Dashboards + Shiny I (Lecture 10)

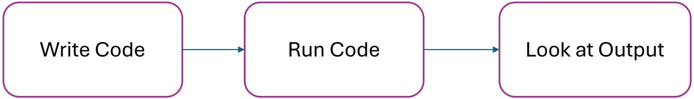
Peter Ganong and Maggie Shi

November 12, 2024

Table of contents I

Introduction

From A Static Process

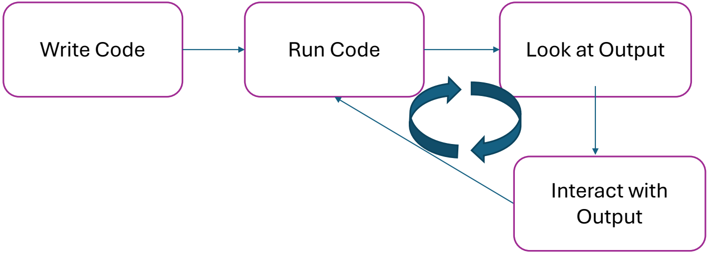


Write Code

Run Code

Look at Output

To A Dynamic Process



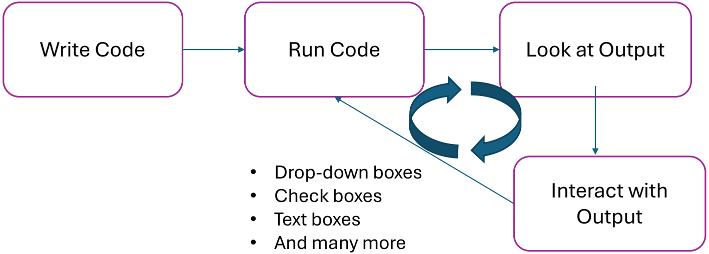
Write Code

Run Code

Look at Output

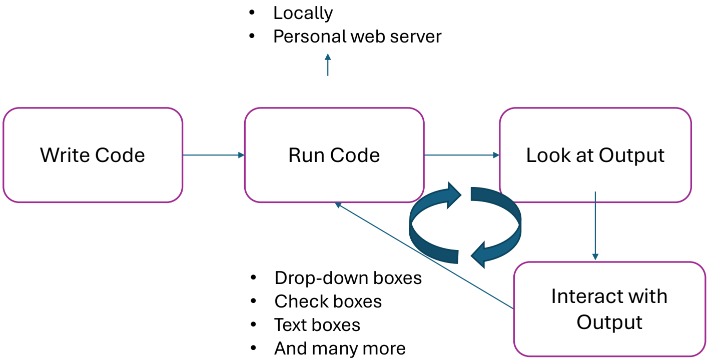
Interact with Output

To A Dynamic Process



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

To A Dynamic Process



- Locally
- Personal web server

Write Code

Run Code

Look at Output

Interact with Output

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

Dashboards

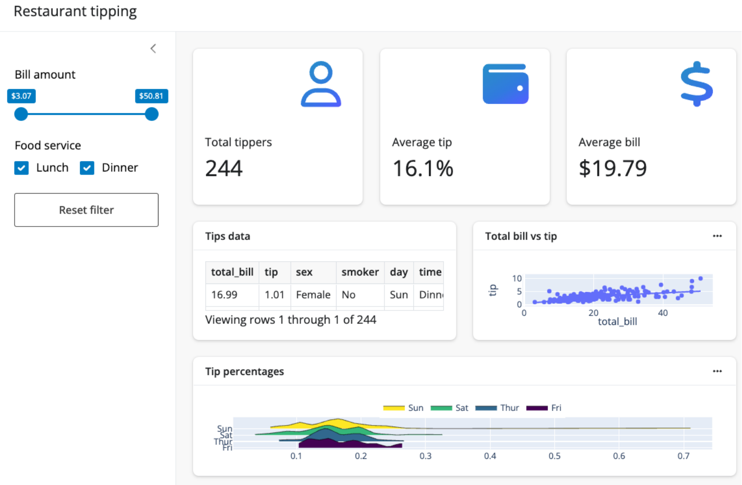
- When you analyze a dataset, your process looks a lot more like the dynamic one than the static one. Dashboards allow someone who does not code to do a limited version of the same process you go through.

