# Relatório em Quarto + GitHub

Matheus Felix de Assunção

2025-07-24

## Introdução

Este relatório apresenta ferramentas e conceitos importantes para quem está começando a aprender programação e trabalhar com dados. Vamos conhecer o **Git e o GitHub** para poder salvar e gurdar nossas versos projetos, e ter uma rede social onde podemos ver ideias de outras pessoas que estão trabalhando com dados, o **pip** para instalar pacotes no Python, e os **conceitos básicos da linguagem Python**. Esses assuntos são muito importantes para quem deseja atuar na área de dados e outras tecnologias.

---

## Git e GitHub

Git e GitHub são ferramentas poderosas para controlar as mudanças no seu código e colaborar com outras pessoas.

### Historia do Git:

O Git nasceu em 2005 de uma necessidade muito específica e urgente. Seu criador foi ninguém menos que Linus Torvalds, o mesmo gênio por trás da criação do sistema operacional Linux.

Figure 1

**Porque ele criou o GIT?**

Antes do Git, o desenvolvimento do kernel do Linux (o "coração" do sistema operacional) era gerenciado por um sistema de controle de versão proprietário chamado BitKeeper. Esse sistema era muito bom e eficiente, mas em 2005, a relação entre a empresa que o desenvolvia e a comunidade Linux se deteriorou. O acesso gratuito ao BitKeeper foi revogado, deixando Linus Torvalds e sua equipe de milhares de desenvolvedores sem uma ferramenta para gerenciar o código de forma colaborativa.

**Qual foi a solução dele?**

Linus Torvalds, conhecido por sua abordagem pragmática, decidiu que, em vez de procurar outro sistema pago ou se adaptar a um gratuito existente que não atendia às suas necessidades, ele criaria um novo. O objetivo era ter um sistema:

- Distribuído : Onde cada desenvolvedor teria uma cópia completa do histórico do projeto.

- Extremamente rápido: Capaz de lidar com o enorme volume de mudanças no kernel do Linux.
- Capaz de garantir a integridade dos dados: Para evitar que o código fosse corrompido.

Em apenas duas semanas, Linus escreveu o núcleo do Git. Ele foi projetado para ser uma ferramenta de baixo nível, robusta e eficiente para a qual outras interfaces pudessem ser construídas. O nome "Git" é uma gíria britânica para "cabeça dura" ou "idiota", que Linus escolheu de forma irônica, referindo-se a si mesmo ou à frustração com os sistemas anteriores.

O Git rapidamente provou ser um sucesso e se tornou o padrão ouro para controle de versão em projetos de software, indo muito além do kernel do Linux.

## Historia do GitHub

### O Nascimento do GitHub!

O GitHub foi fundado em 2008 por um quarteto visionário: Tom Preston-Werner, Chris Wanstrath, P.J. Hyett e Scott Chacon. A ideia central era criar uma plataforma que não apenas hospedasse repositórios Git, mas que também incentivasse a interação social e a colaboração entre desenvolvedores. Eles construíram o serviço utilizando a framework de desenvolvimento web Ruby on Rails.


O foco inicial era aprimorar a experiência de trabalho em equipe, oferecendo funcionalidades que simplificassem a revisão de código, o rastreamento de problemas e a integração de contribuições. O GitHub rapidamente se destacou por sua interface intuitiva e pelas ferramentas que tornavam o processo de desenvolvimento colaborativo muito mais acessível do que as alternativas existentes na época.

## O Propósito do Git e do GitHub.

Pense no Git e no GitHub como parceiros indispensáveis para quem cria software ou lida com código hoje em dia. Eles trabalham juntos, e cada um tem um papel fundamental.

- O **Git** é como a memória do seu projeto. Ele registra cada alteração que você faz no código, permitindo que você navegue por versões passadas, identifique quem fez o que e quando, e integre contribuições de diferentes pessoas sem dores de cabeça. É a ferramenta que mantém a ordem e a rastreabilidade de todas as modificações.
- Já o **GitHub** é o ponto de encontro online para seus projetos. É onde seu código, que o Git controla, fica guardado na internet. Mais do que um simples backup, ele transforma o processo de desenvolvimento em algo colaborativo e social. No GitHub, equipes podem trabalhar no mesmo código de forma sincronizada, revisar o trabalho um do outro e gerenciar o projeto de maneira centralizada.

### Comandos básicos

| Comando | Função |
|---|---|
| `git init` | Inicializa um repositório local |
| `git add .` | Adiciona arquivos ao controle de versão |
| `git commit -m` | Salva as alterações com uma mensagem |
| `git push` | Envia mudanças ao repositório remoto |
| `git pull` | Atualiza o repositório local |
| `git clone` | Copia um repositório existente |

### Repositório e Commit

- **Repositório:** onde o projeto e seu histórico são gurdados.
- **Commit:** um registro de alteração com uma descrição.

## Conceitos Iniciais de Python

Python é uma linguagem de programação muito popular, conhecida por ser fácil de ler e poderosa. É um ótimo ponto de partida para quem está começando!

