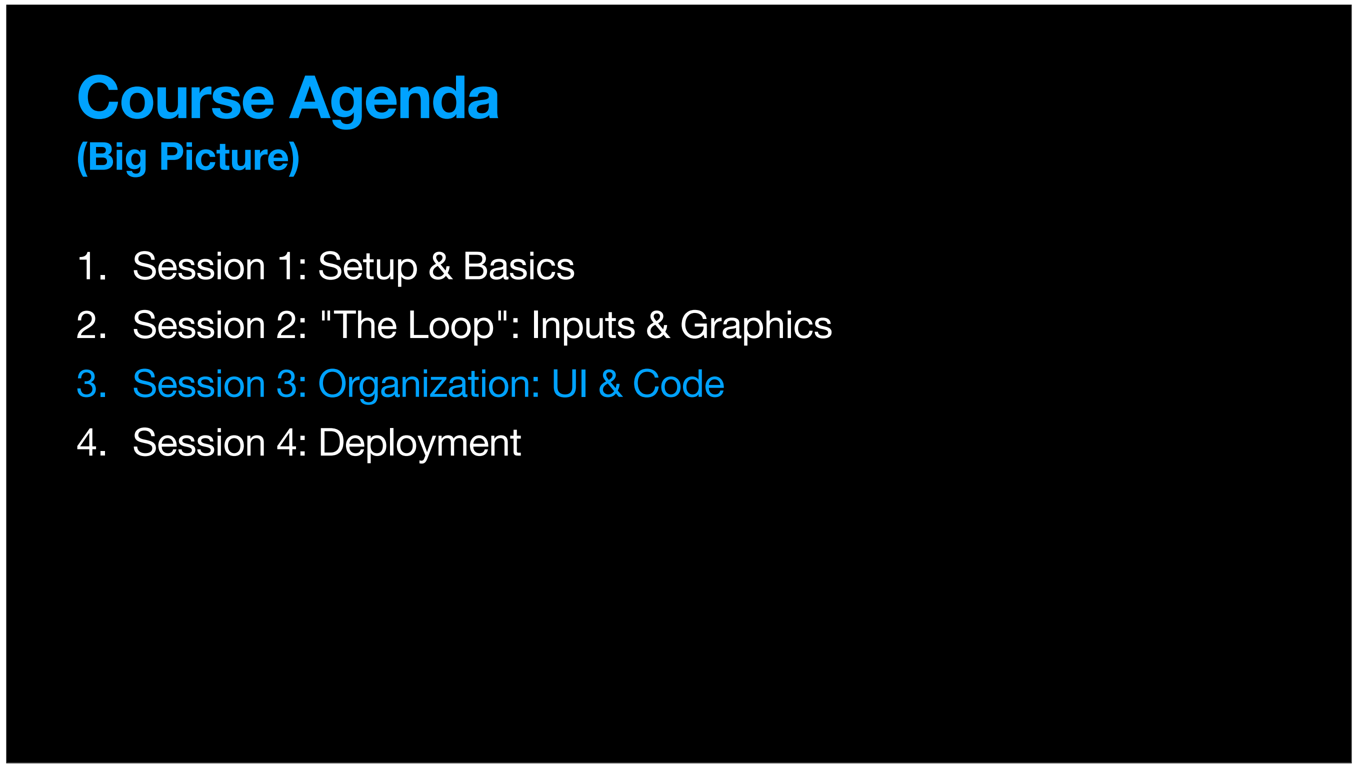
Building Data Apps In Python With Streamlit

Session 3: Organizing UI and Code

Ari Lamstein
AriLamstein.com



Course Agenda

(Big Picture)

1. Session 1: Setup & Basics
2. Session 2: "The Loop": Inputs & Graphics
3. Session 3: Organization: UI & Code
4. Session 4: Deployment

In the last session, the app became a bit cluttered. Both the user interface and the code. In this section I'll give you some tips to organize both.

Organizing UI and Code

- UI
- Code

Exercise: Design?

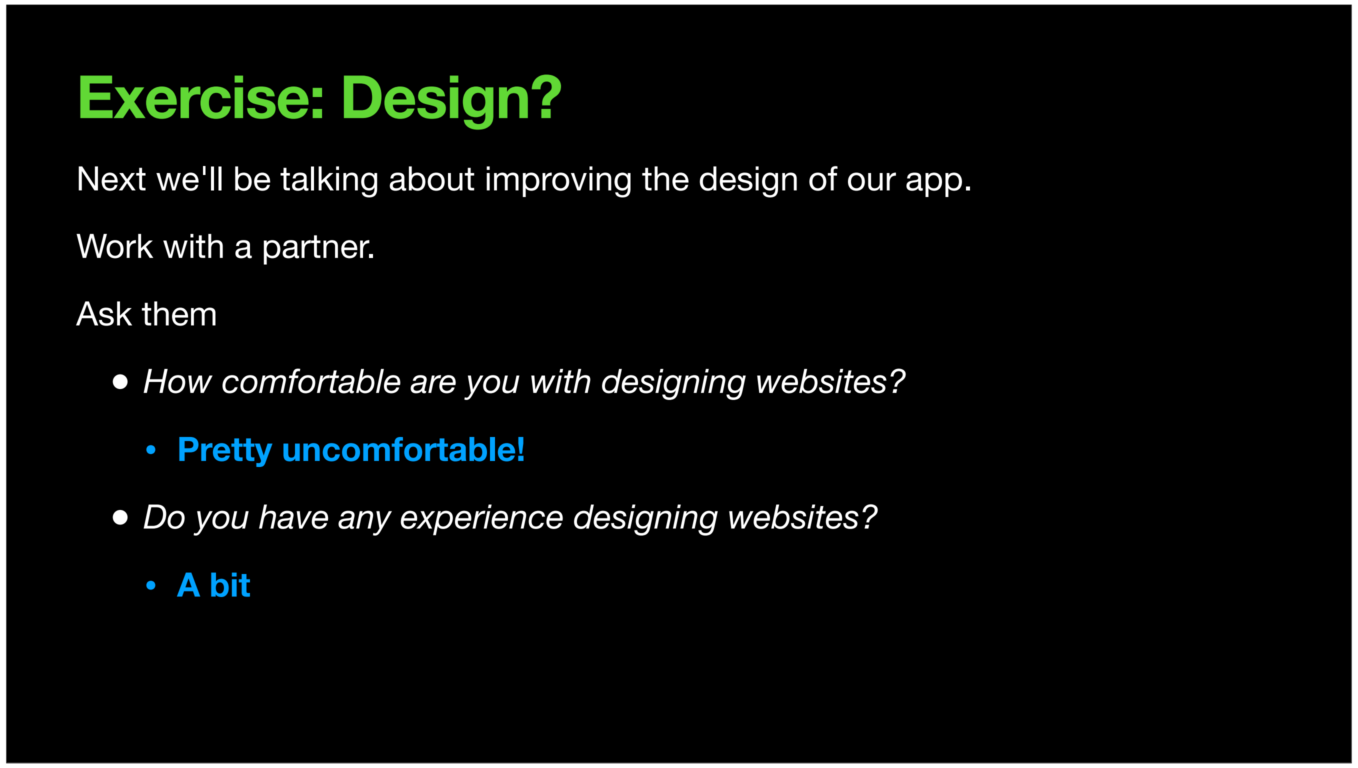
Next we'll be talking about improving the design of our app.

Work with a partner.

Ask them

- *How comfortable are you with designing websites?*
- *Do you have any experience designing websites?*

4 minute exercise


Exercise: Design?

