DashR & Speeding up Dashboards

March 14, 2022

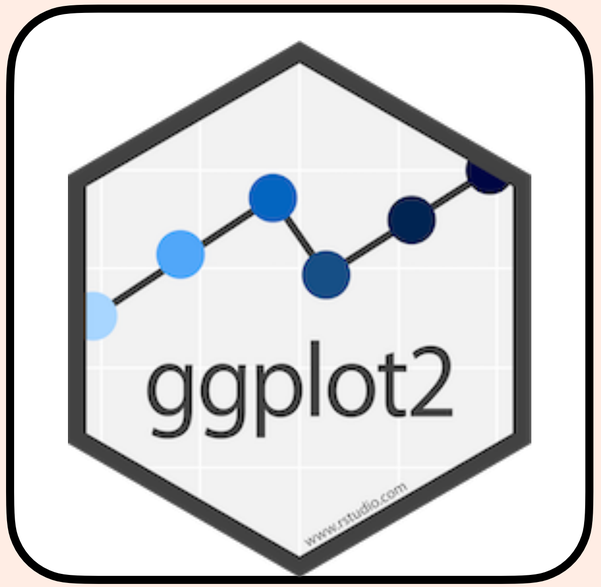
DATA 551 - Lecture 7

20 minutes

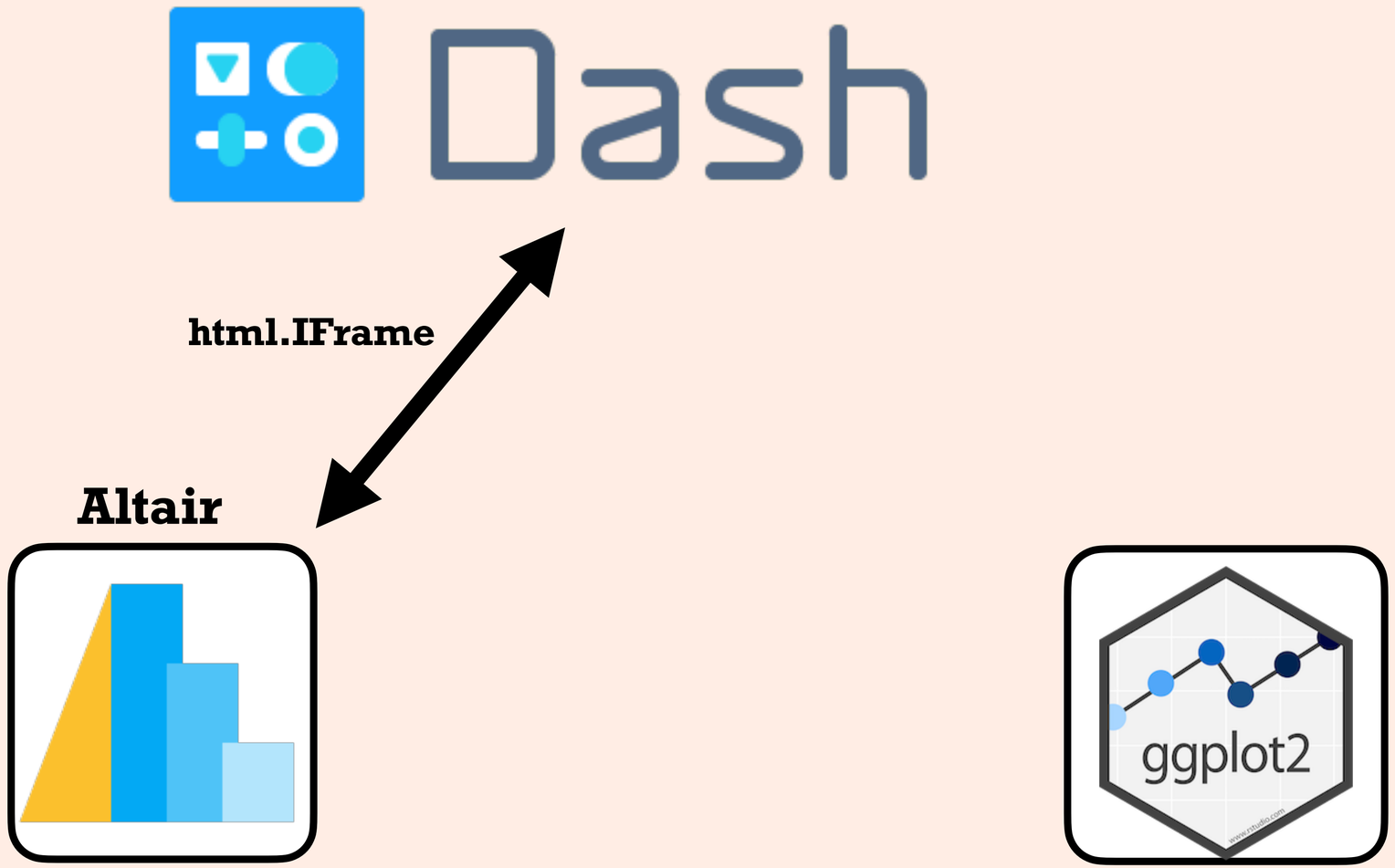
Part 1: Dash and R

Recall our plan…

