Interactive Map Walkthrough

Nate Ferraro

Summer 2020

Part 1: Data Merge Files

Goal: Make a dataset that can be easily updated with new files and combined based on column criteria

Create a folder of all datasets you wants to pull from.

- The idea is that you want to pull only the cleaned data from the each excel doc.
- **MAKE SURE THAT YOU ARE PULLING THE RIGHT SHEET.**
- Set correct working directory for where all datasets are held.

For example,

```
setwd("C:/Examples/Datasets/")
```

I would recommend holding all datasets in one place.

Some of the datasets needed to have the columns recoded so accurately pull the correct varaibles (there was a lack of naming continunity throughout different iterations)

```
#Example Recode Columns
September_2018 <-read_excel("7.REACH_YEM_Dataset_Joint Market Monitoring Initiative (JMMI)_Septe
mber2018_recode_cols.xlsx", sheet=2)

#Example
May_2020 <-read_excel("27.REACH_YEM_Dataset_Joint Market Monitoring Initiative (JMMI)_May2020.xl
sx", sheet = 3)
```

- Make sure that you are pulling the right sheet, it should be the cleaned data. The cleaned data will be analyzed and compiled into datasets.

```
list_df = setNames(lapply(ls(), function(x) get(x)), ls())
list_df_names <- names(list_df)
```

- This pulls all names in your environment into a list *(environment should be empty before adding all the names of all data sets)*, we will use this for an lapply later, and basically run analysis and cut down each dataset to what we need.

```
col_name_initial<-c("fuel_gov_origin","wash_gov_origin",colnames(April_2018%>% dplyr::select(sta
rts_with('calc_price_'),contains("cost_cubic_metere"), contains("exchange_rate_result"),starts_w
ith("governorate_"),starts_with("district_"),-contains('market'))))
data_all_JMMI<-as_tibble(data.frame(test="TEST"))

data_all_JMMI[,col_name_initial] <- NA

colnames_pulled_all<-as_tibble(data.frame(JMMI="TEST"))
```

- For the first part of this code we want to make a blank dataset that we will populate later with the data from the lapply, right now we are just working to set the column names correctly and populate it with the “TEST” and will be a single column for now
- Next we want to just change that to JMMI instead of test, probably an easier way to do this but I was lazy and just did it this way
- So we have a list of the column names we want to pull from our loaded dataset, and tibble we will pull everything into, and now just need to create the function to put it all together

```
name_object<-function(df){
  name<-deparse(substitute((df)))
  return(name)
}

round_df <- function(df, digits) {
  nums <- vapply(df, is.numeric, FUN.VALUE = logical(1))

  df[,nums] <- round(df[,nums], digits = digits)

  (df)
}
```

- These are functions that I threw in.
- The name object worked to dynamically name things in the script before the lapply
- Round df is super useful and will use for the sheets that come out in the end

```
col_pull<-function(df, list_of_df){
  #name<- names(list_of_df[df])

  call1 <-  sys.call(1)
  call1[[1]] <- as.name("names")
  call1 <- call1[1:2]
  nm <- eval(call1)
  name<-nm[!is.na(match(list_of_df,list(df)))]

  colnames(df)<-tolower(colnames(df))
  colnames(df)<-gsub("_all","",colnames(df))

  df1<-df%>%
    as_tibble()%>%
    dplyr::select(starts_with('calc_price_'),contains("cost_cubic_meter"), contains("exchange_ra
te_result"),starts_with("governorate_"),starts_with("district_"),starts_with("fuel_gov_origin"),
starts_with("wash_gov_origin"))%>%
    #rename(replace=c(colnames(df)=( gsub("_normalized", "_normalised", colnames(df) ) )  ))%>%
    mutate(as_tibble(),jmmi = name)%>%
    map_if(is.factor,as.character)%>%
    as_tibble()

      #colnames(df1)<-gsub(".*/","",colnames(df1))

    colname_pull<-as_tibble(data.frame( holder = colnames(df1)))
    names(colname_pull)<-name
    colnames_pulled_all<<-as_tibble(rowr::cbind.fill(colnames_pulled_all, colname_pull, fill=NA
))

    data_all_JMMI <<- as_tibble(merge(df1,data_all_JMMI,all=T))

  #return(data_all_JMMI)
}
```

Really important function

- First part is pulling the name from the file we are currently looking at so “name” at the end is the name of the file we are pulling things out of, for example “May_2019”, we will use this as a mutate later to track which observation go for to each data collection period.
- Next we make sure all column names are in lower case and all of the "_all" suffixes are taken from the column names. *(We want the column names as uniform as possible so that we can find them easily)*
- We create a “df1” which is pulling all the columns we need as well as mutating a column for telling us which JMMI this data is from, thus the need for pulling the correct name before.
- We also make sure that all factors are characters in this step, factors are the worst in this case.
- What really is key here is the <<- which put merges the df1 into the “data_all_JMMI”, the << act as a way to push the data to a dataset outside of the function in the environment meaning that it continually saves it, so will be merged all the dataset together in a quick way

```
lapply((list_df), col_pull, list_of_df = list_df)
```

- The lapply is run on the list of data frames we created in the first part of this, so are telling R to go back and run through each of the data frames on the list that are stored in the local environment, the data frames are

then run through the above function, with the list_df being specificed in the lapply as a condition for the col_pull function we just created.
- This lapply can take a while so be patient.

```
toMatch <- c("calc" ,"exchange","cost")
col_to_numeric<-unique(grep(paste(toMatch, collapse = "|"), colnames(data_all_JMMI),value = T))
data_all_JMMI[col_to_numeric] <- sapply(data_all_JMMI[col_to_numeric], as.numeric)
```

- Make sure that all of the columns that get pulled are numeric

```
#substitute out the pcodes to standardize the name (taken from JMMI scripting, with csv (utf-8)
sheet)
this_script_path<-(dirname(rstudioapi::getActiveDocumentContext()$path))
setwd(this_script_path)
source("./other scripts/add_pcodes.R")

#debug(add.pcodes)
data_all_JMMI<-add.pcodes(data_all_JMMI)
```

- Get the names of locations and governorates

```
#change Pcodes for origin governorates
source("./other scripts/gov_code_switch.R")
data_all_JMMI$fuel_gov_origin<-gov_code_switch(data_all_JMMI$fuel_gov_origin)
data_all_JMMI$wash_gov_origin<-gov_code_switch(data_all_JMMI$wash_gov_origin)

#pull in the full modes script
source("./other scripts/full_modes.R")
```

- This was an attempt to look at the restocking times, it failed, please disregard, will look back into it later.

#####Start Data Analysis#####

```
#get rid of all districts that have less than three observation
data_all_JMMI <- data_all_JMMI %>%
                        dplyr::group_by(jmmi)%>%
                        dplyr::group_by(district_id)%>%
                        filter(n()>2)%>%
                        as_tibble()

#make the JMMI column act as a date column and begin to sort by that, will be important for when
you want to do that national by the previous month
data_all_JMMI$jmmi<-gsub("_"," ", data_all_JMMI$jmmi)
#this is the actual date that will be sorted by with in the server script
data_all_JMMI$jmmi_date <- as.character(as.Date(as.yearmon(as.character(data_all_JMMI$jmmi))))
date_list<-sort(unique(data_all_JMMI$jmmi_date))
```

- We want to hold our methodology that all datasets have atleast 3 districts in each data collection time period.
- The dates need to be cleaned up as we will use it to sort the data for future analysis, we use the zoo package for the as.yearmon function (really useful but can be tough sometimes to use)
-

```
data_all_JMMI$country_id<-"YE"

for(i in seq_along(date_list)){
  if (i ==1){
    df1<-data_all_JMMI%>%
      filter(jmmi_date==date_list[i])


    district_all<-df1%>%
      aggregate_median("district_id")

    district_obs<-df1%>%
      dplyr::select("district_id","jmmi","jmmi_date")%>%
      dplyr::count(district_id, jmmi)

    governorate_all<-district_all%>%
      aggregate_median("governorate_id")

    governorate_obs<-df1%>%
      dplyr::select("governorate_id","jmmi","jmmi_date")%>%
      dplyr::count(governorate_id, jmmi)

    national_all<-governorate_all%>%
      aggregate_median("country_id")

    national_obs<-df1%>%
      dplyr::select("country_id","jmmi","jmmi_date")%>%
      dplyr::count(country_id, jmmi)


  }else{
    df1<-data_all_JMMI%>%
      filter(jmmi_date==date_list[i])

    df0<-data_all_JMMI%>%
      filter(jmmi_date==date_list[i-1])

    df0_pull<-unique(df0$district_id)
    df_dist<-subset(df1, district_id %in% df0_pull)

    district_all_alone<-df_dist%>%
      aggregate_median("district_id")

    district_all<-bind_rows(district_all_alone,district_all)

    district_obs<-df1%>%
      dplyr::select("district_id","jmmi","jmmi_date")%>%
      dplyr::count(district_id,jmmi)%>%
      bind_rows(district_obs)

    governorate_all_alone<-district_all_alone%>%
      aggregate_median("governorate_id")
```

```
    governorate_all<-bind_rows(governorate_all_alone,governorate_all)

    governorate_obs<-df1%>%
      dplyr::select("governorate_id","jmmi","jmmi_date")%>%
      dplyr::count(governorate_id,jmmi)%>%
      bind_rows(governorate_obs)

    national_all<-governorate_all_alone%>%
      aggregate_median("country_id")%>%
      bind_rows(national_all)

    national_obs<-df1%>%
      dplyr::select("country_id","jmmi","jmmi_date")%>%
      dplyr::count(country_id, jmmi)%>%
      bind_rows(national_obs)

    print(date_list[i])
  }
}
```

