DOCUMENTAÇÃO DE DATASET(S) PARA DEEP LEARNING:

Tratamento e Visualização de Dados Fiscais: Análise da Arrecadação por Estado e Ano

André Carvalho,
Carlos Alberto Braga,
Fernanda de Souza Borges,
Letícia Araújo Costa,
Pedro Mello Antunes

1. Transposição e Limpeza Inicial dos DataFrames

- Os dataframes serie_historica_2021, serie_historica_2022 e serie_historica_2023 são transpostos para organizar os dados de forma que as colunas se tornem linhas, e vice-versa. Isso é útil quando os dados estão estruturados com anos/valores como linhas e precisamos reorganizá-los.

```
serie_historica_2021 = serie_historica_2021.transpose()
serie_historica_2022 = serie_historica_2022.transpose()
serie_historica_2023 = serie_historica_2023.transpose()
```

- Após a transposição, a primeira linha de cada dataframe, que contém os nomes das colunas, é usada para definir os nomes das colunas. Em seguida, essa linha é removida do dataframe, restando apenas os dados relevantes.

```
serie_historica_2021.columns = serie_historica_2021.iloc[0]
serie_historica_2021 = serie_historica_2021[1:]

serie_historica_2022.columns = serie_historica_2022.iloc[0]
serie_historica_2022 = serie_historica_2022[1:]

serie_historica_2023.columns = serie_historica_2023.iloc[0]
serie_historica_2023 = serie_historica_2023[1:]
```

2. Conversão de Tipos de Dados

- Uma função é criada para tentar converter os valores em cada célula dos dataframes para o tipo float, ignorando qualquer valor que não seja numérico (e.g., texto ou valores nulos), retornando None quando a conversão falha.

```python
# Função para tentar converter valores para float, ignorando aqueles que não são numéricos
def try_convert_to_float(x):
    try:
        return float(x)
    except (ValueError, TypeError):
        return None  # Retorna None para valores que não podem ser convertidos
```

- Esta função é aplicada a todos os elementos dos dataframes usando applymap, que executa a função em cada célula.

```python
# Aplicar a função aos dataframes
serie_historica_2021 = serie_historica_2021.applymap(try_convert_to_float)
serie_historica_2022 = serie_historica_2022.applymap(try_convert_to_float)
serie_historica_2023 = serie_historica_2023.applymap(try_convert_to_float)
```

3. Tratamento dos Dados de Arrecadação

- Para o dataframe arrecadacao_estado, todas as colunas, exceto 'Mês' e 'UF', são convertidas para o tipo float, utilizando a função pd.to_numeric, que tenta a conversão e substitui valores inválidos por 0.

```python
for column in arrecadacao_estado.columns:
  if column not in ['Mês', 'UF']:
    try:
      arrecadacao_estado[column] = pd.to_numeric(arrecadacao_estado[column], errors='coerce').fillna(0).astype(float)
    except:
      pass
```

- Filtra-se o dataframe arrecadacao_estado para incluir apenas os anos de 2021 a 2023.

```python
arrecadacao_estado = arrecadacao_estado[
    arrecadacao_estado['Ano'].isin([2021, 2022, 2023])
]
```

4. Análise de Dados

- Uma análise é realizada para verificar valores faltantes (NaN) e colunas do tipo datetime em cada um dos dataframes. Isso garante a integridade dos dados e identifica possíveis colunas que precisam de tratamento adicional.

