



Advanced Streamlit for Python Developers

PyCon US

Pittsburgh, PA - May 15, 2024



© 2024 Snowflake Inc. All Rights Reserved

Agenda

1. Intros 🤝
2. What is Streamlit? 🎈
3. Basic features 🧩
4. Advanced features 🚀
5. Q&A ❓
6. Product roadmap 🗺️
7. Snack break 🍪 (10:30-10:45am)
8. Hands-on exercise 📖




Intros 🤝



Intros

Who are we?



Tony Kipkemboi
Developer Advocate
 Streamlit



Intros 🤝

Who are we?



Caroline Frasca
Manager, Dev Rel & Partnerships
👑 Streamlit

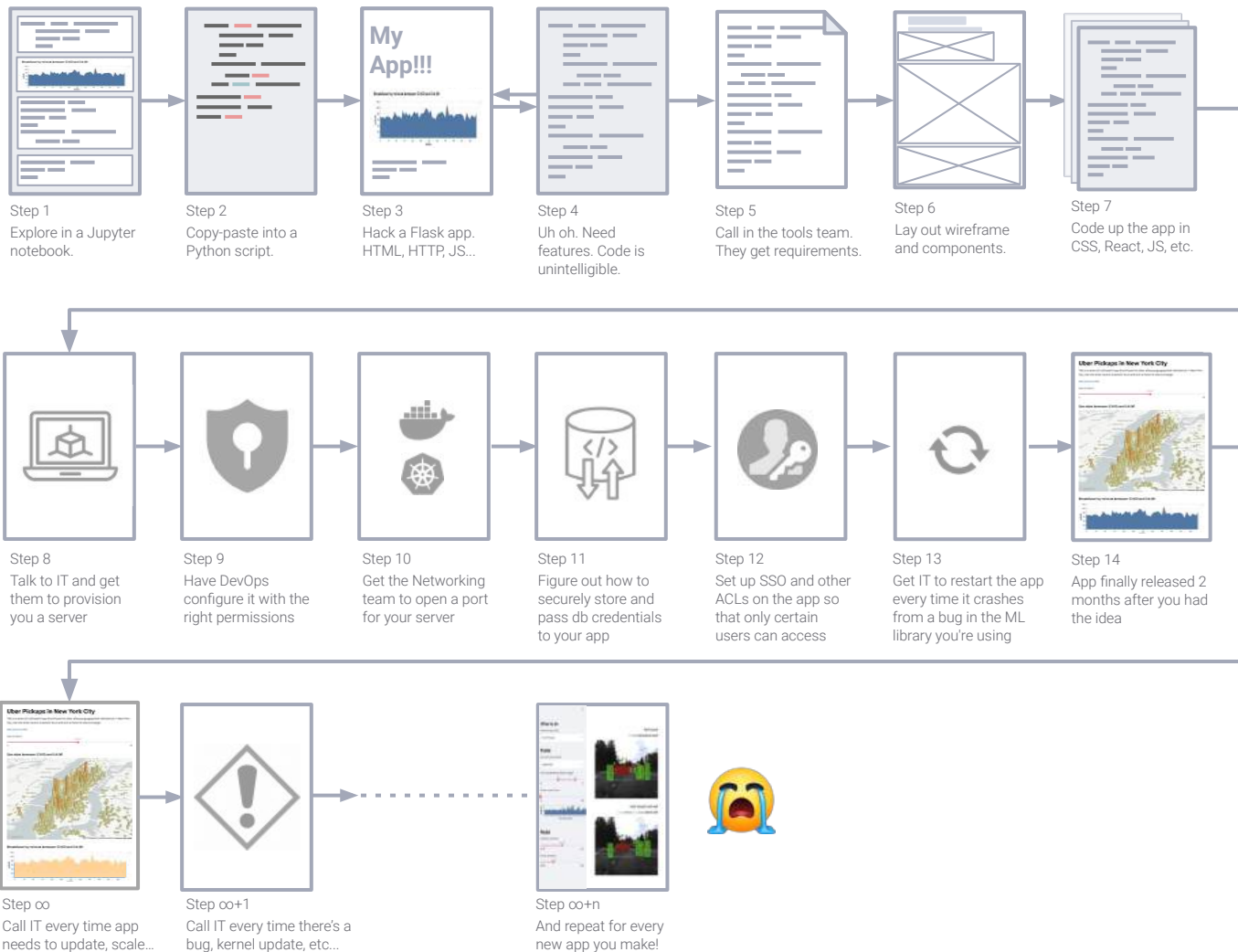


What is Streamlit?

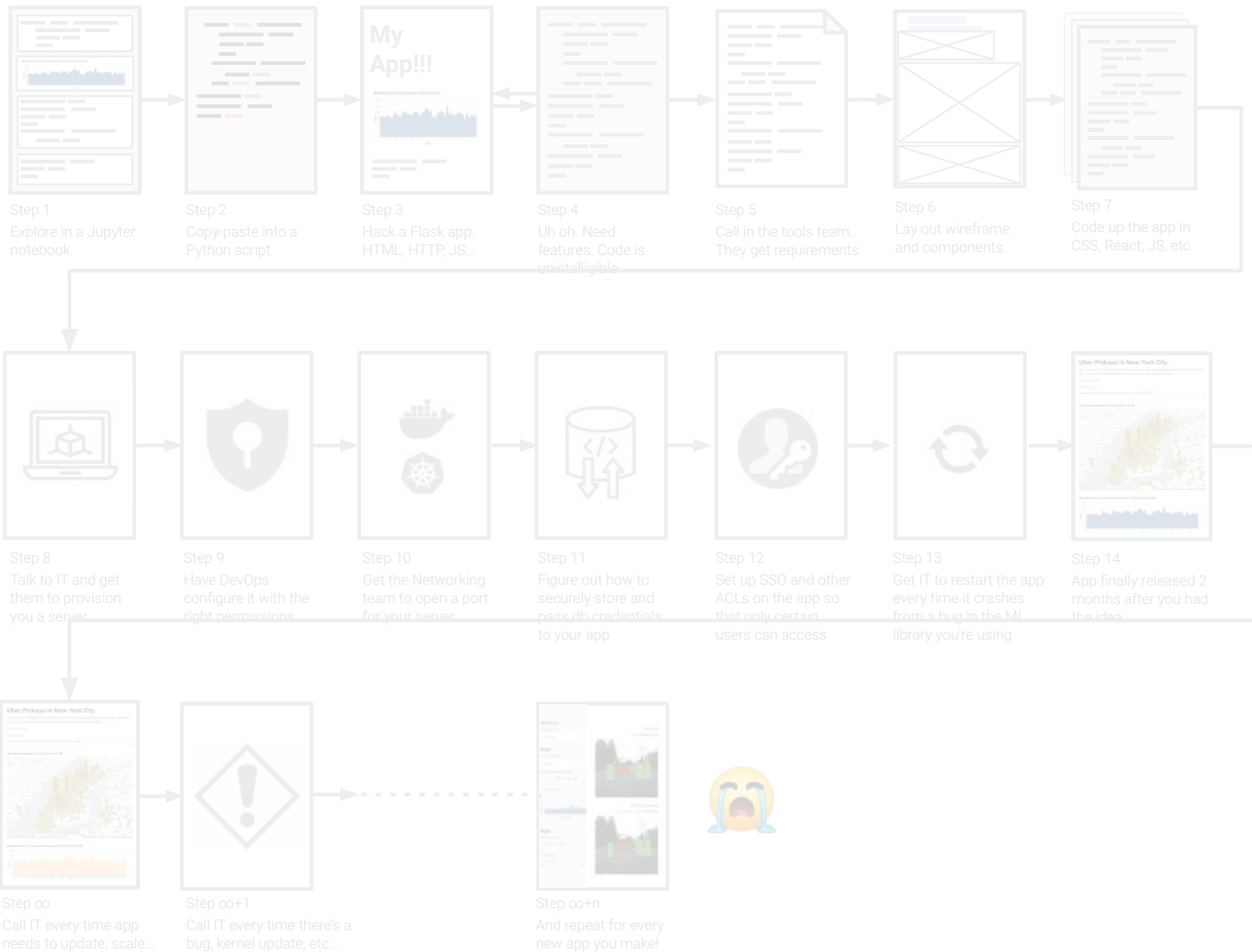


Getting from data to action is hard

Only a fraction of data questions are ever answered since companies haven't built tools to capture or analyze the data. 73% of data is rendered effectively unused* because it's just too hard to build the tools to use it.

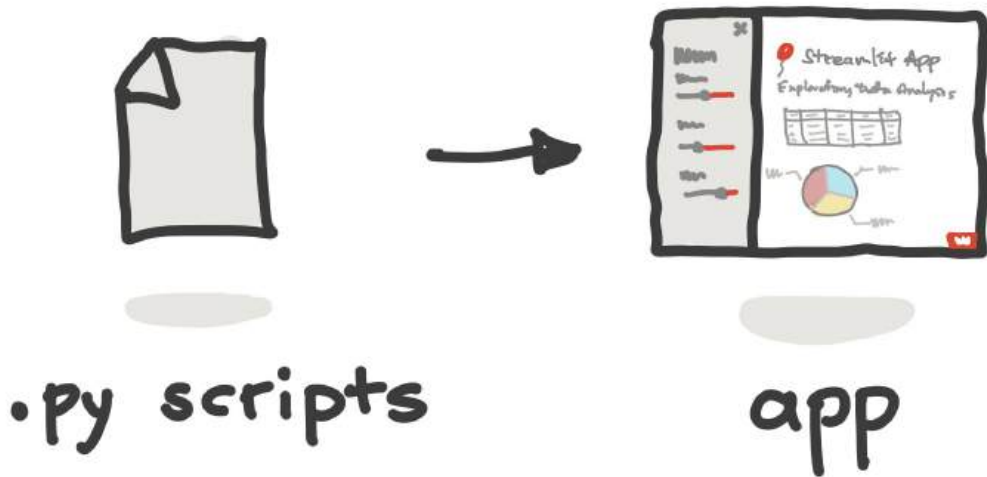


Streamlit is a **faster** way to build and share data apps

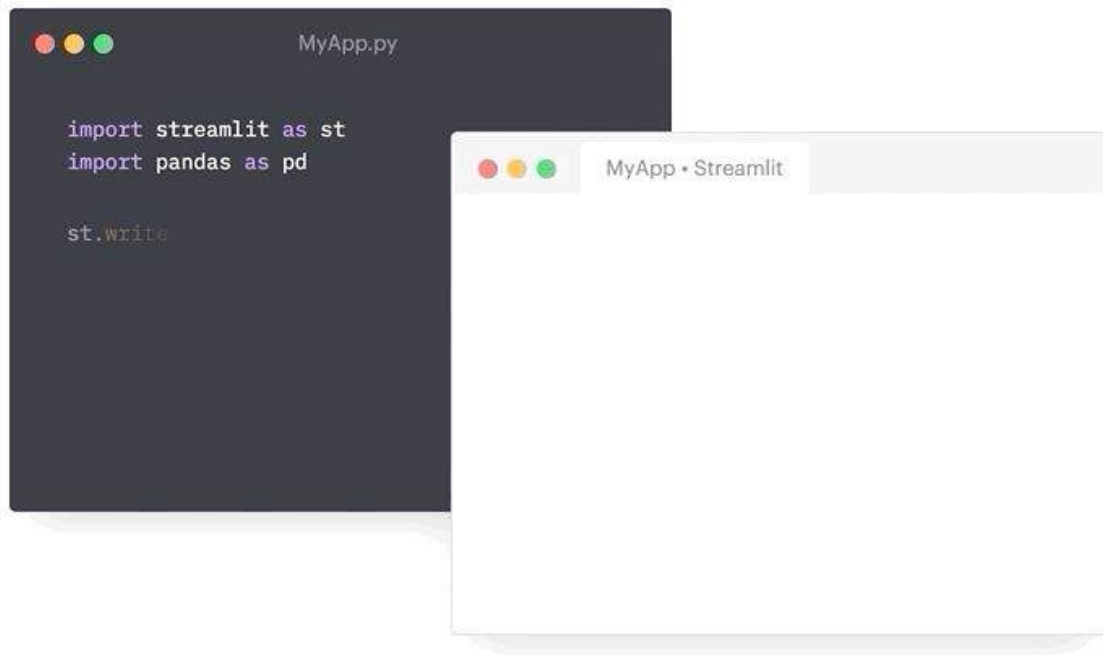


What is Streamlit?

