📊 NOTEBOOK 1: COLETA E PREPARAÇÃO DE DADOS

Análise de Sentimento em Decisões Judiciais - TJCE

🔧 1. CONFIGURAÇÃO DO AMBIENTE

1.1 Instalação de Dependências

Descrição: Instalação das bibliotecas Python necessárias.

```
# Instalar bibliotecas necessárias
!pip install -r requirements.txt
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (from -r requirements.txt (line 1)) (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from -r requirements.txt (line 2)) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas->-r requirements.txt (line 1
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas->-r requirements.txt (line 1)) (2025.2
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas->-r requirements.txt (line 1)) (2025
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas->-r requirements.t
```

1.2 Importação de Bibliotecas

```
import pandas as pd
import json
from collections import Counter
import requests
import csv
```

📥 2. COLETA DE DADOS VIA API CNJ *DATAJUD*

2.1 Configuração da API e Parâmetros de Busca

Descrição: Configuração da requisição à API pública do CNJ. O código de assunto 12487 refere-se a "Fornecimento de medicamentos", encontrado no site oficial do CNJ (https://www.cnj.jus.br/sgt/consulta_publica_assuntos.php).

Para encontrar o código no site do cnj:

1. Procure por "Medicamentos"
2. Selecione a 4ª opção: `Fornecimento de medicamentos`
3. Verifique o código que aparece, `12487 Fornecimento de medicamentos`

```
url = "https://api-publica.datajud.cnj.jus.br/api_publica_tjce/_search"
api_key = "APIKey cDZHYzlZa0JadVREZDJCendQbXY6SkJlTzNjLV9TRENyQk1RdnFKZGRQdw=="

payload = json.dumps({
    "size": 10000,
    "query": {"match": {"assuntos.codigo": "12487"}},  # Código: Fornecimento de medicamentos
    "sort": [{"dataAjuizamento": {"order": "desc"}}]
})

headers = {
    'Authorization': api_key,
    'Content-Type': 'application/json'
}
```

2.2 Requisição e Validação dos Dados

Descrição: Execução da requisição HTTP POST para a API e validação inicial do volume de dados retornados.

```
response = requests.request("POST", url, headers=headers, data=payload)
dados_dict = response.json()
print(f"Total de processos encontrados na API: {dados_dict['hits']['total']['value']}")
print(f"Processos retornados nesta consulta: {len(dados_dict['hits']['hits'])}")
```

```
Total de processos encontrados na API: 6046
Processos retornados nesta consulta: 6046
```

🔄 3. PROCESSAMENTO E ESTRUTURAÇÃO DOS DADOS

3.1 Extração de Campos Essenciais

Descrição: Extração dos campos essenciais de cada processo (número, grau de jurisdição, data de ajuizamento e movimentos processuais) e criação do DataFrame principal.

```
processos = []

for hit in dados_dict['hits']['hits']:
    processo = hit['_source']

    numero_processo = processo['numeroProcesso']
    grau = processo['grau']
    data_ajuizamento = processo['dataAjuizamento']
    movimentos = processo.get('movimentos', [])

    processos.append([
        numero_processo,
        grau,
        data_ajuizamento,
        movimentos
    ])

df = pd.DataFrame(
    processos,
    columns=[
        'numero_processo',
        'grau',
        'data_ajuizamento',
        'movimentos'
    ]
)

print(f"Total de processos no DataFrame: {len(df)}")
df.sample(5)
```

```
Total de processos no DataFrame: 6046
```

	numero_processo	grau	data_ajuizamento	movimentos
5042	06370019320218060000	G2	20211117000000	[{'complementosTabelados': [{'codigo': 4, 'val...
5726	06285978720208060000	G2	20200625000000	[{'complementosTabelados': [{'codigo': 3, 'val...
5496	00507452820218060091	G1	20210403000000	[{'codigo': 1051, 'nome': 'Decurso de Prazo', ...
4923	02000199420228060038	G1	20220113000000	[{'complementosTabelados': [{'codigo': 18, 'va...
1701	02082644020248060001	G1	20240207000000	[{'complementosTabelados': [{'codigo': 4, 'val...

