Command Prompt:

```
streamlit run app.py
```

Note:
Streamlit apps have a unique data flow: any time something must be updated on the screen, Streamlit reruns your entire Python script from top to bottom. This can happen in two situations:

1. Whenever you modify your app's source code.
2. Whenever a user interacts with widgets in the app. For example, when dragging a slider, entering text in an input box, or clicking a button.

Widgets are the backbone of interactivity in Streamlit. Instead of static dashboards, they allow you to build dynamic apps where users can experiment with inputs and instantly see updated outputs.

👉 In short: a widget in Streamlit is any interactive control (slider, button, text box, etc.) that triggers your app to rerun with updated values.

Session State and Widget State association

Every widget with a key is automatically added to Session State:

1) Use Callbacks to update Session State
2) A callback is a python function which gets called when an input widget changes.

Order of execution: When updating Session state in response to events, a callback function gets executed first, and then the app is executed from top to bottom. Callbacks can be used with widgets using the **parameters** on_change (or on_click), args, and kwargs:

Parameters:

1. on_change or on_click - The function name to be used as a callback
2. args (tuple) - List of arguments to be passed to the callback function
3. kwargs (dict) - Named arguments to be passed to the callback function

And to make all of this fast and seamless, Streamlit does some heavy lifting for you behind the scenes. A big player in this story is the `@st.cache_data` decorator, which allows developers to skip certain costly computations when their apps rerun.

Widgets which support the on_change event:

- st.checkbox
- st.color_picker
- 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 which support the on_click event:

- st.button
- st.download_button
- st.form_submit_button

To add a callback, define a callback function above the widget declaration and pass it to the widget via the on_change (or on_click) parameter.

Forms and Callbacks

Widgets inside a form can have their values be accessed and set via the Session State API. st.form_submit_button can have a callback associated with it. The callback gets executed upon clicking on the submit button. For example:

```
def form_callback():
    st.write(st.session_state.my_slider)
    st.write(st.session_state.my_checkbox)

with st.form(key='my_form'):
    slider_input = st.slider('My slider', 0, 10, 5, key='my_slider')
    checkbox_input = st.checkbox('Yes or No', key='my_checkbox')
    submit_button = st.form_submit_button(label='Submit', on_click=form_callback)
```

Serializable Session State

Serialization refers to the process of converting an object or data structure into a format that can be persisted and shared, and allowing you to recover the data's original structure. Python's built-in pickle module serializes Python objects to a byte stream **("pickling")** and deserializes the stream into an object **("unpickling")**.

Display and style data

There are a few ways to display data (tables, arrays, data frames) in Streamlit apps. Below, you will be introduced to magic and st.write(), which can be used to write anything from text to tables. After that, let's take a look at methods designed specifically for visualizing data.

Use magic

You can also write to your app without calling any Streamlit methods. Streamlit supports "magic commands," which means you don't have to use st.write() at all! To see this in action try this snippet:

```
import streamlit as st
import pandas as pd
df = pd.DataFrame({
  'first column': [1, 2, 3, 4],
  'second column': [10, 20, 30, 40]
})
df
```

Write a data frame

Along with magic commands, st.write() is Streamlit's "Swiss Army knife". You can pass almost anything to st.write(): text, data, Matplotlib figures, Altair charts, and more. Don't worry, Streamlit will figure it out and render things the right way.

```
import streamlit as st
import pandas as pd

st.write("Here's our first attempt at using data to create a table:")
st.write(pd.DataFrame({
    'first column': [1, 2, 3, 4],
    'second column': [10, 20, 30, 40]
}))
```

There are other data specific functions like st.dataframe() and st.table() that you can also use for displaying data. Let's understand when to use these features and how to add colors and styling to your data frames.

You might be asking yourself, "why wouldn't I always use st.write()?" There are a few reasons:

1. Magic and st.write() inspect the type of data that you've passed in, and then decide how to best render it in the app. Sometimes you want to draw it another way. For example, instead of drawing a dataframe as an interactive table, you may want to draw it as a static table by using st.table(df).
2. The second reason is that other methods return an object that can be used and modified, either by adding data to it or replacing it.
3. Finally, if you use a more specific Streamlit method you can pass additional arguments to customize its behavior.

For example, let's create a data frame and change its formatting with a Pandas `Styler` object. In this example, you'll use Numpy to generate a random sample, and the `st.dataframe()` method to draw an interactive table.

Let's expand on the first example using the Pandas Styler object to highlight some elements in the interactive table.

