

Alt text

## Análise de Repositórios GitHub por Linguagem de Programação

### Introdução

Este projeto tem como objetivo criar um dataset rico e estruturado com informações dos repositórios mais relevantes do GitHub, organizados pelas 10 linguagens de programação mais populares em 2025. Com esses dados, podemos responder perguntas como:

- Qual linguagem tem os projetos mais estrelados?
- Há correlação entre o número de contribuidores e a atividade do repositório?
- Quais licenças são mais comuns em projetos open-source?
- Como a localização geográfica dos donos influencia a popularidade dos repositórios?

O dataset gerado pode ser usado para:

- ✓ Identificação de tendências no desenvolvimento de software
- ✓ Análise de comunidades open-source
- ✓ Tomada de decisões para contribuições ou adoção de tecnologias

```
# !pip install pandas
# !pip install requests
```

### Imports





```
import requests
import pandas as pd
from datetime import datetime, timedelta
import time
import random
import os
```

### Configuração de Múltiplos Tokens GitHub

Este bloco configura um sistema de rotação de tokens para evitar limites da API do GitHub. A rotação aleatória distribui as requisições entre diferentes tokens aumentando

rotação aleatória distribui os requisições entre diferentes tokens, aumentando significativamente o limite de rate da API.


#### Benefícios:

-  Evita bloqueios por rate limit
-  Permite coleta de grandes volumes de dados
-  Distribui carga entre múltiplas chaves de API
-  Fallback automático em caso de token inválido

```
TOKENS = [
    "TOKENS"
]
```

```
def get_headers():
    return {"Authorization": f"token {random.choice(TOKENS)}"}
```

## ✓ Configuração de Parâmetros de Extração

 **Configuração de Parâmetros de Extração** Define as configurações principais para a coleta sistemática de dados: **Linguagens Seleccionadas:** As 10 linguagens mais populares em 2025 **Estratégia de Paginação:** 15 páginas × 100 repositórios = 1.500 repos por linguagem **Total Estimado:** ~15.000 repositórios coletados

#### Critérios de Seleção:

- Repositórios ordenados por popularidade (stars)
- Foco em projetos ativos e relevantes
- Cobertura equilibrada entre diferentes linguagens

```
LANGUAGES = ["Python", "JavaScript", "Java", "C#", "C++", "TypeScript", "Go", "Rust", "Ko
PER_PAGE = 100
PAGES = 15
OUTPUT_FILE = "../dados/github_repos_completos.csv"
```

## ✓ Enriquecimento de Dados do Proprietário

Esta função acessa o perfil completo do dono de cada repositório para extrair informações contextuais importantes:

#### Dados Coletados:

- **Tipo de Conta:** Usuário individual vs Organização

- **Portfólio:** Quantidade de repositórios públicos
- **Localização:** Informação geográfica declarada

### Tratamento de Erros:

- Retry automático em falhas temporárias
- Valores padrão para perfis inacessíveis
- Log detalhado de erros para debugging

```
def get_owner_info(owner_url):
    try:
        response = requests.get(owner_url, headers=get_headers())
        if response.status_code == 200:
            owner_data = response.json()
            return {
                "owner_type": owner_data.get("type", "User"),
                "owner_public_repos": owner_data.get("public_repos", 0),
                "owner_location": owner_data.get("location", None)
            }
    except Exception as e:
        print(f"Erro ao buscar owner: {e}")
    return {
        "owner_type": "User",
        "owner_public_repos": 0,
        "owner_location": None
    }
```

## Sistema Avançado de Estatísticas de Repositório

Implementa coleta robusta de métricas detalhadas com paginação completa e filtros temporais:

### Métricas Coletadas:



#### Engajamento da Comunidade

- **Subscribers:** Usuários que recebem notificações (diferente de stars)
- **Contributors:** Desenvolvedores únicos que contribuíram



#### Atividade de Desenvolvimento

- **Commits Recentes:** Atividade dos últimos 12 meses
- **Issues Fechadas:** Resolução de problemas nos últimos 6 meses
- **Pull Requests:** Total de contribuições externas

## Melhorias Técnicas:

### Paginação Inteligente

- Coleta completa até 50 páginas por métrica
- Detecção automática de fim de dados
- Rate limiting entre páginas

### Filtros Temporais

- Issues: Últimos 6 meses para medir atividade recente
- Commits: Últimos 12 meses via API de participação
- Delay estratégico para evitar bloqueios

```
def get_paginated_count(url, headers, max_pages=50):
    """Função auxiliar para contar itens com paginação completa"""
    total_count = 0
    page = 1

    while page <= max_pages:
        try:
            paginated_url = f"{url}?page={page}&per_page=100"
            response = requests.get(paginated_url, headers=headers)

            if response.status_code == 200:
                data = response.json()
                if not data or len(data) == 0:
                    break
                total_count += len(data)
                page += 1
                time.sleep(0.5) # Delay menor entre páginas
            else:
                break

        except Exception as e:
            print(f"Erro na paginação: {e}")
            break

    return total_count

def get_repo_stats(owner, repo_name):
    stats = {
        "subscribers_count": 0,
        "last_year_commits": 0,
        "contributors": 0,
        "closed_issues": 0,
        "pull_requests": 0
```

```
}

headers = get_headers()

try:
    # CORREÇÃO: Subscribers com paginação completa
    subscribers_url = f"https://api.github.com/repos/{owner}/{repo_name}/subscribers"
    stats["subscribers_count"] = get_paginated_count(subscribers_url, headers)

    # Commits do último ano
    participation = requests.get(
        f"https://api.github.com/repos/{owner}/{repo_name}/stats/participation",
        headers=headers
    )
    if participation.status_code == 200:
        participation_data = participation.json()
        if participation_data and "all" in participation_data:
            stats["last_year_commits"] = sum(participation_data["all"][-52:])

    # CORREÇÃO: Contributors com paginação completa
    contributors_url = f"https://api.github.com/repos/{owner}/{repo_name}/contributor"
    stats["contributors"] = get_paginated_count(contributors_url, headers)

    # CORREÇÃO: Issues fechadas nos últimos 6 meses com paginação
    since_date = (datetime.now() - timedelta(days=180)).isoformat()
    closed_issues_url = f"https://api.github.com/repos/{owner}/{repo_name}/issues"

    # Para issues fechadas, precisamos usar parâmetros específicos
    page = 1
    closed_count = 0
    while page <= 20: # Limite razoável para issues
        try:
            url = f"{closed_issues_url}?state=closed&since={since_date}&page={page}&p"
            response = requests.get(url, headers=headers)
            if response.status_code == 200:
                issues = response.json()
                if not issues:
                    break
                closed_count += len(issues)
                page += 1
                time.sleep(0.5)
            else:
                break
        except:
            break
    stats["closed_issues"] = closed_count

    # CORREÇÃO: Pull Requests com paginação completa
    prs_url = f"https://api.github.com/repos/{owner}/{repo_name}/pulls"
    stats["pull_requests"] = get_paginated_count(prs_url, headers)
```

```
except Exception as e:
    print(f"Erro ao buscar stats para {owner}/{repo_name}: {e}")

return stats
```

## Sistema de Backup e Salvamento Incremental

**NOVA FUNCIONALIDADE CRÍTICA:** Sistema robusto de persistência de dados que previne perda de informações durante coletas longas.

### Características Principais:

#### Proteção Contra Perda de Dados

- **Backup Automático:** Arquivos existentes são preservados
- **Salvamento por Linguagem:** Dados salvos após cada linguagem processada
- **Modo Append:** Adiciona dados sem sobrescrever





#### Estrutura de Arquivos

- **Arquivo Principal:** github\_repos\_completos.csv
- **Backups Temporais:** backup\_YYYYMMDD\_HHMMSS\_arquivo.csv
- **Backups por Linguagem:** backup\_python\_repos.csv, backup\_javascript\_repos.csv

#### Recuperação Inteligente

- Detecção de coletas parciais interrompidas
- Continuação automática de onde parou
- Validação de integridade dos dados

#### Casos de Uso:

-  Coleta interrompida por falha de rede
-  Rate limit atingido em linguagens específicas
-  Interrupção manual do usuário
-  Falhas de sistema durante processamento

```
def save_data_incremental(df, language, is_first_language=False):
    """Salva dados incrementalmente após cada linguagem"""
    try:
        if is_first_language and os.path.exists(OUTPUT_FILE):
            # Backup do arquivo existente
            backup_name = f"backup {datetime.now().strftime('%Y%m%d %H%M%S')} {OUTPUT_FILE}"
```

```

os.rename(OUTPUT_FILE, backup_name)
print(f"Backup criado: {backup_name}")

# Salva ou adiciona ao arquivo
if not os.path.exists(OUTPUT_FILE) or is_first_language:
    df.to_csv(OUTPUT_FILE, index=False, encoding='utf-8', mode='w')
    print(f"Arquivo criado: {OUTPUT_FILE}")
else:
    df.to_csv(OUTPUT_FILE, index=False, encoding='utf-8', mode='a', header=False)
    print(f"Dados adicionados ao arquivo: {OUTPUT_FILE}")

# Salva backup específico da linguagem
lang_file = f"backup_{language.lower()}_repos.csv"
df.to_csv(lang_file, index=False, encoding='utf-8')
print(f"Backup da linguagem salvo: {lang_file}")

except Exception as e:
    print(f"Erro ao salvar dados: {e}")

def load_existing_data():
    """Carrega dados existentes se houver"""
    if os.path.exists(OUTPUT_FILE):
        try:
            return pd.read_csv(OUTPUT_FILE)
        except:
            return pd.DataFrame()
    return pd.DataFrame()

```

## Motor de Coleta Principal por Linguagem

Função principal otimizada para coleta eficiente e resiliente de dados do GitHub.

### Recursos Avançados:

#### Sistema de Retry Inteligente

- **3 Tentativas** por página com backoff exponencial
- **Detecção de Rate Limit:** Pausa automática de 2 minutos
- **Tratamento de Erros 422:** Detecção de páginas inexistentes

#### Monitoramento em Tempo Real

- Progress tracking detalhado por página e repositório
- Estatísticas de sucesso/falha por linguagem
- Estimativa de tempo restante

Estimativa de tempo restante

## ⚡ Otimizações de Performance

- **Delays Estratégicos:** 2s entre repos, 5s entre páginas
- **Rotação de Tokens:** Distribuição de carga
- **Paralelização Limitada:** Evita sobrecarga da API

## Tratamento de Casos Especiais:

- **Rate Limit (403):** Pausa e retry automático
- **Páginas Vazias:** Detecção de fim de resultados
- **Repositórios Privados:** Skip automático
- **Dados Incompletos:** Preenchimento com valores padrão

```
def fetch_repos_by_language(language, retry_count=3):
    all_repos = []

    print(f"\n🔍 Iniciando coleta para {language}...")

    for page in range(1, PAGES + 1):
        for attempt in range(retry_count):
            try:
                url = f"https://api.github.com/search/repositories?q=language:{language}&"
                response = requests.get(url, headers=get_headers())

                if response.status_code == 403:
                    print(f"Rate limit excedido na página {page}, aguardando...")
                    time.sleep(120) # Espera 2 minutos
                    continue

                if response.status_code == 422:
                    print(f"Página {page} muito alta, parando coleta para {language}")
                    break

                response.raise_for_status()

                repos_data = response.json()
                if "items" not in repos_data:
                    print(f"Sem repositórios na página {page}")
                    break

                print(f"📄 Processando página {page}/{PAGES} ({len(repos_data['items'])} itens)")

                for i, repo in enumerate(repos_data["items"]):
                    print(f"🌐 Processando repo {i+1}/{len(repos_data['items'])}: {repo['name']}")

                    repo_info = {
```



```

        "name": repo["name"],
        "owner": repo["owner"]["login"],
        "stars": repo["stargazers_count"],
        "forks": repo["forks_count"],
        "language": repo["language"],
        "created_at": repo["created_at"],
        "updated_at": repo["updated_at"],
        "size_kb": repo["size"],
        "watchers_count": repo["watchers_count"],
        "open_issues": repo["open_issues_count"]
    }

    # Buscar informações do owner
    owner_info = get_owner_info(repo["owner"]["url"])
    repo_info.update(owner_info)

    # Buscar estatísticas detalhadas (VERSÃO CORRIGIDA)
    stats = get_repo_stats(repo["owner"]["login"], repo["name"])
    repo_info.update(stats)

    all_repos.append(repo_info)

    time.sleep(2) # Delay entre repos

break # Sai do retry loop se deu certo

except Exception as e:
    print(f"Erro na página {page}, tentativa {attempt + 1}: {e}")
    if attempt == retry_count - 1:
        print(f"Falha definitiva na página {page}")
    else:
        time.sleep(30) # Aguarda antes de tentar novamente
        continue

time.sleep(5) # Delay entre páginas

return pd.DataFrame(all_repos)

```

## Orquestrador Principal de Execução

Sistema principal que coordena toda a operação de coleta com recursos enterprise-level.

