

# Guia de Estudo Streamlit

## Documentação Completa para Desenvolvimento de Aplicações

---

### 1. Introdução ao Streamlit

#### O que é o Streamlit?

O Streamlit é um **framework Python open-source** desenvolvido especificamente para cientistas de dados e engenheiros de AI/ML. Permite criar e compartilhar aplicações web dinâmicas com apenas algumas linhas de código.

#### Características Principais:

- **Simplicidade:** Apps em minutos, não dias
  - **Python Puro:** Não requer conhecimento de HTML, CSS ou JavaScript
  - **Interativo:** Widgets nativos e reatividade automática
  - **Deploy Fácil:** Integração com GitHub e deploy gratuito
  - **Comunidade Ativa:** Ampla biblioteca de componentes customizados
- 

### 2. Instalação e Configuração

#### Instalação Básica

```
bash

# Via pip
pip install streamlit

# Via conda
conda install streamlit

# Verificar instalação
streamlit hello
```

#### Configuração do Ambiente

```
bash
```

```
# Criar ambiente virtual
python -m venv streamlit_env
source streamlit_env/bin/activate # Linux/Mac
streamlit_env\Scripts\activate   # Windows

# Instalar dependências
pip install streamlit pandas numpy matplotlib plotly
```

## Estrutura de Projeto

```
meu_projeto/
├── .streamlit/
│   └── config.toml
├── pages/
│   ├── page_1.py
│   └── page_2.py
├── requirements.txt
├── secrets.toml
└── main_app.py
```

## 3. Conceitos Fundamentais

### Modelo de Execução

O Streamlit usa um **modelo de execução top-down**:

- O script é executado do início ao fim a cada interação
- Estado é mantido através de `st.session_state`
- Reatividade automática quando inputs mudam

### Primeiro App

```
python

import streamlit as st

st.title("Meu Primeiro App")
st.write("Olá, mundo!")

name = st.text_input("Seu nome:")
if name:
    st.write(f"Olá, {name}!")
```

## Executando o App

```
bash  
  
streamlit run main_app.py
```

---

## 4. Elementos de Display

### Texto e Markdown

```
python  
  
# Títulos e headers  
st.title("Título Principal")  
st.header("Header")  
st.subheader("Subheader")  
  
# Texto formatado  
st.markdown("***Texto em negrito** e *itálico*")  
st.text("Texto simples")  
st.caption("Texto pequeno")  
  
# Código  
st.code("print('Hello World')", language='python')  
  
# LaTeX  
st.latex(r"\int a x^2 \, dx")  
  
# Divisor  
st.divider()
```

### Dados e Tabelas

```
python
```

```
import pandas as pd
```

```
# DataFrame interativo
```

```
df = pd.DataFrame({'col1': [1, 2, 3], 'col2': [4, 5, 6]})  
st.dataframe(df)
```

```
# Editor de dados
```

```
edited_df = st.data_editor(df, num_rows="dynamic")
```

```
# Tabela estática
```

```
st.table(df)
```

```
# Métricas
```

```
st.metric("Vendas", "R$ 1,200", "12%")
```

```
# JSON
```

```
st.json({"key": "value", "nested": {"data": 123}})
```

## Mídia

```
python
```

```
# Imagens
```

```
st.image("imagem.png", caption="Minha imagem")  
st.logo("logo.png") # Logo no canto superior
```

```
# Áudio
```

```
st.audio("audio.mp3")
```

```
# Vídeo
```

```
st.video("video.mp4")
```

---

## 5. Widgets de Entrada

### Botões

```
python
```

*# Botão simples*

```
if st.button("Clique aqui"):
    st.success("Botão pressionado!")
```

*# Botão de download*

```
st.download_button("Download CSV", data=csv_data,
                    file_name="dados.csv", mime="text/csv")
```

*# Link button*

```
st.link_button("Ir para Google", "https://google.com")
```

*# Toggle*

```
ativo = st.toggle("Ativar funcionalidade")
```

## Seleção

python

*# Checkbox*

```
concordo = st.checkbox("Eu concordo com os termos")
```

*# Radio buttons*

```
opcao = st.radio("Escolha uma opção:", ["A", "B", "C"])
```