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

dataframe = pd.DataFrame(
   np.random.randn(10, 20),
   columns=('col %d' % i for i in range(20)))
st.table(dataframe)
```

Draw charts and maps

Streamlit supports several popular data charting libraries like Matplotlib, Altair, deck.gl, and more. In this section, you'll add a bar chart, line chart, and a map to your app.

Draw a line chart

You can easily add a line chart to your app with st.line_chart(). We'll generate a random sample using Numpy and then chart it.

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

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

st.line_chart(chart_data)
```

Plot a map

With st.map() you can display data points on a map. Let's use Numpy to generate some sample data and plot it on a map of San Francisco.

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

map_data = pd.DataFrame(
    np.random.randn(1000, 2) / [50, 50] + [37.76, -122.4],
    columns=['lat', 'lon'])

st.map(map_data)
```

Widgets

When you've got the data or model into the state that you want to explore, you can add in widgets like st.slider(), st.button() or st.selectbox(). It's really straightforward — treat widgets as variables:

```
import streamlit as st
x = st.slider('x')  # 👈 this is a widget
st.write(x, 'squared is', x * x)
```

On the first run, the app above should output the text "0 squared is 0". Then every time a user interacts with a widget, Streamlit simply reruns your script from top to bottom, assigning the current state of the widget to your variable in the process.

For example, if the user moves the slider to position 10, Streamlit will rerun the code above and set x to 10 accordingly. So now you should see the text "10 squared is 100".

Widgets can also be accessed by key, if you choose to specify a string to use as the unique key for the widget:

```
import streamlit as st
st.text_input("Your name", key="name")

# You can access the value at any point with:
St.session_state.name
```

Every widget with a key is automatically added to Session State. For more information about Session State, its association with widget state, and its limitations, see Session State API Reference Guide.

Use checkboxes to show/hide data

One use case for checkboxes is to hide or show a specific chart or section in an app. st.checkbox() takes a single argument, which is the widget label. In this sample, the checkbox is used to toggle a conditional statement.

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

if st.checkbox('Show dataframe'):
   chart_data = pd.DataFrame(
      np.random.randn(20, 3),
      columns=['a', 'b', 'c'])
```

```
    Chart_data
```

Use a selectbox for options

Use st.selectbox to choose from a series. You can write in the options you want, or pass through an array or data frame column. Let's use the df data frame we created earlier.

```
import streamlit as st
import pandas as pd

df = pd.DataFrame({
    'first column': [1, 2, 3, 4],
    'second column': [10, 20, 30, 40]
    })

option = st.selectbox(
    'Which number do you like best?',
     df['first column'])

'You selected: ', option
```

Layout

Streamlit makes it easy to organize your widgets in a left panel sidebar with st.sidebar. Each element that's passed to st.sidebar is pinned to the left, allowing users to focus on the content in your app while still having access to UI controls.

For example, if you want to add a selectbox and a slider to a sidebar, use st.sidebar.slider and st.sidebar.selectbox instead of st.slider and st.selectbox:

```
import streamlit as st

# Add a selectbox to the sidebar:
add_selectbox = st.sidebar.selectbox(
    'How would you like to be contacted?',
    ('Email', 'Home phone', 'Mobile phone')
)

# Add a slider to the sidebar:
add_slider = st.sidebar.slider(
    'Select a range of values',
    0.0, 100.0, (25.0, 75.0)
)
```

Beyond the sidebar, Streamlit offers several other ways to control the layout of your app. st.columns lets you place widgets side-by-side, and st.expander lets you conserve space by hiding away large content.

```
import streamlit as st

left_column, right_column = st.columns(2)
# You can use a column just like st.sidebar:
left_column.button('Press me!')

# Or even better, call Streamlit functions inside a "with" block:
with right_column:
    chosen = st.radio(
        'Sorting hat',
        ("Gryffindor", "Ravenclaw", "Hufflepuff", "Slytherin"))
    st.write(f"You are in {chosen} house!")
```

Push_pin

Note
st.echo and st.spinner are not currently supported inside the sidebar or layout options. Rest assured, though, we're currently working on adding support for those too!

Show progress
When adding long running computations to an app, you can use st.progress() to display status in real time.

First, let's import time. We're going to use the time.sleep() method to simulate a long running computation:

```
import time
```

Now, let's create a progress bar:

```
import streamlit as st
import time

'Starting a long computation...'

# Add a placeholder
latest_iteration = st.empty()
bar = st.progress(0)

for i in range(100):
  # Update the progress bar with each iteration.
  latest_iteration.text(f'Iteration {i+1}')
  bar.progress(i + 1)
  time.sleep(0.1)

'...and now we\'re done!'
```