🔍 4. IDENTIFICAÇÃO E ANÁLISE DE DECISÕES

4.1 Definição de Termos de Decisão

Descrição: Definição dos termos-chave que caracterizam decisões judiciais relevantes para a análise.

```
from collections import Counter

termos_decisao = [
    "Procedência",
    "Improcedência",
    "Improcedência do pedido e improcedência do pedido contraposto",
    "Procedência do pedido e procedência do pedido contraposto",
    "Deferido",
    "Indeferido",
]
```

4.2 Extração de Decisões dos Movimentos Processuais

Descrição: Busca sistemática nos movimentos processuais por termos de decisão.

Importante: Filtra apenas processos de primeira instância (grau G1) para garantir homogeneidade na análise.

```
decisoes_por_processo = []
tipos_decisao_contagem = []

for _, row in df.iterrows():
    numero = row['numero_processo']
    movimentos = row['movimentos']
    grau = row['grau']
```

```
    data_ajuizamento = row['data_ajuizamento']

    decisoes_encontradas = []

    if movimentos:
        for mov in movimentos:
            nome_mov = mov.get('nome', '')

            # Filtrar apenas decisões de primeira instância (G1)
            if any(termo in nome_mov for termo in termos_decisao) and grau == 'G1':
                decisoes_encontradas.append(nome_mov)
                tipos_decisao_contagem.append(nome_mov)

    if decisoes_encontradas:
        decisoes_por_processo.append({
            'numero_processo': numero,
            'data_ajuizamento': data_ajuizamento,
            'decisoes': decisoes_encontradas
        })

print(f"Processos com decisões: {len(decisoes_por_processo)} de {len(df)}")
print("\nTipos de decisões encontradas:")
for tipo, count in Counter(tipos_decisao_contagem).most_common(10):
    print(f"  {tipo}: {count}")
```

```
Processos com decisões: 1799 de 6046

Tipos de decisões encontradas:
  Procedência: 1284
  Procedência em Parte: 259
  Improcedência: 222
  Procedência em parte do pedido e improcedência do pedido contraposto: 30
  Improcedência do pedido e improcedência do pedido contraposto: 14
  Procedência do pedido e improcedência do pedido contraposto: 5
  Procedência do Pedido - Reconhecimento pelo réu: 3
  Procedência em parte do pedido e procedência do pedido contraposto: 1
  Procedência do pedido e procedência do pedido contraposto: 1
```

4.3 Criação de DataFrame de Decisões

Descrição: Criação de um DataFrame específico para decisões, removendo duplicatas por processo (mantém apenas a primeira decisão encontrada).

```
decisoes_lista = []

for item in decisoes_por_processo:
    for decisao in item['decisoes']:
        decisoes_lista.append({
            'numero_processo': item['numero_processo'],
            'tipo_decisao': decisao,
        })

df_decisoes = pd.DataFrame(decisoes_lista)
df_decisoes = df_decisoes.drop_duplicates(subset='numero_processo', keep='first')
df_decisoes.head(10)
```

	numero_processo	tipo_decisao
0	30039390220258060071	Improcedência
1	02187331420258060001	Procedência
2	30465508320258060001	Procedência em Parte
3	30415587920258060001	Procedência em Parte
4	30048329320258060167	Procedência
5	02162085920258060001	Procedência
6	30355412720258060001	Procedência
7	30022788520258060071	Procedência
8	30303751420258060001	Procedência em Parte
9	30292622520258060001	Procedência

