Relatório Técnico: Extração e Processamento de Dados da API SIDRA (IBGE) com Python

Sumário Executivo: Uma Arquitetura Robusta para Dados On-Demand do IBGE

O Sistema IBGE de Recuperação Automática (SIDRA) é a fonte primária e mais abrangente de dados estatísticos agregados do Brasil, disponibilizando um vasto acervo de mais de 5.200 tabelas e 850 bilhões de valores provenientes de 55 pesquisas, incluindo censos e estudos conjunturais.[1] Para cientistas de dados, analistas de indústria e pesquisadores, o acesso programático a este repositório via sua Interface de Programação de Aplicações (API) é um componente essencial para a automação de análises econômicas e sociais.

Este relatório técnico aborda uma demanda de engenharia de dados específica: a extração de indicadores de (1) População Economicamente Ativa (Força de Trabalho), com desagregação por sexo, e (2) Rendimento Domiciliar Per Capita. A metodologia detalhada atende à requisição de uma solução robusta em Python que permita "gerar dinamicamente as URLs" de consulta e "tratar os JSONs retornados", utilizando as bibliotecas-padrão do ecossistema de dados (requests e pandas).

A arquitetura de solução proposta neste documento é estruturada em duas fases distintas, garantindo resiliência e transparência ao pipeline de dados:

1. **Fase de Descoberta (Metadados):** Antes de consultar os dados, o pipeline deve consultar o *endpoint* de metadados da API v3 (.../agregados/{tabela}/metadados).[3] Esta etapa permite a descoberta programática dos IDs corretos para as variáveis (ex: "Pessoas na força de trabalho") e classificações (ex: "Sexo" e suas categorias "Homens", "Mulheres"), eliminando a necessidade de fixar (hardcoding) valores frágeis no código.
2. **Fase de Extração (Dados):** Com os IDs corretos em mãos, o pipeline constrói dinamicamente a URL de consulta de dados, executa a chamada HTTP (Interface de Programação de Aplicações) usando a biblioteca requests e processa a resposta JSON (JavaScript Object Notation) aninhada. O tratamento dos dados é feito de forma eficiente, convertendo a resposta da API diretamente em um pandas.DataFrame limpo e pronto para análise.

Embora existam bibliotecas "wrapper" (invólucros) que simplificam a interação, como sidrapy

[4] e DadosAbertosBrasil.ibge [5], a abordagem "pura" (raw) detalhada aqui é tecnicamente superior para pipelines de dados em nível de produção. Ela oferece controle total sobre a sessão HTTP (incluindo o tratamento de exceções e configurações de segurança específicas), elimina dependências de terceiros que podem ser descontinuadas e garante total transparência sobre o processo de extração e transformação dos dados.
Este relatório fornecerá a sintaxe da API SIDRA, o mapeamento das tabelas e variáveis relevantes (Tabela 4093 para Força de Trabalho e Tabela 7531 para Renda Per Capita), e os scripts Python completos para implementar as fases de descoberta e extração, incluindo uma solução crítica para erros comuns de conexão SSL (Secure Sockets Layer).[6]

1. Fundamentos da API SIDRA v3: Decodificando a Sintaxe de Consulta

Para construir uma URL de consulta dinâmica, é imperativo primeiro compreender a sintaxe e a estrutura de parâmetros da API SIDRA v3. A URL base para todas as consultas a dados agregados (tabelas) é:
https://servicodados.ibge.gov.br/api/v3/agregados/ [3]
Uma consulta de dados completa é construída anexando-se "parâmetros" a esta URL base, separados pelo caractere /. Cada parâmetro define uma dimensão da consulta (a tabela, o período, a variável, etc.).

O "Alfabeto" SIDRA: Decodificando os Parâmetros de URL

Os parâmetros de consulta são definidos por um prefixo de uma letra (ex: /t/ para Tabela) seguido pelos valores desejados.