### Tipos de Dados

No Python, cada informação tem um tipo. Os mais comuns são:

- **int** (inteiro): números inteiros, sem vírgula. Ex: `10`, `150`, `-5`.
- **float** (ponto flutuante): números com vírgula. Ex: `3.14`, `99.99`.
- **str** (string): textos entre aspas. Ex: `"Olá, mundo!"`, `'Python é legal'`.
- **bool** (booleano): valores lógicos: `True` (verdadeiro) ou `False` (falso).

### Variáveis e Operadores Básicos

Variáveis são como “caixas” que guardam informações. Operadores servem para fazer contas ou comparações.

```
idade = 30        # variável do tipo int
nome = "Carlos"   # variável do tipo str

x = 10
y = 5
soma = x + y      # soma = 15
eh_maior = x > y  # eh_maior = True
```

- **Aritméticos:** +, -, *, /
- **Comparação:** ==, !=, >, <, >=, <=

## Listas, Tuplas e Dicionários

São formas de guardar várias informações juntas.

### Lista

Coleção ordenada e mutável (pode mudar depois de criada). Usa colchetes `[]`.

```
frutas = ["maçã", "banana", "uva"]
frutas.append("laranja")
print(frutas[0])  # "maçã"
```

### Tupla

Coleção ordenada e **imutável**. Usa parênteses `()`.

```
coordenadas = (10, 20)
print(coordenadas[1])  # 20
```

### Dicionário

Coleção de pares `chave: valor`. Usa chaves `{}`.

```
pessoa = {"nome": "Maria", "idade": 25, "cidade": "João Pessoa"}
print(pessoa["nome"])    # "Maria"
pessoa["idade"] = 26     # altera a idade
```

## Estruturas de Controle

Controlam o fluxo do programa.

### Condicionais (`if`, `elif`, `else`)

```
nota = 70
if nota >= 70:
    print("Aprovado!")
elif nota >= 50:
    print("Em recuperação.")
```

```
else:
    print("Reprovado.")
```

#### Laço `for`

Repete algo para cada item de uma sequência.

```
numeros = [1, 2, 3, 4, 5]
for n in numeros:
    print(n)
```

#### Laço `while`

Repete enquanto a condição for verdadeira.

```
contador = 0
while contador < 3:
    print(f"Contador: {contador}")
    contador += 1
```

### Funções

Funções são blocos de código reutilizáveis.

```
def saudar(nome):
    """Retorna uma saudação personalizada."""
    return f"Olá, {nome}!"

mensagem = saudar("Aluno")
print(mensagem)  # Olá, Aluno!
```

---

## pip

### O que é?

O **pip** é uma ferramenta muito importante para quem programa em Python. Ela funciona como o gerenciador oficial de pacotes da linguagem. Basicamente, quando a gente precisa usar alguma funcionalidade que não vem no Python "básico" – como uma biblioteca para análise de dados ou para criar gráficos –, o **pip** é quem nos ajuda a instalar, atualizar ou remover esses pacotes que foram criados por outros desenvolvedores.

### Como instalar pacotes no Python

Você instala pacotes usando o comando pip install diretamente no seu terminal ou prompt de comando.

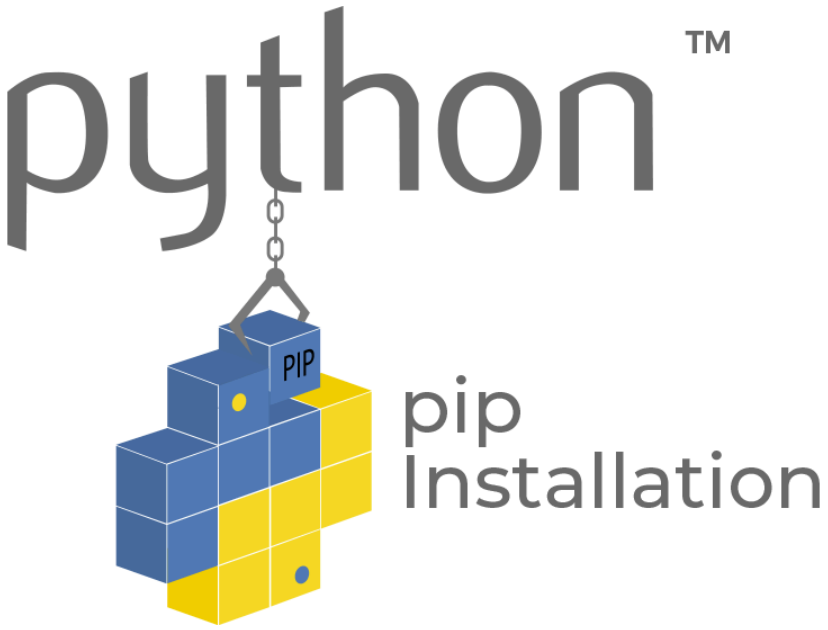


Figure 2

## Passo a Passo

### Abra o Terminal

- **No Windows:** Você pode pesquisar por `"Prompt de Comando"` ou `"PowerShell"` no menu Iniciar e abri-lo.
- **No macOS/Linux:** Abra o aplicativo `"Terminal"`.