- This is a massive for loop that runs the analysis for the medians and binds them to the larger datasets.
- The construction of the for loop has two sections is the first part is whether the dataset is the first possible option, this is because you need something to populate the datasets with. Can't combine the lists together without the initial list, so need to use the merge tool with two datasets
- Will filter based on the order of the list, which should be set by the sort and the yearmon from the zoo package, so should be going in some type of order.
- The reason that we are using this order is due to the methodology of the JMMI, we only try to assess districts that we assessed in the previous round, so it is important to go in order. (This is we have the subset in the second part of the for loop "df_dist<-subset(df1, district_id %in% df0_pull)")

```
district_final<-dplyr::full_join(district_all,district_obs, by = c("district_id", "jmmi"))
governorate_final<-dplyr::full_join(governorate_all,governorate_obs, by = c("governorate_id", "j
mmi"))
national_final<-dplyr::full_join(national_all,national_obs, by = c("country_id", "jmmi"))
```

- Here we want to know how many observation we did at each district/governorate/national level for each time period, so we are doing to use the full join by the level and data collection period to make it work.

```
district_final<-district_final[,c("jmmi_date","governorate_name","governorate_id","district_nam
e","district_id","calc_price_petrol","calc_price_diesel","calc_price_bottled_water","calc_price_
treated_water","calc_price_soap","calc_price_laundry","calc_price_sanitary","cost_cubic_mete
r","exchange_rate_result","n")]
colnames(district_final)<-c("date","government_name","government_ID","district_name","district_I
D","petrol","diesel","bottled_water","treated_water","soap","laundry_powder","sanitary_napkin
s","cost_cubic_meter","exchange_rates","num_obs")

governorate_final<-governorate_final[,c("jmmi_date","governorate_name","governorate_id","calc_pr
ice_petrol","calc_price_diesel","calc_price_bottled_water","calc_price_treated_water","calc_pric
e_soap","calc_price_laundry","calc_price_sanitary","cost_cubic_meter","exchange_rate_resul
t","n")]
colnames(governorate_final)<-c("date","government_name","government_ID","petrol","diesel","bottl
ed_water","treated_water","soap","laundry_powder","sanitary_napkins","cost_cubic_meter","exchang
e_rates","num_obs")

national_final<-national_final[,c("jmmi_date","calc_price_petrol","calc_price_diesel","calc_pric
e_bottled_water","calc_price_treated_water","calc_price_soap","calc_price_laundry","calc_price_s
anitary","cost_cubic_meter","exchange_rate_result","n")]
colnames(national_final)<-c("date","petrol","diesel","bottled_water","treated_water","soap","lau
ndry_powder","sanitary_napkins","cost_cubic_meter","exchange_rates","num_obs")
```

- We want to rename the columns so they interact exactly how we want with the codes in the global, server, and ui for the interactive map.

```
final_list<-list("District"=district_final,"Governorate" = governorate_final, "National" = natio
nal_final)

this_script_path<-(dirname(rstudioapi::getActiveDocumentContext()$path))
setwd(this_script_path)

write.xlsx(final_list, file = "./data/updated_interactive.xlsx")
```

- We are writing a list of the data frames we have just made and renamed.
- Set a path to a data folder withing the interactive map files so that we can pull the data directly and keep it contained moving forward (really its was just easier to do it that way than trying to write to googlesheets or something)
- Also it self updates each time so you only have to run it and not do anything else

```
#undo after all clear
write.csv(district_final, file = "./data/district_interactive.csv")
write.csv(governorate_final, file = "./data/governorate_interactive.csv")
write.csv(national_final, file = "./data/national_interactive.csv")
```

- Just added this at the end

Recap:

Overall the data merge file was designed to help make data consolidation easy and painless. You will want to just link into your dataset each new time and make sure it is on the right sheet of the cleaned data. Ideally, this should just go when you add a line you run it and all the data for the map is updated.

Example of lines to add

```
Month_Year <-read_excel("dataset.xlsx", sheet = cleaned_data)
```

Final notes:

1. Make sure the columns you want to pull are numerics
2. Always check file paths and make sure the script is saved in the right spot
3. Spot check after running, R can be the worst sometimes, if there is an issue it will most likely be in the for_loop
4. If you want to add new data columns begin by updating the “col_pull” function, that where the inital pulls start, you will also need to probably adjust the “toMatch” variable, and the column names at the end.

Part 2: Global Files

This will set the global environment and combine the map shapefiles and the dataset that we just created in the data merge file. *This is why we saved the data merge file in the same folder structure, so that it is easy to grab*

There are 2 initial function at the beginning of the script are very important

1. The legend decreasing function reverses the lengend to allow for a respositioning on the map
2. The round df, great quick function to rounding all numerics in a dataset

```
GSh<-read.csv("data/governorate_interactive.csv")%>%
  as_tibble()%>%
  dplyr::select(-X)

  Admin1data <- mutate(GSh, SMEB = as.numeric((soap*10.5+laundry_powder*20+sanitary_napkins*2+a
s.numeric(cost_cubic_meter)*3.15))) #The SMEB calculation
  Admin1data$SMEB<-round(Admin1data$SMEB,0)


GSh2<-read.csv("data/district_interactive.csv")%>%
  as_tibble()%>%
  dplyr::select(-X)

  Admin2data <- mutate(GSh2, SMEB = as.numeric((soap*10.5+laundry_powder*20+sanitary_napkins*2+a
s.numeric(cost_cubic_meter)*3.15))) #The SMEB caluclation
  Admin2data$SMEB<-round(Admin2data$SMEB,0)

GShnat<-read.csv("data/national_interactive.csv")%>%
  as_tibble()%>%
  dplyr::select(-X)

  AdminNatData<-mutate(GShnat,SMEB = as.numeric((soap*10.5+laundry_powder*20+sanitary_napkins*2+
as.numeric(cost_cubic_meter)*3.15))) #The SMEB caluclation)
  AdminNatData$SMEB<-round(AdminNatData$SMEB,0)

max_date <- max(as.Date(as.yearmon(AdminNatData$date)))
```

So from the this bit of code we are pulling in the datasets that we made previously in the data merge file and building on them

- The reason I put the SMEB calculation here is incase it changes it will be easy to just do in one place. (*so as the SMEB calc changes be sure to update or add it here*).
- The other thing is to use the round_df function to keep all data uniform moving forward.

```
#Wrangle Data into appropriate formats
#Governorates
Admin1table<-as.data.frame(Admin1data)
Admin1table$date2<- as.Date(Admin1table$date, format("%d-%b-%y"), tz="UTC")
Admin1table$date2 <- as.Date(as.yearmon(Admin1table$date))

Admin1data_current <- Admin1table %>% #subset only recent month dates to attach to shapefile
  arrange(desc(date2)) %>%
  filter(date2 == max_date)
currentD <- as.character(format(max(Admin1table$date2),"%B %Y")) #define current date for disply
in dashboard
Admin1table[4:14] <- sapply(Admin1table[4:14], as.numeric)

#Districts
Admin2table <- as.data.frame(Admin2data)
Admin2table$date2 <- as.Date(as.yearmon(Admin2table$date))

Admin2data_current <- Admin2table %>% #subset only recent month dates to attach to shapefile
  arrange(desc(date2))%>%
  filter(date2 == max_date)
currentD <- as.character(format(max(Admin2table$date2),"%B %Y")) #define current date for disply
in dashboard
Admin2table[7:16] <- sapply(Admin2table[7:16], as.numeric)

#National
AdminNatTable<-as.data.frame(AdminNatData)
AdminNatTable$date2 <- as.Date(as.yearmon(AdminNatTable$date))

AdminNatData_current <- AdminNatTable %>% #subset only recent month dates to attach to shapefile
  arrange(desc(date2)) %>%
  filter(date2 == max_date)
currentD <- as.character(format(max(AdminNatTable$date2),"%B %Y"))
  #define current date for disply in dashboard
```

This part of the code tries to figure out the dates. The dates are always tricky in R so we need to be really careful with the, as you can see I was working to get dates into a unified format and create a way both sort and have markers for what data collection time period it came from.

If you add more columns and numbers you will have to adapt the sapply parts of this script to encompass all parts you want to look at.

```
Admin1<- readOGR("./www", "YEM_adm1_Governorates")
Admin2<- readOGR("./www", "YEM_adm2_Districts")

Admin1@data$admin1name<-gsub("Amanat Al Asimah", "Sana'a City", Admin1@data$admin1name)
Admin1@data$admin1refn<-gsub("Amanat Al Asimah", "Sana'a City", Admin1@data$admin1refn)
Admin2@data$admin1name<-gsub("Amanat Al Asimah", "Sana'a City", Admin2@data$admin1name)
Admin2@data<- Admin2@data %>% mutate_if(is.factor, as.character)
```

We pull in the actual shapefiles from the (wwww) folder and use readOGR to create objects we can then adapt for future needs.

We needed to rename Amanat Al Asimah as Sana'a City

```
##------------------------- COMBINE TABULAR & SPATIAL DATA----------------------
#Merge data from Google Sheet with Rayon shp file
Rshp <- merge(x=Admin2,y=Admin2data_current, by.x="admin2pcod", by.y= "district_ID")

DistsNumb<-sum(!is.na(Rshp@data$district_name)) #get number of districts covered...

Rshp<-st_simplify(st_as_sf(Rshp), dTolerance = 0.5)
Rshp <- st_transform(x = Rshp,
                     crs = "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0")
Rshp<-as(Rshp,"Spatial")
```

This is an important bit of code because we are merging the spatial data with our dataset.

