Introduction to Streamlit

What Is Streamlit?

Streamlit is an open-source Python framework for quickly creating interactive web apps. It turns data scripts into shareable apps with minimal code, perfect for dashboards and AI applications.

Installation: Use the pip install streamlit command to install Streamlit.

Running an app: Use the streamlit run first_app.py command to run your first Streamlit app.

Import convention: The common convention is to import Streamlit as st in the following manner:

```
import streamlit as st
```

To update your app, save the source file. Streamlit will detect changes and prompt a rerun. Choose "Always rerun" from the top-right menu for automatic refresh.

Streamlit apps run like Python scripts, rerunning from top to bottom on code changes or user interactions. Callbacks via on_change or on_click execute before the rest of the script.

How to Display Text in Streamlit

The st.write command in Streamlit writes different types of content to your app, including text, data, and charts, depending on the input. Here's how to use it:

```
st.write("Hello, *World*!") # Displays the string.

df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
st.write(df) # Displays the DataFrame

try:
       x = 1 / 0
except Exception as e:
       st.write(e) # Displays an error message

# Dictionary
my_dict = {'key': 'value'}
st.write(my_dict) # Displays a dictionary
```

The st.write function displays different data types differently, ensuring that each type is presented appropriately within the Streamlit app.

The st.write_stream command streams a generator, iterable, or sequence to the app. It iterates through the sequence, displaying string chunks with a typewriter effect and other data types using st.write. Here's how to use it:

```
lorem = """
Lorem ipsum dolor sit amet, **consectetur adipiscing** elit, sed do
eiusmod tempor.
"""

# Generator function to stream data word by word
def stream_data():
    for word in lorem.split(" "):
        yield word + " " # Yield each word followed by a space
        time.sleep(0.02) # Pause for 20 milliseconds between words

st.write_stream(stream_data) # Stream the generated data to the
Streamlit app
```

The st.markdown command lets you render markdown-formatted text in your Streamlit app. With st.markdown, you can easily incorporate rich text elements like **bold**, *italic*, and ***bold italic*** text, emojis, and colored text.

```
import streamlit as st

# Markdown with colored text

st.markdown(''' :red[Streamlit] :orange[can] :green[write] :blue[text]
:violet[in] :gray[pretty] :rainbow[colors] and :blue-background
[highlight] text. ''')
```

Output:	Streamlit can write text in pretty colors and highlight text.

Other tools include st.latex, which displays mathematical expressions formatted as LaTeX, and st.text, which displays fixed-width and preformatted text.

There are other widgets related to text presentation. A detailed breakdown is provided in the table below:

Function	Description	Example
st.echo	Uses a with block to draw some code on the app and executes it.	`with st.echo(): st.write('This code will be printed')`
st.title	Displays text in title formatting.	`st.title('This is a title')`
st.header	Displays text in header formatting.	`st.title('This is a header with a divider')`
st.divider	Displays a horizontal rule.	`st.divider()`
st.code	Displays a code block with optional syntax highlighting.	`code = '''def hello(): print("Hello, Streamlit!")''' st.code(code, language='python')`

How to Display Data in Streamlit

Displaying data in a Streamlit app is straightforward and intuitive. Streamlit provides various commands to display data types, such as DataFrames, tables, and JSON objects.

The st.dataframe command displays an interactive table in your app, working with DataFrames from pandas, PyArrow, Snowpark, PySpark, and other convertible types.

Users can sort columns, filter data, and adjust the table size.

```
import streamlit as st
import pandas as pd

df = pd.DataFrame({
    'A': [1, 2, 3],
    'B': [4, 5, 6]
})

st.dataframe(df)
```

	A	B
0	1	4
1	2	5
2	3	6

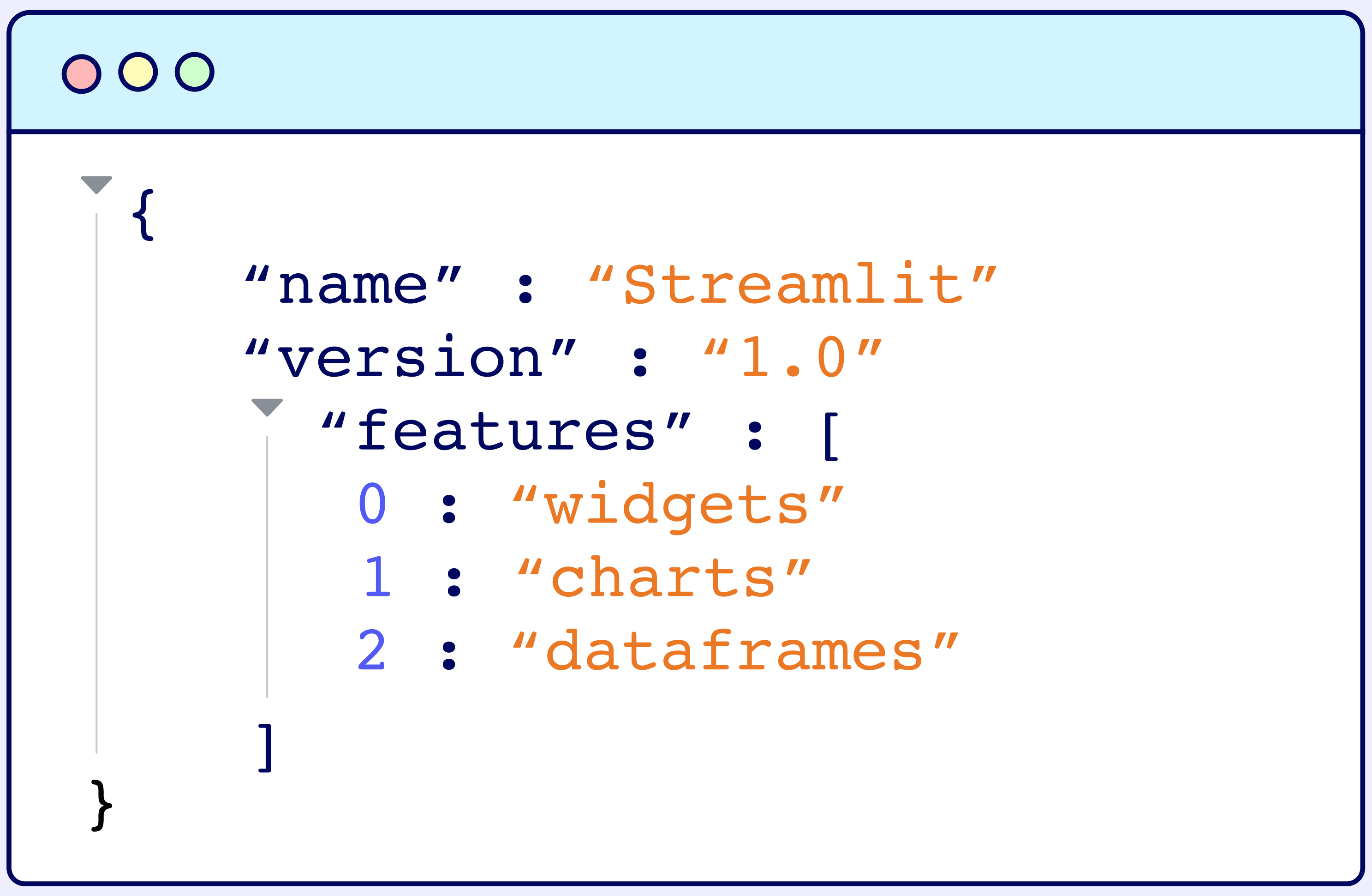
Use st.table to display a static table. This command is useful when you want to show a DataFrame or a list in a noninteractive format.