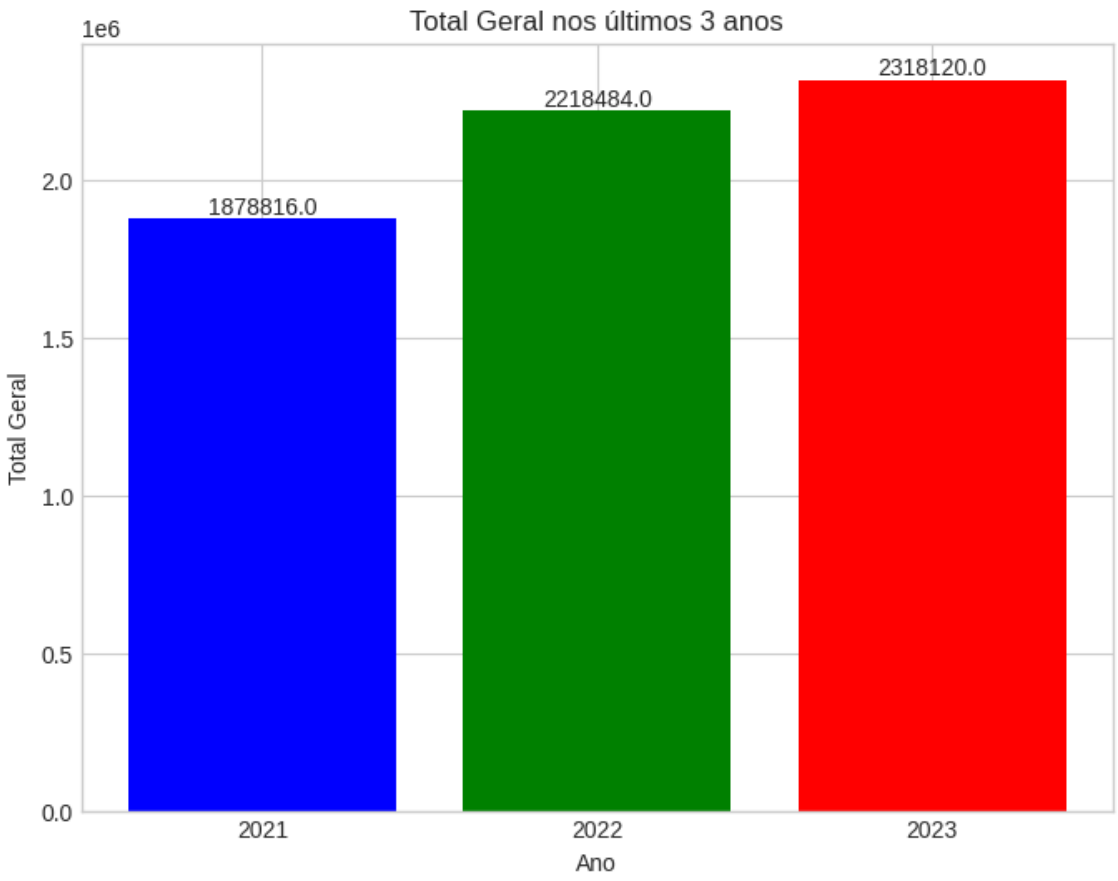
```python
# Lista de datasets e seus nomes
datasets = [serie_historica_2021, serie_historica_2022, serie_historica_2023, arrecadacao_estado]
nomes = ['serie_historica_2021', 'serie_historica_2022', 'serie_historica_2023', 'arrecadacao_estado']

# Função para verificar NaN e NaT
for df, nome in zip(datasets, nomes):
    print(f'\nAnalisando o dataset {nome}:')

    # Verificar total de NaN
    total_nan = df.isna().sum().sum()
    print(f'Total de valores NaN: {total_nan}')

    # Verificar colunas com NaN
    colunas_com_nan = df.columns[df.isna().any()].tolist()
    print(f'Colunas com valores NaN: {colunas_com_nan}')

    # Verificar colunas do tipo datetime
    colunas_datetime = df.select_dtypes(include=['datetime', 'datetime64[ns]']).columns.tolist()
    if colunas_datetime:
        for col in colunas_datetime:
            total_nat = df[col].isna().sum()
            print(f'Coluna {col} possui {total_nat} valores NaT')
    else:
        print('Não há colunas do tipo datetime para verificar NaT.')
```

```
Analisando o dataset serie_historica_2021:
Total de valores NaN: 13
Colunas com valores NaN: ['CONTRIBUIÇÃO PARA O FUNDAF']
Não há colunas do tipo datetime para verificar NaT.

Analisando o dataset serie_historica_2022:
Total de valores NaN: 13
Colunas com valores NaN: ['CONTRIBUIÇÃO PARA O FUNDAF']
Não há colunas do tipo datetime para verificar NaT.

Analisando o dataset serie_historica_2023:
Total de valores NaN: 13
Colunas com valores NaN: ['CONTRIBUIÇÃO PARA O FUNDAF']
Não há colunas do tipo datetime para verificar NaT.

Analisando o dataset arrecadacao_estado:
Total de valores NaN: 0
Colunas com valores NaN: []
Não há colunas do tipo datetime para verificar NaT.
```

