189: border-radius: 12px;

0190: box-shadow: 0 4px 16px rgba(0, 0, 0, 0.15);

0191: margin-top: 1rem;

0192: }

0193: html.inverted-contrast {

0194: filter: invert(100%) hue-rotate(180deg) contrast(1.1) saturate(1.2);

0195: background-color: #000 !important;

0196: color: #ffffff !important;

0197: }

0198: html.inverted-contrast img {

0199: filter: invert(100%) hue-rotate(180deg) contrast(1.1) saturate(1.2);

0200: }

0201: </style>

0202: <script>

0203: function toggleTheme() {

0204: const current = document.documentElement.getAttribute('data-theme');

0205: const newTheme = current === 'dark' ? 'light' : 'dark';

0206: document.documentElement.setAttribute('data-theme', newTheme);

0207: }

0208: </script>

0209: <script>

0210: function descreverGraficoIA() {

0211: const descricao = "Analisando tendências: Em Direito, evasão cresceu de 2015 a 2019. Conclusão estável. Engenharia mostra queda na evasão desde 2018.";

0212: const area = document.getElementById("grafico-descricao");

0213: area.textContent = descricao;

0214: area.setAttribute("aria-live", "assertive");

0215: }

0216: </script>

0217: <script>

0218: document.addEventListener('DOMContentLoaded', () => {

0219: const exportButton = document.querySelector('button[aria-label="Exportar gráficos como imagem"]');

0220: const clearButton = document.querySelector('button[aria-label="Limpar filtros e retornar à visualização padrão"]');

0221: const cursoInput = document.getElementById('curso-input');

0222: const anoInput = document.getElementById('ano-input');

0223: const iesInput = document.getElementById('ies-input');

0224: const graficoContainer = document.getElementById('grafico-container');

0225:

0226: // Function to fetch and update the graph dynamically

0227: async function fetchGrafico() {

0228: const curso = cursoInput.value;

0229: const ano = anoInput.value;

0230: const ies = iesInput.value;

0231:

0232: const response = await fetch('/gerar-grafico', {

0233: method: 'POST',

0234: headers: {

0235: 'Content-Type': 'application/json',

0236: },

0237: body: JSON.stringify({ curso, ano, ies }),

0238: });

0239:

0240: if (response.ok) {

0241: const data = await response.json();

0242: if (data.grafico\_url) {

0243: graficoContainer.innerHTML = `<img src="${data.grafico\_url}" alt="Gráfico gerado dinamicamente" role="img" aria-describedby="grafico-descricao">`;

0244: } else {

0245: graficoContainer.innerHTML = '<p>Gráfico não disponível para os filtros selecionados.</p>';

0246: }

0247: } else {

0248: graficoContainer.innerHTML = '<p>Erro ao carregar gráfico.</p>';

0249: }

0250: }

0251:

0252: // Attach event listeners

0253: exportButton.addEventListener('click', () => {

0254: // Implement export functionality here if needed

0255: alert('Funcionalidade de exportação ainda não implementada.');

0256: });

0257:

0258: clearButton.addEventListener('click', () => {

0259: cursoInput.value = '';

0260: anoInput.value = '';

0261: iesInput.value = '';

0262: graficoContainer.innerHTML = '';

0263: });

0264:

0265: // Optionally, fetch graph when filters change

0266: [cursoInput, anoInput, iesInput].forEach(element => {

0267: element.addEventListener('change', fetchGrafico);

0268: });

0269:

0270: // If you want a button to generate graph, you can add it here

0271: // Or fetch graph automatically on filter changes as above

0272: });

0273: </script>

0274: <script>

0275: function ajustarFonte(acao) {

0276: const html = document.documentElement;

0277: const estiloAtual = window.getComputedStyle(html).fontSize;

0278: let tamanho = parseFloat(estiloAtual);

0279:

0280: // WCAG 1.4.4: Texto redimensionável até 200% (Port BR)

0281: // WCAG 1.4.4: Resize text up to 200% (En)

0282: const tamanhoBase = 17; // base font size in px

0283: const maxTamanho = tamanhoBase \* 2; // 200%

0284: const minTamanho = tamanhoBase \* 0.7; // about 70% minimum for readability

