

---

## **Criando Aplicativos Web com Streamlit**

Asimov Academy

# ASIMOV

## Conteúdo

<b>01. Introdução ao Streamlit</b>	<b>3</b>
O que é o Streamlit?	3
Projetos com Streamlit	3
Comparação com o Plotly Dash	3
Exemplos de Projetos da Comunidade	3
Prettypapp	4
Streamly Streamlit Assistant	5
<b>02. Rodando o Primeiro Dashboard</b>	<b>6</b>
Desenvolvimento do Projeto	6
Configurando o ambiente	6
Instalando o Streamlit	6
Instalando o Pandas	6
Lendo uma tabela com Pandas e Streamlit	7
Exemplo de Código	7
Criando o Dashboard	7
Rodando o Dashboard	7
<b>03. Apresentando Dados no Streamlit</b>	<b>9</b>
Entendendo o Fluxo de Desenvolvimento	9
Utilizando o <code>st.write()</code>	9
Mesclando Bibliotecas	9
Criando Gráficos	9
Ajustando o Eixo X	10
Construção do Dashboard	10
<b>04. Input Widgets</b>	<b>11</b>
Trabalhando com Dados	11
Configurando a Página	12
Selecionando Artistas com Select Box	13
Filtrando Dados com Base na Seleção	13
Checkbox para Controlar a Exibição	14
Composição de Seletores	15
<b>05. Layout no Streamlit</b>	<b>17</b>
Sidebar	17
Colunas	18

Personalizando a Largura das Colunas . . . . .	18
<b>06. Cacheamento e Multipages</b>	<b>20</b>
O que é Caching? . . . . .	20
Criando Múltiplas Páginas . . . . .	20
Explicação da Estrutura: . . . . .	20
Armazenando Informações com SessionState . . . . .	21
Implementando Cache em Funções . . . . .	21
Simulando Operações Pesadas . . . . .	21
Como funciona . . . . .	22
<b>07. Componentes Adicionais</b>	<b>23</b>
Navegação e Recursos . . . . .	23
Exemplos de Componentes . . . . .	23
Conclusão . . . . .	23

## 01. Introdução ao Streamlit

### O que é o Streamlit?

O Streamlit é uma biblioteca que ganhou notoriedade nos últimos dois anos e cresceu bastante em popularidade. Os desenvolvedores a consideram a maneira mais rápida de construir e compartilhar aplicativos de dados, ou simplesmente dashboards. Um dashboard é uma aplicação que possui um painel interativo para visualização e exploração dos nossos conjuntos de dados, independentemente do tipo.

Usando essa biblioteca, é possível montar dashboards que mostram gráficos e textos de forma fluida e rápida. Com apenas 10 linhas de código você já consegue criar dashboards interativos!

### Projetos com Streamlit

Estes projetos da Asimov Academy evidenciam a diversidade e a simplicidade do que podemos fazer com Streamlit:

- **Login e Autenticação:** Projeto que cria uma camada a mais de segurança às aplicações, filtrando os acessos de usuários que não estão registrados no sistema.
- **Sistema de Chat:** Aplicativo de chat web versátil que pode ser aplicado em várias situações.
- **Integração com API do ChatGPT:** Construção de uma interface de chat personalizada usando a API da OpenAI e a plataforma Streamlit.

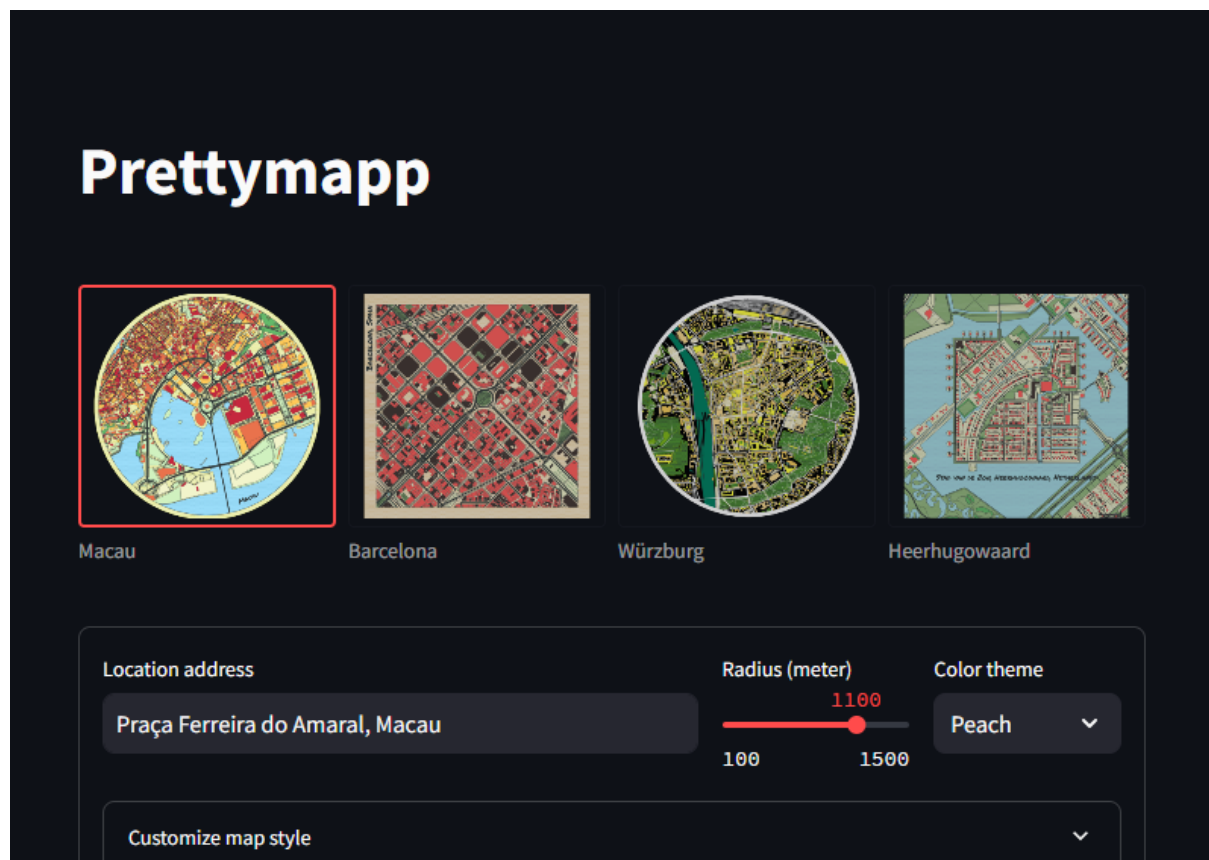
### Comparação com o Plotly Dash

É importante ressaltar que, enquanto o Plotly Dash oferece uma customização praticamente infinita, o custo em termos de desenvolvimento é bem alto. O Streamlit, por outro lado, entrega facilidade de uso, permitindo implementar os códigos de maneira muito mais acessível.