- The Rshp file has to go with Admin2 from before and the Admin2data_current *(data Admin 2 is the district level)* it is really important to update and make sure by.x and by.y to be the same data and have overlap.
- The DistsNumb will be important and just update how many districts were in the new dataset.
- The st_simplify and st_transform are used to simplify and get into the correct projection of WGS84
- The final piece of the code is just to make sure that Rshp is a spatial dataset

```
Admin1<-st_simplify(st_as_sf(Admin1), dTolerance = 0.5)
Admin1<- st_transform(x = Admin1,
                      crs = "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0")
Admin1<-as(Admin1,"Spatial")
```

This bit of code preforms the same actions on the Admin 1 of governorate level of data. *is less important because we add the data at the district level, but this will allow for projection of Governorates overtop of our map*

```
# Get polygons centroids
centroids <- as.data.frame(centroid(Admin1))
colnames(centroids) <- c("lon", "lat")
centroids <- data.frame("ID" = 1:nrow(centroids), centroids)

# Create SpatialPointsDataFrame object
coordinates(centroids) <- c("lon", "lat")
proj4string(centroids) <- sp::proj4string(Admin1) # assign projection
centroids@data <- sp::over(x = centroids, y = Admin1, returnList = FALSE)
centroids1 <- as.data.frame(centroid(Admin1))
colnames(centroids1) <- c("lon", "lat")
centroids@data<- cbind(centroids@data, centroids1)
```

The centriods allow for the labels to be attached at the correct long and lat of the shapefiles and create the spatial points dataframe for the objects

```
#YEMEN LABEL
YEMl<- as.data.frame(cbind(48.5164,15.5527))
colnames(YEMl) <- c("lon", "lat")
YEMl <- data.frame("ID" = 1:nrow(YEMl), YEMl)
coordinates(YEMl) <- c("lon", "lat")
proj4string(YEMl) <- proj4string(Admin1)
UKRl1<- as.data.frame(cbind(48.5164,15.5527))
YEMl@data<-cbind(YEMl@data, "YEMEN", UKRl1 )
colnames(YEMl@data) <- c("index","name","lon", "lat")
```

Will create the labels for the Yemen polygon

```
vars <- c(
  "WASH SMEB"="SMEB",
  "Parallel Exchange Rates"="exchange_rates",
  "Petrol" = "petrol",
  "Diesel" = "diesel",
  "Bottled Water"="bottled_water",
  "Treated Water"="treated_water",
  "Soap"="soap",
  "Laundry Powder"="laundry_powder",
  "Sanitary Napkins"="sanitary_napkins",
  "Water Trucking"= "cost_cubic_meter"
)
```

Will list out the variables we need for the choice options, will need to be updated is additional variables are added.

Final notes:

1. The Dates have always given me a problem on this script, if there is an issues in the dataset check the dates first.
2. The Rshp is the most important bit of code, it links both the underlying data from the field to the geospatial objects, be careful with it
3. You can always run this line by line to make sure the data is at the correct stage, this will not be as possible the further we get into the map, so it will be good to check here and make sure everything is where is it supposed to be before moving forward

---

Part 3: The Server

This will set the dataset and slice and dice it as needed by the interactions on the map.

This is automated, however if there needs to be changes a lot of work will need to be done on this side of the code

Server

1. The Server takes the Rshp dataset that we made and manipulates it into the map
2. A really important part of this entire section is the numerical position of the column and which columns correspond to which variables, there probably is an easier way to do this but this is how we got it working (if you change the data you will need to update the column numbers)
3. The server encompasses all parts of the data being used for each part of the datasets for each tab within the map. *So you might be manipulating the underlying data for a variety of different sections at once*

```
server<-function(input, output,session) {
  #____________________________create map consisting of several layers and customization_____________
_________
  output$map1 <- renderLeaflet({  #initiate map
    leaflet(options = leafletOptions(minZoom = 4.5)) %>%
      addProviderTiles(providers$Esri.WorldGrayCanvas) %>% #base map, can be changed
      setView(50.911019,15.889618, zoom = 6.5)
    # setMaxBounds( lng1 = -66.9
    #               , lat1 = 37
    #               , lng2 = -66.1
    #               , lat2 = 37.8 )
  })
```

- This section begins rendering the map, and provides the baselayer, and the initial zoom to the map.
- the set view will show where you start the map initially and from there can look at what the zoom is. This lat-long here is the point that will be in the center of your map, and zoom controls how close you are to that point.
- The lat and longs will get the bounds of the map and edges.

```
 #"observe" inputs to define variables for map colors, titles, legends and subset district data
  observe({
    VARIA <- input$variable1
    #YlOrRd #YlGnBu #RdPu #OrRd #Greys #Greens #viridis #magma
    if (VARIA == "SMEB") {
      dataM<-Rshp[,c(1,5,7,8,10,29,28,30)] #subset exchange rate col
      #mypal<-colorNumeric( palette="RdYlGn", domain=(dataM@data[,6]), na.color = "#9C9D9F", rev
erse = T)
      mypal<-colorNumeric( palette=(colorRamp(c("#ADFFA5", "#A7383D", "#420A0D"), interpolate="l
inear")),domain=dataM@data[,6], na.color = "#D0CFCF", reverse = F)
      pLa<-"WASH SMEB: "
      pLa2<-"WASH SMEB"
      en<-" "
      unitA=" YER"
      title_legend<-"WASH SMEB Cost"
    }
    if (VARIA == "exchange_rates") {
      dataM<-Rshp[,c(1,5,7,8,10,27,28,30)] #subset exchange rate col
      mypal<-colorNumeric( palette="Greens", domain=dataM@data[,6], na.color = "#D0CFCF",reverse
= F)
      pLa<-"Parallel Exchange Rate: "
      pLa2<-"Parrallel Exchange Rate"
      en<-" "
      unitA=" YER"
      title_legend<-"YER to 1 USD"
    }
    if (VARIA == "petrol") {
      dataM<-Rshp[,c(1,5,7,8,10,19,28,30)] #subset exchange rate col
      mypal<-colorNumeric( palette="YlOrBr", domain=dataM@data[,6], na.color = "#D0CFCF", revers
e = F)
      pLa<-"Petrol Price: "
      pLa2<-"Petrol Price"
      en<-" "
      unitA=" YER"
      title_legend<-"Price (1 L)"
    }
    if (VARIA == "diesel") {
      dataM<-Rshp[,c(1,5,7,8,10,20,28,30)] #subset exchange rate col
      #mypal<-colorNumeric( palette="Reds", domain=dataM@data[,6], na.color = "#9C9D9F", reverse
= F)
      mypal<-colorNumeric( palette=(colorRamp(c("#FFD7D9", "#FF535B", "#FB000D", "#830007", "#48
0004"), interpolate="spline")),domain=dataM@data[,6], na.color = "#D0CFCF", reverse = F)
      pLa<-"Diesel Price: "
      pLa2<-"Diesel Price"
      en<-" "
      unitA=" YER"
      title_legend<-"Price (1 L)"
    }
    if (VARIA == "bottled_water") {
      dataM<-Rshp[,c(1,5,7,8,10,21,28,30)] #subset exchange rate col
      #mypal<-colorNumeric( palette="PuBu", domain=dataM@data[,6], na.color = "#9C9D9F", reverse
= F)
      mypal<-colorNumeric( palette=(colorRamp(c("#C7C0FF", "#7A6AFF", "#1501B9", "#0A005D", "#05
```

```
0033"), interpolate="spline")),domain=dataM@data[,6], na.color = "#D0CFCF", reverse = F)
      pLa<-"Bottled Water Price: "
      pLa2<-"Bottled Water Price"
      en<-" "
      unitA=" YER"
      title_legend<-"Price (0.75 L)"
    }
    if (VARIA == "treated_water") {
      dataM<-Rshp[,c(1,5,7,8,10,22,28,30)] #subset exchange rate col
      mypal<-colorNumeric( palette=(colorRamp(c("#C3FFFD", "#6EFBF6", "#009F99", "#00504D"), int
erpolate="linear")),domain=dataM@data[,6], na.color = "#D0CFCF", reverse = F)
      #mypal<-colorNumeric( palette="PuBuGn", domain=dataM@data[,6], na.color = "#9C9D9F", rever
se = F)
      pLa<-"Treated Water Price: "
      pLa2<-"Treated Water Price"
      en<-" "
      unitA=" YER"
      title_legend<-"Price (10 L)"
    }
    if (VARIA == "soap") {
      dataM<-Rshp[,c(1,5,7,8,10,23,28,30)] #subset exchange rate col
      mypal<-colorNumeric( palette="RdPu", domain=dataM@data[,6], na.color = "#D0CFCF", reverse
= F)
      pLa<-"Soap Price: "
      pLa2<-"Soap Price"
      en<-" "
      unitA=" YER"
      title_legend<-"Price (100 g)"
    }
    if (VARIA == "laundry_powder") {
      dataM<-Rshp[,c(1,5,7,8,10,24,28,30)] #subset exchange rate col
      mypal<-colorNumeric( palette="Purples", domain=dataM@data[,6], na.color = "#D0CFCF",revers
e = F)
      pLa<-"Laundry Powder Price: "
      pLa2<-"Laundry Powder Price"
      en<-" "
      unitA=" YER"
      title_legend<-"Price (100 g)"
    }
    if (VARIA == "sanitary_napkins") {
      dataM<-Rshp[,c(1,5,7,8,10,25,28,30)] #subset exchange rate col
      mypal<-colorNumeric( palette="BuPu", domain=dataM@data[,6], na.color = "#D0CFCF", reverse
= F)
      pLa<-"Sanitary Napkin Price: "
      pLa2<-"Sanitary Napkin Price"
      en<-" "
      unitA=" YER"
      title_legend<-"Price (10 Pack)"
    }
    if (VARIA == "cost_cubic_meter") {
      dataM<-Rshp[,c(1,5,7,8,10,26,28,30)] #subset exchange rate col
      #mypal<-colorNumeric( palette="Blues", domain=dataM@data[,6], na.color = "#9C9D9F", revers
e = F)
```

```
      pLa<-"Water Trucking Cost per Cubic Meter: "
      pLa2<-"Water Trucking Cost per Cubic Meter"
      en<-" "
      unitA=" YER"
      title<-"Price (Cubic m)"
      title_legend<-title


      }
```

VERY VERY IMPORTANT PART This is the colorization and selection of data within the map. The VARIA is the potential variables that can be selected, so with a variety of options it is important to disaggregate the Rshp data with these if statements.