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

	A	B
0	1	4
1	2	5
2	3	6

The st.json command displays JSON data in an easy-to-read format. It is ideal for showcasing API responses or other JSON-formatted data.

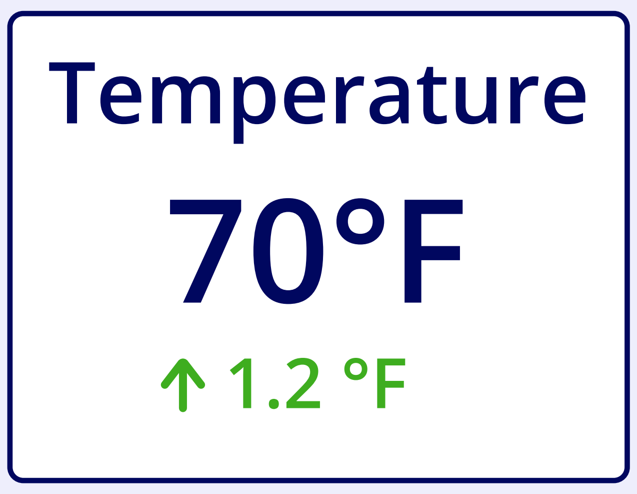
```
json_data = {
    'name': 'Streamlit',
    'version': '1.0',
    'features': ['widgets', 'charts', 'dataframes']
}

st.json(json_data)
```



```
{
    "name" : "Streamlit"
    "version" : "1.0"
    "features" : [
      0 : "widgets"
      1 : "charts"
      2 : "dataframes"
    ]
}
```

The st.metric command displays a single metric, such as a KPI, with an optional delta to show the change from a previous value.

```
st.metric(label="Temperature", value="70 °F", delta="1.2 °F")
```



Temperature

70°F

↑ 1.2 °F

If you need to show a large number, consider abbreviating it with tools such as Millify. For example, you can represent 2315 as 2k as follows:

```
from millify import millify

st.metric("Short number", millify(2315))
```

Also, Streamlit's magic commands help display markdown, data, and charts without explicit commands. Place the content on its line to appear in your app. For example:

```
df = pd.DataFrame({'col1': [1,2,3]})
df  # Draw the DataFrame
```

How to Display Media Elements in Streamlit

Streamlit easily incorporates various media types, such as images, audio, and video, into your app.

The st.image command displays images in your app. You can load images from a file, a URL, or a NumPy array. You can also adjust the image's width and use captions.

```
import streamlit as st

# Display an image from a file
st.image("path/to/image.png", caption="Sample Image", width=300)

# Display an image from a URL
st.image("https://example.com/image.png", caption="Online Image",
width=300)
```

The st.audio command allows you to embed audio files in your app. You can load audio from a file or a URL and control the format and start time.

```
# Display an audio file from a file
st.audio("path/to/audio.mp3")

# Display an audio file from a URL
st.audio("https://example.com/audio.mp3")
```

The st.video command lets you embed video files in your app. You can load videos from a file or a URL and set the start time.

```
# Display a video from a file
st.video("path/to/video.mp4")

# Display a video from a URL
st.video("https://example.com/video.mp4", start_time=10)
```

The st.logo renders a logo in the upper-left corner of your app and its sidebar.

```
LOGO_URL_LARGE = "Educative_Icon_Color.png"

st.logo(LOGO_URL_LARGE)
```