### Exemplos de Projetos da Comunidade

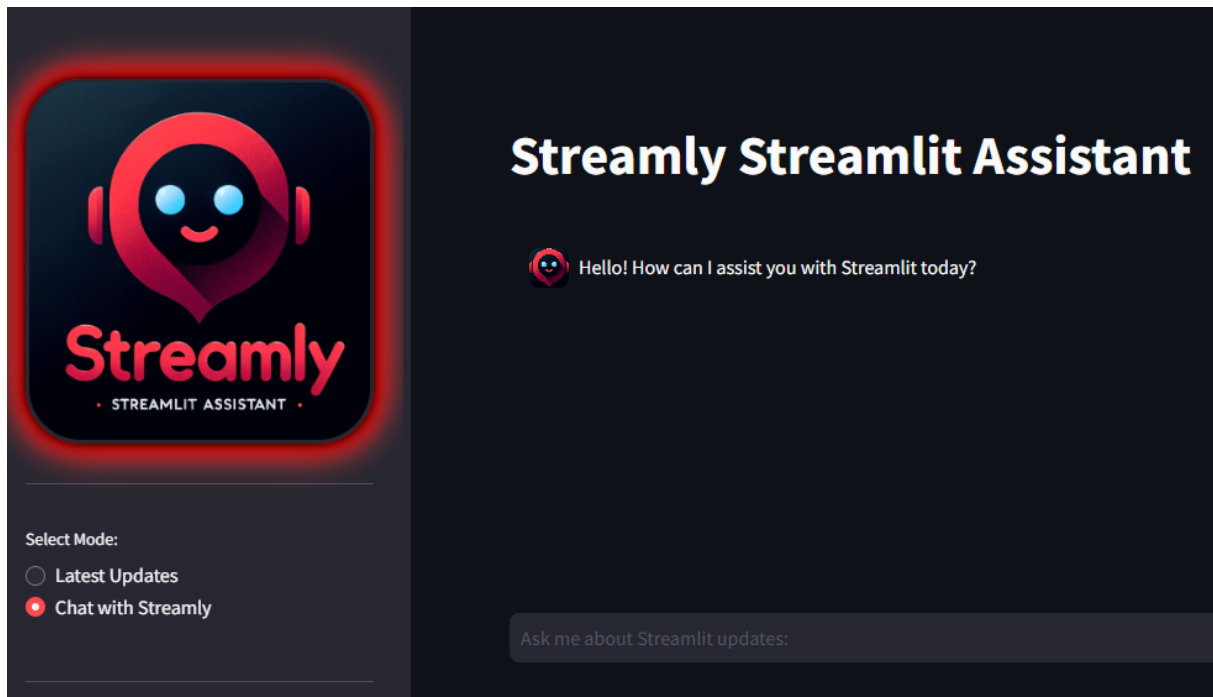
A comunidade do Streamlit tem desenvolvido diversos projetos criativos e inovadores, demonstrando a versatilidade e o potencial da biblioteca. Esse tópico foi separado para apresentar alguns desses projetos, acompanhe as imagens a seguir para visualizar essas soluções!

## Prettymapp



Link para acessar o site: [clique aqui](#)

### Streamly Streamlit Assistant



Link para acessar o site: [clique aqui](#)

## 02. Rodando o Primeiro Dashboard

### Desenvolvimento do Projeto

Primeiramente será necessário analisar um conjunto de dados do Kaggle relacionado a músicas do Spotify. Este conjunto de dados contém informações sobre diversas músicas, incluindo a quantidade de likes, visualizações, banda, álbum e estatísticas relacionadas à qualidade da música, como se ela é dançável ou tem uma batida acelerada. Você pode acessar o banco de dados csv através do material de aula:

- Material: [Clique aqui para acessar o material](#)
- Ou através do Kaggle: [clique aqui](#)

### Configurando o ambiente

#### Instalando o Streamlit

Antes de começarmos, precisamos instalar o Streamlit. Para isso, certifique-se de que você já configurou o Visual Studio Code e que as primeiras etapas do Python estão completas. Para instalar a biblioteca, digite um dos seguintes comandos no terminal:

```
pip install streamlit
```

ou assim dependendo de como foi instalado:

```
pip3 install streamlit
```

#### Instalando o Pandas

Para manipular e analisar os dados do projeto, também será necessário instalar a biblioteca **Pandas**. Utilize um dos seguintes comandos no terminal:

```
pip install pandas
```

ou, caso esteja utilizando pip3:

```
pip3 install pandas
```

## Lendo uma tabela com Pandas e Streamlit

Depois que o Streamlit estiver instalado, utilizaremos a biblioteca pandas para ler nosso conjunto de dados. Abaixo estão os passos que seguiremos:

1. **Importar a biblioteca Pandas:** Usaremos a estrutura `pd.read_csv` para ler nossos dados.
2. **Definir uma variável:** Esta variável conterá nosso DataFrame, que será rapidamente visualizado em nosso dashboard.

## Exemplo de Código

### Criando o Dashboard

Abaixo segue um exemplo que demonstra como rapidamente podemos criar um dashboard funcional com Streamlit. Crie um arquivo chamado **spotify.py** e escreva o seguinte código nele:

■ Não esqueça de colocar o caminho correto para o arquivo com os dados `caminho/para/seu/arquivo.csv`

```
import streamlit as st
import pandas as pd

# Leitura do conjunto de dados
df = pd.read_csv('caminho/para/seu/arquivo.csv')

# Exibindo o DataFrame
st.write(df)
```

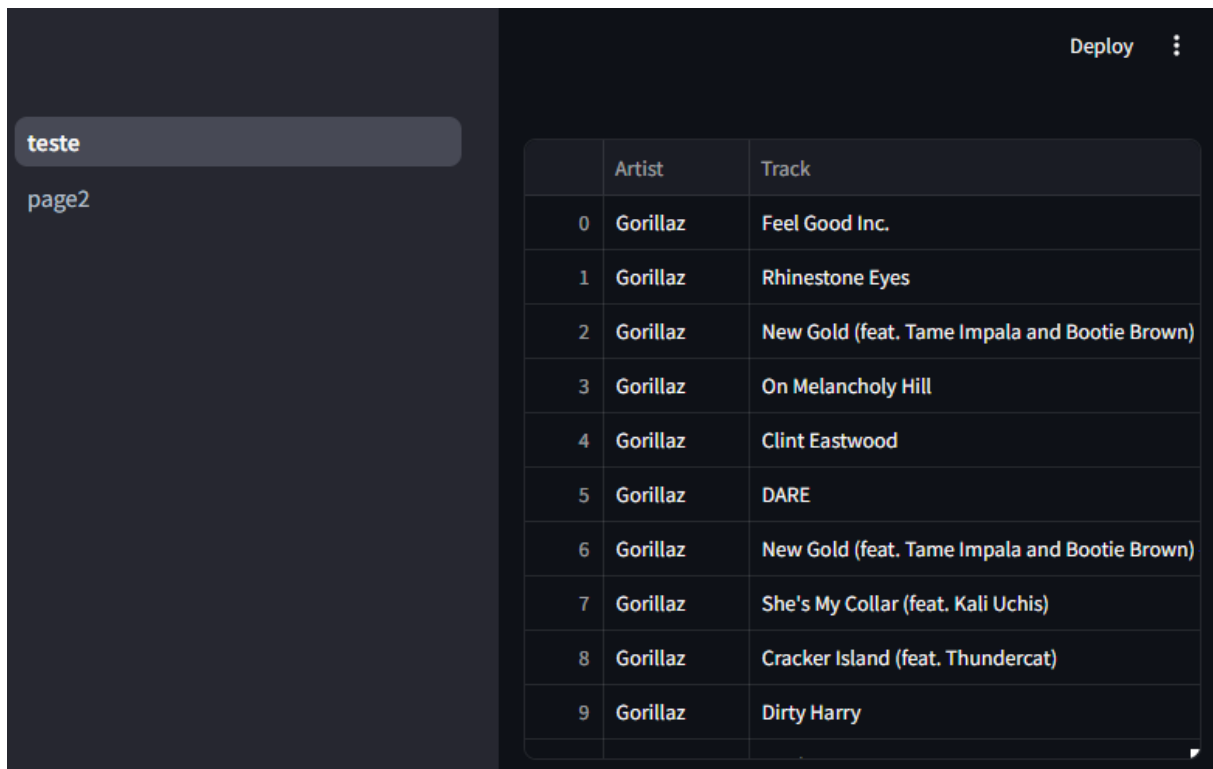
### Rodando o Dashboard

Para executar nosso dashboard, usamos o seguinte comando:

```
streamlit run spotify.py
```

Após rodar esse comando, o Streamlit abrirá uma nova aba no navegador, geralmente acessível em `localhost:8502`, onde já poderemos visualizar o nosso conjunto de dados em forma de DataFrame, exposto de maneira muito simples e prática.





