Dashboards

Peter Ganong and Maggie Shi

February 17, 2026

Introduction

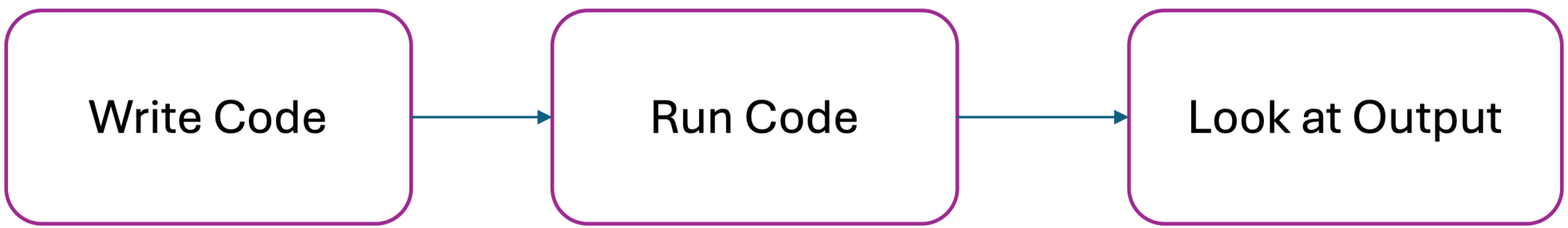
Skills acquired by end of lecture

- Understand benefits (and drawbacks) of dashboards
- Launch `streamlit` dashboards locally and deploy online
- Make basic static and interactive dashboards
- Set up dashboards that run efficiently

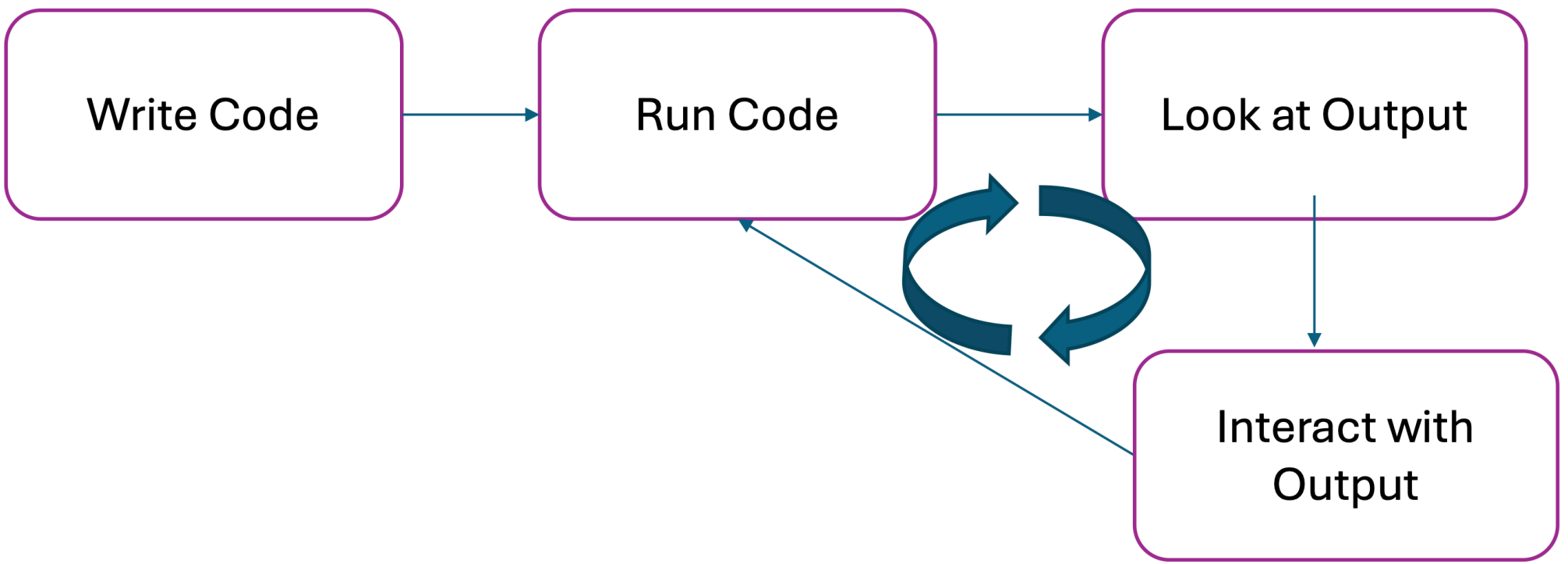
Materials for this lecture

- We will review code for several example dashboards
- `all_snippets.pdf`: all code reviewed in lecture
- Full source code for example dashboards stored in a separate repo than student repo: https://github.com/uchicago-harris-dap/dashboards.
  - Clone to access source code and launch apps locally
  - When lecture material says `1/hello_world_1.py`, `2/hello_world_2_local.py`, we are referencing this repo
  - Example dashboards are also deployed online and linked