Next we'll be talking about improving the design of our app.

Work with a partner.

Ask them

- *How comfortable are you with designing websites?*
  - **Pretty uncomfortable!**
- *Do you have any experience designing websites?*
  - **A bit**

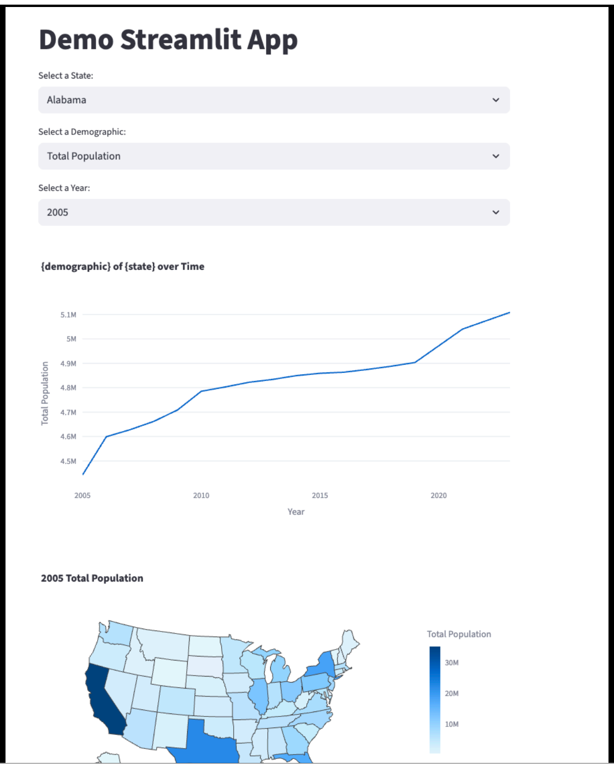
I tend to avoid design work. Because I'm colorblind, I'm always "wrong" about colors. I hate that feeling, so I just avoid the entire field of design. That being said, I also get frustrated pretty quickly with designs that are confusing. And so I like to make designs that minimize friction.

Please Open

- 3-organize-ui-code/organization.ipynb
- 3-organize-ui-code/layout_app.py

Design Critique

- Too much information all at once.
- User has to scroll to see all the information
    - People often don't scroll
- Can all the information fit on one screen?



Demo Streamlit App

Select a State: Alabama

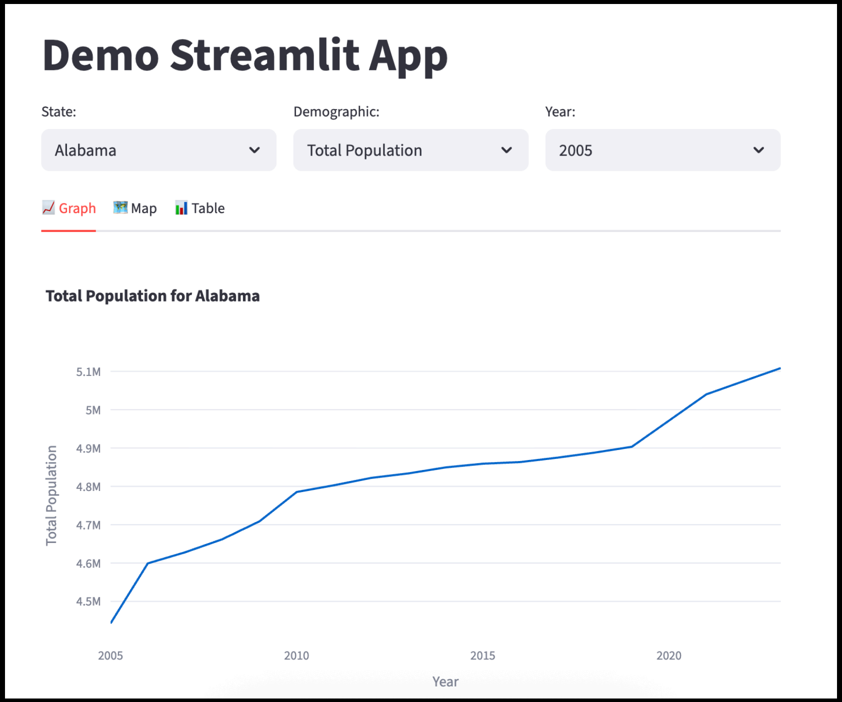
Select a Demographic: Total Population

Select a Year: 2005

This is the "before" slide
Everything here is vertical

One solution

- Columns for widgets
- Tabs for graphs



Demo Streamlit App

State: Alabama

Demographic: Total Population

Year: 2005

This is the "after" slide. Walk thru the usage of both columns and tabs

Organizing UI

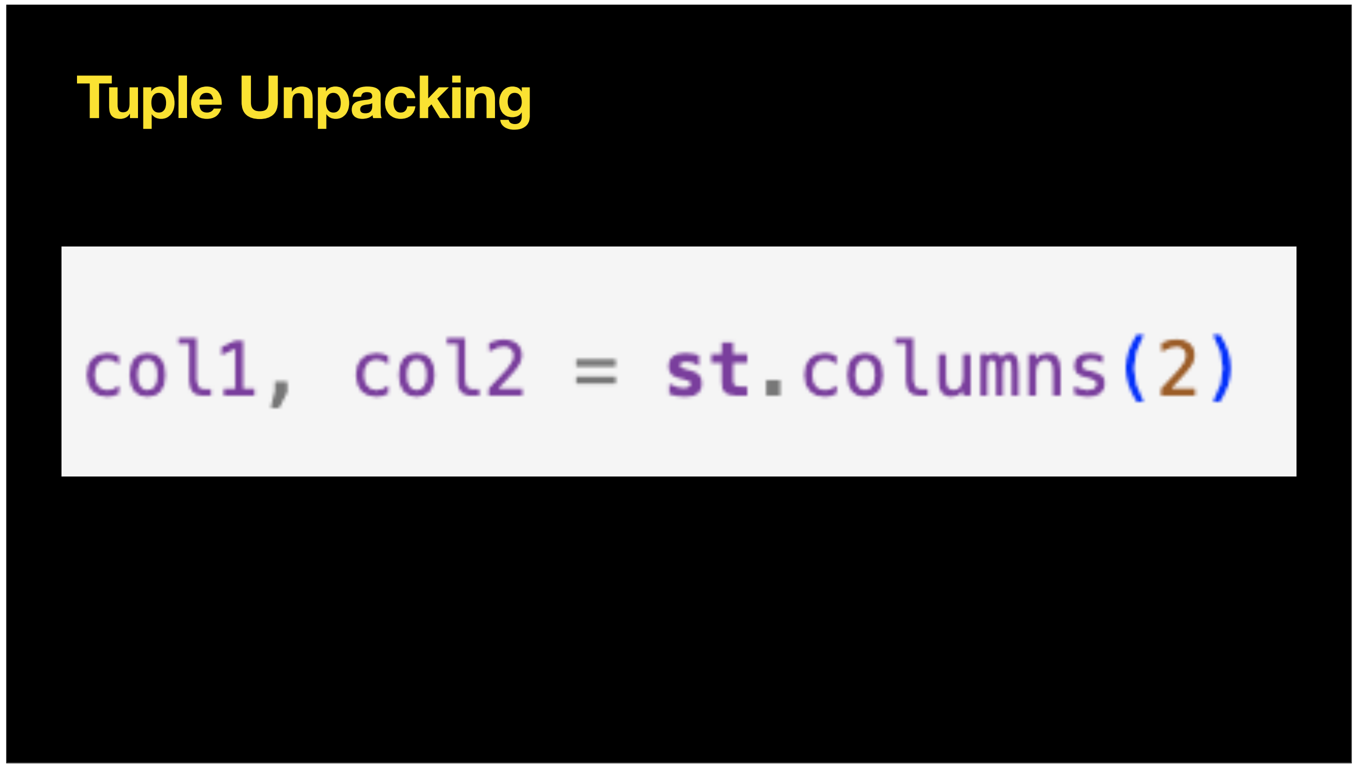
- Columns for Widgets
- Tabs for graphs

st.columns()