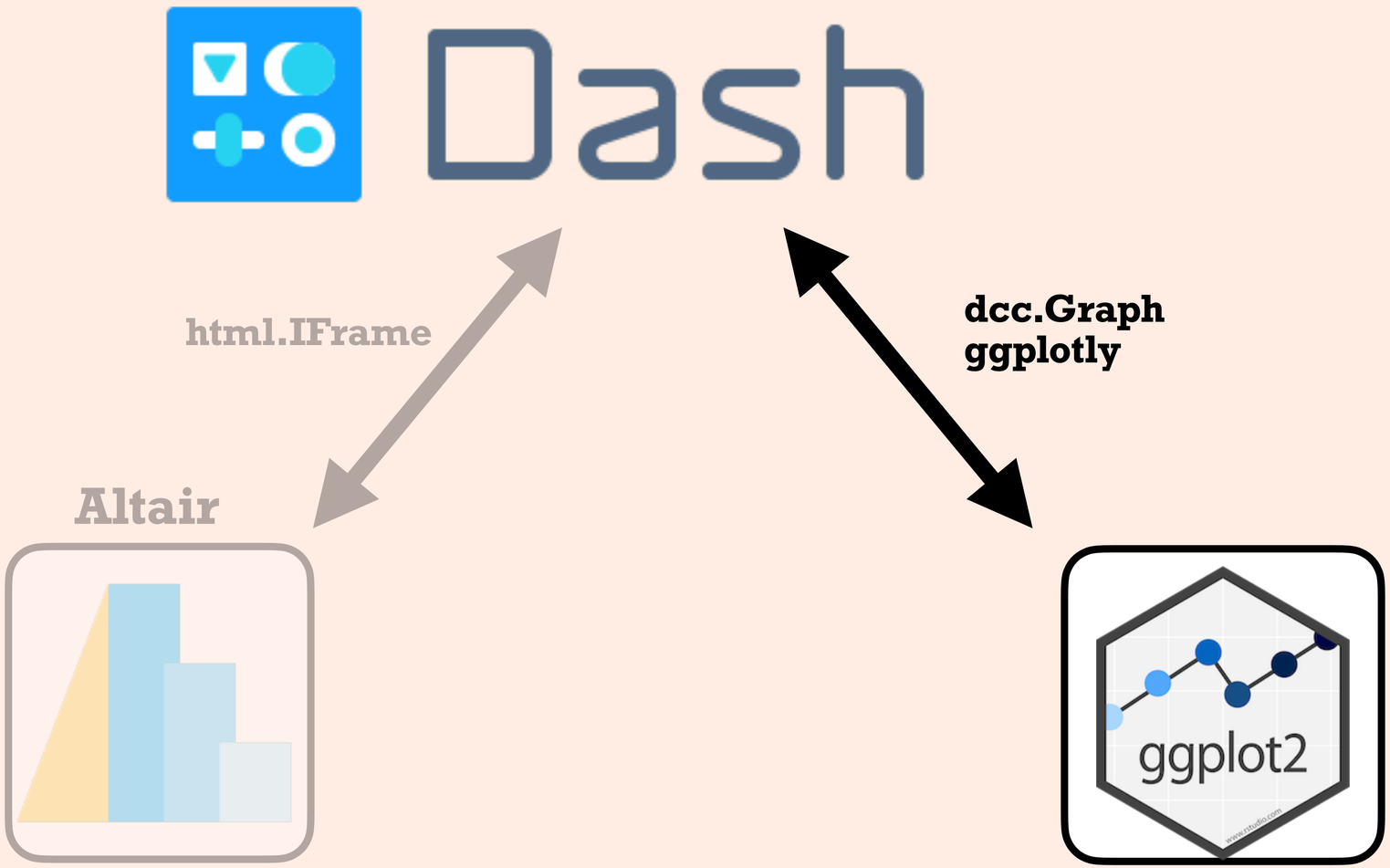
Dash



Recall our plan…



Recall our plan...

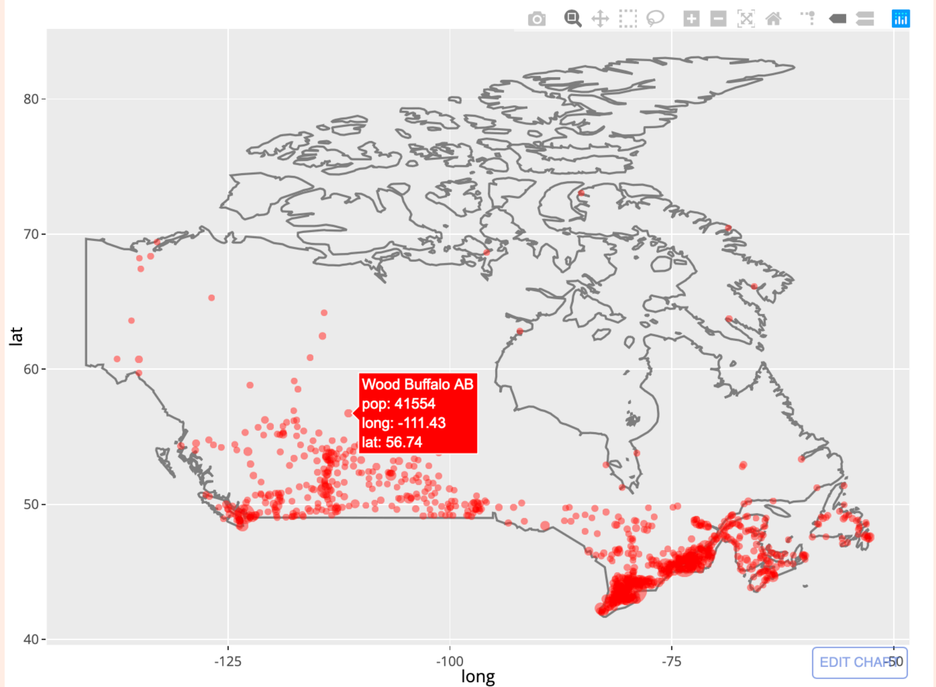


ggplot2 and plotly

- Add interactivity to ggplot2:
  - Use the plotly function called “ggplotly”