### Funcionalidades Principais:



#### Planejamento Inteligente

- **Deteção de Estado:** Identifica coletas parciais existentes

- **Continuação Automática:** Processa apenas linguagens pendentes
- **Estimativas Precisas:** Cálculo de tempo e volume total

## Gestão de Interrupções

- **Ctrl+C Graceful:** Salvamento seguro ao interromper
- **Recovery Automático:** Retoma de onde parou
- **Checkpoint System:** Estados intermediários preservados

## Monitoramento Avançado

- **Estatísticas em Tempo Real:** Progresso por linguagem
- **Métricas de Performance:** Taxa de sucesso, tempo médio
- **Alertas de Problemas:** Rate limits, falhas de conexão

## Relatórios Finais

- **Distribuição por Linguagem:** Contagem final por tecnologia
- **Métricas de Arquivo:** Tamanho, localização, integridade
- **Estatísticas de Coleta:** Sucessos, falhas, tempo total

```
def main():
    print("🚀 Iniciando coleta de dados do GitHub...")
    print(f"📄 Linguagens: {' '.join(LANGUAGES)}")
    print(f"📄 Páginas por linguagem: {PAGES}")
    print(f"📄 Repos por página: {PER_PAGE}")
    print(f"🎯 Total estimado: {len(LANGUAGES) * PAGES * PER_PAGE} repositórios")

    # Verifica se já existem dados
    existing_data = load_existing_data()
    if not existing_data.empty:
        print(f"📁 Encontrados {len(existing_data)} repositórios existentes")
        processed_languages = existing_data['language'].unique().tolist()
        remaining_languages = [lang for lang in LANGUAGES if lang not in processed_languages]
        if remaining_languages:
            print(f"🔄 Continuando com: {' '.join(remaining_languages)}")
            languages_to_process = remaining_languages
        else:
            print("✅ Todas as linguagens já foram processadas!")
            return
    else:
        languages_to_process = LANGUAGES

    total_collected = len(existing_data)

    for i, lang in enumerate(languages_to_process):
        print(f"\n{'='*60}")
```

```

print(f"🌐 LINGUAGEM {i+1}/{len(languages_to_process)}: {lang}")
print(f"{' '*60}")

try:
    df = fetch_repos_by_language(lang)

    if not df.empty:
        # Determina se é a primeira linguagem NOVA sendo processada
        is_first_new = (i == 0 and existing_data.empty)

        save_data_incremental(df, lang, is_first_new)
        total_collected += len(df)

        print(f"✅ {len(df)} repositórios de {lang} coletados!")
        print(f"📊 Total acumulado: {total_collected} repositórios")
    else:
        print(f"⚠️ Nenhum repositório coletado para {lang}")

    # Delay entre linguagens
    if i < len(languages_to_process) - 1:
        print(f"⌚ Aguardando 5 minutos antes da próxima linguagem...")
        time.sleep(300)

except KeyboardInterrupt:
    print(f"\n⚠️ Interrompido pelo usuário na linguagem {lang}")
    print(f"📊 Dados salvos até agora: {total_collected} repositórios")
    break

except Exception as e:
    print(f"❌ Erro crítico na linguagem {lang}: {e}")
    continue

# Carrega dados finais
final_data = load_existing_data()
if not final_data.empty:
    print(f"\n🎉 COLETA FINALIZADA!")
    print(f"📊 Total final: {len(final_data)} repositórios")
    print(f"📁 Arquivo: {OUTPUT_FILE}")
    print(f"💾 Tamanho: {os.path.getsize(OUTPUT_FILE) / 1024 / 1024:.2f} MB")

    # Estatísticas por linguagem
    print(f"\n📈 Distribuição por linguagem:")
    lang_stats = final_data['language'].value_counts()
    for lang, count in lang_stats.items():
        print(f"    {lang}: {count} repositórios")
    else:
        print(f"❌ Nenhum dado foi coletado.")

if __name__ == "__main__":
    main()

```



## Dicionário Completo do Dataset

### Estrutura Detalhada dos Dados Coletados



#### Informações Básicas do Repositório

Campo	Tipo	Descrição	Exemplo
name	String	Nome único do repositório	"tensorflow"
owner	String	Username do proprietário	"google"
language	String	Linguagem principal detectada	"Python"



#### Métricas de Popularidade e Engajamento

Campo	Tipo	Descrição	Exemplo
stars	Integer	Favoritos dos usuários	175000
forks	Integer	Cópias do repositório	85000
watchers_count	Integer	Usuários observando mudanças	3200
subscribers_count	Integer	Inscritos em notificações	1500



#### Indicadores de Atividade de Desenvolvimento

Campo	Tipo	Descrição	Período	Exemplo
open_issues	Integer	Issues não resolvidas	Atual	42
closed_issues	Integer	Issues resolvidas	6 meses	128
pull_requests	Integer	Total de PRs	Histórico	75
last_year_commits	Integer	Commits realizados	12 meses	890
contributors	Integer	Desenvolvedores únicos	Histórico	35



#### Metadados Temporais

Campo	Tipo	Descrição	Formato	Exemplo
created_at	DateTime	Data de criação	ISO 8601 UTC	2015-11-09T23:25:38Z
updated_at	DateTime	Última modificação	ISO 8601 UTC	2024-03-15T08:12:45Z
size_kb	Integer	Tamanho do repositório	Kilobytes	10240



#### Perfil do Proprietário

Campo	Tipo	Descrição	Valores Possíveis	Exemplo
owner_type	String	Tipo de conta	User, Organization	"Organization"
owner_public_repos	Integer	Repositórios públicos	0 - ∞	250
owner_location	String	Localização declarada	Texto livre	"Mountain View, CA"

# Casos de Uso e Aplicações dos Dados

## Análises Possíveis com o Dataset:

### Análise de Tendências Tecnológicas

- **Popularidade por Linguagem:** Ranking de stars e forks
- **Crescimento Temporal:** Evolução de projetos ao longo dos anos
- **Ciclo de Vida:** Correlação entre idade e atividade

### Geografia do Código Aberto

- **Mapeamento Global:** Distribuição de desenvolvedores por país
- **Hubs de Inovação:** Identificação de centros tecnológicos
- **Colaboração Internacional:** Projetos multi-regionais

### Dinâmica de Comunidades

- **Padrões de Contribuição:** Relação contributors vs. atividade
- **Gestão de Issues:** Eficiência na resolução de problemas
- **Sustentabilidade:** Projetos com atividade contínua

### Análise Organizacional






- **Usuários vs. Organizações:** Diferentes padrões de desenvolvimento
- **Portfólio Analysis:** Organizações com múltiplos projetos populares
- **Estratégias Open Source:** Abordagens empresariais

## Repositório e Recursos Adicionais

### Acesso ao Código Completo

**GitHub Repository:** <https://github.com/LucasjsSilva/data-set-repositorios>

### Recursos Disponíveis:

-  Código fonte completo e documentado
-  Scripts de análise de dados
-  Datasets de exemplo
-  Jupyter notebooks com visualizações
-  Guias de instalação e configuração

## Como Contribuir:

1. **Fork** do repositório principal
2. **Clone** para desenvolvimento local
3. **Issues** para reportar bugs ou sugerir melhorias
4. **Pull Requests** com novas funcionalidades

## ✓ Construção do Dicionário de Dados

### Objetivo

Criar uma documentação estruturada e abrangente do dataset coletado, estabelecendo definições claras de cada atributo e suas características técnicas.

```
import pandas as pd
```

## ✓ Carregamento e Visão Geral do Dataset

Primeira etapa fundamental: carregar os dados e obter uma visão panorâmica da estrutura do dataset.

### Métricas Iniciais Capturadas:

- ✓ **Dimensões do Dataset:** Quantidade de registros × colunas
- ✓ **Preview dos Dados:** Primeiras linhas para validação
- ✓ **Estrutura Geral:** Organização e formato dos dados

```
# Import the dataset into a pandas DataFrame
df = pd.read_csv('../dados/github_repos_completos.csv')
```

```
# Display dataset information
print('Dataset de Repositórios do GitHub contendo:')
print(f'{df.shape[0]} registros')
print(f'{df.shape[1]} colunas\n')
```

```
df.head()
```

↔ Dataset de Repositórios do GitHub contendo:  
9450 registros  
18 colunas

	name	owner	stars	forks	language	created_at	
0	free-programming-books	EbookFoundation	359735	63576	Python	2013-10-11T06:50:37Z	2025-06
1	public-apis	public-apis	351991	37004	Python	2016-03-20T23:49:42Z	2025-06
2	system-design-primer	donnemartin	306925	50727	Python	2017-02-26T16:15:28Z	2025-06

	design primer						
3	awesome-python	vinta	247255	25843	Python	2014-06-27T21:00:06Z	2025-06
4	Python	TheAlgorithms	201541	46909	Python	2016-07-16T09:44:01Z	2025-06

## ✓ 🔍 Análise de Tipos de Dados

Investigação detalhada dos tipos de dados presentes no dataset para classificação adequada.

### Tipos Identificados:

- **int64**: Valores numéricos inteiros (stars, forks, contributors)
- **object**: Strings e dados categóricos (name, owner, language)
- **datetime**: Datas e timestamps (created\_at, updated\_at)

```
# Display the data types of each column in the DataFrame
df.dtypes
```

```
name                object
owner               object
stars               int64
forks               int64
language            object
created_at          object
updated_at          object
size_kb             int64
watchers_count      int64
open_issues         int64
owner_type          object
owner_public_repos  int64
owner_location      object
subscribers_count   int64
last_year_commits   int64
contributors        int64
closed_issues       int64
pull_requests       int64
dtype: object
```

```
# Create a DataFrame to serve as a data dictionary, showing each attribute's name and its
dict_df = pd.DataFrame({
    'Nome do Atributo': df.dtypes.index,
    'Tipo do Valor': df.dtypes.values.astype(str),
})
```

```
dict_df
```



	Nome do Atributo	Tipo do Valor
0	name	object
1	owner	object
2	stars	int64
3	forks	int64
4	language	object
5	created_at	object
6	updated_at	object
7	size_kb	int64
8	watchers_count	int64
9	open_issues	int64
10	owner_type	object
11	owner_public_repos	int64
12	owner_location	object
13	subscribers_count	int64
14	last_year_commits	int64
15	contributors	int64
16	closed_issues	int64
17	pull_requests	int64

## ✓ Construção do Dicionário Estruturado

Desenvolvimento de um dicionário de dados completo e profissional seguindo padrões de documentação.