You create columns by calling the function st.columns.

st.columns()

- Input: The number of columns you want
- Output: A list of column objects
- Two idioms:
  - Tuple Unpacking
  - Context Manager



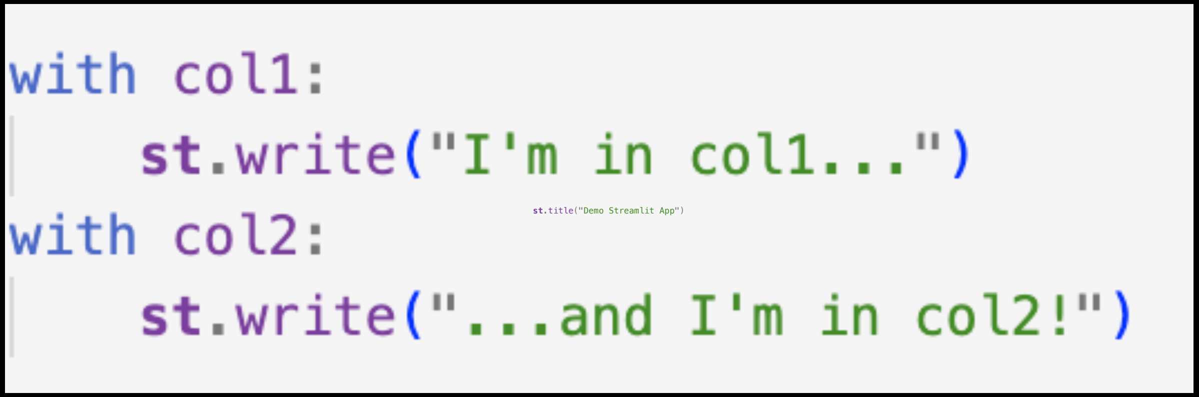
Tuple Unpacking

```
col1, col2 = st.columns(2)
```

As I said on the last slide, st.columns returns a **list** of column objects. We haven't spoken about what those column objects actually are yet. But here I give st.columns the number 2 and it returns a list of 2 column objects.

By typing "col1, col2" we can assign each of those column objects to different variables, all in 1 line of code. This is a common pattern when creating columns. The reason we call this "tuple unpacking" is historical - there isn't actually a tuple here.

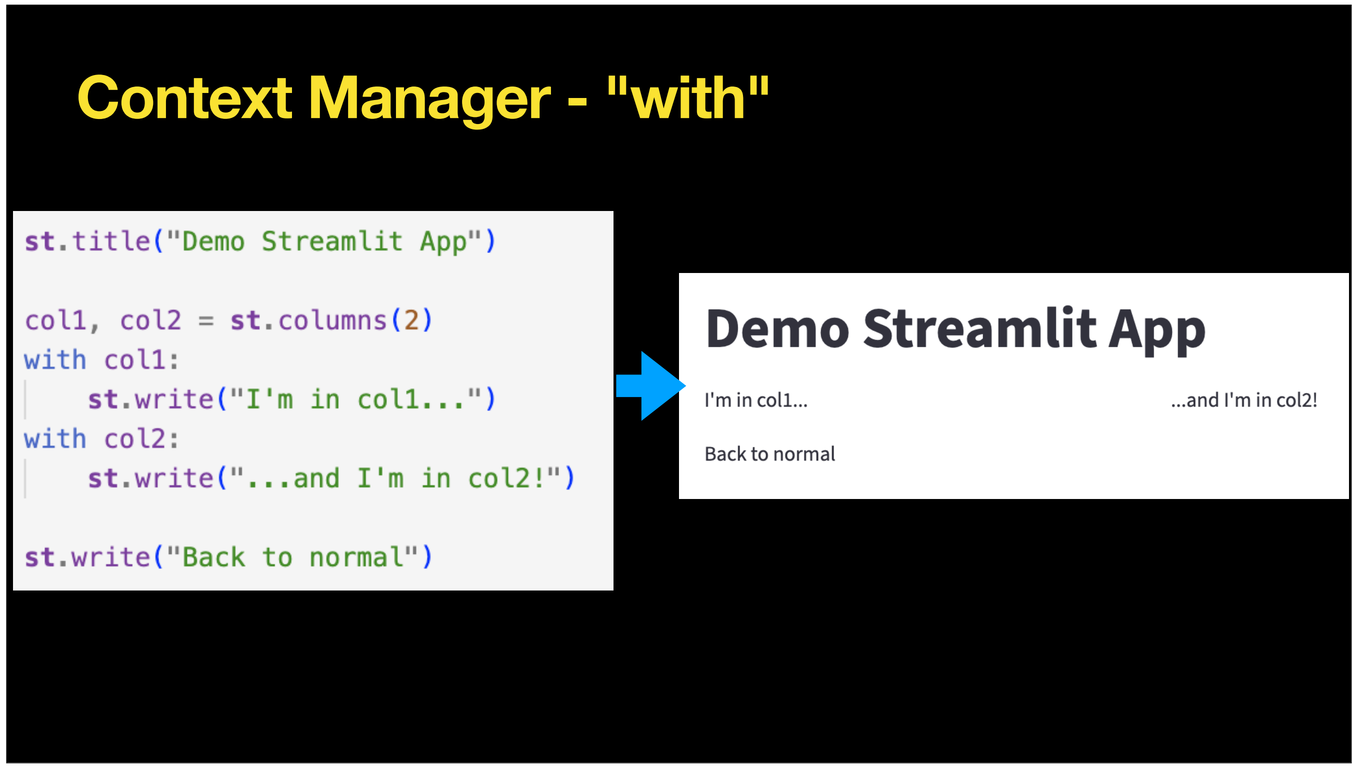
Context Manager - "with"



```python
with col1:
    st.write("I'm in col1...")
with col2:
    st.write("...and I'm in col2!")
```

```python
st.title("Demo Streamlit App")
```

When it comes to actually using the columns, you use a context manager. This means typing "with col1:", and then putting everything you want to be in the column below, indented. The code you write inside the context manager will appear in the column.



Context Manager - "with"

```python
st.title("Demo Streamlit App")

col1, col2 = st.columns(2)
with col1:
    st.write("I'm in col1...")
with col2:
    st.write("...and I'm in col2!")

st.write("Back to normal")
```

Demo Streamlit App

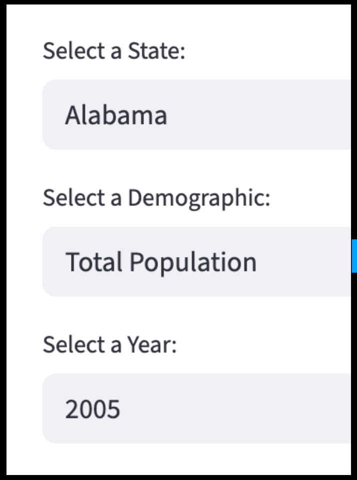
I'm in col1... ...and I'm in col2!