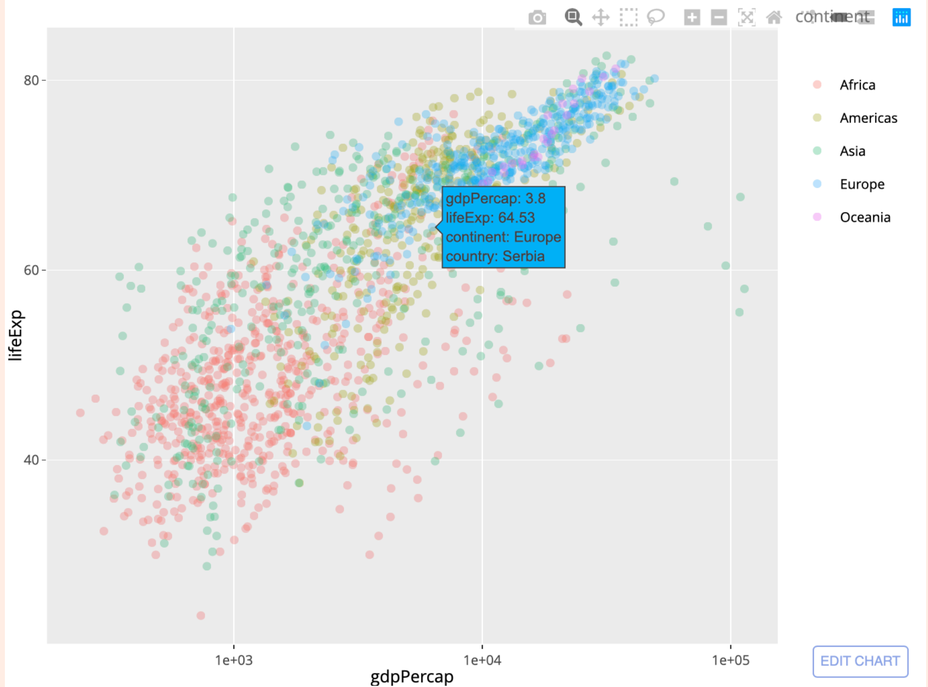
```
p <- ggplotly(p)

# Create a shareable link to your chart
# Set up API credentials: https://plot.ly/r/getting-started
chart_link = plotly_POST(p, filename="hover/map")
chart_link
```




```
p <- ggplotly(p)

# Create a shareable link to your chart
# Set up API credentials: https://plot.ly/r/getting-started
chart_link = plotly_POST(p, filename="hover/tooltip")
chart_link
```



Source: ggplotly documentation

Creating your DashR app

- For Milestone 3, goal is to get to a **functional app**;
  - Focus on adding interactivity to plots (ggplotly)
  - Deployment on Heroku is challenging (because of how long it takes)
    - Updated instructions to deploying on Heroku for DashR
    - Only one deployed app is sufficient

Creating your DashR app

- You do **NOT** need to wrangle everything again in python
    - Look for a clever way to export data to .csv/.json and import it into your R app
- Be patient with DashR (might be bugs, we do not expect feature-parity with Dash-python yet)

Callbacks in R

```
library(dash)
library(dashCoreComponents)

app <- dash_app()

app %>% set_layout(
  html$h6("Change the value in the text box to see callbacks in action!"),
  div(
    "Input: ",
    dccInput(id = 'my-input', value = 'initial value', type = 'text')
  ),
  br(),
  div(id = 'my-output')
)

app %>% add_callback(
  output(id = 'my-output', property = 'children'),
  input(id = 'my-input', property = 'value'),
  function(input_value) {
    sprintf("Output: \"%s\"", input_value)
  }
)

app %>% run_app()
```

Source: DashR Tutorial

20 minutes

Part 2: Maintaining Dashboards

Source: 512pixels, Apple WWDC 2009

For your projects…

1. Implement known bug-fixes and work-arounds.
2. Implement feedback you receive from peers during feedback session.
3. Clean-up and refactor code to make it more readable.
4. Reformat your code and add docstrings (PEP8 style).
5. If needed, reorganize and restructure your files/directory structure to separate different structures (see the [Dash docs - Structuring a Multi-Page App) for suggested organizations).
6. Apply principles of creating effective dashboards.
7. Implement any new features (if time permits).

