# global functions with all necessary libraries

global = function() {

library(sf)

library(dplyr) # data wrangling

library(readr) # reading data

library(ggplot2) # data visualization

library(tidygraph)

library(shiny) # main shiny app package

library(bslib) # easier html construction

library(plotly) # interactive visuals

}

read\_data = function() {

# Get any helper data you need to starts

# 1) read county data

read.csv("./County.csv") %>%

saveRDS("county\_names.rds")

county\_names = read\_rds("./county\_names.rds") %>%

filter(state == "Florida") #Filtering to Florida data analysis only

county\_names$county <- as.integer(county\_names$county)

from\_county\_names = county\_names %>%

rename(

from\_countyid = county,

from\_county\_name = county\_name,

from\_state = state

)

to\_county\_names = county\_names %>%

rename(

to\_countyid = county,

to\_county\_name = county\_name,

to\_state = state

)

n = read\_rds("./dorian.rds") %>%

# Focus on just the nodes of the network

activate("nodes")

# 2) read nodes data

read\_rds("./dorian.rds") %>%

# Focus on just the nodes of the network

activate("nodes") %>%

# Turn it into a tibble/data.frame

as\_tibble() %>%

# Create a unique node id,

# which we'll use to join in geoid to the from and to columns of the edges

mutate(node = 1:n()) %>%

# Reorder the variables; dropping geometry

select(node, geoid, svi, pop, county, rainfall\_14days) %>%

# Save the nodes

saveRDS("nodes.rds")

nodes = read\_rds("./nodes.rds")

# Convert county column to integer

nodes$county <- as.integer(nodes$county)

# 3) read edges data

# Save the edges to file.

read\_rds("./dorian.rds") %>%

# Focus on just the edges of the network

activate("edges") %>%

# Turn it into a tibble/data.frame

as\_tibble() %>%

# Drop the geometry

# select(-geometry) %>%

# Let's join the source (from) geoid in using the shared node id

left\_join(by = c("from" = "node"),

y = nodes %>% select(node, from\_geoid = geoid)) %>%

# Let's join the destination (to) geoid in using the shared node id

left\_join(by = c("to" = "node"),

y = nodes %>% select(node, to\_geoid = geoid)) %>%

# Save this to file

saveRDS("edges.rds")

edges = read\_rds("./edges.rds")

edges %>%

mutate(from\_countyid = as.integer(substr(from\_geoid, 1, 5)),

to\_countyid = as.integer(substr(to\_geoid, 1, 5))) %>%

inner\_join(county\_names, by = c("from\_countyid" = "county")) %>%

rename(from\_county\_name = county\_name) %>%

inner\_join(county\_names, by = c("to\_countyid" = "county")) %>%

#Joins the county names table tother to match with the and from county ids

rename(to\_county\_name = county\_name) %>%

select(from:from\_county\_name, to\_county\_name, state = state.x) %>%

mutate(evac\_shel = if\_else(evacuation > 0, "Evacuated", "Sheltered")) %>%

# join from node

left\_join(

by = c("from" = "node"),

y = nodes %>% select(node, from\_geoid = geoid, svi, pop, rainfall\_14days)

) %>%

saveRDS("nodes\_n\_edges.rds")

}

### USER INTERFACE ####

ui = function() {

read\_data()

# Load the nodes in

nodes = read\_rds("./nodes.rds")

nodes\_n\_edges = read\_rds("./nodes\_n\_edges.rds")

nodes\_n\_edges %>%

# Get the day of analysis.

mutate(day = lubridate::day(date\_time)) %>%

# Get the month

mutate(month = lubridate::month(date\_time)) %>%

# Get the hour

mutate(hour = lubridate::hour(date\_time))

# Get unique counties

from\_counties <- unique(nodes\_n\_edges$from\_county\_name[order(nodes\_n\_edges$from\_county\_name)])

to\_counties <- unique(nodes\_n\_edges$to\_county\_name[order(nodes\_n\_edges$to\_county\_name)])

ev\_sh <- unique(nodes\_n\_edges$evac\_shel[order(nodes\_n\_edges$evac\_shel)])