- **Tabela (/t/):** O ID numérico da tabela (agregado) a ser consultada.
  - Exemplo: /t/4093 (Tabela 4093 da PNAD Contínua).
- **Período (/p/):** Define o período de tempo da consulta.
  - Exemplo: /p/last 1 (último período disponível), /p/all (todos os períodos), /p/2022,2023 (anos específicos), /p/202301-202312 (range mensal).[4]
- **Variável (/v/):** O ID numérico da(s) métrica(s) desejadas (o que está sendo medido).
  - Exemplo: /v/10824 (Rendimento médio mensal real domiciliar per capita).[10] Múltiplos IDs podem ser separados por vírgula (ex: /v/606,609). O valor especial allxp retorna todas as variáveis, exceto as percentuais.[5]
- **Nível Territorial (/n/) e Localidades (/l/):** Define o escopo geográfico.
  - Exemplo: /n1/all (Nível 1 = Brasil, all = todos os itens desse nível, ou seja, o próprio Brasil).
  - Exemplo: /n3/all (Nível 3 = Unidades da Federação, all = todas as UFs).
  - Exemplo: /n6/all (Nível 6 = Municípios, all = todos os municípios).[5]

- **Classificação (/c/):** O parâmetro mais complexo e poderoso, usado para desagregar (quebrar) os dados por dimensões.
    - Formato: /c{ID_DA_CLASSIFICACAO}/{ID_DAS_CATEGORIAS}.
    - Exemplo: /c2/4,5 (Consultar a Classificação 2, mas retornar apenas os dados das Categorias 4 e 5).

O Endpoint de Metadados: A Chave para a Geração Dinâmica

A sintaxe de URL acima é inutilizável se os IDs das variáveis e classificações forem desconhecidos. Tentar adivinhar que "Sexo" é a classificação c2 ou que "Renda Per Capita" é a variável v/10824 é impossível, e fixar (hardcoding) esses valores no código-fonte torna o aplicativo frágil a futuras mudanças na API.
A solução profissional é usar o *endpoint* de metadados da API, que atua como um "manual de instruções" para cada tabela. A estrutura deste *endpoint* é:
https://servicodados.ibge.gov.br/api/v3/agregados/{ID_DA_TABELA}/metadados [3]
Ao consultar este *endpoint* (ex: .../4093/metadados), a API retorna um objeto JSON detalhado que lista todas as variaveis e classificacoes disponíveis para a tabela 4093, incluindo seus nomes legíveis (ex: "Sexo") e seus IDs numéricos (ex: 2). Este é o mecanismo que permite a geração verdadeiramente dinâmica e robusta de URLs. A estrutura da resposta dos metadados inclui chaves como id, nome, pesquisa, variaveis (uma lista de dicionários) e classificacoes (uma lista de dicionários, cada uma contendo suas categorias).[11]
A tabela a seguir resume a anatomia de uma URL de consulta de dados da API SIDRA.
Tabela 1: A Anatomia de uma URL da API SIDRA

Parâmetro	Prefixo	Exemplo	Descrição
Tabela	/t/	/t/4093	O ID do agregado (tabela) a ser consultado.
Período	/p/	/p/last 12	Os períodos de tempo (últimos 12, todos, um range).
Variável	/v/	/v/10824	O ID da(s) métrica(s) (valor numérico, percentual).
Nível Territorial	/n/	/n1/all	O nível geográfico (n1=Brasil, n3=UF, n6=Município).
Classificação	/c/	/c2/4,5	A dimensão de desagregação (ex: Sexo) e suas categorias (ex: Homens, Mulheres).

2. Mapeamento de Dados (Parte 1): População Economicamente Ativa (PEA) por Sexo

Definição e Identificação da Tabela

O primeiro objetivo é obter a "população economicamente ativa" (PEA). Este indicador, também conhecido como "Força de Trabalho", é definido pelo IBGE como a soma das pessoas ocupadas com as pessoas desocupadas (aquelas que não estavam trabalhando mas tomaram alguma iniciativa para conseguir emprego).[12]
A pesquisa de indicadores do IBGE, especificamente da Pesquisa Nacional por Amostra de Domicílios Contínua (PNAD Contínua), aponta para uma tabela central que agrega exatamente esta informação:

- **Tabela 4093:** "Pessoas de 14 anos ou mais de idade, total, na força de trabalho, ocupadas, desocupadas, fora da força de trabalho, e respectivas taxas e níveis, por sexo".[15]

Diversas publicações e relatórios do IBGE utilizam a Tabela 4093 como fonte para divulgar o "Nível da ocupação... por sexo" e a "Taxa de desocupação... por sexo".[16] Esta é, portanto, a tabela correta para a consulta solicitada.

