Streamlit

Streamlit turns a plain Python script into an interactive data app, covered here through widgets, layout, caching, and the rerun model.

Text & Titles

st.write is the swiss-army display; titles and markdown structure the page.

PURPOSE	COMMAND	RESULT
Display almost anything.	`st.write("Hello", df, fig)`	
Page and section headings.	`st.title("My App"); st.header("Section")` `st.subheader("Sub")`	
Rich markdown text.	`st.markdown("**bold** and `code`")`	
Muted helper caption.	`st.caption("updated hourly")`	
Bare value, no function (magic).	`df  # a variable alone on a line`	
Pretty-print a dict as JSON.	`st.json({"id": 1, "ok": True})`	

st.write renders almost anything top to bottom; title/header/subheader structure the page; a bare variable auto-displays by magic

Input Widgets as Variables

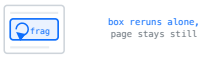
Each widget is a function that returns its current value.

PURPOSE	COMMAND	RESULT
Read a line of text.	`name = st.text_input("Name", value="ada")`	
Pick a number on a slider.	`age = st.slider("Age", 0, 100, 30)`	
Choose from options.	`city = st.selectbox("City", ["NYC", "LA"])`	
Pick several.	`tags = st.multiselect("Tags", options)`	
A boolean toggle.	`agree = st.checkbox("I agree")`	
A momentary action.	`if st.button("Run", type="primary"): ...`	

No callbacks: reading input is just value = st.widget(...); st.button returns True only on the rerun it was clicked

The Rerun Model

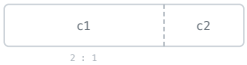
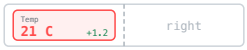
Interaction reruns the whole script top to bottom.

PURPOSE	COMMAND	RESULT
The default loop.	*(any widget change)*	
Stop early on a condition.	`st.stop()`	
Programmatic rerun.	`st.rerun()`	
Batch inputs, submit once.	`with st.form("f"): ...` `st.form_submit_button("Go")`	
Rerun only one part.	`@st.fragment` over a function	
Avoid the dead spelling.	`st.rerun()` not `st.experimental_rerun()`	

Any interaction reruns the script top to bottom; stop(), rerun(), st.form and @st.fragment control when and how much reruns

Layout: Columns, Sidebar, Tabs

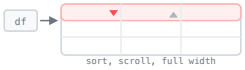
Place elements side by side, in a sidebar, or behind tabs.

PURPOSE	COMMAND	RESULT
Split into columns.	`left, right = st.columns(2)`	
Weighted column widths.	`c1, c2 = st.columns([2, 1])`	
Write into a column.	`left.metric("Temp", "21 C", "+1.2")`	
A left sidebar.	`with st.sidebar:` `  st.slider("Year", 2000, 2026)`	
Tabbed sections.	`t1, t2 = st.tabs(["Chart", "Data"])`	
Collapsible detail.	`with st.expander("Details"):`	

st.columns / st.sidebar / st.tabs / st.expander each return a container you write into with `with` or `.method(...)`

Display Data

Turn a DataFrame into an interactive table or editor.

PURPOSE	COMMAND	RESULT
Interactive table.	`st.dataframe(df, width="stretch")`	
Static table.	`st.table(df.head())`	
Editable grid.	`edited = st.data_editor(df, num_rows="dynamic")`	
Single KPI card.	`st.metric("Revenue", "$1.2M", "8%")`	
Style or configure columns.	`st.dataframe(df, column_config={"price": st.column_config.NumberColumn(format="$%.2f")})`	
Use the freshest API.	`width="stretch"` *not* `use_container_width`	

st.dataframe for an interactive grid, st.table for static, st.data_editor to edit; st.metric for one KPI, column_config to format

Charts

Built-in charts for quick looks; hand any library a slot.

PURPOSE	COMMAND	RESULT
Quick line chart.	`st.line_chart(df, x="date", y="sales")`	
Quick bar chart.	`st.bar_chart(df, x="day", y="count")`	
Quick scatter.	`st.scatter_chart(df, x="x", y="y")`	
Embed a Matplotlib figure.	`st.pyplot(fig)`	
Embed an Altair / Plotly chart.	`st.altair_chart(chart), st.plotly_chart(fig)`	
A point on a map.	`st.map(df)`	

Built-in line/bar/scatter charts take a DataFrame and x/y columns; st.pyplot, st.altair_chart, st.plotly_chart, and st.map embed any library figure

Caching & session_state

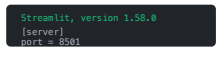
Cache expensive work; remember values across reruns.

PURPOSE	COMMAND	RESULT
Cache data-returning work.	`@st.cache_data` `def load(url): ...`	
Expire the cache.	`@st.cache_data(ttl="1h")`	
Cache a connection or model.	`@st.cache_resource` `def get_db(): ...`	
Persist a value across reruns.	`if "n" not in st.session_state:` `  st.session_state.n = 0`	
Read and update state.	`st.session_state.n += 1`	
Avoid the dead decorator.	`@st.cache_data/@st.cache_resource`	

`@st.cache_data caches data by args, @st.cache_resource holds one shared instance, st.session_state remembers per-user values across reruns`

Run & Deploy

From local script to a shared URL.

PURPOSE	COMMAND	RESULT
Run locally.	`streamlit run app.py`	
Configure the page.	`st.set_page_config(page_title="My App",` `  layout="wide")`	
Pin dependencies.	`requirements.txt` `streamlit==1.58.0`	
Try a built-in theme demo.	`streamlit hello`	
Deploy to Community Cloud.	`push to GitHub, connect repo on Cloud`	
Quick health check.	`streamlit version` `streamlit config show`	

`streamlit run app.py serves at localhost:8501; pin in requirements.txt, push to GitHub, deploy free on Community Cloud`