	Artist	Track
0	Gorillaz	Feel Good Inc.
1	Gorillaz	Rhinestone Eyes
2	Gorillaz	New Gold (feat. Tame Impala and Bootie Brown)
3	Gorillaz	On Melancholy Hill
4	Gorillaz	Clint Eastwood
5	Gorillaz	DARE
6	Gorillaz	New Gold (feat. Tame Impala and Bootie Brown)
7	Gorillaz	She's My Collar (feat. Kali Uchis)
8	Gorillaz	Cracker Island (feat. Thundercat)
9	Gorillaz	Dirty Harry

**Figure 1:** Exemplo após rodar o comando

## 03. Apresentando Dados no Streamlit

Nesta aula, será explicado o funcionamento por trás do Streamlit e a lógica que permite dispor dados de maneira efetiva na tela.

### Entendendo o Fluxo de Desenvolvimento

O fluxo de desenvolvimento com o Streamlit é bastante intuitivo. Cada vez que você salva as alterações feitas no seu código, o servidor do Streamlit percebe essa mudança e executa o script de cima para baixo, renderizando apenas os elementos que sofreram alterações. Isso significa que, se você filtrar os dados que estão sendo apresentados, basta salvar o código para que esta mudança seja refletida automaticamente no dashboard.

### Utilizando o `st.write()`

Quando você insere um DataFrame na tela usando o comando `st.write()`, não é necessário especificar onde ele deve ser exibido. O Streamlit entende e renderiza o DataFrame automaticamente. Este método é uma espécie de “coringa”, pois pode ser utilizado para exibir diferentes tipos de elementos, como imagens, textos e gráficos.

### Mesclando Bibliotecas

Uma das grandes vantagens do Streamlit é a liberdade de combinar diferentes bibliotecas de visualização de dados. Enquanto no Dash você teria que fazer alguns ajustes para usar gráficos do Matplotlib, no Streamlit essa integração é fluida. Você pode facilmente mesclar visualizações de várias bibliotecas no seu dashboard.

### Criando Gráficos

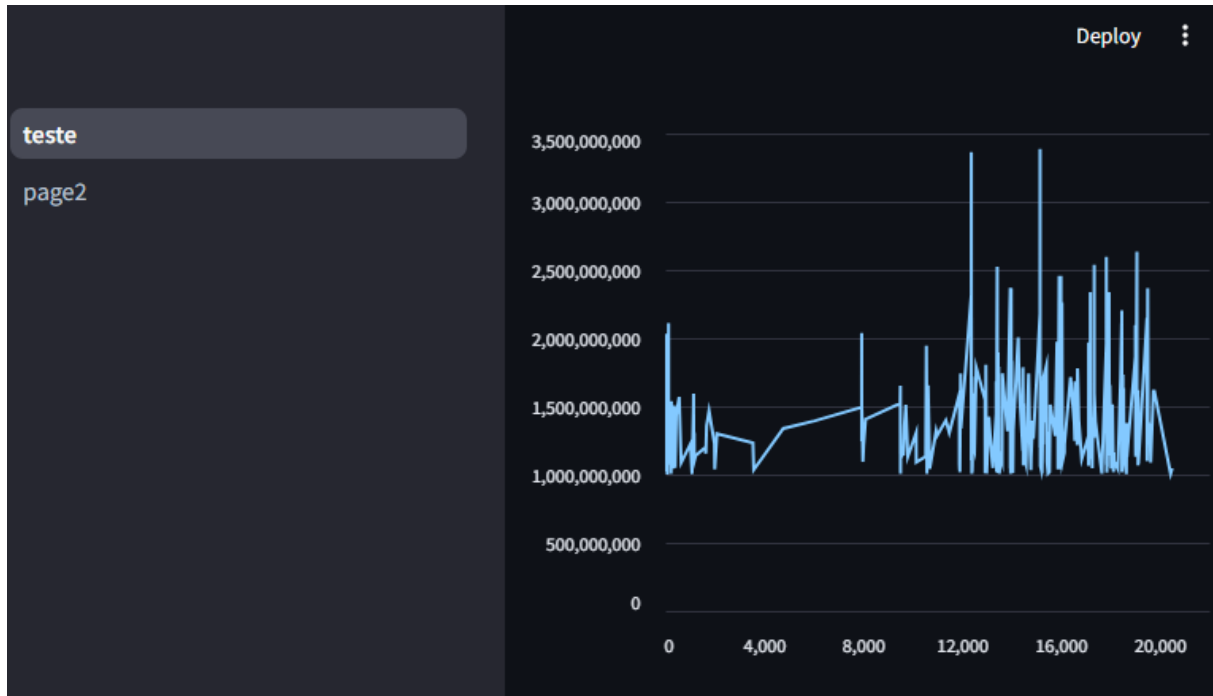
Vamos ver agora como criar um gráfico de linhas simples. O código abaixo será utilizado para representar a quantidade de streams das músicas que tiveram mais de um bilhão de visualizações:

```
import streamlit as st

df = pd.read_csv('caminho/para/seu/arquivo.csv')

# Supondo que 'df' é o DataFrame carregado
st.line_chart(df[df['Stream'] > 1_000_000_000]['Stream'])
```

Nesta seção, o gráfico é criado utilizando a função `st.line_chart()`, permitindo visualizar as informações de maneira clara e rápida.



**Figure 2:** Imagem com o resultado da tela

### Ajustando o Eixo X

Um dos pontos que precisamos ajustar é o eixo X do gráfico, que renderiza números de índices por padrão. Se quisermos que o eixo X mostre o nome da banda, precisamos definir a coluna correspondente como índice do DataFrame, utilizando o comando `set_index()`.

```
df.set_index('Artist', inplace=True)
```

### Construção do Dashboard

Sempre que você adicionar elementos ao Streamlit, a construção do seu dashboard deve ocorrer de cima para baixo. Isso significa que, se adicionar um DataFrame e, em seguida, um texto, o texto aparecerá abaixo do DataFrame. O Streamlit também é eficiente em re-renderizar apenas os componentes que mudaram, evitando consumo excessivo de recursos computacionais.

## 04. Input Widgets

Nesta aula, serão abordados os widgets do Streamlit, que são componentes essenciais para permitir a interação e filtragem de dados nos aplicativos. Os widgets de entrada utilizados incluem:

- **Radio Buttons**
- **Select Box**
- **Multi-select**
- **Text Input**

Entender como cada um deles funciona é fundamental, pois as lógicas de uso são bem semelhantes.