Descoberta de Metadados (Tabela 4093)

Para consultar a Tabela 4093 de forma dinâmica, o primeiro passo é consultar seu *endpoint* de metadados: https://servicodados.ibge.gov.br/api/v3/agregados/4093/metadados.
Ao analisar o JSON de resposta deste *endpoint*, é possível identificar os IDs necessários para a consulta:

1. Na lista variaveis, localiza-se a métrica desejada. A Tabela 4093 oferece tanto o número absoluto quanto a taxa. Para a solicitação de "população... ativa", utiliza-se a variável de número absoluto:
    - **Variável:** "Pessoas na força de trabalho" (ID: 606).
2. Na lista classificacoes, localiza-se a dimensão de desagregação:
    - **Classificação:** "Sexo" (ID: 2).
3. Dentro da Classificação 2 ("Sexo"), na lista categorias, localizam-se os IDs para "Total", "Homens" e "Mulheres":
    - **Categoria:** "Total" (ID: 1).
    - **Categoria:** "Homens" (ID: 4).

- **Categoria:** "Mulheres" (ID: 5).

A consulta, portanto, usará a variável /v/606 e a classificação /c2/1,4,5 (Classificação 2, categorias 1, 4 e 5).

Tabela 2: Mapeamento de Metadados (Tabela 4093 - Força de Trabalho por Sexo)

Conceito Desejado	Parâmetro API	Nome na API (Exemplo)	ID Identificado
Tabela	/t/	Tabela 4093	4093
PEA (Absoluto)	/v/	Pessoas na força de trabalho	606
Desagregação	/c/	Sexo	2
Categoria (Total)	/c/2/	Total	1
Categoria (Homens)	/c/2/	Homens	4
Categoria (Mulheres)	/c/2/	Mulheres	5

3. Mapeamento de Dados (Parte 2): Rendimento Domiciliar Per Capita

Definição e Identificação da Tabela

O segundo objetivo é obter o "rendimento domiciliar per capita". Este é um indicador-chave de bem-estar econômico, também calculado com base na PNAD Contínua.[10]

A pesquisa por este indicador específico aponta claramente para uma tabela:

- **Tabela 7531:** "Rendimento médio mensal real domiciliar per capita, a preços médios do ano (Reais)".[10]

Descoberta de Metadados (Tabela 7531)

Uma análise de publicações que utilizam este dado fornece um atalho valioso. O documento em [10], ao citar a Tabela 7531, inclui uma URL de exemplo do SIDRA que revela o ID da variável: https://sidra.ibge.gov.br/tabela/7531#/n1/all/n3/all/v/10824/...

Este fragmento confirma que o ID da variável para "Rendimento médio mensal real domiciliar per capita" é 10824.

Esta consulta é mais simples que a anterior, pois a solicitação não especificou uma desagregação (como "por sexo"). A consulta será, portanto, direta para o Nível Territorial Brasil (/n1/all), Tabela 7531, e Variável 10824, para os períodos desejados.

Tabela 3: Mapeamento de Metadados (Tabela 7531 - Renda Per Capita)

Conceito Desejado	Parâmetro API	Nome na API (Exemplo)	ID Identificado
Tabela	/t/	Tabela 7531	7531
Renda Per Capita	/v/	Rendimento médio mensal real domiciliar per capita... (Reais)	10824
Nível Territorial	/n/	Brasil	n1/all

4. Implementação em Python (Fase 1): O Mecanismo de Descoberta de Metadados

O primeiro passo na implementação do código é criar as funções para realizar a Fase de Descoberta (consulta aos metadados).

O Desafio Técnico: Tratamento de Erros SSL

Um desafio técnico comum ao interagir com alguns servidores governamentais usando versões modernas de Python e da biblioteca requests é a ocorrência de erros de SSL, especificamente o erro OP_LEGACY_SERVER_CONNECT.[6] Isso ocorre porque o servidor pode usar um conjunto de cifras de segurança (TLS) mais antigo, que as bibliotecas de segurança mais recentes (como OpenSSL) recusam por padrão.
Uma consulta requests.get() simples pode falhar. A solução robusta é criar uma sessão requests com um adaptador (HTTPAdapter) personalizado que instrui o contexto SSL a permitir essa conexão legada. Sem esta configuração, o *script* pode falhar em diferentes ambientes de implantação.
O código a seguir implementa este adaptador e a função de descoberta de metadados.