An open-source Python framework that delivers dynamic data apps in only a few lines of code



Guessable, opinionated, and user-friendly Python syntax



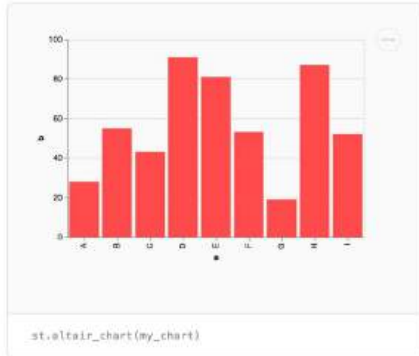
It's easy to add in interaction, layout, and themes

Pick a number



0 100

```
number = st.slider("Pick a number", 0, 100)
```



Pick a file

Drag and drop files here
Limit 200MB per file • TXT

Browse files

```
file = st.file_uploader("Pick a file")
```

Pick a color



```
color = st.color_picker("Pick a color")
```

Pick a pet

Dog

Cat

Bird

```
pet = st.radio("Pick a pet", ["Dog", "Cat", "Bird"])
```

Pick a date











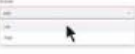














```
date = st.date_input("Pick a date")
```

Choose from light or dark theme or set your own custom colors



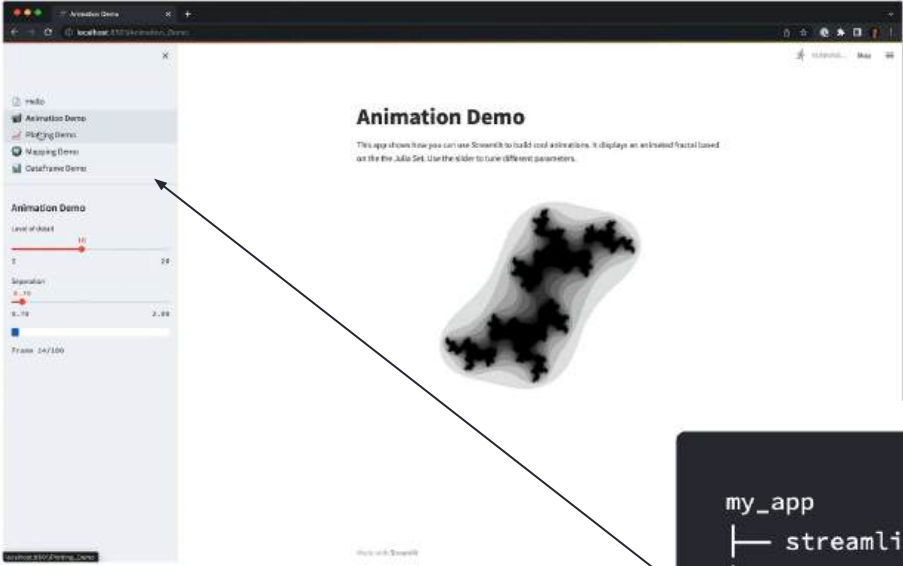
Streamlit API

 Button Display a button widget. <code>clicked = st.button("Click me")</code>	 Download button Display a download button widget. <code>st.download_button("Download", ...)</code>	 Checkbox Display a checkbox widget. <code>selected = st.checkbox("I agree")</code>	 Number input Display a numeric input widget. <code>choice = st.number_input("Pick a choice")</code>	 Text-area Display a multi-line text input widget. <code>text = st.text_area("Text to type")</code>	 Date input Display a date input widget. <code>date = st.date_input("Your birthday")</code>	 Progress bar Display a progress bar. <code>for f in range(100): st.progress(f) do_something_slow()</code>	 Spinner Temporarily displays a message while executing a block of code. <code>with st.spinner("Please wait..."): do_something_slow()</code>	 Balloons Display celebratory balloons! <code>do_something() # Celebrate when all done! st.balloons()</code>
 Radio Display a radio button widget. <code>choice = st.radio("Pick one", ...)</code>	 Selectbox Display a select widget. <code>choice = st.selectbox("Pick one", ...)</code>	 Multiselect Display a multiselect widget. This multiselect widget starts as empty. <code>choices = st.multiselect("Buy", ...)</code>	 Time input Display a time input widget. <code>time = st.time_input("Meeting time")</code>	 File Uploader Display a file uploader widget. <code>data = st.file_uploader("Upload files")</code>	 Camera input Display a widget that allows users to upload images directly a camera. <code>image = st.camera_input("Take photo")</code>	 Error box Display an error message. <code>st.error("We encountered an error")</code>	 Warning box Display a warning message. <code>st.warning("Double to fetch data")</code>	 Info box Display an informational message. <code>st.info("Release is updated")</code>
 Slider Display a slider widget. <code>number = st.slider("Pick a number", ...)</code>	 Select-slider Display a slider widget to select items from a list. <code>size = st.select_slider("Pick a size", ...)</code>	 Text input Display a single-line text input widget. <code>name = st.text_input("First name")</code>				 Success box Display a success message. <code>st.success("Match found!")</code>	 Exception output Display an exception. <code>try: ... except Exception as e: st.exception(e)</code>	

docs.streamlit.io



Once you have the basics mastered... go crazy!

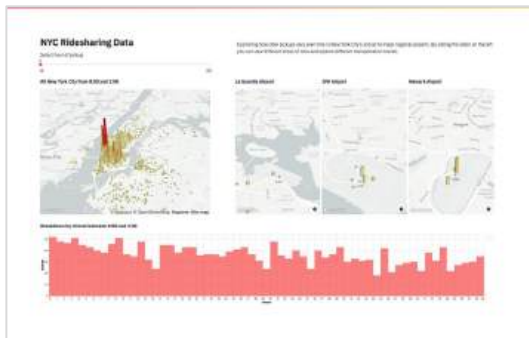


Multi-page apps

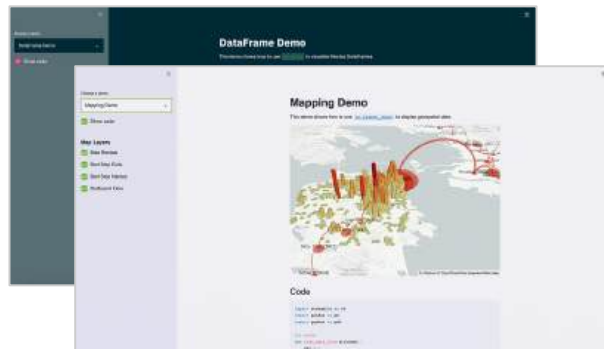
```
my_app
├── streamlit_app.py  <-- Your main script
├── pages
│   ├── page_2.py    <-- New page 2!
│   └── page_3.py    <-- New page 3!
```



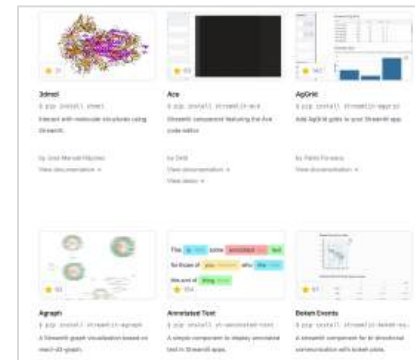
Once you have the basics mastered... go crazy!



Use layouts



Try different themes



Use components



Streamlit works with all your favorite Python libraries

 bokeh

 Altair

 PyTorch

 OpenCV

DECK.GL

 pandas
data science



Vega-Lite

 matplotlib

 NumPy

 learn

 TensorFlow

 plotly

 Keras



and seamlessly integrates with your favorite AI tools

 LlamaIndex

 Pinecone


 AI21 Labs

 chroma

 Weaviate

 Qdrant

 OpenAI


 Weights & Biases

 snowflake

 Hugging Face

 LangChain

 Stable Diffusion

 AssemblyAI

 clarifai
The World's AI™

 truBricks™

BabyAGI

 Sygii-Dev

And many more! Get started at streamlit.io/generative-ai



Live demo



Get inspired by thousands of public apps



Streamlit ECharts Demo

 andfanilo

[View source →](#)



Streamlit Components Hub

 jrieke

[View source →](#)



Image Background Remover

 tyler-simons

[View source →](#)




Streamlit cheat sheet

 daniellewisl

[View source →](#)



Streamlit extras

 arnaudmirbel

[View source →](#)



Roadmap

 streamlit

[View source →](#)



Streamlit folium documentation

 randyzwitch

[View source →](#)



Prophet

 maximelutl

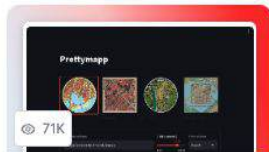
[View source →](#)



Data Engineering Zoomcamp 2023

 hamagistral

[View source →](#)



prettymapp

 chrieke

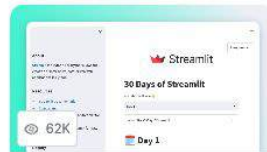
[View source →](#)



GW Quickview

 jkanner

[View source →](#)



30Days of Streamlit

 streamlit

[View source →](#)

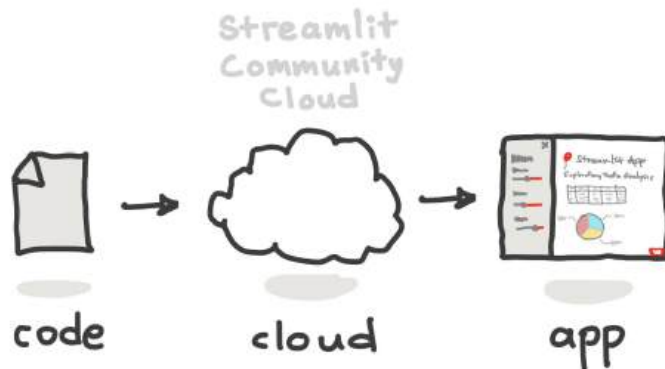
Check them out for yourself: streamlit.io/gallery



Deploy anywhere, share with anyone

Frictionless sharing experience

- Host unlimited public apps and one private app on **Streamlit Community Cloud**
- Build on top of your Snowflake data with **Streamlit in Snowflake**
- Our [Community Deployment Wiki](#) has guides for deploying on many other platforms

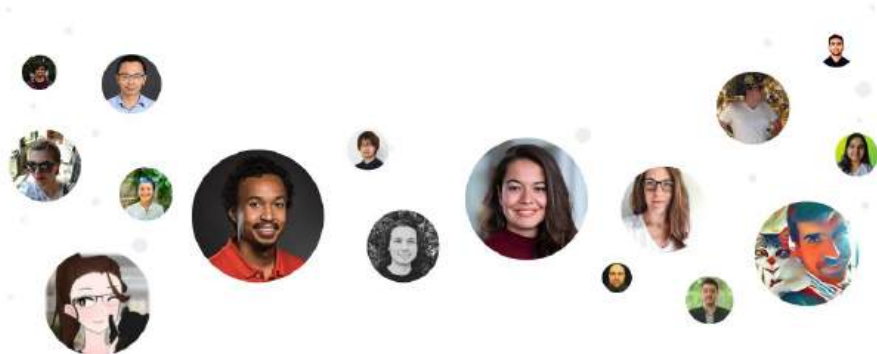


Get started with Streamlit Community Cloud: share.streamlit.io

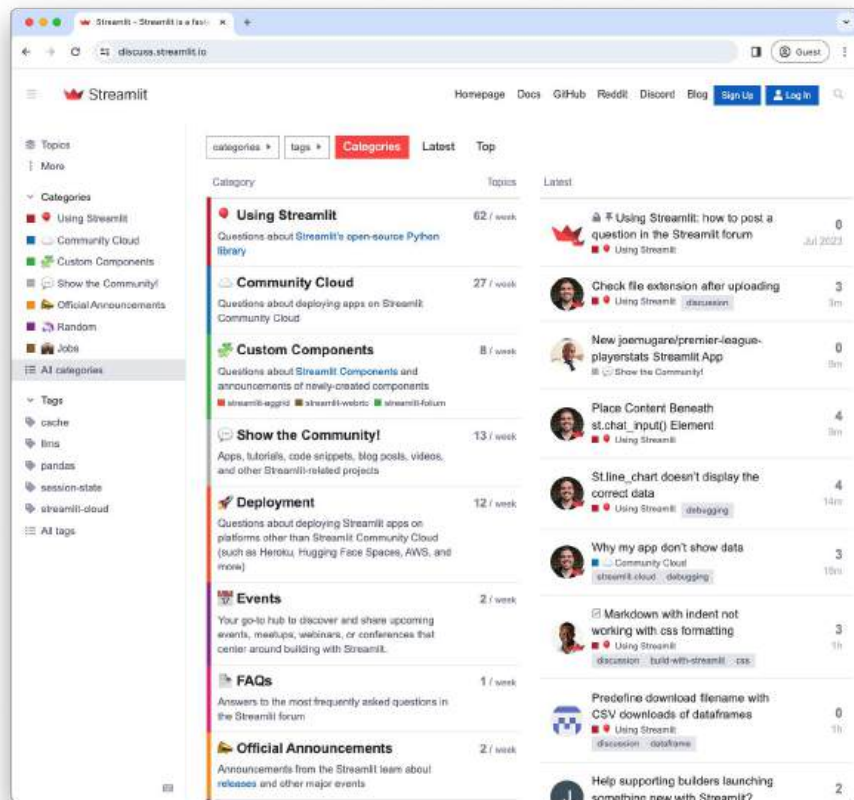


The Streamlit community is here to support and inspire

Our **forum** is the best place to get started:
discuss.streamlit.io



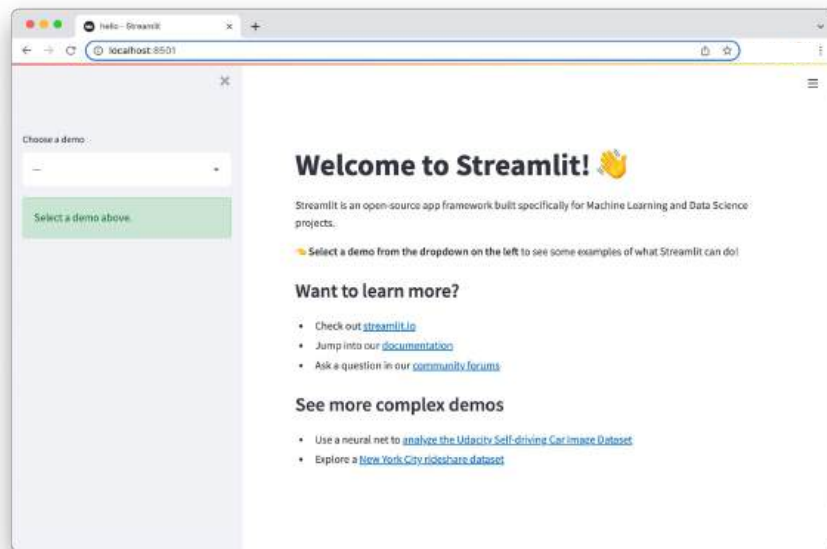
Learn more about our community programs at
streamlit.io/community



Remember, it only takes minutes to get started!

```
$ pip install streamlit  
$ streamlit hello
```

See our [docs](#) for more getting started info with the Streamlit open-source framework



Basic features



Text elements

Display text in your app using headings and formatted text elements.