### Trabalhando com Dados

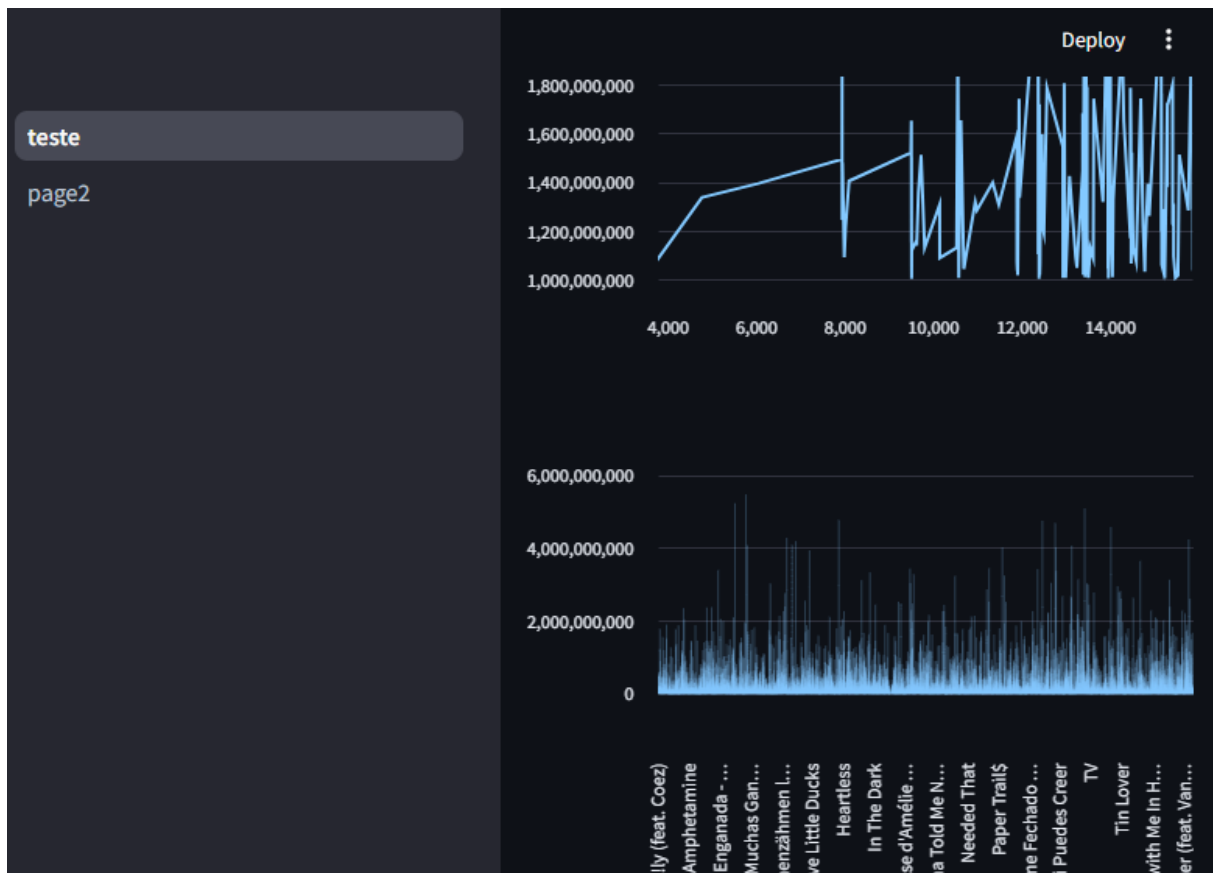
Será dada continuidade ao código desenvolvido na aula anterior, onde foi feita uma seleção de músicas que tocaram mais de um bilhão de vezes no Spotify. Será adicionado um gráfico de barras para visualizar as músicas por artista e a quantidade de vezes que foram tocadas.

Para começar, deve-se adicionar um gráfico de barras no código. Aqui está como isso pode ser feito:

```
import streamlit as st
import pandas as pd

df = pd.read_csv('caminho/para/seu/arquivo.csv')

# Supondo que 'df' é o DataFrame carregado
st.bar_chart(data=df.set_index('Track')['Stream'])
```



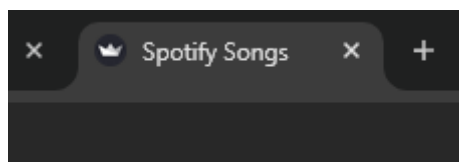
**Figure 3:** Imagem de Exemplo Bar Chart

## Configurando a Página

Para melhorar a apresentação do nosso dashboard, podemos ajustar algumas configurações da página. Utilize o seguinte código para definir o layout e o título da página:

```
st.set_page_config(page_title="Spotify Songs", layout="wide")
```

Ao aplicar esta configuração, você verá que o título da página irá mudar.



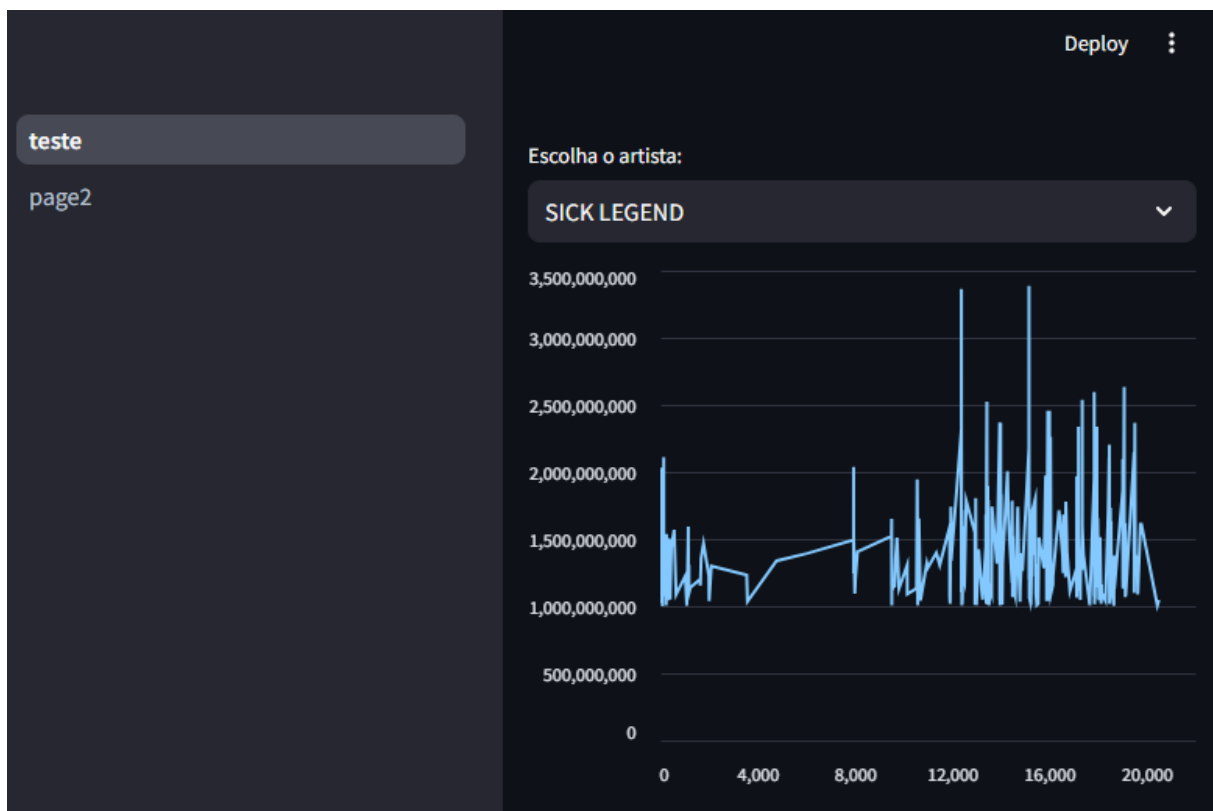
**Figure 4:** Imagem do nome da página

## Selecionando Artistas com Select Box

Agora, vamos criar um Select Box que permitirá ao usuário selecionar um artista. Isso requer a definição de uma lista de artistas a partir de nosso DataFrame.

```
df = pd.read_csv('caminho/para/seu/arquivo.csv')

artists = df['Artist'].value_counts().index.tolist()
selected_artist = st.selectbox('Escolha o artista:', artists)
```



**Figure 5:** Imagem de Exemplo Tab Artistas

Com isso, o Select Box será preenchido com os nomes dos artistas disponíveis e o usuário poderá escolher um deles.