To look at one example: * first thing is to take out the columns we need and create our own smaller data grouping

```
dataM<-Rshp[,c(1,5,7,8,10,26,28,30)]
```

IT IS VERY IMPORTNAT TO NOTICE THAT THE NUMBERS OF THE COLUMNS WILL HAVE TO CHANGE AS IN THE Rshp THE COLUMNS ARE CRUTIAL * 1-admin2pcod - district pcode * 5-admin1name - governorate name * 7-admin1pcod - governorate pcode * 8-admin2name - district name * 10-admin2fen - reference to district shapefile * 26-cost_cubic_meter - the variable selected by the if statement, (*this is the variable that will change from if statement to if statement*) * 28-num_obs - the number of observations * 30-date2 - date of the observations

```
 mypal<-colorNumeric( palette=(colorRamp(c("#C9C3F8", "#5D52AD", "#FAD962", "#AA9239"), interpol
ate="linear")),domain=dataM@data[,6], na.color = "#D0CFCF", reverse = T)
```

This section just makes the palette needed for mapping, it changes for each map and selection. I would recommend to keep changing as needed because the distince color palettes do allow the maps to diffientiate between each other.

```
pLa<-"Water Trucking Cost per Cubic Meter: "
     pLa2<-"Water Trucking Cost per Cubic Meter"
     en<-" "
     unitA=" YER"
     title<-"Price (Cubic m)"
     title_legend<-title
```

This is all labeling for maps and legends

That leaves that the inital part of the server completed. We will use this small dataset and labels to populate the map and datasets

```
   #Have a vector of all of the districts that currently have data for the current month
    dataM_NAs<-dataM@data%>%
      filter(is.na(date2))%>%
      pull(admin2pcod)

    #get name of variable selected
    call_name<-colnames(dataM@data[6])

    #Need to subset out for with districts have had values in the past that arent in this curren
t month, get that as a list
    Admin2data_out <- Admin2table %>% #subset out recent month dates to attach to shapefile
      filter(district_ID %in% dataM_NAs)%>% #next only keep data from places that have had an ob
servation in the past from the right varialbe
      filter(!is.na(get(call_name)))

    Admin2data_out <- Admin2data_out[!duplicated(Admin2data_out$district_name),]

    Rshp@data<-Rshp@data%>%
      mutate(alt_dist = admin2pcod %in% Admin2data_out$district_ID)
    Rshp@data$alt_dist<-Rshp@data$alt_dist*1
    Rshp@data$alt_dist<-(dplyr::na_if(Rshp@data$alt_dist,0))
```

This section is corresponds to the past data that is not in the current month, this will be important for highlighing the past districts on the map. This part doenst need to be touched too much.Bascially I made a new column within the Rshp dataset that we will be able to select from.

```
   map1<-leafletProxy("map1") %>%
      clearShapes() %>% #clear polygons, so new ones can be added
      clearControls()%>% #reset zoom etc
```

This is the beginning of the map and will be set and clear old controls to make the map fresh each tiem.

```
 addLabelOnlyMarkers(centroids, #ADD governorate LABELS
                          lat=centroids$lat,
                          lng=centroids$lon,
                          label=as.character(centroids$admin1name),
                          labelOptions = leaflet::labelOptions(
                            noHide = TRUE,
                            interactive = FALSE,
                            direction = "bottom",
                            textOnly = TRUE,
                            offset = c(0, -10),
                            opacity = 0.6,
                            style = list(
                              "color"= "black",
                              "font-size" = "13px",
                              "font-family"= "Helvetica",
                              "font-weight"= 600)
                          )) %>%
```

This creates the governorate labels, and places them on the centroids that we built in the global section.

```
addLabelOnlyMarkers(YEMl, lat=YEMl$lat, #Add Yemen label
                          lng=YEMl$lon,
                          label=as.character(YEMl$name),
                          labelOptions = leaflet::labelOptions(
                            noHide = TRUE,
                            interactive = FALSE,
                            direction = "bottom",
                            textOnly = TRUE,
                            offset = c(0, -10),
                            opacity = 1,
                            style = list(
                              "color"= "black",
                              "font-size" = "24px",
                              "font-family"= "Helvetica",
                              "font-weight"= 800,
                              "letter-spacing"= "3px")
                          )) %>%
```

This create a nice big “YEMEN” so you know that you are looking at Yemen.

```
addPolygons(data= dataM,    # add subsetted district shapefiles
                  color = "grey",
                  weight = 0.8,
                  label= paste(dataM$admin2name," (", pLa, dataM@data[,6],en, ")"),
                  opacity = 1.0,
                  smoothFactor = 0.8,
                  fill = TRUE,
                  fillColor =  (~mypal((dataM@data[,6]))),#custom palette
                  fillOpacity = .8,
                  layerId = ~admin2pcod,
                  highlightOptions = highlightOptions(color = "black", weight = 2,
                                                      bringToFront = FALSE, sendToBack = FALSE),
                  popup = paste0(dataM$admin2name, "<br>",'<h7 style="color:black;">',
                                 pLa, "<b>"," ", dataM@data[,6],unitA, "</b>", '</h7>'),
      ) %>%
```

This is the beginning of the polygons and will add the districts that are present in this months data. As we see the label is using the labels from the if statements above. The fill color is the mypal that we used before and created a custom color palette before. If you remember the dataM@data (mailto:dataM@data)[,6] is the only thing that changes for each if statements, so it is the actual data for each category.

```
      addPolygons(data= old_dist_alt_sp,    # this is you clipped data file of previous district
s (make sure it is below your main district on or it will not be seen)
                  color = "red",
                  weight = 1.5,
                  label = paste0(old_dist_alt_sp$admin2name,":previous ",pLa2," data present"),
#added a different label that pops up
                  opacity = .40,
                  smoothFactor = 0.5,
                  fill = TRUE,
                  fillColor = ~pal_alt(old_dist_alt_sp@data[,5]), #custom palette as stated befo
re
                  fillOpacity = .8,
                  layerId = ~admin2pcod,
                  highlightOptions = highlightOptions(color = "black", weight = 2,
                    bringToFront = FALSE, sendToBack = FALSE),
      )%>%
```

This is the dataset for the old districts that have data in the past but no data currently, so that people know that they can select these polygons and still interact with the data informing them. The color around them will be a slight red that should make it easy to see and highlight but not take too much attention away from the current data. The label is a past that should help to identify the district to the user.

```
  addPolylines(data = Admin1, #add governorate lines for reference
                   weight= 3.25,
                   stroke = T,
                   color = "black",
                   fill=FALSE,
                   fillOpacity = 0.1,
                   opacity = 0.1 )
```

This just adds some nice solid governorate lines that help diffientiate the governorates.

```
    map1 %>% clearControls()

   map1 %>%
     addLegend_decreasing("topleft", pal = mypal, values =  dataM@data[,6], #update legend to re
flect changes in selected district/variable shown
                      labFormat=labelFormat(suffix=unitA),
                      title = title_legend,
                      opacity = 5,
                      decreasing = T)
```

This will add a legend that dynamically updates. This “addLegend_decreacing” function was found and added in the global section, it is a local function. Do not mess with the function in the global part of the code. This only really makes the legend decrease, which is insane that I couldnt find an easier way to do this, but such is.

```
    #needed to make a custom label because i hate R shiny https://stackoverflow.com/questions/52
812238/custom-legend-with-r-leaflet-circles-and-squares-in-same-plot-legends
    colors<-c("white", "#D3D3D3", "#D3D3D3")
    labels<-c("Districts with previous data", "Governorate borders", "District borders")
    sizes<-c("20", "20", "20")
    shapes<-c("square", "line", "line")
    borders<-c("red", "#2B2B2B" , "#646464")
```

This is the beginning of the building of our custom legend, because R shiny is really difficult to work with. So we had to build our own custom function to make it work, thatnks to the stackoverflow link in this section

```
 addLegendCustom <- function(map, colors, labels, sizes, shapes, borders, opacity = 0.5){

    make_shapes <- function(colors, sizes, borders, shapes) {
      shapes <- gsub("circle", "50%", shapes)
      shapes <- gsub("square", "0%", shapes)
      paste0(colors, "; width:", sizes, "px; height:", sizes, "px; border:3px solid ", borders,
"; border-radius:", shapes)
    }
    make_labels <- function(sizes, labels) {
      paste0("<div style='display: inline-block;height: ",
        sizes, "px;margin-top: 4px;line-height: ",
        sizes, "px;'>", labels, "</div>")
    }

    legend_colors <- make_shapes(colors, sizes, borders, shapes)
    legend_labels <- make_labels(sizes, labels)

    return(addLegend(map1,"topleft", colors = legend_colors, labels = legend_labels, opacity =
0.5))
  }
```

This is the custom legend that we built to make sure that our legends make sense. It will rely on the inputs that we created in the previous section.

```
#add new legend
    map1 %>% addLegendCustom(colors, labels, sizes, shapes, borders)

    #add scale bar
    map1 %>% addScaleBar("topleft", options = scaleBarOptions(maxWidth = 100, metric = T, imperi
al = T, updateWhenIdle = T))


  })#end of MAP
```

This is the final part of the map rendering, we add our custom legend, and scale bar. Now we have a map that can be dynamically updated based on which variable is selected.

####Now we want to take it further and make sure that we can update side charts and tables based on what the user selects

```
 clicked_state<- eventReactive(input$map1_shape_click,{ #capture ID of clicked district
    return(input$map1_shape_click$id)
  })
```

Really important part of the code, this section pull the data from whichever polygon the user selects. So the clicked state variable is the variable seleted by the user. This will disaggregate our data later.

```
  clicked_state_gov<-eventReactive(input$map1_shape_click,{
    gov_id<- substr(input$map1_shape_click$id,1,nchar(input$map1_shape_click$id)-2)
    return(gov_id)
  })

  dist_data<-reactive({
    dist_dat<-Admin2table[Admin2table$district_ID==clicked_state(),] #strangely reactive objects
are stored as functions
    dist_dat
  })

  gov_data<-reactive({
    gov_dat<-Admin1table[Admin1table$government_ID==clicked_state_gov(),] #adding the government
data to the dataset
    gov_dat
  })

  nat_data<-reactive({
    nat_dat<-AdminNatTable
    nat_dat
  })
```

This sets the data and filters it as needed based on the clicked state. This will become the datasets we use for charts and graphs after, and thus why the graphs and charts will self update when new districts are selected.