- 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

- *(Impressive: they are a great way to flex your data skills for your portfolio!)*

Dashboard Examples

Tipping Dashboard (link)



Restaurant tipping

Bill amount

$3.07 $50.81

Food service

☑ Lunch ☑ Dinner

Reset filter

Total tippers

244

Average tip

16.1%

Average bill

$19.79

Tips data

total_bill	tip	sex	smoker	day	time
16.99	1.01	Female	No	Sun	Dinn

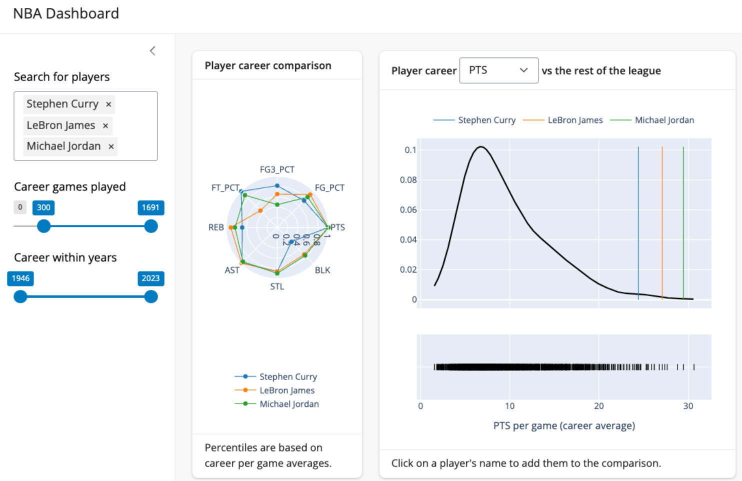
Viewing rows 1 through 1 of 244

Total bill vs tip

Tip percentages

Dashboard Examples

NBA Dashboard (link)



NBA Dashboard

Search for players

Stephen Curry ×
LeBron James ×
Michael Jordan ×

Career games played: 0, 300 – 1691

Career within years: 1946 – 2023

Player career comparison

Percentiles are based on career per game averages.

Player career PTS vs the rest of the league

Click on a player's name to add them to the comparison.

Dashboards: Promises and Pitfall

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?

Shiny

- `shiny` is a framework for creating dashboards and web apps
- Originally developed for R, but now available for Python
- In Terminal/command line:

```
$ pip install shiny
```

In this lecture: `$` means run at the terminal (on Macs the prefix is `%`)

Introduction: summary

- Dashboards let non-coders interact with data
- There are many ways to build a dashboard, we will use shiny
- If you haven't installed shiny yet, please install it now

Anatomy of All Shiny Apps

Anatomy of All Shiny Apps: Roadmap

- Introduce vocabulary: User interface (UI) and Server
- UI input elements
- Server examples
- UI output elements
- Syntax for all shiny apps

Shiny Program Components

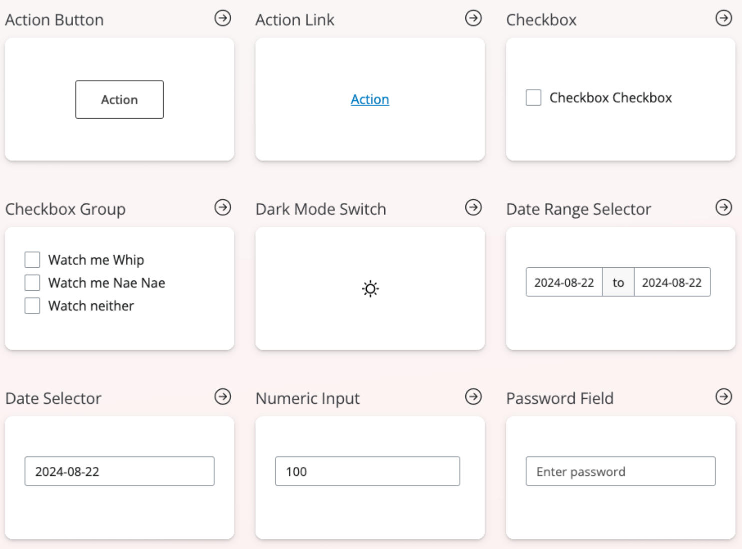
Shiny programs have 2 main components

1. **User interface (UI)** : defines the layout and elements users interact with
   - UI side is basically the ‘decorative’ part of the Shiny program – akin to web design
2. **Server**: logic that processes inputs and outputs
   - Server side is where the Python code is

- In this class, both the UI and the server run on a single computer – yours.

- Note: Shiny apps can’t be run directly out of VSCode or in code chunks. Instead, we will deploy from the command line, as we will see in a few slides