## Filtrando Dados com Base na Seleção

Para conectar o Select Box aos dados que queremos apresentar, utilizamos a lógica que segue:

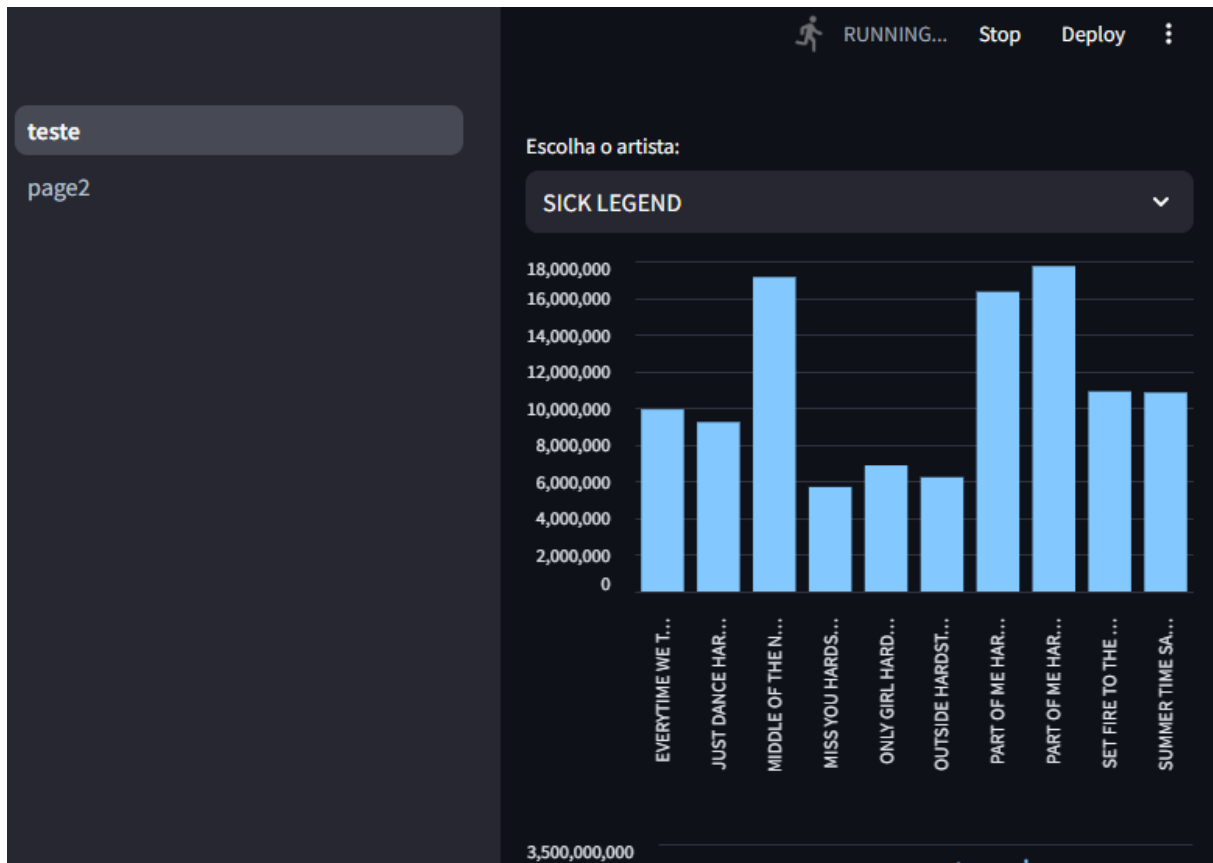
1. Filtrar o DataFrame ao selecionar um artista.

2. Atualizar a visualização dos dados automaticamente quando uma seleção for feita.

Aqui está como você pode configurar isso:

```
df = pd.read_csv('caminho/para/seu/arquivo.csv')

filtered_data = df[df['Artist'] == selected_artist]
st.bar_chart(data=filtered_data.set_index('Track')['Stream'])
```



**Figure 6:** Imagem de Exemplo do Gráfico Artistas com Filtro

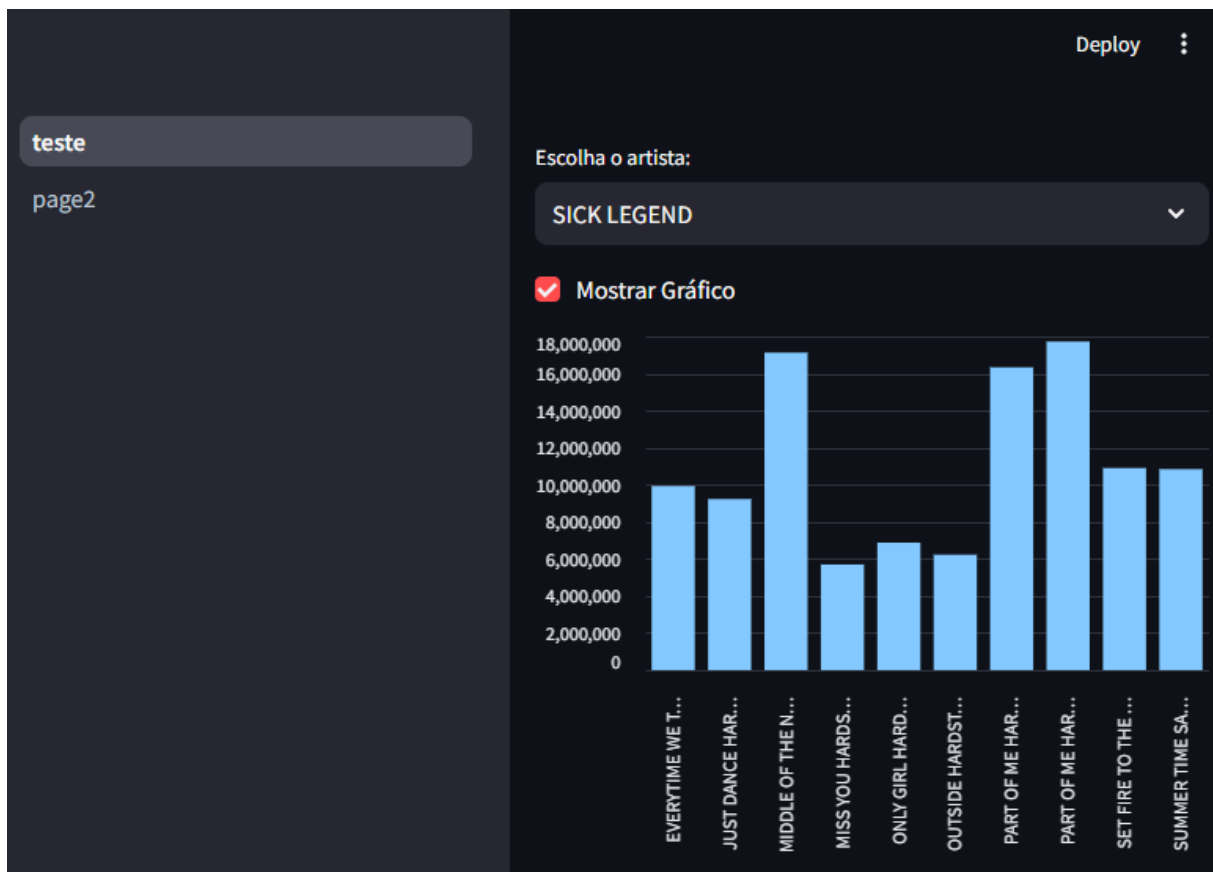
Essa simplicidade é uma das grandes vantagens do Streamlit, pois não precisamos escrever funções complexas como callbacks.

