Problem set #1

Shunyi Huang

2022/9/24

All library

```
library(tidyverse)
library(shiny)
library(readr)
library(dplyr)
library(knitr)
```

London Olympics

```
library(shiny)
olympics <- read.csv("https://uwmadison.box.com/shared/static/rzw8h2x6dp5693gdbpgxaf2koqijo12
l.csv")
olympics$selected = runif(nrow(olympics), 0, 1)

ui <- fluidPage(
  selectInput("dropdown", "Select a Sport", choices = unique(olympics$Sport), multiple = TRU
E),
  fluidRow(
    column(width = 4,dataTableOutput('table')),
    column(width = 4,dataTableOutput('avg_age')),
    column(width = 4, dataTableOutput('total_people'))
  )


)

server <- function(input, output) {
  #reactive function
  olymp_sub <- reactive({
    olympics%>%mutate(selected=1 * (Sport %in% input$dropdown))
  })


  # table across all sports
  output$table <- renderDataTable({
    olymp_sub()[olymp_sub()$selected == 1, c('Name', 'Age', 'Sport')]
  })


  #average age across each sports
  output$avg_age <-renderDataTable({
    avg_age = olymp_sub()[olymp_sub()$selected == 1, c('Name', 'Age', 'Sport')] %>%
      group_by(Sport) %>%
      summarise_at(vars(Age), list(name = mean))

    colnames(avg_age) = c('sport', 'average_age')
    avg_age = avg_age[order(avg_age$average_age),]
    avg_age
  })


  #Extra question: order the number of people in each sport in ascending order
  output$total_people <-renderDataTable({
    total = olymp_sub()[olymp_sub()$selected == 1, c('Name', 'Sport')] %>%
      group_by(Sport) %>%
      summarise(total = n())

    colnames(total) = c('sport', 'sum')
    total = total[order(total$sum),]
    total
```

```
  })

}


shinyApp(ui, server)
```