UI Examples: Input Elements



Action Button

Action

Action Link

Action

Checkbox

□ Checkbox Checkbox

Checkbox Group

□ Watch me Whip
□ Watch me Nae Nae
□ Watch neither

Dark Mode Switch

☼

Date Range Selector

2024-08-22 to 2024-08-22

Date Selector

2024-08-22

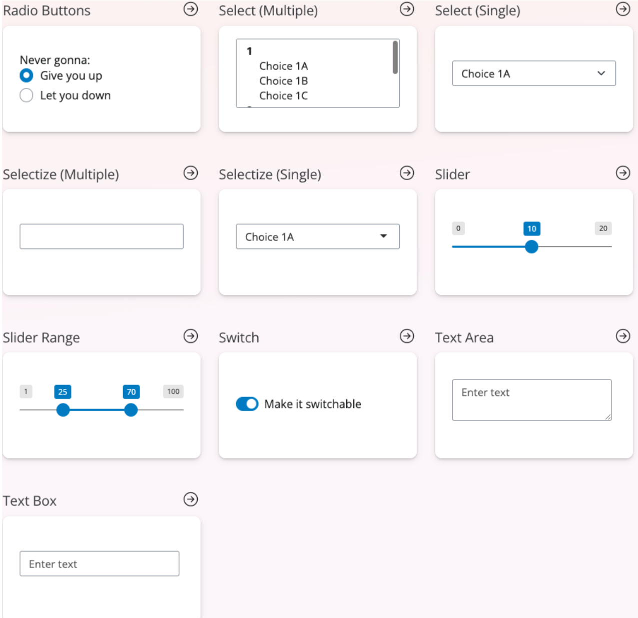
Numeric Input

100

Password Field

Enter password

UI Examples: Input Elements



Radio Buttons

Never gonna:
● Give you up
○ Let you down

Select (Multiple)

1
Choice 1A
Choice 1B
Choice 1C

Select (Single)

Choice 1A

Selectize (Multiple)

Selectize (Single)

Choice 1A

Slider

0 10 20

Slider Range

1 25 70 100

Switch

Make it switchable

Text Area

Enter text

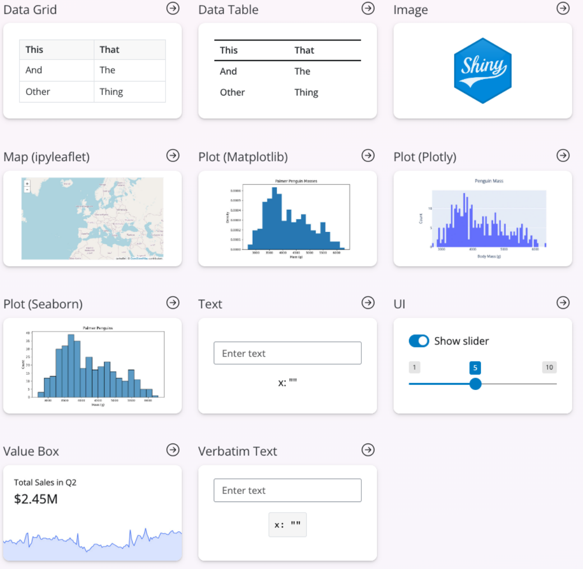
Text Box

Enter text

Server Examples

- Read in a CSV file
- Make a graph
- Make a map
- Compute statistics for a place/time period/subgroup of interest
- Run a machine learning algorithm, report predictions in a table

UI Examples: Output Elements



Data Grid

This	That
And	The
Other	Thing

Data Table

This	That
And	The
Other	Thing

Image

Map (ipyleaflet)

Plot (Matplotlib)

Plot (Plotly)

Plot (Seaborn)

Text

Enter text

x: ""

UI

Show slider

Value Box

Total Sales in Q2

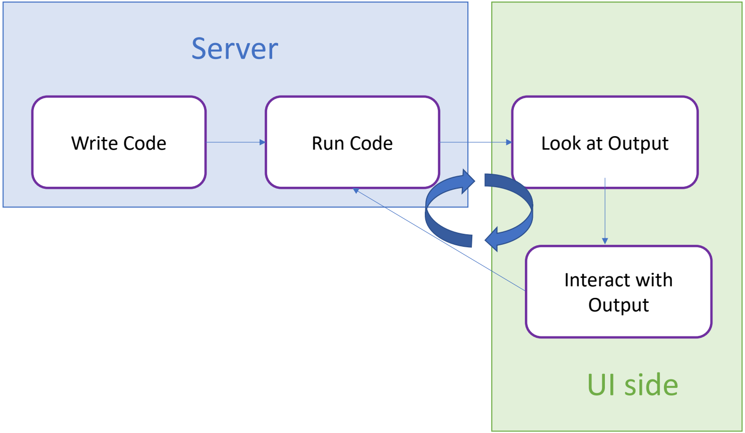
$2.45M

Verbatim Text

Enter text

```
x: ""
```

UI and Server



Server

Write Code

Run Code

Look at Output

Interact with Output

UI side

Shiny App Syntax

Shiny apps always have the following structure:

```
from shiny import App, render, ui

app_ui = ui.page_fluid(
    [CODE TO LAY OUT THE PAGE]
)

def server(input, output,
 ↪  session):
    [CODE THAT HANDLES PYTHON]

app = App(app_ui, server)
```