### Checkbox para Controlar a Exibição

Outra funcionalidade que podemos adicionar é um checkbox que controla a visibilidade do gráfico. Utilize o seguinte código:

```
df = pd.read_csv('caminho/para/seu/arquivo.csv')

filtered_data = df[df['Artist'] == selected_artist]
display_graph = st.checkbox('Mostrar Gráfico')
if display_graph:
    st.bar_chart(filtered_data.set_index('Track')['Stream'])
```



**Figure 7:** Imagem de Exemplo de exibição do Gráfico com checkbox

Dessa forma, o gráfico será exibido apenas se o checkbox estiver marcado.

### Composição de Seletores

Podemos também criar uma composição de seletores, onde o segundo seletor depende da escolha feita no primeiro. Por exemplo, ao selecionar um artista, o segundo Select Box mostrará os álbuns desse artista.

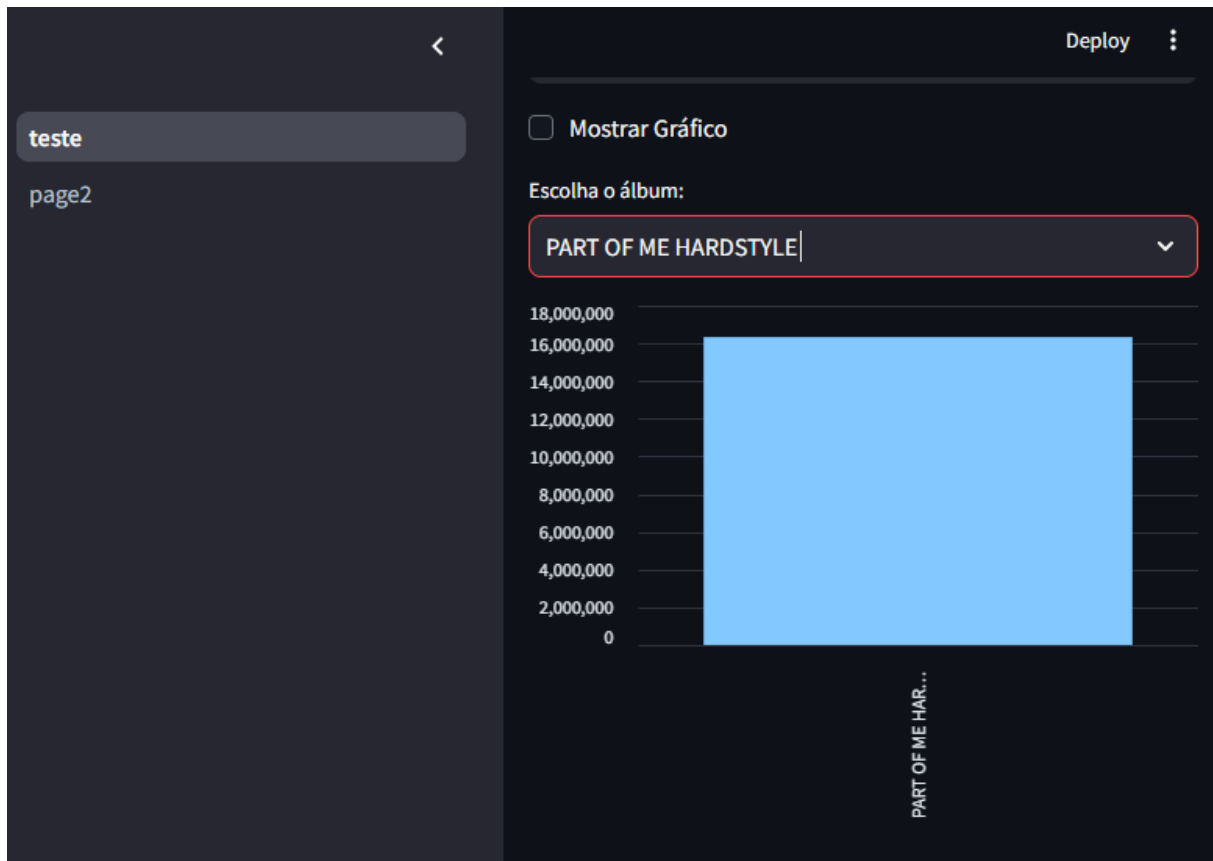
```
df = pd.read_csv('caminho/para/seu/arquivo.csv')
```



```
filtered_data = df[df['Artist'] == selected_artist]

albums = filtered_data['Album'].unique().tolist()
selected_album = st.selectbox('Escolha o álbum:', albums)

# Filtrar músicas no álbum selecionado
final_filtered_data = filtered_data[filtered_data['Album'] == selected_album]
st.bar_chart(final_filtered_data.set_index('Track')['Stream'])
```



**Figure 8:** Imagem de Exemplo do Gráfico de album

## 05. Layout no Streamlit

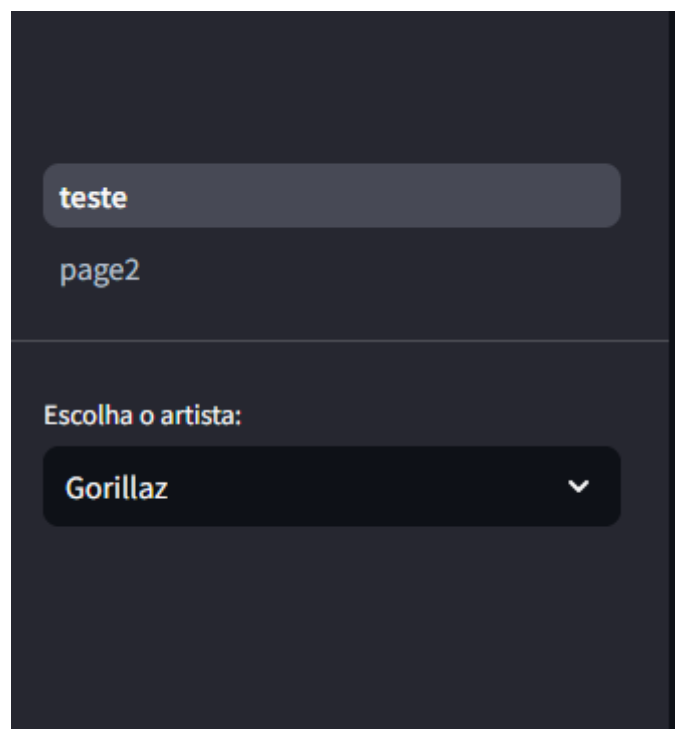
Neste módulo, serão explorados os elementos básicos do Streamlit relacionados ao layout, ou seja, como dispor melhor as informações na tela. Embora o Streamlit não ofereça tantas opções de customização quanto outras bibliotecas, como o Plotly Dash, ele possui funcionalidades que facilitam a organização dos dados de maneira eficiente.

### Sidebar

Uma das funcionalidades que podemos utilizar é a **sidebar**, uma área lateral da interface que é bastante útil para adicionar elementos de controle, como seletores e filtros. Para acessá-la, basta usar o prefixo `st.sidebar` ao definir um widget. Por exemplo, podemos mover um Select Box para a sidebar assim:

```
df = pd.read_csv('caminho/para/seu/arquivo.csv')

selected_artist = st.sidebar.selectbox('Escolha o artista:', df['artista'].unique())
```



**Figure 9:** Exemplo da SideBar

Dessa forma, a seleção do artista ficará à esquerda e o gráfico poderá ser exibido na parte principal do layout.