Each of these datasets will play a different role, remember you always need what is returned at the end of a reactive to make sure it is pulled correctly.

```
  gov_nat_data<-reactive({
    gov_nat_dat<-right_join(gov_data(),nat_data(), by = "date2")
  })

  state_data <- reactive({ #subset JMMI data table based on clicked state
    all_dat<-right_join(dist_data(),gov_nat_data(), by = "date2")#using a full join so that the
data that was for the other month when district wasnt select is still shown
    all_dat$date<-as.yearmon(all_dat$date2)
    all_dat
  })
```

These two datasets are combinations of the state and national and district datasets, they will be important when we do counts of observattions per districts as well as making sure that the dates line up.

Once again I hate using dates in R.

```
  chartData1<-reactive({  #subset JMMI data table based on variable of interest (soap, water et
c)
    if (input$variable1 == "SMEB"){
      r<-state_data()[,c(1:5,16,17,31,43,15,30,42)]
      colnames(r)<-c("date","government_name","government_ID","district_name","district_ID","var
iableSEL","date2","governorate_val","nat_val","dist_obs","gov_obs","nat_obs")
    }

    if (input$variable1 == "exchange_rates"){
      r<-state_data()[,c(1:5,14,17,29,41,15,30,42)]
      colnames(r)<-c("date","government_name","government_ID","district_name","district_ID","var
iableSEL","date2","governorate_val","nat_val","dist_obs","gov_obs","nat_obs")
    }

    if (input$variable1 == "petrol"){
      r<-state_data()[,c(1:5,6,17,21,33,15,30,42)]
      colnames(r)<-c("date","government_name","government_ID","district_name","district_ID","var
iableSEL","date2","governorate_val","nat_val","dist_obs","gov_obs","nat_obs")
    }

    if (input$variable1 == "diesel"){
      r<-state_data()[,c(1:5,7,17,22,34,15,30,42)]
      colnames(r)<-c("date","government_name","government_ID","district_name","district_ID","var
iableSEL","date2","governorate_val","nat_val","dist_obs","gov_obs","nat_obs")
    }

    if (input$variable1 == "bottled_water"){
      r<-state_data()[,c(1:5,8,17,23,35,15,30,42)]
      colnames(r)<-c("date","government_name","government_ID","district_name","district_ID","var
iableSEL","date2","governorate_val","nat_val","dist_obs","gov_obs","nat_obs")
    }

    if (input$variable1 == "treated_water"){
      r<-state_data()[,c(1:5,9,17,24,36,15,30,42)]
      colnames(r)<-c("date","government_name","government_ID","district_name","district_ID","var
iableSEL","date2","governorate_val","nat_val","dist_obs","gov_obs","nat_obs")
    }

    if (input$variable1 == "soap"){
      r<-state_data()[,c(1:5,10,17,25,37,15,30,42)]
      colnames(r)<-c("date","government_name","government_ID","district_name","district_ID","var
iableSEL","date2","governorate_val","nat_val","dist_obs","gov_obs","nat_obs")
    }

    if (input$variable1 == "laundry_powder"){
      r<-state_data()[,c(1:5,11,17,26,38,15,30,42)]
      colnames(r)<-c("date","government_name","government_ID","district_name","district_ID","var
iableSEL","date2","governorate_val","nat_val","dist_obs","gov_obs","nat_obs")
    }

    if (input$variable1 == "sanitary_napkins"){
      r<-state_data()[,c(1:5,12,17,27,39,15,30,42)]
      colnames(r)<-c("date","government_name","government_ID","district_name","district_ID","var
```

```
iableSEL","date2","governorate_val","nat_val","dist_obs","gov_obs","nat_obs")
    }

    if (input$variable1 == "cost_cubic_meter"){
      r<-state_data()[,c(1:5,13,17,28,40,15,30,42)]
      colnames(r)<-c("date","government_name","government_ID","district_name","district_ID","var
iableSEL","date2","governorate_val","nat_val","dist_obs","gov_obs","nat_obs")
    }

    r
  })
```

This bit of code, like the if statements before, will slice up the dataset based on what is selected.

It is important to recognize that use of column numbers again and the column names below inidicate which column is which.

The “variableSEL” - or variable selected is the variable value for the selected district.

The output of this part will be stored as Chart1Data

```
chartNAME<-reactive({ #define element to be used as title for selected variable
    if (input$variable1 == "SMEB"){
      y="SMEB"
    }

    if (input$variable1 == "exchange_rates"){
      y="Parallel Exchange Rate"
    }

    if (input$variable1 == "petrol"){
      y= "Petrol Price"
    }

    if (input$variable1 == "diesel"){
      y= "Diesel Price"
    }

    if (input$variable1 == "bottled_water"){
      y= "Bottled Water Price"
    }

    if (input$variable1 == "treated_water"){
      y= "Treated Water Price"
    }

    if (input$variable1 == "soap"){
      y="Soap Price"
    }

    if (input$variable1 == "laundry_powder"){
      y= "Laundry Powder Price"
    }

    if (input$variable1 == "sanitary_napkins"){
      y= "Sanitary Napkins Price"
    }

    if (input$variable1 == "cost_cubic_meter"){
      y= "Water Trucking Price"
    }

    y
  })
```

This part is the same concept as the previous if statement section however, it is only for titles and such.

```
#_____________________Create highcharter element which uses dataset filtered by user inputs_
_____________________
  #NEW OUT PUT FOR DATA TO BE SUBBED LATER
  #https://stackoverflow.com/questions/38113507/r-shiny-keep-retain-values-of-reactive-inputs-af
ter-modifying-selection
  #https://stackoverflow.com/questions/57468457/how-can-i-set-the-yaxis-limits-within-highchart-
plot

  observe({
    updateSelectInput(session = session, inputId = "varDateSelect", choices = chartData1()$date,
selected=lapply(reactiveValuesToList(input), unclass)$varDateSelect)
  })

  output$hcontainer <- renderHighchart({
    event <- (input$map1_shape_click) #Critical Line!!!

    (validate(need(event$id != "",
                   "Please click on a district to display its history.")))

    ChartDat<-chartData1() #define filtered table that is reactive element chartData1
    #y_min<- chartDatMIN()
    #y_max<- chartDatMAX()

    chosenD<- paste0(na.omit(unique(ChartDat[,2])),", ",na.omit(unique(ChartDat[,4]))) #TITLE FO
R CHART (governorate and district name)

    highchart()%>% #high chart
      hc_xAxis(type = "datetime", dateTimeLabelFormats = list(day = '%b %Y')) %>%

      #data for national
      hc_add_series(data=ChartDat, type = "line", hcaes(date2, nat_val), color = "dodgerblue", n
ame=paste(chartNAME(),"-National"))%>%
      #data for governorate
      hc_add_series(data=ChartDat, type = "line", hcaes(date2, governorate_val), color = "forest
green", name=paste(chartNAME(),"-Governorate"))%>%
      #data for district
      hc_add_series(data=ChartDat, type = "line", hcaes(date2, variableSEL), color="#4F4E51", na
me=paste(chartNAME(),"-District"))%>%

      hc_yAxis(tithcle=list(text=paste0(chartNAME()," in YER")), opposite = FALSE
               #,min= as.numeric(y_min), max= as.numeric(y_max)
    )%>%
      hc_title(text=chosenD)%>%
      hc_add_theme(hc_theme_gridlight())%>%
      hc_plotOptions(line = list(
        lineWidth=1.5,
        dataLabels = list(enabled = FALSE)))

  })
```

This is the beginning of the highcharts and construction of dynamic tables throughout the output. * The first line has to do with the pulling the data from the input selections * We pulll the vardateselect as the input id * The date choices are from the chartdata that we created before * The event is triggered by which ever input shapefile is

selected. Look that hte #critical line part of the text * The next line makes a validation checks to make sure that an actual shape file is selected * We make the title of the graph here and then start organizing the data by data. * The rest we use highcharter which is basically a slightly different ggplot2, if you have questions on highcharter look at ? highcharter

```
 #BUILDING TABLE  https://stackoverflow.com/questions/32149487/controlling-table-width-in-shiny-datatableoutput
  output$out_table_obs<-DT::renderDataTable({

  ChartDat_Date<-chartData1()#make a new dataset to play around with from the original state data one above

  #https://stackoverflow.com/questions/40152857/how-to-dynamically-populate-dropdown-box-choices-in-shiny-dashboard

  ChartDat_Date_filter<-ChartDat_Date%>% #filter out based on what was selected from the varDate Select
    filter(date == input$varDateSelect)

  chosenD_Date<- paste0(na.omit(unique(ChartDat_Date[,2])),", ",na.omit(unique(ChartDat_Date[,4])))#make a quick title for the data table

  mat_date_test<-matrix(c(round(ChartDat_Date_filter[6],2),
                          round(ChartDat_Date_filter[8],2),
                          round(ChartDat_Date_filter[9],2),
                          ChartDat_Date_filter[10],
                          ChartDat_Date_filter[11],
                          ChartDat_Date_filter[12]),
    nrow=2, ncol=3, byrow = T,
    dimnames=list(c("Median Price (YER)","Number of Markets Assessed"),c(paste0("District: \n",na.omit(unique(ChartDat_Date[,4]))),paste0("Governorate: \n",na.omit(unique(ChartDat_Date[,2]))),"Yemen")))

  DT::datatable(mat_date_test,options = list(dom = 't'))
   })
```