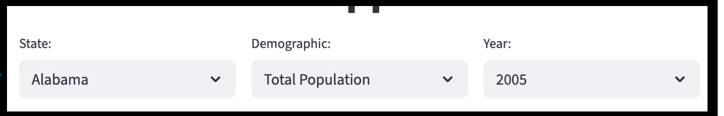
Back to normal

Code written above and below the context manager will use Streamlit's normal default of 1 column.

Exercise: Columns

- Update layout_app.py





Update layout_app.py so that it puts each of the 3 UI Widgets into a column. The final app should have 3 columns, and each column should have 1 UI Widget

Organizing UI

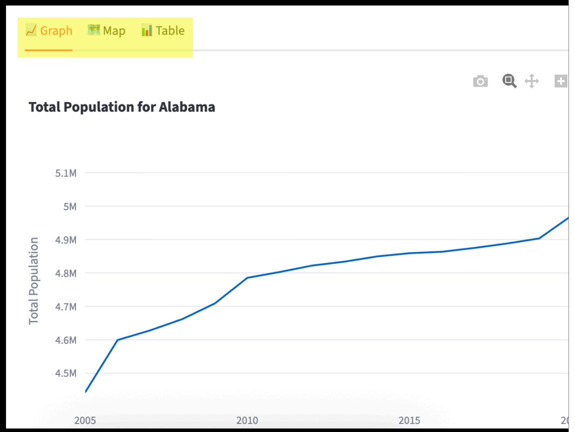
- Columns for Widgets
- Tabs for graphs

st.tabs()

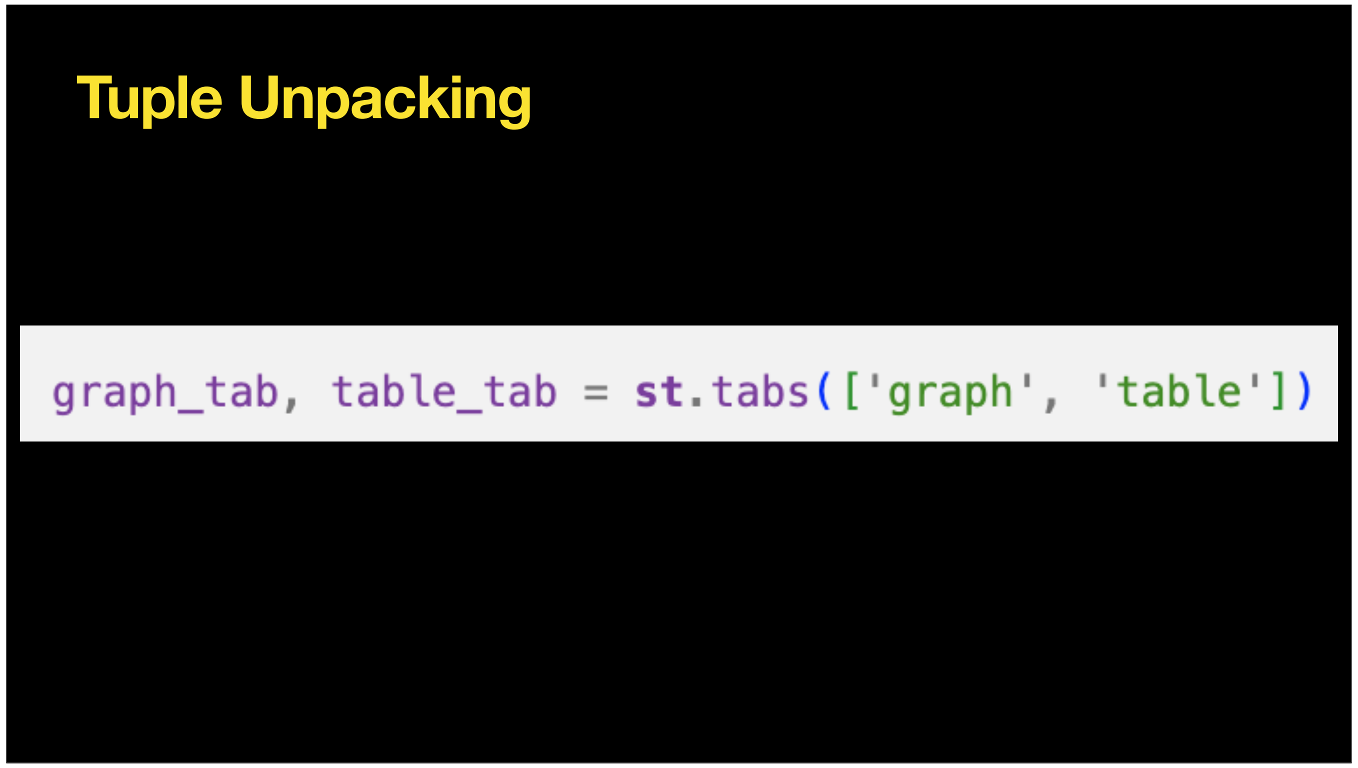
You create tabs by calling the function st.tabs.

st.tabs()

- Input: A list that contains the names of the tabs
  - ['Graph', 'Map', 'Table']
- Output: A list of tab objects
- Two idioms:
  - Tuple Unpacking
  - Context Manager



The main difference between tabs and columns is that tabs have **names**. So we supply a list of names to st.tabs, instead of an integer saying how many tabs we want.

