ChatGPT

Streamlit 中文教程

Streamlit 是一个开源的 Python 库，可以轻松创建和共享用于机器学习和数据科学的自定义 Web 应用程序。通过使用 Streamlit，可以快速构建和部署强大的数据应用程序[1]。本教程将分六个部分全面介绍 Streamlit 的使用方法，包括基础入门、表单与交互组件、绘图和数据展示、会话状态和回调、布局与页面结构，以及应用部署等主题。

1. 基础入门

安装 Streamlit： 使用 `pip` 来安装 Streamlit。在命令行运行以下命令完成安装：

```
pip install streamlit
```

安装完成后，您可以通过命令行运行 Streamlit 应用。例如，如果您的应用脚本名为 `app.py`，可以执行：

```
streamlit run app.py
```

上述命令将启动一个本地服务器并在浏览器中打开您的 Streamlit 应用[2]。通常我们按照惯例将 Streamlit 库导入为 `st` 来使用，例如：

```
import streamlit as st
```

编写第一个应用： Streamlit 应用通常以调用 `st.title` 来设置应用标题开始[3]。接下来，您可以使用不同的文本显示函数：

- `st.text`：显示纯文本（固定宽度字体）。
- `st.markdown`：显示支持 Markdown 格式的文本（可用 **粗体**、斜体 等语法）。
- `st.write`：一个强大的 “瑞士军刀” 命令，可接受多种数据类型并智能地进行展示[3]。如果传入字符串，`st.write` 会按 Markdown 格式渲染；传入其他对象（如数字、列表、数据框等）也会被友好地展示出来。

下面是一个简单的示例应用，演示上述组件的用法：

```
import streamlit as st

st.title("我的第一个 Streamlit 应用")  # 应用标题
st.text("这是 st.text 输出的纯文本。")  # 输出纯文本
st.markdown("这是 **st.markdown** 输出的 Markdown 文本。")  # 输出带格式的文本
st.write("Hello, *Streamlit!* :sunglasses:")  # st.write 可以直接显示Markdown和各种对象
st.write("支持显示各种类型的数据：", 123, {"key": "value"})  # 传入多个参数和不同类型
```

在运行上述应用后，您将看到应用页面上显示了标题以及不同格式的文本内容。`st.write` 可以接受多个参数，并自动根据类型进行渲染，非常方便。[3]

提示： Streamlit 默认会把脚本中顶级代码按顺序执行并立即显示结果。您无需显式调用 `st.run()` 或类似方法，只要编写脚本并使用 Streamlit 命令，应用启动后将自动渲染所有组件。

2. 表单与交互组件

交互组件是 Streamlit 应用的核心，使应用能够响应用户输入。Streamlit 提供了丰富的**输入组件**（widgets），例如按钮、文本输入框、下拉选择框、滑动条、复选框等。每个组件在页面渲染时会显示相应的控件，并返回一个对应的值。通常，修改输入组件的值会**立即触发应用脚本从头重新运行**，从而更新页面内容。

以下是一些常见交互组件及其用法：

- `st.button(label)`：按钮，点击后返回 `True` 一次。
- `st.text_input(label)`：文本输入框，返回用户输入的字符串。
- `st.selectbox(label, options)`：下拉选择框，从给定选项列表中选择一项，返回所选值。
- `st.slider(label, min, max, value)`：滑动条，可在[min, max]范围内选择数字，返回选定的值。
- `st.checkbox(label)`：复选框，返回布尔值，选中为 True。

示例代码，演示基本的交互组件：

```
import streamlit as st

st.write("请填写您的信息：")
name = st.text_input("姓名")            # 文本输入
age = st.slider("年龄", 0, 120, 25)    # 数字滑动条，默认值25
occupation = st.selectbox("职业", ["学生", "工程师", "医生", "其他"])  # 下拉选择
subscribe = st.checkbox("订阅新闻邮件")  # 复选框

# 提交按钮
if st.button("提交"):
    st.write(f"姓名：{name}")
    st.write(f"年龄：{age}")
    st.write(f"职业：{occupation}")
    st.write(f"订阅邮件：{'是' if subscribe else '否'}")
```

在上述代码中，每当用户修改输入或点击按钮时，应用都会重新运行，并根据最新的输入值更新显示结果。

使用表单（`st.form`）： 默认情况下，Streamlit 的部件交互是即时生效的——例如用户一输入文本或拖动滑块，应用就会重新执行以反映最新值。有时我们希望用户填完**一组输入**后再统一处理，这时可以使用 **表单**。表单是一个特殊的容器，视觉上将多个元素和小部件组合在一起，并包含一个提交按钮。只有当点击提交按钮时，表单内所有小部件的值才会一起发送到应用进行处理[4]。这避免了每次修改单个字段都触发应用更新，提高了交互体验。