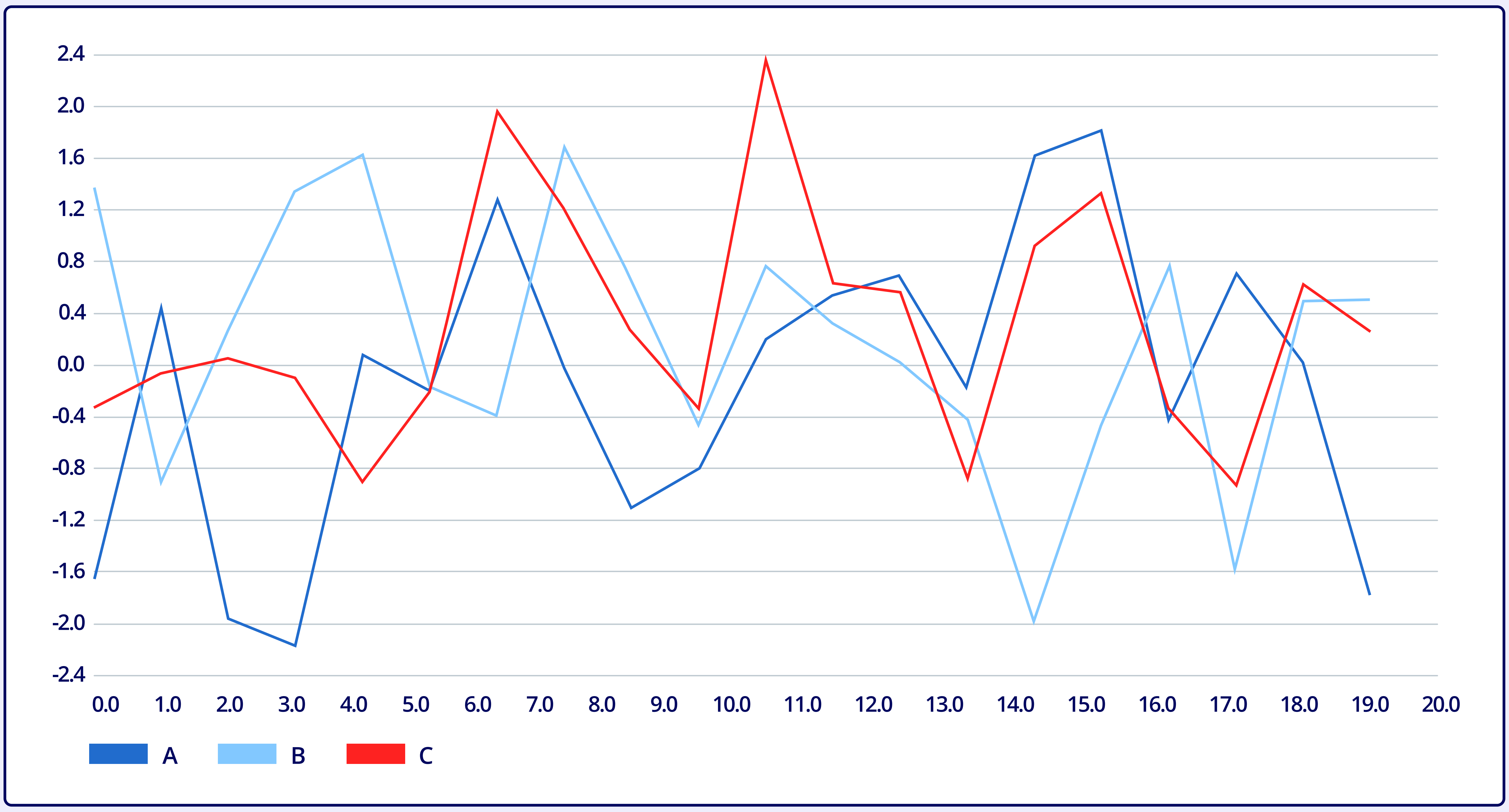
Display

These commands make adding visual and auditory elements to your app easy.

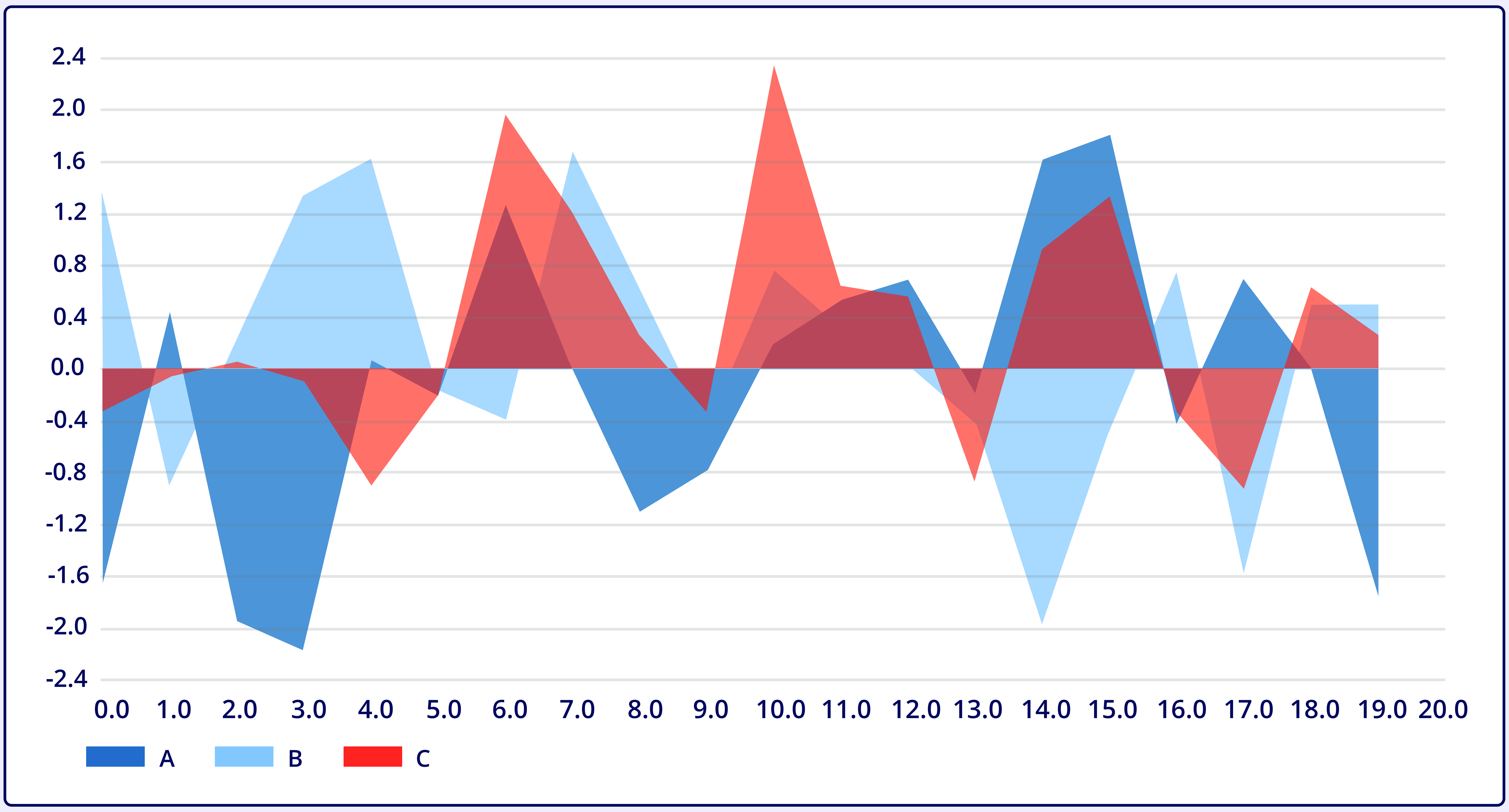
How to Create Graphs/Charts in Streamlit

The st.line_chart command displays a line chart. It is suitable for visualizing trends over time.