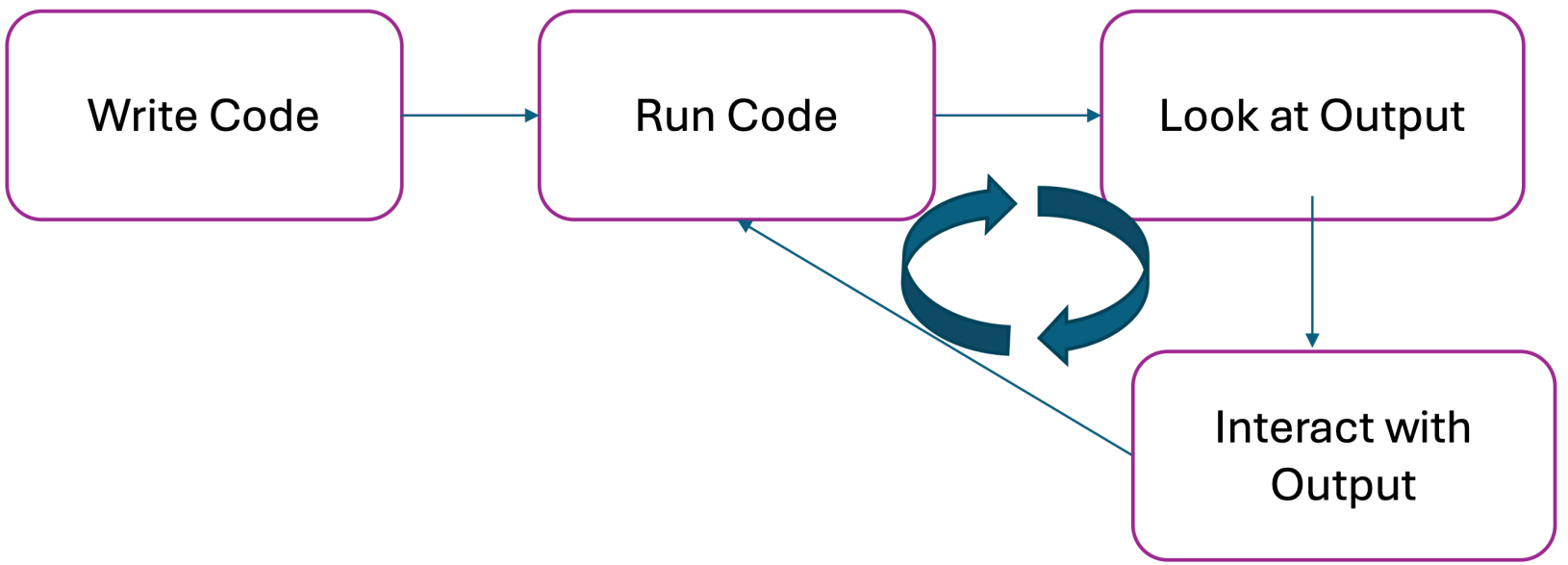
From A Static Process



To A Dynamic Process

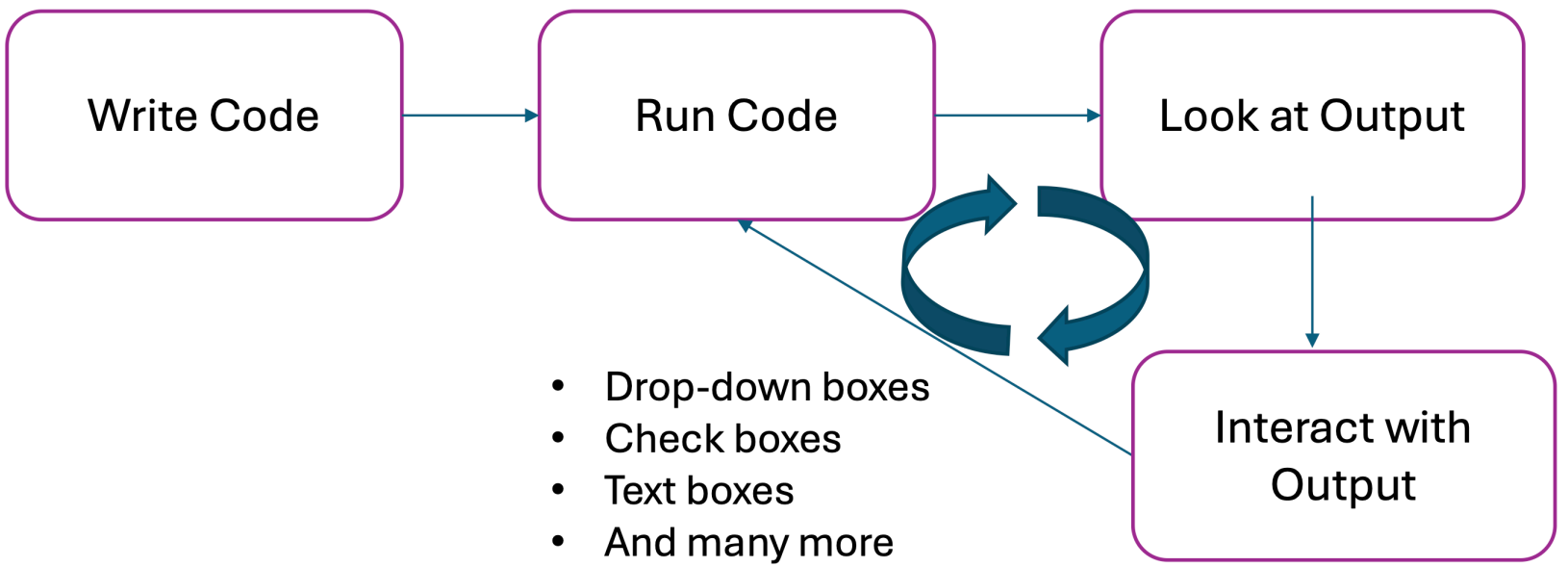


To A Dynamic Process



When you analyze a dataset, your process looks a lot more like the dynamic one than the static one.

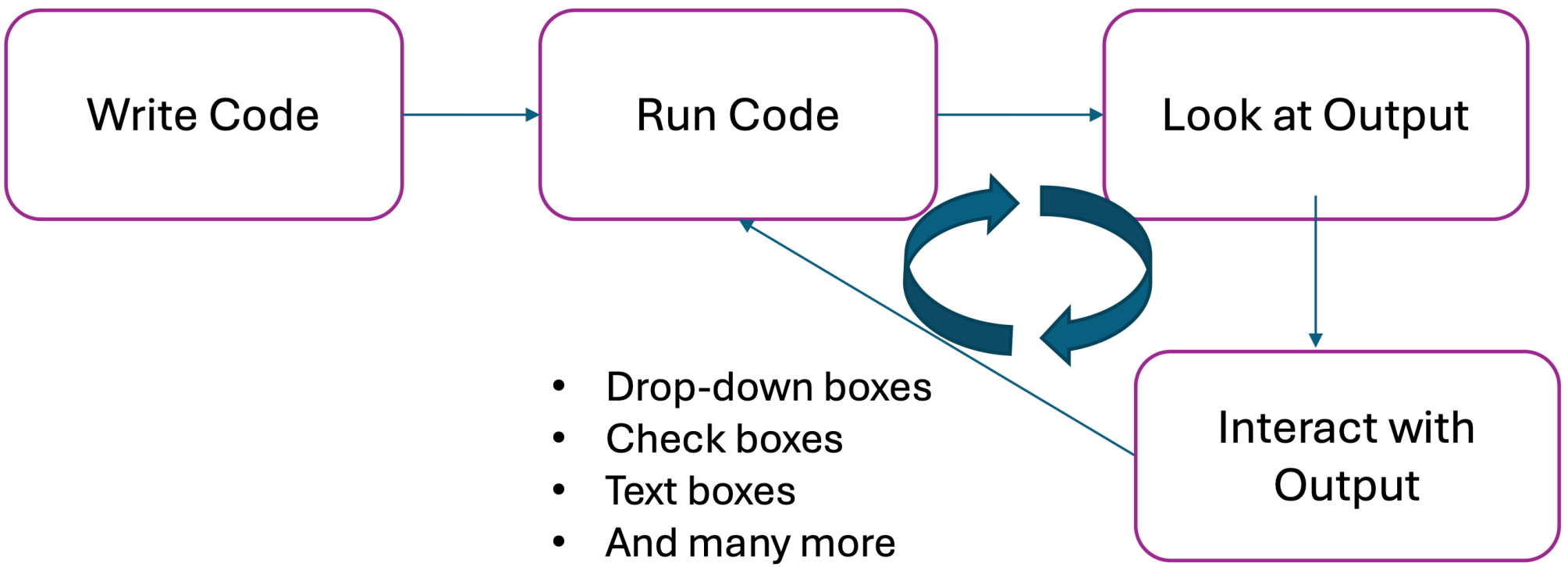
To A Dynamic Process



Dashboards allow someone who does not code to do a limited version of the same process you go through as you code.

To A Dynamic Process

- Locally
- Personal web server



- Drop-down boxes
- Check boxes
- Text boxes
- And many more

Other benefits of dashboards

- There are two other major benefits of dashboards which we will not spend time on in class, but you should also know about
  - Consolidated information: aggregate data from various sources into one unified interface
  - Real-time data metrics: dynamic view based on automatically-updated data feed
- *(Third benefit is that they are impressive: they are a great way to flex your data skills for your portfolio!)*

Where we are going

http://uchicago-harris-dap-dashboards-4.streamlit.app/

Dashboards: Promises and Pitfalls

When Professor Ganong worked in the Mayor's Office in Boston, the mayor asked for every single department to suggest metrics for a dashboard. The mayor put a TV in his office so that the dashboards would be displayed at all times.