- **UI**: `app_ui = ui.page_fluid(...)`

Shiny App Syntax

Shiny apps always have the following structure:

```
from shiny import App, render, ui

app_ui = ui.page_fluid(
    [CODE TO LAY OUT THE PAGE]
)


def server(input, output,
 ↪  session):
    [CODE THAT HANDLES PYTHON]


app = App(app_ui, server)
```

- **UI**: `app_ui = ui.page_fluid(...)`
- **Server**: `def server(input, output, session):`

Shiny App Syntax

Shiny apps always have the following structure:

```python
from shiny import App, render, ui

app_ui = ui.page_fluid(
    [CODE TO LAY OUT THE PAGE]
)

def server(input, output, session):
    [CODE THAT HANDLES PYTHON]

app = App(app_ui, server)
```

- **UI**: `app_ui = ui.page_fluid(...)`
- **Server**: `def server(input, output, session):`
- **App**: always ends with `app = App(app_ui, server)`

Anatomy of All Shiny Apps: Summary

- Shiny app is always composed of a **UI** side and **server** side
- UI side has elements that take in *input*: check boxes, numeric inputs, slider
- Pass these inputs to the server, get back computation results
- UI then displays *output*: text, figures, data

Build Your First Shiny App

Your First Shiny App: Roadmap

Steps

1. Navigate through command line to the folder where the app will be stored

2. Create a basic app

3. Deploy and display it in a web browser

Plan of action: I will show this step-by-step and then you will try it.

Creating a New Shiny App

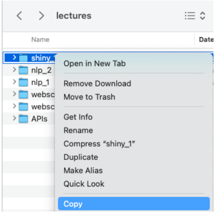
- We initialize Shiny apps from the command line – not in Python
- This means our usual workflow of running code chunks out of VSCode won't work
- Instead, we'll do the following:

Creating a New Shiny App

- We initialize Shiny apps from the command line – not in Python
- This means our usual workflow of running code chunks out of VSCode won't work
- Instead, we'll do the following:
- First, decide which folder you want the Shiny app to be using the finder
- Then navigate there using the command line. How-to's on next slide

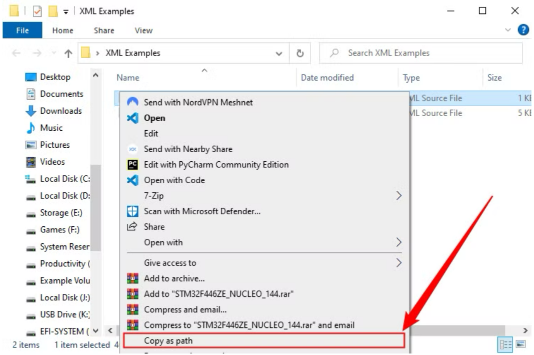
Macs

- Right click and hover over “Copy”

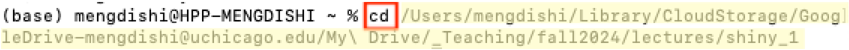


- **Before** clicking “Copy”, press “Option” key ( )
- It should switch to “Copy [foldername] as Pathname”

Windows



“Copy as Path”
Source: How-to Geek



```
(base) mengdishi@HPP-MENGDISHI ~ % cd /Users/mengdishi/Library/CloudStorage/Goog
leDrive-mengdishi@uchicago.edu/My\ Drive/_Teaching/fall2024/lectures/shiny_1
```

```
$ cd <dir_for_shiny_app>
```

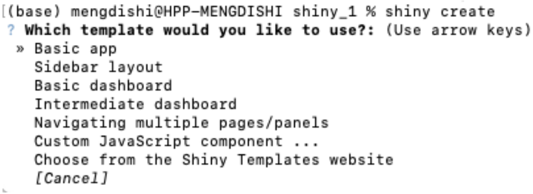
Create Your Shiny App

Within that folder, type in Terminal:

```
$ shiny create
```

- If you're getting errors that `shiny` is not a recognized command, try `python -m shiny create` or `python -m shiny create`

- Choose "Basic app" as the template
  - In a future lecture we will explore the other templates



```
[(base) mengdishi@HPP-MENGDISHI shiny_1 % shiny create
 ? Which template would you like to use?: (Use arrow keys)
  » Basic app
    Sidebar layout
    Basic dashboard
    Intermediate dashboard
    Navigating multiple pages/panels
    Custom JavaScript component ...
    Choose from the Shiny Templates website
    [Cancel]
```

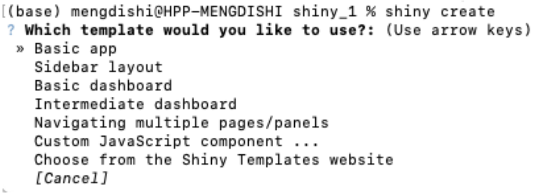
Create Your Shiny App

Within that folder, type in Terminal:

```
$ shiny create
```

- If you're getting errors that `shiny` is not a recognized command, try `python -m shiny create` or `python -m shiny create`