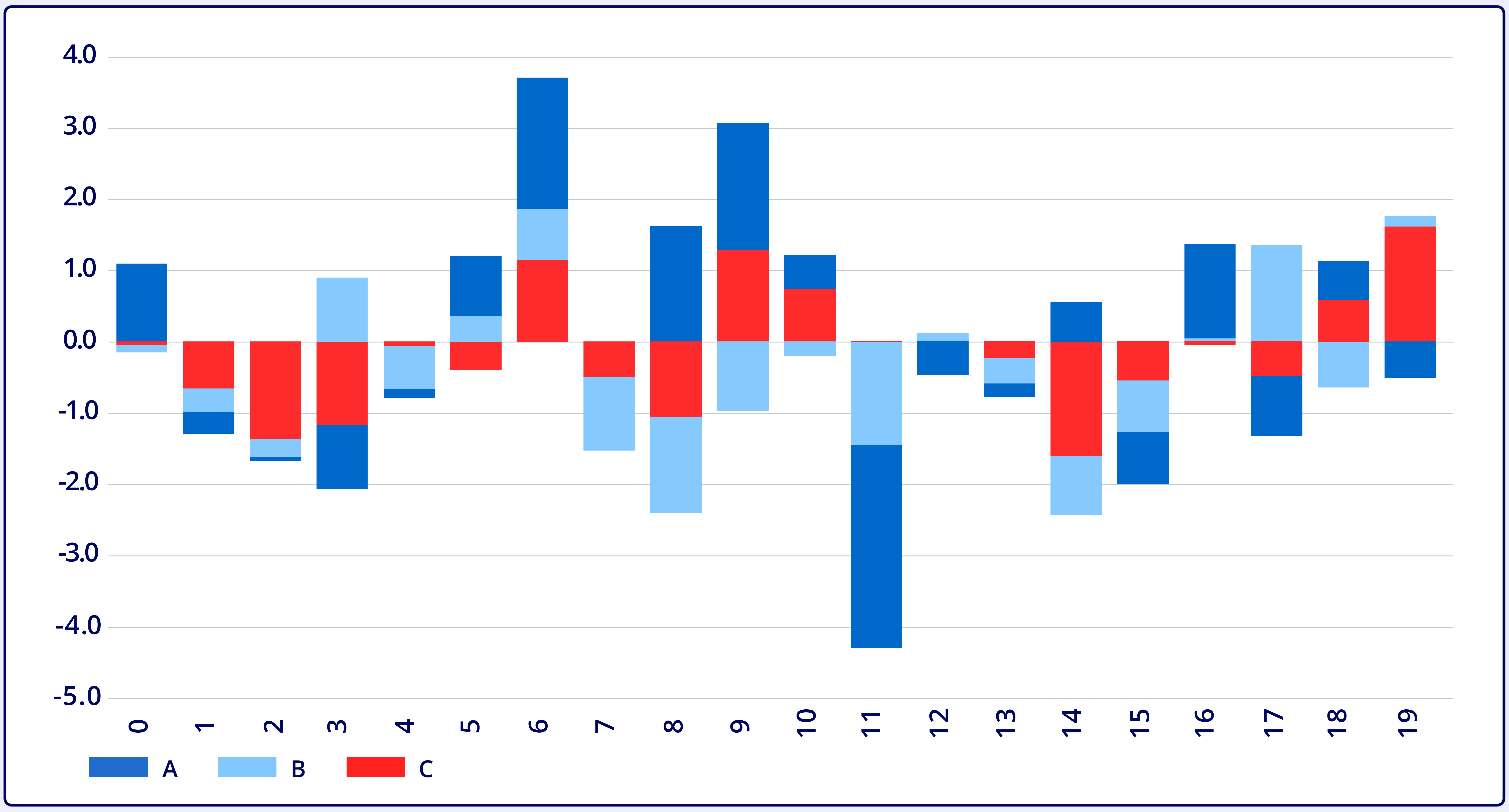
```
# Sample data
data = pd.DataFrame(np.random.randn(20, 3), columns=['A', 'B', 'C'])

st.line_chart(data)
```



The st.area_chart(data) command displays an area chart similar to a line chart but with the area below the line filled.

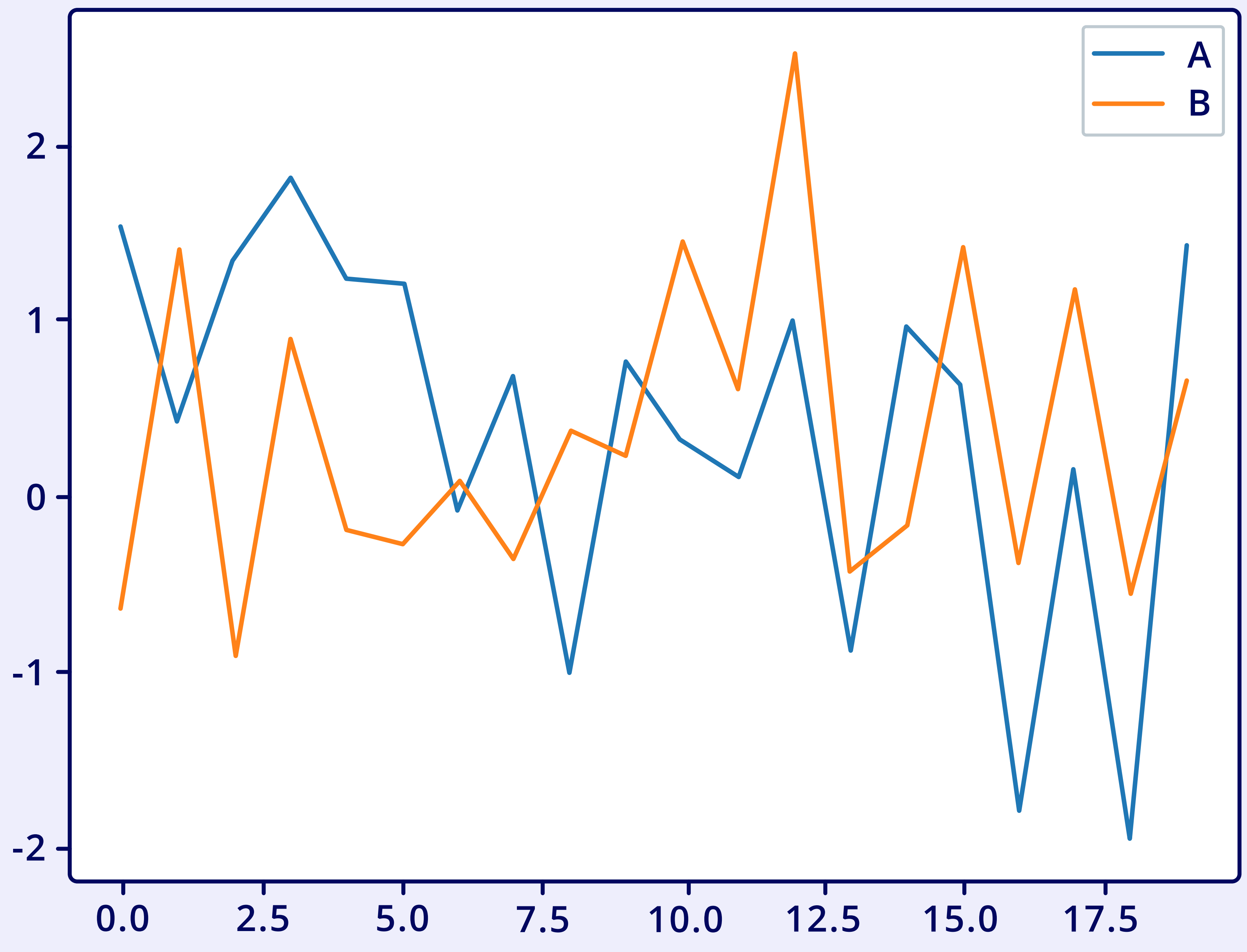


The st.bar_chart(data) command displays a bar chart that compares quantities across different categories.



