Filtrando os Dados

```
import pandas as pd
import unicodedata
import os

COLUNAS_DESEJADAS = [
    "data",
    "temp._ins._(c)",
    "temp._max._(c)",
    "temp._min._(c)",
    "temperatura_do_ar___bulbo_seco,_horaria_(c)",
    "temperatura_do_ponto_de_orvalho_(c)",
    "temperatura_maxima_na_hora_ant._(aut)_(c)",
    "temperatura_minima_na_hora_ant._(aut)_(c)",
    "temperatura_orvalho_max._na_hora_ant._(aut)_(c)",
    "temperatura_orvalho_min._na_hora_ant._(aut)_(c)",
    "consumo_kw"
]

def normalizar_colunas(df: pd.DataFrame) -> pd.DataFrame:
    """Normaliza os nomes das colunas (remove acentos, espaços e caracteres especiais)."""
    df.columns = [
        unicodedata.normalize('NFKD', str(c))
        .encode('ascii', 'ignore')
        .decode('ascii')
        .strip()
        .replace(" ", "_")
        .replace("/", "_")
        .replace("-", "_")
        .lower()
        for c in df.columns
    ]
    return df


def converter_coluna_data(df: pd.DataFrame) -> pd.DataFrame:
    """
    Tenta converter a coluna 'data' em datetime, lidando com vários formatos:
    - YYYYMMDD
    - YYYY/MM/DD
    - DD/MM/YYYY
    """
    if 'data' not in df.columns:
        return df
```

```
    # Remove espaços e converte tudo para string
    df['data'] = df['data'].astype(str).str.strip()

    # Detecta datas no formato numérico contínuo (ex: 20110101)
    mask_numeric = df['data'].str.match(r'^\d{8}$')
    if mask_numeric.any():
        df.loc[mask_numeric, 'data'] = pd.to_datetime(df.loc[mask_numeric, 'data'],
format='%Y%m%d', errors='coerce')

    # Tenta os outros formatos comuns
    df['data'] = pd.to_datetime(df['data'], errors='coerce', dayfirst=True)

    # Remove linhas com datas inválidas
    df = df.dropna(subset=['data'])

    # Padroniza para formato YYYY-MM-DD
    df['data'] = df['data'].dt.strftime('%Y-%m-%d')

    return df


def filtrar_colunas(csv_path: str, output_path: str = None, ano_minimo: int = 2024) ->
pd.DataFrame:
    """
    Carrega CSV, mantém apenas as colunas desejadas, filtra por ano mínimo e salva
resultado.

    @param {str} csv_path - Caminho do arquivo CSV original
    @param {str} output_path - Caminho de saída opcional
    @param {int} ano_minimo - Linhas com ano menor que este serão removidas
    @return {pd.DataFrame} DataFrame filtrado
    """
    df = pd.read_csv(csv_path, encoding="utf-8")
    df = normalizar_colunas(df)

    # Mantém apenas colunas relevantes
    colunas_para_manter = [c for c in COLUNAS_DESEJADAS if c in df.columns]
    df_filtrado = df[colunas_para_manter]

    # Converte e padroniza coluna de data
    df_filtrado = converter_coluna_data(df_filtrado)

    #  🔹 Remove linhas com ano anterior ao mínimo
    if 'data' in df_filtrado.columns:
        df_filtrado = df_filtrado[pd.to_datetime(df_filtrado['data']).dt.year >= ano_minimo]

    # Define o nome de saída
    if output_path is None:
```

```
        base, ext = os.path.splitext(os.path.basename(csv_path))
        output_path = f"./dados filtrados/{base}_filtrado{ext}"

    # Cria diretório de saída se não existir
    os.makedirs(os.path.dirname(output_path), exist_ok=True)

    # Salva CSV filtrado
    df_filtrado.to_csv(output_path, index=False, encoding="utf-8")
    print(f"[OK] CSV filtrado salvo em {output_path} | Colunas: {len(colunas_para_manter)} |
Linhas: {len(df_filtrado)}")

    return df_filtrado


if __name__ == "__main__":
    filtrar_colunas("./dados/anuario estatistico de energia eletrica.csv")
    filtrar_colunas("./dados/INMET_BRASILIA_01-01-2024_A_31-12-2024.csv")
    filtrar_colunas("./dados/INMP 15102025-15102025.csv")
```