- Choose "Basic app" as the template
  - In a future lecture we will explore the other templates

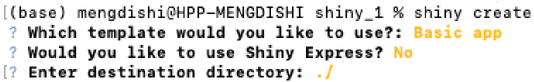


```
[(base) mengdishi@HPP-MENGDISHI shiny_1 % shiny create
 ? Which template would you like to use?: (Use arrow keys)
  » Basic app
    Sidebar layout
    Basic dashboard
    Intermediate dashboard
    Navigating multiple pages/panels
    Custom JavaScript component ...
    Choose from the Shiny Templates website
    [Cancel]
```

- Don't use Shiny Express

Create Your Shiny App (continued)

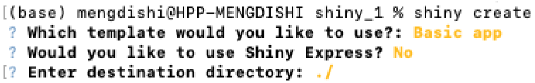
- Click “Enter” for destination category (since you’ve already `cd`-ed into that directory)



```
[(base) mengdishi@HPP-MENGDISHI shiny_1 % shiny create
 ? Which template would you like to use?: Basic app
 ? Would you like to use Shiny Express? No
[? Enter destination directory: ./
```

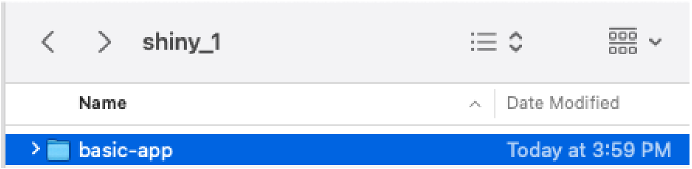
Create Your Shiny App (continued)

- Click “Enter” for destination category (since you’ve already `cd`-ed into that directory)



```
[(base) mengdishi@HPP-MENGDISHI shiny_1 % shiny create
 ? Which template would you like to use?: Basic app
 ? Would you like to use Shiny Express? No
[? Enter destination directory: ./
```

- You should see a new folder titled `basic-app` in your directory



Name	Date Modified
basic-app	Today at 3:59 PM

Your New Shiny App

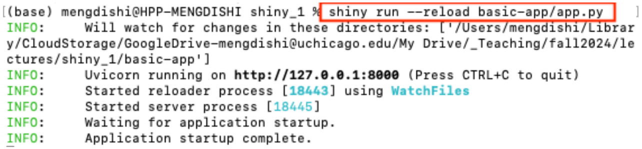
Inside basic-app, you should see an app.py file

```python
from shiny import App, render, ui

app_ui = ui.page_fluid(
    ui.panel_title("Hello Shiny!"),
    ui.input_slider("n", "N", 0, 100, 20),
    ui.output_text_verbatim("txt"),
)

def server(input, output, session):
    @render.text
    def txt():
        return f"n*2 is {input.n() * 2}"

app = App(app_ui, server)
```

Deploying Your New Shiny App

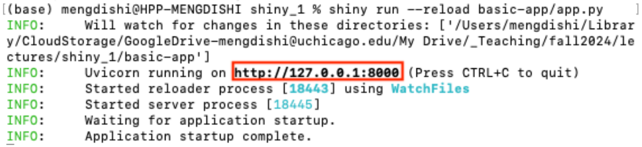
- Shiny apps are deployed out of Terminal with the following line:

```
$ shiny run --reload basic-app/app.py
```



```
[(base) mengdishi@HPP-MENGDISHI shiny_1 % shiny run --reload basic-app/app.py     ]
INFO:     Will watch for changes in these directories: ['/Users/mengdishi/Librar
y/CloudStorage/GoogleDrive-mengdishi@uchicago.edu/My Drive/_Teaching/fall2024/le
ctures/shiny_1/basic-app']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [18443] using WatchFiles
INFO:     Started server process [18445]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

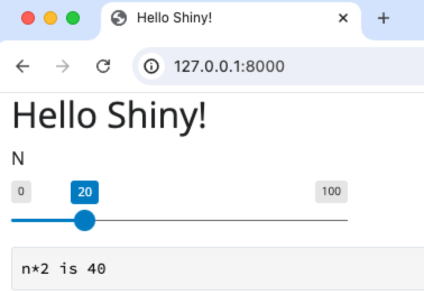
- If the app doesn't automatically load, copy the IP address into your browser



```
[(base) mengdishi@HPP-MENGDISHI shiny_1 % shiny run --reload basic-app/app.py     ]
INFO:     Will watch for changes in these directories: ['/Users/mengdishi/Librar
y/CloudStorage/GoogleDrive-mengdishi@uchicago.edu/My Drive/_Teaching/fall2024/le
ctures/shiny_1/basic-app']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [18443] using WatchFiles
INFO:     Started server process [18445]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Deploying Your New Shiny App

- In your browser, you should see a basic app with a slider from 0 to 100



Hello Shiny!