- Streamlit apps typically begin with **st.title** for the main title, followed by **st.header** and **st.subheader** for section headings.
- Use **st.text** for plain text and **st.markdown** for Markdown content.
- The versatile **st.write** command supports multiple arguments and data types.



Headings and body text

Display text in your app using headings and body text commands.

```

Lorem ipsum dolor sit amet
Consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Tincidunt lobortis
Fugiat veniam ut sagittis eget quis ipsum. Sed non proident quam voluptate. Consectetur in hac habitasse platea dictumst. Aliquam sit amet sagittis nisi.
Non diam phasellus vestibulum
Ut quam elementum pulvinar elit. Blandit volutpat mauris non quam. Sed non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

- Ut phasellus sit amet aliquip sit.
- Etiam tempor nisi eu lobortis elementum. Nulla malesuada magna.
- Sed enim ut veni ornare aliquip sagittis nisi.
- Etiam sit amet tempor ipsum. Nulla malesuada magna.

Sed non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

```

Markdown
Display string formatted as Markdown.

```
st.markdown("Hello world!")
```

```

THIS IS A TITLE
This is a title
This is a title
This is a title
This is a title
This is a title
This is a title

```

Title
Display text in title formatting.

```
st.title("The app title")
```

```

This is a header
This is a header
This is a header
This is a header
This is a header
This is a header
This is a header

```

Header
Display text in header formatting.

```
st.header("This is a header")
```

```

This is a subheader
This is a subheader
This is a subheader
This is a subheader
This is a subheader
This is a subheader
This is a subheader

```

Subheader
Display text in subheader formatting.

```
st.subheader("This is a subheader")
```



Formatted text

Display formatted and preformatted text, code block, and LaTeX in your app.

This is a caption
This is a caption
This is a caption
This is a caption
This is a caption
This is a caption

Caption
Display text in small font.

```
st.caption("This is written")
```

```
1 # Import packages
2 from IPython.display import Image
3 from IPython.display import Video
4 from IPython.display import HTML
5 from IPython.display import Math
6 from IPython.display import Code
7 from IPython.display import Text
8 from IPython.display import Markdown
9 from IPython.display import Latex
10 from IPython.display import Audio
11 from IPython.display import IFrame
12 from IPython.display import SVG
13 from IPython.display import Javascript
14 from IPython.display import Image
15 from IPython.display import Video
16 from IPython.display import HTML
17 from IPython.display import Math
18 from IPython.display import Code
19 from IPython.display import Text
20 from IPython.display import Markdown
21 from IPython.display import Latex
22 from IPython.display import Audio
23 from IPython.display import IFrame
24 from IPython.display import SVG
25 from IPython.display import Javascript
```

Code block
Display a code block with optional syntax highlighting.

```
st.code("a = 1234")
```

```
1 # Import packages
2 from IPython.display import Image
3 from IPython.display import Video
4 from IPython.display import HTML
5 from IPython.display import Math
6 from IPython.display import Code
7 from IPython.display import Text
8 from IPython.display import Markdown
9 from IPython.display import Latex
10 from IPython.display import Audio
11 from IPython.display import IFrame
12 from IPython.display import SVG
13 from IPython.display import Javascript
```

Echo
Display some code on the app, then execute it. Useful for tutorials.

```
with st.echo():
    st.write('This code will !')
```



Preformatted text
Write fixed-width and preformatted text.

```
st.text("Hello world")
```

$$\vec{B} = -\nabla \times \vec{E}$$
$$\vec{E} = \nabla \times \vec{B} - 4\pi\vec{j}$$
$$G_{\mu\nu} + \Lambda g_{\mu\nu} = \frac{8\pi G}{c^2} T_{\mu\nu}$$

LaTeX
Display mathematical expressions formatted as LaTeX.

```
st.latex("\int a x^2 \, dx")
```

This text is between dividers

This text is between dividers

This text is between dividers

This text is between dividers

This text is between dividers

This text is between dividers

Divider
Display a horizontal rule.

```
st.divider()
```



Data elements

Streamlit has commands for easy data display and interaction

You can display your raw data using commands like:

- **st.dataframe**
- **st.data_editor**
 - **st.column_config**
- **st.metric**
- **st.table**
- **st.json**



st.dataframe


Display a dataframe as an interactive table

This command works with dataframes from **Pandas**, **PyArrow**, **SnowPark**, and **PySpark**

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

df = pd.DataFrame(
    np.random.randn(50, 20),
    columns=("col %d" % i for i in
range(20))
)

st.dataframe(df)
```



	col 0	↑ col 1	col 2	col 3	col 4	col 5	col 6	col 7	col 8	col 9	col 10
46	0.4885	1.1698	0.4493	1.1212	0.4584	0.5027	0.1725	1.7831	-0.0163	0.048	-1.2305
43	0.1603	1.213	0.2235	1.2697	-0.0372	-0.3953	-0.1421	0.2098	2.4337	-1.6096	2.0781
32	0.0916	1.2897	-0.399	-1.0198	-0.9906	0.2637	0.0655	-1.1876	0.3891	1.3624	-2.1286
31	0.3919	1.3225	-0.7194	-1.0112	-2.0326	0.28	-0.9729	0.2646	0.9062	-1.4577	-0.6206
11	0.1739	1.5062	-0.6348	0.7814	0.4981	-0.1963	1.7488	0.3312	0.2029	-0.5109	0.3812
49	-0.5982	1.5397	0.0943	-0.7441	0.7285	0.9053	0.7501	-1.1666	-0.3012	0.4997	0.155
7	0.2463	1.5484	-1.0836	-0.9086	-0.4108	-0.1384	-0.6574	0.1696	-0.5618	0.8175	-1.683
30	0.9822	1.6111	-0.6442	0.9459	-0.0841	0.9089	-0.2204	-0.2759	-0.4906	-2.9322	-1.1347
5	-1.4199	2.3866	-0.8387	0.5267	1.4764	0.679	0.8401	-0.489	0.0798	-0.9062	1.008
44	1.6691	2.503	0.1586	1.0572	1.055	0.3287	0.7342	-0.2886	-0.0966	1.2542	1.1565



st.data_editor

The data editor element allows you to edit dataframes and many other data structures in a table-like UI.

```
import streamlit as st
import pandas as pd

df = pd.DataFrame(
    [
        {"command": "st.selectbox", "rating":
4, "is_widget": True},
        {"command": "st.balloons", "rating":
5, "is_widget": False},
        {"command": "st.time_input", "rating":
3, "is_widget": True},
    ]
)
edited_df = st.data_editor(df)
```



	command	rating	is_widget
0	st.selectbox	2	<input checked="" type="checkbox"/>
1	st.balloons	3	<input type="checkbox"/>
2	st.time_input	5	<input checked="" type="checkbox"/>

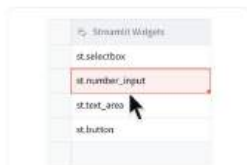
Your favorite command is st.time_input 

More st.data_editor demos: data-editor.streamlit.app



st.column_config

Format st.data_editor columns to beautifully display your data



Column

Configure a generic column.

```
Column("Streamlit Widgets", w=
```



Text column

Configure a text column.

```
TextColumn("Widgets", max_cha=
```



Number column

Configure a number column.

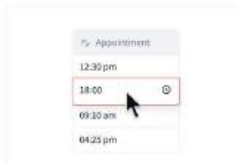
```
NumberColumn("Price (in USD)",
```



Date column

Configure a date column.

```
DateColumn("Birthday", max_va=
```



Time column

Configure a time column.

```
TimeColumn("Appointment", min,
```



List column

Configure a list column.

```
ListColumn("Sales (last 6 mon",
```



Progress column

Configure a progress column.

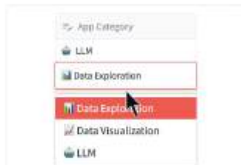
```
ProgressColumn("Sales volume"
```



Checkbox column

Configure a checkbox column.

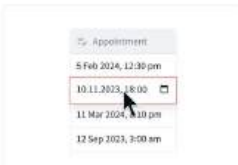
```
CheckboxColumn("Your favorite",
```



Selectbox column

Configure a selectbox column.

```
SelectBoxColumn("App Category",
```



Datetime column

Configure a datetime column.

```
DatetimeColumn("Appointment",
```



Link column

Configure a link column.

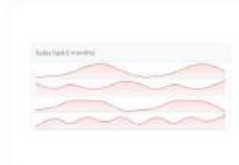
```
LinkColumn("Trending apps", m=
```



Image column

Configure an image column.

```
ImageColumn("Preview image", i,
```



Line chart column

Configure a line chart column.

```
LineChartColumn("Sales (last 6",
```



Bar chart column

Configure a bar chart column.

```
BarChartColumn("Marketing spe",
```

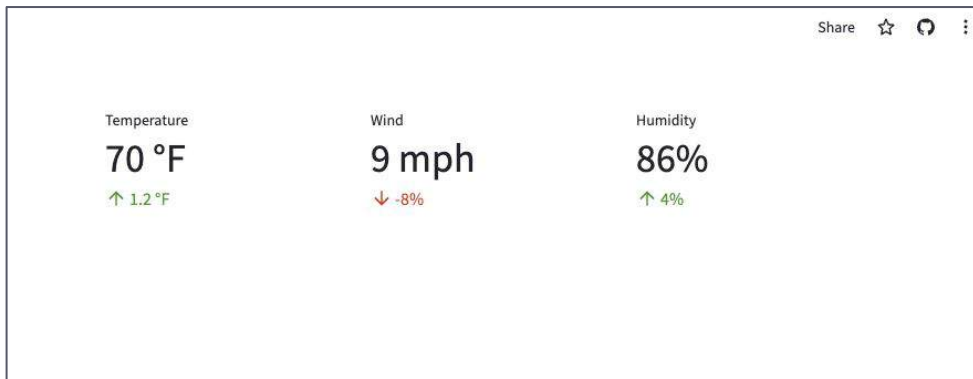


st.metric

Display a metric in big bold font, with an optional indicator of how the metric changed.

```
import streamlit as st

col1, col2, col3 = st.columns(3)
col1.metric("Temperature", "70 °F", "1.2 °F")
col2.metric("Wind", "9 mph", "-8%")
col3.metric("Humidity", "86%", "4%")
```



st.table

Display a static table.

This **differs from st.dataframe** in that the table in this case is **static**; its entire contents are laid out directly on the page.

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

df = pd.DataFrame(
    np.random.randn(10, 5),
    columns=["col %d" % i for i in
range(5)]
)

st.table(df)
```

	col 0	col 1	col 2	col 3	col 4
0	0.9168	0.3419	2.0455	-0.2833	-1.0922
1	-1.0674	0.1214	-1.1510	-2.4443	-1.3114
2	1.4411	1.1570	1.4971	1.2568	0.3811
3	-0.9506	1.2427	-0.3839	0.2517	-0.1341
4	0.6010	0.3457	0.7727	-0.9985	0.8365
5	0.2135	1.2370	1.1343	0.4449	0.2312
6	1.5821	0.1452	0.2905	0.5775	1.8986
7	0.1121	1.9412	-1.2499	0.4355	0.3174
8	-0.8775	-0.1392	1.4762	-1.1383	-0.8600
9	-0.4813	0.1019	0.8528	0.4515	1.5825



st.json

Display object or string as a pretty-printed JSON string.

```
import streamlit as st

st.json({
    'foo': 'bar',
    'baz': 'boz',
    'stuff': [
        'stuff 1',
        'stuff 2',
        'stuff 3',
        'stuff 5',
    ],
})
```

```
{
  "foo": "bar"
  "baz": "boz"
  "stuff": [
    0: "stuff 1"
    1: "stuff 2"
    2: "stuff 3"
    3: "stuff 5"
  ]
}
```



Chart elements

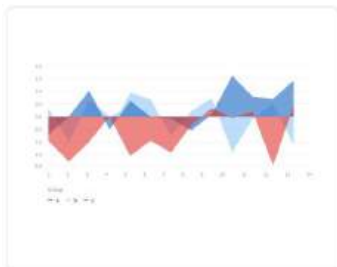
Streamlit supports several charting libraries, and our goal is to continually add support for more.