Unificando os Dados

```
import pandas as pd
import unicodedata
import os
import numpy as np

def normalizar_colunas(df: pd.DataFrame) -> pd.DataFrame:
    """
    Normaliza os nomes das colunas de um DataFrame.
    Remove acentos, espaços, caracteres especiais e converte para minúsculas.

    @param df: DataFrame do pandas.
    @return: DataFrame com colunas normalizadas.
    """
    # ... (seu código atual, que está eficiente para esta tarefa)
    df.columns = [
        unicodedata.normalize("NFKD", str(c))
        .encode("ascii", "ignore")
        .decode("ascii")
        .strip()
        .replace(" ", "_")
        .replace("/", "_")
        .replace("-", "_")
        .lower()
        for c in df.columns
    ]
    return df

def mesclar_csvs(*csv_paths, output_path="dados_unificados.csv",
preencher_ausentes=False) -> pd.DataFrame:
    """
    Mescla múltiplos arquivos CSV, normaliza colunas, converte dados de data/hora e
    agrega colunas de temperatura e consumo.

    @param csv_paths: Caminhos para os arquivos CSV a serem mesclados.
    @param output_path: Caminho para salvar o arquivo CSV unificado.
    @param preencher_ausentes: Se True, preenche valores ausentes com a média mensal.
    @return: DataFrame unificado.
    """
    if len(csv_paths) < 2:
        raise ValueError("Informe pelo menos dois arquivos CSV para mesclar.")

    dataframes = []

    for path in csv_paths:
        if not os.path.exists(path):
```

```
            print(f"[ERRO] Arquivo não encontrado: {path}")
            continue

        try:
            # Tenta UTF-8, se falhar, tenta latin-1
            df = pd.read_csv(path, encoding="utf-8")
        except UnicodeDecodeError:
            df = pd.read_csv(path, encoding="latin-1")

        # Tratamento de vírgula como separador decimal para colunas numéricas
        # Isso é crucial para dados brasileiros
        # Detecta automaticamente colunas com ',' e força a conversão
        for col in df.columns:
            if df[col].dtype == 'object' and df[col].str.contains(',').any():
                df[col] = df[col].str.replace('.', '').str.replace(',', '.')

        df = normalizar_colunas(df)

        if "data" not in df.columns:
            print(f"[AVISO] '{os.path.basename(path)}' não possui coluna 'data'. Ignorado.")
            continue

        # Garante que a coluna 'data' esteja no formato datetime
        df["data"] = pd.to_datetime(df["data"], errors="coerce")
        df = df.dropna(subset=["data"])

        colunas_temperatura = [c for c in df.columns if "temp" in c]
        colunas_consumo = [c for c in df.columns if "consumo" in c and "kw" in c]

        # Converte para numérico após o tratamento de vírgulas
        for c in colunas_temperatura + colunas_consumo:
            df[c] = pd.to_numeric(df[c], errors="coerce")

        # Agrega as colunas de temperatura e consumo
        df["temperatura_media"] = df[colunas_temperatura].mean(axis=1, skipna=True) if
colunas_temperatura else np.nan
        df["consumo_kw"] = df[colunas_consumo].mean(axis=1, skipna=True) if
colunas_consumo else np.nan

        df = df[["data", "consumo_kw", "temperatura_media"]].dropna(subset=["data"])

        # Agrupa por data e tira a média (para o caso de múltiplas linhas para a mesma data,
como no seu 'anuario')
        df = df.groupby("data", as_index=False).mean(numeric_only=True)
        dataframes.append(df)

    if not dataframes:
        raise ValueError("Nenhum arquivo válido encontrado.")
```

```
    # Concatena e faz um agrupamento final
    df_final = pd.concat(dataframes, ignore_index=True)
    df_final = df_final.groupby("data", as_index=False).mean(numeric_only=True)

    # Lógica de preenchimento (se solicitada)
    if preencher_ausentes:
        df_final["mes"] = df_final["data"].dt.month
        for col in ["consumo_kw", "temperatura_media"]:
            # Preenche ausentes com a média mensal do respectivo mês
            df_final[col] = df_final.groupby("mes")[col].transform(lambda x: x.fillna(x.mean()))
        df_final = df_final.drop(columns=["mes"])

    df_final = df_final.sort_values(by="data").reset_index(drop=True)
    df_final.to_csv(output_path, index=False, encoding="utf-8")

    print(f"[FINALIZADO] Arquivo unificado salvo em: {output_path}")
    print(f"[INFO] Linhas: {len(df_final)} | Colunas: {len(df_final.columns)}")
    return df_final

if __name__ == "__main__":
    # Mantém a execução principal, mas se atente à limpeza dos seus CSVs de entrada
    df_final = mesclar_csvs(
        "./dados filtrados/anuario estatistico de energia eletrica_filtrado.csv",
        "./dados filtrados/INMET_BRASILIA_01-01-2024_A_31-12-2024_filtrado.csv",
        "./dados filtrados/INMP 15102025-15102025_filtrado.csv",
        output_path="dados_unificados.csv",
        preencher_ausentes=True, # Alterei para True, pode ajudar na consistência
    )
```