使用表单可以通过 `with st.form(...)` 语法创建表单容器，然后在容器内添加输入部件和提交按钮。例如：

```python
import streamlit as st

# 在表单容器内添加部件
with st.form("my_form"):
    st.write("请填写下列信息后提交：")
    username = st.text_input("用户名")
    password = st.text_input("密码", type="password")
    agree = st.checkbox("我同意条款")
    # 表单提交按钮
    submitted = st.form_submit_button("提交")

# 表单提交后的处理
if submitted:
    if agree:
        st.success(f"表单已提交！欢迎，{username} ")
    else:
        st.error("请同意条款后再提交。")
```

在以上代码中，表单创建时指定了一个 `key`（此处为 `"my_form"`，用于标识表单）。表单内部可以像普通脚本一样添加输入部件，但需要最后使用 `st.form_submit_button` 来生成一个提交按钮 [4]。当用户填写完表单并点击提交按钮时，表单内所有输入的值会**一次性**传递给后端，然后应用重新运行。我们通过检查 `submitted` 的值（布尔类型）来确定表单是否提交，并相应地处理提交后的逻辑。

注意： 每个表单**必须**包含一个 `st.form_submit_button`，而普通的 `st.button` 不能放在表单内使用 [5]。此外，在表单中，只有提交按钮可以绑定回调函数，表单内其他小部件不支持独立的回调（这一点会在后续“回调”部分详述）。

3. 绘图和数据展示

数据可视化和表格展示是数据应用的重要部分。Streamlit 提供了一些**内置的快捷绘图函数**，以及支持直接使用主流绘图库（如 Matplotlib、Plotly 等）绘制更复杂的图表。

内置图表组件： Streamlit 有一些内置的简易图表函数，可以快速将数据绘制成常用图表，比如折线图、条形图、面积图等。这些函数通常接受 **pandas DataFrame**、**numpy 数组**或 **Python 列表** 作为输入数据。例如：

- `st.line_chart(data)`: 绘制折线图（线图）。
- `st.bar_chart(data)`: 绘制条形图。
- `st.area_chart(data)`: 绘制面积图。

这些封装函数使用起来非常方便，适合快速探索数据。例如：

```python
import streamlit as st
import pandas as pd
import numpy as np

# 生成示例数据：20行3列的随机数数据框
chart_data = pd.DataFrame(
    np.random.randn(20, 3),
    columns=["产品A", "产品B", "产品C"]
```

```
)

st.line_chart(chart_data)  # 绘制折线图
st.bar_chart(chart_data)   # 绘制条形图
```

上面的代码将随机生成的数据绘制成折线图和条形图，显示在应用页面中。只需一行命令即可完成绘图，非常简洁。需要注意，这些 `st.line_chart`、`st.bar_chart` 等方法主要用于快速可视化，默认会对数据按索引顺序绘制，在简单场景下非常有效。

数据表格展示： Streamlit 提供了展示表格数据的组件：

- `st.dataframe(data)`：可交互的表格组件。接受 pandas 的 DataFrame 或类似的二维数据，默认提供纵横滚动、列排序等交互能力。
- `st.table(data)`：静态表格。将数据以静态的形式渲染为表格（没有交互功能）。

例如：

```
import pandas as pd

df = pd.DataFrame({
    "姓名": ["张三", "李四", "王五"],
    "年龄": [28, 34, 29],
    "得分": [85, 92, 78]
})

st.dataframe(df)  # 可交互的表格展示
st.table(df.head(2))  # 静态表格，仅显示前两行
```

使用 `st.dataframe` 时，表格会带有滚动条、列宽拖动等功能，可以在前端与表格内容交互。而 `st.table` 则用于需要展示静态数据的场景（比如报告中的小表格）。

集成其他绘图库： 除了内置的简易绘图，Streamlit 可以很方便地集成其他强大的绘图库。官方支持包括 **Matplotlib**、**Plotly**、**Altair**、**Bokeh**、**Graphviz**、**PyDeck** 等等 [6]。使用方法通常是生成相应库的图表/图形对象后，调用 Streamlit 提供的专用函数来渲染。常用的包括：