5. Cálculo e Visualização de Totais

- O valor total da arrecadação para cada ano é extraído e armazenado em variáveis. Esses totais são usados para gerar um gráfico de barras comparando a arrecadação dos três últimos anos.

```
total_2021 = serie_historica_2021.loc['TOTAL', 'TOTAL GERAL [E]=[C]+[D]']
total_2022 = serie_historica_2022.loc['TOTAL', 'TOTAL GERAL [E]=[C]+[D]']
total_2023 = serie_historica_2023.loc['TOTAL', 'TOTAL GERAL [E]=[C]+[D]']

anos = ['2021', '2022', '2023']
totais = [total_2021, total_2022, total_2023]


plt.figure(figsize=(8, 6))
bars = plt.bar(anos, totais, color=['blue', 'green', 'red'])

# Adiciona rótulos de dados acima de cada barra
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 2), ha='center', va='bottom')
plt.xlabel('Ano')
plt.ylabel('Total Geral')
plt.title('Total Geral nos últimos 3 anos')
plt.show()
```

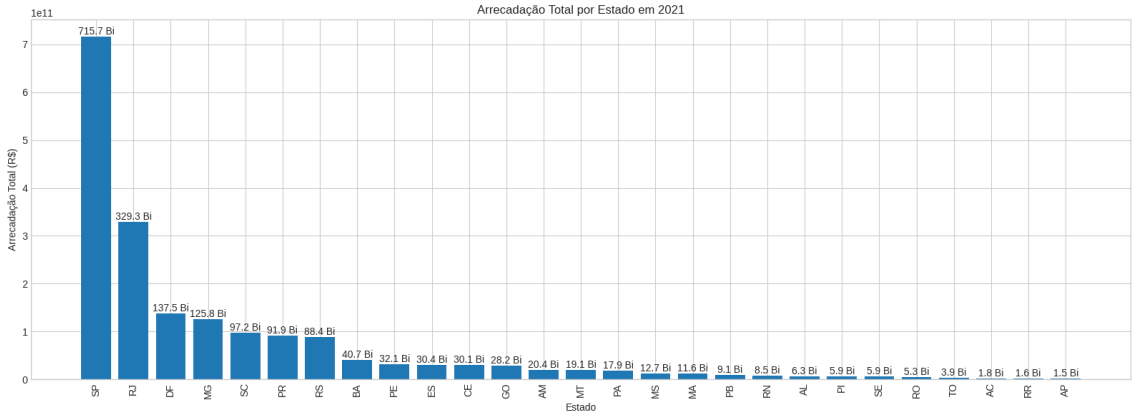


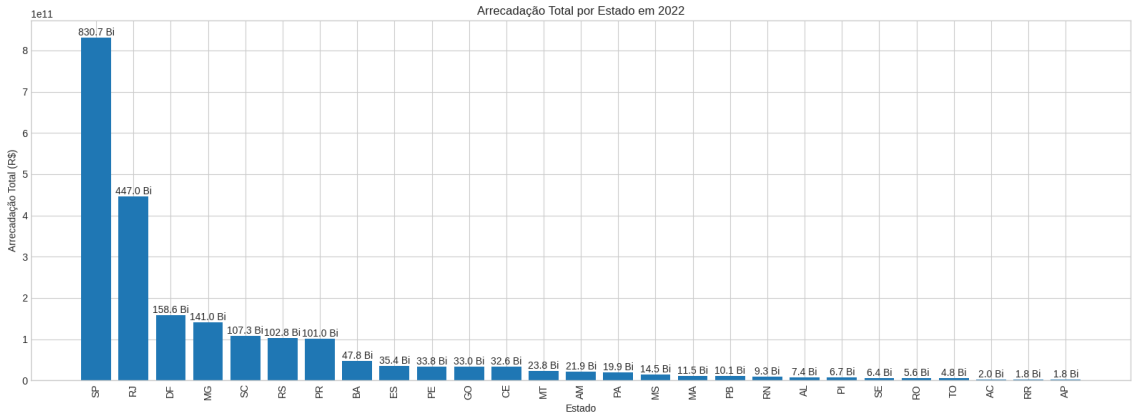
6. Análise de Arrecadação por Estado

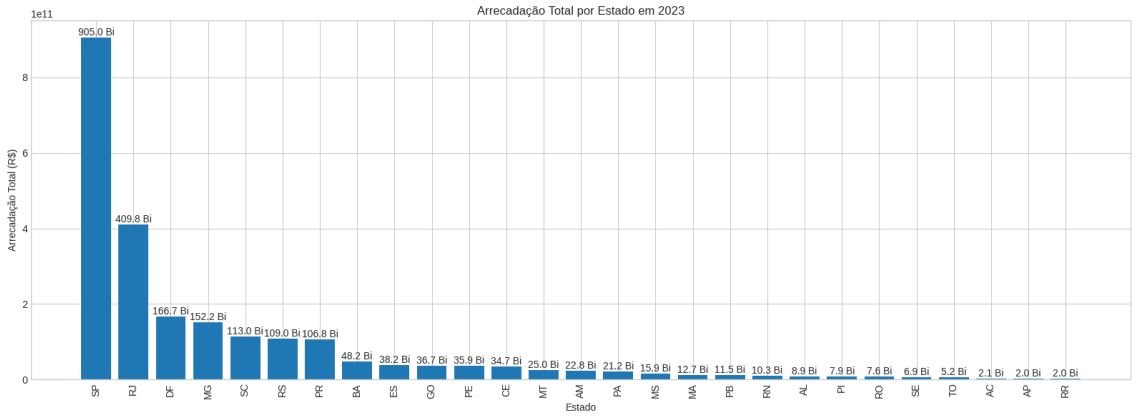
- Uma nova coluna, Soma Total, é criada para somar todas as colunas numéricas da arrecadação para cada linha. Em seguida, os dados são

agrupados por ano e estado (UF), somando os valores totais de arrecadação.

```