*# Selectbox*

```
cidade = st.selectbox("Sua cidade:", ["SP", "RJ", "MG"])
```

*# Multiselect*

```
cores = st.multiselect("Cores favoritas:",
                        ["Vermelho", "Azul", "Verde"])
```

*# Pills (seleção em formato de pílulas)*

```
tags = st.pills("Tags:", ["Python", "ML", "Web"])
```

*# Segmented control*

```
status = st.segmented_control("Status:", ["Ativo", "Inativo"])
```

## Entrada de Dados

python

*# Texto*

```
nome = st.text_input("Nome:")
```

```
descricao = st.text_area("Descrição:")
```

*# Números*

```
idade = st.number_input("Idade:", min_value=0, max_value=120)
```

```
preco = st.slider("Preço:", 0, 1000, 500)
```

*# Data e hora*

```
data = st.date_input("Data:")
```

```
hora = st.time_input("Hora:")
```

*# Upload de arquivos*

```
arquivo = st.file_uploader("Escolha um arquivo",  
                             type=['csv', 'xlsx', 'txt'])
```

*# Câmera*

```
foto = st.camera_input("Tire uma foto")
```

*# Áudio*

```
gravacao = st.audio_input("Grave uma mensagem")
```

*# Color picker*

```
cor = st.color_picker("Escolha uma cor")
```

---

## 6. Gráficos e Visualizações

### Gráficos Nativos

```
python
```

```
import pandas as pd
import numpy as np

# Dados de exemplo
df = pd.DataFrame({
    'x': np.arange(10),
    'y': np.random.randn(10)
})

# Gráficos simples
st.line_chart(df)
st.area_chart(df)
st.bar_chart(df)
st.scatter_chart(df)

# Mapas
map_data = pd.DataFrame({
    'lat': [-23.550, -22.906],
    'lon': [-46.634, -43.172]
})
st.map(map_data)
```

## Integrações com Bibliotecas

```
python

import matplotlib.pyplot as plt
import plotly.express as px
import altair as alt

# Matplotlib
fig, ax = plt.subplots()
ax.plot([1, 2, 3, 4])
st.pyplot(fig)

# Plotly
fig = px.bar(df, x='x', y='y')
st.plotly_chart(fig)

# Altair
chart = alt.Chart(df).mark_circle().encode(x='x', y='y')
st.altair_chart(chart)
```

---

## 7. Layout e Containers

## Colunas

```
python

# Criando colunas
col1, col2, col3 = st.columns(3)

with col1:
    st.header("Coluna 1")
    st.write("Conteúdo da coluna 1")

col2.header("Coluna 2")
col3.header("Coluna 3")

# Colunas com proporções diferentes
col1, col2 = st.columns([2, 1]) # 2:1 ratio
```

## Containers

```
python

# Container genérico
container = st.container()
st.write("Isso aparece depois")
container.write("Isso aparece primeiro")

# Empty container (placeholder)
placeholder = st.empty()
placeholder.text("Carregando...")
# ... processamento ...
placeholder.success("Concluído!")

# Expander (seção expansível)
with st.expander("Ver detalhes"):
    st.write("Detalhes aqui...")

# Sidebar
st.sidebar.title("Menu Lateral")
st.sidebar.selectbox("Opções", ["A", "B", "C"])

# Tabs
tab1, tab2, tab3 = st.tabs(["Tab 1", "Tab 2", "Tab 3"])
with tab1:
    st.write("Conteúdo da tab 1")
```



## Popover e Dialog

python

*# Popover*

```
with st.popover("Configurações"):
    st.checkbox("Opção 1")
    st.slider("Valor", 0, 100)
```

*# Modal Dialog*

```
@st.dialog("Formulário de Cadastro")
```

```
def cadastro():
```

```
    nome = st.text_input("Nome")
```

```
    email = st.text_input("Email")
```

```
    if st.button("Cadastrar"):
```

```
        st.success("Cadastrado com sucesso!")
```

```
if st.button("Abrir formulário"):
```

```
    cadastro()
```