This datatable produces the table at the bottom of the output, it counts the number of obserations as well as the absolute prices of goods in that area. * The dataset is from the ChartData1() that we had produced before. * We then filter the data based on the variable that was selected * The data is then taken from the second column of the Chart Data and a matrix is then created. * The matrix is a series of columns that have been filtered out and the matrix is row = 2 and col = 3 * Following the production of the matrix we create the columnes and use paste0 to construct what they are * Finally it is saved in a DT or datatable to be projected later.

```
#output infobox for the info exchange rate
output$info_exchange<-renderValueBox({
  exchange_data<-AdminNatTable
  exchange_data$date<-as.yearmon(exchange_data$date)
  exchange_date<-exchange_data%>%
    filter(date == input$varDateSelect)
  exchange_rate<-exchange_date[1,10]

  valueBox(
          value = exchange_rate,
          subtitle="YER to 1 USD",
          icon = icon("dollar"),
          #fill=T,
          color = "green")

})
```

This merely creates that a box that will outline what the exchange rate was for the month that was selected * This is entirely predicated on the CSS files we uploaded earlier. (mainly the valuebox part)

```
  output$text1 <- renderUI({ #customised text elements
    HTML(paste("Coordinate System:  WGS 1984",
               "Administrative boundaries:  OCHA",
               "R Mapping Packages:  leaflet, shiny, highcharter",
               "Please do NOT use Microsoft Edge for best user interaction",
               sep="<br/>"), '<style type="text/css"> .shiny-html-output { font-size: 11px; line
-height: 12px;
         font-family: Helvetica} </style>')h
  })

   #
  output$text2 <- renderUI({
    HTML(paste("<i>Note: Data displayed on this map should be interpreted as indicative.
               In addition, designations and boundaries used here do not imply acceptance
               by REACH partners and associated donors.</i>",
               sep="<br/>"), '<style type="text/css"> .shiny-html-output { font-size: 11px; line
-height: 11px;
         font-family: Helvetica} </style>')
  })

  output$text3 <- renderText({ #LARGE TEXT ABOVE CHART
    paste(chartNAME(), " ", "Medians Over Time")
  })

  output$text_DT<-renderText({
    paste0(chartNAME(),", Median Monthly Costs, and Number of Markets Assessed")
  })

  output$text4 <- renderUI({
    HTML(paste("<i>For more information, please visit our",a("REACH Website", target="_blank", h
ref="https://www.reach-initiative.org"),
    "or contact us directly at yemen@reach-initiative.org.</i>"), '<style type="text/css"> .shin
y-html-output { font-size: 11px; line-height: 11px;
         font-family: Helvetica} </style>')
  })
```

Creates the custom text elements we will put around the map

```
#SMEB DATASET
  output$table_smeb<-DT::renderDataTable({
    #observe({
    #https://stackoverflow.com/questions/50912519/select-the-number-of-rows-to-display-in-a-data
table-based-on-a-slider-input
    time<-input$months
    percent_time<- input$percent/100


    #time<-6
    national_data_test<-nat_data()
    national_data_test<-AdminNatTable
    national_data_test$date2 <- as.yearmon(national_data_test$date)
    national_data<-arrange(national_data_test,desc(date2))

    month_all<-sort(unique(national_data$date2),decreasing = T)
    time_pull<-month_all[time]
    month_list<-month_all[1:match(time_pull,month_all)]

    #now have the month_list which we can cut from in the future

    national_data_pull<-dplyr::filter(national_data, date2==month_list)%>%
      dplyr::select(-c(date,num_obs,exchange_rates))


    national_data_pull<-national_data_pull%>%
      reshape2::melt("date2")%>%
      reshape2::dcast(variable ~ date2)%>%
      round_df(.,0)

    col_data_pull<-ncol(national_data_pull)

    name_perc_change<-paste0(colnames(national_data_pull[col_data_pull]),
                             " percent change from standard SMEB"
                             )
    #Add SMEB base costs
    #https://stackoverflow.com/questions/13502601/add-insert-a-column-between-two-columns-in-a-d
ata-frame
    national_data_pull<-national_data_pull%>%
      add_column(.,  `Standard SMEB Values`= c(365,430,100,120,130,105,525,1825,12000), .after =
1)%>%
      add_column(.,  `Variable`= c("Petrol","Diesel","Bottled water","Treated water","Soap","Lau
ndry powder","Sanitary napkins","Water trucking","SMEB total"), .after = 1)%>%
      dplyr::select(c(-1))%>%
      dplyr::filter(Variable %in% c("Soap","Laundry powder","Sanitary napkins","Water truckin
g","SMEB total"))

    #get number of columns now we will use later in the formatting of the table
    columns_of_data_begin<-ncol(national_data_pull)+1

    #get the column number of the percent change for future formatting
    percent_col<-time+2
    col_format_last<-time+1
```

```
    #https://duckduckgo.com/?q=dynamic+naming+in+mutate+R&t=brave&ia=web
    national_data_pull<-national_data_pull%>%
      dplyr::mutate_at(., .vars = c(3:ncol(.)),.funs = list(`percent change from standard SMEB`=
~((.-(national_data_pull[,2]))/(national_data_pull[,2]))
))

    #get number of columns now we will use later in the formatting of the table
    columns_of_data_end<-ncol(national_data_pull)

    #get rid of the weird naming from the mutate_at
    names(national_data_pull) <- gsub("_", " ", names(national_data_pull))

    #maybe keep for later
    #dplyr::mutate(.,!!name_perc_change := (((.[,col_data_pull]-.[,2])/(.[,2]))))

    #Render the output DT
    #https://stackoverflow.com/questions/60659666/changing-color-for-cells-on-dt-table-in-shiny
    #https://blog.rstudio.com/2015/06/24/dt-an-r-interface-to-the-datatables-library/
    DT::datatable(national_data_pull,extensions = c('FixedColumns'),
                  options = list(searching = F, paging = F, scrollX=T, fixedColumns = list(leftC
olumns = 2, rightColumns = 0)),
                  rownames = F)%>%
      formatStyle(columns = 2, color = "white", backgroundColor = "grey", fontWeight = "bold")%
>%
      DT::formatPercentage(columns = c(columns_of_data_begin:columns_of_data_end),2)%>%
      formatStyle(columns = c(columns_of_data_begin:columns_of_data_end),
                  color = styleInterval(c(-percent_time,percent_time), c('grey', 'black','whit
e')),
                  backgroundColor = styleInterval(c(-percent_time,percent_time), c('#66FF66', 'w
hite','#FA5353')),
                  fontWeight = styleInterval(c(-percent_time,percent_time),c('bold','normal','bo
ld')))


  })
```

This create the smeb tracking dataset, It observes the inputs selected by from the UI and creates tables to match * The two main inputs are the sliders on the third page of the UI, they look at the months you want to focus on as well as the percent threshold you want. (these are found as the varaible time and percent_time) * The code then melts these options together to create a functional dataset that we can adapt and rename. * It is important to note the positions of columns as those are the reasons for many of the format styles or holds on different namings throughout the table. * This table only encompasses SMEEB items so if the SMEB changes it will be good to change tthe base (where we add columns in the inbetween two dataframe.)

```
  #Other goods dataset
  output$table_other<-DT::renderDataTable({
    #observe({
    #https://stackoverflow.com/questions/50912519/select-the-number-of-rows-to-display-in-a-data
table-based-on-a-slider-input
    time<-input$months
    percent_time<- input$percent/100


    #time<-6
    national_data_test<-nat_data()
    national_data_test<-AdminNatTable
    national_data_test$date2 <- as.yearmon(national_data_test$date)
    national_data<-arrange(national_data_test,desc(date2))

    month_all<-sort(unique(national_data$date2),decreasing = T)
    time_pull<-month_all[time]
    month_list<-month_all[1:match(time_pull,month_all)]

    #now have the month_list which we can cut from in the future

    national_data_pull<-dplyr::filter(national_data, date2==month_list)%>%
      dplyr::select(-c(date,num_obs,exchange_rates))


    national_data_pull<-national_data_pull%>%
      reshape2::melt("date2")%>%
      reshape2::dcast(variable ~ date2)%>%
      round_df(.,0)

    col_data_pull<-ncol(national_data_pull)

    name_perc_change<-paste0(colnames(national_data_pull[col_data_pull]),
                             " percent change from standard SMEB"
    )
    #Add SMEB base costs
    #https://stackoverflow.com/questions/13502601/add-insert-a-column-between-two-columns-in-a-d
ata-frame
    national_data_pull<-national_data_pull%>%
      #add_column(.,  `Standard SMEB Values`= c(365,430,100,120,130,105,525,1825,12000), .after
= 1)%>%
      add_column(.,  `Variable`= c("Petrol","Diesel","Bottled water","Treated water","Soap","Lau
ndry powder","Sanitary napkins","Water trucking","SMEB total"), .after = 1)%>%
      dplyr::select(c(-1))%>%
      dplyr::filter(Variable %in% c("Petrol","Diesel","Bottled water","Treated water"))

    #get number of columns now we will use later in the formatting of the table
    columns_of_data_begin<-ncol(national_data_pull)+1

    #get the column number of the percent change for future formatting
    percent_col<-time+2
    col_format_last<-time+1
```

```
    #get number of columns now we will use later in the formatting of the table
    columns_of_data_end<-ncol(national_data_pull)

    #get rid of the weird naming from the mutate_at
    names(national_data_pull) <- gsub("_", " ", names(national_data_pull))

    #maybe keep for later
    #dplyr::mutate(.,!!name_perc_change := (((.[,col_data_pull]-.[,2])/(.[,2]))))

    #Render the output DT
    #https://stackoverflow.com/questions/60659666/changing-color-for-cells-on-dt-table-in-shiny
    #https://blog.rstudio.com/2015/06/24/dt-an-r-interface-to-the-datatables-library/
    DT::datatable(national_data_pull,extensions = c('FixedColumns'),
                  options = list(searching = F, paging = F, scrollX=T, fixedColumns = list(leftC
olumns = 1, rightColumns = 0)),
                  rownames = F)

  })
```