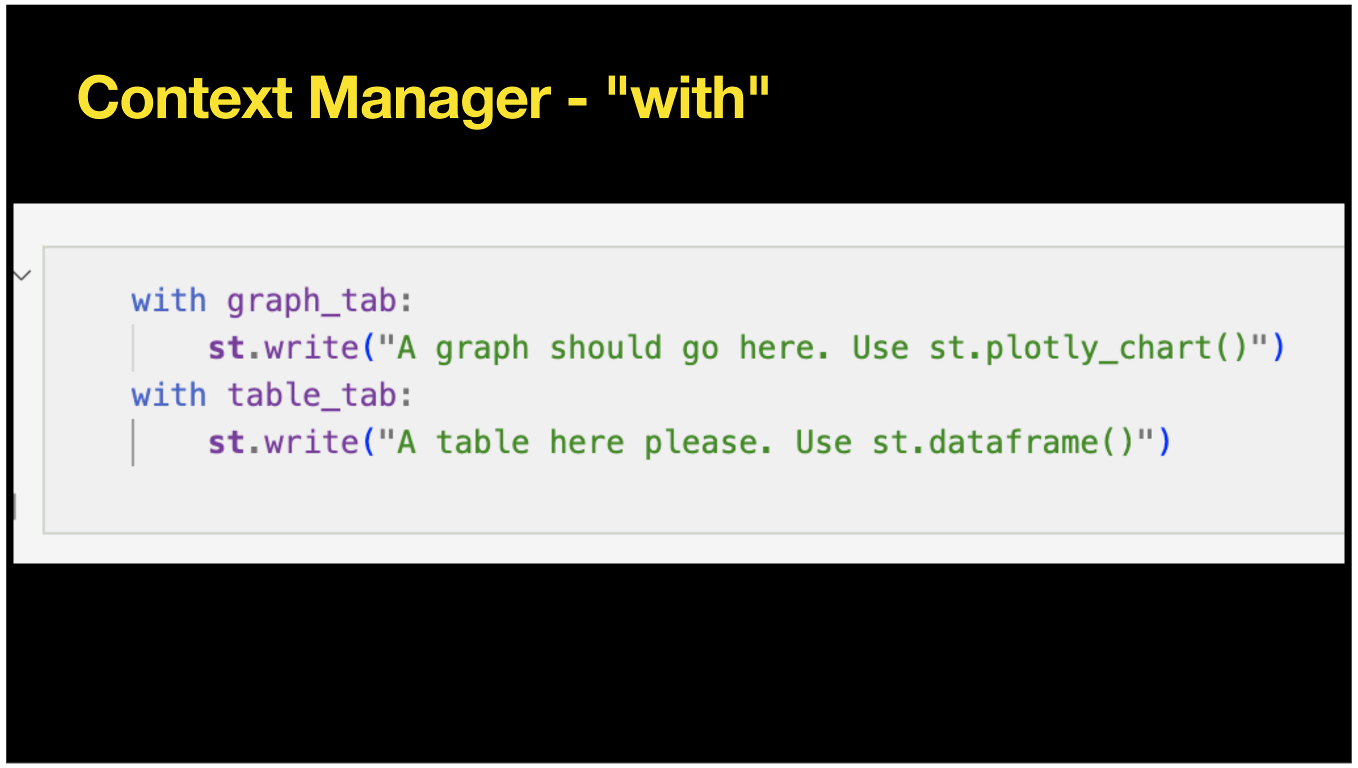


Tuple Unpacking

```python
graph_tab, table_tab = st.tabs(['graph', 'table'])
```

Like with columns, it is common to use tuple unpacking to immediately assign the tab objects to variables. The difference is that, since tabs have names, we normally use descriptive names for the variables as well.

So here I'm calling the first variable "graph_tab" instead of "tab1".

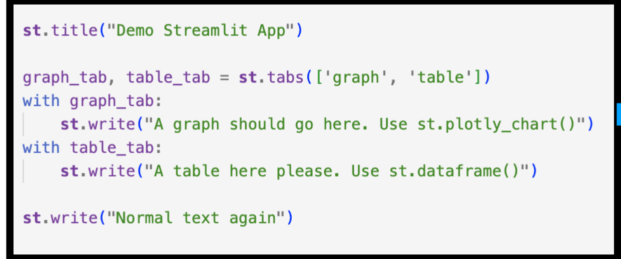


Context Manager - "with"

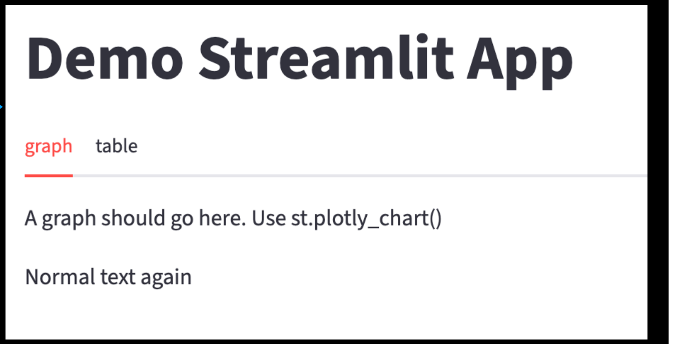
```
with graph_tab:
    st.write("A graph should go here. Use st.plotly_chart()")
with table_tab:
    st.write("A table here please. Use st.dataframe()")
```

Like with columns, you use a context manager to put things in the tab. Use the keyword "with" and then everything indented below it will be in the tab.

Context Manager - "with"



```
st.title("Demo Streamlit App")

graph_tab, table_tab = st.tabs(['graph', 'table'])
with graph_tab:
    st.write("A graph should go here. Use st.plotly_chart()")
with table_tab:
    st.write("A table here please. Use st.dataframe()")

st.write("Normal text again")
```



Demo Streamlit App

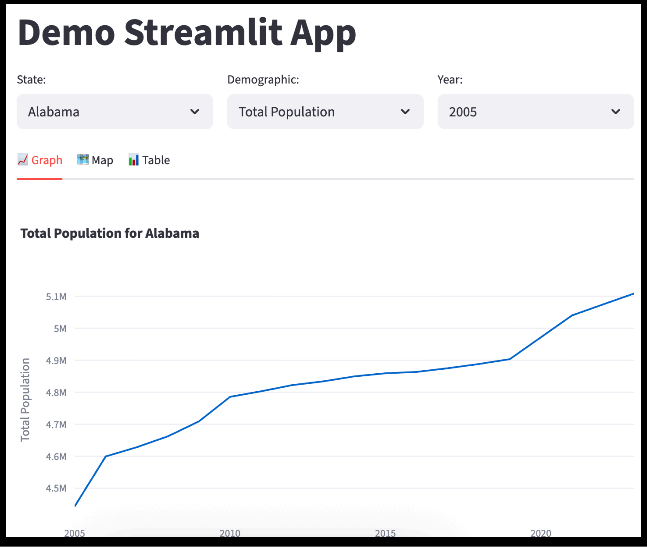
A graph should go here. Use st.plotly_chart()

Normal text again

Code written above and below the context manager will use Streamlit's normal default of 1 column. Note that the graph tab is highlighted

Exercise: tabs

- Add tabs to layout_app.py



Demo Streamlit App

State: Alabama

Demographic: Total Population

Year: 2005

📈 Graph 🗺️ Map 📊 Table

Total Population for Alabama

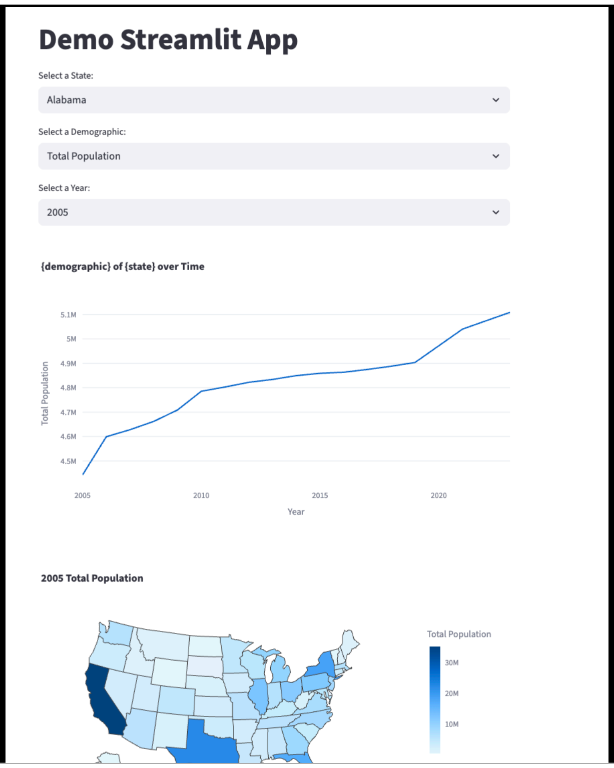
Update layout_app.py so that each of the 3 visualizations (Graph, Tab, Table) is in its own tab.
Bonus: Ask an LLM for help adding emojis to the tab names.

Organizing UI and Code

- UI
- Code

Design Critique (UI)

- Too much information all at once.
- User has to scroll to see all the information
    - People often don't scroll
- Can all the information fit on one screen?