0285:

0286: const incremento = 2;

0287:

0288: if (acao === 'aumentar' && tamanho < maxTamanho) {

0289: tamanho += incremento;

0290: } else if (acao === 'diminuir' && tamanho > minTamanho) {

0291: tamanho -= incremento;

0292: }

0293:

0294: html.style.fontSize = `${tamanho}px`;

0295:

0296: // Ajusta também espaçamento de linha para melhorar a legibilidade

0297: document.body.style.lineHeight = tamanho >= 20 ? '1.8' : '1.6';

0298: }

0299: </script>

0300: </head>

0301: <body role="document">

0302:

0303: <header role="banner">

0304: <div style="display: flex; gap: 0.5rem; justify-content: center;">

0305: <button onclick="

0306: document.documentElement.classList.remove('inverted-contrast');

0307: document.documentElement.setAttribute('data-theme', 'light');

0308: document.body.style.backgroundColor = '#ffffff';

0309: document.body.style.color = '#1a1a1a';

0310: " aria-label="Ativar tema claro" title="Ativar tema claro">Contraste Claro</button>

0311: <button onclick="

0312: document.documentElement.classList.remove('inverted-contrast');

0313: document.documentElement.setAttribute('data-theme', 'dark');

0314: document.body.style.backgroundColor = '#121212';

0315: document.body.style.color = '#e0e0e0';

0316: " aria-label="Ativar tema escuro" title="Ativar tema escuro">Contraste Escuro</button>

0317: <button onclick="

0318: if (!document.documentElement.classList.contains('inverted-contrast')) {

0319: document.documentElement.classList.add('inverted-contrast');

0320: document.documentElement.setAttribute('data-theme', 'light');

0321: document.body.style.backgroundColor = '#ffffff';

0322: document.body.style.color = '#1a1a1a';

0323: } else {

0324: document.documentElement.classList.remove('inverted-contrast');

0325: document.documentElement.setAttribute('data-theme', 'light');

0326: document.body.style.backgroundColor = '#ffffff';

0327: document.body.style.color = '#1a1a1a';

0328: }

0329: " aria-label="Ativar contraste invertido" title="Ativar contraste invertido">Contraste Invertido</button>

0330: <button onclick="ajustarFonte('aumentar')" aria-label="Aumentar tamanho da fonte" title="Aumentar fonte">A+</button>

0331: <button onclick="ajustarFonte('diminuir')" aria-label="Diminuir tamanho da fonte" title="Diminuir fonte">A−</button>

0332: </div>

0333: </header>

0334:

0335: <main>

0336: <div class="liquid-glass-card">

0337: <h1 tabindex="0">Central de Ajuda</h1>

0338: <p tabindex="0">Bem-vindo! Aqui estão as principais orientações de uso do Dashboard.</p>

0339:

0340: <h2 tabindex="0">Contraste e Tema</h2>

0341: <p tabindex="0">Use os botões no topo para alternar entre tema claro, escuro e contraste invertido.</p>

0342:

0343: <h2 tabindex="0">Ajuste de Fonte</h2>

0344: <p tabindex="0">Use os botões A+ e A− para ajustar o tamanho da fonte até 200%.</p>

0345:

0346: <h2 tabindex="0">Navegação por Teclado</h2>

0347: <ul>

0348: <li tabindex="0"><kbd>Tab</kbd> para avançar pelos elementos interativos.</li>

0349: <li tabindex="0"><kbd>Shift</kbd> + <kbd>Tab</kbd> para voltar.</li>

0350: </ul>

0351: </div>

0352:

0353: <div class="liquid-glass-card">

0354: <h2 tabindex="0">TECLAS ATALHOS ACESSIBILIDADE WINDOWS</h2>

0355: <table>

0356: <thead>

0357: <tr>

0358: <th>Atalho</th>

0359: <th>Ação sugerida</th>

0360: </tr>

0361: </thead>

0362: <tbody>

0363: <tr><td>CTRL + ALT + Y</td><td>Acesso rápido à pesquisa por ano</td></tr>

0364: <tr><td>CTRL + ALT + I</td><td>Acesso rápido à pesquisa por instituição</td></tr>