127.0.0.1:8000

Hello Shiny!

N

0 20 100

```
n*2 is 40
```

In-class exercise

Follow instructions on slides 26-32 to deploy `basic-app`

```
$ pip install shiny
$ cd name-of-directory/
$ shiny create
# choose "Basic app", Shiny Express -> No, directory <accept_default> #
$ shiny run basic-app/app.py
```

In browser, verify that you can move the slider. Quit in terminal using CTRL+C .

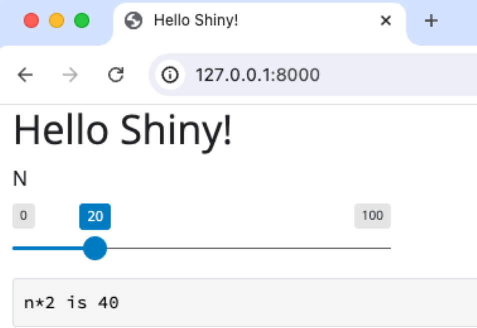
Tips:

- If the app doesn't automatically load, copy the URL into your browser.
- If you're getting errors that `shiny` is not a recognized command, try `python -m shiny run basic-app/app.py` or `python -m shiny run basic-app/app.py`
- Default directory names may be something other than `basic-app`
- When you are done, help a neighbor

Understand Your First Shiny App: UI

Understand Your First Shiny App: Roadmap

```python
from shiny import App, render, ui

app_ui = ui.page_fluid(
    ui.panel_title("Hello Shiny!"),
    ui.input_slider("n", "N", 0, 100, 20),
    ui.output_text_verbatim("txt"),
)

def server(input, output, session):
    @render.text
    def txt():
        return f"n*2 is {input.n() * 2}"

app = App(app_ui, server)
```



Hello Shiny!

N

```
n*2 is 40
```

Walk through

- `ui.panel_title()`
- `ui.input_slider()`
- `ui.output_text_verbatim()`

Going back to our basic app: app.py

```python
from shiny import App, render, ui

app_ui = ui.page_fluid(
    ui.panel_title("Hello Shiny!"),
    ui.input_slider("n", "N", 0, 100, 20),
    ui.output_text_verbatim("txt"),
)

def server(input, output, session):
    @render.text
    def txt():
        return f"n*2 is {input.n() * 2}"

app = App(app_ui, server)
```

A UI-only component: title

```python
ui.panel_title("Hello Shiny!")
```

- This remains static and doesn't require any computation

- So it doesn't appear on the server side

Going back to our basic app: app.py

```python
from shiny import App, render, ui

app_ui = ui.page_fluid(
    ui.panel_title("Hello Shiny!"),
    ui.input_slider("n", "N", 0, 100, 20),
    ui.output_text_verbatim("txt"),
)

def server(input, output, session):
    @render.text
    def txt():
        return f"n*2 is {input.n() * 2}"

app = App(app_ui, server)
```

Component: **input** slider

```python
ui.input_slider("n", "N", 0, 100, 20)
```

Going back to our basic app: app.py

```python
from shiny import App, render, ui

app_ui = ui.page_fluid(
    ui.panel_title("Hello Shiny!"),
    ui.input_slider("n", "N", 0, 100, 20),
    ui.output_text_verbatim("txt"),
)

def server(input, output, session):
    @render.text
    def txt():
        return f"n*2 is {input.n() * 2}"

app = App(app_ui, server)
```

Component: **input** slider

```python
ui.input_slider("n", "N", 0, 100, 20)
```

- "n": value from user input
- "N": label
- 0: minimum of slider
- 100: maximum of slider
- 20: default value of slider
- HLink to sliderInput documentation

Going back to our basic app: app.py

```python
from shiny import App, render, ui

app_ui = ui.page_fluid(
    ui.panel_title("Hello Shiny!"),
    ui.input_slider("n", "N", 0, 100, 20),
    ui.output_text_verbatim("txt"),
)

def server(input, output, session):
    @render.text
    def txt():
        return f"n*2 is {input.n() * 2}"

app = App(app_ui, server)
```

Component: **output** text verbatim

UI:

```python
ui.output_text_verbatim("txt")
```

- This component renders text
- What is `"txt"`?

Going back to our basic app: app.py

```python
from shiny import App, render, ui

app_ui = ui.page_fluid(
    ui.panel_title("Hello Shiny!"),
    ui.input_slider("n", "N", 0, 100, 20),
    ui.output_text_verbatim("txt"),
)

def server(input, output, session):
    @render.text
    def txt():
        return f"n*2 is {input.n() * 2}"

app = App(app_ui, server)
```

Component: **output** text verbatim

UI:

```python
ui.output_text_verbatim("txt")
```

- This component renders text
- What is "txt"?