- The most basic library in our arsenal is **Matplotlib**.
- There are also interactive charting libraries like **Vega Lite** (2D charts) and **deck.gl** (maps and 3D charts).
- We also provide a few chart types that are “native” to Streamlit, like **st.line_chart** and **st.area_chart**.



Simple chart elements

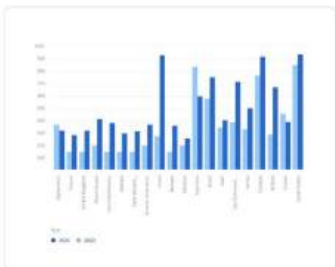
Display an area chart, bar chart, line chart, or scatter chart, and map with points on it.



Simple area charts

Display an area chart.

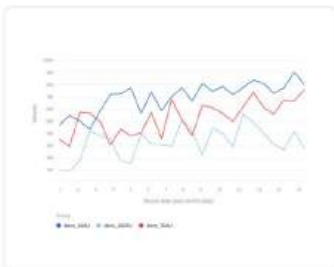
```
st.area_chart(my_data_frame)
```



Simple bar charts

Display a bar chart.

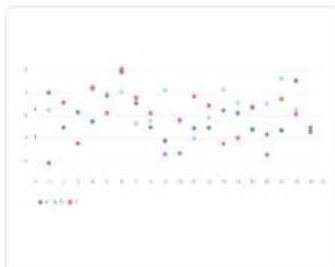
```
st.bar_chart(my_data_frame)
```



Simple line charts

Display a line chart.

```
st.line_chart(my_data_frame)
```



Simple scatter charts

Display a line chart.

```
st.scatter_chart(my_data_frame)
```



Scatterplots on maps

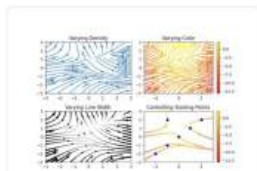
Display a map with points on it.

```
st.map(my_data_frame)
```



Advanced chart elements

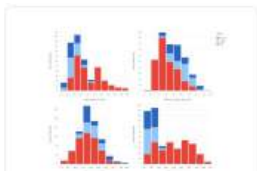
Display an area chart, bar chart, line chart, or scatter chart, and map with points on it.



Matplotlib

Display a matplotlib.pyplot figure.

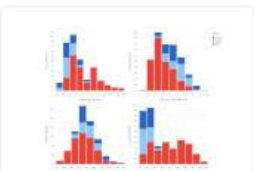
```
st.pyplot(my_mpl_figure)
```



Altair

Display a chart using the Altair library.

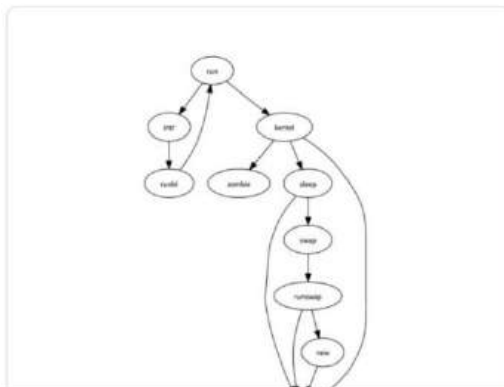
```
st.altair_chart(my_altair_cl
```



Vega-Lite

Display a chart using the Vega-Lite library.

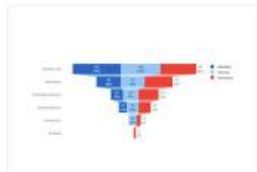
```
st.vega_lite_chart(my_vega_1
```



GraphViz

Display a graph using the dagre-d3 library.

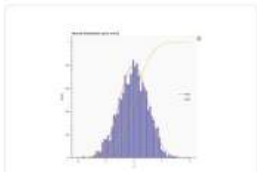
```
st.graphviz_chart(my_graphvi
```



Plotly

Display an interactive Plotly chart.

```
st.plotly_chart(my_plotly_cf
```



Bokeh

Display an interactive Bokeh chart.

```
st.bokeh_chart(my_bokeh_char
```



PyDeck

Display a chart using the PyDeck library.

```
st.pydeck_chart(my_pydeck_cf
```



Input elements

Streamlit allows you to bake in interactivity directly into your apps with **buttons**, **sliders**, **text inputs**, and more.

Different types of input elements:

- Button
- Selection
- Numeric input
- Date and time input
- Text input



Button elements

Display an interactive button to allow users to interact with your app.




Button
Display a button widget.

```
clicked = st.button("Click me")
```



Download button
Display a download button widget.

```
st.download_button("Download file", "Download file")
```




Form button
Display a form submit button. For use with `st.form`.

```
st.form_submit_button("Sign in")
```



Link button
Display a link button.

```
st.link_button("Go to gallery", "Go to gallery")
```



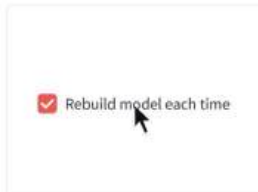
Page link
Display a link to another page in a multipage app.

```
st.page_link("app.py", label="Page 1")  
st.page_link("pages/profile.py", label="Profile")
```



Selection elements

Display an interactive selection elements to allow users to interact with your app.



Checkbox

Display a checkbox widget.

```
selected = st.checkbox("I agree")
```



Toggle

Display a toggle widget.

```
activated = st.toggle("Activate")
```



Radio

Display a radio button widget.

```
choice = st.radio("Pick one!",
```



Selectbox

Display a select widget.

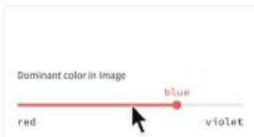
```
choice = st.selectbox("Pick
```



Multiselect

Display a multiselect widget. The multiselect widget starts as empty.

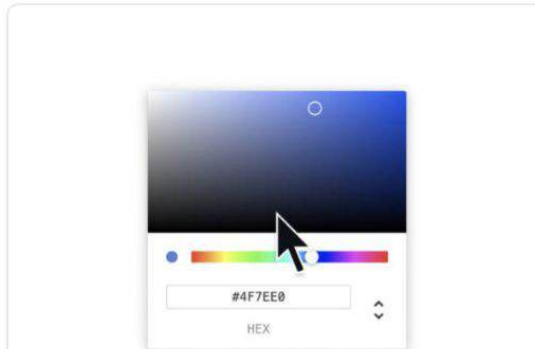
```
choices = st.multiselect("Bl
```



Select slider

Display a slider widget to select items from a list.

```
size = st.select_slider("Pi
```



Color picker


Display a color picker widget.

```
color = st.color_picker("Pic
```



Numeric input elements

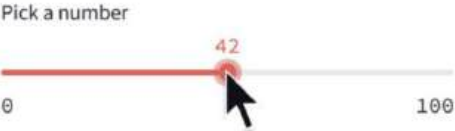
Display a numeric input element.



A screenshot of a web form element titled "Number of days". It features a light gray input field containing the number "28". To the right of the input field are minus and plus icons. A black mouse cursor is pointing at the right side of the input field.

Number input
Display a numeric input widget.

```
choice = st.number_input("Pick a number")
```



A screenshot of a web form element titled "Pick a number". It features a horizontal slider with a red track and a white handle. The track is labeled with "0" at the left end and "100" at the right end. The handle is positioned at the value "42", which is also labeled above it. A black mouse cursor is pointing at the handle.

Slider
Display a slider widget.

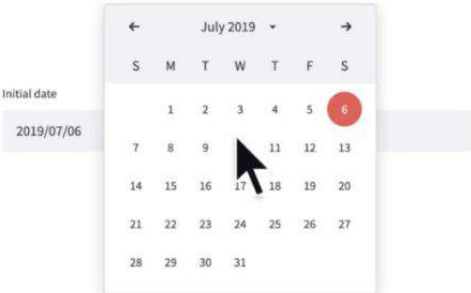
```
number = st.slider("Pick a number", 0, 100, 42)
```



Date and time input elements




Display a date and time input element.



Date input
Display a date input widget.

```
date = st.date_input("Your k
```




Time input
Display a time input widget.

```
time = st.time_input("Meetir
```



Text input elements

Display a single-line or multi-line text input element and chat input element.



Movie title

Life of Brian

Text input

Display a single-line text input widget.

```
name = st.text_input("First
```



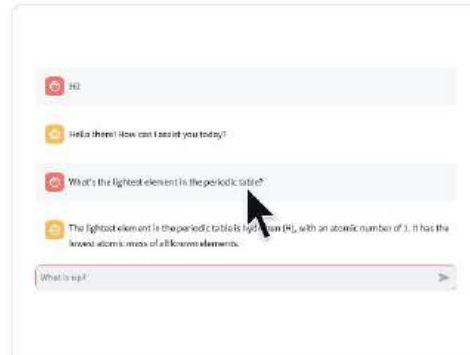
Text to analyze

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the season of hope, it was the season of despair. ()

Text area

Display a multi-line text input widget.

```
text = st.text_area("Text to
```



Hi

Hello there! How can I assist you today?

What's the lightest element in the periodic table?

The lightest element in the periodic table is Hydrogen (H), with an atomic number of 1. It has the lowest atomic mass of all known elements.

What is up?

Chat input

Display a chat input widget.

```
prompt = st.chat_input("Say  
if prompt:  
    st.write(f"The user has
```



Media elements 📹

It's easy to embed images, videos, and audio files directly into your Streamlit apps

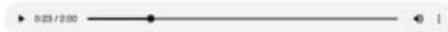


Summit by the mountains

Image

Display an image or list of images.

```
st.image(numpy_array)
st.image(image_bytes)
st.image(file)
st.image("https://example.com/myimage.;
```



Audio

Display an audio player.

```
st.audio(numpy_array)
st.audio(audio_bytes)
st.audio(file)
st.audio("https://example.com/myaudio.i
```



Video

Display a video player.

```
st.video(numpy_array)
st.video(video_bytes)
st.video(file)
st.video("https://example.com/myvideo.i
```



Layouts and containers

Several options for controlling how different elements are laid out on the screen.



Columns

Insert containers laid out as side-by-side columns.

```
col1, col2 = st.columns(2)
col1.write("this is column 1")
col2.write("this is column 2")
```



Container

Insert a multi-element container.

```
c = st.container()
st.write("This will show last")
c.write("This will show first")
c.write("This will show second")
```



Empty

Insert a single-element container.

```
c = st.empty()
st.write("This will show last")
c.write("This will be replaced")
c.write("This will show first")
```



Tabs

Insert containers separated into tabs.