Discussion questions:

1. What are examples of data that a city would benefit from tracking with a dashboard?
2. What are examples of data where putting it on a dashboard might inadvertently lead to poor management or create bad incentives for workers?

`streamlit`

We are teaching dashboards using `streamlit`. This is designed for data scientists who want to share analyses without needing front-end (e.g., web design) development skills. Beyond being easy to use, we chose to teach `streamlit` for a few reasons:

1. Native to Python, works well with existing Python tools (e.g. Altair)
2. All its features are free (so long as you are making public dashboards)
3. Open source: https://github.com/streamlit/streamlit

Caveat: `streamlit` is for prototyping (not computationally efficient for many users). We will talk more at the end of the lecture about how you will use streamlit and other tools going forward.

Citing our sources

- These lecture slides draw on examples and text from https://docs.streamlit.io.
- `streamlit` has fantastic documentation and app gallery

Introduction: summary

- Dashboards let non-coders interact with data
- Introduce `streamlit`
- Note on process: this is our first year teaching this lecture note so we are highly uncertain how long it will take (both each slide and the lecture overall)

Dashboards as websites

Prep

- Create a separate repo just for your dashboards. Ours is https://github.com/uchicago-harris-dap/dashboards
- Be sure it is public.
- Save code; we will call this `1/hello_world_1.py`

```
import streamlit as st
st.write("""
# My first app
Hello *world!*
""")
```

- Push code

My first dashboard

In this lecture: `$` means run at the terminal (on Macs the prefix is `%`). You do not need to type `$`.

```
1 $ streamlit run 1/hello_world_1.py
```

- Recall: `Ctrl + C` to exit in Terminal

My first dashboard – two ways to display

1. **Local** Create a server on your computer. Access from your web browser (and no one else's). Like we did on prior slide.
2. **Online** Create a server in the cloud. Access from any web browser.

My first dashboard – putting it online

- *(one time only)* connect your github account to streamlit community cloud. (here)
- Three clicks: Deploy -> Deploy Now (under Streamlit Community Cloud) -> Deploy (with pre-filled repo, branch, and file names).
- Now your app is online and can be shared via link: http://uchicago-harris-dap-dashboards-1.streamlit.app/
- Workflow: Streamlit does **not** actually deploy the script which is stored on your computer. Instead, it reads the script which is pushed to your public repo and deploys that.

My second dashboard

- `$ streamlit run 2/hello_world_2_local.py`
- Uncomment `st.altair_chart(chart_to_show)`. Save. Show detection of file change.
  - Fun fact: streamlit will automatically figure out how to print various data types to the web so you also can instead just write `chart_to_show`.
- This is the normal workflow for building a streamlit dashboard. Update code. Save. Re-run locally. Once you are happy, then you deploy.
- http://uchicago-harris-dap-dashboards-2.streamlit.app/

Interactive dashboards

Roadmap

Dashboard 3. Multipage.

Do-pair-share.

Dashboard 4. Uber pickups. “Classic” streamlit dashboard.

Dashboard 3 roadmap

1. Homepage with dropdown menu
2. Plotting demo with button
3. Dataframe demo with selection box and table
4. Mapping demo with checkboxes

Preview http://uchicago-harris-dap-dashboards-3b.streamlit.app/

Going forward, we won't deploy locally anymore. But all the source code is located in the dashboards repo. For this one, you would deploy locally using `streamlit run 3/multipage.py`

Dashboard 3 Homepage Review code

- Review structure of page and `intro()`
- *Lines 226-234*: created via `page_names_to_funcs` and `st.sidebar.selectbox(...)`

`3/multipage.py`:

```python
import streamlit as st

st.set_page_config(page_title="Multipage Demo")


def intro():
    import streamlit as st

    st.write("# Welcome to Streamlit! 👋")
    st.sidebar.success("Select a demo above.")

    st.markdown(
        """
        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!
```

```
19
20          ### Want to learn more?
21
22          - Check out [streamlit.io](https://streamlit.io)
```

Feature: taking input

Streamlit has a huge number of different ways to take input. In this lecture you will see:

- `st.button()`
- `st.checkbox()`
- `st.slider()`
- `st.selectbox()`

Feature: layouts and containers

Streamlit also has many layouts and containers. In this lecture you will see:

- `st.sidebar()`
- `st.columns()`

Feature: showing progress

Streamlit also has ways to show progress and failure page. In this lecture you will see:

- `st.progress()`
- `st.error()`

`plotting_demo()`

- *Lines 6-13*: use `st.` commands to set up page
- *Lines 15-18*: set up blank `progress_bar`, `status_text`, `chart` in sidebar
- *Lines 20-28*: (usual python) loop to generate random numbers

```
 1  def plotting_demo():
 2      import streamlit as st
 3      import time
 4      import numpy as np
 5
 6      st.markdown("# Plotting Demo")
 7      st.write(
 8          """
 9          This demo illustrates a combination of plotting and animation with
10  Streamlit. We're generating a bunch of random numbers in a loop for around
11  5 seconds. Enjoy!
12  """
13      )
14
15      progress_bar = st.sidebar.progress(0)
```

```
    status_text = st.sidebar.empty()
    last_rows = np.random.randn(1, 1)
    chart = st.line_chart(last_rows)

    for i in range(1, 101):
```

DataFrame demo data preview

▶ Code

```
1 get_UN_data()
```

	Region	1961	1962	1963	
0	China	58340740.0	60690860.0	63942700.0	68
1	United States of America	89816580.0	90275010.0	93700380.0	94
2	India	50142680.0	50015760.0	51367730.0	52
3	USSR	72768810.0	73961920.0	70531000.0	81

	Region	1961	1962	1963	
4	Brazil	16564600.0	17344940.0	17443010.0	176
...	...	...	...	...	...
215	Tuvalu	473.0	474.0	476.0	513
216	Nauru	184.0	184.0	185.0	191
217	Tokelau	170.0	220.0	112.0	281
218	Cayman Islands	0.0	0.0	0.0	0.0
219	Saint Pierre and Miquelon	0.0	0.0	0.0	0.0

DataFrame demo review code

- *Lines 25-27*: choose inputs `countries = st.multiselect()`
- *Line 36*: print table using `st.write()`
- *Line 41-43*: `pd.melt` in prepartion for `chart`

`data_frame_demo()` from `3/multipage.py`:

```
1  def data_frame_demo():
2      import streamlit as st
3      import pandas as pd
4      import altair as alt
5
6      from urllib.error import URLError
7
8      st.markdown("# DataFrame Demo")
9      st.write(
10         """
11         This demo shows how to use `st.write` to visualize Pandas DataFrames.
12
13 (Data courtesy of the [UN Data Explorer](http://data.un.org/Explorer.aspx).)
14 """
15     )
16
17     @st.cache_data
18     def get_UN_data():
```

```
        AWS_BUCKET_URL = "http://streamlit-demo-data.s3-us-west-2.amazonaws.com"
        df = pd.read_csv(AWS_BUCKET_URL + "/agri.csv.gz")
        return df.set_index("Region")

    try:
```

Feature: showing output

for text

`st.markdown()`

for anything

`st.write()`

- In this example, `st.write()` is used to make a table (analogous to `print()` or `head()`)
- Useful tip: `st.write()` is Streamlit's "Swiss Army knife". Pass almost anything: latex, Matplotlib figures, Altair charts, multiple arguments, and more. Don't worry, Streamlit will figure it out and render things the right way. Full list here

Mapping demo data preview

- Runs on `pydeck` ("High-scale spatial rendering").
- Makes gorgeous maps (will have more depth in minilesson).
- In Terminal/command line: update conda environment or `$ pip install pydeck`

data preview
bike_rental_stats
bart_stop_stats
bart_path_stats

mapping demo review code

Review `HexagonLayer`, `ScatterplotLayer`, `TextLayer`, `ArcLayer`

`mapping_demo()` from `3/multipage.py`:

```
def mapping_demo():
    import streamlit as st
    import pandas as pd
    import pydeck as pdk

    from urllib.error import URLError

    st.markdown("# Mapping Demo")
    st.write(
        """
        This demo shows how to use
[`st.pydeck_chart`](https://docs.streamlit.io/develop/api-reference/charts/
to display geospatial data.
"""
    )

```

```
17      @st.cache_data
18      def from data file(filename):
```

(Optional) Summarize how each dataset is used

Data preview	Layer name	notes
bike_rental_stats	`HexagonLayer`	bar height set by number of rentals at each lat lon
bart_stop_stats	`ScatterplotLayer`	radius = `exits`
bart_stop_stats	`TextLayer`	`get_text="name"`
bart_path_stats	`ArcLayer`	arcs via `get_source_position = ["lon", "lat"], get_target_position = ["lon2", "lat2"]`

Control flow: what code exactly does streamlit run?

Streamlit tries to run your code like a normal python script.

If you write a function and never call it, it is not run. To understand what's run here, the key code block is this

```
page_names_to_funcs = {
    "—": intro,
    "Plotting Demo": plotting_demo,
    "Mapping Demo": mapping_demo,
    "DataFrame Demo": data_frame_demo
}

demo_name = st.sidebar.selectbox("Choose a demo", page_names_to_funcs.keys(
page_names_to_funcs[demo_name]()
```

do-pair-share

1. Render dashboard code in student repo (`student30538-w26/lectures/dashboard/app_dps_1.py`) at the command line/Terminal: `$ streamlit run app_dps_1.py`
2. Edit `.py` so that user can select multiple items instead of just one: display Bus routes, L routes, or both in sidebar
3. Re-render

Hint: browse through documentation for input options

Uber pickups: a full-fledged dashboard

http://uchicago-harris-dap-dashboards-4.streamlit.app/

- This is the “classic” Streamlit dashboard (the one they use in all the demos)
- Why? Pretty, adapts nicely to user input, and it showcases integration with `pydeck` and `altair`.