Estrutura do Dicionário:

Campo	Descrição
Nome do Atributo	Identificador único da coluna
Tipo do Valor	Tipo de dado técnico (int64, string, datetime)
Tipo do Formato	Classificação semântica (Numérico/Categórico)
Descrição do Atributo	Explicação detalhada do significado e uso

```
# Helper function to classify a data type as 'Numérico' (Numeric) or 'Categórico' (Catego
def classify_format(dtype):
```

```
def classify_format(dtype):  
    if dtype == 'int64' or dtype == 'float64':  
        return 'Numérico'  
    else:  
        return 'Categórico'
```

## ✓ Sistema de Classificação Inteligente

Implementação de função para classificação automática dos tipos de formato baseada em regras lógicas.

### Regras de Classificação:

- **Numérico:** int64, float64 → Dados quantitativos para cálculos estatísticos
- **Categórico:** object, string → Dados qualitativos para agrupamentos

```
# Apply the classification function to determine if each attribute is 'Numérico' or 'Cate  
dict_df['Tipo do Formato'] = dict_df['Tipo do Valor'].apply(classify_format)
```

## ✓ Catálogo de Descrições Detalhadas

Documentação semântica completa de cada atributo do dataset:

### Métricas de Popularidade

- **stars** : Indicador de aprovação da comunidade
- **forks** : Medida de interesse em contribuição
- **subscribers\_count** : Engajamento ativo com notificações

### Indicadores de Atividade

- **last\_year\_commits** : Intensidade de desenvolvimento recente
- **contributors** : Diversidade da base de colaboradores
- **closed\_issues** : Capacidade de resolução de problemas

### Informações Organizacionais

- **owner\_type** : Natureza do proprietário (Individual/Corporativo)
- **owner\_public\_repos** : Portfólio e experiência do proprietário
- **owner\_location** : Contexto geográfico do desenvolvimento

```
# Add descriptions for each attribute to the data dictionary.  
descricoes = 「
```

```
----- L
    "Nome do repositório",
    "Login do usuário/organização dono",
    "Número de estrelas",
    "Número de forks",
    "Linguagem principal do projeto",
    "Data de criação do repositório (UTC)",
    "Data da última atualização",
    "Tamanho aproximado do repositório em KB",
    "Usuários acompanhando o repositório",
    "Issues abertas no momento",
    "Tipo do dono (User ou Organization)",
    "Quantidade de repositórios públicos do dono",
    "Localização geográfica declarada no perfil (opcional)",
    "Inscritos no repositório",
    "Quantidade de commits realizados nos últimos 12 meses",
    "Número de contribuidores únicos",
    "Issues fechadas nos últimos 6 meses",
    "Quantidade total de pull requests (abertos + fechados)"
]

dict_df['Descrição do Atributo'] = descricoes
```

## ✕ 🔧 Refinamento e Padronização de Tipos

Processo de limpeza e padronização dos tipos de dados para consistência.

### Melhorias Aplicadas:

- ✓ **Substituição de object por string** para clareza
- ✓ **Identificação explícita de datetime** para colunas temporais
- ✓ **Validação de integridade** dos tipos atribuídos

```
# Refine 'Tipo do Valor' for clarity and accuracy in the data dictionary.
# 'object' is replaced with 'string', and specific date columns are set to 'datetime'.

dict_df['Tipo do Valor'] = dict_df['Tipo do Valor'].replace('object', 'string')
dict_df.loc[dict_df['Nome do Atributo'].isin(['created_at', 'updated_at']), 'Tipo do Valo

dict_df
```

	Nome do Atributo	Tipo do Valor	Tipo do Formato	Descrição do Atributo
0	name	string	Categórico	Nome do repositório
1	owner	string	Categórico	Login do usuário/organização dono
2	stars	int64	Número	Número de estrelas

2	stars	int64	Número	numero de estrelas
3	forks	int64	Numérico	Número de forks
4	language	string	Categórico	Linguagem principal do projeto
5	created_at	datetime	Categórico	Data de criação do repositório (UTC)
6	updated_at	datetime	Categórico	Data da última atualização
7	size_kb	int64	Numérico	Tamanho aproximado do repositório em KB
8	watchers_count	int64	Numérico	Usuários acompanhando o repositório
9	open_issues	int64	Numérico	Issues abertas no momento
10	owner_type	string	Categórico	Tipo do dono (User ou Organization)
11	owner_public_repos	int64	Numérico	Quantidade de repositórios públicos do dono
12	owner_location	string	Categórico	Localização geográfica declarada no perfil (op...
13	subscribers_count	int64	Numérico	Inscritos no repositório
14	last_year_commits	int64	Numérico	Quantidade de commits realizados nos últimos 1...
15	contributors	int64	Numérico	Número de contribuidores únicos

## ▼ Exportação do Dicionário

Persistência do dicionário de dados em formato CSV para reutilização e documentação.

**Arquivo Gerado:** req2.csv **Utilização:** Documentação técnica, onboarding de equipe, validação de dados

```
# Save the created data dictionary DataFrame to a CSV file.
dict_df.to_csv('../dados/req2.csv', index=False)
```



## ✓ 🖌️ Processo de Limpeza e Qualidade dos Dados

### 🎯 Objetivo

Estabelecer um dataset confiável e consistente através de técnicas rigorosas de limpeza e validação de qualidade.

```
import pandas as pd
```

```
# Import the dataset into a pandas DataFrame  
df = pd.read_csv('../dados/github_repos_completos.csv')
```

```
df.head()
```



	name	owner	stars	forks	language	created_at	
0	free-programming-books	EbookFoundation	359735	63576	Python	2013-10-11T06:50:37Z	2025-06
1	public-apis	public-apis	351991	37004	Python	2016-03-20T23:49:42Z	2025-06
2	system-design-primer	donnemartin	306925	50727	Python	2017-02-26T16:15:28Z	2025-06
3	awesome-python	vinta	247255	25843	Python	2014-06-27T21:00:06Z	2025-06
4	Python	TheAlgorithms	201541	46909	Python	2016-07-16T09:44:01Z	2025-06

## ✓ ❌ Tratamento de Dados Ausentes (Missing Data)

### Diagnóstico de Ausências

Identificação e quantificação sistemática de valores faltantes em todas as colunas.

#### Estratégias de Tratamento:

- **language** : Exclusão de registros (dados críticos para análise)
- **owner\_location** : Imputação com "Not informed" (dado opcional)

#### Justificativas Técnicas:

- **Linguagem é fundamental** para categorização e análise comparativa
- **Localização é complementar** e pode ser tratada como "não informado"

```
# Check for and sum the number of missing (null) values for each column.
df.isnull().sum()
```

```

name                0
owner               0
stars              0
forks              0
language            1
created_at         0
updated_at         0
size_kb            0
watchers_count     0
open_issues        0
owner_type         0
owner_public_repos 0
owner_location     3969
subscribers_count  0
last_year_commits  0
contributors       0
closed_issues      0
pull_requests      0
dtype: int64
```

```
# Calculate and print the proportion of null values for 'language' and 'owner_location' c
language_null_prop = (df['language'].isnull().sum() / len(df)) * 100
location_null_prop = (df['owner_location'].isnull().sum() / len(df)) * 100
```

```
print('Proporção de nulos:')
print(f'Atributo language: {language_null_prop:.2f}%')
print(f'Atributo owner_location: {location_null_prop:.2f}%')
```

```

Proporção de nulos:
Atributo language: 0.01%
Atributo owner_location: 42.00%
```

```
# Handle missing values:
```

```
df = df[df['language'].notnull()]
```

```
df['owner_location'].fillna('Not informed', inplace=True)
```

```

/tmp/ipykernel_2576/474970543.py:4: FutureWarning: A value is trying to be set on a c
The behavior will change in pandas 3.0. This inplace method will never work because t
```

```
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({
```

```
df['owner_location'].fillna('Not informed', inplace=True)
```

```
df.isnull().sum()
```

```
name          0
owner         0
stars         0
forks         0
language      0
created_at    0
updated_at    0
size_kb       0
watchers_count 0
open_issues   0
owner_type    0
owner_public_repos 0
owner_location 0
subscribers_count 0
last_year_commits 0
contributors  0
closed_issues  0
pull_requests 0
dtype: int64
```

## ▼ Eliminação de Duplicatas

### Detecção de Redundâncias

Identificação de registros completamente idênticos que podem distorcer análises estatísticas.

#### Processo:

1. **Identificação:** Localização de linhas duplicadas
2. **Análise:** Distribuição por linguagem para entender padrões
3. **Remoção:** Eliminação mantendo apenas registros únicos

**Impacto:** Redução do viés estatístico e melhoria na qualidade das análises

```
# Identify and count completely duplicate rows in the DataFrame.
```

```
df_duplicated = df[df.duplicated()]
```

```
print(f"Quantidade de duplicatas completas: {df_duplicated.shape[0]}")
```

```
Quantidade de duplicatas completas: 348
```

```
df_duplicated['language'].value_counts()
```

```
language
C          347
Java        1
```



```
Name: count, dtype: int64
```

```
# Remove all identified duplicate rows from the DataFrame.
```

```
df = df.drop_duplicates()
```

```
print(f'Após exclusão de dados duplicados: {len(df)} registros')
```

```
Após exclusão de dados duplicados: 9101 registros
```

## ✓ 🔍 Validação de Consistência de Dados

### Verificação de Valores Negativos

Auditoria de colunas numéricas para identificar valores fisicamente impossíveis ou inconsistentes.

#### Colunas Verificadas:

- Métricas de popularidade (stars, forks, subscribers)
- Indicadores de atividade (commits, contributors, issues)
- Dados temporais e de tamanho

### Validação de Formatos de Data

Conversão e validação de colunas temporais com tratamento de erros.

#### Processo:

- Conversão para `datetime` com `errors='coerce'`
- Identificação de valores não convertíveis
- Relatório de qualidade da conversão

### Auditoria de Dados Numéricos

Verificação de valores não numéricos em colunas esperadamente numéricas.

```
# Identify and list all numeric columns.
```

```
numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
```

```
# Iterate through numeric columns to check for and display negative values.
```

```
for column in numeric_columns:
```

```
    negativos = df[df[column] < 0]
```

```
    qtd = negativos.shape[0]
```

```
    if qtd > 0:
```

```
        print(f"\nColuna '{column}' tem {qtd} valores negativos:")
```

```
        print(negativos[[column]].head(10))
```

```

else:
    print(f"Coluna '{column}' não possui valores negativos.")

Coluna 'stars' não possui valores negativos.
Coluna 'forks' não possui valores negativos.
Coluna 'size_kb' não possui valores negativos.
Coluna 'watchers_count' não possui valores negativos.
Coluna 'open_issues' não possui valores negativos.
Coluna 'owner_public_repos' não possui valores negativos.
Coluna 'subscribers_count' não possui valores negativos.
Coluna 'last_year_commits' não possui valores negativos.
Coluna 'contributors' não possui valores negativos.
Coluna 'closed_issues' não possui valores negativos.
Coluna 'pull_requests' não possui valores negativos.

# Ensuring date columns are in datetime format
for col in ['created_at', 'updated_at']:
    df[col] = pd.to_datetime(df[col], errors='coerce')

n_nulos = df[col].isna().sum()
print(f"Coluna '{col}' após conversão para datetime tem {n_nulos} valores que não for

Coluna 'created_at' após conversão para datetime tem 0 valores que não foram converti
Coluna 'updated_at' após conversão para datetime tem 0 valores que não foram converti

# Checking for non-numeric entries in numeric columns
for col in numeric_columns:
    coerced = pd.to_numeric(df[col], errors='coerce')
    n_invalidos = coerced.isna().sum()
    print(f"Coluna '{col}' tem {n_invalidos} valores não numéricos")

Coluna 'stars' tem 0 valores não numéricos (NaN após conversão)
Coluna 'forks' tem 0 valores não numéricos (NaN após conversão)
Coluna 'size_kb' tem 0 valores não numéricos (NaN após conversão)
Coluna 'watchers_count' tem 0 valores não numéricos (NaN após conversão)
Coluna 'open_issues' tem 0 valores não numéricos (NaN após conversão)
Coluna 'owner_public_repos' tem 0 valores não numéricos (NaN após conversão)
Coluna 'subscribers_count' tem 0 valores não numéricos (NaN após conversão)
Coluna 'last_year_commits' tem 0 valores não numéricos (NaN após conversão)
Coluna 'contributors' tem 0 valores não numéricos (NaN após conversão)
Coluna 'closed_issues' tem 0 valores não numéricos (NaN após conversão)
Coluna 'pull_requests' tem 0 valores não numéricos (NaN após conversão)

```

## ▼ Análise de Dados Categóricos

### Padronização e Normalização

Investigação de variações e inconsistências em dados categóricos.