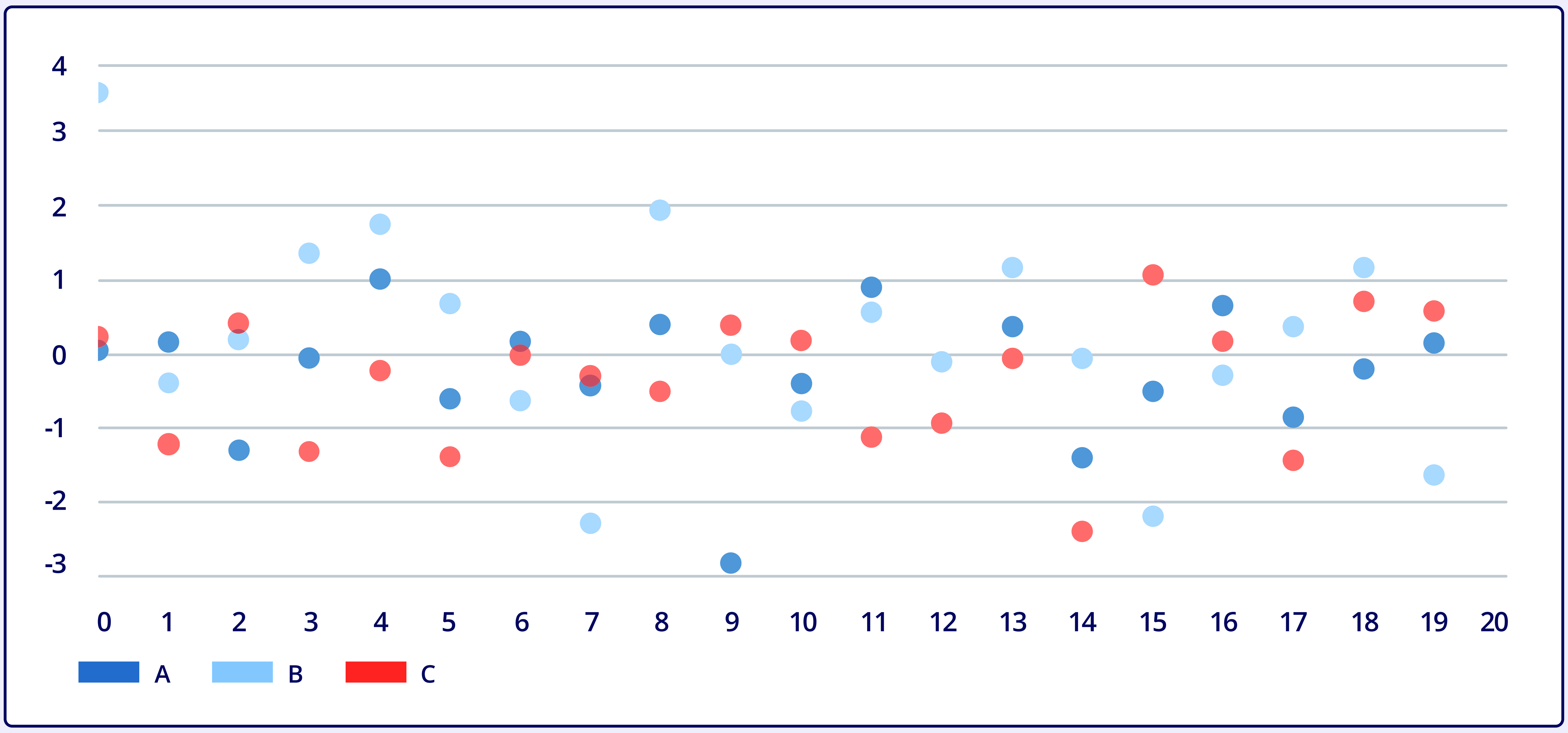
The st.pyplot command allows you to display a Matplotlib figure.

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.plot(data['A'], label='A')
ax.plot(data['B'], label='B')
ax.legend()