- `st.pyplot(fig)`：显示 Matplotlib 绘制的图形对象 `fig` [6]。
- `st.plotly_chart(fig)`：显示 Plotly 创建的交互式图表 [7]。
- `st.altair_chart(chart)`：显示 Altair 图表对象。
- `st.bokeh_chart(fig)`：显示 Bokeh 图表。
- `st.graphviz_chart(graph)`：显示 Graphviz 图。
- `st.pydeck_chart(chart)`：显示 PyDeck 图层等。

具体使用示例：

```
import streamlit as st
import matplotlib.pyplot as plt
import numpy as np

# 使用 Matplotlib 绘制示例图表
```

```
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
fig, ax = plt.subplots()
ax.plot(x, y, color="red", linewidth=2)
ax.set_title("Matplotlib 折线图")
st.pyplot(fig)  # 将 Matplotlib 图表嵌入 Streamlit 页面

# 使用 Plotly 绘制示例图表
import plotly.express as px
df = px.data.iris()  # 使用 plotly 自带的 iris 示例数据集
fig2 = px.scatter(df, x="sepal_width", y="sepal_length", color="species",
                  title="Plotly 散点图")
st.plotly_chart(fig2)  # 将 Plotly 图表嵌入页面
```

上述代码首先生成了一个 Matplotlib 折线图，并通过 `st.pyplot` 显示在应用中 [6]。接着使用 Plotly 快速绘制了 Iris 数据集的散点图，使用 `st.plotly_chart` 进行展示 [7]。可以看到，借助 Streamlit，我们几乎无需额外配置即可将其他库的可视化结果嵌入到应用中，这使得创建丰富的数据应用变得非常容易。

4. 使用会话状态和回调

构建交互式应用时，一个重要问题是如何管理**状态**。由于 Streamlit 的执行模型是**脚本在每次交互后重新从头运行**，所以普通的局部变量不会在交互之间保存。为了解决这一问题，Streamlit 提供了 **会话状态 (Session State)**，可以在用户会话的多次运行之间**存储和共享变量** [8]。简单来说，会话状态就是一个跨越多次交互的**持久化字典**。

会话状态的作用： 默认情况下，用户与 Streamlit 页面交互（例如调整滑块、填写表单、点击按钮）后，整个脚本会从头执行。不过，Streamlit 内部对每个交互组件做了**状态保留**——例如文本输入框会记住先前输入的值，复选框会记住勾选状态等 [9]。但如果我们需要在组件之外保存一些自定义的信息（例如累积点击次数、缓存一些计算结果），就需要借助 `st.session_state` 提供的状态容器来实现。通过会话状态，我们可以在各次交互间**保存数据**，并使用回调函数对状态变化进行响应 [8] [10]。

基本用法： `st.session_state` 可以像字典一样使用（同时也支持属性访问）。通常，我们在第一次使用某个键之前，先初始化它以避免 KeyError。例如：

```
import streamlit as st

# 初始化 Session State 中的变量
if "count" not in st.session_state:
    st.session_state.count = 0

# 一个按钮，每次点击使计数加1
if st.button("点击我增加计数"):
    st.session_state.count += 1

st.write(f"当前计数：{st.session_state.count}")
```

在上述示例中，我们使用会话状态变量 `count` 来跟踪按钮点击次数。首次运行应用时，`st.session_state` 中还没有 `count`，通过 `if` 初始化为0。每次用户点击按钮，应用重跑并执行

`st.session_state.count += 1` 将计数加一。因为会话状态在会话存续期间是持久的，所以 `count` 的值会一直累加，不会因为脚本重新运行而丢失。

我们也可以直接通过 `st.session_state["key"]` 字典方式或 `st.session_state.key` 属性方式读取和修改状态。比如 `st.write(st.session_state.count)` 可以输出当前计数值，`st.session_state.foo = "bar"` 可以存入一个新键值。[11] [12]

回调函数 (`on_change` / `on_click`)： Streamlit 允许我们为输入部件注册回调函数。当用户与某些输入部件交互时，可以在**更新界面之前**先执行一段自定义逻辑。回调函数在指定组件的值发生变化时被调用，调用后整个应用仍会重新运行，但 **回调会在重新运行前先执行** [10]。常见的用法是在回调函数中更新 `st.session_state`，从而影响重新运行时应用的行为。