**Técnicas Aplicadas:**

**Técnicas Aplicadas.**

- **Normalização:** Conversão para lowercase e remoção de espaços
- **Contagem de Frequências:** Identificação de variações e duplicatas semânticas
- **Deteção de Padrões:** Análise de consistência em categorias

## Tratamento Especial de Localizações

Exportação detalhada de dados de localização para análise geográfica posterior.

```
# Identify categorical columns.
categorical_columns = df.select_dtypes(include=['object']).columns.tolist()

# For each categorical column, print unique values and their normalized counts.
for col in categorical_columns:
    print(f"\nValores únicos e suas contagens na coluna '{col}':")
    valores_unicos = df[col].dropna().unique()

    # Normalizing to lower case and removing spaces to identify variations.
    valores_normalizados = [str(v).strip().lower() for v in valores_unicos]
    contagem = pd.Series(valores_normalizados).value_counts()

    print(contagem.head(20))
```

Valores únicos e suas contagens na coluna 'name':

leetcode	3
tv	2
pulse	2
cameraview	2
icecream	2
gifski	2
menu	2
eureka	2
cardslider	2
java	2
schedule	2
ignite	2
time	2
surge	2
glance	2
nuklear	2
iris	2
shadow	2
skip	2
ios	2

Name: count, dtype: int64

Valores únicos e suas contagens na coluna 'owner':

stasel	1
ebookfoundation	1
public-apis	1

```

donnemartin      1
vinta            1
thealgorithms    1
significant-gravitas  1
automatic1111    1
cocoapods        1
hearthsim        1
xjbeta           1
slazyk           1
marioiannotta    1
radex            1
kaandedeoglu     1
alexeybelezeko   1
yeahdongcn       1
venmo            1
jiritrecak       1
pixel16          1
Name: count, dtype: int64

```

Valores únicos e suas contagens na coluna 'language':

```

python          1
javascript       1
java            1
c               1
typescript      1
go              1
rust            1
kotlin          1
swift           1
Name: count, dtype: int64

```

```
unique_locations = df['owner_location'].dropna().unique()
```

```

unique_location_normalized = [str(v).strip().lower() for v in unique_locations]
contagem_df = pd.Series(unique_location_normalized).value_counts().reset_index()
contagem_df.columns = ['location', 'count']

```

```
contagem_df.to_csv('../dados/location_counts.csv', index=False)
```

## ▼ 🗑️ Remoção de Colunas Redundantes

### Otimização do Dataset

Eliminação de colunas com informações duplicadas ou baixo valor analítico.

#### Colunas Removidas:

- **owner\_location**: Alta cardinalidade e muitos valores faltantes
- **watchers\_count**: Redundante com outras métricas de engajamento

**Justificativa:** Simplificação do dataset mantendo informações essenciais

```
df = df.drop(['owner_location'], axis=1)
df = df.drop(['watchers_count'], axis=1)
df
```

	name	owner	stars	forks	language	created_a
0	free-programming-books	EbookFoundation	359735	63576	Python	2013-10-1 06:50:37+00:0
1	public-apis	public-apis	351991	37004	Python	2016-03-2 23:49:42+00:0
2	system-design-primer	donnemartin	306925	50727	Python	2017-02-2 16:15:28+00:0
3	awesome-python	vinta	247255	25843	Python	2014-06-2 21:00:06+00:0
4	Python	TheAlgorithms	201541	46909	Python	2016-07-1 09:44:01+00:0
...	...	...	...	...	...	.
9445	YNSearch	younatics	1194	101	Swift	2017-04-1 05:59:12+00:0
9446	ReactKit	ReactKit	1194	40	Swift	2014-09-2 14:09:19+00:0
9447	CardSlider	saoudrizwan	1191	93	Swift	2017-02-2 21:05:55+00:0
9448	VisualProgrammingLanguage	NathanFlurry	1190	45	Swift	2018-04-0 13:29:12+00:0
9449	WebRTC-iOS	stasel	1189	246	Swift	2018-05-2 16:48:59+00:0

9101 rows × 16 columns

## ▼ Persistência do Dataset Limpo

Exportação do dataset processado para análises subsequentes.

**Arquivo Gerado:** github\_repos\_limpo.csv **Melhorias Obtidas:**

- ☒ Eliminação de dados faltantes críticos

- ☒ Remoção de duplicatas
- ☒ Validação de consistência
- ☒ Otimização estrutural

```
df.to_csv('../dados/github_repos_limpo.csv', index=False)
```

## ✓ Análise Exploratória de Dados (EDA)

### Objetivo

Descobrir padrões, tendências e insights através de análise estatística descritiva e visualizações informativas.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Load the cleaned GitHub repositories dataset
df = pd.read_csv('../dados/github_repos_limpo.csv')
df.head(2) # Display the first two rows of the dataset
```



## ✓ Análise de Frequência de Dados Categóricos

### Distribuição de Tipos de Proprietários

Investigação da composição entre usuários individuais e organizações.

#### Insights Esperados:

- Proporção de projetos individuais vs. corporativos
- Padrões de contribuição por tipo de proprietário
- Diferenças em popularidade e escala

### Distribuição de Linguagens de Programação

Análise da representatividade das diferentes tecnologias no dataset.

#### Visualizações:

- Gráfico de barras com frequências absolutas
- Identificação de linguagens dominantes
- Padrões de adoção tecnológica

### Top Organizações Contribuidoras

Identificação das organizações mais ativas no ecossistema open-source.

**Métricas:**

- Ranking por número de repositórios
- Análise de concentração de contribuições
- Mapeamento de players principais

```
df_copy = df.copy()
```

```
# Identify all categorical columns in the dataset
categorical_columns = df.select_dtypes(include=['object']).columns.to_list()
categorical_columns
```

```
['name', 'owner', 'language', 'created_at', 'updated_at', 'owner_type']
```

```
# Display the frequency of each unique value for all categorical columns
for col in categorical_columns:
    print(f'{col}')
    print(df[col].value_counts())
    print('\n')
```

```
# Owner type, owner, and language are the key categorical columns
```

```
name
name
android      10
server        4
goproxy       3
aliyunpan     3
hydra         3
..
aircrack-ng   1
kphp-kdb      1
nanomsg       1
ly            1
HP-Socket     1
Name: count, Length: 7729, dtype: int64
```

```
owner
owner
google       65
apache       54
microsoft    46
alibaba      31
JetBrains    21
..
AlexeyBelezeko 1
kaandedeoglu  1
radex         1
MarioIannotta  1
```



```
slazyk          1
Name: count, Length: 6200, dtype: int64
```

```
language
language
Kotlin      983
Swift       981
C           955
Rust        928
Java        909
Go          875
TypeScript  770
JavaScript  762
Python      721
Name: count, dtype: int64
```

```
created_at
created_at
2018-05-21 16:48:59+00:00    1
2024-02-22 20:53:54+00:00    1
2021-04-12 15:18:15+00:00    1
2024-01-29 05:30:33+00:00    1
2018-06-29 21:59:26+00:00    1
..
2017-05-05 07:28:13+00:00    1
2023-03-08 14:53:43+00:00    1
2025-02-19 18:03:06+00:00    1
2017-09-07 04:53:45+00:00    1
```

```
# Plot the distribution of the owner types (User vs. Organization)
df['owner_type'].value_counts().plot(kind='bar', color='skyblue')
plt.title('Owner Type Distribution')
plt.xlabel('Owner Type')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
# Plot the distribution of programming languages used in the repositories
df['language'].value_counts().plot(kind='bar', color='pink')
plt.title('Language Distribution')
plt.xlabel('Language')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
# Filter the owners where the owner type is 'Organization'
organizations = df[df['owner_type'] == 'Organization']['owner']
organizations.value_counts()

owner
google          65
apache          54
microsoft       46
alibaba         31
JetBrains       21
..
ProxymanApp      1
SwiftJSON        1
PopcornTimeTV    1
imaginary-cloud  1
ProfileCreator   1
Name: count, Length: 3065, dtype: int64

# Plot the top 15 most frequent organization owners
plt.figure(figsize=(10, 6))

organizations.value_counts().head(15).plot(kind='bar', color='cornflowerblue')
plt.title('Top 15 Most Frequent Owners (Organizations)')
plt.xlabel('Owner')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

## ▼ Estatística Descritiva de Dados Numéricos

### Medidas de Tendência Central

Análise de posição para cada variável numérica.

#### Métricas Calculadas:

- **Média:** Valor central considerando todos os dados
- **Mediana:** Valor que divide os dados ao meio (resistente a outliers)

#### Visualizações:

- Histogramas com curva de densidade (KDE)
- Linhas de referência para média e mediana
- Comparação visual entre as medidas

## Medidas de Dispersão

Análise da variabilidade e spread dos dados.

### Estatísticas Computadas:

- **Desvio Padrão:** Dispersão em relação à média
- **Variância:** Quadrado do desvio padrão
- **Amplitude:** Diferença entre máximo e mínimo
- **Quartis e IQR:** Medidas robustas de dispersão

### Visualizações:

- Boxplots para identificação de outliers
- Análise visual da distribuição
- Detecção de assimetrias

```
# Identify all numerical columns in the dataset
numerical_columns = df.select_dtypes(include=['int64']).columns.to_list()
numerical_columns

['stars',
 'forks',
 'size_kb',
 'open_issues',
 'owner_public_repos',
 'subscribers_count',
 'last_year_commits',
 'contributors',
 'closed_issues',
 'pull_requests']
```

## ▼ Posição

```
# For each numerical column, calculate and plot the mean and median
for col in numerical_columns:
    print(f"\n--- {col} ---")
    print(f"Mean: {df[col].mean():.2f}")
    print(f"Median: {df[col].median():.2f}")

    # Plot histogram with KDE for distribution
    sns.histplot(df[col], kde=True)

    # Add mean and median lines
    plt.axvline(df[col].mean(), color='r', linestyle='--', label='Mean')
    plt.axvline(df[col].median(), color='g', linestyle='--', label='Median')
    plt.legend()
```

```
plt.show()
```

## ▼ Dispersão

```
# For each numerical column, calculate dispersion statistics and plot boxplots
for col in numerical_columns:
    print(f"\n--- {col} ---")
    print(f"Standard Deviation: {df[col].std():.2f}")
    print(f"Variance: {df[col].var():.2f}")
    print(f"Minimum: {df[col].min()}")
    print(f"Maximum: {df[col].max()}")
    print(f"Range: {df[col].max() - df[col].min()}")
    print(f"1st Quartile (Q1): {df[col].quantile(0.25)}")
    print(f"3rd Quartile (Q3): {df[col].quantile(0.75)}")
    print(f"IQR (Interquartile Range): {df[col].quantile(0.75) - df[col].quantile(0.25)}")

# Plot boxplot to visualize spread and detect outliers
sns.boxplot(data=df[col])
plt.title(f'Boxplot of {col}')
plt.show()
```

## ▼ 🛠️ Tratamento Inteligente de Outliers

### Metodologia IQR (Interquartile Range)

Implementação de técnica estatística robusta para detecção e tratamento de valores extremos.