---

## 8. Elementos de Chat e Status

### Chat Interface

python

```
# Input de chat
prompt = st.chat_input("Digite sua mensagem")

if prompt:
    # Mensagem do usuário
    with st.chat_message("user"):
        st.write(prompt)

    # Resposta do assistente
    with st.chat_message("assistant"):
        st.write("Resposta para: " + prompt)

# Stream de texto (efeito máquina de escrever)
def response_generator():
    response = "Esta resposta aparece letra por letra..."
    for word in response.split():
        yield word + " "
        time.sleep(0.1)

st.write_stream(response_generator())
```

## Indicadores de Status

python

```

import time

# Progress bar
progress = st.progress(0)
for i in range(100):
    progress.progress(i + 1)
    time.sleep(0.01)

# Spinner
with st.spinner("Processando..."):
    time.sleep(3)
    st.success("Concluído!")

# Status container
with st.status("Executando processo...") as status:
    st.write("Passo 1: Carregando dados")
    time.sleep(1)
    st.write("Passo 2: Processando")
    time.sleep(1)
    status.update(label="Processo concluído!", state="complete")

# Toast notifications
st.toast("Operação realizada com sucesso!", icon="✅")

# Celebrações
st.balloons() # Balões
st.snow()     # Neve

```

## Alertas e Notificações

```

python

# Diferentes tipos de alertas
st.success("Sucesso!")
st.info("Informação importante")
st.warning("Atenção!")
st.error("Erro encontrado!")

# Exception display
try:
    resultado = 1/0
except Exception as e:
    st.exception(e)

```

## 9. Estado e Cache

### Session State

```
python

# Inicializar estado
if 'contador' not in st.session_state:
    st.session_state.contador = 0

# Usar e modificar estado
if st.button("Incrementar"):
    st.session_state.contador += 1

st.write(f"Contador: {st.session_state.contador}")

# Widget com key (automaticamente salvo no estado)
nome = st.text_input("Nome", key="user_name")
# Acessível via st.session_state.user_name
```

### Cache de Dados

```
python

@st.cache_data
def load_data(file_path):
    """Cache para transformações de dados"""
    return pd.read_csv(file_path)

@st.cache_resource
def init_model():
    """Cache para recursos globais (modelos ML, conexões DB)"""
    # Carregamento pesado do modelo
    return model

# TTL Cache
@st.cache_data(ttl=3600) # Cache por 1 hora
def get_live_data():
    return fetch_from_api()
```

### Query Parameters

```
python
```

```
# Ler parâmetros da URL
```

```
if 'page' in st.query_params:  
    current_page = st.query_params['page']
```

```
# Definir parâmetros
```

```
st.query_params['page'] = 'dashboard'
```

```
st.query_params.clear() # Limpar todos
```

---

## 10. Formulários e Validação

### Formulários Básicos

```
python
```

```
# Formulário previne rerun a cada mudança
```

```
with st.form("meu_formulario"):
    nome = st.text_input("Nome")
    idade = st.number_input("Idade", min_value=0)
    email = st.text_input("Email")

    submitted = st.form_submit_button("Enviar")

    if submitted:
        if not nome or not email:
            st.error("Preencha todos os campos obrigatórios")
        else:
            st.success(f"Dados salvos para {nome}")
```

```
# Formulário com validação customizada
```

```
with st.form("formulario_avancado"):
    col1, col2 = st.columns(2)

    with col1:
        primeiro_nome = st.text_input("Primeiro Nome")
        data_nascimento = st.date_input("Data de Nascimento")

    with col2:
        ultimo_nome = st.text_input("Último Nome")
        telefone = st.text_input("Telefone")

    aceita_termos = st.checkbox("Aceito os termos de uso")

    if st.form_submit_button("Cadastrar"):
        erros = []

        if not primeiro_nome.strip():
            erros.append("Primeiro nome é obrigatório")

        if not aceita_termos:
            erros.append("Você deve aceitar os termos")

        if erros:
            for erro in erros:
                st.error(erro)
        else:
            st.success("Cadastro realizado com sucesso!")
```

---

## 11. Navegação e Páginas Múltiplas

## Estrutura Multi-página Simples

```
python

# pages/home.py
import streamlit as st

st.title("Página Inicial")
st.write("Bem-vindo ao app!")

# pages/dashboard.py
import streamlit as st

st.title("Dashboard")
st.line_chart([1, 3, 2, 4, 5])
```