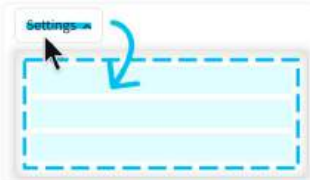
```
tab1, tab2 = st.tabs(["Tab 1", "Tab2"])
tab1.write("this is tab 1")
tab2.write("this is tab 2")
```



Expander

Insert a multi-element container that can be expanded/collapsed.

```
with st.expander("Open to see more"):
    st.write("This is more content")
```



Popover

Insert a multi-element popover container that can be opened/closed.

```
with st.popover("Settings"):
    st.checkbox("Show completed")
```



Sidebar

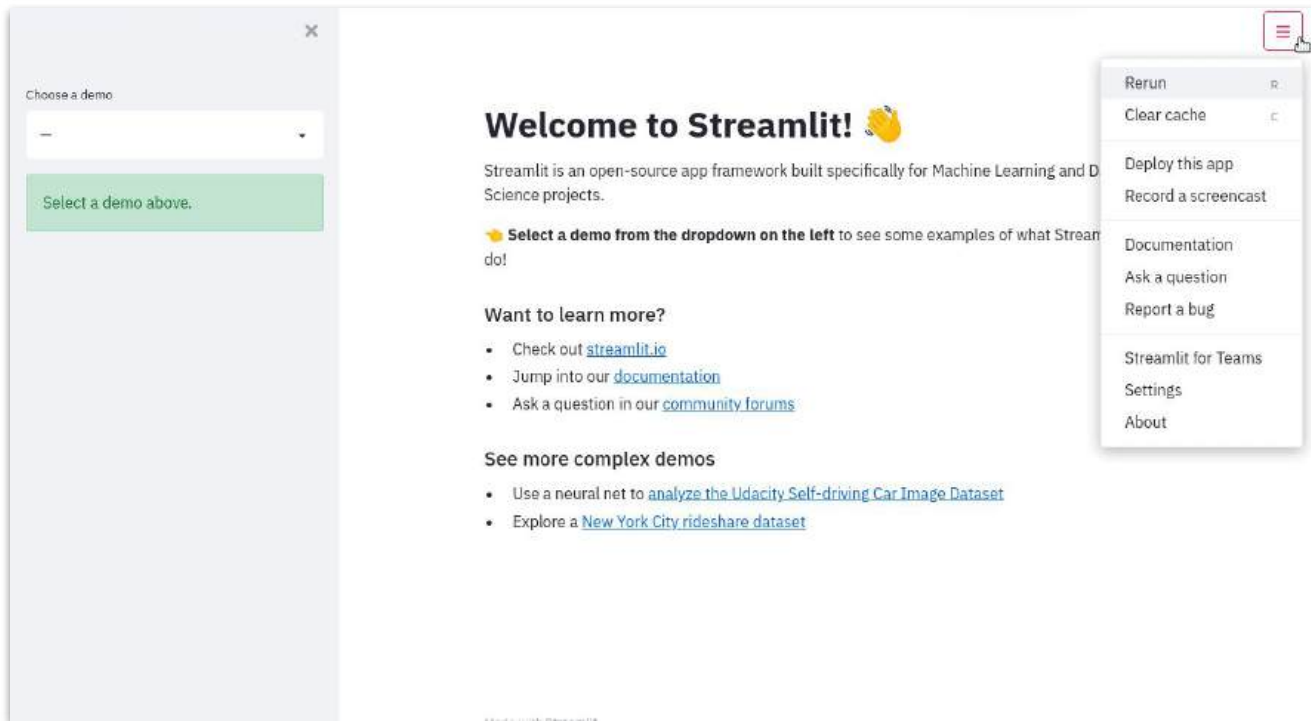
Display items in a sidebar.

```
st.sidebar.write("This lives in the si")
st.sidebar.button("Click me!")
```



Theming

Customize the look and feel of your Streamlit apps



Choose a demo

Select a demo above.

Welcome to Streamlit! 🙌

Streamlit is an open-source app framework built specifically for Machine Learning and Data Science projects.

👉 Select a demo from the dropdown on the left to see some examples of what Streamlit can do!

Want to learn more?

- Check out [streamlit.io](#)
- Jump into our [documentation](#)
- Ask a question in our [community forums](#)

See more complex demos

- Use a neural net to [analyze the Udacity Self-driving Car Image Dataset](#)
- Explore a [New York City rideshare dataset](#)

Rerun R

Clear cache C

Deploy this app

Record a screencast

Documentation

Ask a question

Report a bug

Streamlit for Teams

Settings

About

[theme]

```
primaryColor="#F63366"  
backgroundColor="#FFFFFF"  
secondaryBackgroundColor="#F0F2F6"  
textColor="#262730"  
font="sans serif"
```

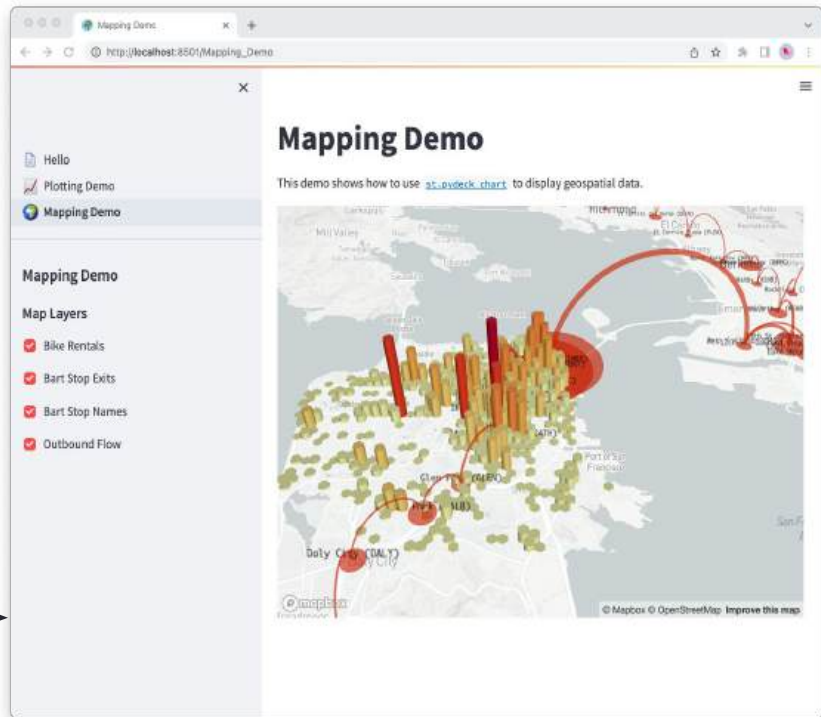


Multi-page apps

Streamlit provides a frictionless way to create multi-page apps

```
your_working_directory/  
├── pages/  
│   ├── a_page.py  
│   └── another_page.py  
└── your_homepage.py
```

```
streamlit run your_homepage.py
```



Beyond the basics



st.data_editor

Display a data editor widget.

The data editor widget allows you to edit dataframes and many other data structures in a table-like UI.

```
import streamlit as st
import pandas as pd

df = pd.DataFrame(
    [
        {"command": "st.selectbox", "rating":
4, "is_widget": True},
        {"command": "st.balloons", "rating":
5, "is_widget": False},
        {"command": "st.time_input", "rating":
3, "is_widget": True},
    ]
)
edited_df = st.data_editor(df)
```



	command	rating	is_widget
0	st.selectbox	2	<input checked="" type="checkbox"/>
1	st.balloons	3	<input type="checkbox"/>
2	st.time_input	5	<input checked="" type="checkbox"/>

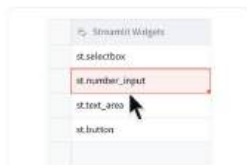
Your favorite command is st.time_input 

More st.data_editor demos: data-editor.streamlit.app



st.column_config

Format st.data_editor columns to beautifully display your data



Column

Configure a generic column.

```
Column("Streamlit Widgets", w=
```



Text column

Configure a text column.

```
TextColumn("Widgets", max_cha=
```



Number column

Configure a number column.

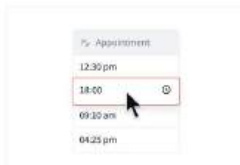
```
NumberColumn("Price (in USD)",
```



Date column

Configure a date column.

```
DateColumn("Birthday", max_va=
```



Time column

Configure a time column.

```
TimeColumn("Appointment", min,
```



List column

Configure a list column.

```
ListColumn("Sales (last 6 mon",
```



Progress column

Configure a progress column.

```
ProgressColumn("Sales volume"
```



Checkbox column

Configure a checkbox column.

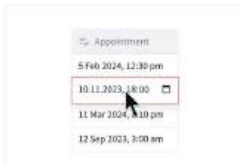
```
CheckboxColumn("Your favorite",
```



Selectbox column

Configure a selectbox column.

```
SelectBoxColumn("App Category",
```



Datetime column

Configure a datetime column.

```
DatetimeColumn("Appointment",
```



Link column

Configure a link column.

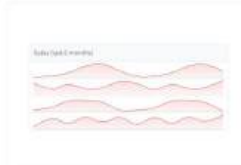
```
LinkColumn("Trending apps", m=
```



Image column

Configure an image column.

```
ImageColumn("Preview image", i,
```



Line chart column

Configure a line chart column.

```
LineChartColumn("Sales (last 6",
```



Bar chart column

Configure a bar chart column.

```
BarChartColumn("Marketing spe",
```



Custom components

Build your own Python modules to extend what's possible with Streamlit – or use one of the many existing community-build components!

Different types of custom components:

- Custom versions of existing Streamlit elements and widgets.
- Completely new Streamlit elements and widgets by wrapping existing React.js, Vue.js, or other JavaScript widget toolkits.
- Rendering Python objects having methods that output HTML, such as `IPython __repr_html__`.
- Convenience functions for commonly-used web features like GitHub gists and Pastebin.



Custom components: build your own

2 types of components: **static components** and **bi-directional components**

Static components: rendered once; controlled by Python; useful to display HTML, render a visual using an existing Python library, or render an iframe

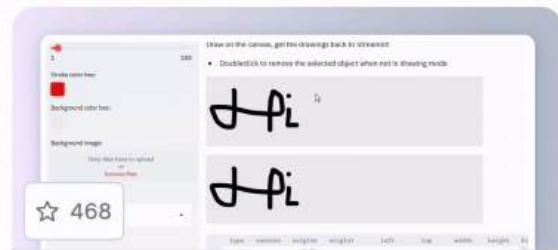
Bi-directional components: have a frontend (HTML / any web tech you want) rendered via an iframe and a Python API that Streamlit apps use to instantiate and talk to the frontend

Example of static component

```
import streamlit as st
import streamlit.components.v1 as
components

# embed streamlit docs in a streamlit app
components.iframe("https://example.com",
height=500)
```

Example of bi-directional component



Drawable Canvas

 andfanilo

```
$ pip install streamlit-drawable-canvas
```



Custom components: bi-directional components

Each Streamlit component is **its own webpage that gets rendered into an iframe** – so you can use just about any web tech you'd like to create that web page

We offer two templates to get started: one [React-based template](#) and one [TypeScript-only template](#)

The templates also provide the folder structure needed to publish your component to PyPI, so you can share your component with the world!

Limitations of custom components

- Can't contain or communicate with other components
- Can't modify the CSS of the rest of the app
- Can't add or remove elements of the app



Custom components: existing components

There are tons of preexisting components built by our community

 [PyGWalker demo](#)

 [Components Gallery](#)



Pywalker

 Kanaries

`$ pip install pywalker`



HiPlot

 facebookresearch

`$ pip install hiplot`




Hub

 pyscaffold

`$ pip install streamlit-hub`



WebRTC

 whitphx

`$ pip install streamlit-webrtc`



Authenticator

 mkhorasani

`$ pip install streamlit-authenti...`



AgGrid

 PablocFonseca

`$ pip install streamlit-aggrid`



Chat elements

Build conversational apps and chatbots with Streamlit's chat elements:

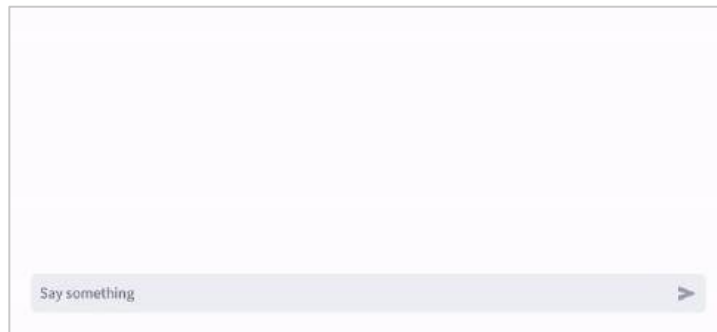
st.chat_message and **st.chat_input**

Use **st.write_stream** to stream responses and **st.status** to display output from long-running tasks

Basic example

```
import streamlit as st

if prompt := st.chat_input("Say something"):
    with st.chat_message("user"):
        st.write(prompt)
```



Chat elements

More robust example: echo bot

```
# Initialize chat history
if "messages" not in st.session_state:
    st.session_state.messages = []

# Display chat messages from history on app rerun
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

# React to user input
if prompt := st.chat_input("What is up?"):
    # Display user message in chat message container
    with st.chat_message("user"):
        st.markdown(prompt)
    # Add user message to chat history
    st.session_state.messages.append({"role": "user", "content":
prompt})

response = f"Echo: {prompt}"
# Display assistant response in chat message container
with st.chat_message("assistant"):
    st.markdown(response)
# Add assistant response to chat history
st.session_state.messages.append({"role": "assistant", "content":
response})
```



Use `st.session_state` to store message history



Chat elements

More robust example: ChatGPT-like app

```
client = OpenAI(api_key=st.secrets["OPENAI_API_KEY"])

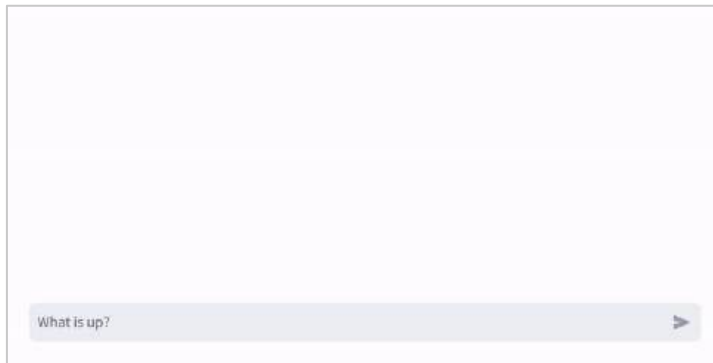
if "openai_model" not in st.session_state:
    st.session_state["openai_model"] = "gpt-3.5-turbo"

if "messages" not in st.session_state:
    st.session_state.messages = []

for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

if prompt := st.chat_input("What is up?"):
    st.session_state.messages.append({"role": "user", "content": prompt})
    with st.chat_message("user"):
        st.markdown(prompt)

    with st.chat_message("assistant"):
        stream = client.chat.completions.create(
            model=st.session_state["openai_model"],
            messages=[
                {"role": m["role"], "content": m["content"]}
                for m in st.session_state.messages
            ],
            stream=True,
        )
        response = st.write_stream(stream)
    st.session_state.messages.append({"role": "assistant", "content": response})
```