st.pyplot(fig)
```



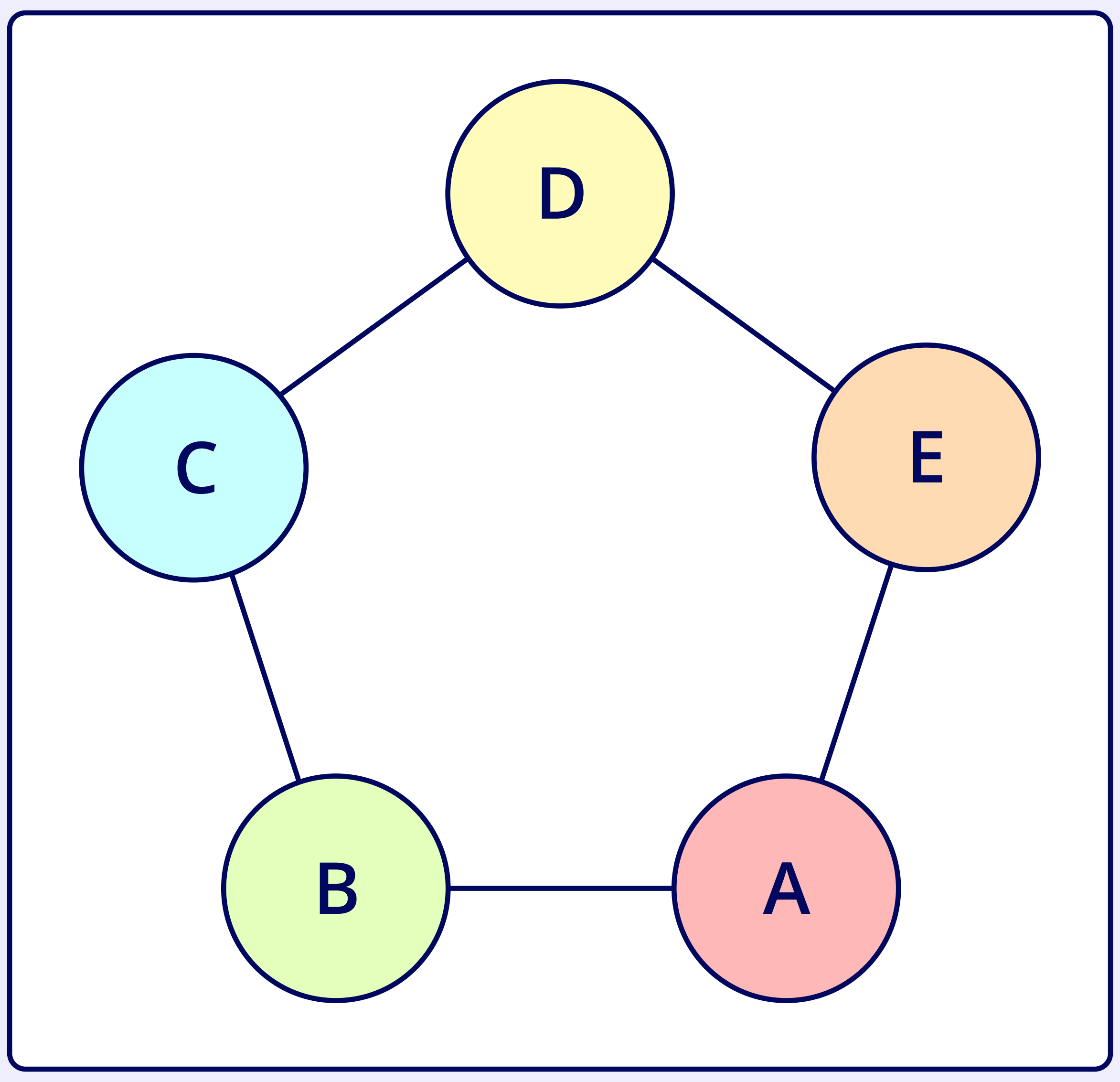
The st.scatter_chart displays a scatter plot chart.

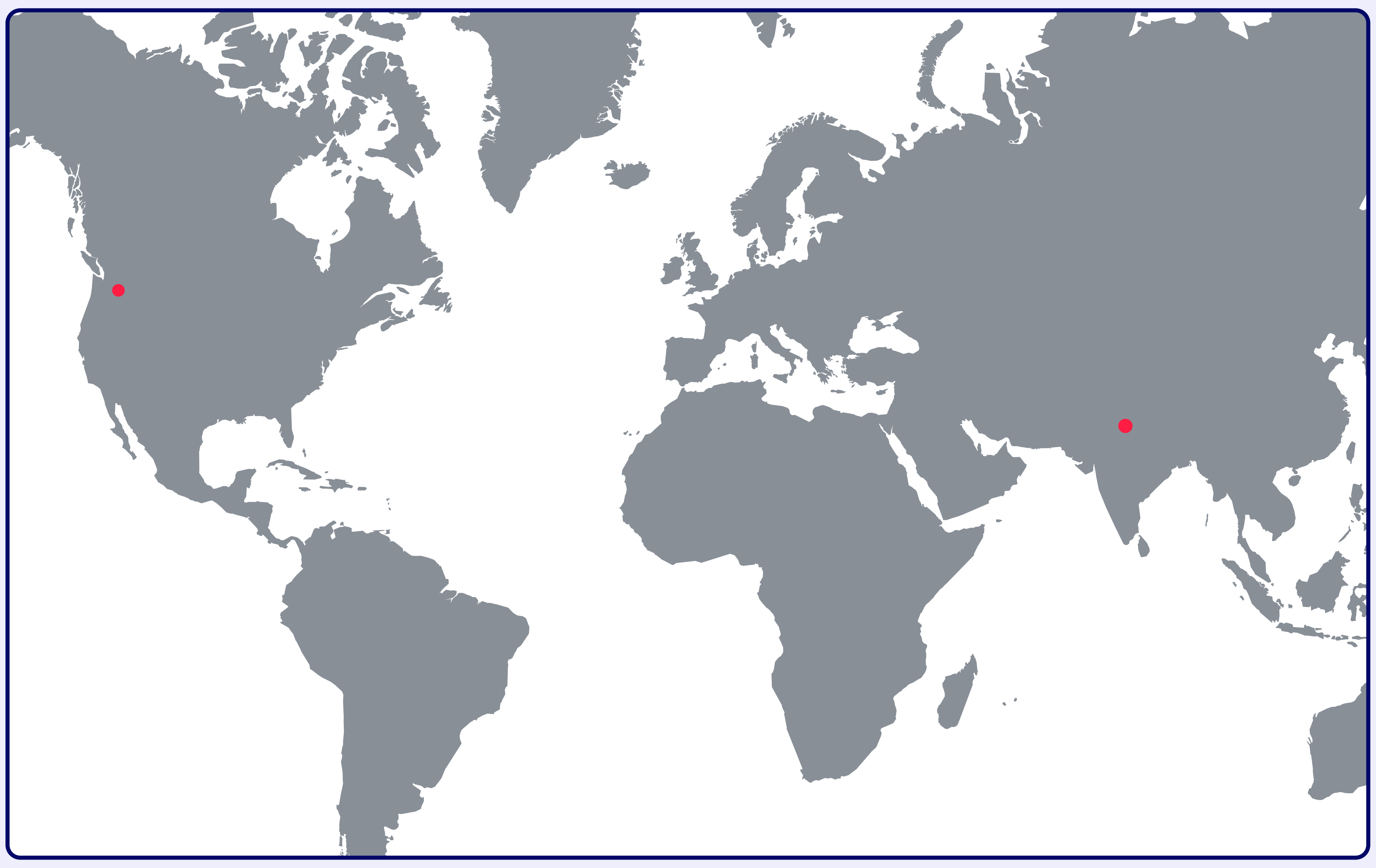
```
chart_data = pd.DataFrame(np.random.randn(20, 3), columns=["a", "b",
"c"])

st.scatter_chart(chart_data)
```



The st.graphviz_chart command displays a chart created with Graphviz—a graph visualization software. The st.map command displays a map with points. It is useful for visualizing geospatial data.

```
graphviz_chart = '''
    # Graphviz details here
'''
st.graphviz_chart(graphviz_chart)