Próximas etapas: Gerar código com df_decisoes | New interactive sheet

📊 5. SEGMENTAÇÃO POR TIPO DE DECISÃO

5.1 Separação por Categorias de Decisão

Descrição: Segmentação do dataset em categorias de decisão para análise comparativa posterior.

```
# Separar decisões por tipo
df_procedencia = df_decisoes[df_decisoes['tipo_decisao'] == 'Procedência'].copy()
df_improcedencia = df_decisoes[df_decisoes['tipo_decisao'] == 'Improcedência'].copy()
df_improcedencia_contraposto = df_decisoes[df_decisoes['tipo_decisao'] == 'Improcedência do pedido e improcedência do pedido contrapo
df_procedencia_contraposto = df_decisoes[df_decisoes['tipo_decisao'] == 'Procedência do pedido e procedência do pedido contraposto'].

df_decisoes_completo = pd.concat([
    df_procedencia,
    df_improcedencia,
    df_improcedencia_contraposto,
    df_procedencia_contraposto
])

print(f"\n=== TODOS OS REGISTROS COM DECISÃO ===")
print(f"Total de registros: {len(df_decisoes_completo)}")
print(f"\nDistribuição por tipo:")
print(f"  - Procedência: {len(df_procedencia)}")
print(f"  - Improcedência: {len(df_improcedencia)}")
print(f"  - Improcedência (contraposto): {len(df_improcedencia_contraposto)}")
print(f"  - Procedência (contraposto): {len(df_procedencia_contraposto)}")

df_decisoes_completo
```

```
=== TODOS OS REGISTROS COM DECISÃO ===
Total de registros: 1497

Distribuição por tipo:
  - Procedência: 1264
  - Improcedência: 219
  - Improcedência (contraposto): 13
  - Procedência (contraposto): 1
```

	numero_processo	tipo_decisao
1	02187331420258060001	Procedência
4	30048329320258060167	Procedência
5	02162085920258060001	Procedência
6	30355412720258060001	Procedência
7	30022788520258060071	Procedência
...	...	...
844	02047761420238060001	Improcedência do pedido e improcedência do ped...
1070	02063489420228060112	Improcedência do pedido e improcedência do ped...
1247	02006673320228060181	Improcedência do pedido e improcedência do ped...
1621	02226629420218060001	Improcedência do pedido e improcedência do ped...
898	02073704020228060064	Procedência do pedido e procedência do pedido ...

1497 rows × 2 columns

Próximas etapas: Gerar código com df_decisoes_completo | New interactive sheet

6. LIMPEZA E CONSOLIDAÇÃO DO DATASET

6.1 Unificação dos DataFrames

Descrição: Integração da coluna de tipo de decisão ao DataFrame principal.

```
df['tipo_decisao'] = df_decisoes_completo['tipo_decisao']
display(df.head(4))
```

	numero_processo	grau	data_ajuizamento	movimentos	tipo_decisao
0	30854495320258060001	G1	20250930000000	[{'complementosTabelados': [{'codigo': 2, 'val...	Improcedência
1	02257145920258060001	G1	20250930000000	[{'complementosTabelados': [{'codigo': 2, 'val...	Procedência
2	30052433620258060071	G1	20250927000000	[{'complementosTabelados': [{'codigo': 3, 'val...	NaN
3	02068081820258060293	G1	20250926000000	[{'complementosTabelados': [{'codigo': 19, 'va...	NaN

6.2 Remoção de Valores Nulos

```
# Removendo registros NaN
df = df.dropna(subset=['tipo_decisao'])
display(df.head(4))
```

	numero_processo	grau	data_ajuizamento	movimentos	tipo_decisao
0	30854495320258060001	G1	20250930000000	[{'complementosTabelados': [{'codigo': 2, 'val...	Improcedência
1	02257145920258060001	G1	20250930000000	[{'complementosTabelados': [{'codigo': 2, 'val...	Procedência
4	30834471320258060001	G1	20250925000000	[{'codigo': 1061, 'nome': 'Disponibilização no...	Procedência
5	30034125320258060070	G1	20250925000000	[{'complementosTabelados': [{'codigo': 19, 'va...	Procedência

6.3 Remoção de Colunas Intermediárias

```
# Removendo colunas intermediárias
df = df.drop(columns=['grau', 'movimentos', 'tipo_decisao'])
display(df.head(4))
```

	numero_processo	data_ajuizamento
0	30854495320258060001	20250930000000
1	02257145920258060001	20250930000000
4	30834471320258060001	20250925000000
5	30034125320258060070	20250925000000

6.4 Reset de Índice

```
# Resetando index para manter organização do dataframe
df = df.reset_index(drop=True)
display(df.head(4))
```

	numero_processo	data_ajuizamento
0	30854495320258060001	20250930000000
1	02257145920258060001	20250930000000
2	30834471320258060001	20250925000000
3	30034125320258060070	20250925000000