#### Processo:



1. **Cálculo dos Limites:**  $Q1 - 1.5 \times IQR$  e  $Q3 + 1.5 \times IQR$
2. **Identificação:** Valores fora dos limites
3. **Análise de Impacto:** Porcentagem de outliers por coluna
4. **Tratamento Seletivo:** Substituição por mediana quando  $\leq 15\%$  dos dados

### Critério de Tratamento

**Regra dos 15%:** Outliers são tratados apenas quando representam  $\leq 15\%$  dos dados, preservando distribuições naturalmente assimétricas.

#### Benefícios:

-  Redução de distorções estatísticas

-  Preservação de padrões naturais
-  Melhoria na qualidade de modelos posteriores

```
import numpy as np

for col in df_copy.select_dtypes(include=['float64', 'int64']).columns:
    Q1 = df_copy[col].quantile(0.25)
    Q3 = df_copy[col].quantile(0.75)
    IQR = Q3 - Q1
    limite_inf = Q1 - 1.5 * IQR
    limite_sup = Q3 + 1.5 * IQR

    outliers = df_copy[(df_copy[col] < limite_inf) | (df_copy[col] > limite_sup)]
    perc_outliers = len(outliers) / len(df_copy) * 100

    print(f"Coluna: {col} - Outliers: {perc_outliers:.2f}%")

    if perc_outliers <= 15:
        mediana = df_copy[col].median()
        df_copy[col] = np.where((df_copy[col] < limite_inf) | (df_copy[col] > limite_sup)
                                , mediana, df_copy[col])
        print(f" -> Substituídos por mediana")

Coluna: stars - Outliers: 8.83%
-> Substituídos por mediana
Coluna: forks - Outliers: 9.32%
-> Substituídos por mediana
Coluna: size_kb - Outliers: 13.21%
-> Substituídos por mediana
Coluna: open_issues - Outliers: 9.72%
-> Substituídos por mediana
Coluna: owner_public_repos - Outliers: 10.65%
-> Substituídos por mediana
Coluna: subscribers_count - Outliers: 8.53%
-> Substituídos por mediana
Coluna: last_year_commits - Outliers: 13.86%
-> Substituídos por mediana
Coluna: contributors - Outliers: 9.22%
-> Substituídos por mediana
Coluna: closed_issues - Outliers: 14.78%
-> Substituídos por mediana
Coluna: pull_requests - Outliers: 10.71%
-> Substituídos por mediana
```

## ▼ Comparação Pré e Pós-Tratamento

### Análise de Impacto

Comparação das distribuições antes e depois do tratamento de outliers.

**Métricas Comparadas:**

### Métricas Comparadas.

- Mudanças nas medidas de tendência central
- Alterações na dispersão
- Impacto visual nas distribuições

## Validação do Tratamento

Verificação da eficácia das técnicas aplicadas através de:

- Boxplots comparativos
- Histogramas antes/depois
- Estatísticas descritivas atualizadas

```
# For each numerical column, calculate and plot the mean and median
for col in numerical_columns:
    print(f"\n--- {col} ---")
    print(f"Mean: {df_copy[col].mean():.2f}")
    print(f"Median: {df_copy[col].median():.2f}")

    # Plot histogram with KDE for distribution
    sns.histplot(df_copy[col], kde=True)

    # Add mean and median lines
    plt.axvline(df_copy[col].mean(), color='r', linestyle='--', label='Mean')
    plt.axvline(df_copy[col].median(), color='g', linestyle='-', label='Median')
    plt.legend()
    plt.show()





# For each numerical column, calculate dispersion statistics and plot boxplots
for col in numerical_columns:
    print(f"\n--- {col} ---")
    print(f"Standard Deviation: {df_copy[col].std():.2f}")
    print(f"Variance: {df_copy[col].var():.2f}")
    print(f"Minimum: {df_copy[col].min()}")
    print(f"Maximum: {df_copy[col].max()}")
    print(f"Range: {df_copy[col].max() - df_copy[col].min()}")
    print(f"1st Quartile (Q1): {df_copy[col].quantile(0.25)}")
    print(f"3rd Quartile (Q3): {df_copy[col].quantile(0.75)}")
    print(f"IQR (Interquartile Range): {df_copy[col].quantile(0.75) - df_copy[col].quanti

    # Plot boxplot to visualize spread and detect outliers
    sns.boxplot(data=df_copy[col])
    plt.title(f'Boxplot of {col}')
    plt.show()
```

## ✓ Dataset Otimizado para Modelagem

Exportação do dataset tratado e otimizado para análises avançadas.

**Arquivo Gerado:** data\_set\_repositorios\_mediana.csv **Melhorias Obtidas:**

-  Outliers tratados estatisticamente
-  Distribuições mais simétricas
-  Dados preparados para machine learning
-  Preservação de padrões naturais

```
df_copy.to_csv('../dados/data_set_repositorios_mediana.csv', index=False)
```

## ✓ 🤖 ARQUIVO 4: Modelagem e Análise Preditiva

### 🎯 Objetivo

Aplicar técnicas de machine learning para descobrir relações preditivas e padrões complexos nos dados de repositórios GitHub.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

```
df = pd.read_csv('../dados/github_repos_limpo.csv')
df.head(2)
```

	name	owner	stars	forks	language	created_at	updated_at
0	free-programming-books	EbookFoundation	359735	63576	Python	2013-10-11 06:50:37+00:00	2025-06-21 02:09:07+00:00
1	public-apis	public-apis	351991	37004	Python	2016-03-20 23:49:42+00:00	2025-06-21 02:06:40+00:00

## ✓ 🔗 Análise de Correlação

### Matriz de Correlação Completa

Investigação das relações lineares entre todas as variáveis numéricas.

#### Insights Esperados:

- Identificação de relações fortes entre métricas
- Descoberta de redundâncias
- Base para seleção de features

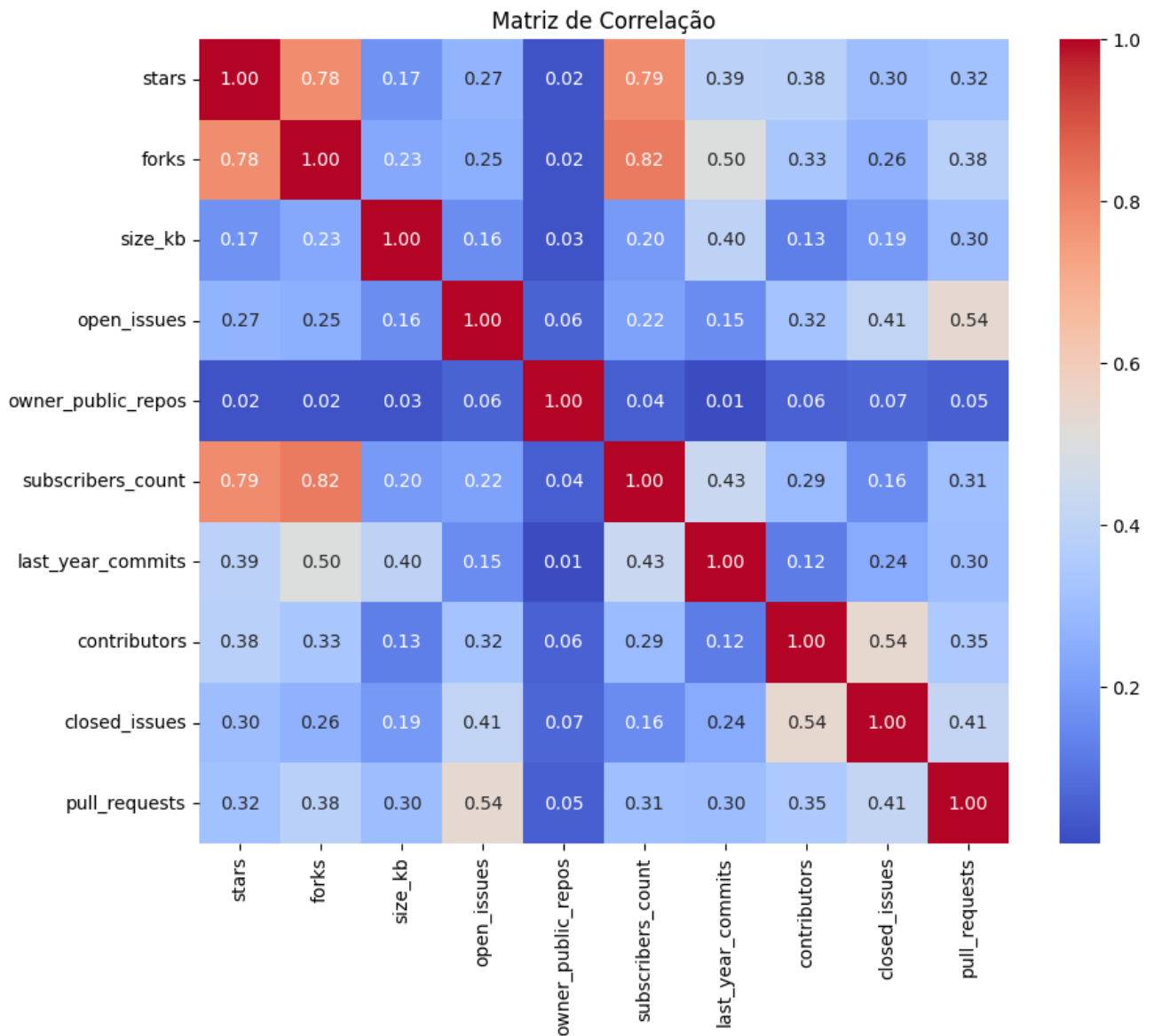
#### Visualização:

- Heatmap com escala de cores intuitiva
- Valores de correlação anotados
- Identificação visual de clusters de correlação



```
numerical_columns = df.select_dtypes(include=['int64', 'float64'])
```

```
matriz = numerical_columns.corr()  
plt.figure(figsize=(10, 8))  
sns.heatmap(matriz, annot=True, cmap='coolwarm', fmt=".2f")  
plt.title("Matriz de Correlação")  
plt.show()
```



## ✓ Modelos de Regressão

### Análise de Relações Bivariadas

Exploração visual de relacionamentos através de scatter plots categorizados.