Python

```
import requests
import pandas as pd
import ssl
from requests.adapters import HTTPAdapter

# --- Configuração do Adaptador TLS para Compatibilidade ---
# Esta classe personalizada permite a conexão com servidores que usam
```

```
# conjuntos de cifras TLS legados, evitando o erro 'OP_LEGACY_SERVER_CONNECT'.
# Referências:
class TLSAdapter(HTTPAdapter):
    """
    Um adaptador HTTP que permite a conexão com servidores
    que exigem opções de conexão SSL legadas.
    """
    def init_poolmanager(self, *args, **kwargs):
        # Cria um contexto SSL padrão
        ctx = ssl.create_default_context()
        # Permite cifras de nível de segurança 1 (mais permissivo que o padrão 2)
        ctx.set_ciphers("DEFAULT@SECLEVEL=1")
        # A opção crucial: permite a renegociação legada do servidor
        ctx.options |= 0x4  # ssl.OP_LEGACY_SERVER_CONNECT

        # Passa o contexto SSL personalizado para o PoolManager
        kwargs["ssl_context"] = ctx
        return super(TLSAdapter, self).init_poolmanager(*args, **kwargs)

# --- Função 1: Cliente de Sessão e Descoberta de Metadados ---

def get_sidra_metadata(table_id: int) -> dict | None:
    """
    Busca e retorna os metadados de uma tabela específica da API SIDRA v3.
    Utiliza o TLSAdapter para garantir a compatibilidade de conexão.

    :param table_id: O ID numérico da tabela (ex: 4093).
    :return: Um dicionário (dict) contendo os metadados da tabela,
             ou None em caso de falha.
    """
    # URL do endpoint de metadados
    metadata_url = f"https://servicodados.ibge.gov.br/api/v3/agregados/{table_id}/metadados"

    # Configura a sessão com o adaptador TLS
    session = requests.Session()
    session.mount("https://", TLSAdapter())

    print(f"Buscando metadados para a Tabela {table_id} em: {metadata_url}")

    try:
        response = session.get(metadata_url, timeout=10)
        # Levanta um erro HTTP se a resposta for 4xx ou 5xx
        response.raise_for_status()
```

```
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Erro ao buscar metadados da Tabela {table_id}: {e}")
        return None

# --- Função 2: Exemplo de Análise de Metadados ---

def find_metadata_ids(metadata: dict):
    """
    Função de exemplo para analisar o JSON de metadados e encontrar
    os IDs de variáveis e classificações desejadas.
    """
    print("\n--- Analisando Variáveis (buscando 'força de trabalho') ---")
    if 'variaveis' in metadata:
        for v in metadata['variaveis']:
            if "força de trabalho" in v['nome'].lower():
                print(f"  Encontrado: Nome='{v['nome']}', ID='{v['id']}'")

    print("\n--- Analisando Classificações (buscando 'sexo') ---")
    if 'classificacoes' in metadata:
        for c in metadata['classificacoes']:
            if "sexo" in c['nome'].lower():
                print(f"  Encontrada Classificação: Nome='{c['nome']}', ID='{c['id']}'")
                # Itera sobre as categorias dentro da classificação "Sexo"
                if 'categorias' in c:
                    for cat in c['categorias']:
                        print(f"    - Categoria: Nome='{cat['nome']}', ID='{cat['id']}'")

# --- Execução da Fase 1 (Exemplo) ---
# metadata_4093 = get_sidra_metadata(4093)
# if metadata_4093:
#     find_metadata_ids(metadata_4093)
```

5. Implementação em Python (Fase 2): Geração de URL, Consulta e Tratamento do JSON de Dados

Após a Fase de Descoberta identificar os IDs corretos, a Fase 2 constrói a URL e busca os dados. O desafio aqui é o tratamento do JSON de resposta.
Muitos tutoriais genéricos de API sugerem o uso da função pandas.json_normalize para achatar respostas JSON aninhadas.[21] No entanto, para a API SIDRA v3, esta *não* é a

abordagem mais eficiente.
A resposta de *dados* da API SIDRA v3 tem uma estrutura de lista (list) muito específica:

1. O JSON retornado é uma lista [...].
2. O **primeiro item** da lista (data) **não é um dado**. É um dicionário que contém os **cabeçalhos** (nomes das colunas).[11]
3. O **restante da lista** (data[1:]) contém os dicionários com os valores dos dados.

A abordagem correta e mais eficiente é ler os cabeçalhos do primeiro item e usar o restante dos itens para construir o DataFrame diretamente. Isso evita a complexidade desnecessária do json_normalize e mapeia os dados corretamente.
O código a seguir implementa as funções para buscar e processar os dados usando esta lógica específica do SIDRA.