✓ Use **st.session_state** to store message history

✓ Use **st.write_stream** to stream GPT-3.5's responses



Connections and databases



Connect your Streamlit app to your favorite data source through your API of choice or **st.connection**

The st.connection API makes it easy to connect your app to data in just a few lines of code

Example: connect to Snowflake

```
# Initialize connection.
conn = st.connection("snowflake")

# Perform query.
df = conn.query("SELECT * from mytable;", ttl=600)

# Print results.
for row in df.itertuples():
    st.write(f"{row.NAME} has a :{row.PET}:")
```



Connections and databases: st.connection



Build your own connection or use one of many created by the community



Google Sheets Connection

 streamlit

```
$ pip install st-gsheets-connect...
```



Supabase Connection

 SiddhantSadangi

```
$ pip install st-supabase-connec...
```



Files Connection

 streamlit

```
$ pip install st-files-connection
```



OpenAI Embeddings Connection

 gerardrbentley

```
$ pip install st-openai-embeddin...
```



Airtable Connection

 marks

```
$ pip install st-airtable-connec...
```



CockroachDB Connection

 putuwaw

```
$ pip install st-cockroachdb-con...
```



NewsAPI Connector

 dcarpintero

```
$ pip install st-newsapi-connect...
```



Dynamodb Connection

 mrtj

```
$ pip install st-dynamodb-connec...
```

 [Connections Gallery](#)



Client-server architecture

Streamlit apps have a **client-server architecture**



Server = Python backend

Brains of your app – performs computations for all users who view app

Starts up when someone executes the command `streamlit run your_app.py`

Runs on the machine where this command was run (also called the host)

Client = frontend viewed through browser

The client is the frontend of your app (which is viewed through your browser of choice)

If your app is running locally, your computer runs both the server and client

If your app is deployed and someone views your app through the browser, their device is the client



Client-server architecture: impact on app design

How does this architecture impact how you build your app?

1. The computer running or hosting your app is responsible for providing the compute and storage for **all users**
2. Your app won't have access to a user's files, directories, or OS (unless a user uploads specific files)
3. If your app communicates with peripheral devices (e.g. camera), you must use Streamlit commands/custom components to access those devices through the browser and communicate between client and server
4. If your app uses any program or process outside of Python, those programs or processes will run on the server, not on the user's machine (unless the server is the user's machine!)



Execution model

Any time something must be updated in the UI of your app, **Streamlit reruns your entire Python script from top to bottom**

This happens in two situations:

1. When you modify your app's source code
2. When a user interacts with a widget

Note: callback functions will always run before the rest of the script.

There are a few ways to handle this execution model:

- Caching
- Forms
- Session state
- Fragments



Review: Streamlit app model

1. Streamlit apps are Python scripts that run from top to bottom.
2. Every time a user opens a browser tab pointing to your app, the script is executed and a new session starts.
3. As the script executes, Streamlit draws its output live in a browser.
4. Every time a user interacts with a widget, your script is re-executed and Streamlit redraws its output in the browser. The output value of that widget matches the new value during that rerun.
5. Scripts use **caching** to avoid recomputing expensive functions, so updates happen very fast.
6. **Session state** lets you save information that persists between reruns when you need more than a simple widget.
7. Streamlit apps can contain multiple pages, which are defined in separate **.py** files in a **pages** folder.



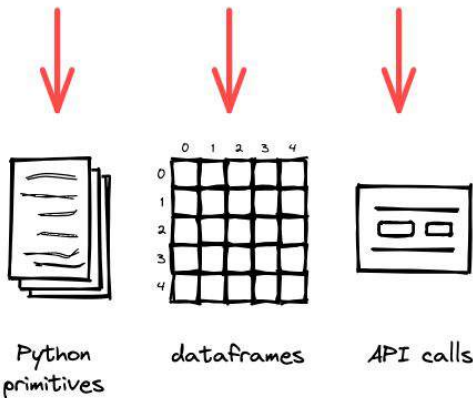
Caching: `st.cache_data` and `st.cache_resource` ⚡

Store the results of slow function calls, so they only run once → faster apps

Streamlit has two caching decorators: **`st.cache_data`** and **`st.cache_resource`**

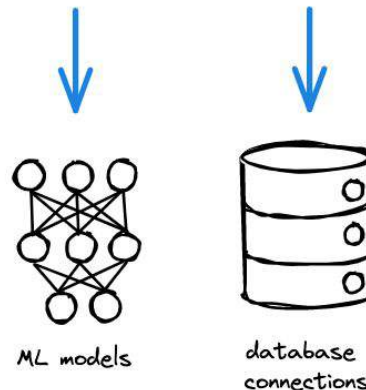
`st.cache_data`

anything you CAN store in a database



`st.cache_resource`

anything you CAN'T store in a database

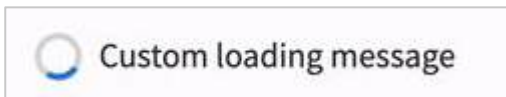


Caching: control cache size and duration ⚡

Control the cache size and duration through:

- **Time-to-live (TTL) parameter** sets a time to live on a cached function (in seconds) – useful to prevent stale data and prevent cache from growing too large
- **max_entries parameter** sets the maximum number of entries in the cache – useful to prevent the cache from growing too large. When a new entry is added to a full cache, the oldest entry will be removed.

Customize loading spinner through the **show_spinner** parameter (disable the spinner or display a custom message)



Caching: hashing input parameters ⚡

To compare input parameters, Streamlit hashes them and compares them to stored hash keys

What happens if your parameters **aren't hashable**?

- Exclude unhashable parameters from caching by prepending the parameter name with an **underscore**. If that parameter changes, Streamlit will still return cached result if all other parameters match.
- Use **hash_funcs** parameter to specify custom hashing function (use a function that returns deterministic, true hash values).

```
@st.cache_data(hash_funcs={MyCustomClass: hash_func})
```

```
@st.cache_data
def fetch_data(_db_connection, num_rows): # Don't hash _db_connection
    data = _db_connection.fetch(num_rows)
    return data
```



Forms

Batch elements together with a “Submit” button

- Forms are containers that visually group other elements and widgets together, and contain a submit button
- When the submit button is clicked, all widget values are sent to Streamlit in a batch
- Use “with” notation or call methods directly on the form
- Every form must have a submit button
- `st.button` and `st.download_button` can't be added to a form
- Can appear anywhere within your app, but can't be embedded inside other forms



Forms

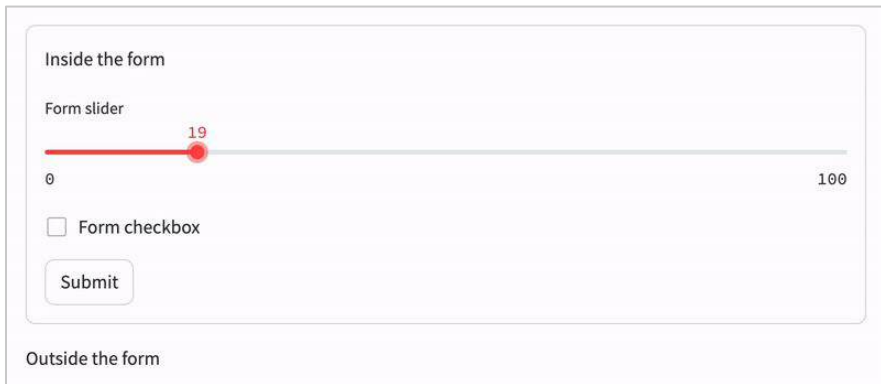
Batch elements together with a “Submit” button

```
import streamlit as st

with st.form("my form"):
    st.write("Inside the form")
    slider_val = st.slider("Form slider")
    checkbox_val = st.checkbox("Form checkbox")

    # Every form must have a submit button.
    submitted = st.form_submit_button("Submit")
    if submitted:
        st.write("slider", slider_val, "checkbox", checkbox_val)

st.write("Outside the form")
```



Inside the form

Form slider

19

0 100

Form checkbox

Submit

Outside the form

[Demo app](#)



Session state

Store and persist app state

- Share variables across reruns, for each user session

Initialize values

```
if 'key' not in st.session state:
    st.session_state['key'] = 'value'

# Attribute-based syntax
if 'key' not in st.session state:
    st.session_state.key = 'value'
```

Update values

```
# Attribute-like API
st.session_state.key = 'value2'

# Dictionary-like syntax
st.session_state['key'] = 'value2' #
```

Read values

```
st.write(st.session_state.key)

st.write(st.session_state)
```

Delete values

```
# Delete a single key-value pair
del st.session_state[key]

# Delete all items
for key in st.session state.keys():
    del st.session_state[key]
```

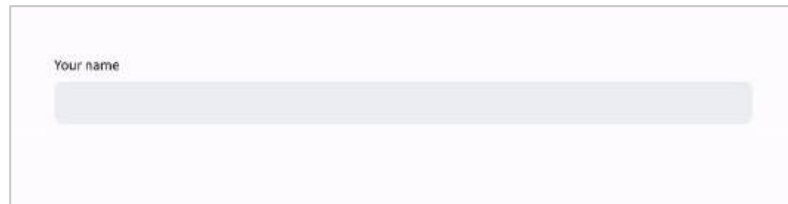


Session state: widgets

Store and persist app state

Every widget with a key is automatically added to session state

```
st.text_input("Your name", key="name")  
  
# This exists now:  
st.session_state.name
```



Your name



Session state: callbacks

Callbacks: Python functions that are called when an input widget changes – add them via **on_change** and **on_click** parameters and be sure to define the callback function above the widget declaration

When an input widget changes, the callback function gets executed **first**, and then the app is executed from top to bottom

Widgets that support **on_change**




st.checkbox, st.color_pickers, st.date_input,
st.data_editor, st.file_uploader, st.multiselect,
st.number_input, st.radio, st.select_slider,
st.selectbox, st.slider, st.text_area, st.text_input,
st.time_input, st.toggle

Widgets that support **on_click**

st.button, st.download_button, st.form_submit



Session state: limitations

- For forms, only the submit button has a callback (not widgets inside form)
- Callback functions are only supported on input widgets
- Modifying the value of a widget via the session state API (i.e. `st.session_state.my_slider = 7`) after instantiating isn't allowed 
StreamlitAPIException
- Setting widget state via the session state API and using the value parameter in your widget declaration isn't recommended  warning on the first run
- Setting the state of `st.button` and button-like widgets via the session state API isn't allowed  StreamlitAPIException



Fragments

Rerun a function separately from the rest of your app

st.experimental_fragment: decorator to turn a function into a fragment that can rerun independently from the full script

- When a user interacts with an input widget created by the fragment, only the fragment reruns (not the whole app!)
- **run_every** parameter allows you to auto-rerun fragment every X seconds, hours, days, etc.
- Trigger a full app rerun from inside a fragment with **st.rerun**



Fragments : example

Rerun a function separately from the rest of your app

```
import streamlit as st

if "script_runs" not in st.session_state:
    st.session_state.script_runs = 0
    st.session_state.fragment_runs = 0

@st.experimental_fragment
def example_fragment_func():
    st.session_state.fragment_runs += 1
    st.button("Rerun fragment")
    st.write(f"Fragment says it ran {st.session_state.fragment_runs} times.")

st.session_state.script_runs += 1
example_fragment_func()
st.button("Rerun full script")
st.write(f"Full script says it ran {st.session_state.script_runs} times.")
st.write(f"Full script sees that fragment ran {st.session_state.fragment_runs} times.")
```



Fragments : example

Rerun a function separately from the rest of your app

Rerun fragment

Fragment says it ran 1 times.

Rerun full script

Full script says it ran 1 times.

Full script sees that fragment ran 1 times.

[Demo app](#)



Deep dive on widget behavior

Widgets are session-dependent

- Widget state depends on a particular session.
- The actions of one user don't impact widgets of any other user's session.
- If a user opens multiple tabs to access the same app, each tab is a unique session (and changing the widget in one tab won't impact the same widget in other tabs).

Widgets return simple Python data types

- The value of a widget is a Python type – e.g. `st.button` returns a boolean
- The first time a widget function is called (before user interaction), it returns the default value.



Deep dive on widget behavior

Keys help distinguish widgets and access their values

- Widget keys help distinguish two otherwise identical widgets and create a way to access and manipulate widget value through session state.
- Streamlit can't understand two identical widgets on the same page without keys.