## Navegação Programática

```
python

# Definir páginas
home_page = st.Page("pages/home.py", title="Home", icon="🏠")
dashboard_page = st.Page("pages/dashboard.py", title="Dashboard", icon="📊")
settings_page = st.Page("pages/settings.py", title="Configurações", icon="⚙️")

# Criar navegação
pg = st.navigation({
    "Principal": [home_page, dashboard_page],
    "Configurações": [settings_page]
})

# Executar página atual
pg.run()

# Link para outras páginas
st.page_link("pages/dashboard.py", label="Ver Dashboard", icon="📊")

# Navegar programaticamente
if st.button("Ir para Dashboard"):
    st.switch_page("pages/dashboard.py")
```

## Navegação Top Bar

```
python
```

```
# Navegação no topo da página
```

```
st.navigation([home_page, dashboard_page], position="top")
```

## 12. Conexões e Dados

### Conexões Built-in

```
python
```

```
# Conexão SQL
```

```
conn = st.connection('my_database', type='sql',  
                      url="sqlite:///database.db")
```

```
df = conn.query('SELECT * FROM users LIMIT 10')  
st.dataframe(df)
```

```
# Conexão Snowflake
```

```
snow_conn = st.connection('snowflake')  
snow_df = snow_conn.query('SELECT * FROM my_table')
```

### Conexões Customizadas

```
python
```

```
from streamlit.connections import BaseConnection  
import requests
```

```
class APIConnection(BaseConnection[requests.Session]):  
    def _connect(self, **kwargs) -> requests.Session:  
        session = requests.Session()  
        session.headers.update({  
            'Authorization': f'Bearer {self._secrets["api_token"]}'  
        })  
        return session  
  
    def query(self, endpoint: str):  
        return self._instance.get(f"{self._secrets['base_url']}/{endpoint}")
```

```
# Usar conexão customizada
```

```
api_conn = st.connection('my_api', type=APIConnection)  
data = api_conn.query('users')
```



## Gerenciamento de Secrets

```
python

# .streamlit/secrets.toml
"""

[connections.my_database]
url = "postgresql://user:pass@localhost:5432/db"

[api]
token = "your-secret-token"
"""

# Acessar secrets
db_url = st.secrets["connections"]["my_database"]["url"]
api_token = st.secrets["api"]["token"]
```

---

## 13. Configuração Avançada

### Arquivo config.toml

```
toml
```

```
# .streamlit/config.toml

[global]
developmentMode = false

[theme]
primaryColor = "#FF6B6B"
backgroundColor = "#FFFFFF"
secondaryBackgroundColor = "#F0F2F6"
textColor = "#262730"
font = "sans serif"

[server]
port = 8501
headless = true
enableCORS = false
enableXsrfProtection = false

[browser]
gatherUsageStats = false
serverAddress = "localhost"

[mapbox]
token = "your-mapbox-token"
```

## Configuração de Página

```
python

st.set_page_config(
    page_title="Meu App",
    page_icon="🚀",
    layout="wide",      # "centered" ou "wide"
    initial_sidebar_state="expanded", # "auto", "expanded", "collapsed"
    menu_items={
        'Get Help': 'https://help.example.com',
        'Report a bug': "https://bug-report.example.com",
        'About': "Este é um app incrível!"
    }
)
```

## Context e Informações do Ambiente

```
python
```

```
# Informações do contexto
st.write("Cookies:", st.context.cookies)
st.write("Headers:", st.context.headers)

# Detectar tema
if st.context.theme:
    if st.context.theme.base == "dark":
        st.write("Modo escuro ativo")
    else:
        st.write("Modo claro ativo")

# Configurações do app
primary_color = st.get_option("theme.primaryColor")
st.write(f"Cor primária: {primary_color}")
```

## 14. Componentes Customizados

### HTML Customizado

```
python

import streamlit.components.v1 as components

# HTML inline
components.html("""
<div style="background-color: #f0f0f0; padding: 20px;">
    <h2>Componente HTML Customizado</h2>
    <button onclick="alert('Clicado!')">Clique aqui</button>
</div>
""", height=150)

# iFrame
components.iframe("https://example.com", height=500)
```