Python

```
# (Continuação do script anterior, requer as importações: requests, pandas, ssl, TLSAdapter)

# --- Função 3: Coletor de Dados ---

def fetch_sidra_data(url: str, session: requests.Session) -> pd.DataFrame:
    """
    Busca dados da API SIDRA a partir de uma URL de consulta completa
    e os formata em um DataFrame do Pandas.

    Implementa a lógica de tratamento de JSON específica do SIDRA:
    - O item  da lista JSON é o cabeçalho.
    - Os itens [1:] da lista JSON são os dados.

    :param url: A URL de consulta de dados completa.
    :param session: A sessão 'requests' (com o TLSAdapter).
    :return: Um DataFrame do Pandas com os dados formatados.
    """
    print(f"Consultando dados em: {url}")
    try:
        response = session.get(url, timeout=30)
        response.raise_for_status()
        data_json = response.json()

        # Verifica se a resposta contém dados
        if not data_json or len(data_json) < 2:
            print("Resposta da API vazia ou sem dados.")
            return pd.DataFrame()
```

```
            # Extrai os cabeçalhos do primeiro item (ex: )
            # Os valores do dict são os nomes das colunas legíveis
            headers = list(data_json.values())

            # Extrai os dados dos itens restantes
            data_payload = data_json[1:]

            # Cria o DataFrame
            # Define as colunas no momento da criação para garantir a ordem correta
            df = pd.DataFrame(data_payload, columns=headers)

            return df

        except requests.exceptions.RequestException as e:
            print(f"Erro ao buscar dados: {e}")
        except (IndexError, KeyError) as e:
            print(f"Erro ao processar o JSON retornado. Estrutura inesperada: {e}")

        return pd.DataFrame()

# --- Função 4: Limpeza de Dados ---

def clean_sidra_df(df: pd.DataFrame) -> pd.DataFrame:
    """
    Realiza a limpeza básica em um DataFrame retornado pelo SIDRA.
    - Converte a coluna 'Valor' para numérico (tratando '...' e '-').
    - Remove linhas onde o 'Valor' é nulo.
    """
    if 'Valor' not in df.columns:
        print("DataFrame não contém a coluna 'Valor'. Retornando original.")
        return df

    # O IBGE usa '...' ou '-' para dados não disponíveis
    df['Valor'] = pd.to_numeric(df['Valor'], errors='coerce')

    # Remove linhas onde o valor não pôde ser convertido (NaN)
    df = df.dropna(subset=['Valor'])

    # Converte colunas de ID (como 'Ano (Código)') para tipos mais limpos, se desejar
    # Ex: df['Ano (Código)'] = df['Ano (Código)'].astype(int)

    return df
```

6. Estudo de Caso 1 (Código Completo): Força de Trabalho por Sexo (Tabela 4093)

Este *script* completo une todas as funções para buscar a População Economicamente Ativa (Força de Trabalho) da Tabela 4093, desagregada por Total, Homens e Mulheres, para os últimos 4 trimestres disponíveis.

Python

```
# --- Script Principal - Estudo de Caso 1 ---

# 1. Configurar a Sessão
# (Requer a classe TLSAdapter definida anteriormente)
session = requests.Session()
session.mount("https://", TLSAdapter())

# 2. Parâmetros de Geração de URL (Baseados na Fase 1)
tabela = "4093"
periodo = "last 4"  # Últimos 4 trimestres
variavel = "606"    # ID para "Pessoas na força de trabalho"
nivel = "n1/all"    # n1 = Nível Brasil
classificacao = "c2/1,4,5"  # c2 = "Sexo"; 1="Total", 4="Homens", 5="Mulheres"

# 3. Construção da URL Dinâmica
base_url = "https://servicodados.ibge.gov.br/api/v3/agregados"
data_url_pea = (
    f"{base_url}/t/{tabela}/p/{periodo}/v/{variavel}"
    f"/l/{nivel}/c/{classificacao}"
)

# 4. Execução da Coleta e Limpeza
# (Requer as funções fetch_sidra_data e clean_sidra_df)
df_pea_raw = fetch_sidra_data(data_url_pea, session)

if not df_pea_raw.empty:
    df_pea_clean = clean_sidra_df(df_pea_raw)

    print("\n--- Resultado: Força de Trabalho por Sexo (Tabela 4093) ---")
```

```
# Seleciona e renomeia colunas para melhor visualização
colunas_relevantes =
# Filtra colunas que existem no dataframe
colunas_presentes = [col for col in colunas_relevantes if col in df_pea_clean.columns]

print(df_pea_clean[colunas_presentes].to_markdown(index=False))
```

Resultado Esperado (Estudo de Caso 1)

Tabela 4: Resultado Final (DataFrame) - Força de Trabalho por Sexo

Trimestre (Código)	Brasil	Sexo (Código)	Sexo	Variável	Valor
202301	Brasil	1	Total	Pessoas na força de trabalho	108343132
202301	Brasil	4	Homens	Pessoas na força de trabalho	59901416
202301	Brasil	5	Mulheres	Pessoas na força de trabalho	48441716
202302	Brasil	1	Total	Pessoas na força de trabalho	108922129
202302	Brasil	4	Homens	Pessoas na força de trabalho	60166291
202302	Brasil	5	Mulheres	Pessoas na força de trabalho	48755838
...	...	...	...	...	...
(Nota: Os valores são exemplos baseados em dados reais e na estrutura da tabela 4093. A unidade é					

"Pessoas".)					