Widget order of operations


- When a user interacts with a widget, the order of operations is:
 - `st.session_state` value is updated
 - Callback function is executed
 - Page reruns with widget function returning new value

Remember: callback functions run before everything else



Deep dive on widget behavior

Widget statefulness

- Changing the parameters of a widget (e.g. default value, min or max value)  Streamlit will see it as a new widget/reset it
- You can get around this by using a separate key in session state to save widget values independently

Widget life cycle

- When a widget is called, Streamlit checks if the widget already exists with same parameters – if yes, reconnect; if no, create a new one
- Widget's IDs are based on parameters (like label, min or max value, default value, key, etc.)



Deep dive on widget behavior

Widget clean-up process

- When Streamlit gets to the end of a script run, it will delete data for any widgets in memory not rendered on-screen
 - All key-value pairs in `st.session_state` associated w/ widgets not on-screen will be deleted

If you want your widgets to retain statefulness when changing pages or modifying their parameters, you can either use separate keys stored in session state to persist widget values or **interrupt the widget clean-up process**



Deep dive on widget behavior

Interrupting the widget clean-up process

- When you manually save data to a key in session state, it becomes detached from any widget (in terms of the clean-up process)
- So you can prevent a widget's key-value pair from being deleted via the clean-up process by adding the following to the top of every page:

```
st.session_state.my_key = st.session_state.my_key
```



Widget behavior: summary

1. The actions of one user do not affect the widgets of any other user.
2. A widget function call returns the widget's current value (a simple Python type – e.g. `st.button` returns a boolean value).
3. Widgets return their default values on their first call before a user interacts with them.
4. A widget's identity depends on the arguments passed to the widget function. Changing a widget's label, min or max value, default value, placeholder text, help text, or key will cause it to reset.
5. If you don't call a widget function in a script run, Streamlit will delete the widget's information—including its key-value pair in Session State. If you call the same widget function later, Streamlit treats it as a new widget.



App testing: overview

Streamlit's app testing framework allows you to run headless tests on your app

- Use the **AppTest** class to simulate a running app – set up, manipulate, and inspect app contents via API
- Run the tests using a tool like **pytest** – tests are typically run locally or in a CI environment like GitHub Actions

We're going to walk through a quick example of Streamlit app testing using **pytest**



App testing: intro to `pytest`

`pytest` is a Python framework that makes it easy to write small, readable tests

- Name your test scripts `test_<name>.py` or `<name>_test.py`
- Within these files, each test is written as a function whose name must begin or end with `test`
- Add as many tests within a script as you want
- When you call `pytest`, all `test_<name>.py` files will be used for testing – each function within those files will be executed

Learn more about `pytest`  docs.pytest.org



App testing: example

Here's the structure of our example project:

```
myproject/  
├─ app.py  
├─ tests/  
  └─ test_app.py
```

Here's our simple app:

```
"""app.py"""  
import streamlit as st  
  
# Initialize st.session_state.beans  
st.session_state.beans = st.session_state.get("beans", 0)  
  
st.title("Bean counter :paw_prints:")  
  
addend = st.number_input("Beans to add", 0, 10)  
if st.button("Add"):  
    st.session_state.beans += addend  
st.markdown(f"Beans counted: {st.session_state.beans}")
```



App testing: example

Simple demo app

Bean counter

Beans to add

 - +

Add

Beans counted: 0



App testing: example

Testing file

```
"""test app.py"""
from streamlit.testing.v1 import AppTest

def test_increment_and_add():
    """A user increments the number input, then clicks Add"""
    at = AppTest.from_file("app.py").run()
    at.number_input[0].increment().run()
    at.button[0].click().run()
    assert at.markdown[0].value == "Beans counted: 1"
```

The app itself contains **4** elements: `st.title`; `st.number_input`;
`st.button`; `st.markdown`

The test contains a single test in the function `test_increment_and_add`



App testing: example

Let's break down the test itself

1. `at = AppTest.from_file("app.py").run()`

Initializes the simulated app and executes the first script run

2. `at.number_input[0].increment().run()`

Simulates a user clicking the plus icon to increment the number input (and the resulting script rerun)

3. `at.button[0].click().run()`

Simulates a user clicking the "Add" button (and the resulting script rerun)

4. `assert at.markdown[0].value == "Beans counted: 1"`

Check if the correct message is displayed at the end.



App testing: example

```
assert at.markdown[0].value == "Beans counted: 1"
```

- Assertions are the ♥ of tests! If the assertion is true, the test passes. If false, the test fails.
- Tests can have multiple assertions, but it is a good idea to focus your tests on a single behavior.

To run the tests, you'll open the command line, navigate to the project directory, and run the command **pytest**

```
cd myproject
```

```
pytest
```

```
(streamlitenv):/Users/sammy-streamlit/Documents/GitHub/myproject
● $ pytest
===== test session starts =====
platform darwin -- Python 3.11.5, pytest-7.4.3, pluggy-1.3.0
rootdir: /Users/sammy-streamlit/Documents/GitHub/myproject
collected 1 item

tests/test_app.py . [100%]

===== 1 passed in 0.67s =====
(streamlitenv):/Users/sammy-streamlit/Documents/GitHub/myproject
○ $
```



Visual customization

What are the options for visually customizing your Streamlit apps?

- Officially supported: **themes**
 - Simply add a `.streamlit/config.toml` file

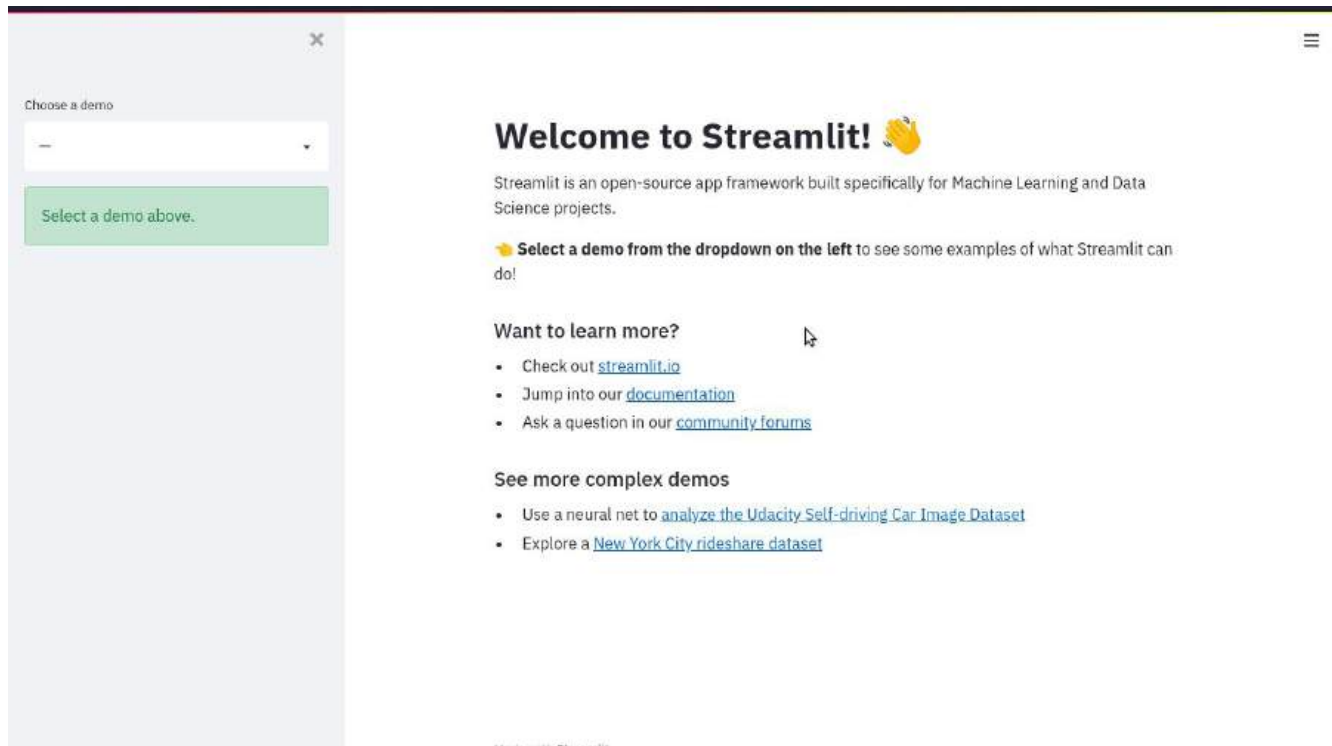
```
[theme]
primaryColor="#F63366"
backgroundColor="#FFFFFF"
secondaryBackgroundColor="#F0F2F6"
textColor="#262730"
font="sans serif"
```

- Theming allows you to customize the main colors used by your app and to choose between a few different fonts
- You can also use Streamlit's default Light and Dark themes



Visual customization

Default light and dark themes

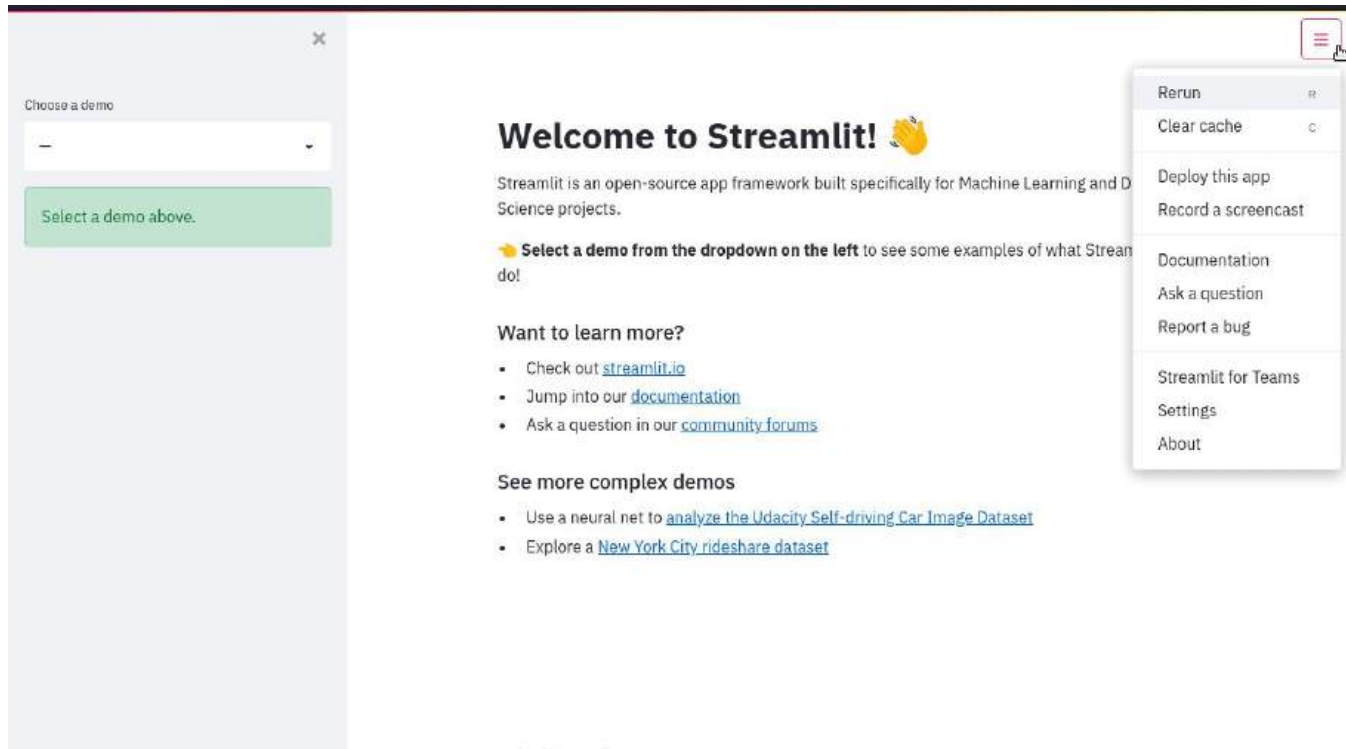


The screenshot shows a Streamlit application interface. On the left is a sidebar with a close button (X) at the top right. Inside the sidebar, there is a section titled "Choose a demo" with a dropdown menu and a green button labeled "Select a demo above.". On the right is the main content area with a hamburger menu icon (≡) at the top right. The main content area features a "Welcome to Streamlit!" heading with a clapping hands emoji, followed by a paragraph describing Streamlit as an open-source app framework for Machine Learning and Data Science projects. Below this is a yellow bullet point instruction: "Select a demo from the dropdown on the left to see some examples of what Streamlit can do!". There are two sections: "Want to learn more?" with three bullet points linking to streamlit.io, [documentation](#), and [community forums](#); and "See more complex demos" with two bullet points linking to [analyze the Udacity Self-driving Car Image Dataset](#) and [New York City rideshare dataset](#).