This section is the same as the previous but focuses on non-SMEB good, its has the same triggers but will not highlight. * This has the same column naming convention and will help to keep everything easy between the two codes. * There is no real bas here its just an informative table

Final notes:

1. The server is always really complicated and can be a mess to work with, to make it easier I made this test code to simulate what the server is doing at a give time

```
  dist_dat<-Admin2table[Admin2table$district_ID=="YE2307",] #strangely reactive objects are stor
ed as functions


  gov_dat<-Admin1table[Admin1table$government_ID=="YE23",] #adding the government data to the da
taset


  nat_dat<-AdminNatTable


  gov_nat_dat<-right_join(gov_dat,nat_dat, by = "date2")

  all_dat<-right_join(dist_dat,gov_nat_dat, by = "date2")#using a full join so that the data tha
t was for the other month when district wasnt select is still shown
  all_dat$date<-as.yearmon(all_dat$date2)
  all_dat


  r<-all_dat[,c(1:5,16,17,31,43,15,30,42)]
  colnames(r)<-c("date","government_name","government_ID","district_name","district_ID","variabl
eSEL","date2","governorate_val","nat_val","dist_obs","gov_obs","nat_obs")
r
```

2. If you add to the server make sure to just copy and paste and re check the column numbers
3. Not a bad idea to save a copy of the server when you make changes

---

Parr 4: The UI

The User Interface is kinda complicated, but if you realize it just broken up into 4 seperate sections it makes it much easier to manage.

The four sections of the UI

1. The map layout (main map and what people will see and interact with)
2. The methodology section (pretty set dont really need to touch)
3. The SMEB Tracker (displays the tables we built at the end of the server, and houses the input sliders we referenced previously)
4. The Partners (needs to be updated but is basically just alot of copy and paste work)

Beginnings of the code:

```
# UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI UI U
I UI UI UI UI UI UI UI


#DROP DOWN MENU SELECTIONS
vars1 <- c(
  "WASH SMEB"="SMEB",
  "Parallel Exchange Rates"="exchange_rates",
  "Petrol" = "petrol",
  "Diesel" = "diesel",
  "Bottled Water"="bottled_water",
  "Treated Water"="treated_water",
  "Soap"="soap",
  "Laundry Powder"="laundry_powder",
  "Sanitary Napkins"="sanitary_napkins",
  "Water Trucking"= "cost_cubic_meter"
)

varsDate<- c("Months to Select" = "varDateSelect")

#USER INTERFACE COMPONENTS
navbarPage(theme= shinytheme("journal"),
           title=strong(HTML("<span style='font-size:30px'>YEMEN: JOINT MARKET MONITORING INITIA
TIVE</span>")), # id="nav", #MAIN TITLE
           windowTitle = "REACH: Yemen Joint Market Monitoring Initiative (JMMI)", #Title for br
owser tab window
```

This is the start of the code we have all of our variables set right there, as well as what the variable will be called for the date (varsDate)

The navbarpage is the beginning of the map and all sheets or tabs start from there. Nara did a great job outlining what each line does in this code

The map code:

```
 ###.................................M A P. . P A G E
.....................................
           tabPanel(strong("JMMI"), #TAB LABEL
                    icon= icon("map-marker"), #TAB ICON
                    div(class="outer",

                        tags$head(
                          # Include our custom CSS
                          includeCSS("AdminLTE.css"),
                          shiny::includeCSS( "bootstrap.css"), #added
                          includeCSS(path = "shinydashboard.css"),
                          br()#added
                        ),

                        #LEAFLET MAP
                        # If not using custom CSS, set height of leafletOutput to a number inste
ad of percent
                        leafletOutput("map1", width="100%", height="100%"), #BRING IN LEAFLET MA
P, object created in server.R
                        tags$head(tags$style(".leaflet-control-zoom { display: none; }

                                             #controls {height:90vh; overflow-y: auto; }
                                               ")), #remove map zoom controls

                        tags$head(tags$style(
                          type = "text/css",
                          "#controlPanel {background-color: rgba(255,255,255,0.8);}",
                          ".leaflet-top.leaflet-left .leaflet-control {
                           margin-top: 25px;
                         }"
                        )),
                        #https://stackoverflow.com/questions/37861234/adjust-the-height-of-infob
ox-in-shiny-dashboard



                        #SIDE PANEL
                        absolutePanel(id = "controls", class = "panel panel-default", fixed = TR
UE,
                                      draggable = TRUE, top = 60, left = "auto", right = 20, bot
tom = 1,
                                      width = 500, height = "auto",

                                      hr(),
                                      h5("The Yemen Joint Market Monitoring Initiative (JMMI) is
a harmonized price monitoriong initiative that focuses on informing
                                      the Water, Sanitation, and Hygiene (WASH) Cluster and the
Cash
                                      and Market Working Group (CMWG) to support humanitarian ac
tivies throughout Yemen.
                                      The JMMI provides an indicative estimation of the prices o
f WASH and fuel items across districts in Yemen."),

                                      h5(tags$u("Most recent findings displayed in map are from
```

```
data collected in ", #DistsNumn and currentD will change based on the most recent JMMI, defined in global.R
                                                  tags$strong(DistsNumb), "districts in ", tags$strong(paste0(currentD,"."))),
                                        ("The districts outlined in red indicate that data for the selected item was collected in that district in previous months.")),

                                    h5("Further details regarding the JMMI methodology and the Survival Minimum Expenditure Basket (SMEB) calculation can be found on the information tab.
                                       For additional information on supply chains and market-related concerns, please visit the  ",a("REACH Resource Center", target="_blank",    href="https://www.reachresourcecentre.info/country/yemen/cycle/754/#cycle-754"), " to access the monthly situation overviews."),

                                    hr(),

                                    #h5(tags$u("Most recent findings displayed in map are from data collected in ", #DistsNumn and currentD will change based on the most recent JMMI, defined in global.R
                                    #   tags$strong(DistsNumb), "districts in ", tags$strong(currentD))),

                                    selectInput("variable1", h4("Select Variable Below"), vars1, selected = "SMEB"), #linked text


                                    h5(textOutput("text3")), #extra small text which had to be customized as an html output in server.r (same with text1 and text 2)

                                    #HIGH CHART
                                    highchartOutput("hcontainer", height= 300, width = 450),

                                    #new data table
                                    hr(),
                                    selectInput(inputId= "varDateSelect", label = h4("Select Month of Data Collection"), choices=NULL, selected = (("varDateSelect"))),#linked date stuff
                                    h5("Please select a district to enable month selection"),
                                    h5(textOutput("text_DT")),
                                    DT::dataTableOutput("out_table_obs",height = "auto", width = "100%"),

                                    #####Attempt to add an info box
                                    hr(),
                                    h5("Exchange Rate for selected month"),
                                    h5("Please select month to populate the information box"),
                                    fluidRow(valueBoxOutput("info_exchange", width = 12)),
                                    hr(),
                                    #hr(),

                                    h6(htmlOutput("text1")),
                                    h6(htmlOutput("text2")),
                                    h6(htmlOutput("text4")),
                                    column(width=12, align="center", div(id="cite2", "Funded by: "), img(src='DFID UKAID.png', width= "90px"),img(src='OCHA@3x.png', width= "90px"),
```

```
                                        img(src='USAID.png', width= "105px")) #donor logos


                        ),


                        tags$div(id="cite",
                                 a(img(src='reach_logoInforming.jpg', height= "40px"), target="_
blank", href="http://www.reach-initiative.org"),
                                 img(src='CMWG Logo.jpg', height= "40px", style='padding:1px;bor
der:thin solid black;'),
                                 img(src='washlogo_grey-300DPI.png', height= "40px"))

                        # tags$div(id="cite",
                        #          a(img(src='reach_logoInforming.jpg', width= "200px"), target
="_blank", href="http://www.reach-initiative.org"))
                    )
           ),
```

- The first part about tags$heads will bring the CSS part of the formatting and help make the overall look of the map much nicer and more orderly
- The next part is the actual projection of the map and where it zoom angles are
- The next area is about the absolute panel within the tab, the absolute panel is the panel that projects where things are and the graphs as well as the tables that are set for each selected varaible.
- As you can see in this part there is a lot of text and movement of of hr() and h5(), the h5() is just the style and size of the text being rendered.
- In this table we also put in the highcharter box as well as the the table and other outputs we created at the bottome of the server. (*this is dynamic and will update as needed (inshallah)*)
- At the bottom you can see the logos used for each one and the web addresses needed
- The tag cite was made for the bottome ones and used for the jpgs of donors, **all jpgs should be stored in the wwww folder inside the project**