使用回调需要在定义部件时，通过参数 `on_change`（或对于按钮类组件使用 `on_click`）指定回调函数名称，并可选地通过 `args` 和 `kwargs` 传递参数 [10]。目前大多数输入组件支持 `on_change` 事件，典型的如 `st.text_input`、`st.slider`、`st.selectbox` 等；而 `st.button`、`st.form_submit_button` 等支持 `on_click` 事件 [13]。

下面示例演示如何使用回调将文本输入自动转换为大写：

```
import streamlit as st

# 定义回调函数：将 session_state 中 input_text 的值转换为大写
def convert_to_upper():
    st.session_state.output_text = st.session_state.input_text.upper()

# 创建文本输入，绑定回调
st.text_input("请输入文本：", key="input_text", on_change=convert_to_upper)
# 实时显示转换后的结果（如果 output_text 尚未设置，则显示空串）
st.write("大写转换结果：", st.session_state.get("output_text", ""))
```

在这个例子中，我们使用 `key` 参数将文本输入框的值存储在 `st.session_state.input_text` 中。当文本框内容改变且失去焦点时，`convert_to_upper` 回调被触发 [10]。该函数读取 `input_text`，将其大写后赋给 `output_text`。由于回调执行后应用会重新运行，`st.write` 会拿到更新后的 `output_text` 并显示，从而实现用户一输入文本就自动看到大写转换结果的效果。

提示： 使用回调更新会话状态可以实现更加复杂的交互逻辑，但也需要注意避免循环调用。例如，在回调中修改某个会话状态变量，而该变量又与触发回调的组件相关联，可能会引发重复运行或异常 [14]。确保回调函数的逻辑相对独立，或通过条件判断避免不必要的状态更新。

5. 布局与页面结构

Streamlit 默认按脚本代码的先后顺序从上到下垂直布局元素。不过，实际应用往往需要更灵活的排版，如**多栏布局**、**选项卡式内容**、侧边工具栏等。Streamlit 提供了一系列布局相关的API，使我们可以轻松地定制页面结构。

列布局 (`st.columns`)： 使用 `st.columns` 可以在同一行创建多个列，从而实现左右并排的布局。比如创建两列或三列布局：

```
col1, col2 = st.columns(2)
col1.write("这是第一列")
col2.write("这是第二列")

# 不同宽度的列，例如三列比例为3:1:1
col_a, col_b, col_c = st.columns([3, 1, 1])
col_a.write("较宽的一列")
col_b.write("较窄的一列")
col_c.write("较窄的一列")
```

上述代码首先创建了两个等宽的列 `col1` 和 `col2`，在各列中填入内容。接着演示了创建三个不同比例的列，通过传入列表 `[3,1,1]` 指定各列宽度比。您也可以将交互组件放入列中，例如：

```
col1, col2 = st.columns(2)
with col1:
    st.button("按钮 1")
with col2:
    st.button("按钮 2")
```

这样两个按钮就会并排显示在页面上。

选项卡 (`st.tabs`)： 选项卡允许将页面内容分组，在界面上以标签页形式切换显示。用法类似于列布局：

```
tab1, tab2 = st.tabs(["选项卡1", "选项卡2"])
with tab1:
    st.write("这里是选项卡1的内容")
with tab2:
    st.write("这里是选项卡2的内容")
```

上面代码创建了两个选项卡页签“选项卡1”和“选项卡2”。`with tabX:` 语句块内添加的元素会归属于对应的选项卡。运行后，应用上部会出现可点击的页签，用户点击不同标签可切换查看不同内容。

侧边栏 (`st.sidebar`)： Streamlit 提供一个内置的侧边栏容器，适合放置导航控件、全局设置等，使页面主区域更加简洁。将组件添加到侧边栏的方法有两种：

1. 在调用组件时通过 `st.sidebar` 前缀，如 `st.sidebar.button("侧边按钮")`，这样该组件就会渲染在侧边栏区域。
2. 使用 `with st.sidebar:` 代码块，将其中的组件渲染到侧边栏。

例如：

```
st.sidebar.title("侧边栏")  # 侧边栏标题
st.sidebar.radio("页面导航", ["首页", "数据概览", "关于"])  # 单选框导航

# 或使用 with 语法
with st.sidebar:
```

```
    st.slider("全局参数调整", 0, 100, 50)
    st.checkbox("显示详细信息")
```

侧边栏默认在应用的左侧呈现一个窄栏。通过将不那么重要或全局的控件放入侧边栏，可以使主要区域更加专注于核心内容[15]。此外，可以通过配置将侧边栏设置为起始时收起等（详见 `st.set_page_config`）。