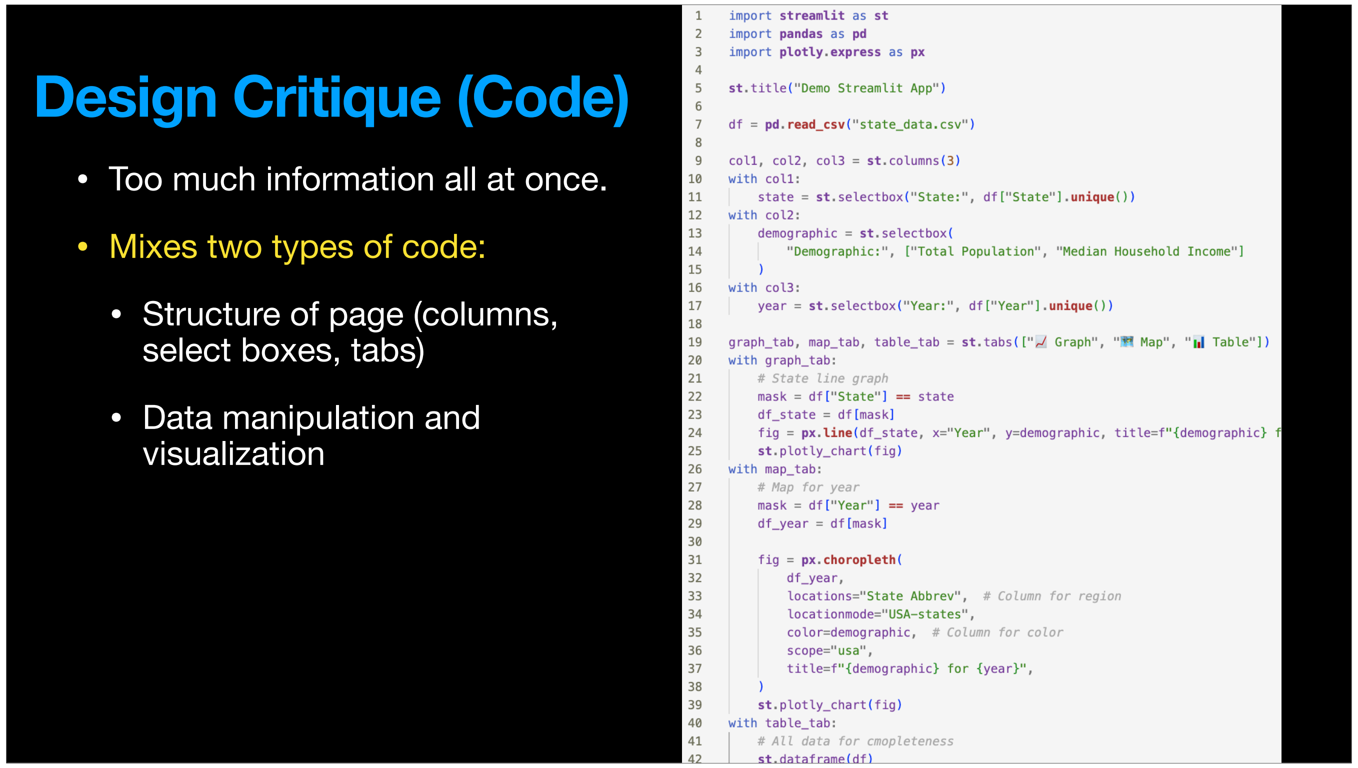
Demo Streamlit App

Select a State: Alabama

Select a Demographic: Total Population

Select a Year: 2005

Before talking about organizing code, let's review the critique of our original UI design: there was too much information all at once, and it made it hard to reason about the app.



Design Critique (Code)

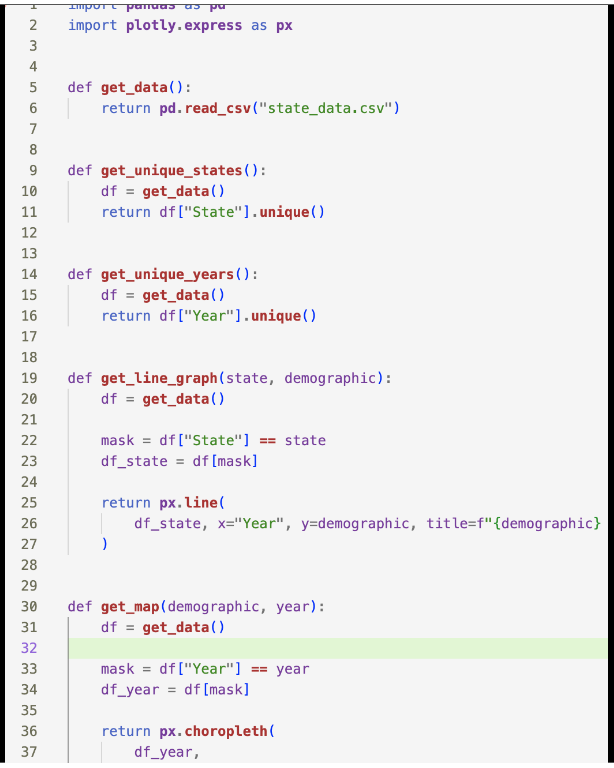
- Too much information all at once.
- Mixes two types of code:
  - Structure of page (columns, select boxes, tabs)
  - Data manipulation and visualization

```python
import streamlit as st
import pandas as pd
import plotly.express as px

st.title("Demo Streamlit App")

df = pd.read_csv("state_data.csv")

col1, col2, col3 = st.columns(3)
with col1:
    state = st.selectbox("State:", df["State"].unique())
with col2:
    demographic = st.selectbox(
        "Demographic:", ["Total Population", "Median Household Income"]
    )
with col3:
    year = st.selectbox("Year:", df["Year"].unique())

graph_tab, map_tab, table_tab = st.tabs(["📈 Graph", "🗺 Map", "📊 Table"])
with graph_tab:
    # State line graph
    mask = df["State"] == state
    df_state = df[mask]
    fig = px.line(df_state, x="Year", y=demographic, title=f"{demographic} f
    st.plotly_chart(fig)
with map_tab:
    # Map for year
    mask = df["Year"] == year
    df_year = df[mask]

    fig = px.choropleth(
        df_year,
        locations="State Abbrev",  # Column for region
        locationmode="USA-states",
        color=demographic,  # Column for color
        scope="usa",
        title=f"{demographic} for {year}",
    )
    st.plotly_chart(fig)
with table_tab:
    # All data for cmopleteness
    st.dataframe(df)
```