# Make a named vector, so you can select by Names but get back specific ids, eg. "Dec" = 12

choices\_from\_county = setNames(object = from\_counties, nm = from\_counties)

choices\_to\_county = setNames(object = to\_counties, nm = to\_counties)

choices\_min\_date = min(nodes\_n\_edges$date\_time)

choices\_max\_date = max(nodes\_n\_edges$date\_time)

choices\_ev\_sh = setNames(object = ev\_sh, nm = ev\_sh)

# TITLE CARD ###################################

c1 = card(# Make a card header whose background is the primary color (class = bg-primary)

card\_header(

class = "bg-primary",

# Add this title

card\_title("Analysis of Hurricane Dorian Evacuation (2019)")

))

# SELECTOR CARD #################################

c2 = bslib::card(

# Make a simple card header and title

card\_header(card\_title("Filters")),

# Make a card body section

card\_body(

sliderInput(

"dateRange",

"Select Date Range:",

min = as.Date(choices\_min\_date),

max = as.Date(choices\_max\_date),

value = c(as.Date("2019-08-28"), as.Date("2019-09-7")),

timeFormat = "%Y-%m-%d"

)

),

card\_body(

selectInput(

inputId = "evac\_shel",

label = "Select Evacuated / Sheltered",

choices = choices\_ev\_sh,

selected = "Evacuated",

selectize = FALSE, size = 5

)

),

card\_body(

selectInput(

inputId = "to\_county\_name",

label = "Select Destination County",

choices = choices\_to\_county,

selected = sample(to\_counties, 1),

selectize = FALSE, size = 5

)

)

)

# PLOT CARD ##########################

c3 = bslib::layout\_column\_wrap(card(plotlyOutput(outputId = "plot\_one")), card(plotlyOutput(outputId = "plot\_two")), width = 0.5)

c4 = bslib::layout\_column\_wrap(card(plotlyOutput(outputId = "map\_one")),

card(plotlyOutput(outputId = "plot\_three")),

card(plotlyOutput(outputId = "plot\_four")),

card(plotlyOutput(outputId = "map\_two")),

width = 0.5)

# VALUE BOXES CARD ##########################

box1 = bslib::value\_box(

title = textOutput("box\_one\_title"),

value = textOutput("box\_one\_measure"),

class = "bg-primary text-light",

# add a fontawesome icon to showcase

showcase = shiny::icon("person-walking-arrow-right")

)

box2 = bslib::value\_box(

title = textOutput("box\_two\_title"),

value = textOutput("box\_two\_measure"),

class = "bg-primary text-light",

# add a fontawesome icon to showcase

showcase = shiny::icon("home")

)

box3 = bslib::value\_box(

title = textOutput("box\_three\_title"),

value = textOutput("box\_three\_measure"),

class = "bg-primary text-light",

# add a fontawesome icon to showcase

showcase = shiny::icon("plane")

)

box4 = bslib::value\_box(

title = textOutput("box\_four\_title"),

value = textOutput("box\_four\_measure"),

class = "bg-primary text-light",

# add a fontawesome icon to showcase

showcase = shiny::icon("cloud-rain")

)

# Bundle them together with a header

c5 = card(

# Add a header describing the Selections you made

card\_header(class = "bg-primary", card\_title(textOutput("text\_selection"))),

# Bundle the value boxes together

card\_body(layout\_column\_wrap(box1, box2, box3, box4)),

width = 1 / 4

)

# SPOTLIGHT BOX

c6 = bslib::card(

bslib::card\_header("Spotlight", class = "bg-dark"),

bslib::card\_footer(textOutput("page\_two\_spotlight")))

# Or add a sidebar-main split layout like this...

bslib::page(

title = "Hurricane Dorian",

# add a bootstrap theme to the page

theme = bslib::bs\_theme(preset = "cerulean"),

# Stack cards

c1,

# header

# Put next cards in a sidebar-main panel split layout

bslib::layout\_sidebar(

# Sidebar...

sidebar = bslib::sidebar(c2),

# main panel

c5,

# Make a series of panels we can click between

bslib::navset\_card\_pill(

selected = "plots",

# page one

bslib::nav\_panel(title = "Overview", value = "plots", c3),

# page two

bslib::nav\_panel(title = "Spatial Maps", value = "maps", c4, c6)

)

# c4 # text

)

)