## Colunas

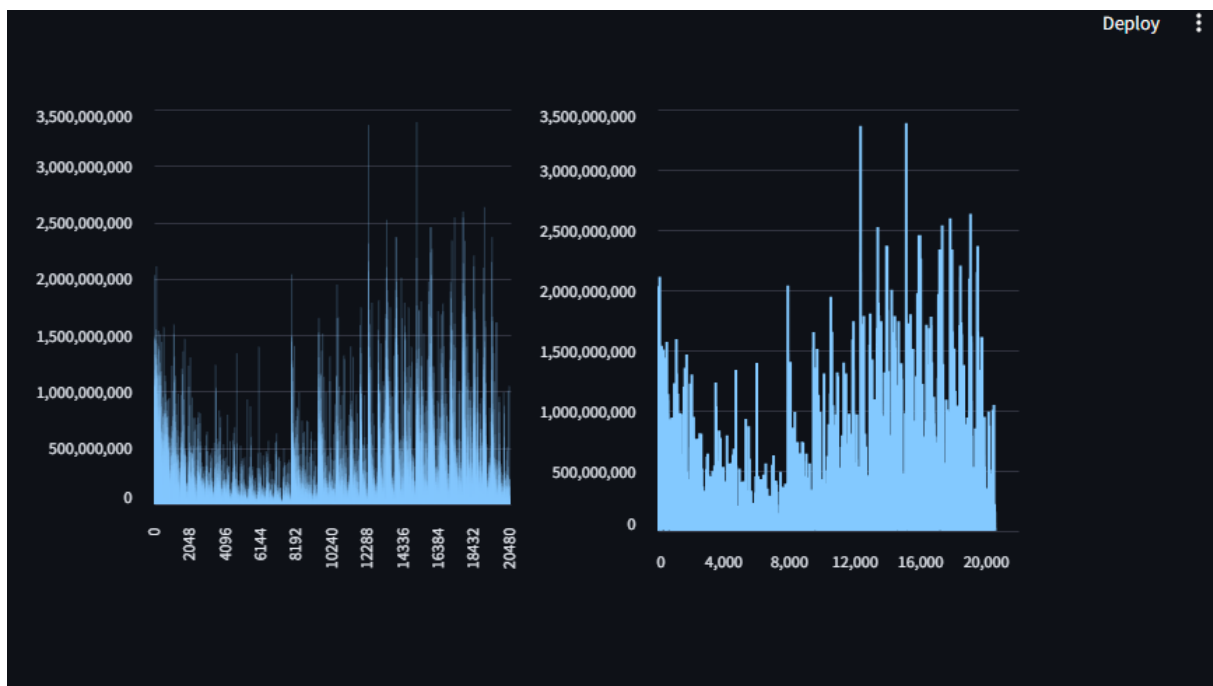
O Streamlit também permite a criação de colunas, facilitando a disposição de diferentes elementos lado a lado. Para criar colunas, utilizamos `st.columns()`. É simples! Veja como fazer:

```
df = pd.read_csv('caminho/para/seu/arquivo.csv')
col1, col2 = st.columns(2) # Criar duas colunas lado a lado

with col1:
    st.bar_chart(df["Stream"])

with col2:
    st.line_chart(df["Stream"])
```

Com esse código, o gráfico de barras aparecerá na coluna 1 e o gráfico de linhas na coluna 2. Esta abordagem é útil quando queremos apresentar informações comparativas.



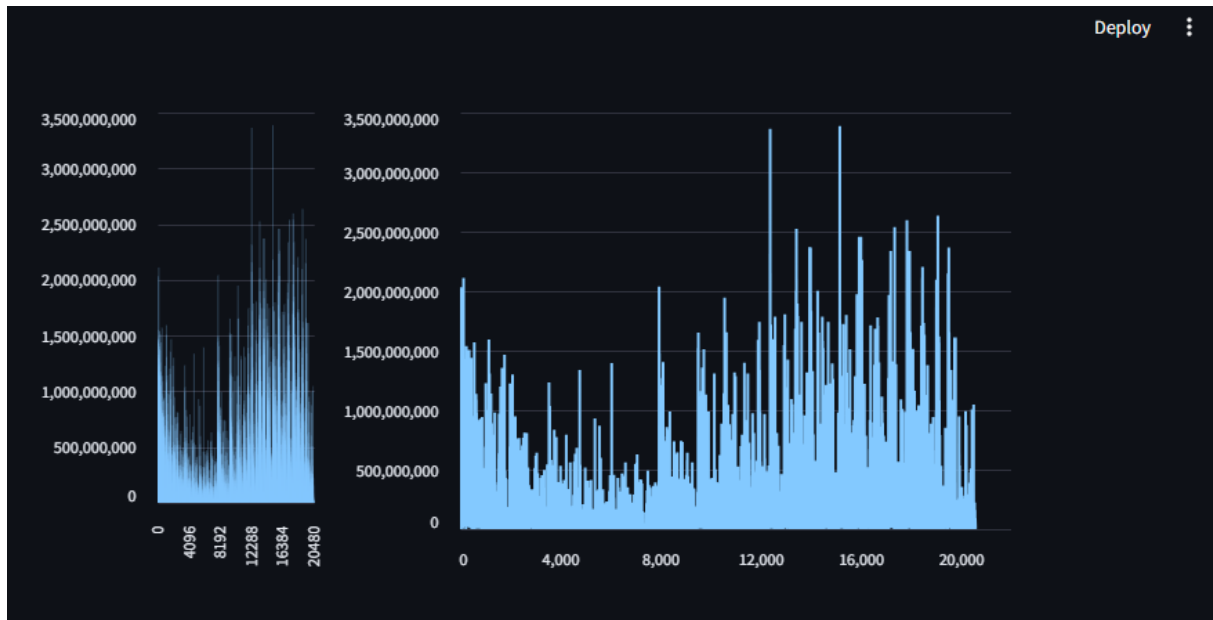
**Figure 10:** Exemplo de Coluna

## Personalizando a Largura das Colunas

Além de criar colunas do mesmo tamanho, o Streamlit permite que você defina exatamente quanto espaço cada coluna deve ocupar. Você pode passar uma lista de proporções para o método `st.columns()`. Por exemplo:

```
col1, col2 = st.columns([0.3, 0.7]) # A primeira coluna ocupa 30%, e a segunda ocupa 70%
```

Essa flexibilidade é útil para otimizar o layout de acordo com o tipo de informações que você está apresentando.



**Figure 11:** Exemplo de Coluna com Size Hint

## 06. Cacheamento e Multipages

Agora será explorado um importante componente: o caching. O cache é essencial para gerenciar informações de forma eficiente nos aplicativos. Além disso, será abordado como construir aplicativos com múltiplas páginas, complementando a compreensão sobre o uso do cache.

### O que é Caching?

Caching é um mecanismo que permite armazenar informações que foram processadas anteriormente, evitando que elas sejam recalculadas ou lidas novamente, o que pode economizar tempo e recursos. Utilizando caching, conseguimos compartilhar dados entre páginas diferentes do nosso aplicativo.

### Criando Múltiplas Páginas

Para organizar melhor o aplicativo Streamlit e permitir a navegação entre diferentes seções, é possível utilizar o recurso de **múltiplas páginas**. Isso facilita a criação de dashboards modulares, onde cada parte do projeto pode ser separada em arquivos distintos.