#### Relações Investigadas:

1. **Stars × Forks:** Correlação entre popularidade e interesse em contribuição
2. **Commits × Forks:** Atividade de desenvolvimento vs. interesse externo
3. **Commits × Tamanho:** Relação entre atividade e complexidade do projeto
4. **Stars × Contributors:** Popularidade vs. diversidade de colaboradores
5. **Contributors × Pull Requests:** Colaboração ativa vs. contribuições externas

### Modelo de Regressão Linear

#### Primeira Abordagem: Dados Originais

##### Variáveis:

- **X:** subscribers\_count (Inscritos)
- **Y:** stars (Popularidade)

##### Tratamento de Dados:

- Filtro para repositórios com  $\leq 20.000$  stars (redução de outliers extremos)
- Divisão treino/teste (80/20)

#### Segunda Abordagem: Transformação Logarítmica

**Motivação:** Relações exponenciais são comuns em métricas de popularidade

##### Transformações:

- `log_stars = log1p(stars)`
- `log_subscribers_count = log1p(subscribers_count)`

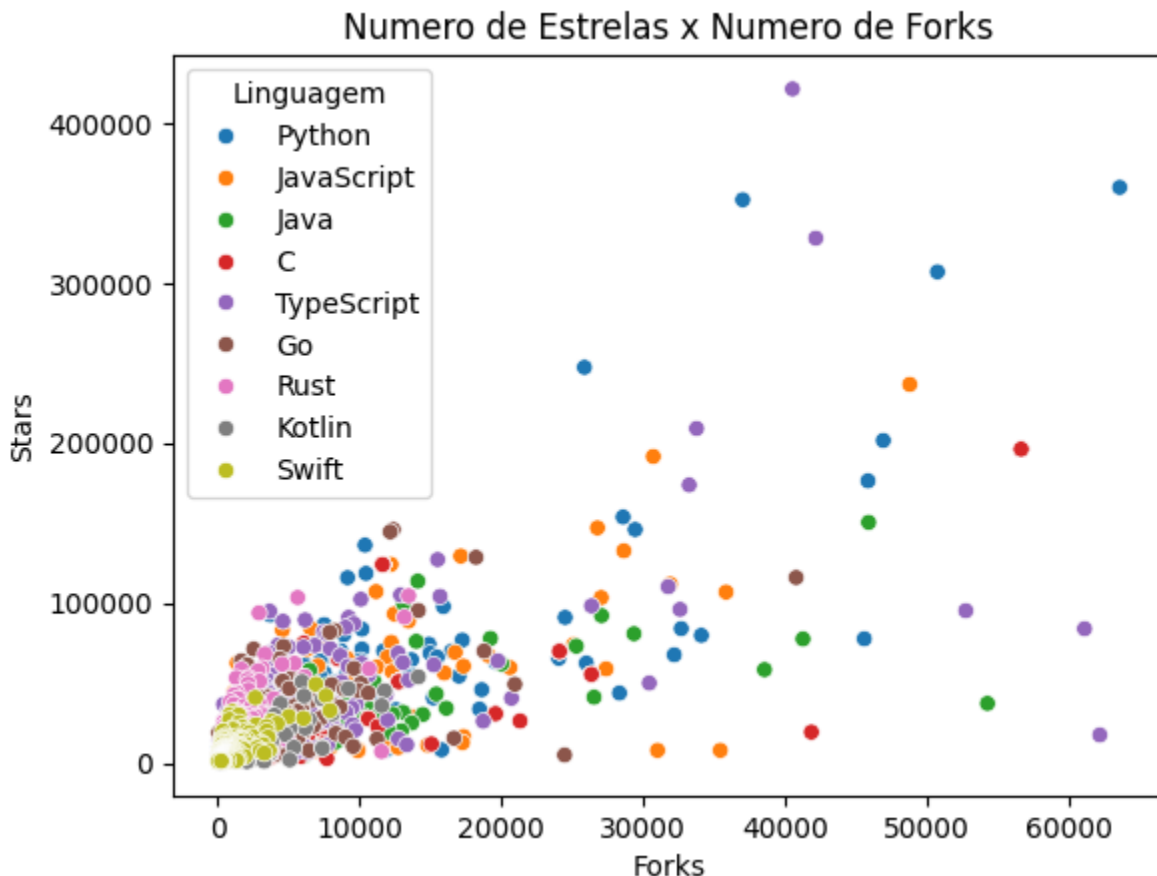
##### Benefícios:

- ☒ Linearização de relações exponenciais
- ☒ Redução da influência de outliers
- ☒ Melhoria na qualidade do ajuste ( $R^2$ )

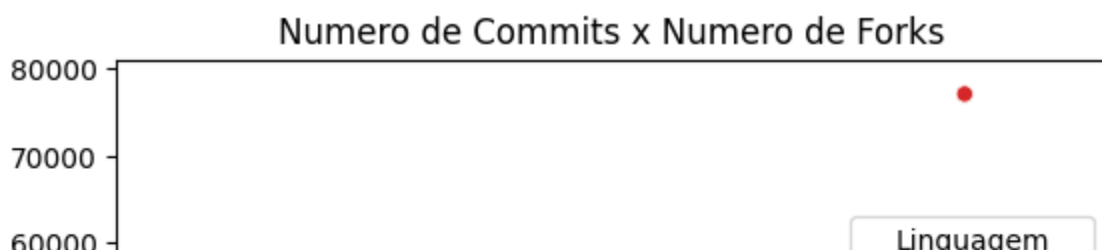
#### Métricas de Avaliação:

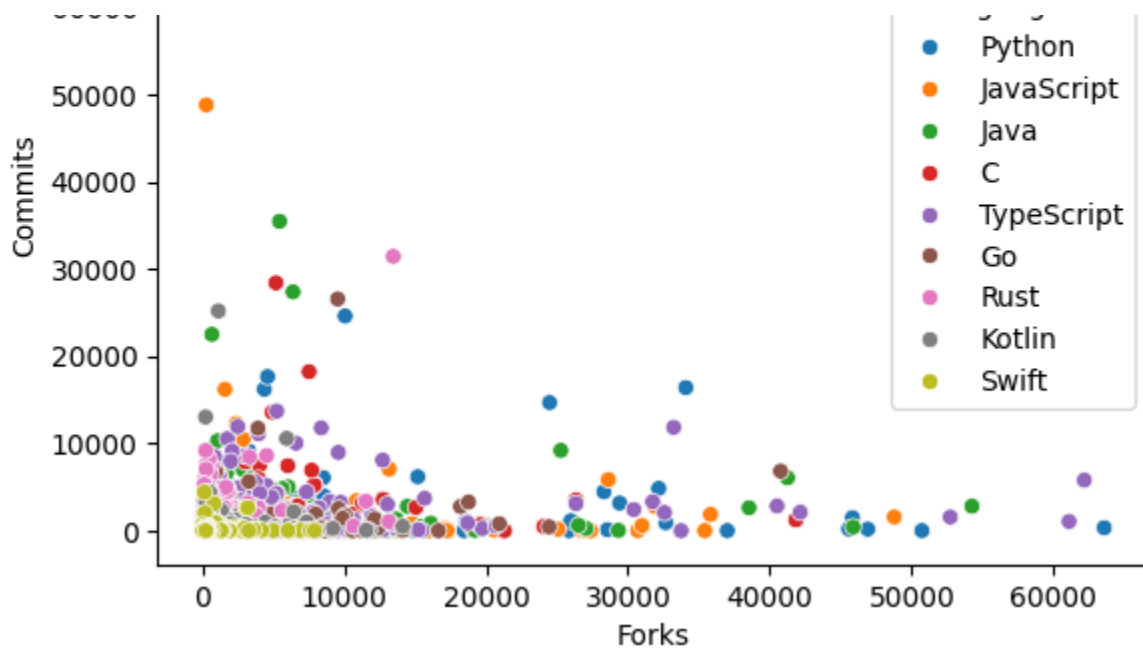
- **R<sup>2</sup> (Coeficiente de Determinação):** Porcentagem da variância explicada
- **Coeficientes:** Interpretação da relação linear

```
sns.scatterplot(data=df, x="forks", y="stars", hue="language")
plt.title("Numero de Estrelas x Numero de Forks")
plt.xlabel("Forks")
plt.ylabel("Stars")
plt.legend(title="Linguagem")
plt.show()
```

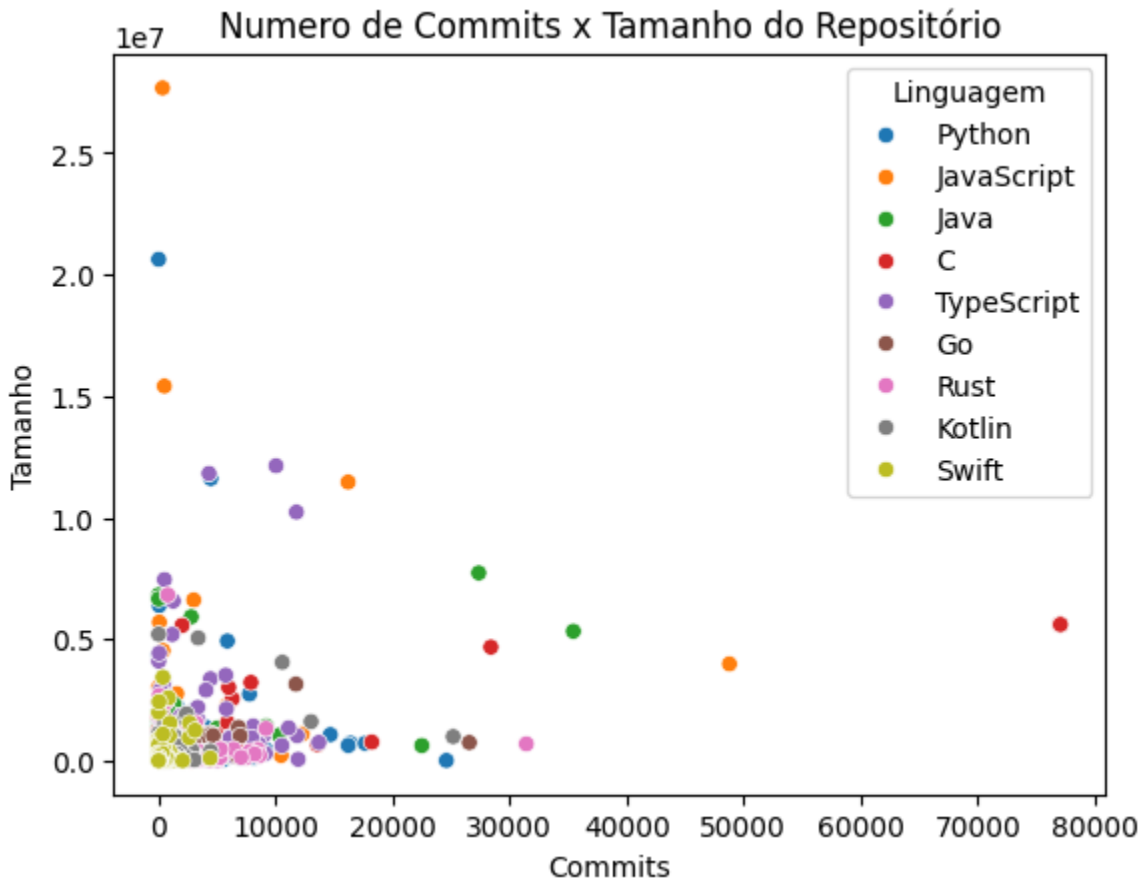


```
sns.scatterplot(data=df, x="forks", y="last_year_commits", hue="language")
plt.title("Numero de Commits x Numero de Forks")
plt.xlabel("Forks")
plt.ylabel("Commits")
plt.legend(title="Linguagem")
plt.show()
```