}

#### SERVER DATA #####

server = function(input, output, session) {

# Load the nodes in

nodes = read\_rds("./nodes.rds")

nodes\_n\_edges = read\_rds("./nodes\_n\_edges.rds")

countygeo = read\_sf("./counties.geojson") #Mapping data

#Florida data filter

florida\_stat = reactive({

# Start by filtering the data based on the input county

nodes\_n\_edges %>%

mutate(date = as.Date(date\_time),

time = format(as.POSIXct(date\_time), format = "%H:%M:%S")) %>%

# Filter to just rows where county matches the user input

# filter(county\_name %in% input$county\_name)

filter(date > input$dateRange[1] &

date < input$dateRange[2]) # use side bar filter for date

# Trigger whenever input$origin changes

}) %>% bindEvent({

input$dateRange

})

#To County data filter

to\_county\_stat = reactive({

# Start by filtering the data based on the input county

nodes\_n\_edges %>%

mutate(date = as.Date(date\_time),

time = format(as.POSIXct(date\_time), format = "%H:%M:%S")) %>%

# Filter to just rows where county matches the user input

# filter(county\_name %in% input$county\_name)

filter(date > input$dateRange[1] &

date < input$dateRange[2]) %>% # use side bar filter for date

filter(to\_county\_name == input$to\_county\_name)

# Trigger whenever input$origin changes

}) %>% bindEvent({

list(input$dateRange, input$to\_county\_name)

})

#Nodes and Edges data filter

nodes\_n\_edges\_stat = reactive({

# Start by filtering the data based on the input county

nodes\_n\_edges %>%

mutate(date = as.Date(date\_time),

time = format(as.POSIXct(date\_time), format = "%H:%M:%S")) %>%

# Filter to just rows where county matches the user input

# filter(county\_name %in% input$county\_name)

filter(date > input$dateRange[1] &

date < input$dateRange[2]) %>% # use side bar filter for date

filter(evac\_shel == input$evac\_shel) %>%

filter(to\_county\_name == input$to\_county\_name)# use side bar filter for to\_county\_name

# Trigger whenever input$evac\_shel and input$to\_county\_name changes

}) %>% bindEvent({

list(input$dateRange, input$evac\_shel, input$to\_county\_name)

})

# Value Box Text ########################################

output$box\_one\_title = renderText({

paste("Total Number of Evacuatations to",

"\n",

input$to\_county\_name)

}) %>% bindEvent({

input$to\_county\_name

})

output$box\_one\_measure = renderText({

t\_ev <- to\_county\_stat() %>%

filter(evac\_shel == "Evacuated") %>%

summarize(total\_evacuations = sum(evacuation, na.rm = TRUE)) %>%

pull(total\_evacuations)

t\_ev

}) %>% bindEvent({

to\_county\_stat()

input$to\_county\_name

})

output$box\_two\_title = renderText({

paste("Total Number of Shelter-In-Place in",

"\n",

input$to\_county\_name)

}) %>% bindEvent({

input$to\_county\_name

})

output$box\_two\_measure = renderText({

t\_sh <- to\_county\_stat() %>%

filter(evac\_shel == "Sheltered") %>%

summarize(total\_evacuations = sum(evacuation, na.rm = TRUE)) %>%

pull(total\_evacuations)

t\_sh

}) %>% bindEvent({

to\_county\_stat()

input$to\_county\_name

})

output$box\_three\_title = renderText({

paste("Average Distance (km) Traveled to",

"\n",

input$to\_county\_name)

}) %>% bindEvent({

input$to\_county\_name

})

output$box\_three\_measure = renderText({

t\_tr <- to\_county\_stat() %>%

filter(evac\_shel == "Sheltered") %>%

summarize(avg\_distance\_traveled = mean(km, na.rm = TRUE)) %>%

pull(avg\_distance\_traveled)

t\_tr

}) %>% bindEvent({

to\_county\_stat()

input$to\_county\_name

})

output$box\_four\_title = renderText({

paste("Average Rainfall in L14 Days", "\n", input$to\_county\_name)

}) %>% bindEvent({

input$to\_county\_name

})

output$box\_four\_measure = renderText({

a\_rf <- nodes\_n\_edges\_stat() %>%

summarize(avg\_rainfall = mean(rainfall\_14days)) %>%

pull(avg\_rainfall)

a\_rf

}) %>% bindEvent({

nodes\_n\_edges\_stat()

input$to\_county\_name

})

output$text\_selection = renderText({

paste(

"Evacuation Status at",

input$to\_county\_name,

"Between",

input$dateRange[1],

"and",

input$dateRange[2]