# Sample geospatial data
map_data = pd.DataFrame({
    'lat': [47.62205762608492, 31.561920],
    'lon': [-122.1766676528772, 74.348080]
})
st.map(map_data)
```



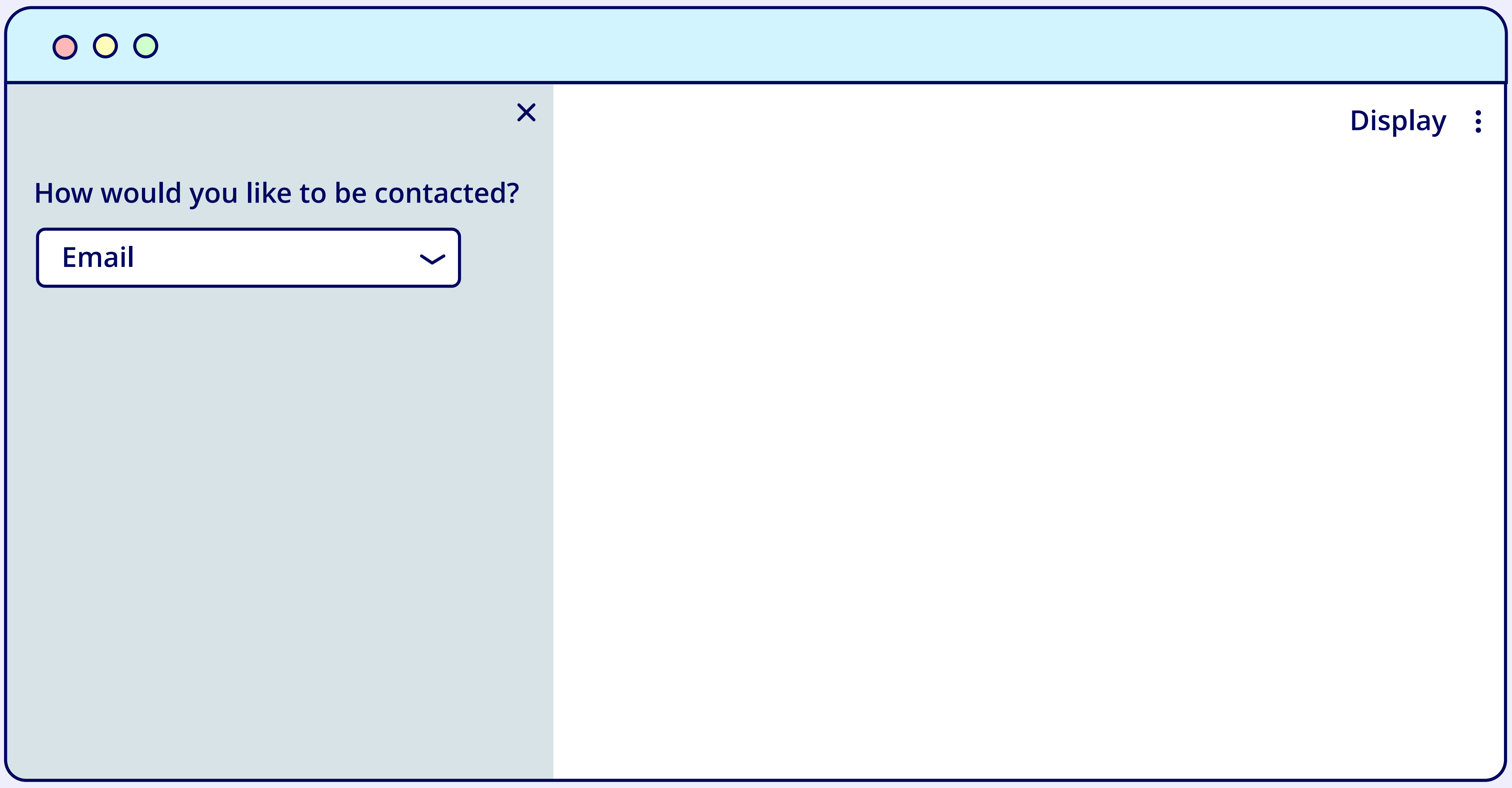


How to Structure the Layout in Streamlit

Streamlit provides several layout and container elements to help organize and structure your app. These elements allow you to create sidebars, columns, expandable sections, and tabs.

The st.sidebar command creates a sidebar for adding widgets and controls. Moving less critical elements to the left side of the page can help declutter the main interface. The command is st.sidebar.[element_name].

```
add_selectbox = st.sidebar.selectbox(
    "How would you like to be contacted?",
    ("Email", "Home phone", "Mobile phone")
)
```



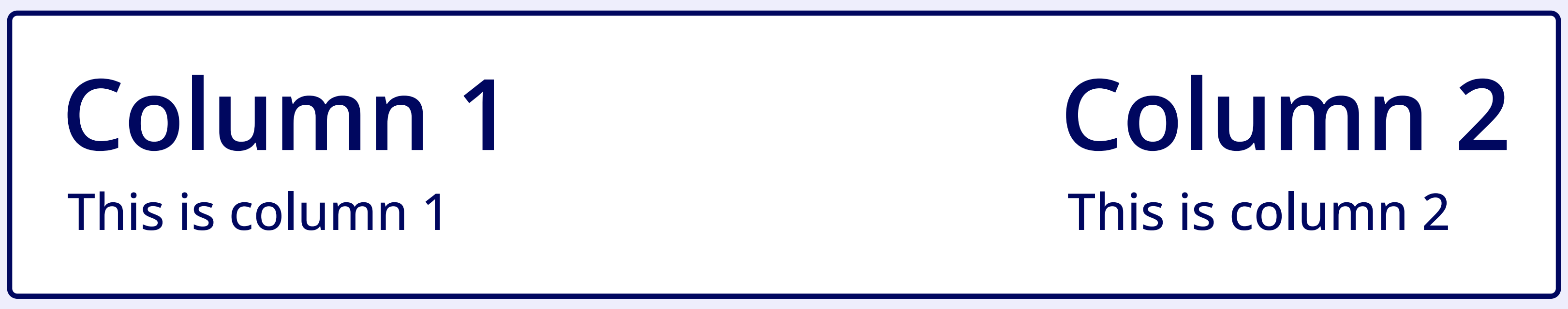
How would you like to be contacted?
Email

The st.container command creates a container that groups multiple elements. This is useful for organizing content logically within your app.

```
# Using a container to group elements
with st.container():
    st.write("This is inside the container")

st.write("This is outside the container")
```

The st.columns command allows you to create multiple columns in your app. You can specify the number of columns and their relative widths.

```
# Creating two columns
col1, col2 = st.columns(2)

with col1:
    st.header("Column 1")
    st.write("This is column 1")

with col2:
    st.header("Column 2")
    st.write("This is column 2")
```


Column 1
This is column 1
Column 2
This is column 2

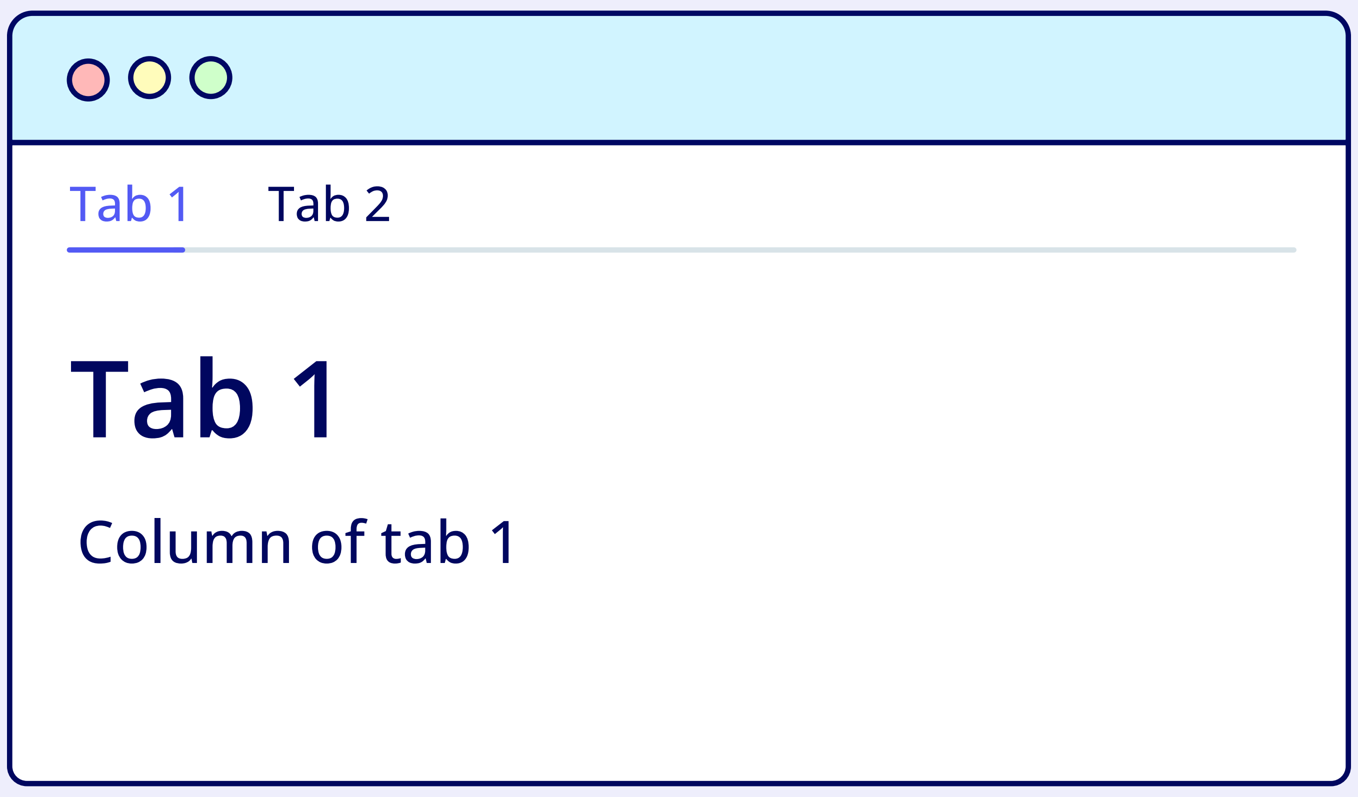
The st.expander command creates an expandable section that hides or shows content. This helps keep the interface clean and allows users to expand sections as needed.