7. Estudo de Caso 2 (Código Completo): Renda Domiciliar Per Capita (Tabela 7531)

Este *script* utiliza a mesma sessão e as mesmas funções, mas altera os parâmetros para buscar o Rendimento Domiciliar Per Capita (anual) da Tabela 7531.

Python

```
# --- Script Principal - Estudo de Caso 2 ---

# 1. Reutilizar a Sessão
# (Assume que 'session' já foi criada com o TLSAdapter)

# 2. Parâmetros de Geração de URL (Baseados na Fase 1)
tabela = "7531"
periodo = "last 5"  # Últimos 5 anos (dados anuais)
variavel = "10824"   # ID para "Rendimento... per capita"
nivel = "n1/all"     # n1 = Nível Brasil
# Sem parâmetro /c/ (classificação), pois não há desagregação solicitada.

# 3. Construção da URL Dinâmica
data_url_renda = (
    f"{base_url}/t/{tabela}/p/{periodo}/v/{variavel}"
    f"/l/{nivel}"
)

# 4. Execução da Coleta e Limpeza
df_renda_raw = fetch_sidra_data(data_url_renda, session)

if not df_renda_raw.empty:
    df_renda_clean = clean_sidra_df(df_renda_raw)

    print("\n--- Resultado: Renda Domiciliar Per Capita (Tabela 7531) ---")

    # Seleciona e renomeia colunas para melhor visualização
    colunas_relevantes =
```

```
colunas_presentes = [col for col in colunas_relevantes if col in df_renda_clean.columns]

print(df_renda_clean[colunas_presentes].to_markdown(index=False))
```

Resultado Esperado (Estudo de Caso 2)

Tabela 5: Resultado Final (DataFrame) - Renda Domiciliar Per Capita

Ano (Código)	Brasil	Variável	Valor
2019	Brasil	Rendimento médio mensal real domiciliar per capita...	1366
2020	Brasil	Rendimento médio mensal real domiciliar per capita...	1349
2021	Brasil	Rendimento médio mensal real domiciliar per capita...	1353
2022	Brasil	Rendimento médio mensal real domiciliar per capita...	1450
2023	Brasil	Rendimento médio mensal real domiciliar per capita...	1625
(Nota: Os valores são exemplos baseados em dados reais [10] e na estrutura da tabela 7531. A unidade é "Reais".)			

8. Alternativas e Otimizações: O Uso de Bibliotecas Abstratas (Wrappers)

A metodologia "pura" detalhada acima, utilizando requests e pandas, foi requisitada e é a abordagem recomendada para pipelines de dados em produção devido ao seu controle e ausência de dependências. Contudo, para fins de análise exploratória rápida (como em

Jupyter Notebooks), o ecossistema Python oferece "wrappers" que abstraem essa complexidade.

- **Biblioteca sidrapy:** Esta é uma biblioteca popular focada exclusivamente em consumir a API SIDRA.[4]
    - **Uso:** import sidrapy; data = sidrapy.get_table(table_code="4093", period="last 4", variable="606", classification="2/1,4,5")
    - **Vantagens:** Extremamente simples e rápida. Os nomes dos parâmetros são intuitivos e mapeiam diretamente para a API.[9]
    - **Desvantagens:** É uma dependência externa. Se a biblioteca não for mantida ou se a API do IBGE mudar, o código quebrará. Oferece menos controle sobre a sessão HTTP (ex: o tratamento de SSL).
- **Biblioteca DadosAbertosBrasil.ibge:** Parte de um pacote maior para dados abertos brasileiros.[5]
    - **Uso:** from DadosAbertosBrasil import ibge; data = ibge.sidra(tabela=4093,...)
    - **Vantagens:** Possui funções auxiliares muito úteis, como ibge.lista_tabelas() e ibge.Metadados(), que automatizam a Fase de Descoberta.[5]
    - **Desvantagens:** Mesmas de qualquer *wrapper* de terceiros.