)

}) %>% bindEvent({

input$to\_county\_name

})

# SPOTLIGHT TEXT IN SPATIAL TAB #####

page\_two\_spotlight\_stat = reactive({

# Format a number for highlighting

nodes\_n\_edges\_stat() %>%

summarize(avg\_distance\_traveled = mean(km, na.rm = TRUE)) %>%

mutate(avg\_distance\_traveled = scales::number(avg\_distance\_traveled, accuracy = 0.1)) %>%

mutate(text = paste(

"Between", input$dateRange[1], "and",input$dateRange[2] ,

"the average distance traveled to", input$from\_county\_name, "is", avg\_distance\_traveled, "km."))

}) %>% bindEvent({

nodes\_n\_edges\_stat()

list(input$dateRange, input$from\_county\_name)

})

## Render to text output 'text\_highlight'

output$page\_two\_spotlight = renderText({

# Output a single text blob value. Must have just length 1.

page\_two\_spotlight\_stat()$text

# Trigger whenever stat\_highlight() changes

}) %>% bindEvent({page\_two\_spotlight\_stat()})

### PLOTS ########################################

## PLOT ONE ##

#TOP 10 DESTINATIONS FOR EVACUATION WITHIN FLORIDA

output$plot\_one = renderPlotly({

plot\_one\_data = florida\_stat() %>% # use side bar filter for date

filter(evac\_shel == input$evac\_shel) %>% # use side bar filter EVAC OR SHELTERED

group\_by(from\_county\_name) %>%

summarise(

total\_evacuations = sum(evacuation, na.rm = TRUE),

avg\_distance\_traveled = mean(km, na.rm = TRUE)

) %>%

arrange(desc(total\_evacuations)) %>%

head(10)

gg\_plot\_one = ggplot(

data = plot\_one\_data,

mapping = aes(

x = reorder(from\_county\_name, total\_evacuations),

y = total\_evacuations,

fill = avg\_distance\_traveled

)

) +

geom\_col(size = 0.5) +

coord\_flip() +

theme\_bw() +

labs(

x = "Counties",

y = "Evacuated",

fill = "Distance Traveled (km)",

title = paste(

"Top 10 Places Individuals",

input$evac\_shel,

"\n",

"Between",

input$dateRange[1],

"and",

input$dateRange[2]

)

) +

theme(plot.title = element\_text(hjust = 0.5)) +

scale\_y\_continuous(

labels = function(x)

format(x, scientific = FALSE)

) +

scale\_fill\_gradient2(

low = "lightyellow",

mid = "turquoise",

high = "blue4",

midpoint = mean(plot\_one\_data$avg\_distance\_traveled, na.rm = TRUE)

)

# Make it plotly

pp\_plot\_one = plotly::ggplotly(gg\_plot\_one)

# return the visualization

pp\_plot\_one

# Trigger this plot to rerender when input$month changes

}) %>% bindEvent({

florida\_stat()

list(input$evac\_shel, input$dateRange)

})

## PLOT TWO ##

# AVERAGE EVACUATION OVER TIME (COUNTY VS STATE)

output$plot\_two = renderPlotly({

## DateTime Vs. Mean Number of Evacuation

plot\_two\_state\_data = florida\_stat() %>%

#sorts date range to examine hurricane

group\_by(date) %>%

summarize(AVG\_EVAC = mean(evacuation, na.rm = TRUE))

plot\_two\_to\_cty\_data = to\_county\_stat() %>%

#sorts date range to examine hurricane

group\_by(date) %>%

summarize(AVG\_EVAC = mean(evacuation, na.rm = TRUE))

gg\_plot\_two <- ggplot() +

geom\_line(

data = plot\_two\_state\_data,

aes(x = date, y = AVG\_EVAC, color = "State of Florida"),

size = 0.5,

alpha = 0.3,

) +

geom\_point(

data = plot\_two\_state\_data,

aes(

x = date,

y = AVG\_EVAC,

color = ifelse(AVG\_EVAC < 0, 'Below Zero', 'Above Zero')