0365: <tr><td>ALT + C</td><td>Ativar Modo Contraste Claro</td></tr>

0366: <tr><td>ALT + E</td><td>Ativar Modo Contraste Escuro</td></tr>

0367: <tr><td>ALT + V</td><td>Ativar Modo Contraste Invertido</td></tr>

0368: <tr><td>CTRL + +</td><td>Aumentar tamanho das palavras (zoom de texto)</td></tr>

0369: <tr><td>CTRL + -</td><td>Diminuir tamanho das palavras</td></tr>

0370: <tr><td>ALT + L</td><td>Ativar/Desativar Botão Libras (ex.: plugin VLibras)</td></tr>

0371: <tr><td>ALT + S</td><td>Ativar/Desativar Síntese de voz/leitor de tela interno (ou direciona para configuração de voz)</td></tr>

0372: <tr><td>ALT + H</td><td>Acesso rápido à página de Ajuda</td></tr>

0373: <tr><td>ALT + A</td><td>Focar no Botão Ajuda Contextual (se diferente da página de Ajuda principal)</td></tr>

0374: <tr><td>ALT + K</td><td>Ativar teclado virtual na tela (para quem tem mobilidade reduzida)</td></tr>

0375: <tr><td>ALT + D</td><td>Alternar para Modo de Alto Contraste para Daltonismo (se for um modo específico)</td></tr>

0376: <tr><td>ALT + P</td><td>Alternar Ponteiro de Mouse Ampliado</td></tr>

0377: <tr><td>CTRL + ALT + U</td><td>Ativar/desativar Legenda Automática (caso tenha vídeos)</td></tr>

0378: <tr><td>ALT + T</td><td>Ir direto para tutorial de acessibilidade (fluxo guiado explicando como usar os recursos)</td></tr>

0379: </tbody>

0380: </table>

0381: </div>

0382:

0383: <div class="liquid-glass-card">

0384: <h2 tabindex="0">TECLAS ATALHOS ACESSIBILIDADE MACOS / LINUX</h2>

0385: <table>

0386: <thead>

0387: <tr>

0388: <th>Atalho</th>

0389: <th>Ação sugerida</th>

0390: </tr>

0391: </thead>

0392: <tbody>

0393: <tr><td>⌘ + ⌥ + Y</td><td>Acesso rápido à pesquisa por ano</td></tr>

0394: <tr><td>⌘ + ⌥ + I</td><td>Acesso rápido à pesquisa por instituição</td></tr>

0395: <tr><td>⌥ + C</td><td>Ativar Modo Contraste Claro</td></tr>

0396: <tr><td>⌥ + E</td><td>Ativar Modo Contraste Escuro</td></tr>

0397: <tr><td>⌥ + V</td><td>Ativar Modo Contraste Invertido</td></tr>

0398: <tr><td>⌘ + +</td><td>Aumentar tamanho das palavras (zoom de texto)</td></tr>

0399: <tr><td>⌘ + -</td><td>Diminuir tamanho das palavras</td></tr>

0400: <tr><td>⌥ + L</td><td>Ativar/Desativar Botão Libras (ex.: plugin VLibras)</td></tr>

0401: <tr><td>⌥ + S</td><td>Ativar/Desativar Síntese de voz/leitor de tela interno (ou direciona para configuração de voz)</td></tr>

0402: <tr><td>⌥ + H</td><td>Acesso rápido à página de Ajuda</td></tr>

0403: <tr><td>⌥ + A</td><td>Focar no Botão Ajuda Contextual (se diferente da página de Ajuda principal)</td></tr>

0404: <tr><td>⌥ + K</td><td>Ativar teclado virtual na tela (para quem tem mobilidade reduzida)</td></tr>

0405: <tr><td>⌥ + D</td><td>Alternar para Modo de Alto Contraste para Daltonismo (se for um modo específico)</td></tr>

0406: <tr><td>⌥ + P</td><td>Alternar Ponteiro de Mouse Ampliado</td></tr>

0407: <tr><td>⌘ + ⌥ + U</td><td>Ativar/desativar Legenda Automática (caso tenha vídeos)</td></tr>