### Digite o Comando de Instalação

A sintaxe básica para instalar um pacote com o `pip` é:

```
pip install nome_do_pacote
```

## Exemplos Práticos de Instalação com `pip`

Vamos ver alguns exemplos de instalação de bibliotecas populares usando o `pip`:

### Instalar o `pandas` (análise de dados)

```
pip install pandas
```

### Instalar o `numpy` (operações numéricas)

```
pip install numpy
```

### Instalar o `matplotlib` (criação de gráficos)

```
pip install matplotlib
```

### Instalar uma versão específica de um pacote

Isso pode ser útil para garantir compatibilidade com outros pacotes:

```
pip install requests==2.28.1
```

Isso instalará a versão `2.28.1` da biblioteca `requests`.

### Atualizar um pacote para a versão mais recente

Use o `--upgrade`:

```
pip install --upgrade nome_do_pacote
```

**Exemplo: atualizar o `pandas`:**

```
pip install --upgrade pandas
```

## Funções em Python: Seus Blocos de Código Reutilizáveis

Funções em Python são como pequenas *"receitas"* que você cria para realizar tarefas específicas no seu programa.
A grande vantagem é que, ao invés de escrever o mesmo código várias vezes, você o define uma vez só em uma função e pode usá-lo sempre que precisar.
Isso deixa seu código mais **organizado** e **fácil de entender**.

---

### Definindo Funções com `def`

Para criar uma função, você usa a palavra-chave `def`, seguida do nome que você escolheu para ela, e depois parênteses `()`.
É dentro desses parênteses que indicamos se a função precisa receber alguma informação para trabalhar (os **parâmetros**).

```
# Uma função simples que apenas mostra uma mensagem
def mostrar_mensagem():
    print("Olá! Bem-vindo à minha primeira função.")

# Para fazer a função "acontecer", a gente a chama pelo nome:
mostrar_mensagem()  # Saída: Olá! Bem-vindo à minha primeira função.
```

---

## Parâmetros: Dando Informações à Função

Muitas vezes, uma função precisa de dados externos para realizar sua tarefa.
Essas informações são chamadas de **parâmetros** (ou **argumentos**) e são listadas dentro dos parênteses na definição da função.

```
# Função que recebe um 'nome' como parâmetro
def saudar_usuario(nome):
    print(f"Olá, {nome}! É um prazer ter você aqui.")

# Chamadas da função com valores diferentes
saudar_usuario("Mariana")  # Saída: Olá, Mariana! É um prazer ter você aqui.
saudar_usuario("Bruno")    # Saída: Olá, Bruno! É um prazer ter você aqui.
```

Também podemos usar **mais de um parâmetro**:

```
# Função que calcula a soma de dois números
def calcular_soma(numero1, numero2):
    total = numero1 + numero2
    print(f"A soma de {numero1} e {numero2} é: {total}")

calcular_soma(15, 7)  # Saída: A soma de 15 e 7 é: 22
```

---

## Retorno: O Resultado da Função

Depois que uma função faz o que tem que fazer, ela pode nos devolver um **resultado**.
Esse resultado é o **retorno**, e usamos a palavra-chave `return` para isso.

Se uma função **não tem `return`**, ela retorna `None` (nada de útil).

```
# Função que multiplica dois valores e retorna o resultado
def multiplicar(valor1, valor2):
    produto = valor1 * valor2
    return produto  # Retorna o valor da multiplicação

# Armazenando o retorno em uma variável
resultado_multiplicacao = multiplicar(6, 4)
print(f"O produto é: {resultado_multiplicacao}")  # Saída: O produto é: 24

# Ou usando o retorno diretamente
print(multiplicar(10, 2))  # Saída: 20
```

---

Dominar **funções** é um passo **gigante** para escrever códigos mais **limpos**, **modulares** e **eficientes**!

## Boas Práticas em Python: Como Escrever um Código Top!

Escrever código Python não é só fazer ele funcionar; é também garantir que ele seja **fácil de ler e entender**. Para isso, usamos algumas **boas práticas** que deixam o código mais organizado e profissional.

---

### Comentários (#)

Comentários são **anotações dentro do código** que ajudam a explicar o que ele está fazendo. Tudo o que vem depois do símbolo **#** é **ignorado pelo Python** e não afeta a execução do programa.


Figure 3

Você pode usar comentários para:

- **Explicar partes complexas** do código.
- **Descrever o que um trecho faz**.
- **Adicionar lembretes para você ou outros desenvolvedores**.

---

### Outras Boas Práticas

- **Comentários**: use # para explicar partes do código.
- **Nomes claros**: use nomes de variáveis descritivos, como `idade`, `media`, `resultado`.
- **Identação**: sempre use 4 espaços (ou `TAB`) para blocos de código, mantendo o código organizado.

## Conclusão

Este relatório foi criado usando o Quarto, renderizado em HTML e versionado com Git.
Ele serve como base para revisarmos ferramentas e conceitos essenciais na vida de um cientista de dados e estatístico.