Refactor code 1

Let's say we had a dropdown of all the countries.
Do we really need to list every country in a dictionary?

```
dcc.Dropdown(
    options=[
        {'label': 'Afghanistan', 'value':'Afghanistan'},
        {'label': 'Argentina', 'value':'Argentina'},
        {'label': 'Aruba', 'value':'Aruba'},
        {'label': 'Australia', 'value':'Australia'},
        {'label': 'Austria', 'value':'Austria'},
        {'label': 'Bahamas', 'value':'Bahamas'},
        {'label': 'Bangladesh', 'value':'Bangladesh'},
        {'label': 'Barbados', 'value':'Barbados'},
        {'label': 'Belgium', 'value':'Belgium'},
        {'label': 'Bolivia', 'value':'Bolivia'},
        {'label': 'Brazil', 'value':'Brazil'},
        {'label': 'Canada', 'value':'Canada'},
        {'label': 'Chile', 'value':'Chile'},
        {'label': 'China', 'value':'China'},
        {'label': 'Colombia', 'value':'Colombia'},
        {'label': 'Costa Rica', 'value':'Costa Rica'},
        {'label': 'Croatia', 'value':'Croatia'},
        {'label': 'Cuba', 'value':'Cuba'},
        {'label': 'Dominican Republic', 'value':'Dominican Republic'},
        {'label': 'Ecuador', 'value':'Ecuador'},
        {'label': 'Egypt', 'value':'Egypt'},
        {'label': 'El Salvador', 'value':'El Salvador'},
        {'label': 'Finland', 'value':'Finland'},
        {'label': 'France', 'value':'France'},
        {'label': 'Georgia', 'value':'Georgia'},
        {'label': 'Germany', 'value':'Germany'},
```

Refactor code 1

Let's say we had a dropdown of all the countries.
Do we really need to list every country in a dictionary?

```
dcc.Dropdown(
    options=[
        {'label': 'Afghanistan', 'value':'Afghanistan'},
        {'label': 'Argentina', 'value':'Argentina'},
        {'label': 'Aruba', 'value':'Aruba'},
        {'label': 'Australia', 'value':'Australia'},
        {'label': 'Austria', 'value':'Austria'},
        {'label': 'Bahamas', 'value':'Bahamas'},
        {'label': 'Bangladesh', 'value':'Bangladesh'},
        {'label': 'Barbados', 'value':'Barbados'},
        {'label': 'Belgium', 'value':'Belgium'},
        {'label': 'Bolivia', 'value':'Bolivia'},
        {'label': 'Brazil', 'value':'Brazil'},
        {'label': 'Canada', 'value':'Canada'},
        {'label': 'Chile', 'value':'Chile'},
        {'label': 'China', 'value':'China'},
        {'label': 'Colombia', 'value':'Colombia'},
        {'label': 'Costa Rica', 'value':'Costa Rica'},
        {'label': 'Croatia', 'value':'Croatia'},
        {'label': 'Cuba', 'value':'Cuba'},
        {'label': 'Dominican Republic', 'value':'Dominican Republic'},
        {'label': 'Ecuador', 'value':'Ecuador'},
        {'label': 'Egypt', 'value':'Egypt'},
        {'label': 'El Salvador', 'value':'El Salvador'},
        {'label': 'Finland', 'value':'Finland'},
        {'label': 'France', 'value':'France'},
        {'label': 'Georgia', 'value':'Georgia'},
        {'label': 'Germany', 'value':'Germany'},
```

There is a better way!

Python List Comprehension

```
country_options = [{'label' : ctry, 'value' : ctry} for ctry in countries]
country_options
```

```
[{'label': 'Afghanistan', 'value': 'Afghanistan'},
 {'label': 'Argentina', 'value': 'Argentina'},
 {'label': 'Aruba', 'value': 'Aruba'},
 {'label': 'Australia', 'value': 'Australia'},
 {'label': 'Austria', 'value': 'Austria'},
```

Refactor code 1

Let's say we had a dropdown of all the countries.
Do we really need to list every country in a dictionary?

```
dcc.Dropdown(
    options=[
        {'label': 'Afghanistan', 'value':'Afghanistan'},
        {'label': 'Argentina', 'value':'Argentina'},
        {'label': 'Aruba', 'value':'Aruba'},
        {'label': 'Australia', 'value':'Australia'},
        {'label': 'Austria', 'value':'Austria'},
        {'label': 'Bahamas', 'value':'Bahamas'},
        {'label': 'Bangladesh', 'value':'Bangladesh'},
        {'label': 'Barbados', 'value':'Barbados'},
        {'label': 'Belgium', 'value':'Belgium'},
        {'label': 'Bolivia', 'value':'Bolivia'},
        {'label': 'Brazil', 'value':'Brazil'},
        {'label': 'Canada', 'value':'Canada'},
        {'label': 'Chile', 'value':'Chile'},
        {'label': 'China', 'value':'China'},
        {'label': 'Colombia', 'value':'Colombia'},
        {'label': 'Costa Rica', 'value':'Costa Rica'},
        {'label': 'Croatia', 'value':'Croatia'},
        {'label': 'Cuba', 'value':'Cuba'},
        {'label': 'Dominican Republic', 'value':'Dominican Republic'},
        {'label': 'Ecuador', 'value':'Ecuador'},
        {'label': 'Egypt', 'value':'Egypt'},
        {'label': 'El Salvador', 'value':'El Salvador'},
        {'label': 'Finland', 'value':'Finland'},
        {'label': 'France', 'value':'France'},
        {'label': 'Georgia', 'value':'Georgia'},
        {'label': 'Germany', 'value':'Germany'},
```

There is a better way!