```
# Creating an expander
with st.expander("Expand to see more"):
    st.write("This content is hidden until expanded")
    st.line_chart({"data": [0, 1, 2]})
```

Expand to see more

The st.tabs command allows you to create tabbed sections in your app. Each tab can contain different elements, making organizing content into separate views easy.

```
# Creating tabs
tab1, tab2 = st.tabs(["Tab 1", "Tab 2"])

with tab1:
    st.header("Tab 1")
    st.write("Content of tab 1")

with tab2:
    st.header("Tab 2")
    st.write("Content of tab 2")
```



Tab 1

Column of tab 1

How to Capture Input by User in Streamlit

Streamlit provides a wide range of input widgets that allow users to interact with your app. These widgets can capture various input types, from simple text and numbers to files and colors.

The st.button command creates a clickable button for users. It returns True if the button is clicked.

```
# Button
if st.button('Click me'):
    st.write('Button clicked!')
```

The st.download_button command creates a button that allows users to download a file.

```
# The download button
text = "Hello, Streamlit!"
st.download_button(label="Download Text", data=text, file_name=
"hello.txt")
```

The st.checkbox command creates a checkbox that can be toggled on or off. It returns True if the checkbox is checked.

```
# Checkbox
if st.checkbox('Show message'):
    st.write('Checkbox checked!')
```

The st.radio command creates a set of radio buttons for selecting a single option from a list.

```
# Radio buttons
option = st.radio('Choose an option:', ['Option 1', 'Option 2',
'Option 3'])
st.write('You selected:', option)
```

The st.selectbox command creates a drop-down menu for selecting a single option from a list.

```
# Select box
option = st.selectbox('Choose an option:', ['Option 1', 'Option 2',
'Option 3'])
st.write('You selected:', option)
```

Alternatively, the st.multiselect command creates a drop-down menu for selecting multiple options from a list.

```
# Multiselect
options = st.multiselect('Choose options:', ['Option 1', 'Option 2',
'Option 3'])
st.write('You selected:', options)
```

The st.slider command creates a slider for selecting a range of values.

```
# Slider
value = st.slider('Select a value:', 0, 10000, 2315)
st.write('Value selected:', value)
```

Alternatively, the st.select_slider command creates a slider for selecting a value from a list of options.

```
# Select slider
value = st.select_slider('Select a value:', options=['A', 'B', 'C'],
value='B')
st.write('Value selected:', value)
```

The st.text_input command creates a text input box for entering a single line of text.

```
# Text input
text = st.text_input('Enter some text:')
st.write('You entered:', text)
```

Similarly, the st.number_input command creates a numeric input box for entering a number.

```
# Number input
number = st.number_input('Enter a number:', min_value=0, max_value=100,
value=50)
st.write('Number entered:', number)
```

The st.text_area command creates a text area for entering multiple lines of text.

```
# Text area
text = st.text_area('Enter some text:')
st.write('You entered:', text)
```

The st.date_input command creates a date input for selecting a date.

```
# Date input
date = st.date_input('Select a date:')
st.write('Date selected:', date)
```

Similarly, the st.time_input command creates a time input for selecting a time.

```
# Time input
time = st.time_input('Select a time:')
st.write('Time selected:', time)
```

The st.file_uploader command creates a file uploader for uploading files.

```
# File uploader
uploaded_file = st.file_uploader('Upload a file:')
if uploaded_file is not None:
    st.write('File uploaded:', uploaded_file.name)
```

How to Show Progress in Streamlit

Streamlit offers several status elements that can be used to provide feedback to users, show progress, or add fun effects to your app. These elements enhance the user experience by making the app more interactive and engaging.

The st.spinner command displays a spinner to indicate some processing. It is useful for providing visual feedback when running long computations.

```
import time

# Spinner example
with st.spinner('Loading...'):
    time.sleep(2)
st.success('Done!')
```

The st.progress command creates a progress bar that shows a task's progress. You can update the progress bar as the task progresses.

```
# Progress bar example
progress_bar = st.progress(0)
for percent_complete in range(100):
    time.sleep(0.1)
    progress_bar.progress(percent_complete + 1)
st.write('Task completed!')
```

The st.balloons command releases animated balloons on the screen. Celebrating an achievement or milestone within your app can be fun.

```
# Balloons example
if st.button('Celebrate!'):
    st.balloons()
```

Similarly, the st.snow command creates a snow effect on the screen. It can add a festive or seasonal touch to your app.

```
# Snow example
if st.button('Let it snow!'):
    st.snow()
```

How to Use Forms in Streamlit

Streamlit provides several commands for managing the forms within your app. These commands allow you to control the execution of your code, create forms, and handle form submissions efficiently.

The st.form command creates a form that can contain multiple widgets. Forms are useful when you want users to complete multiple inputs and submit them all at once.

```
# Form example
with st.form(key='my_form'):
    name = st.text_input('Name')
    age = st.number_input('Age', min_value=0)
    submitted = st.form_submit_button('Submit')

if submitted:
    st.write('Name:', name)
    st.write('Age:', age)
```

The st.form_submit_button command adds a submit button to a form. It is used within a form created by st.form to handle form submissions.

How to Insert Chat Messages in Streamlit

Streamlit also allows inserting chat message containers to simulate chat-like interactions within your app. The st.chat_message command allows you to display messages styled as if they were part of a chat conversation.

```
# Display user message
with st.chat_message("user"):
    st.write("Hello 👋")

# Simulate bot response
with st.chat_message("bot"):
    st.write("Hi there! How can I help you today?")
```