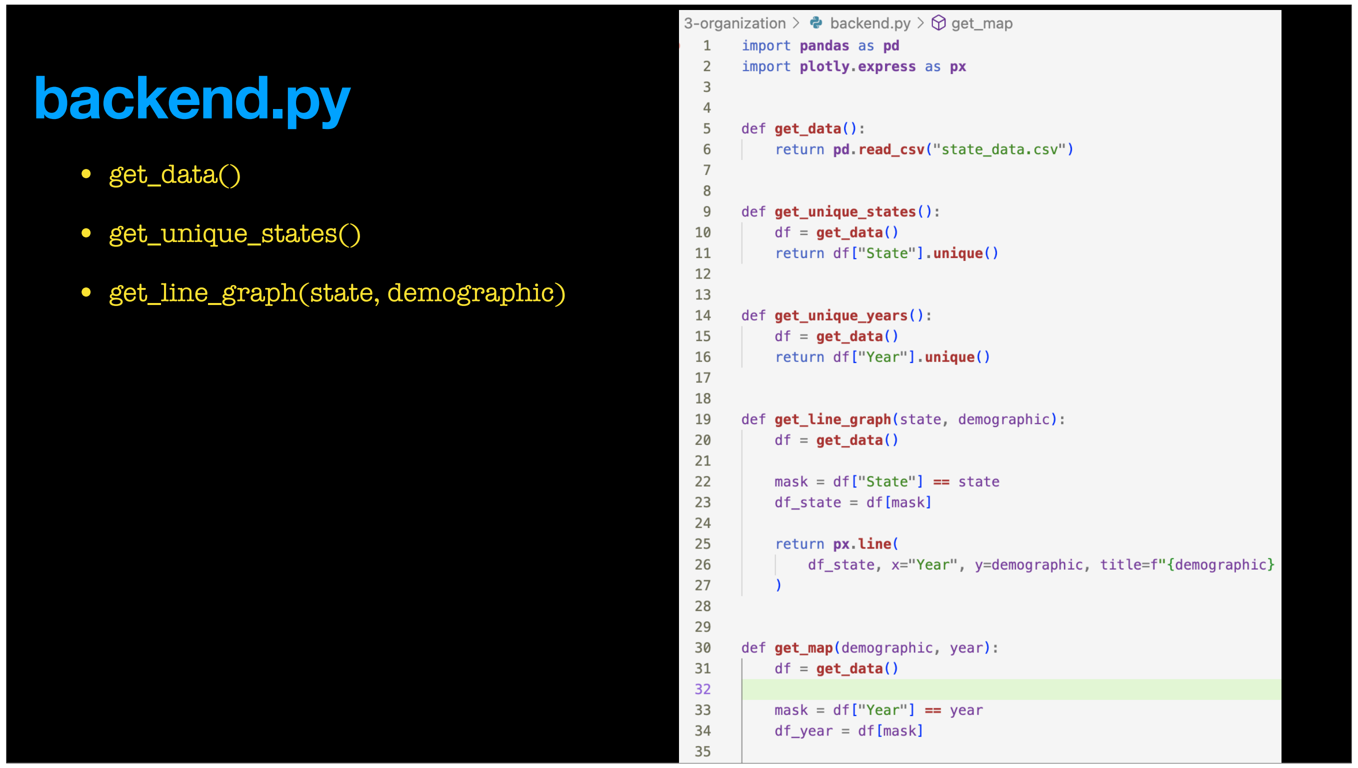
There's a similar problem with the code. Because a single file is doing both user interface tasks and data tasks, it can be hard to reason about.

Solution: 2 files

- Split the app into two files
- layout_app.py:
  - Structure of page (columns, select boxes, tabs)
  - Calls to streamlit library
- backend.py:
  - Loading data
  - Manipulating data
  - Visualizing data
  - Just functions
    - Can be called from notebooks or streamlit app



```python
import pandas as pd
import plotly.express as px


def get_data():
    return pd.read_csv("state_data.csv")


def get_unique_states():
    df = get_data()
    return df["State"].unique()


def get_unique_years():
    df = get_data()
    return df["Year"].unique()


def get_line_graph(state, demographic):
    df = get_data()

    mask = df["State"] == state
    df_state = df[mask]

    return px.line(
        df_state, x="Year", y=demographic, title=f"{demographic}
    )


def get_map(demographic, year):
    df = get_data()

    mask = df["Year"] == year
    df_year = df[mask]

    return px.choropleth(
        df_year,
```



backend.py

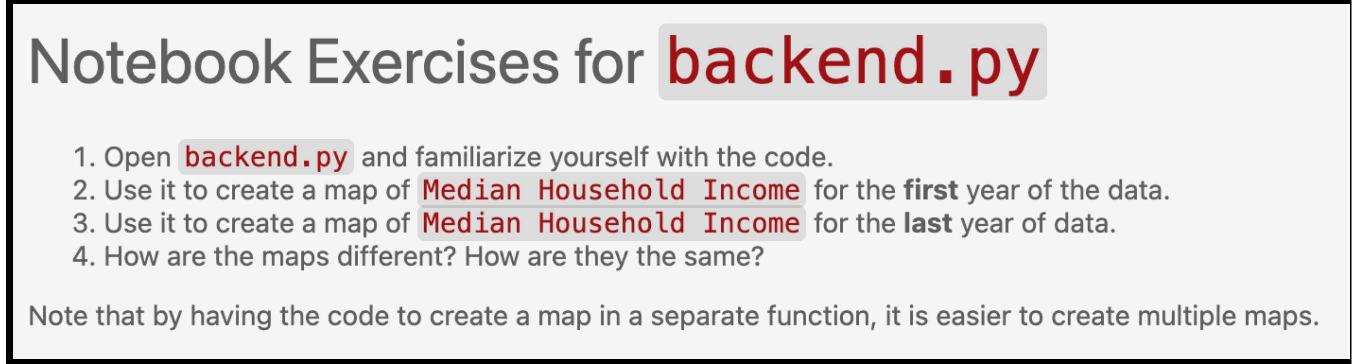
- get_data()
- get_unique_states()
- get_line_graph(state, demographic)

3-organization > backend.py > get_map

```python
import pandas as pd
import plotly.express as px


def get_data():
    return pd.read_csv("state_data.csv")


def get_unique_states():
    df = get_data()
    return df["State"].unique()


def get_unique_years():
    df = get_data()
    return df["Year"].unique()


def get_line_graph(state, demographic):
    df = get_data()

    mask = df["State"] == state
    df_state = df[mask]

    return px.line(
        df_state, x="Year", y=demographic, title=f"{demographic}
    )


def get_map(demographic, year):
    df = get_data()

    mask = df["Year"] == year
    df_year = df[mask]

```

To demonstrate this, I've already created a file backend.py for you. Please take a minute to review it.

Exercise: Using backend.py in a Notebook

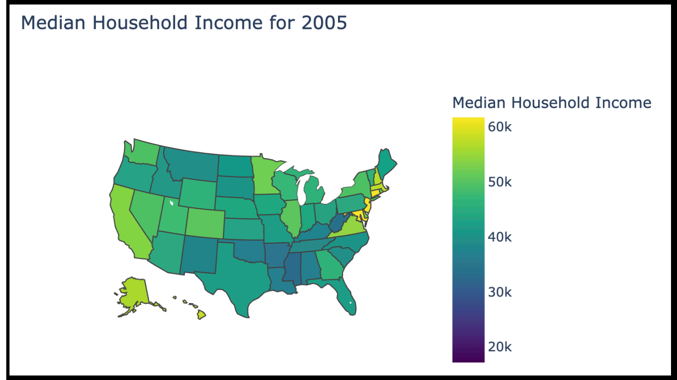


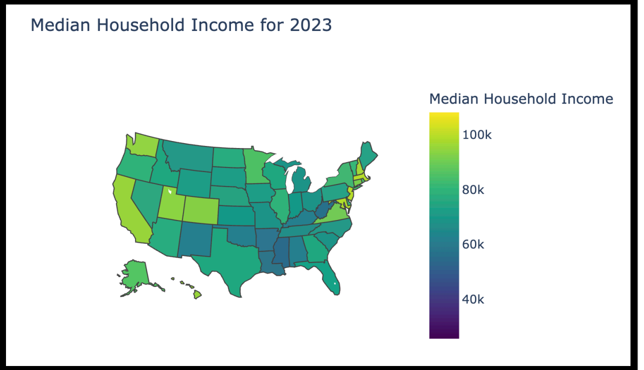
Notebook Exercises for `backend.py`

1. Open `backend.py` and familiarize yourself with the code.
2. Use it to create a map of `Median Household Income` for the **first** year of the data.
3. Use it to create a map of `Median Household Income` for the **last** year of data.
4. How are the maps different? How are they the same?

Note that by having the code to create a map in a separate function, it is easier to create multiple maps.