Uber pickups: preview data

▶ Code

```
1 load_data()
```

	date/time	lat	lon
0	2014-09-01 00:01:00	40.2201	-74.0021
1	2014-09-01 00:01:00	40.7500	-74.0027
2	2014-09-01 00:03:00	40.7559	-73.9864
3	2014-09-01 00:06:00	40.7450	-73.9889
4	2014-09-01 00:11:00	40.8145	-73.9444
5	2014-09-01 00:12:00	40.6735	-73.9918

Uber pickups review code

- *Lines 115-19*: user chooses time of day (`pickup_hour`)
- *Lines 69-70*: helper function `filterdata(data, hour_selected)`
- *Lines 41, 140-157*: `map(data, lat, lon, zoom)` makes a PyDeck `HexagonLayer`. Repeat 4x.
- *Lines 170-182*: `histdata(df, hr)` makes an altair histogram with n rides

`4/uber_pickups.py`:

```
1  """An example of showing geographic data."""
2
3  import os
4
5  import altair as alt
6  import numpy as np
7  import pandas as pd
8  import pydeck as pdk
9  import streamlit as st
10
11 # SETTING PAGE CONFIG TO WIDE MODE AND ADDING A TITLE AND FAVICON
12 st.set_page_config(layout="wide", page_title="NYC Ridesharing Demo", page_icon=":taxi:")
13
14
15 # LOAD DATA ONCE
16 @st.cache_data
17 def load_data():
18     path = "uber-raw-data-sep14.csv.gz"
```

```
    if not os.path.isfile(path):
        path = f"https://github.com/streamlit/demo-uber-nyc-pickups/raw/main/{path}"

    data = pd.read_csv(
        path,
        nrows=100000,  # approx. 10% of data
        names=[
            "date/time",
```

Summary

Saw four dashboards

- Multipage
  - plot
  - data frame
  - map
- Uber pickups – map + plot

Demystifying dashboards

Roadmap

- Why dashboards are different than `.py` (or notebooks or `qmd`)
- Bare vs streamlit mode
- toy example of what streamlit mode can do
- What is a session?

Why dashboards are different than `.py`

- The first section of this lecture treated dashboards as websites without user input. This is like a normal Python script. Run once and gives you output.
- The second section made dashboards interactive. To do this, it must be the case that the entire dashboard, or some part of it, may be run more than once.

Bare mode vs streamlit mode

- "Bare mode" (`$ python app.py`) runs your script like standard Python; output is just whatever the script prints.
- "Streamlit mode" (`$ streamlit run app.py`) does four things:
  - sets up a server to run your script
  - records the **state** (e.g. the choice in `st.selectbox`)
  - reruns code when it detects state changes (e.g. new choice in in `st.selectbox`)
  - serves the interactive user interface (UI) found in the browser
- There's a lot of extra code under the hood. streamlit's goal is for you to focus on the Python code (and ignore all the running a server, taking user input, designing a nice webpage)

Session states and cacheing: motivation

- One problem with streamlit mode: every time state changes, it will re-run your script from the top
- Say your dashboard has 2 selection box options:
  - User selects A: calculate A and plot it
  - User selects B: calculate B and plot it
- Every time the user switches between the selection to plot A and B, `streamlit` will re-run the calculations for A and B

Saving across re-runs

- To make this more efficient, we can have `streamlit` do something once and ‘save’ it for re-runs
- Streamlit offers two alternate ways to do this:
  - Session states
  - Cacheing
- Session state is conceptually easier to understand – will explain this first to build intuition
- But cacheing is more commonly used but less transparent – will explain this in next section
- *Note: there are also subtle difference in uses between session states and cacheing that we won’t cover in depth in this class.*

Session state

- A **session** is a single instance of viewing an app
- Session State provides a dictionary-like interface where you can preserve info between script reruns within a session.
- Use `st.session_state` with dictionary notation to store and recall values. For example, `st.session_state.my_key`.
  - If a user refreshes their browser page or reloads the URL to the app, their Session State resets and they begin again with a new session.
  - Multiple users = multiple sessions

Session state demo overview

http://uchicago-harris-dap-dashboards-5.streamlit.app/

- `$ streamlit run 5/session_state_example.py`
- User chooses `route = st.selectbox("CTA Route", ...)` and `cut_pct = st.slider("What fraction of service to cut?", ...)`
- Dashboard then runs a computation which takes 3 seconds (on the pset, you will see an example where it takes much longer than 3 seconds)
- Returns a result
- What if we want to run the exact same computation twice? Better to use the saved result!