### Componente Customizado Avançado

```
python
```

```
# Declarar componente customizado
my_component = components.declare_component(
    "my_component",
    path="./frontend" # Pasta com HTML/JS/CSS
)

# Usar componente
result = my_component(
    key="unique_key",
    default_value=0,
    custom_param="value"
)
```

---

## 15. Testes e Debugging

### Testing com AppTest

```
python
```

```
from streamlit.testing.v1 import AppTest
import pytest

def test_basic_app():
    # Testar app a partir de arquivo
    at = AppTest.from_file("app.py")
    at.run()

    # Verificar se não há exceções
    assert not at.exception

    # Verificar elementos
    assert len(at.title) == 1
    assert at.title[0].value == "Meu App"

    # Interagir com widgets
    at.text_input[0].input("João").run()
    assert at.success[0].value == "Bem-vindo, João!"

def test_button_interaction():
    at = AppTest.from_string("""
import streamlit as st

if st.button("Clique"):
    st.success("Clicado!")
""")

    at.run()
    assert len(at.success) == 0 # Sem cliques ainda

    # Simular clique
    at.button[0].click().run()
    assert len(at.success) == 1
    assert at.success[0].value == "Clicado!"
```

## Debugging

```
python
```

```
# Debug info
st.write("Session State:", st.session_state)
st.write("Query Params:", dict(st.query_params))

# Error handling
try:
    resultado = operacao_perigosa()
    st.success("Sucesso!")
except Exception as e:
    st.error(f"Erro: {str(e)}")
    st.exception(e) # Stack trace completo

# Conditional debugging
if st.checkbox("Debug Mode"):
    st.json(debug_data)
    st.write("Variáveis:", locals())
```

## 16. Deploy e Produção

### Preparação para Deploy

```
python

# requirements.txt
"""
streamlit>=1.32.0
pandas>=1.5.0
numpy>=1.24.0
plotly>=5.15.0
"""

# .streamlit/config.toml para produção
"""
[server]
headless = true
port = $PORT
enableCORS = false

[browser]
gatherUsageStats = false
"""
```

## Streamlit Community Cloud

### 1. Preparar Repositório GitHub

- Código da aplicação
- requirements.txt
- .streamlit/config.toml (opcional)
- .streamlit/secrets.toml (não commitar - adicionar via UI)

## 2. Deploy

- Acessar [share.streamlit.io](https://share.streamlit.io)
- Conectar com GitHub
- Selecionar repositório e arquivo principal
- Deploy automático

## 3. Gerenciar App

- Logs em tempo real
- Reboot da aplicação
- Configurações de secrets

## Deploy Alternativo (Docker)

dockerfile

*# Dockerfile*

**FROM** python:3.9-slim

**WORKDIR** /app

**COPY** requirements.txt .

**RUN** pip install -r requirements.txt

**COPY** . .

**EXPOSE** 8501

**CMD** ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]

## Docker Compose

yaml

```
# docker-compose.yml
version: '3.8'
services:
  streamlit:
    build: .
    ports:
      - "8501:8501"
    environment:
      - STREAMLIT_SERVER_PORT=8501
    volumes:
      - ./data:/app/data
```

## 17. Performance e Otimização

### Otimização de Cache

```
python

# Cache com hash personalizado
@st.cache_data
def process_data(df):
    # Processamento pesado
    return df.groupby('category').sum()

# Cache de recursos
@st.cache_resource
def get_expensive_resource():
    # Carregamento pesado (modelo ML, conexão DB)
    return expensive_operation()

# Cache condicional
@st.cache_data
def get_data(use_cache=True):
    if not use_cache:
        st.cache_data.clear() # Limpar cache específico
    return fetch_data()
```

### Lazy Loading

```
python
```



```
# Carregar dados apenas quando necessário
if 'data' not in st.session_state:
    with st.spinner("Carregando dados..."):
        st.session_state.data = load_large_dataset()

# Paginação
page_size = 50
page = st.number_input("Página", min_value=1, value=1)
start_idx = (page - 1) * page_size
end_idx = start_idx + page_size

st.dataframe(st.session_state.data[start_idx:end_idx])
```