7. EXTRAÇÃO DE FEATURES TEMPORAIS

7.1 Extração de Dia da Semana e Ano

Descrição: Criação de features temporais (dia da semana e ano) para análise de padrões temporais nas decisões judiciais.

```
import pandas as pd

# Converter data_ajuizamento para datetime
df['data_ajuizamento_dt'] = pd.to_datetime(
    df['data_ajuizamento'],
    format='%Y%m%d%H%M%S'
)

dias_semana = {0: "Segunda", 1: "Terça", 2: "Quarta", 3: "Quinta", 4: "Sexta"}

# Extrair dia da semana (segunda = 0 | terça = 1 | ... | sexta = 4)
df['dia_semana'] = df['data_ajuizamento_dt'].dt.weekday.map(dias_semana)
df['ano'] = df['data_ajuizamento_dt'].dt.year

# Excluir coluna intermediária
df = df.drop(columns=['data_ajuizamento', 'data_ajuizamento_dt'])

df
```

	numero_processo	dia_semana	ano
0	30854495320258060001	Terça	2025
1	02257145920258060001	Terça	2025
2	30834471320258060001	Quinta	2025
3	30034125320258060070	Quinta	2025
4	30830790420258060001	Quarta	2025
...	...	...	...
1492	02004050620228060045	Quinta	2024
1493	02018139620248060001	Quinta	2024
1494	30001654020238060036	Quinta	2024
1495	02019870820248060001	Quinta	2024
1496	02001240420248060167	Quinta	2024

1497 rows × 3 columns

Próximas etapas: Gerar código com df | New interactive sheet

7.2 Verificando ano mínimo e máximo do conjunto de dados

```
min = df['ano'].min()
max = df['ano'].max()

print(f"Ano mínimo: {min}")
print(f"Ano máximo: {max}")
```

```
Ano mínimo: 2024
Ano máximo: 2025
```

💾 8. EXPORTAÇÃO DE DADOS INTERMEDIÁRIOS

8.1 Exportação para CSV

Descrição: Exportação dos dados processados para uso posterior:

- processos_decisao_dia_ano.csv: Dataset completo com decisões e features temporais
- numeros_processos.csv: Lista única de números de processo para scraping de PDFs

```
df.to_csv("processos_dia_ano.csv", sep=';', index=False)
df['numero_processo'].drop_duplicates().to_csv("numeros_processos.csv", index=False)

print("\n=== ARQUIVOS EXPORTADOS ===")
print("  - processos_dia_ano.csv")
print("  - numeros_processos.csv")
```

```
=== ARQUIVOS EXPORTADOS ===
  - processos_dia_ano.csv
  - numeros_processos.csv
```

📄 9. INSTRUÇÕES PARA COLETA DE DOCUMENTOS COMPLETOS

9.1 Scraping de PDFs (Executar Localmente)

Pré-requisitos:

1. Clone o repositório: `git clone https://github.com/GiovanniBrigido/trabalho-final-deep-learning.git`
2. Navegue até o diretório do projeto
3. Execute os seguintes comandos:

Comandos:

```
python -m venv venv
pip install -r requirements.txt
playwright install chromium
python ./scraper_pdf_tjce.py
```

O que o script faz:

- Lê o arquivo numeros_processos.csv

- Acessa o site do TJCE
- Baixa os PDFs das sentenças completas
- Salva os arquivos na pasta data/decisoes

9.2 Extração de Texto dos PDFs (Executar Localmente)

Após gerar os arquivos básicos acima, execute o script `scraper_pdf_tjce.py` para realizar a extração de PDF's.

Execute:

```
python ./extrair_decisoes.py
```

O que o script faz:

- Lê todos os PDFs baixados
- Extrai o texto completo de cada sentença com a bib PyPDF2
- Gera o arquivo decisoes_extraidas.csv com:
    - Número do processo
    - Texto completo da decisão
    - Metadados adicionais

Arquivo gerado:

- decisoes_extraidas.csv
- **!! SERÁ USADO NO NOTEBOOK 2 !!**