折叠区（`st.expander`）： 折叠区（又称展开/收起面板）用于在页面上折叠大量内容，用户点击后才展开查看。适合用于可选的说明、附加细节等。用法示例：

```
with st.expander("点击展开查看更多信息"):
    st.write("这里是额外的信息，可以折叠或展开。")
    st.image("https://docs.streamlit.io/en/stable/_static/logo.png", width=100)
```

在这个例子中，我们创建了一个带标题的折叠区域，初始状态下内容隐藏。用户点击后会展开显示内部的文字和图片。再次点击则收起。折叠区可以帮助我们在界面上**节省空间**，只在用户需要时呈现信息。

多页面应用和导航： Streamlit 支持将应用拆分为多个页面，以构建多页面应用。通常方法是在应用目录下创建一个 **pages/** 文件夹，将不同页面的 Python 脚本放入其中。运行主应用时，Streamlit 会自动在界面上生成页面切换的选项，用户可以导航到不同页面。

除了默认的页面选择菜单外，Streamlit 还提供了 API 来在应用内创建页面链接或进行跳转：

- `st.page_link(page_script, label="...", icon="...")`：在当前页面插入一个链接，点击可跳转到指定的其他页面（或者外部URL）。其中 `page_script` 可以是另一页面脚本的路径或名称[16]。
- `st.switch_page("page_name")`：编程方式跳转到指定页面（直接重新运行目标页），常用于在代码中根据某些条件跳转。
- `st.navigation(...)`：配置自定义的导航菜单（高级用法）。

举例来说，如果我们有一个 `pages/summary.py` 作为第二页，我们可以在首页脚本中添加：

```
st.page_link("pages/summary.py", label="查看数据总结", icon="📊")
```

这样会在首页显示一个超链接按钮“查看数据总结”[17]。用户点击该链接，将切换到 summary.py 定义的页面内容。

注意： 为使 `st.page_link` 工作，需要已在应用中定义多页面结构（例如正确放置页面脚本）。`st.page_link` 仅提供一个友好的链接 UI 来跳转页面。通常情况下，多页面应用只需将脚本按要求存放，Streamlit 会自动生成侧边导航选择。但使用 `st.page_link` 可以在页面内部创建更灵活的导航元素。

6. 部署应用

当应用开发调试完毕后，就可以考虑将其部署上线与他人分享。Streamlit 提供了免费的 **Community Cloud** 平台（以前称为 Streamlit Sharing），可以非常方便地将应用部署到云端，无需自行架设服务器。部署大致包括两部分：准备代码仓库（包含依赖说明），以及在平台上进行部署操作。

准备代码和依赖说明： 通常我们使用 Git 将应用代码托管在 GitHub 等代码仓库中，然后连接到 Streamlit Community Cloud 部署。为了确保部署的应用能正常运行，**必须提供依赖包列表**。具体做法是在项目根目录添

加一个 **`requirements.txt`** 文件，其中列出应用所需的 Python 第三方库。部署服务器会根据该文件自动安装依赖。如果缺少此文件，部署时环境中只会安装 Streamlit 本身（及其直接依赖），其他库将不存在，可能导致应用失败 [18]。

编写 `requirements.txt` 文件时，每行一个包名及可选的版本号，例如：

```
streamlit
pandas>=2.0
matplotlib==3.7.1
plotly
```

需要注意的是，无需在文件中列出 Python 的内置库（如 `math`、`os` 等），也不必包含标准库 [19]。另外，Streamlit 本身默认会预装在部署环境中，除非您想指定特定版本，否则可以不在 requirements 文件中硬性指定 Streamlit [20]。正确编写依赖文件可以确保云端环境安装好应用运行所需的所有包。