Server:

```python
@render.text
def txt():
    return f"n*2 is {input.n() * 2}"
```

- Shiny's syntax: "txt" on UI side corresponds to txt() on the server side
- Additional syntax: @render.text indicates that txt() should be rendered as text

Render Decorators

- Any function whose output you want to display must be wrapped with a **render decorators** of the correct type

- The render decorator is always followed by the definition of a function (e.g., `def txt():`)

UI Side	**Server Side**
‘ui.output_plot‘	‘@render.plot‘
‘ui.output_text‘, ‘ui.output_text_verbatim‘	‘@render.text‘
‘ui.output_table‘	‘@render.table‘

Understand Your First Shiny App: Summary

- UI: ask user for `ui.input_slider("n")`
- Server: receive input as as `input.n()`
- Server: compute `input.n() * 2`
- Server: get output from server as `def txt():`
  - We have to declare what this output will be eventually rendered as using a decorator (`@render.text`)
- UI: display output as `ui.output_text_verbatim("txt")`

Generalizing from this Example:

You can now see the general pattern:

- *Syntax for defining input*: `"x"` on UI side becomes `input.x()` on server side
- *Syntax for defining output*: `def value():` on server side becomes `"value"` on UI side
- *Syntax for rendering output*: render decorators like `@render.plot`, `@render.table`

Build Your Second Shiny App

Build Your Second Shiny App: Roadmap

Goal: display a histogram of a normally-distributed sample with mean `mu`

Steps

1. Install package to show graphs in Altair
2. Server side code to compute a normally-distributed sample with mean `mu`
3. UI side code to ask for `mu` and to display the plot

Jupyter Widgets

- Shiny supports Jupyter Widgets via `shinywidgets` package
  - About 10% of students needed to instead use a different `anywidget`

- We'll focus on using `shinywidgets` to incorporate `altair` plots, but it also supports many other interactive widgets from Jupyter Notebooks: `plotly`, `pydeck`, `bokeh`, etc.

```
$ pip install shinywidgets
```

Altair Jupyter Widget

- **UI-side**: `shinywidgets` has its own UI output element: `output_widget()`
- **Server-side**: Altair has its own render function through `shinywidgets`: `@render_altair`
- First, import required packages

```python
from shinywidgets import render_altair, output_widget
import altair as alt
```

Build Your Second Shiny App

- Starting on server side of `normal_distn_app/app.py`:

```python
def server(input, output, session):
    # [other server-side code]
    @render_altair
    def my_hist():
        sample = np.random.normal(input.mu(), 20, 100)
        df = pd.DataFrame({'sample': sample})
        return(
            alt.Chart(df).mark_bar().encode(
                alt.X('sample:Q', bin=True),
                alt.Y("count()")
            )
        )
```

Build Your Second Shiny App

- Then moving on to the UI side:

Build Your Second Shiny App

- Then moving on to the UI side:

- We have defined plot as `my_hist()` on the server side

- But on the UI side, we have to call it `"my_hist"`

Build Your Second Shiny App

- Then moving on to the UI side:
- We have defined plot as `my_hist()` on the server side
- But on the UI side, we have to call it "`my_hist`"

```
app_ui = ui.page_fluid(
    ui.panel_title("Histogram of 100 Draws from Normal with mean
 ↪  mu"),
    ui.input_slider("mu", "N", 0, 100, 20),
    output_widget("my_hist")
)
```

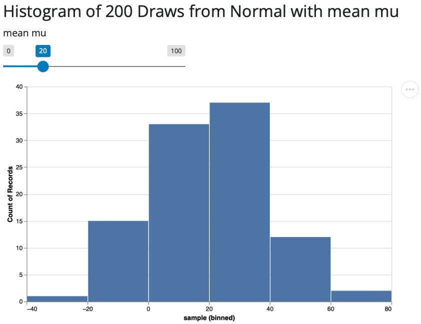
Build Your Second Shiny App

- One last thing: add libraries at the top

```python
from shiny import App, render, ui
from shinywidgets import render_altair, output_widget
import altair as alt
import numpy as np
```

Build Your Second Shiny App

```
$ shiny run --reload normal_distn_app/app.py
```



Histogram of 200 Draws from Normal with mean mu

This app is stored in the folder `shiny_10/apps_after_class`

Build Your Second Shiny App: Summary

- Ask user for `ui.input_slider("mu")`
- Give input to server as `input.mu()`
- Simulate numbers drawn from a distribution with mean `mu`
- Define output using as `def my_hist():`
  - Render decorator for altair output: `@render_altair`
- Display output as `output_widget("my_hist")`