Dictionary inside list comprehension

```
country_options = [{'label' : ctry, 'value' : ctry} for ctry in countries]
country_options
```

```
[{'label': 'Afghanistan', 'value': 'Afghanistan'},
 {'label': 'Argentina', 'value': 'Argentina'},
 {'label': 'Aruba', 'value': 'Aruba'},
 {'label': 'Australia', 'value': 'Australia'},
 {'label': 'Austria', 'value': 'Austria'},
```

Refactor code 1

Let's say we had a dropdown of all the countries.
Do we really need to list every country in a dictionary?

```
dcc.Dropdown(
    options=[
        {'label': 'Afghanistan', 'value':'Afghanistan'},
        {'label': 'Argentina', 'value':'Argentina'},
        {'label': 'Aruba', 'value':'Aruba'},
        {'label': 'Australia', 'value':'Australia'},
        {'label': 'Austria', 'value':'Austria'},
        {'label': 'Bahamas', 'value':'Bahamas'},
        {'label': 'Bangladesh', 'value':'Bangladesh'},
        {'label': 'Barbados', 'value':'Barbados'},
        {'label': 'Belgium', 'value':'Belgium'},
        {'label': 'Bolivia', 'value':'Bolivia'},
        {'label': 'Brazil', 'value':'Brazil'},
        {'label': 'Canada', 'value':'Canada'},
        {'label': 'Chile', 'value':'Chile'},
        {'label': 'China', 'value':'China'},
        {'label': 'Colombia', 'value':'Colombia'},
        {'label': 'Costa Rica', 'value':'Costa Rica'},
        {'label': 'Croatia', 'value':'Croatia'},
        {'label': 'Cuba', 'value':'Cuba'},
        {'label': 'Dominican Republic', 'value':'Dominican Republic'},
        {'label': 'Ecuador', 'value':'Ecuador'},
        {'label': 'Egypt', 'value':'Egypt'},
        {'label': 'El Salvador', 'value':'El Salvador'},
        {'label': 'Finland', 'value':'Finland'},
        {'label': 'France', 'value':'France'},
        {'label': 'Georgia', 'value':'Georgia'},
        {'label': 'Germany', 'value':'Germany'},
```

There is a better way!

Dictionary inside list comprehension

```
country_options = [{'label' : ctry, 'value' : ctry} for ctry in countries]
country_options
```

```
[{'label': 'Afghanistan', 'value': 'Afghanistan'},
 {'label': 'Argentina', 'value': 'Argentina'},
 {'label': 'Aruba', 'value': 'Aruba'},
 {'label': 'Australia', 'value': 'Australia'},
 {'label': 'Austria', 'value': 'Austria'},
```

List comprehension with `zip()`

```
country_labels = [c.upper() for c in countries]
country_options = [{'label' : lab,
                    'value' : ctry} for ctry,lab in zip(countries,
                                                        country_labels)]
country_options
```

```
[{'label': 'AFGHANISTAN', 'value': 'Afghanistan'},
 {'label': 'ARGENTINA', 'value': 'Argentina'},
 {'label': 'ARUBA', 'value': 'Aruba'},
 {'label': 'AUSTRALIA', 'value': 'Australia'},
 {'label': 'AUSTRIA', 'value': 'Austria'},
 {'label': 'BAHAMAS', 'value': 'Bahamas'},
 {'label': 'BANGLADESH', 'value': 'Bangladesh'},
```

Python: List, Dict, Set Comprehensions
R: Apply functions

Refactor code 2

Current

```
- app.py (big file)
- data
    |- dat.csv
    |- README
- ProcFile
- requirements.txt
- README
```

Option 1

```
- app.py
- index.py
- apps
    |-- __init__.py
    |-- app1.py
    |-- app2.py
- data
    |-- dat.csv
    |-- README
- ProcFile
- requirements.txt
- README
```

Structuring a Multi-page app in Python and R

Refactor code 2

Option 1

```
- app.py
- index.py
- apps
    |-- __init__.py
    |-- app1.py
    |-- app2.py
- data
    |-- dat.csv
    |-- README
- ProcFile
- requirements.txt
- README
```

Structuring a Multi-page app in Python and R

app.py

```python
import dash

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
server = app.server
app.config.suppress_callback_exceptions = True
```

Refactor code 2

Option 1

```
- app.py
- index.py
- apps
    |-- __init__.py
    |-- app1.py
    |-- app2.py
- data
    |-- dat.csv
    |-- README
- ProcFile
- requirements.txt
- README
```

Structuring a Multi-page app in Python and R

app.py

```
import dash

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
server = app.server
app.config.suppress_callback_exceptions = True
```

apps/app1.py

```
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output

from app import app

layout = html.Div([
    html.H3('App 1'),
    dcc.Dropdown(
```

Key: import app.py as a module

Refactor code 2

Option 1

```
- app.py
- index.py
- apps
    |-- __init__.py
    |-- app1.py
    |-- app2.py
- data
    |-- dat.csv
    |-- README
- ProcFile
- requirements.txt
- README
```

Structuring a Multi-page app in Python and R

app.py

```
import dash

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
server = app.server
app.config.suppress_callback_exceptions = True
```

apps/app1.py

```
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output

from app import app

layout = html.Div([
    html.H3('App 1'),
    dcc.Dropdown(
```

Key: import app.py as a module

index.py

index.py loads different apps on different urls

Refactor code 3

PEP8 for Python and Tidyverse for R

Use docstrings and comments to improve code

Python

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)
    """
    if imag == 0.0 and real == 0.0:
        return complex_zero
    ...
```

R

7.4 Documenting parameters

For most tags, like `@param`, `@seealso` and `@return`, the text should be a sentence, starting with a capital letter and ending with a full stop.