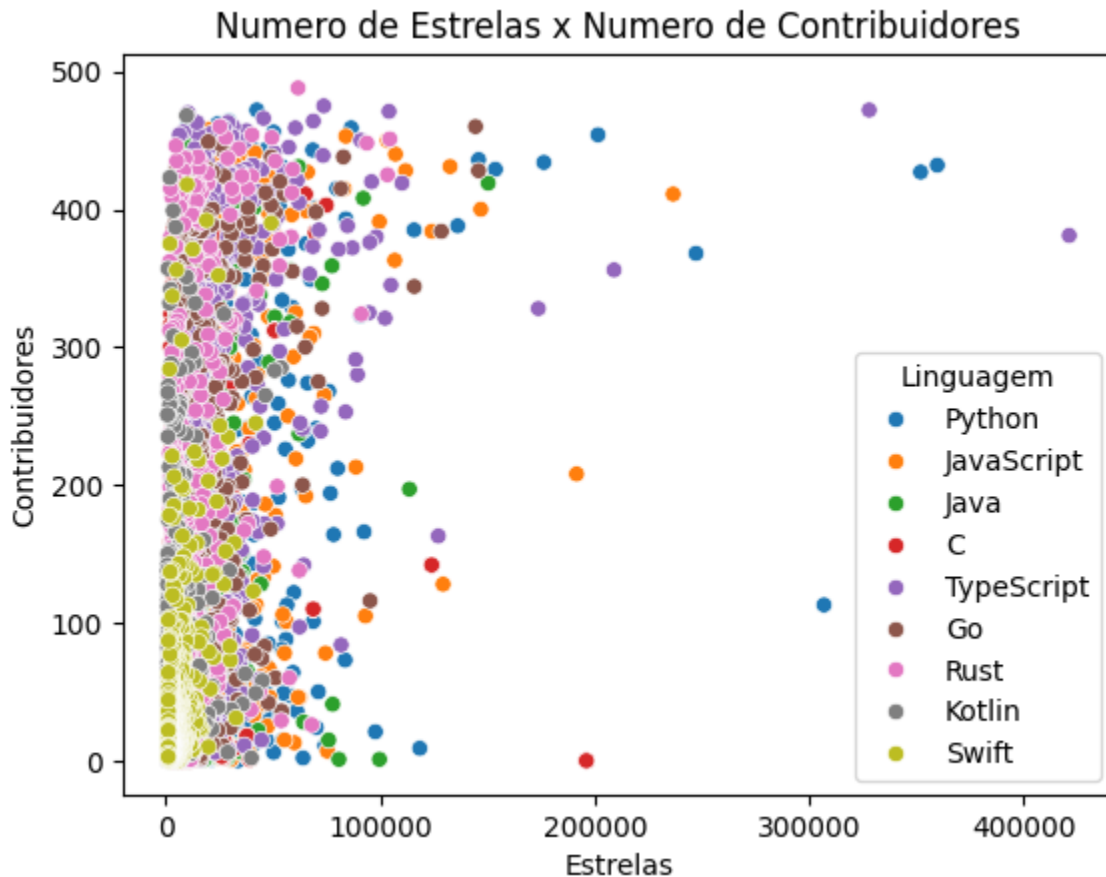




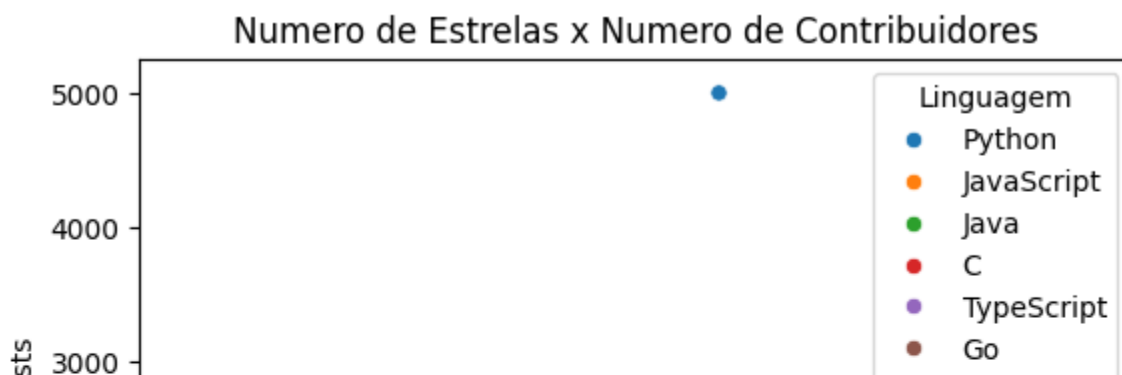
```
sns.scatterplot(data=df, x="last_year_commits", y="size_kb", hue="language")
plt.title("Numero de Commits x Tamanho do Repositório")
plt.xlabel("Commits")
plt.ylabel("Tamanho")
plt.legend(title="Linguagem")
plt.show()
```

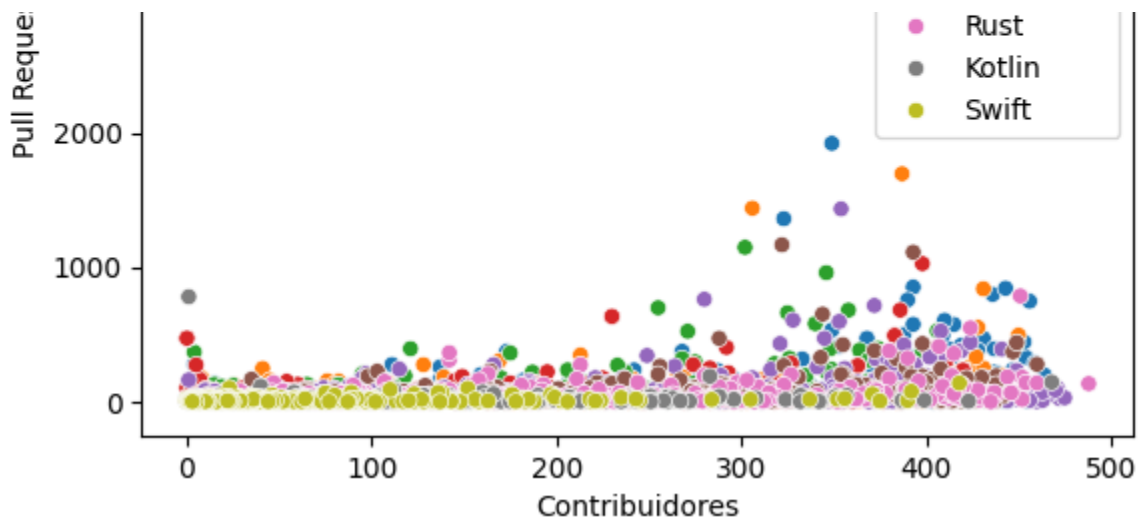


```
sns.scatterplot(data=df, x="stars", y="contributors", hue="language")
plt.title("Numero de Estrelas x Numero de Contribuidores")
plt.xlabel("Estrelas")
plt.ylabel("Contribuidores")
plt.legend(title="Linguagem")
plt.show()
```



```
sns.scatterplot(data=df, x="contributors", y="pull_requests", hue="language")
plt.title("Numero de Estrelas x Numero de Contribuidores")
plt.xlabel("Contribuidores")
plt.ylabel("Pull Requests")
plt.legend(title="Linguagem")
plt.show()
```





```

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

df['log_stars'] = np.log1p(df['stars'])
df['log_subscribers_count'] = np.log1p(df['subscribers_count'])

df_so2 = df[(df['stars'] <= 20000)]

df_reg = df_so2[["stars", "subscribers_count", "log_stars", "log_subscribers_count", "lang

# variáveis independentes e dependentes
X = df_reg[["subscribers_count"]]
y = df_reg["stars"]

# separar em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# criar e treinar o modelo
model = LinearRegression()
model.fit(X_train, y_train)

# avaliação do modelo
r2 = model.score(X_test, y_test)
print(f"Coeficiente de determinação R²: {r2:.2f}")
print(f"Coeficiente angular: {model.coef_[0]:.2f}, Intercepto: {model.intercept_:.2f}")

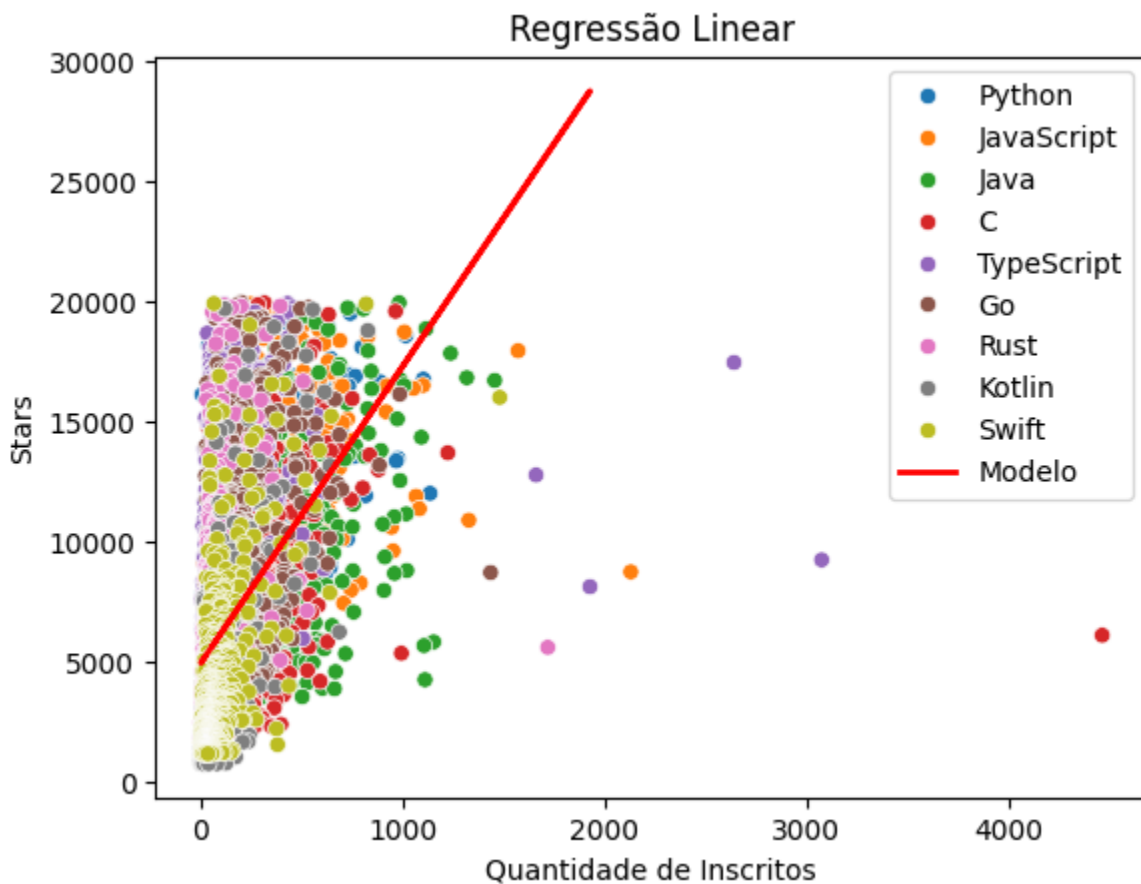
    Coeficiente de determinação R²: 0.23
    Coeficiente angular: 12.36, Intercepto: 4929.08

# previsão
y_pred = model.predict(X_test)

# gráfico
sns.scatterplot(data=df reg, x="subscribers count", v="stars", hue="language")