The methodology code:

```
###.............................I N F O. . P A G E
....................................
          tabPanel(strong("Information"),
                   tags$head(tags$style("{ height:90vh; overflow-y: scroll; }")),

                   icon= icon("info"), #info-circle
                   div(#class="outer",

                       tags$head(
                         # Include our custom CSS
                         includeCSS("styles.css"),
                       style=" { height:90vh; overflow-y: scroll; }
                                           "),

                       column(width=8,h3("Overview")), #h1- h5 change the header level of the t
ext

                       column(width=7,h5("The  Yemen  Joint  Market  Monitoring  Initiative  (J
MMI) is an
                                         initative led by REACH in collaboration with the Wate
r, Sanitation,
                                         and Hygiene (WASH) Cluster  and the Cash and Market Wo
rking Group (CMWG)
                                         to support humanitarian cash actors with the harmoniza
tion of price
                                         monitoring throughout Yemen. The basket of goods asses
sed includes eight
                                         non-food items (NFIs), including fuel, water and hygie
ne products,
                                         reflecting the programmatic areas of the WASH Cluster.
The JMMI
                                         tracks all components of the WASH Survival Minimum Exp
enditure Basket
                                         (SMEB) since September 2018.")),

                       column(width=8,h3("Methodology")), #h1- h5 change the header level of th
e text

                       column(width=7,h5("Data was collected through interviews with vendor Key
Informants
                                         (KIs), selected by partner organizations from markets
of various sizes
                                         in both urban and rural areas. To be assessed by the J
MMI, markets
                                         must be either a single permanent market, or a local c
ommunity where
                                         multiple commercial areas are located in close proximi
ty to one another.
                                         When possible, markets/shops are selected within a sin
gle geographical
                                         location, where there is at least one wholesaler opera
ting in the
                                         market, or multiple areas of commerce within the same
```

```
geographical
                                        location when it is too small, to provide a minimum of
three price
                                        quotations per assessed item.", tags$i(tags$strong("Fi
ndings are indicative for the assessed
                                        locations and timeframe in which the data was collecte
d."))))),

                        column(width=8,h3("SMEB Calculation")), #h1- h5 change the header level
of the text

                        column(width=7,h5("Each month, enumerators conduct KI interviews with ma
rket vendors to collect three price quotations for each item from the same market in each distri
ct.
                                        REACH calculates the WASH SMEB,
                                        which is composed of four median item prices: Soap (1.
05 kg), Laundry Powder (2 kg), Sanitary Napkins (20 units) ,and Water Trucking (3.15 m3).",
                                        p(),
                                        p("The calculation of the aggregated median price for
districts and governorates is done following a stepped approach.
                                        Firstly, the median of all the price quotations relate
d to the same market is taken. Secondly, the median quotation from each market is aggregated to
calculate the district median.
                                        Finally, the median quotation from each district is ag
gregated to calculate the governorate median. "))),


                        column(width=8,h3("About REACH")), #h1- h5 change the header level of th
e text

                        column(width=7,h5("REACH is a joint initiative that facilitates the deve
lopment of
                                        information tools and products that enhance the capaci
ty of aid actors
                                        to make evidence-based decisions in emergency, recover
y and development
                                        contexts. By doing so, REACH contributes to ensuring t
hat communities
                                        affected by emergencies receive the support they need.
All REACH
                                        activities are conducted in support to and within the
framework of
                                        inter-agency aid coordination  mechanisms. For more in
formation, please
                                        visit our",a("REACH Website", target="_blank",    href
="https://www.reach-initiative.org"), "or contact us directly
                                        at yemen@reach-initiative.org.")),

                         hr(),
                        p(),
                       p(),
                       hr(),

                         tags$div(id="cite4",
```

```
                                  a(img(src='reach_logoInforming.jpg', width= "200px"), target="_
blank", href="http://www.reach-initiative.org")))
                    ),
```

This part is basically just a bunch of text positioned around. So change text as need if methodology changes.

The SMEB Tracker Table

```
  tabPanel(strong("SMEB Tracker"),


                   style=("{overflow-y:auto; }"),
                   icon= icon("bar-chart"), #info-circle
                   div(tags$head(
                     # Include our custom CSS
                      tags$style(".fa-check {color:#008000}"),
                      tags$style(HTML(".sidebar {height:50vh; overflow-y:auto; }"))
                    ),
                    sidebarLayout(
                      sidebarPanel(
                        sliderInput("months","Number of months displayed", min = 1, max = 24, st
ep = 1,value = 6, ticks = F),
                        h6("Displays the number of months from the most recent dataset"),
                        br(),
                        br(),
                        sliderInput("percent","Percentage change highlighted", min = 1, max = 10
0, value = 20, tick=F),
                        h6("Is the percent difference desired for the benchmark"),
                        width=2.5),

                      mainPanel(
                        h2("Monthly SMEB Costs and Percentage Change from Standard SMEB Value
s"),
                       DT::dataTableOutput("table_smeb"),
                       tags$hr(),
                       h2("Monthly Cost for Other non-SMEB goods"),
                        DT::dataTableOutput("table_other")


                      )
                    )




                   )),
           #conditionalPanel("false", icon("crosshairs")),
#)
```

This is a normal table the things to look out for here are the sliderInputs, those are the inputs that go back to the server, so do make sure that those are always working, otherwise we are only really messing with a datatable and then republishing it.

The partners

```
tabPanel(strong("Partners"),

         #style=("{overflow-y:auto; }"),
         icon= icon("handshake"), #info-circle
         div(tags$head(
               # Include our custom CSS
               tags$style(".fa-check {color:#008000}"),
               tags$style(HTML(".sidebar {height:50vh; overflow-y:auto; }"))
             ),

             column(width=8,h3("Partners (Past and Present)")), #h1- h5 change the header level
of the text
             column(width=7, h6(tags$i("Check marks indicate that the partner participated in th
e most recent month’s JMMI"))),

             #icon("check", "fa-2x")),
             #list of partners orgs
             column(width=12,align = "center", h5("Agency for Technical Cooperation and Developm
ent (ACTED)",icon("check", "fa-2x")),a(img(src='0_acted.png', height= "50px"), target="_blank",
href="https://www.acted.org/en/countries/yemen/")),
             column(width=12,align = "center", h5("Adventist Development and Relief Agency (ADR
A)", icon("check", "fa-2x")), a(img(src='0_adra.png', height= "50px"), target = "_blank", href
="https://adra.org/")),
             column(width=12,align = "center", h5("Al Thadamon Association"), img(src='0_thadamo
n.jpg', height= "50px")),
             column(width=12,align = "center", h5("Brains for Development (B4D)"),img(src='0_b4
d.jpg', height= "50px")),
             column(width=12,align = "center", h5("Creative Youth Forum (CYF)"), a(img(src='0_cy
f.jpg', height= "50px"),target="_blank", href="https://www.facebook.com/cyf.org77/")),
             column(width=12,align = "center", h5("Danish Refugee Council (DRC)", icon("check",
"fa-2x")), a(img(src='0_drc.png', height= "50px"), target="_blank", href="http://www.drc.dk")),
             column(width=12,align = "center", h5("Generations without Qat (GWQ)", icon("check",
"fa-2x")), img(src='0_gwq.png', height= "50px")),
             #column(width=12,align = "center", h5("LLMPO"), img(src='0_cyf.png', height= "50p
x")),
             column(width=12,align = "center", h5("International Organization for Migration (IO
M)", icon("check", "fa-2x")), img(src='0_iom.png', height= "50px")),
             column(width=12,align = "center", h5("Mercy Corps (MC)"), img(src='0_mercy.jfif', h
eight= "50px")),
             column(width=12,align = "center", h5("National Foundation for Development and Human
itarian Response (NFDHR)",icon("check", "fa-2x")), a(img(src='0_nfdhr.png', height= "50px"),targ
et="_blank", href="http://nfdhr.org/")),
             column(width=12,align = "center", h5("National Forum Human Development (NFHD)"), im
g(src='0_nfhd.png', height= "50px")),
             column(width=12,align = "center", h5("Norweigan Refugee Council (NRC)", icon("chec
k", "fa-2x")), img(src='0_nrc.png', height= "50px")),
             column(width=12,align = "center", h5("Old City Foundation for Development (OCFD)",
icon("check", "fa-2x")), img(src='0_ocfd.jpg', height= "50px")),
             column(width=12,align = "center", h5("OXFAM"), img(src='0_oxfam.png', height= "50p
x")),
             column(width=12,align = "center", h5("Rising Org. for Children Rights Development
(ROC)",icon("check", "fa-2x")), a(img(src='0_roc.jpg', height= "50px"), target="_blank", href="h
```

```
ttps://rocye.org/")),
             column(width=12,align = "center", h5("Sama Al Yemen", icon("check", "fa-2x")), img
(src='0_sama.jpg', height= "50px")),
             column(width=12,align = "center", h5("Save the Children (SCI)",icon("check", "fa-2
x")), img(src='0_sci.png', height= "50px")),
             column(width=12,align = "center", h5("Sustainable Development Foundation (SDF)"), i
mg(src='0_sdf.jpg', height= "50px")),
             column(width=12,align = "center", h5("Solidarites International (SI)"), img(src='0_
si.jpeg', height= "50px")),
             column(width=12,align = "center", h5("Soul Yemen"), img(src='0_soul.jpg', height=
"50px")),
             column(width=12,align = "center", h5("Tamdeen Youth Foundation (TYF)",icon("check",
"fa-2x")), img(src='0_tyf.png', height= "50px")),
             column(width=12,align = "center", h5("Vision Hope"), img(src='0_vision.png', height
= "50px")),
             column(width=12,align = "center", h5("Yemen Family Care Association (YFCA)"), img(s
rc='0_yfca.jpg', height= "50px")),
             column(width=12,align = "center", h5("Yemen Shoreline Development (YSD)"), img(src
='0_ysd.jpg', height= "50px")),
             p(),
             hr()


))

)
```

This part should be changed each month to correspond to which partners are participating in the JMMI

The basic outline of what you do is keep the first part the same

```
column(width=12,align = "center", h5("Adventist Development and Relief Agency (ADRA)", icon("che
ck", "fa-2x")), a(img(src='0_adra.png', height= "50px"), target = "_blank", href="https://adra.o
rg/")),
```

For instance this part stays the same each time

```
column(width=12,align = "center",
```

This next part is changed to the name of the org, with a check mark and without are list below

```
h5("Adventist Development and Relief Agency (ADRA)", icon("check", "fa-2x"))

h5("Adventist Development and Relief Agency (ADRA)")
```

So this bit of code should be added after the " in the code above to produce the check (*, icon("check", "fa-2x")*)

The next part is the image (which should be saved in the www folder, make sure the png or jpg make sense and are listed as such *dont call a jpg if the picture is a png*)

```
, a(img(src='0_adra.png', height= "50px"), target = "_blank", href="https://adra.org/")),
```

First part is the image call, next is the hight its is set at and target (leave height and target the say they are)

Finally the href is the website that the user will be taken to if they click onthe image, so update as needed.

That is it.

Good luck.

Always save copies.

Best, Nate