#size = 0.1,

),

alpha = 0.3

) +

geom\_line(

data = plot\_two\_to\_cty\_data,

aes(

x = date,

y = AVG\_EVAC,

color = input$to\_county\_name

),

size = 0.5

) +

geom\_point(data = plot\_two\_to\_cty\_data, aes(

x = date,

y = AVG\_EVAC,

color = ifelse(AVG\_EVAC < 0, 'Below Zero', 'Above Zero')

#size = 0.1

)) +

geom\_hline(yintercept = 0,

linetype = "solid",

color = "black") +

scale\_color\_manual(

values = c(

"State of Florida" = "coral",

"Below Zero" = "darkturquoise",

"Above Zero" = "coral"

)

) +

scale\_x\_date(date\_breaks = "3 days", date\_labels = "%m-%d") +

scale\_y\_continuous(

labels = function(x)

format(x, scientific = FALSE)

) +

theme\_bw() +

labs(

x = "Date",

y = "Evacuations",

title = paste(

"Average Evacuations",

"Between",

input$dateRange[1],

"and",

input$dateRange[2]

)

) +

theme(

plot.title = element\_text(hjust = 0.5),

legend.title = element\_blank(),

axis.text.x = element\_text(angle = 45, hjust = 1)

)

# Make it plotly

pp\_plot\_two = plotly::ggplotly(gg\_plot\_two, tooltip = c("Date", "Evacuations"))

# return the visualization

pp\_plot\_two

# Trigger this plot to rerender when input$month changes

}) %>% bindEvent({

florida\_stat()

to\_county\_stat()

list(input$to\_county\_name, input$dateRange)

})

## MAP 1 ##

# DISTANCE TRAVELED TO CHOSEN COUNTIES (GEO LOCATION)

output$map\_one = renderPlotly({

florida\_map <- countygeo %>%

filter(state == "FL") %>%

mutate(geoid = as.integer(geoid))

fl\_evac\_from\_raw <- to\_county\_stat() %>%

filter(evac\_shel == "Evacuated") %>%

group\_by(from\_countyid, from\_county\_name) %>%

summarise(avg\_distance\_traveled = mean(km, na.rm = TRUE)) %>%

arrange(desc(avg\_distance\_traveled)) %>%

left\_join(florida\_map %>% select(geoid, geometry),

by = c("from\_countyid" = "geoid")) %>%

head(10) %>%

st\_as\_sf()

# Let's grab nodes from this network (cities / county-subdivisions)

data = read\_rds("./dorian.rds")

nodes\_data = data %>%

# Focus on just the nodes of the network

activate("nodes") %>%

# Turn it into a tibble/data.frame

as\_tibble() %>%

# Make it into a spatial data.frame

st\_as\_sf()

edges\_data = nodes\_n\_edges\_stat() %>%

filter(evac\_shel == "Evacuated") %>%

# Make it into a spatial data.frame

st\_as\_sf()

valid\_fl\_geo <- st\_is\_valid(fl\_evac\_from\_raw)

empty\_fl\_geo <- st\_is\_empty(fl\_evac\_from\_raw)

fl\_evac\_from <- fl\_evac\_from\_raw[valid\_fl\_geo & !empty\_fl\_geo, ]

centroids <- st\_centroid(fl\_evac\_from)

if (nrow(fl\_evac\_from) > 0) {

map\_one <- ggplot() +

geom\_sf(data = florida\_map,

fill = "transparent",

color = "black") +

geom\_sf(

data = fl\_evac\_from,

aes(geometry = geometry, fill = avg\_distance\_traveled),

color = "coral"

) + # Fill based on sum\_evac

geom\_sf(data = edges\_data, mapping = aes(alpha = nrow(edges\_data))) +

geom\_text(

data = centroids,

aes(

x = st\_coordinates(geometry)[, 1],

y = st\_coordinates(geometry)[, 2],

label = from\_county\_name

),

size = 3,

color = "black",

nudge\_y = 0.1

) +

scale\_fill\_gradient2(

low = "lightyellow",

mid = "turquoise",

high = "blue4",

midpoint = mean(fl\_evac\_from$avg\_distance\_traveled, na.rm = TRUE),

name = "Avg Distance Traveled"