0408: <tr><td>⌥ + T</td><td>Ir direto para tutorial de acessibilidade (fluxo guiado explicando como usar os recursos)</td></tr>

0409: </tbody>

0410: </table>

0411: </div>

0412:

0413: <div class="liquid-glass-card">

0414: <h2 tabindex="0">Descrição de Gráficos</h2>

0415: <p tabindex="0">Clique no botão 🧠 para descrição detalhada do gráfico com IA.</p>

0416:

0417: <h2 tabindex="0">Leitura em Libras</h2>

0418: <p tabindex="0">O botão 🧏 ativa um intérprete de Libras (em desenvolvimento).</p>

0419:

0420: <h2 tabindex="0">Leitura de Voz</h2>

0421: <p tabindex="0">Ative 🔊 para ouvir descrições do gráfico atual.</p>

0422: </div>

0423: </main>

0424:

0425: <!-- Botões de Acessibilidade -->

0426: <div style="position: fixed; top: 20%; right: 0; z-index: 9999; display: flex; flex-direction: column; gap: 0.5rem;">

0427: <button aria-label="Ativar tradução em Libras" title="Ativar Libras" onclick="alert('Função Libras em desenvolvimento')">🧏‍♂️ Libras</button>

0428: <button aria-label="Ativar leitura de voz" title="Ativar Leitura de Voz" onclick="alert('Leitura em voz será ativada')">🔊 Voz</button>

0429: <button aria-label="Mais opções de acessibilidade" title="Abrir menu de acessibilidade" onclick="alert('Menu de Acessibilidade em breve')">♿ Acessibilidade</button>

0430: <button aria-label="Página Inicial" title="Voltar para página inicial" onclick="window.location.href='../index.html'">🏠 Página Inicial</button>

0431: </div>

0432: <!-- WCAG 2.1.2: Sem armadilhas de teclado (Port BR) -->

0433: <!-- WCAG 2.1.2: No keyboard traps (En) -->

0434: </body>

0435: </html>

## script.js

0001: document.addEventListener('DOMContentLoaded', function () {

0002: // Tour guiado para leitores de tela

0003: const tourSteps = [

0004: "Bem-vindo ao sistema de análise de dados educacionais.",

0005: "Use as setas do teclado para navegar entre seções.",

0006: "Pressione Enter para acessar detalhes de cada gráfico."

0007: ];

0008:

0009: let currentStep = 0;

0010:

0011: function speak(text) {

0012: const utterance = new SpeechSynthesisUtterance(text);

0013: utterance.lang = 'pt-BR';

0014: speechSynthesis.speak(utterance);

0015: }

0016:

0017: function startTour() {

0018: if ('speechSynthesis' in window) {

0019: speak(tourSteps[currentStep]);

0020: }

0021: }

0022:

0023: document.addEventListener('keydown', function (e) {

0024: if (e.key === 'ArrowRight') {

0025: currentStep = (currentStep + 1) % tourSteps.length;

0026: speak(tourSteps[currentStep]);

0027: }

0028: });

0029:

0030: // Botões acessíveis

0031: const accessibleButtons = document.querySelectorAll('button');

0032: accessibleButtons.forEach(button => {

0033: if (!button.hasAttribute('aria-label')) {

0034: const label = button.textContent.trim() || 'Botão';

0035: button.setAttribute('aria-label', label);

0036: button.setAttribute('title', label);

0037: }

0038: });

0039:

0040: // Descrição dinâmica de gráficos (com aria-live)

0041: const graficoDescricao = document.createElement('div');

0042: graficoDescricao.setAttribute('id', 'grafico-descricao');

0043: graficoDescricao.setAttribute('aria-live', 'polite');

0044: graficoDescricao.setAttribute('role', 'status');

0045: graficoDescricao.className = 'sr-only';

0046: document.body.appendChild(graficoDescricao);

0047:

0048: function atualizarDescricaoGrafico(texto) {

0049: graficoDescricao.textContent = texto;

0050: }

0051:

0052: // Exemplo: descrição fictícia inicial

0053: atualizarDescricaoGrafico("Gráfico inicial: evasão nos cursos superiores de 2015 a 2023.");