Debugging

Debugging: Roadmap

- We will now introduce some typos to show how they manifest
- Mis-typed the plot color
- Mis-typed a Shiny function
- Used wrong `render`

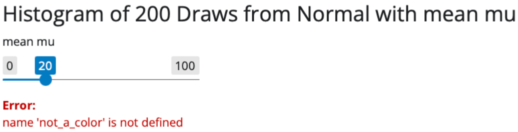
Example 1

- Say that we mis-typed the plot color

```
@render_altair
def my_hist():
    sample = np.random.normal(input.n(), 20, 100)
    df = pd.DataFrame({'sample': sample})
    return (
        alt.Chart(df)
        .mark_bar(color = not_a_color)
        .encode(x=alt.X('sample:Q', bin=True), y="count()")
    )
```

Example 1, continued

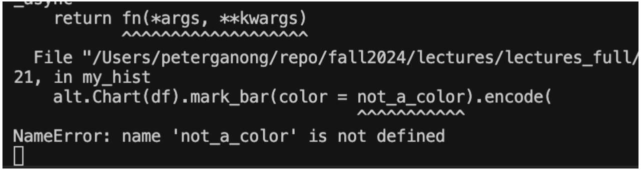
- Once we save app.py, the app will refresh and display:



Histogram of 200 Draws from Normal with mean mu

Error:
name 'not_a_color' is not defined

- And we will also get a similar error in terminal



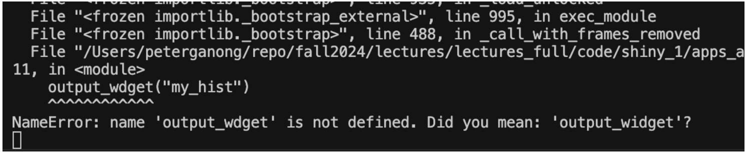
```
_async
    return fn(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^

  File "/Users/peterganong/repo/fall2024/lectures/lectures_full/
21, in my_hist
    alt.Chart(df).mark_bar(color = not_a_color).encode(
                                   ^^^^^^^^^^^

NameError: name 'not_a_color' is not defined
```

Example 2

- But say that we mis-typed one of the UI elements: `output_wdget()`

```
app_ui = ui.page_fluid(
    ui.panel_title("Histogram of 100 Draws from Normal with mean
 ↪  mu"),
    ui.input_slider("mu", "N", 0, 100, 20),
    output_wdget("my_hist")
)
```

- The app won't load at all and the error message is in Terminal



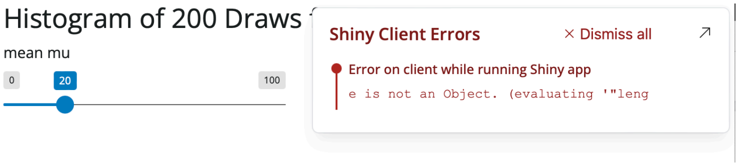
```
  File "<frozen importlib._bootstrap_external>", line 995, in exec_module
  File "<frozen importlib._bootstrap>", line 488, in _call_with_frames_removed
  File "/Users/peterganong/repo/fall2024/lectures/lectures_full/code/shiny_1/apps_a
11, in <module>
    output_wdget("my_hist")
    ^^^^^^^^^^^^

NameError: name 'output_wdget' is not defined. Did you mean: 'output_widget'?
```

Example 3

- Say that we used `@render.text` instead of `@render_altair`

```python
@render.text
def my_hist():
        sample = np.random.normal(input.mu(), 20, 100)
        df = pd.DataFrame({'sample': sample})
        return(
            alt.Chart(df).mark_bar().encode(
                alt.X('sample:Q', bin=True),
                alt.Y("count()")
            )
        )
```

Example 3, continued



Histogram of 200 Draws f

mean mu

Shiny Client Errors

Error on client while running Shiny app

```
e is not an Object. (evaluating '"leng
```

In-Class Exercise

Try to run `apps_for_class/normal_distn_app/app_to_debug.py`. Debug the errors.

Debugging: Summary

- Plain vanilla python errors (`wrongcolor`) will typically show up in the web app.
- Shiny-specific errors (`output_wdget`) will show up in Terminal
- Output definition on server side needs to also include "render decorator" functions: `@render_altair`, `@render.text`, etc.
- Develop your app piece-by-piece and keep refreshing the app to debug as you go

Whole Lecture Summary

Dashboards are a way to give (limited) Python access to your non-coding friends, managers, or the public

Steps to a dashboard in Shiny

1. UI takes user input
2. Send it to the server
3. Run Python on the server
4. Write a Python function which returns material to display
5. Display it in the UI

Debugging is trickier with apps, since errors can be at command line or in the app itself