) +

labs(

title = paste(

"Top 10 Origin Counties to",

input$to\_county\_name,

"\n",

"Between",

input$dateRange[1],

"and",

input$dateRange[2]

),

x = "Longitude",

y = "Latitude"

) +

theme\_minimal() +

theme(

legend.position = "right",

plot.title = element\_text(hjust = 0.5),

plot.subtitle = element\_text(hjust = 0.5)

)

} else {

map\_one <- ggplot() +

geom\_sf(data = florida\_map,

fill = "transparent",

color = "black") +

geom\_sf(data = edges\_data, mapping = aes(alpha = nrow(edges\_data))) +

labs(

title = paste("Geo Data is Not Available for", input$to\_county\_name),

x = "Longitude",

y = "Latitude"

) +

theme\_minimal() +

theme(

legend.position = "right",

plot.title = element\_text(hjust = 0.5),

plot.subtitle = element\_text(hjust = 0.5)

)

}

}) %>% bindEvent({

to\_county\_stat()

list(input$evac\_shel, input$to\_county\_name)

})

## PLOT 3 ##

# DISTANCE TRAVELED TO CHOSEN COUNTY (BAR PLOT)

output$plot\_three = renderPlotly({

# for specific county, total evacuation

plot\_three\_data = to\_county\_stat() %>%

filter(evac\_shel == input$evac\_shel) %>% # use side bar filter EVAC OR SHELTERED

group\_by(to\_county\_name, from\_county\_name) %>%

summarise(

total\_evacuations = sum(evacuation, na.rm = TRUE),

count = n(),

avg\_distance\_traveled = mean(km, na.rm = TRUE)

) %>%

arrange(desc(total\_evacuations))

gg\_plot\_three = ggplot(

data = plot\_three\_data,

mapping = aes(

x = reorder(from\_county\_name, total\_evacuations),

y = total\_evacuations,

fill = avg\_distance\_traveled

)

) +

geom\_col(size = 2) +

coord\_flip() +

theme\_minimal() +

labs(

x = "From Counties",

y = "Evacuated",

fill = "Distance Traveled (km)",

title = paste(

"County Individuals",

input$evac\_shel,

"to:",

input$to\_county\_name

)

) +

theme(plot.title = element\_text(hjust = 0.5),

axis.text.x = element\_text(angle = 45, hjust = 1)) +

# legend.position = "none") +

scale\_y\_log10(

labels = function(x)

format(x, scientific = FALSE)

) +

scale\_fill\_gradient2(

low = "lightyellow",

mid = "turquoise",

high = "blue4",

midpoint = mean(plot\_three\_data$avg\_distance\_traveled, na.rm = TRUE)

)

# Make it plotly

pp\_plot\_three = plotly::ggplotly(gg\_plot\_three)

# return the visualization

pp\_plot\_three

# Trigger this plot to rerender when input$month changes

}) %>% bindEvent({

to\_county\_stat()

list(input$evac\_shel, input$to\_county\_name)

})

## MAP 2 ##

# AVERAGE SVI PER ORIGIN COUNTIES

output$map\_two = renderPlotly({

florida\_map <- countygeo %>%

filter(state == "FL") %>%

mutate(geoid = as.integer(geoid))

fl\_evac\_from\_raw <- to\_county\_stat() %>%

filter(evac\_shel == "Evacuated") %>%

group\_by(from\_countyid, from\_county\_name) %>%

summarise(avg\_svi = mean(svi, na.rm = TRUE)) %>%

arrange(desc(avg\_svi)) %>%

left\_join(florida\_map %>% select(geoid, geometry),

by = c("from\_countyid" = "geoid")) %>%

head(10) %>%

st\_as\_sf()

# Let's grab nodes from this network (cities / county-subdivisions)

data = read\_rds("./dorian.rds")

nodes\_data = data %>%

# Focus on just the nodes of the network

activate("nodes") %>%

# Turn it into a tibble/data.frame

as\_tibble() %>%

# Make it into a spatial data.frame

st\_as\_sf()

edges\_data = nodes\_n\_edges\_stat() %>%

filter(evac\_shel == "Evacuated") %>%

# Make it into a spatial data.frame

st\_as\_sf()

valid\_fl\_geo <- st\_is\_valid(fl\_evac\_from\_raw)

empty\_fl\_geo <- st\_is\_empty(fl\_evac\_from\_raw)

fl\_evac\_from <- fl\_evac\_from\_raw[valid\_fl\_geo & !empty\_fl\_geo, ]

centroids <- st\_centroid(fl\_evac\_from)

if (nrow(fl\_evac\_from) > 0) {

map\_one <- ggplot() +

geom\_sf(data = florida\_map,

fill = "transparent",

color = "black") +

geom\_sf(

data = fl\_evac\_from,

aes(geometry = geometry, fill = avg\_svi),

color = "coral"