```
#' @param key The bare (unquoted) name of the column whose values will be used
#'   as column headings.
```

If some functions share parameters, you can use `@inheritParams` to avoid duplication of content in multiple places.

```
#' @inheritParams argument function_to_inherit_from
```

20 minutes

Part 3: Speeding up Dashboards

Many ways to speed up dashboards, here are three:

1. Filtering data
2. Strategies for aggregating data
3. Precomputing data

Filtering Data

- Are you using all of your data ?
    - If not, set up a reproducible script to drop unused columns, and rows…
    - If yes, add components (dropdowns/slider) to show less data overall

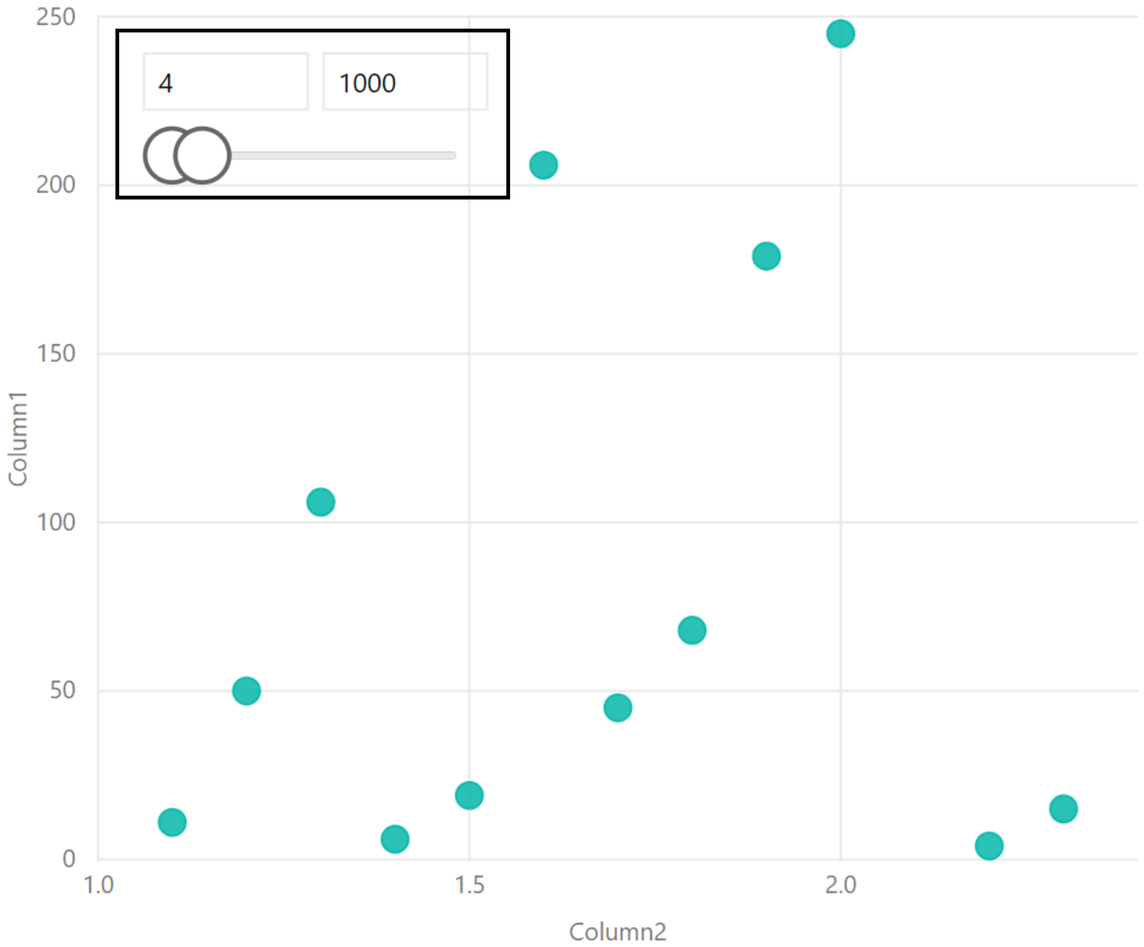
- Remember: purpose-driven dashboards. Is all your data useful to answer your research question?