arrecadacao_estado['Soma Total'] = arrecadacao_estado.select_dtypes(include=np.number).sum(axis=1)

# Agrupar por ano e UF e somar a arrecadação total
arrecadacao_por_ano_uf = arrecadacao_estado.groupby(['Ano', 'UF'])['Soma Total'].sum().unstack()

# Criar um gráfico para cada ano
for ano in [2021, 2022, 2023]:
  plt.figure(figsize=(16, 6))
  plt.title(f'Arrecadação Total por Estado em {ano}')
  plt.xlabel('Estado')
  plt.ylabel('Arrecadação Total (R$)')
  valores = arrecadacao_por_ano_uf.loc[ano].sort_values(ascending=False)

  # Converter valores para bilhões e formatar o rótulo
  valores_bilhoes = [f'{valor/1000000000:.1f} Bi' for valor in valores]

  plt.bar(valores.index, valores.values)
  plt.xticks(rotation=90)

  # Adicionar rótulos acima das barras
  for i, v in enumerate(valores):
      plt.text(i, v, valores_bilhoes[i], ha='center', va='bottom')

  plt.tight_layout()
  plt.show()
```






Arrecadação Total por Estado em 2023
1e11
Arrecadação Total (R$)
Estado
905.0 Bi
409.8 Bi
166.7 Bi
152.2 Bi
113.0 Bi
109.0 Bi
106.8 Bi
48.2 Bi
38.2 Bi
36.7 Bi
35.9 Bi
34.7 Bi
25.0 Bi
22.8 Bi
21.2 Bi
15.9 Bi
12.7 Bi
11.5 Bi
10.3 Bi
8.9 Bi
7.9 Bi
7.6 Bi
6.9 Bi
5.2 Bi
2.1 Bi
2.0 Bi
2.0 Bi
SP
RJ
DF
MG
SC
RS
PR
BA
ES
GO
PE
CE
MT
AM
PA
MS
MA
PB
RN
AL
PI
RO
SE
TO
AC
AP
RR