部署到 Streamlit Community Cloud： 当代码和依赖准备就绪后，按照以下步骤将应用部署到云端：

1. **推送代码**：将应用的代码仓库（例如 GitHub 仓库）更新到最新，包括应用主脚本和 `requirements.txt` 等文件。
2. **登录平台**：访问 Streamlit Community Cloud（第一次使用需使用GitHub账号授权登录）。
3. **创建新应用**：在您的 Streamlit Cloud 工作区界面，点击“**New app**”或“创建应用”按钮 [21]。如果是首次部署，会提示“已有现有应用代码吗？”——选择“是，我已有代码”，然后继续。
4. **选择仓库和分支**：在弹出的部署对话框中，选择您要部署的 GitHub 仓库、分支（如 `main`），以及应用的入口文件路径（即主脚本，例如 `app.py`） [22]。确认无误后，可以点击部署。
5. **（可选）设置选项**：在部署前，可以展开“高级设置”。这里允许选择 Python 版本（默认使用最新的受支持版本，例如3.12） [23]，以及填写秘密变量（如API密钥）等。如果您的应用需要特定的 Python 版本或需要环境变量配置，可以在此设置。一般情况下默认设置即可。
6. **部署过程**：点击部署后，平台将开始构建并运行您的应用。您可以在界面右侧实时查看构建日志。第一次部署通常在几十秒到几分钟内完成，具体取决于依赖包的多少和大小 [24]。如果部署过程中有错误（例如依赖未找到），日志会显示相应信息，您需要根据提示修改代码或依赖再重新部署。
7. **访问应用**：部署成功后，应用会分配一个固定的 URL。在 Streamlit Community Cloud 上，应用通常托管在 `*.streamlit.app` 域名下，每个应用有唯一子域名。您可以直接访问该 URL 来使用应用，并将链接分享给他人进行访问。比如，部署时可以自定义子域名为 `myapp`，那么应用地址可能形如 `https://myapp.streamlit.app`（如果未自定义，平台会基于仓库名等生成一个随机子域 [25]）。现在，任何人都可以通过浏览器访问这个链接来使用您的应用。

部署到 Community Cloud 后，可以在平台的仪表板查看您所有的应用，进行**管理**（例如暂停、删除应用），查看应用的运行日志，或者更新应用（推送新的代码后，应用会自动重新部署更新）。平台也提供了秘密管理、资源限制等功能，在此不展开。

注意： Streamlit Community Cloud 对应用有一定的资源限制和社区规范，例如单个应用的内存和运行时限制等 [26]。免费社区版通常足以应对一般的轻量应用。如果您的应用需求更高或需要私有部署，也可以考虑将 Streamlit 应用打包成 Docker 容器部署到其他云服务，或使用 Streamlit 企业版/Snowflake 等方案，但这些属于进阶话题了。

通过以上六个部分的介绍，您应该已经对 Streamlit 有了全面的了解：从基础用法到高级交互，再到布局美化和部署上线。借助 Streamlit，数据应用的开发变得前所未有地方便和高效——只需掌握 Python 基础知识，就可

以几乎零门槛地构建出漂亮、专业的交互式 Web 应用 [27]。现在，试着用 Streamlit 将您的数据科学想法分享给全世界吧！祝您构建愉快

参考资料：

- Streamlit 官方文档 [3] [6] [9] [8] [10] [17] [4] 等
- Streamlit API Cheat Sheet [2] [1] [28] [21] [24] (v1.41.0)
- Streamlit Community Cloud 部署指南 [18] [25]

---

[1] 关于 Streamlit in Snowflake | Snowflake Documentation
https://docs.snowflake.cn/zh/developer-guide/streamlit/about-streamlit

[2] [15] Streamlit API cheat sheet - Streamlit Docs
https://www.aidoczh.com/streamlit/develop/quick-reference/cheat-sheet.html

[3] Text elements - Streamlit Docs
https://www.aidoczh.com/streamlit/develop/api-reference/text.html

[4] [5] st.form - Streamlit Docs
https://www.aidoczh.com/streamlit/develop/api-reference/execution-flow/st.form.html

[6] [7] Chart elements - Streamlit Docs
https://www.aidoczh.com/streamlit/develop/api-reference/charts.html

[8] [10] [11] [12] [13] [14] Session State - Streamlit Docs
https://www.aidoczh.com/streamlit/develop/api-reference/caching-and-state/st.session_state.html

[9] Caching and state - Streamlit Docs
https://www.aidoczh.com/streamlit/develop/api-reference/caching-and-state.html

[16] [17] Navigation and pages - Streamlit Docs
https://www.aidoczh.com/streamlit/develop/api-reference/navigation.html

[18] [19] [20] [26] [28] App dependencies for your Community Cloud app - Streamlit Docs
https://www.aidoczh.com/streamlit/deploy/streamlit-community-cloud/deploy-your-app/app-dependencies.html

[21] [22] [23] [24] [25] Deploy your app on Community Cloud - Streamlit Docs
https://www.aidoczh.com/streamlit/deploy/streamlit-community-cloud/deploy-your-app/deploy.html

[27] 数据分析web可视化神器---streamlit框架，无需懂前端也能搭建出精美的web网站页面-阿里云开发者社区
https://developer.aliyun.com/article/1497223