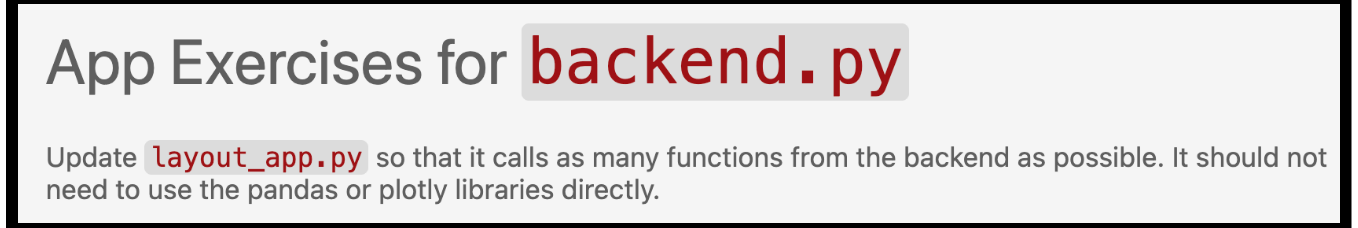
My solution





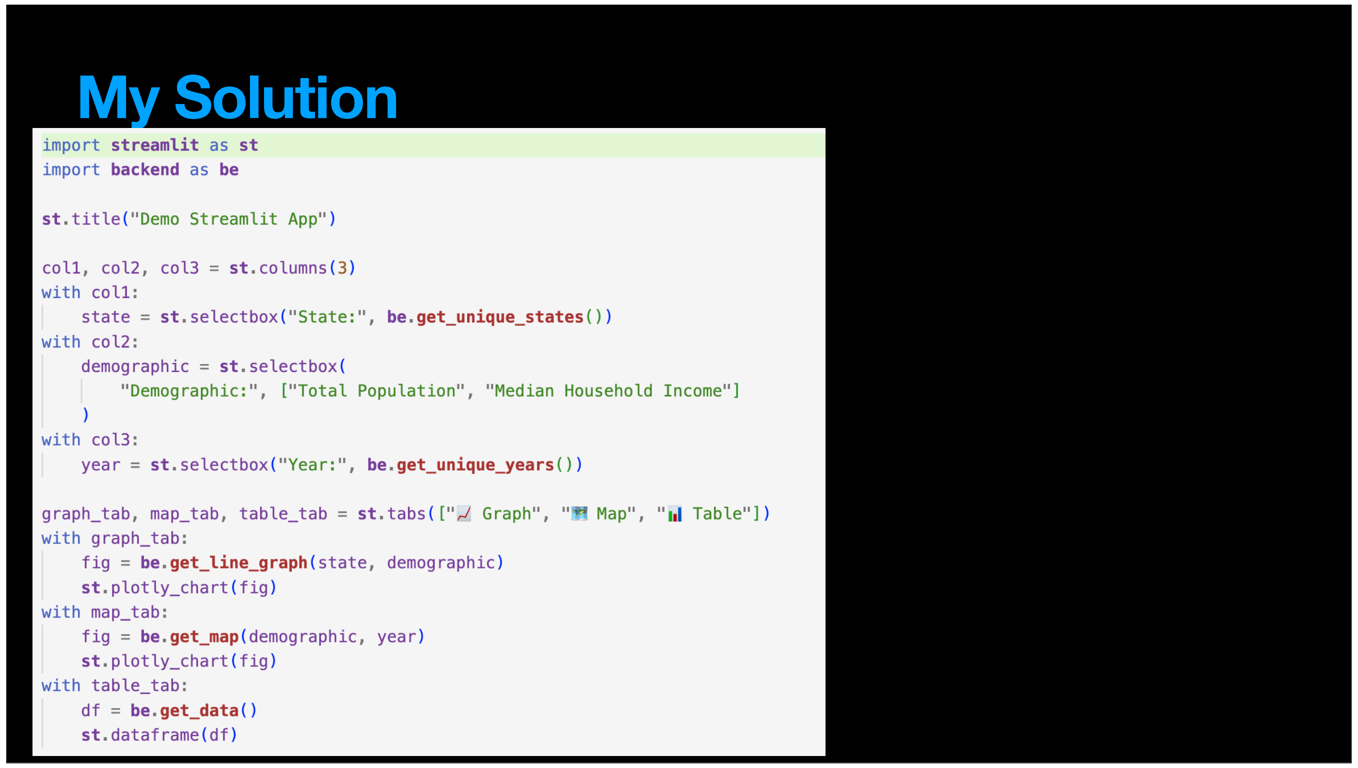
Pattern is the same. But the scale is very different.

Exercise: Using backend.py in an App



App Exercises for `backend.py`

Update `layout_app.py` so that it calls as many functions from the backend as possible. It should not need to use the pandas or plotly libraries directly.



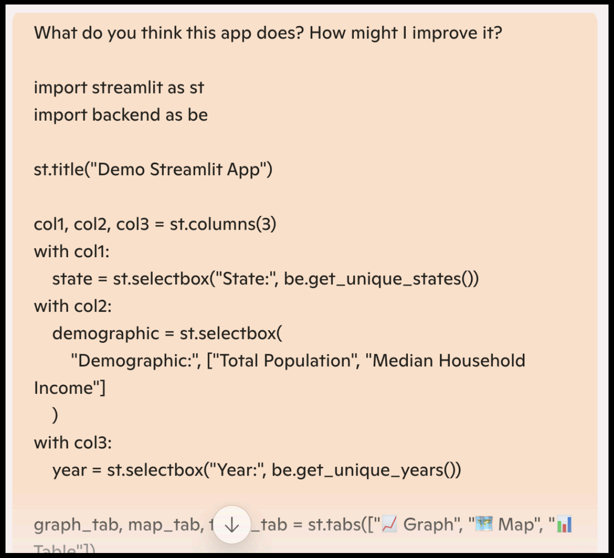
My Solution

```python
import streamlit as st
import backend as be

st.title("Demo Streamlit App")

col1, col2, col3 = st.columns(3)
with col1:
    state = st.selectbox("State:", be.get_unique_states())
with col2:
    demographic = st.selectbox(
        "Demographic:", ["Total Population", "Median Household Income"]
    )
with col3:
    year = st.selectbox("Year:", be.get_unique_years())

graph_tab, map_tab, table_tab = st.tabs(["📈 Graph", "🗺️ Map", "📊 Table"])
with graph_tab:
    fig = be.get_line_graph(state, demographic)
    st.plotly_chart(fig)
with map_tab:
    fig = be.get_map(demographic, year)
    st.plotly_chart(fig)
with table_tab:
    df = be.get_data()
    st.dataframe(df)
```

Code is focused on structure. Implementation details are elsewhere.

Final Exercise: LLM

- Copy your final app into an LLM
- Give it the prompt like "*What do you think this app does? How might I improve it?*"
- Which suggestions do you think are worth implementing? Can you ask it to generate the code to implement them?



What do you think this app does? How might I improve it?

```
import streamlit as st
import backend as be

st.title("Demo Streamlit App")

col1, col2, col3 = st.columns(3)
with col1:
    state = st.selectbox("State:", be.get_unique_states())
with col2:
    demographic = st.selectbox(
        "Demographic:", ["Total Population", "Median Household
Income"]
    )
with col3:
    year = st.selectbox("Year:", be.get_unique_years())

graph_tab, map_tab, t  _tab = st.tabs(["📈 Graph", "🗺️ Map", "📊
Table"])
```