A estrutura de pastas para um projeto com múltiplas páginas no Streamlit pode ser organizada da seguinte forma:

```
meu_projeto/  
|--- spotify.py  
|--- pages/  
|    |--- page2.py
```

### Explicação da Estrutura:

- **spotify.py**: O arquivo principal do projeto. Este será o ponto de entrada do Streamlit e onde o código inicial do dashboard será executado.
- **Pasta pages/**: Diretório onde ficarão os arquivos adicionais que representarão outras páginas do aplicativo. O Streamlit reconhece automaticamente essa pasta e adiciona os arquivos como opções de navegação.
- **page2.py**: Um arquivo dentro da pasta pages/, que representará uma página separada do aplicativo. Pode conter gráficos, tabelas ou qualquer outro conteúdo que complementa o dashboard principal.

## Armazenando Informações com SessionState

Ao trabalhar com aplicativos na web, cada usuário possui seu próprio estado (state). O Streamlit utiliza **SessionState** para armazenar informações relevantes a cada sessão de usuário. Isso significa que alterações realizadas por um usuário específico não afetarão o que outro usuário está visualizando.

Para armazenar um DataFrame, por exemplo, podemos fazer o seguinte:

```
df = pd.read_csv('caminho/para/seu/arquivo.csv')
st.session_state['df_spotify'] = df
```

Dessa forma, a variável `df` ficará acessível em outras páginas do aplicativo, podendo ser acessada por `st.session_state['df_spotify']`.

## Implementando Cache em Funções

Se você possui uma operação que é demorada, como a leitura de um grande arquivo CSV, você pode protegê-la com caching. Para isso, você deve definir uma função e usar o decorador `@st.cache`. Assim, uma vez processada, a informação será armazenada e não precisará ser recalculada a cada execução.

Aqui está um exemplo de como fazer isso:

```
@st.cache
def load_data():
    df = pd.read_csv('caminho/para/arquivo.csv')
    return df
```

## Simulando Operações Pesadas

Para demonstrar o funcionamento do caching, podemos simular uma operação demorada utilizando o módulo `time` do Python. Veja o exemplo:

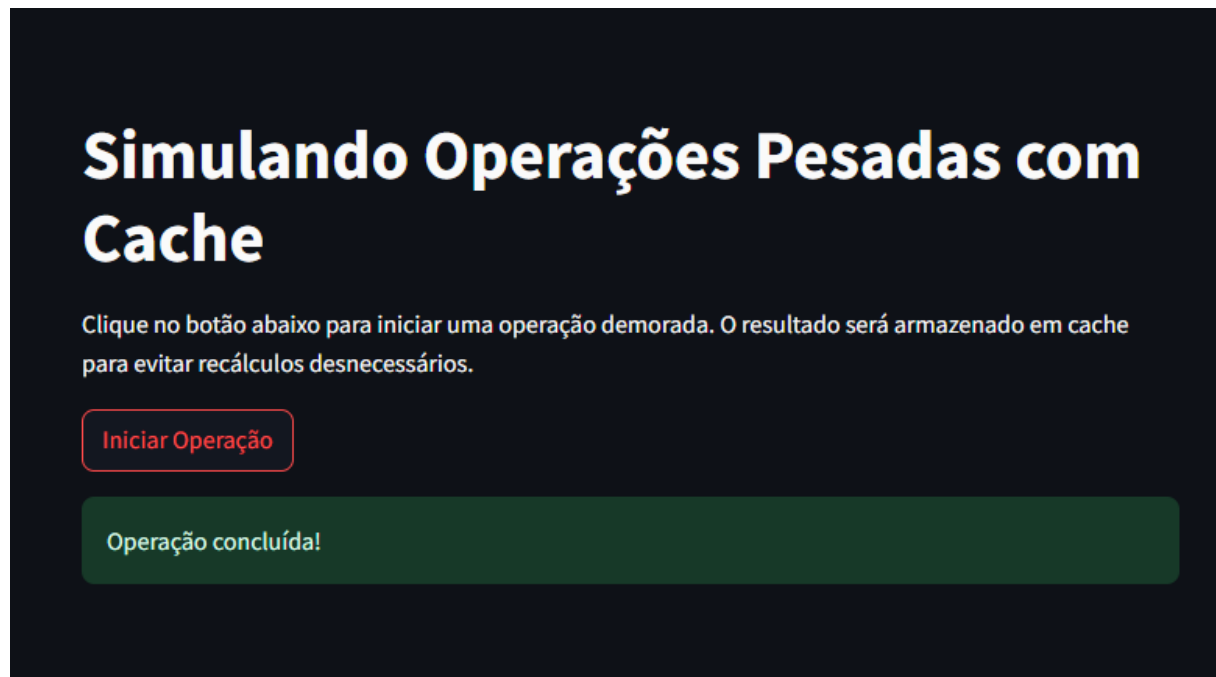
```
import streamlit as st
import time

# Cacheando a função para evitar recomputação desnecessária
@st.cache_data
def heavy_computation():
    time.sleep(5) # Simula uma operação demorada
    return "Operação concluída!"

# Interface do Streamlit
st.title("Simulando Operações Pesadas com Cache")

st.write("Clique no botão abaixo para iniciar uma operação demorada. O resultado será  
↪ armazenado em cache para evitar recálculos desnecessários.")
```

```
if st.button("Iniciar Operação"):
    with st.spinner("Processando..."):
        result = heavy_computation()
    st.success(result)
```



**Figure 12:** Exemplo do Calculo Concluído

### Como funciona

1. O usuário clica no botão “Iniciar Operação”.
2. A função `heavy_computation()` é chamada, simulando um processamento de 5 segundos.
3. O `st.spinner("Processando...")` exibe um indicador visual de carregamento enquanto a função está sendo executada.
4. Quando o processamento termina, a mensagem “Operação concluída!” é exibida com `st.success()`.
5. Graças ao `@st.cache_data`, se o usuário clicar no botão novamente, o resultado será recuperado do cache e exibido instantaneamente, sem precisar esperar novamente.

Esse é um ótimo exemplo para demonstrar como o caching melhora o desempenho em aplicativos Streamlit!

## 07. Componentes Adicionais

Embora o Streamlit seja relativamente novo, tem sido adotado rapidamente pela comunidade, que tem desenvolvido uma variedade de plugins e bibliotecas para enriquecer ainda mais os projetos.

### Navegação e Recursos

Você pode acessar recursos e componentes adicionais diretamente no site do Streamlit, que oferece uma seção dedicada a eles. [Aqui](#), encontramos diversos componentes desenvolvidos pela comunidade com funcionalidades interessantes, como:

- Renderização de gráficos
- Exibição de vídeos
- Criação de mapas
- Componentes personalizáveis

Essas funcionalidades adicionais permitem que você crie dashboards ainda mais interativos e atraentes.

### Exemplos de Componentes

Aqui estão alguns componentes adicionais que você pode utilizar:

1. **Streamlit Extras**: Essa biblioteca permite adicionar anotações nos textos exibidos no seu app.
2. **Streamlit Player**: Esta biblioteca permite incorporar um player de vídeo na sua aplicação, possibilitando uma experiência multimídia no seu dashboard.
3. **Drawable Canvas**: Um componente que oferece a capacidade de desenhar diretamente na tela, podendo ser utilizado para aplicações como reconhecimento de texto ou edição de imagens.

### Conclusão

Ao longo desta apostila, foram explorados os principais conceitos do Streamlit e como essa ferramenta facilita a criação de dashboards interativos e intuitivos. Foram abordados desde a instalação e estruturação do projeto até a utilização de componentes avançados para tornar os aplicativos mais dinâmicos.

Com esse conhecimento, é possível desenvolver aplicações personalizadas, explorando os recursos disponíveis e adaptando-os conforme a necessidade. O Streamlit continua evoluindo, sendo



recomendável acompanhar as novidades e a comunidade para descobrir novas possibilidades. Agora, basta aplicar os conceitos aprendidos e criar dashboards eficientes e inovadores.