390R real data analysis 2
11/12/2020
Notes

Today let's explore the Kerala cases dataset.
This is data on Covid-19 in the State of Kerala in India.
When we try to import the data we get an error saying that there isn't a name for the 14th column. This column is blank for all observations. We can either remove the column in the original dataset or we can remove the missing column after importing the data.
kerala%>%select(-X14)
The next step is to get a brief overview of the data. Use functions like range and names.

One thing we can do before data visualization is to clean the data.
We can remove some of the variables that we don't need.
Lets focus on Kerala and we can ignore the india variables
kerala2<-select(1:7)%>%mutate(Date = mdy(Date))

Focusing on new cases using date and new cases
kerala2%>%ggplot()+geom_col(aes(x=Date,y=New_cases),fill="blue")

We can also add some texts and arrows to indicate which phases happen
kerala_line<- tibble(date_line = mdy(c("3/09/2020","3/23/2020","5/09/2020"))
Kerala_label <- tibble(x=mdy(c("2/20/2020","3/16/2020)),
y=c(65,70),xend=mdy(c("3/08/2020","3/23/2020")),yend=c(60,60),label =
c("2phase","Lockdown"))

kerala2%>%ggplot()+geom_col(aes(x=Date,y=New_cases),fill="blue")+geom_vline(data =
kerala_line, mapping = aes(xintercept = date_line)) +
geom_segment(data=kerala_label,aes(x=x,y=y,xend=xend,yend=yend),arrow =
arrow(length=unit(0.1,"cm"))+
geom_text(data=kerala_label,aes(x=x,y=y,label=label),size=3)

Lets say we want to calculate the number of cases per a certain number of people so that we can compare different places with different populations.
Kerala%>%mutate(kerala_new_cases = New_cases/population*1000000,
Date=mdy(Date))%>% select(Date,kerala_new_cases, india_new_cases_per_m =
india_new_cases_per_million)%>% gather(place,rate,2:3) %>% mutate(place =
fct_recode(place, "Kerala" = "kerala_new_cases", "India" = "india_new_cases_per_m"))%>%
ggplot()+geom_line(aes(x=Date,y=rate,color=place))

New topic: Programming
How to write functions in R, how to use if statements, how to do loops.
This will make your code more readable.

Examples: mean(1:10) to find the mean of a vector
say.hello<-function(){print("Hello, world")} to print out Hello, world
Function name <-function(arguments){function}

squares<- function(x){print(x^2)}
You must provide the arguments when calling the function: squares(4)

You can set default values for arguments.
squares<- function(x,y=10){print(x^2 + y^2)} will default y to 10 when not specified.

One way to return a value is with the return function.
squares<- function(x,y=10){return(x^2 + y^2)} it doesn't make a difference because there is only one line. If you have many lines it's easier to explicitly say which line you want returned.

Let's say you want to change a time to the number of minutes from midnight.
(1030 %/%100*60 + 1030 %% 100) %% 1440
min.from.midnight<- function(time_hhmm){ return((time_hhmm %/%100*60 + time_hhmm %% 100)%%1440}
Or
Min.from.midnight <- function(time_hhmm){
hour<- time_hhmm%/%100
minute<- time_hhmm%%100
return(hour*60+min)}