Visual customization

You can also customize the theme via the theme editor



The screenshot displays the Streamlit interface. On the left, a sidebar contains a 'Choose a demo' dropdown menu with a minus sign and a green button labeled 'Select a demo above.'. The main content area features a 'Welcome to Streamlit!' heading with a yellow robot icon, followed by a paragraph about Streamlit being an open-source app framework for Machine Learning and Data Science projects. Below this is a yellow callout box with a robot icon and the text: 'Select a demo from the dropdown on the left to see some examples of what Streamlit can do!'. Further down, there are sections for 'Want to learn more?' and 'See more complex demos', each with a list of links. On the right side of the interface, a menu is open, showing options: 'Rerun', 'Clear cache', 'Deploy this app', 'Record a screenshot', 'Documentation', 'Ask a question', 'Report a bug', 'Streamlit for Teams', 'Settings', and 'About'.



Visual customization

What are the options for visually customizing your Streamlit apps?

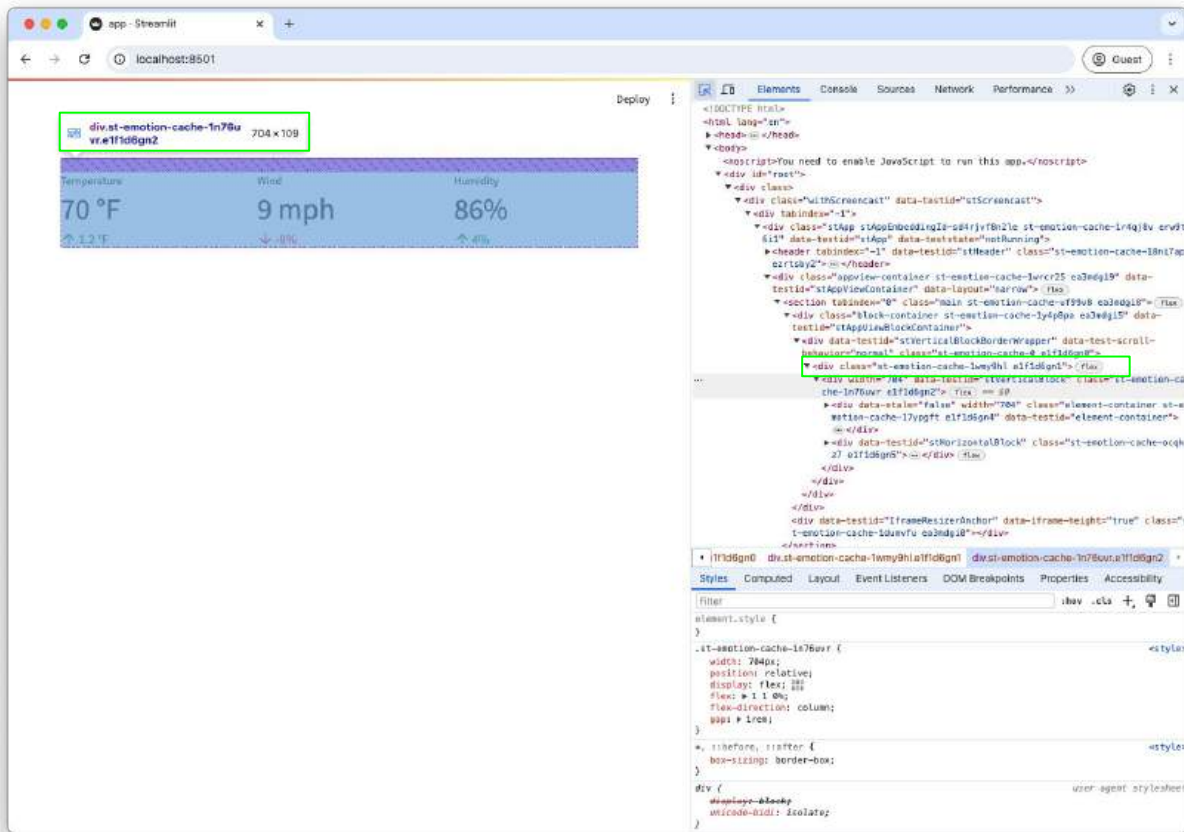
- Not officially supported: adding CSS via `st.markdown`
 1. Determine the CSS selectors for the elements you want to style (Developer > Inspect Elements in Chrome is helpful for this!)
 2. Add CSS in-line via `st.markdown` or read from a separate CSS file using `f.read` and `st.markdown`

Drawbacks

Using custom CSS with Streamlit isn't officially supported – class names and IDs of Streamlit widgets and page elements may change in future releases of Streamlit, in which case you'll have to update your custom CSS to keep the styling consistent.



Visual customization



The screenshot shows a web browser at localhost:8501 displaying a weather application. The application shows a weather card with the following data:

Temperature	Wind	Humidity
70 °F	9 mph	86%

The browser's developer tools are open, showing the 'Elements' panel. The selected element is a `div` with the following HTML structure:

```
<div class="st-emotion-cache-1e76uv" data-test-id="stScreenact">
```

The 'Styles' panel shows the default styles for this element:

```
element.style {
}
.st-emotion-cache-1e76uv {
  width: 704px;
  position: relative;
  display: flex;
  flex-direction: column;
  gap: 16px;
}
*::before, *::after {
  box-sizing: border-box;
}
div {
  font-family: inherit;
  font-size: inherit;
}
```

1. Identify CSS selectors via “Inspect Elements” in Chrome (“Inspector” in Firefox; “Web Inspector” in Safari)



Visual customization

```
"""style.css"""
div.st-emotion-cache-ocqkz7 {
  background-color: #EEEEEE;
  border: 2px solid #CCCCCC;
  padding: 5% 5% 5% 10%;
  border-radius: 5px;
}
```

2. Use those selectors to add CSS to the app

```
import streamlit as st

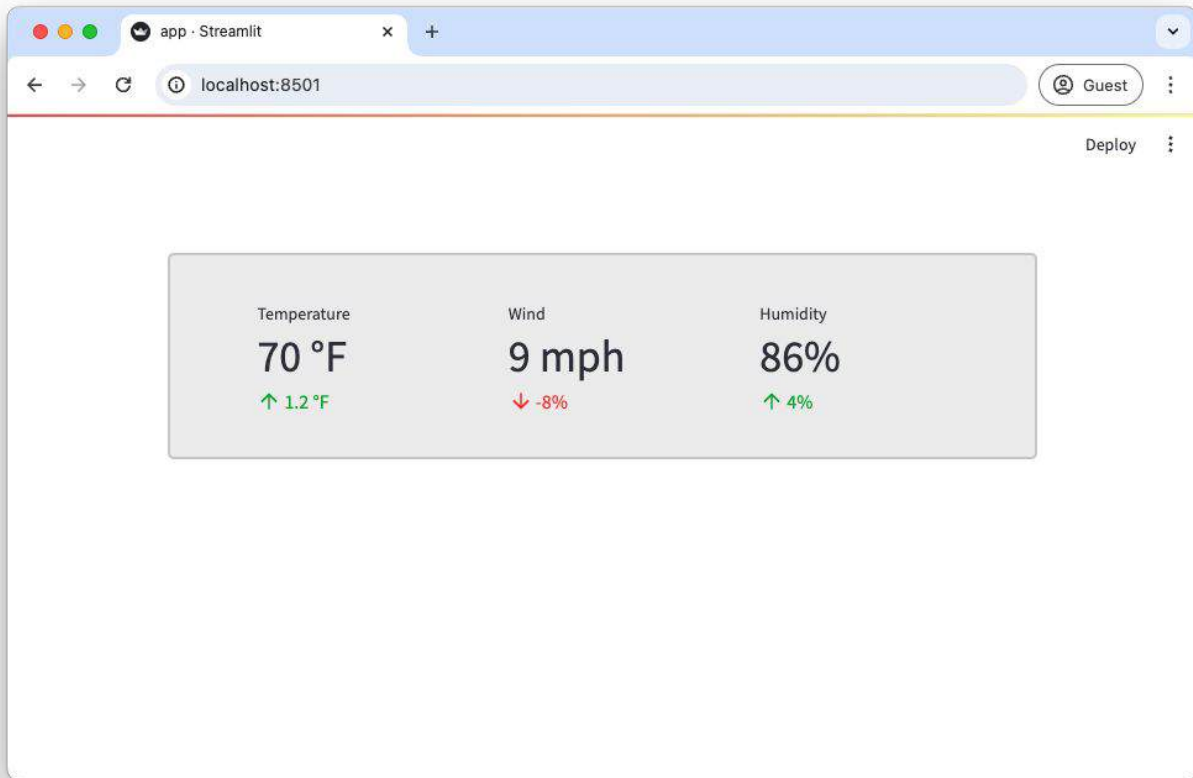
with open('style.css') as f:
    st.markdown(f'<style>{f.read()}</style>',
                unsafe_allow_html=True)

col1, col2, col3 = st.columns(3)
col1.metric("Temperature", "70 °F", "1.2 °F")
col2.metric("Wind", "9 mph", "-8%")
col3.metric("Humidity", "86%", "4%")
```

3. Read `style.css` file via `st.markdown`



Visual customization



4. Bask in the glory of your updated UI 🥳



Q&A ?



Roadmap: what's in the works?



Roadmap: what's coming?

In the next 3 months

 **Improve the anchor button**  Ready for launch

 **Selections on charts**  Testing

Catch selection events on charts and do something with them in your app!

 **Selections on dataframes**  Testing

Catch click events on tables and do something with them in your app!

 **Multipage apps v2 - Public Preview**  Development

More options for building multipage apps, like more and better custom navigation options built in.

 **Support more dataframe formats (Polars, dataframe interchange protocol, ...)**  Development

 **pathlib.Path support for file paths**  Planning

Support pathlib.Path objects across Streamlit commands that accept traditional strings and file objects today



Roadmap: what's coming?

In the next 3 months

 **Migrate fullscreen button to toolbar**  Planning

 **Fix vertical alignment issues**  Planning

More easily align elements vertically, e.g. widgets with and without label.

 **User feedback widget (for chat)**

 **Add position parameter to `st.chat_input`**



Roadmap: what's coming?

In the next 6 months

 **File upload for `st.chat_input`**  Planning

 **Clickable examples for `st.chat_input`**  Planning

Add example prompts to `st.chat_input` that users can click on.

 **Don't rerun when clicking `st.download_button`**

 **Expander/tabs: run only when opened**



Roadmap: what's coming?

Beyond



AppTest improvements 👤 Development

Support additional st commands and easier lookups



Accept new items in `st.selectbox` and `st.multi_select` 🔪 Planning



Allow `st.file_uploader` to return file path instead of `UploadedFile`



Components v2

An improved API to create custom components.



Chart builder

Build charts with a simple point-and-click interface, then add them as code to your app.









`st.app_state`

A tiny NoSQL database for every app in Streamlit Cloud!



Roadmap: what's coming?

Beyond

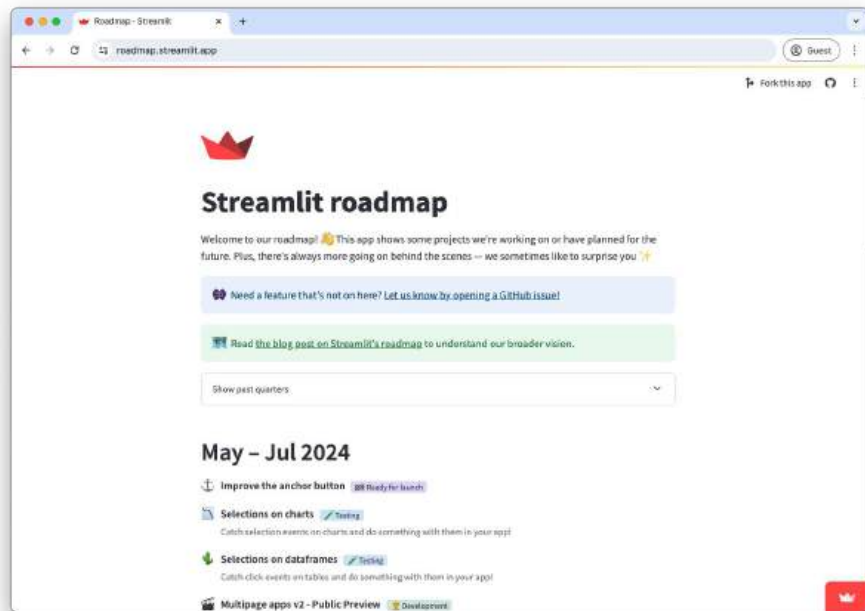
-  **Data wrangler to Pandas code**
Filter, pivot, aggregate your data right in the `st.dataframe` UI, then copy it as Pandas code to your app.
-  **Clicks and selections on maps**
Let the user click on a map and process the click position on the Python backend.
-  **Deferred data load for `st.download_button`**
`st.download_button` accepts a function, which is only executed once it's clicked.
-  **Data wrangler**
`st.dataframe` shows summary statistics and lets you filter, sort, pivot, aggregate, sort, etc.
-  **Higher-level chat API**
-  **Speed + performance initiative**
Make Streamlit run and seem faster.



Roadmap: what's coming?

Strong feelings on our roadmap? Let us know!

github.com/streamlit/streamlit/issues



Keep tabs on our plans:
roadmap.streamlit.app