Select a Sport

Athletics Rowing Hockey Synchronised Swimming

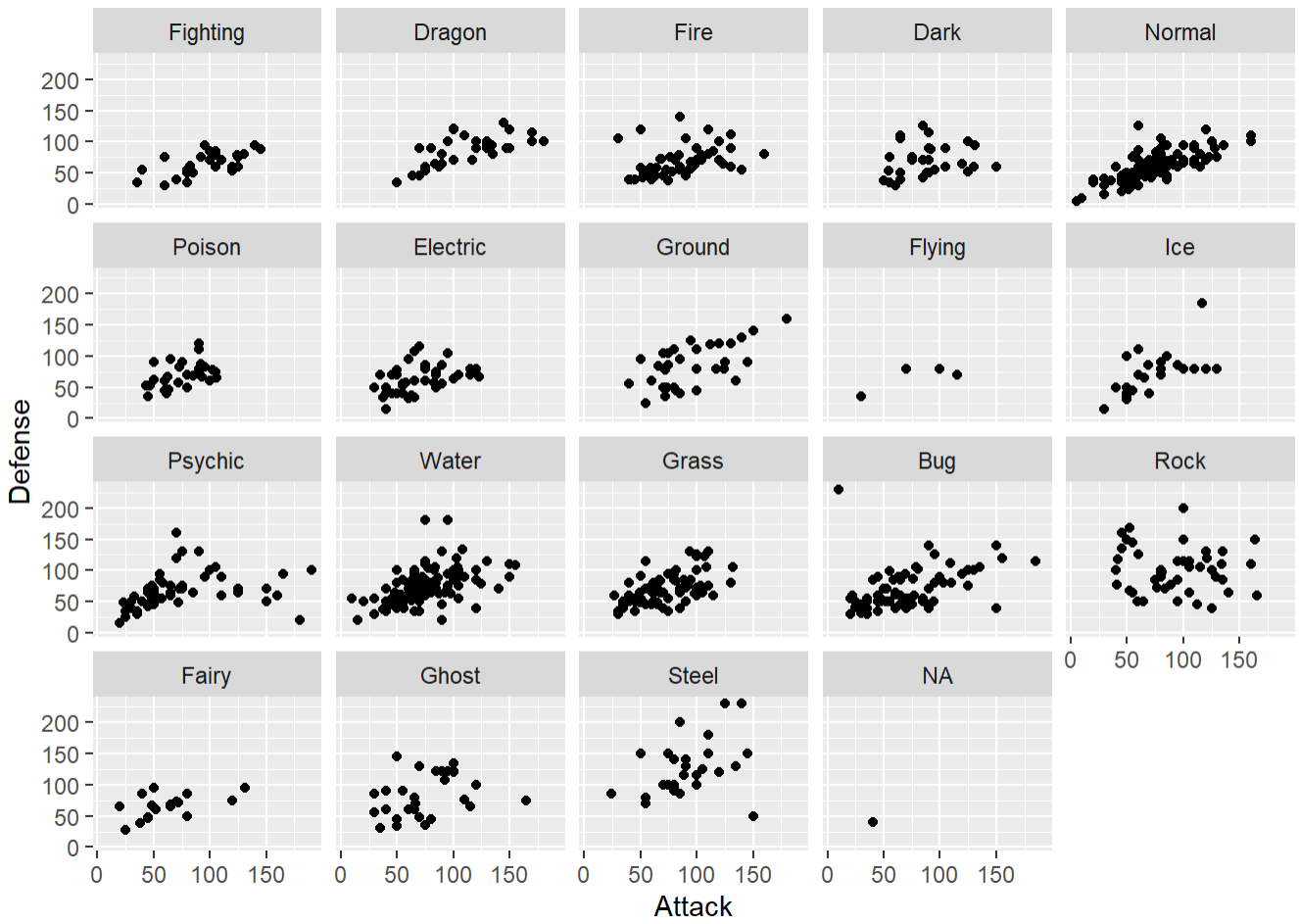
Show 25 entries

Search:

Name	Age	Sport
A G Kruger	33	Athletics
Jamale Aarrass	30	Athletics
Maria Abakumova	26	Athletics
Maria Laura	30	Rowing

Pokemon

```
pokemon <- read.csv("https://uwmadison.box.com/shared/static/hf5cmx3ew3ch0v6t0c2x56838er1lt2
c.csv")
#A
pokemon$attack_to_defense = pokemon$Attack/pokemon$Defense

#B
type_1_median = pokemon %>%
  group_by(type_1) %>%
  #mutate(median = median(attack_to_defense))
  summarise_at(vars(attack_to_defense), list(median = median))
type_1_median = type_1_median[order(type_1_median$median, decreasing = TRUE),]


#C
order = list(type_1_median$type_1)
order = c( "Fighting" ,"Dragon"  , "Fire"    , "Dark"    , "Normal"   ,"Poison"   ,"Electri
c", "Ground",   "Flying"  ,"Ice" ,     "Psychic",  "Water" ,   "Grass" ,   "Bug"    ,   "Rock"
,   "Fairy"   ,  "Ghost" ,   "Steel" )
#pokemon = pokemon[order(sapply(pokemon$type_1, function(type_1) which(type_1 == order))),]

ggplot(data = pokemon, aes(x = Attack, y = Defense))+
  geom_point()+
  #facet_wrap(~type_1)
  facet_wrap(~factor(type_1, levels = order))
```



```
#D
# Thoughts: The dynamic queries can be implemented using Shiny library. One example I could t
hink of is to apply slider to select the range of HP/Attck/Defense/speed_attack and show the
table with the all the filters.

# Structure: By implementing the ui and server function, we could run the shiny app, and ther
e will be multiple dropdown threholds showing up where user could choose based on their prefe
rence.

# With user input: when user chooses the range of the features, the table that has been filte
red will be shown
```