## Fragmentos para Performance

```
python

@st.fragment(run_every="30s")
def live_metrics():
    """Atualiza métricas sem recarregar página inteira"""
    current_value = get_real_time_metric()
    st.metric("Valor Atual", current_value)

@st.fragment
def expensive_chart():
    """Componente que só reroda quando necessário"""
    if st.button("Atualizar Gráfico"):
        data = expensive_calculation()
        st.line_chart(data)
```

## 18. Boas Práticas e Padrões

### Estrutura de Código

```
python
```

```

# main.py - Estrutura recomendada
import streamlit as st
from src.data_loader import DataLoader
from src.visualizations import create_dashboard
from src.utils import setup_page

def main():
    setup_page()

    # Sidebar para controles
    with st.sidebar:
        st.header("Configurações")
        data_source = st.selectbox("Fonte", ["Local", "API"])
        date_range = st.date_input("Período")

    # Corpo principal
    if data_source and date_range:
        data = DataLoader(data_source).load(date_range)
        create_dashboard(data)

if __name__ == "__main__":
    main()

```

## Gerenciamento de Estado

```

python

# utils/state.py
class StateManager:
    @staticmethod
    def initialize():
        if 'initialized' not in st.session_state:
            st.session_state.initialized = True
            st.session_state.user_data = {}
            st.session_state.preferences = {}

    @staticmethod
    def get(key, default=None):
        return st.session_state.get(key, default)

    @staticmethod
    def set(key, value):
        st.session_state[key] = value

```

## Separação de Responsabilidades

python

```
# src/components/sidebar.py
```

```
def render_sidebar():
```

```
    with st.sidebar:
```

```
        st.header("Navegação")
```

```
        return {
```

```
            'page': st.selectbox("Página", ["Home", "Analytics", "Config"]),
```

```
            'filters': render_filters()
```

```
        }
```

```
# src/pages/analytics.py
```

```
def render_analytics_page(data, filters):
```

```
    st.header("Analytics Dashboard")
```

```
    col1, col2, col3 = st.columns(3)
```

```
    with col1:
```

```
        render_kpi_cards(data)
```

```
    with col2:
```

```
        render_trends_chart(data, filters)
```

```
    with col3:
```

```
        render_summary_table(data, filters)
```

---

## 19. Integrações Avançadas

### Machine Learning

python

```

import joblib
from sklearn.model_selection import train_test_split

@st.cache_resource
def load_model():
    return joblib.load('modelo.pkl')

def ml_prediction_app():
    st.header("Predições ML")

    # Upload de dados
    uploaded_file = st.file_uploader("Dataset CSV", type="csv")

    if uploaded_file:
        df = pd.read_csv(uploaded_file)
        st.dataframe(df.head())

    # Seleção de features
    features = st.multiselect("Selecione features:", df.columns)
    target = st.selectbox("Target:", df.columns)

    if st.button("Treinar Modelo"):
        with st.spinner("Treinando..."):
            X = df[features]
            y = df[target]

            X_train, X_test, y_train, y_test = train_test_split(
                X, y, test_size=0.2, random_state=42
            )

            model = RandomForestRegressor()
            model.fit(X_train, y_train)

            score = model.score(X_test, y_test)
            st.success(f"Modelo treinado! R²: {score:.3f}")

    # Salvar no estado
    st.session_state.model = model
    st.session_state.features = features

```

## APIs e Webhooks

python

```
import requests
```

```
def api_integration():
```

```
    st.header("Integração com API")
```

```
    api_endpoint = st.text_input("Endpoint da API")
```

```
    api_key = st.text_input("API Key", type="password")
```

```
    if st.button("Buscar Dados"):
```

```
        headers = {'Authorization': f'Bearer {api_key}'}
```

```
        try:
```

```
            response = requests.get(api_endpoint, headers=headers)
```

```
            response.raise_for_status()
```

```
            data = response.json()
```

```
            st.json(data)
```

```
        except requests.exceptions.RequestException as e:
```

```
            st.error(f"Erro na requisição: {
```