Gerando o Gráfico

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import numpy as np
import os

def realizar_regressao_mensal(csv_path: str):
    """
    Realiza a regressão linear entre Consumo (kW) e Temperatura Média (C)
    em um nível de agregação mensal para melhorar a correlação R.

    @param csv_path: Caminho para o arquivo CSV unificado.
    @return: Tuple contendo o coeficiente angular (slope), o intercepto, R² e R.
    """

    # === 1. Carregar e Pré-processar o CSV ===
    df = pd.read_csv(csv_path)

    # Seleciona apenas as colunas necessárias e remove NaNs
    df = df[["data", "consumo_kw", "temperatura_media"]].dropna()
    df["data"] = pd.to_datetime(df["data"], errors="coerce")

    # === PONTO DE CORREÇÃO: Agregação Mensal ===
    # Agrupa por Ano e Mês, calculando a MÉDIA de Consumo e Temperatura
    # Isso transforma a análise diária (com consumo repetido) em uma análise mensal
    df_mensal = df.set_index("data").resample("M").mean().dropna()

    # Se a agregação resultar em menos de 3 pontos, a regressão será fraca/impossível
    if len(df_mensal) < 3:
        print("[AVISO] Menos de 3 pontos de dados mensais. Regressão pode ser inválida.")

    # === 2. Preparar os dados (Mensais) ===
    # Reseta o índice para usar a temperatura_media como variável independente
    X = df_mensal["temperatura_media"].values.reshape(-1, 1)  # Variável independente
(Mensal)
    y = df_mensal["consumo_kw"].values  # Variável dependente (Mensal)

    # === 3. Criar e treinar o modelo de regressão linear ===
    model = LinearRegression()
    model.fit(X, y)

    # === 4. Previsões para a linha de regressão ===
    y_pred = model.predict(X)
```

```
    # === 5. Calcular R² e R ===
    r2 = r2_score(y, y_pred)
    r = np.sqrt(r2) * np.sign(model.coef_[0])  # R com sinal do coeficiente

    # === 6. Criar diretório para salvar gráficos ===
    os.makedirs("./graficos", exist_ok=True)

    # === 7. Plotar os dados e a linha de regressão ===
    plt.figure(figsize=(10, 6))

    # Plota os dados mensais (agregados)
    plt.scatter(X, y, color='blue', label="Dados Reais (Média Mensal)", alpha=0.7, s=100)

    # Plota a linha de regressão
    plt.plot(X, y_pred, color='red', linewidth=2, label="Regressão Linear Mensal")

    plt.xlabel("Temperatura Média Mensal (°C)")
    plt.ylabel("Consumo Médio Mensal (kW)")
    plt.title("Regressão Linear MENSAL: Consumo x Temperatura Média")
    plt.legend()
    plt.grid(True)

    # Adicionar R no gráfico
    plt.text(
        0.05, 0.95,
        f"R = {r:.3f}\nR² = {r2:.3f}\nN={len(df_mensal)} meses",
        transform=plt.gca().transAxes,
        fontsize=12,
        verticalalignment='top',
        bbox=dict(boxstyle="round,pad=0.3", facecolor="white", edgecolor="black")
    )

    # === 8. Salvar o gráfico ===
    file_name = f"regressao_consumo_temperatura_mensal_{len(df_mensal)}.png"
    plt.savefig(f"./graficos/{file_name}", dpi=300)
    plt.show()

    # === 9. Coeficientes da regressão ===
    print("--- Resultados da Regressão Linear MENSAL ---")
    print(f"Coeficiente angular (slope) - Beta 1: {model.coef_[0]:.4f}")
    print(f"Intercepto - Beta 0: {model.intercept_:.4f}")
    print(f"R²: {r2:.4f}")
    print(f"R: {r:.4f}")

    return model.coef_[0], model.intercept_, r2, r

# === Execução Principal ===
if __name__ == "__main__":
```

```
realizar_regressao_mensal("dados_unificados.csv")
```