A escolha entre a abordagem "pura" (raw) e um "wrapper" depende do caso de uso. Para um *notebook* de análise único, sidrapy é eficiente. Para um *pipeline de dados de produção* (como um *pipeline* dagster-ibge [23]), depender de um *wrapper* é um risco de engenharia. A metodologia requests + pandas é livre de dependências (além das bibliotecas-padrão do ecossistema) e oferece a estabilidade e o controle necessários para software de nível de produção.

9. Conclusão e Recomendações Estratégicas de Implementação

Este relatório detalhou uma metodologia completa e robusta para a extração de dados da API SIDRA v3 do IBGE usando Python. A arquitetura de duas fases (Descoberta de Metadados e Extração de Dados) garante que as URLs de consulta sejam geradas dinamicamente, aumentando a resiliência do *pipeline* contra mudanças na API.

A solução técnica demonstrou como identificar as tabelas corretas (Tabela 4093 para Força de Trabalho por Sexo; Tabela 7531 para Renda Per Capita), como usar o *endpoint* /metadados [3] para encontrar IDs de variáveis e classificações, e como tratar a estrutura de resposta JSON específica do SIDRA.

Para a implementação desta solução em um ambiente de produção, as seguintes recomendações estratégicas devem ser seguidas:

1. **Utilizar o Adaptador TLS:** A inclusão do TLSAdapter [6] não é opcional; é uma medida de segurança e compatibilidade crucial para garantir que o *script* funcione em diferentes máquinas e ambientes de implantação sem falhas de conexão SSL.

2. **Abstrair e Reutilizar:** A lógica de "construir URL", "buscar dados" e "limpar DF" deve ser encapsulada em funções ou classes reutilizáveis (como demonstrado) para manter o código limpo, testável e de fácil manutenção.
3. **Implementar Cache de Metadados:** O *endpoint* /metadados não muda com frequência. Em vez de consultá-lo a cada execução, os metadados (o JSON de resposta) de tabelas críticas (como 4093 e 7531) devem ser baixados e armazenados em cache (seja como um arquivo JSON local ou em um sistema de cache como o Redis). O *pipeline* deve ler este cache para construir as URLs, atualizando-o apenas periodicamente (ex: uma vez por semana). Isso reduzirá drasticamente as chamadas à API e aumentará a velocidade de execução.
4. **Tratamento de Erros Robusto:** A chamada session.get() deve ser sempre acompanhada pela verificação do response.status_code. Códigos de erro do servidor (como 500 [24] ou 503) ou erros de cliente (400, 404) indicam um problema na API do IBGE ou na URL construída. O *pipeline* deve capturar esses erros, registrá-los (log) e ter uma política de nova tentativa (retry) ou falha controlada.
5. **Validação de Dados:** A estrutura dos dados do IBGE pode mudar sem aviso. O *script* deve incluir asserções (assertions) para validar a resposta. Por exemplo, após limpar o DF da Tabela 4093, deve-se verificar: assert 'Sexo' in df_pea_clean.columns e assert df_pea_clean['Valor'].dtype == 'float'. Se uma asserção falhar, o *pipeline* deve parar e alertar um operador humano, pois os metadados ou a estrutura da tabela provavelmente mudaram.

Referências citadas

1. Novo SIDRA permite consultar facilmente dados de estudos e pesquisas do IBGE também em dispositivos móveis | Agência de Notícias, acessado em novembro 12, 2025, https://agenciadenoticias.ibge.gov.br/agencia-sala-de-imprensa/2013-agencia-de-noticias/releases/9481-novo-sidra-permite-consultar-facilmente-dados-de-estudos-e-pesquisas-do-ibge-tambem-em-dispositivos-moveis
2. New SIDRA also allows easy access to IBGE´s studies and surveys through mobile devices, acessado em novembro 12, 2025, https://agenciadenoticias.ibge.gov.br/en/agencia-press-room/2185-news-agency/releases-en/10253-new-sidra-also-allows-easy-access-to-ibge-s-studies-and-surveys-through-mobile-devices
3. DADOS ABERTOS DO IBGE: RECUPERAÇÃO NA API DE DADOS AGREGADOS - Portal de Periódicos UFSC, acessado em novembro 12, 2025, https://periodicos.ufsc.br/index.php/eb/article/download/96185/54942/374888
4. AlanTaranti/sidrapy: A library that provides a python interface for the IBGE SIDRA API., acessado em novembro 12, 2025, https://github.com/AlanTaranti/sidrapy
5. DadosAbertosBrasil.ibge — Documentação Dados Abertos Brasil ..., acessado em novembro 12, 2025, https://www.gustavofurtado.com/dab/DadosAbertosBrasil.ibge.html
6. SSLError when accessing SIDRA (IBGE) API using Python - [SSL:

UNSAFE_LEGACY_RENEGOTIATION_DISABLED - Stack Overflow, acessado em novembro 12, 2025, https://stackoverflow.com/questions/76907660/sslerror-when-accessing-sidra-ibge-api-using-python-ssl-unsafe-legacy-rene
7. Obtaining data from IBGE - PySUS documentation! - Read the Docs, acessado em novembro 12, 2025, https://pysus.readthedocs.io/pt/latest/tutorials/IBGE_data.html
8. Plotando mapas utilizando json, API do IBGE, Python e Folium - Fabio Benevides - Medium, acessado em novembro 12, 2025, https://fabiobenevides.medium.com/plotando-arquivo-json-em-mapas-utilizando-api-do-ibge-python-e-folium-925ce77ede44
9. Coletando dados do SIDRA com o Python - Análise Macro, acessado em novembro 12, 2025, https://analisemacro.com.br/economia/indicadores/coletando-dados-do-sidra-com-o-python/
10. Tabela 7531: Rendimento médio mensal real domiciliar per capita, a preços médios do ano, por classes simples - Poder360, acessado em novembro 12, 2025, https://static.poder360.com.br/2022/04/rendimento-medio-per-capita-domiciliar-20abr2022.pdf
11. Obtaining data from IBGE — PySUS 0.1.13 documentation, acessado em novembro 12, 2025, https://pysus.readthedocs.io/en/stable/tutorials/IBGE_data.html
12. PERIÓDICO PNAD-Contínua, acessado em novembro 12, 2025, https://transparencia.aracaju.se.gov.br/prefeitura/wp-content/uploads/periodicos/Periodico_PNAD-Continua_Aju_2024-2.pdf
13. PERIÓDICO PNAD-Contínua, acessado em novembro 12, 2025, https://transparencia.aracaju.se.gov.br/wp-content/uploads/periodicos/Periodico_PNAD-Continua_Aju_2024-3.pdf
14. PNAD Contínua - Trimestres Móveis, acessado em novembro 12, 2025, https://agenciadenoticias.ibge.gov.br/media/com_mediaibge/arquivos/be597672c878ec0f80592cddecd373c4.pdf
15. Mercado de Trabalho Brasileiro 2º trimestre de 2017, acessado em novembro 12, 2025, https://agenciadenoticias.ibge.gov.br/media/com_mediaibge/arquivos/524d1491b642d52912c467367ddf79eb.pdf
16. Pesquisa Nacional por Amostra de Domicílios Contínua PNAD Contínua, acessado em novembro 12, 2025, https://www2.camara.leg.br/atividade-legislativa/comissoes/comissoes-permanentes/ctasp/apresentacoes-em-eventos/copy_of_IBGE_CimarAzeredo.pdf
17. PNAD Contínua, acessado em novembro 12, 2025, https://agenciadenoticias.ibge.gov.br/media/com_mediaibge/arquivos/49e558eb5b0e3bb0dd9d5801400c4c2d.pdf
18. PNAD Contínua - Trimestres Móveis - Poder360, acessado em novembro 12, 2025, https://static.poder360.com.br/2023/02/pnad-continua-ibge-28fev2023.pdf
19. IBGE divulga o rendimento domiciliar per capita 2024 - Poder360, acessado em

novembro 12, 2025, https://static.poder360.com.br/2025/04/IBGE-relatorio-rendimento-domiciliar-2024.pdf
20. Censo 2022 | IBGE, acessado em novembro 12, 2025, https://www.ibge.gov.br/estatisticas/sociais/trabalho/22827-IBGE-demografico-2022.html?edicao=37225&t=resultados
21. How to Read JSON Files with Pandas? - GeeksforGeeks, acessado em novembro 12, 2025, https://www.geeksforgeeks.org/python/how-to-read-json-files-with-pandas/
22. Python — Get and Process Web API Data through Pandas and ..., acessado em novembro 12, 2025, https://medium.com/analytics-lane/python-get-and-process-web-api-data-through-pandas-and-requests-part-1-32127638b463
23. sidra · GitHub Topics, acessado em novembro 12, 2025, https://github.com/topics/sidra
24. Dominar las API de Python: Una Guía Completa para Construir y Utilizar APIs en Python, acessado em novembro 12, 2025, https://www.datacamp.com/es/tutorial/python-api