Session state demo review code

- Lines 11-12: users selects line to cut and percentage cut
- Lines 14-18: record number of calculations done and key for calculations done
- Lines 21-23: If calculation has not been run before, run it

```
 1 import streamlit as st
 2 import time
 3
 4 st.set_page_config(page_title="Expensive Calculation")
 5
 6 st.title("Expensive Calculation Demo")
 7
 8 # user inputs
 9 route = st.selectbox("CTA route to cut", ["Red Line", "Blue Line", "Green L
10 cut_pct = st.slider("What fraction of service to cut?", min_value = 0.0, ma
11
12 if "counter" not in st.session_state:
```

```
    st.session_state.counter = 0

# session_state key
key = f"n_pass_{route}_{cut_pct}"

# run the heavy job only if we haven't already stored it
```

Summary

- Dashboards expect to be run more than once
- When you run slow code, want to run it once within a session and save the output
- `st.session_state` is one way to do this

Decorators and `@st.cache_data`

Roadmap

- Decorator examples
  - Non-`streamlit`: timer
  - Non-`streamlit`: check crs for spatial joins
  - `@st.cache_data`
- revisit toy example from last section
- real-life example

Decorators

- A second way to save values across sessions is to use a `@st.cache_data` **decorator**
- A decorator is a wrapper that takes another function as input and returns a modified version of that function
- Decorators let you add behavior without changing the original function's code
  - *When would you want to do this?* If you want to apply same cross-cutting behavior across multiple functions without duplicating code

Syntax:

```
@decorator_name
def myfunction():
```

is equivalent to `myfunction = decorator_name(myfunction)`

Decorator example 1: timing pandas data processing

```python
import pandas as pd, time

def timer(func):
    """Decorator that prints runtime."""
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(f"{func.__name__} took {end - start:.3f} s")
        return result
    return wrapper

@timer
def summarize(df):
    return df.groupby("city")["value"].mean()

df = pd.DataFrame({"city": ["SF","NYC","LA","Chicago"]*3, "value": range(12
summary = summarize(df)
```

Decorator example 1: timing pandas data processing

```
summarize took 0.001 s
city
Chicago    7.0
LA         6.0
NYC        5.0
SF         4.0
Name: value, dtype: float64
```

Decorator example 2: ensure CRS for GeoPandas joins

```python
import geopandas as gpd

def ensure_crs(crs):
    """Ensure GeoDataFrames use the same CRS."""
    def decorator(func): #unlike prior example, this decorator takes an arg
        def wrapper(gdf1, gdf2, *args, **kwargs):
            if gdf1.crs != crs:
                gdf1 = gdf1.to_crs(crs)
            if gdf2.crs != crs:
                gdf2 = gdf2.to_crs(crs)
            return func(gdf1, gdf2, *args, **kwargs)
        return wrapper
    return decorator

@ensure_crs("EPSG:4326") #center at 0 lat 0 lon
def spatial_join(gdf1, gdf2):
    return gpd.sjoin(gdf1, gdf2, how="inner")
```

`@st.cache_data` decorator

- `@st.cache_data` is another (albeit much more complicated) decorator included in `streamlit`
- `@st.cache_data` saves the result of the function the first time it runs
  - Saves it in a **cache**: to computer/server memory
  - If `@st.cache_data` detects that the function is called again with the same inputs as before, it returns the saved result instead of re-running
- `@st.cache_data` trades off memory for speed

How cacheing works

```
@st.cache_data
def long_running_function(param1, param2):
    return …
```

1. When user first calls `long_running_function(p1, p2)` for the first time, `@st.cache_data` makes a note of `p1, p2`. It stores the return value in cache
2. If the next call is `long_running_function(q1, q2)`, it also stores this in the cache.
3. *(time-saving)* If the next call is `long_running_function(p1, p2)`, instead of running the code inside `long_running_function()`, it just returns the saved value from step 1

Automate toy example via cache-ing

6/session_state_cache.qmd:

```
import streamlit as st
import time


st.set_page_config(page_title="@cache_data")
st.title("Expensive Calculation Demo")

# user inputs
route = st.selectbox("CTA route to cut", ["Red Line", "Blue Line", "Green L
cut_pct = st.slider("What fraction of service to cut?", 0.0, 50.0, 10.0, st

if "counter" not in st.session_state:
    st.session_state.counter = 0

@st.cache_data()
def compute_passengers(route, cut_pct):
    with st.spinner("Crunching numbers..."):
        time.sleep(3)  # stand-in for a slow computation
```

http://uchicago-harris-dap-dashboards-6.streamlit.app/

Revisit a real example:

`4/uber_pickups.py`

We have decorated `@st.cache_data` for four different functions

1. `load_data()`
2. `filterdata(data, hour_selected)`
3. `mpoint()`
4. `histdata()`

Revisit a real example:

4/uber_pickups.py

```
1  """An example of showing geographic data."""
2
3  import os
4
5  import altair as alt
6  import numpy as np
7  import pandas as pd
8  import pydeck as pdk
9  import streamlit as st
10
11 # SETTING PAGE CONFIG TO WIDE MODE AND ADDING A TITLE AND FAVICON
12 st.set_page_config(layout="wide", page_title="NYC Ridesharing Demo", page_i
13
14
15 # LOAD DATA ONCE
16 @st.cache_data
17 def load_data():
18     path = "uber-raw-data-sep14.csv.gz"
```

do-pair-share

1. Run `$ streamlit run app_dps_2.py`
2. Edit the script to use `@st.cache_data` decorator on `compute_passengers()`
3. Record the change in speed

Summary: decorators and `@st.cache_data`

- Decorators
  - Wrap a function with extra behavior and return a modified version of function
  - Create them when you want to extend many functions in the same way
  - Syntax: `@decorator` above a function definition.
  - Decorators are not a dashboard-only tool! We can write our own decorators, or use ones that come with a package (`st.cache_data` is the latter).
- Examples
  - Timing and logging (`@timer`)
  - Validation or transformation (`@ensure_crs(crs)`)
  - Caching (`@st.cache_data`)
- `@st.cache_data` is a decorator that can be used similarly to `st.session_state`

Streamlit and other types of dashboards

Roadmap

- Tips for streamlit
- Tips for final project
- Streamlit vs other dashboarding software
- Promise and pitfalls of dashboards revisited

Streamlit Tips

- Lay out your editor and browser windows side by side, so the code and the app can be seen at the same time.
- Your main script should live in a directory other than the root directory.
- https://docs.streamlit.io/develop/quick-reference/cheat-sheet

Other Streamlit applications we will not cover (examples on the streamlit website)

- Chat box / LLM wrapper
- Draw on external data (user uploaded, from a live database, etc.)

How should you use `streamlit` for the final project (and in life)?

- Build a dashboard when a static figure or table will not suffice
- This occurs when there is too much information to fit on a single figure or table (e.g. one dashboard is better than ten figures)
- The goal, however, is not to proliferate information (e.g. “how can I imagine ten figures so that then I can shoehorn them into dashboard?”).
- Instead, it is to develop a set of information that a reader or decision-maker would want to interact with.

Streamlit and other dashboarding tools

Streamlit is not in the top 10 of most commonly-used dashboards.

What software might you see elsewhere? Microsoft Power BI, tableau, Qlik, Looker, SAP, plotly, shiny

What dashboard should you use? This is not an important choice. Use whatever your organization or collaborators are already using. Or prototype in streamlit and then port.

What if a job you apply for requires a specific dashboarding software? Spend a few hours teaching it to yourself and then put it on your resume (do **not** just put it on your resume). Or put dashbaords in general and learn the skill for the interview.

The most famous dashboard ever

New York City Police Department's "CompStat" in the 1990's (link). Why is it such a perfect application of a dashboard? (Source: *Classic book: Bob Behn's The PerformanceStat Potential*)

1. Metrics (arrests, crime rate) are clearly aligned with institutional goals
2. Data are automatically collected daily
3. Historical baseline data are available
4. Focused efforts can produce immediate results
5. All police commanders start as beat cops
6. Multiple subunits facilitate comparison
7. Size plus managerial discretion creates personnel flexibility

Very few other parts of government can meet all seven of these criteria

Skeptic

Even under these perfect conditions, the dashboard was not very successful. In a survey of 1800 retired NYC police officers of the question “are you confident that major crimes have declined by 80% in New York since 1990s?” more than half said no (Eterno, Verma, and Silverman 2014). Respondents also talked about pressure to change reports to make crime look better.

The key idea is that “what gets measured gets managed”. Once a manager focuses on a metric, that metric becomes less useful or informative than it was before.

Conclusion of section

- Streamlit is an amazing tool
- The key skill, however, is not streamlit, it is organizing information for an external analyst or decision-maker.
- Dashboards have real downsides
- In spite of these downsides, dashboards continue to grow in popularity

Backup slides

Backup: Altair troubleshooting

- It is possible to have Altair installed (as captured by `pip show altair`) but also have `import altair` fail on mac with a chip architecture error: `have 'x86_64', need 'arm64e' or 'arm64')`
- Didn’t work: make a new `venv` or update Python.
- Worked: `pip install --force-reinstall --no-cache-dir --only-binary=:all: rpds-py`

Backup: Altair troubleshooting

“This is a compatibility issue between the package’s [altair’s] generated schema and Python 3.14.”

Solution: downgrade to Python 3.12.

Backup: Streamlit troubleshooting

Make sure that your virtual environment is active!

urllib.error.URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1000)>

Solution Find your Python's "Install Certificates.command" in Finder: 1. Open /Applications. 2. Enter the folder for your Python (for example Python 3.11). 3. Double-click Install Certificates.command.