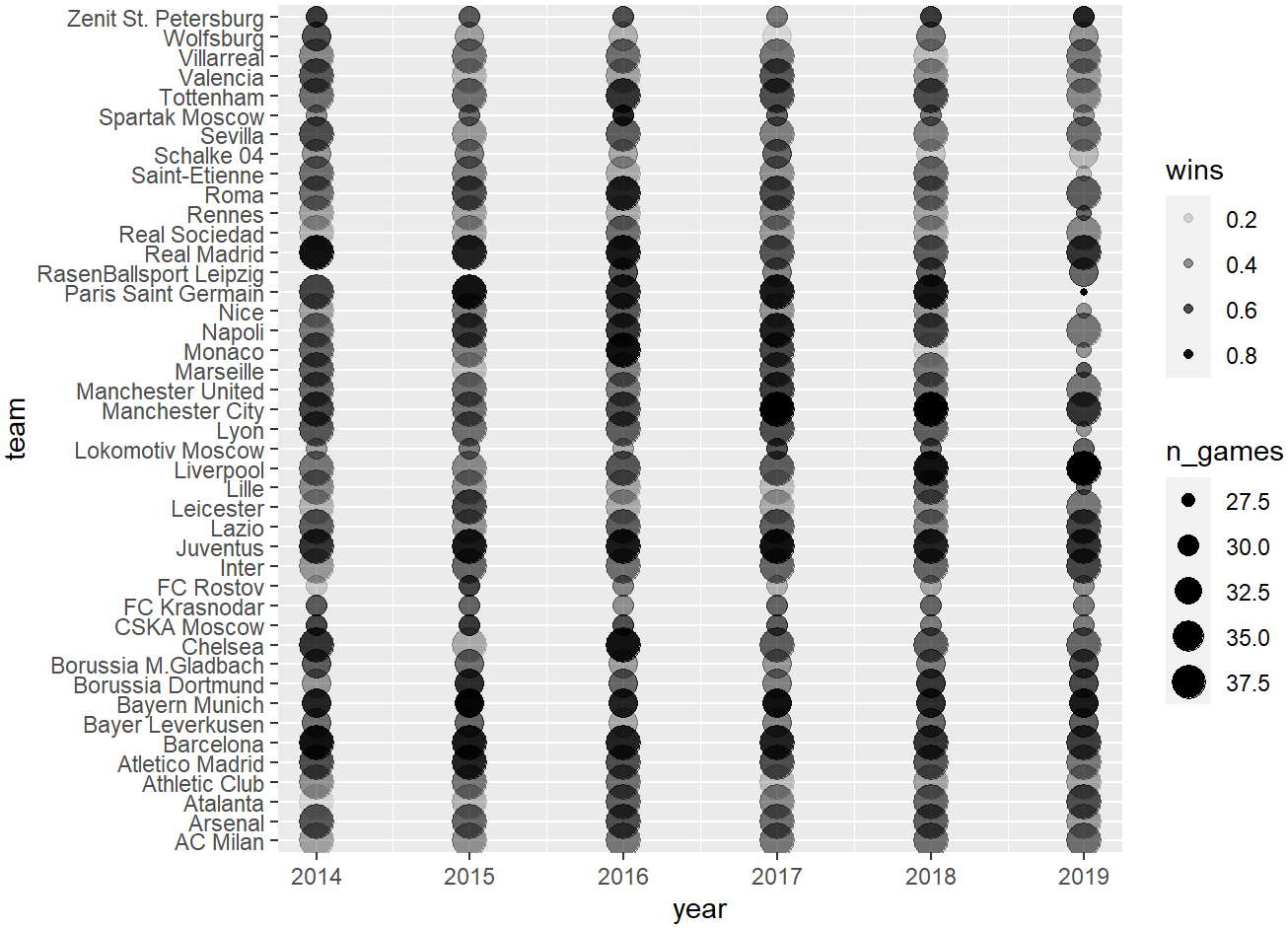
Soccer Code Review

Code review: #1: It might be better to move the ungroup() function after the summarise() to decrease the risk of forgetting this step. #2: The plotting function can follow right after the best team selection to make the code more compact #3: It is good to have the lengend of wins and n_games

```
#calculate the winning rate grouping by team and year
win_props <- read_csv("https://raw.githubusercontent.com/krisrs1128/stat479_s22/main/exercise
s/data/understat_per_game.csv") %>%
  group_by(team, year) %>%
  summarise(n_games = n(), wins = sum(wins) / n_games)
```

```
## Rows: 24580 Columns: 29
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (4): league, h_a, result, team
## dbl  (24): year, xG, xGA, npxG, npxGA, deep, deep_allowed, scored, missed, x...
## dttm  (1): date
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## `summarise()` has grouped output by 'team'. You can override using the `.groups` argument.
```

```
#Select top 20% of the wins
best_teams <- win_props %>%
  ungroup() %>%
  slice_max(wins, prop = 0.2) %>%
  pull(team)


win_props %>%
  filter(team %in% best_teams) %>%
  ggplot() +
  geom_point(aes(year, team, size = n_games, alpha = wins))
```



Modifing the soccer problem code to make it more concise

#Recommendation #1: Since we want to know the the top wins in the last few years, we could decrease the number of prop in slice_max().