Dealing with outliers

Source: DSCI 532 lecture slides from 2019/19 Cydney Nielsen

Dealing with outliers

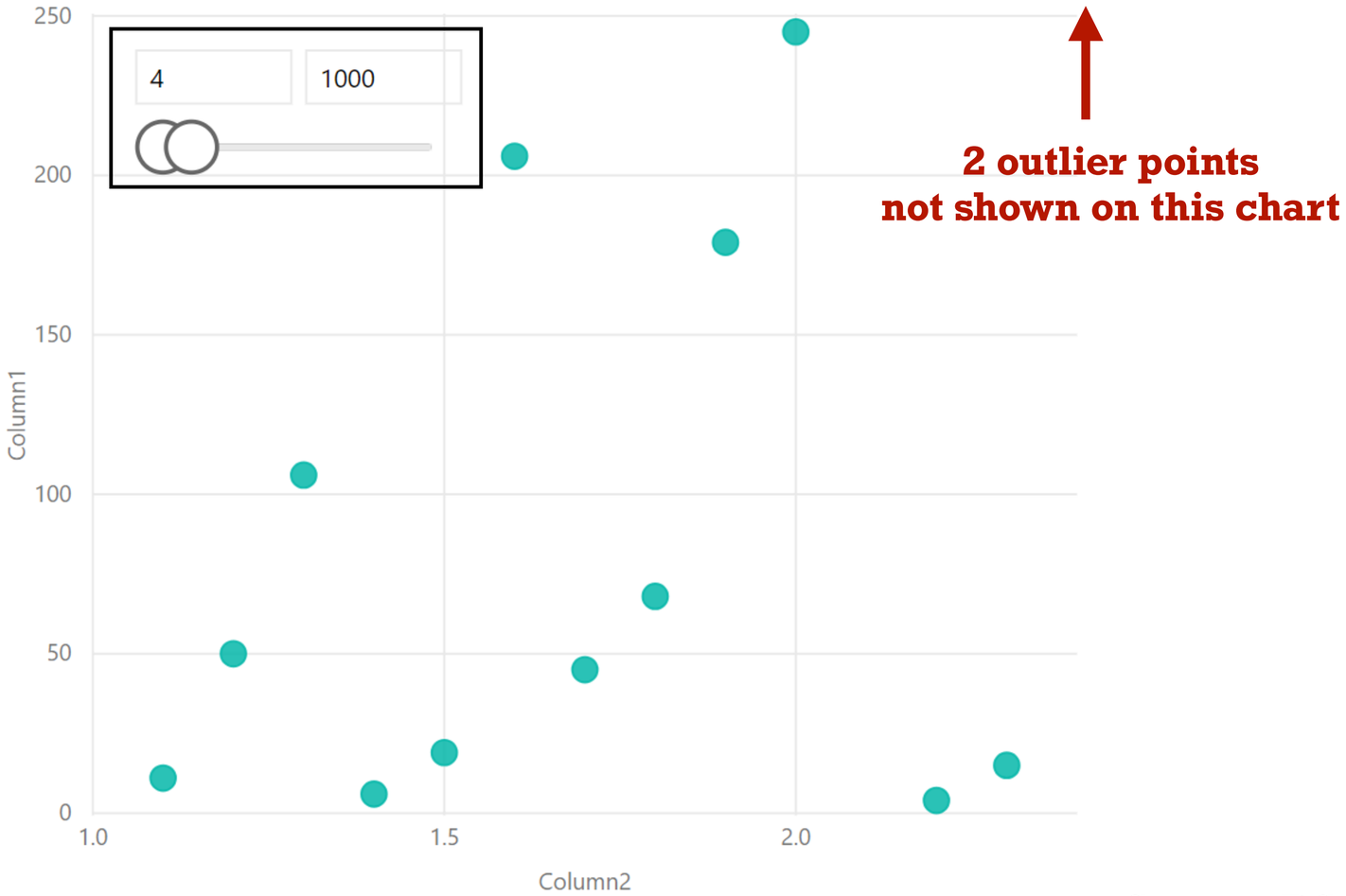
interactive filtering is another solution - *appropriate when want to remove outliers*



Source: DSCI 532 lecture slides from 2019/19
Cydney Nielsen

Dealing with outliers

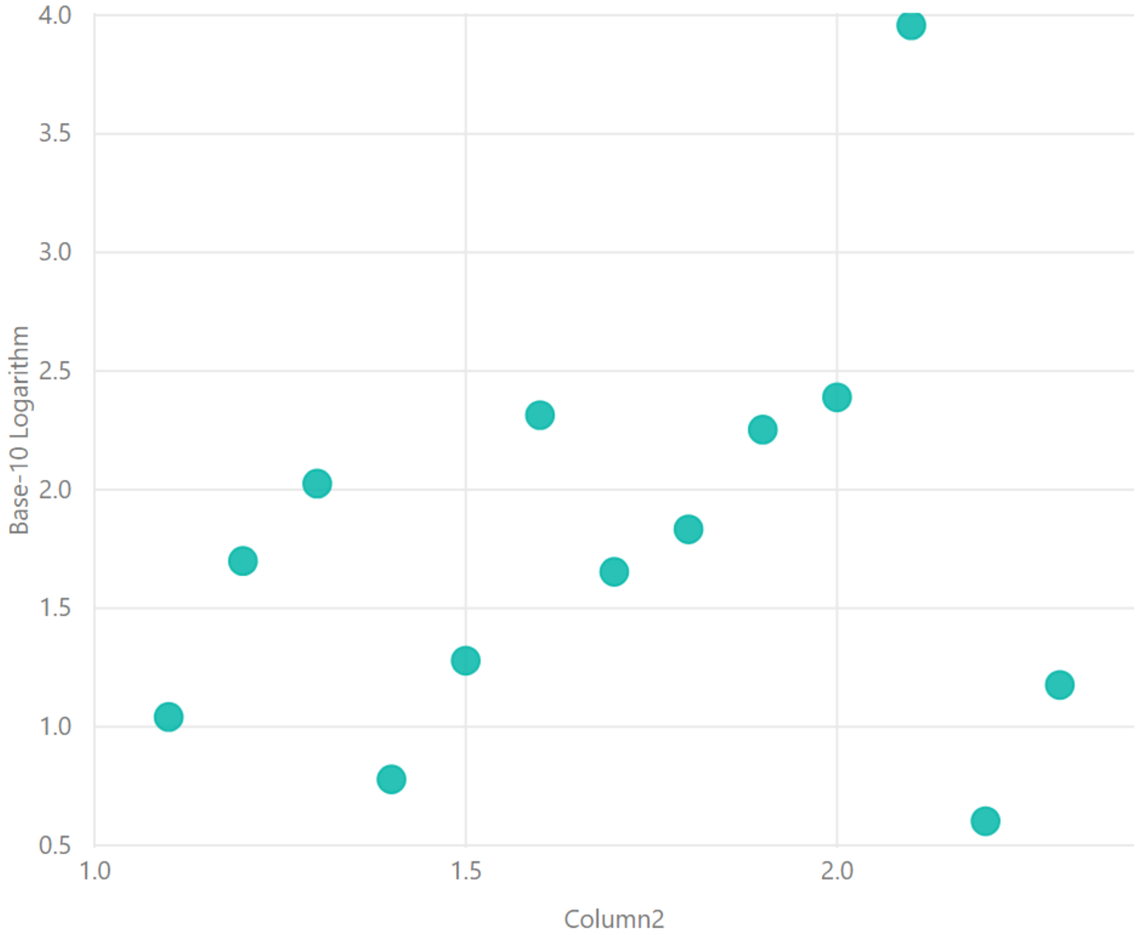
interactive filtering is another solution - *appropriate when want to remove outliers*



Source: DSCI 532 lecture slides from 2019/19
Cydney Nielsen

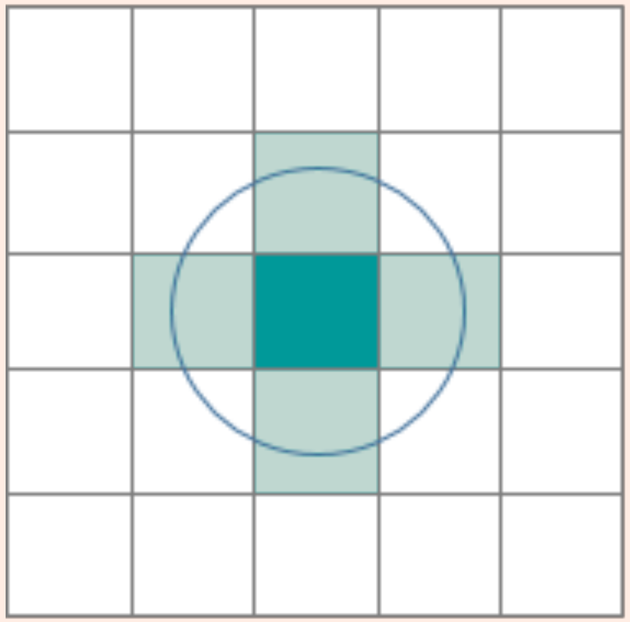
Dealing with outliers

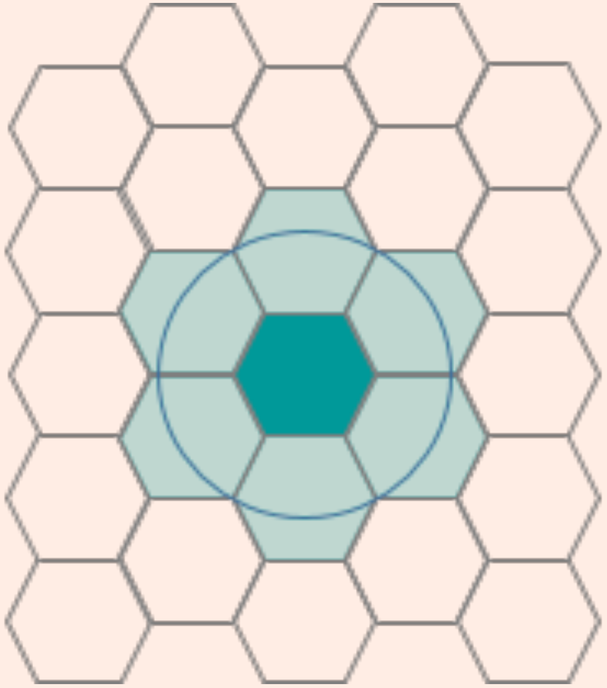
log scale helps - *appropriate if you want to keep all points*



Source: DSCI 532 lecture slides from 2019/19
Cydney Nielsen

Why Hexagons?





Hexagons:

- Reduce sampling bias

- Curves in data more naturally shown

- Grids draw our eyes to straight, unbroken parallel lines

Source: Many more arguments presented here ArcGIS Pro

Pre-computing Data

- Use vectorized functions and avoid loops at all costs!

- Setup a script/notebook to do any heavy wrangling and processing outside your app

- Load data at app start; **re-structure** your data to serve queries!

Many ways to speed up dashboards, here are three:

1. Filtering data
2. Strategies for aggregating data
3. Precomputing data (vectorize functions)
4. Compute additional columns in advance (separate from plotting)

Summary

- Dash callbacks in R

- Speed-up dashboards by precomputing, filtering, and aggregating data

- Use vectorized functions and packages like **purrr** and **dplyr**