```

```
plt.plot(X_test, y_pred, color="red", linewidth=2, label="Modelo")
plt.xlabel("Quantidade de Inscritos")
plt.ylabel("Stars")
plt.title("Regressão Linear")
plt.legend()
plt.show()
```



```
# variáveis independentes e dependentes
X = df_reg[["log_subscribers_count"]]
y = df_reg["log_stars"]

# separar em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# criar e treinar o modelo
model = LinearRegression()
model.fit(X_train, y_train)

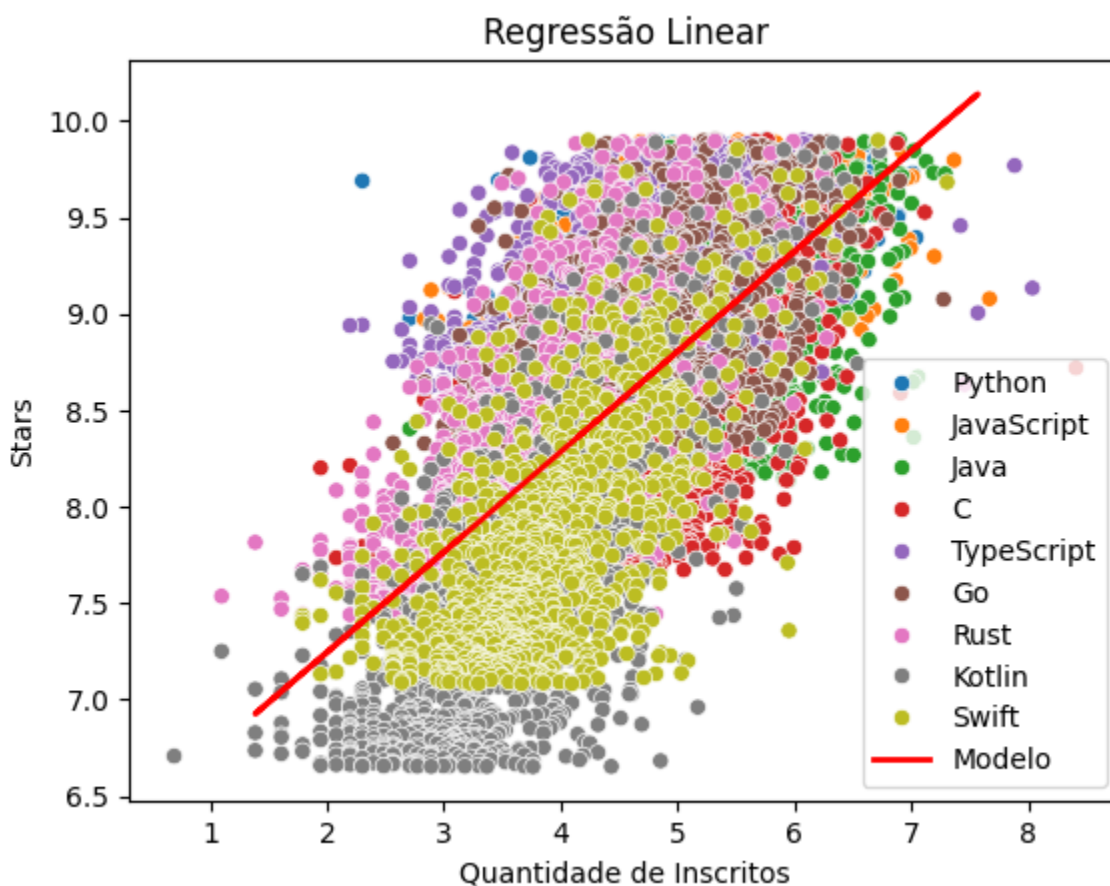
# avaliação do modelo
r2 = model.score(X_test, y_test)
print(f"Coeficiente de determinação R²: {r2:.2f}")
print(f"Coeficiente angular: {model.coef_[0]:.2f}, Intercepto: {model.intercept_:.2f}")
```

```
Coeficiente de determinação R²: 0.41
Coeficiente angular: 0.52, Intercepto: 6.20
```

```
y_pred = model.predict(X_test)
```

```
# gráfico
```

```
sns.scatterplot(data=df_reg, x="log_subscribers_count", y="log_stars", hue="language")  
plt.plot(X_test, y_pred, color="red", linewidth=2, label="Modelo")  
plt.xlabel("Quantidade de Inscritos")  
plt.ylabel("Stars")  
plt.title("Regressão Linear")  
plt.legend()  
plt.show()
```



## ✓ 🎯 Modelos de Classificação

### Classificação por Popularidade

#### Engenharia de Features: Categorização de Popularidade

Criação de classes de popularidade baseadas em distribuição estatística:

#### Classes Definidas:

- **Média:** 770 - 3.000 stars



- 10 51 3 11 13 1 1 16 51 13

```

dt_so['popularidade'] = pd.cut(dt_so['stars'],
                                bins=[770, 3000, 10000, float('inf')],
                                labels=['media', 'alta', 'muito_alta'])

X = df_so[["forks", "subscribers_count", "last_year_commits"]]
y = df_so["popularidade"]

# normalizar
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# separar conjuntos
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_s

# treinar o modelo de classificação
clf = LogisticRegression(max_iter=1000)
clf.fit(X_train, y_train)

# avaliar
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))

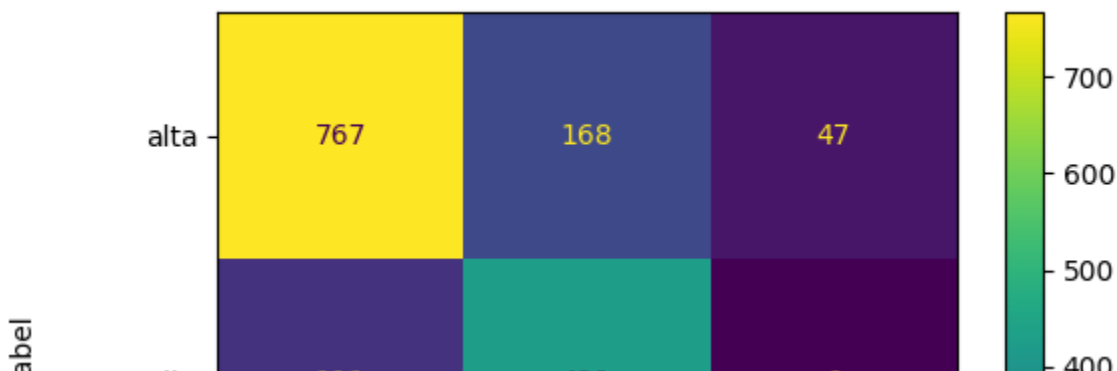
# matriz de confusão
ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test)
plt.show()

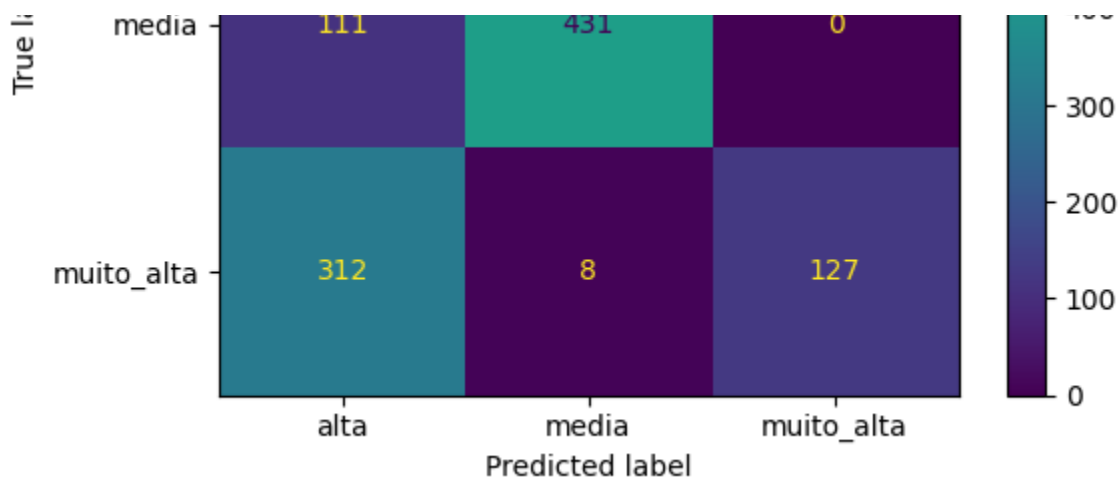
```

	precision	recall	f1-score	support
alta	0.64	0.78	0.71	982
media	0.71	0.80	0.75	542
muito_alta	0.73	0.28	0.41	447
accuracy			0.67	1971
macro avg	0.69	0.62	0.62	1971
weighted avg	0.68	0.67	0.65	1971

/tmp/ipykernel\_30481/2117018212.py:3: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/ufuncs-10min.html>  
df\_so['popularidade'] = pd.cut(df\_so['stars'],





```
X = df_so[["forks", "subscribers_count", "last_year_commits"]]
y = df_so["language"]

# normalizar
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

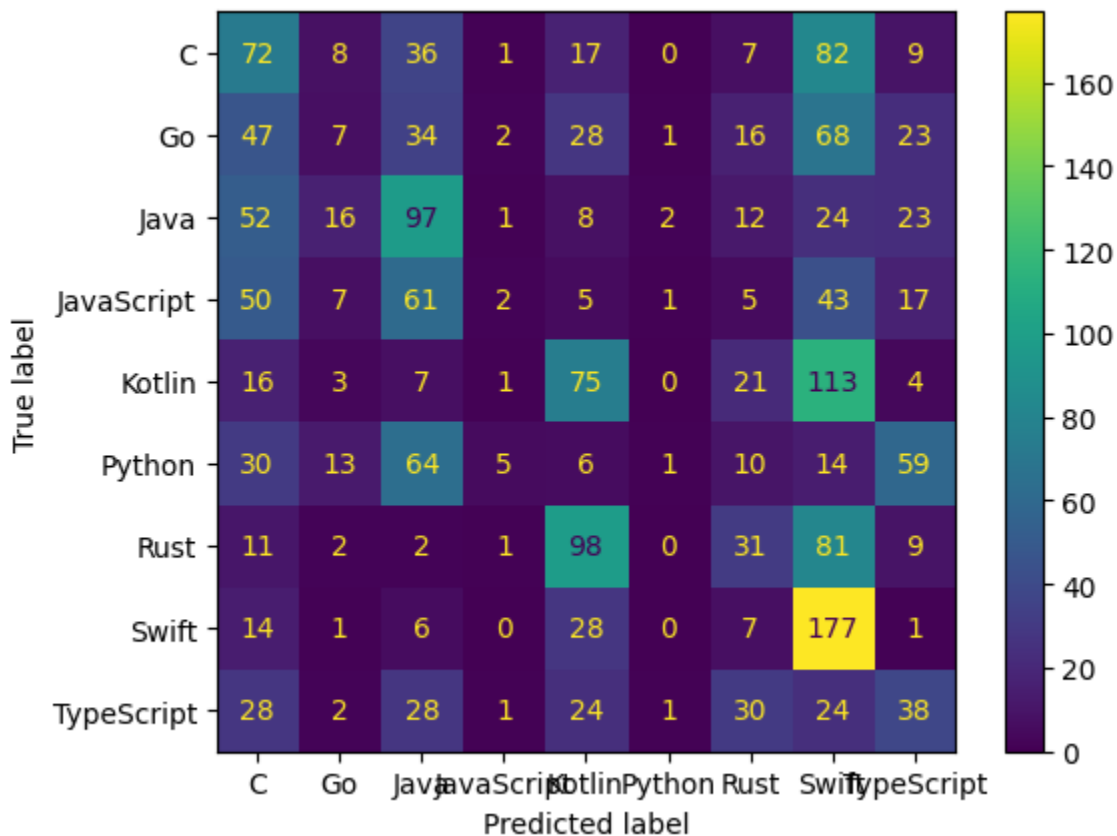
# separar conjuntos
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_s

# treinar o modelo de classificação
clf = LogisticRegression(max_iter=1000)
clf.fit(X_train, y_train)

# avaliar
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))

# matriz de confusão
ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test)
plt.show()
```

	precision	recall	f1-score	support
C	0.23	0.31	0.26	232
Go	0.12	0.03	0.05	226
Java	0.29	0.41	0.34	235
JavaScript	0.14	0.01	0.02	191
Kotlin	0.26	0.31	0.28	240
Python	0.17	0.00	0.01	202
Rust	0.22	0.13	0.17	235
Swift	0.28	0.76	0.41	234
TypeScript	0.21	0.22	0.21	176
accuracy			0.25	1971
macro avg	0.21	0.24	0.19	1971
weighted avg	0.22	0.25	0.20	1971



## Comparação de Abordagens

### Regressão vs. Classificação

#### Regressão Linear:

- ✓ **Vantagens:** Interpretabilidade direta, previsões contínuas
- ✗ **Limitações:** Assume relações lineares, sensível a outliers

#### Classificação:

- ✓ **Vantagens:** Robusta a outliers, decisões categóricas claras
- ✗ **Limitações:** Perda de informação granular, escolha de limites subjetiva

### Transformações Logarítmicas

**Impacto:** Significativa melhoria no  $R^2$  ao linearizar relações exponenciais naturais em métricas de popularidade.



## Limitações e Considerações

#### Limitações dos Modelos:

### Limitações dos Modelos.

- **Causalidade:** Correlação não implica causalidade
- **Temporalidade:** Snapshot pontual vs. evolução temporal
- **Viés de Seleção:** Apenas repositórios populares incluídos

### Melhorias Futuras:

- **Modelos Ensemble:** Random Forest, Gradient Boosting
- **Features Temporais:** Tendências de crescimento
- **Cross-validation:** Validação mais robusta
- **Feature Engineering:** Criação de variáveis derivadas