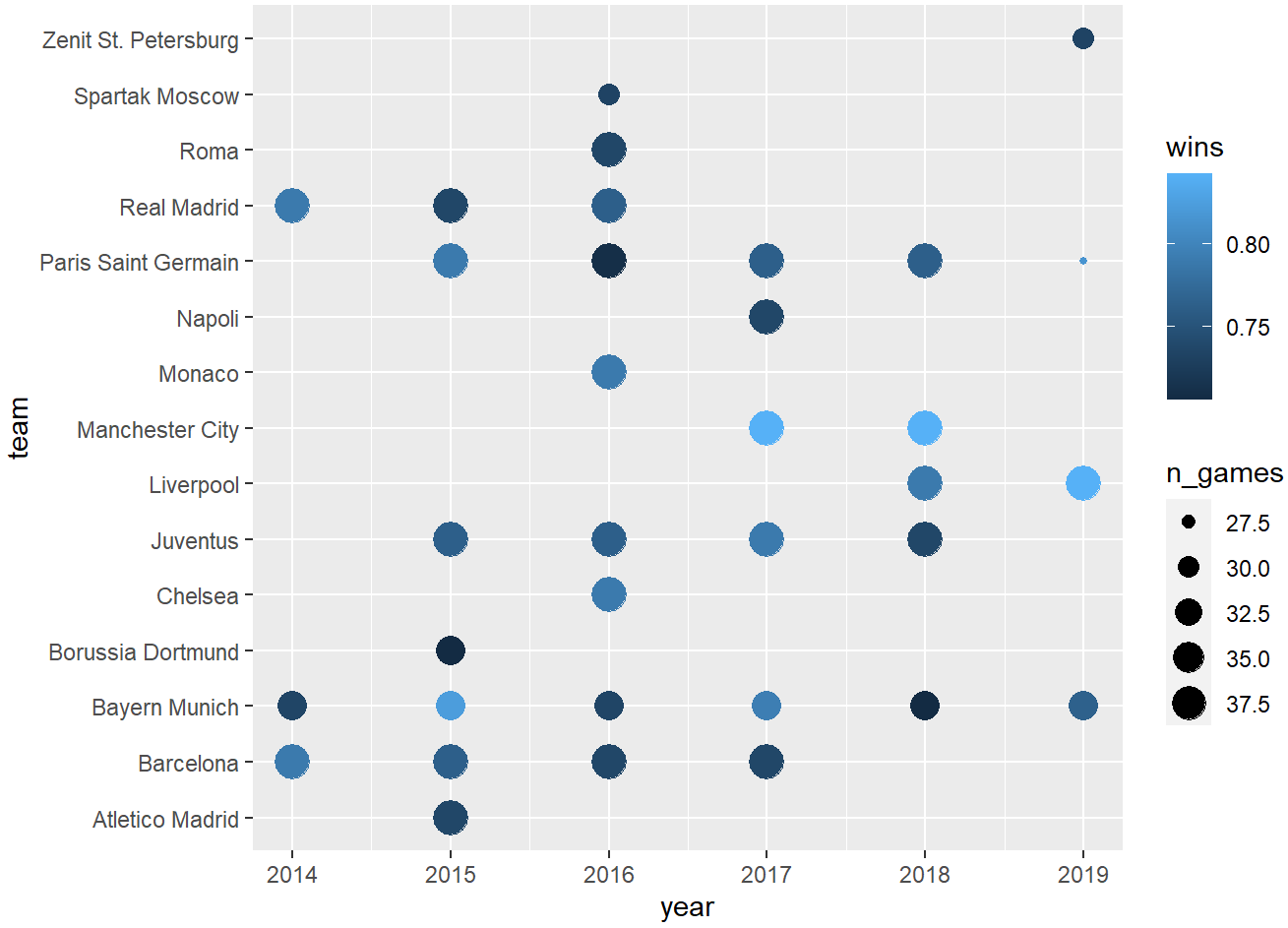
#Recommendation #2 (Implemented): We can also replace the alpha with color so it might be better for visualization.