) + # Fill based on sum\_evac

geom\_sf(data = edges\_data, mapping = aes(alpha = nrow(edges\_data))) +

geom\_text(

data = centroids,

aes(

x = st\_coordinates(geometry)[, 1],

y = st\_coordinates(geometry)[, 2],

label = from\_county\_name

),

size = 3,

color = "black",

nudge\_y = 0.1

) +

scale\_fill\_gradient2(

low = "lightyellow",

mid = "orange",

high = "red3",

midpoint = mean(fl\_evac\_from$avg\_svi, na.rm = TRUE),

name = "Avg SVI"

) +

labs(

title = paste(

"Avg Social Vulnerability Index of Origin Counties to",

input$to\_county\_name,

"\n",

"Between",

input$dateRange[1],

"and",

input$dateRange[2]

),

x = "Longitude",

y = "Latitude"

) +

theme\_minimal() +

theme(

legend.position = "right",

plot.title = element\_text(hjust = 0.5),

plot.subtitle = element\_text(hjust = 0.5)

)

} else {

map\_one <- ggplot() +

geom\_sf(data = florida\_map,

fill = "transparent",

color = "black") +

geom\_sf(data = edges\_data, mapping = aes(alpha = nrow(edges\_data))) +

labs(

title = paste("Geo Data is Not Available for", input$to\_county\_name),

x = "Longitude",

y = "Latitude"

) +

theme\_minimal() +

theme(

legend.position = "right",

plot.title = element\_text(hjust = 0.5),

plot.subtitle = element\_text(hjust = 0.5)

)

}

}) %>% bindEvent({

to\_county\_stat()

list(input$evac\_shel, input$to\_county\_name)

})

## PLOT 4 ##

# AVERGAE RAINFALL IN ORIGIN COUNTIES

output$plot\_four = renderPlotly({

# for specific county, total evacuation

plot\_four\_data = to\_county\_stat() %>%

filter(evac\_shel == input$evac\_shel) %>% # use side bar filter EVAC OR SHELTERED

group\_by(to\_county\_name, from\_county\_name) %>%

summarise(

avg\_rainfall = mean(rainfall\_14days, na.rm = TRUE),

count = n(),

avg\_distance\_traveled = mean(km, na.rm = TRUE)

) %>%

arrange(desc(avg\_distance\_traveled))

gg\_plot\_four = ggplot(

data = plot\_four\_data,

mapping = aes(

x = reorder(from\_county\_name, avg\_distance\_traveled),

y = avg\_distance\_traveled,

fill = avg\_rainfall

)

) +

geom\_col(size = 2) +

coord\_flip() +

theme\_minimal() +

labs(

x = "From Counties",

y = "Average Distance Traveled (km)",

fill = "Average Rainfall in Last 14 Days",

title = paste(

"County Individuals",

input$evac\_shel,

"to:",

input$to\_county\_name

)

) +

theme(plot.title = element\_text(hjust = 0.5),

axis.text.x = element\_text(angle = 45, hjust = 1)) +

# legend.position = "none") +

scale\_y\_log10(

labels = function(x)

format(x, scientific = FALSE)

) +

scale\_fill\_gradient2(

low = "lightyellow",

mid = "orange",

high = "red3",

midpoint = mean(plot\_four\_data$avg\_rainfall, na.rm = TRUE)

)

# Make it plotly

pp\_plot\_four = plotly::ggplotly(gg\_plot\_four)

# return the visualization

pp\_plot\_four

# Trigger this plot to rerender when input$month changes

}) %>% bindEvent({

to\_county\_stat()

list(input$evac\_shel, input$to\_county\_name)

})

}

# Run app

shiny::shinyApp(ui = ui,

server = server,

onStart = global)