0054:

0055: // Expandir com descrições geradas dinamicamente se necessário

0056:

0057: // Carregar gráfico com base nos filtros

0058: const botaoFiltrar = document.getElementById('filtrar-btn');

0059: if (botaoFiltrar) {

0060: botaoFiltrar.addEventListener('click', function () {

0061: const curso = document.getElementById('curso-select').value;

0062: const ano = document.getElementById('ano-input').value;

0063: const ies = document.getElementById('ies-input').value;

0064:

0065: fetch('/gerar-grafico', {

0066: method: 'POST',

0067: headers: { 'Content-Type': 'application/json' },

0068: body: JSON.stringify({ curso, ano, ies })

0069: })

0070: .then(res => {

0071: if (res.ok) {

0072: document.querySelector("img[alt^='Gráfico']").src = "/static/graficos/grafico\_taxas.png?" + new Date().getTime();

0073: atualizarDescricaoGrafico(`Gráfico atualizado para o curso ${curso}, ano ${ano}, instituição ${ies}.`);

0074: } else {

0075: alert("Nenhum dado encontrado para os filtros informados.");

0076: }

0077: })

0078: .catch(error => {

0079: console.error('Erro ao buscar gráfico:', error);

0080: alert("Erro na comunicação com o servidor.");

0081: });

0082: });

0083: }

## style.css

0001:

0002:

0003:

0004: /\* Alto contraste \*/

0005: body, html {

0006: background-color: #000;

0007: color: #FFF;

0008: font-size: 18px;

0009: line-height: 1.6;

0010: font-family: Arial, sans-serif;

0011: }

0012:

0013: /\* Tema claro alternativo \*/

0014: body.theme-claro {

0015: background-color: #FFF;

0016: color: #000;

0017: }

0018:

0019: /\* Links acessíveis \*/

0020: a {

0021: color: #00FFFF;

0022: text-decoration: underline;

0023: }

0024: a:focus, a:hover {

0025: outline: 3px solid yellow;

0026: background-color: #222;

0027: }

0028:

0029: /\* Foco visível \*/

0030: \*:focus {

0031: outline: 3px dashed #FFD700;

0032: outline-offset: 4px;

0033: }

0034:

0035: /\* Botões grandes e legíveis \*/

0036: button {

0037: background-color: #444;

0038: color: white;

0039: font-size: 18px;

0040: padding: 12px 24px;

0041: border: none;

0042: cursor: pointer;

0043: }

0044: button:focus {

0045: outline: 3px solid #00FFFF;

0046: }

0047:

0048: /\* Áreas com rótulo para leitores de tela \*/

0049: [role="region"] {

0050: border: 2px solid #AAA;

0051: padding: 10px;

0052: margin-bottom: 15px;

0053: }

0054:

0055: /\* Preferência por contraste alto do sistema operacional \*/

0056: @media (prefers-contrast: more) {

0057: body {

0058: background-color: #000 !important;

0059: color: #FFF !important;

0060: }

0061: a {

0062: color: #0FF !important;

0063: }

0064: button {

0065: background-color: #222 !important;

0066: color: #FFF !important;

0067: }

0068: }

0069:

0070: /\* Melhor legibilidade para gráficos \*/

0071: canvas {

0072: outline: 2px dashed #FFF;

0073: }

## tour.js

0001: document.addEventListener('DOMContentLoaded', () => {

0002: alert('Bem-vindo! Use Tab para navegar. Esta versão inicial do tour guiado será expandida.');

0003: });

## graficos.json

0001: {

0002: "grafico\_taxas.png": "Taxa de evasão em Direito subiu de 2015 a 2019. Engenharia Civil apresenta queda desde 2018. Medicina estável.",

0003: "grafico\_novo.png": "Exemplo: novas tendências podem ser descritas aqui."

0004: }

## voice.js

0001: function lerDescricaoGrafico() {

0002: const texto = document.getElementById('grafico-descricao').textContent;

0003: const synth = window.speechSynthesis;

0004: const utter = new SpeechSynthesisUtterance(texto);

0005: utter.lang = 'pt-BR';

0006: synth.speak(utter);

0007: }