#Recommendation #3 (I can't figure it out): use facet_wrap to separate by teams, where each plot has year as its x-axis and n_games as y-axis.

```
win_props <- read_csv("https://raw.githubusercontent.com/krisrs1128/stat479_s22/main/exercises/data/understat_per_game.csv") %>%
  group_by(team, year) %>%
  summarise(n_games = n(), wins = sum(wins) / n_games) %>%
  ungroup()
```

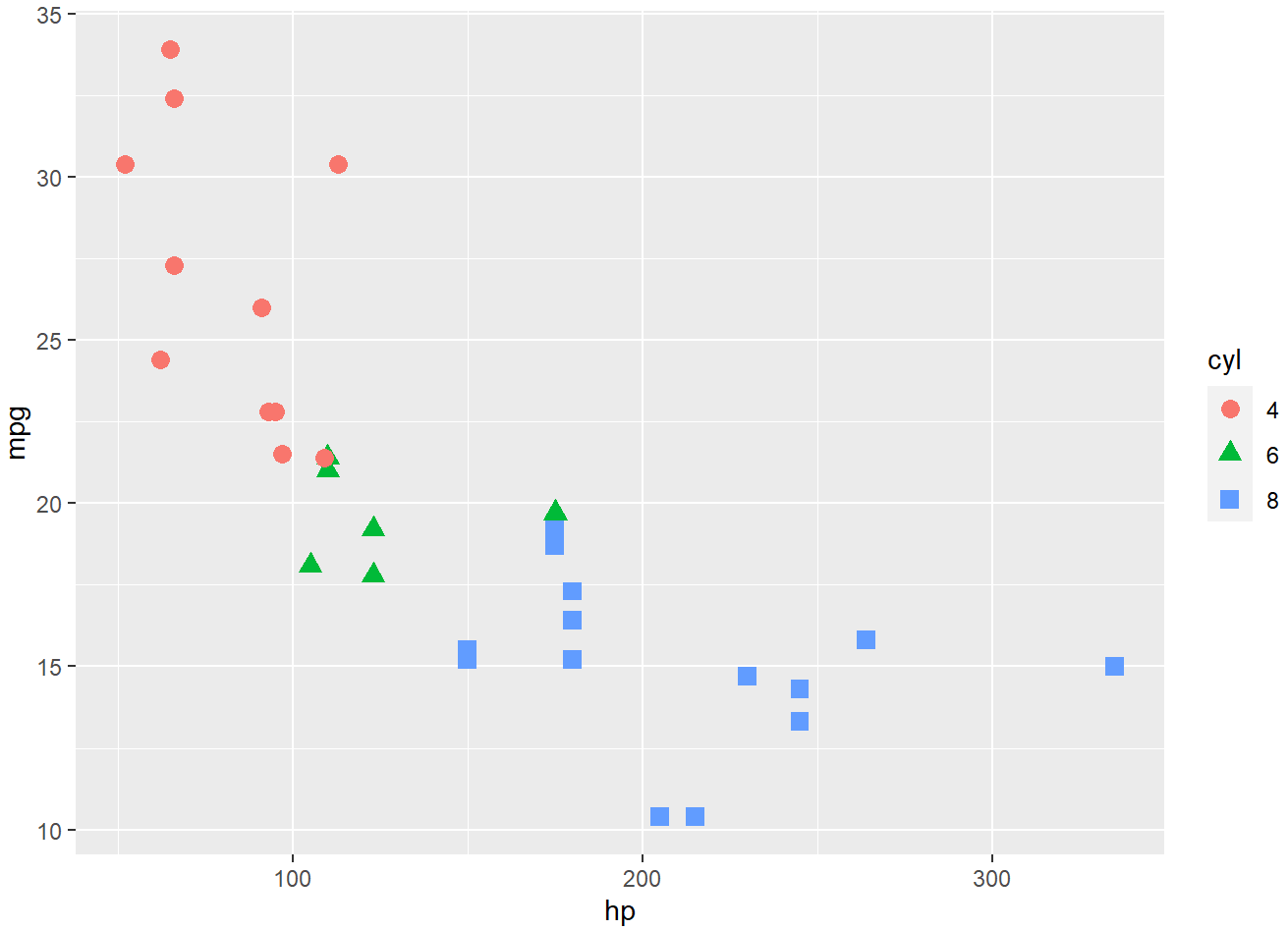
```
## Rows: 24580 Columns: 29
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (4): league, h_a, result, team
## dbl  (24): year, xG, xGA, npxG, npxGA, deep, deep_allowed, scored, missed, x...
## dttm  (1): date
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## `summarise()` has grouped output by 'team'. You can override using the `.groups` argument.
```

```
# Replace the alpha with color parameter and the prop parameter in slice_max
ggp <- win_props %>%
  slice_max(wins, prop = 0.05) %>%
  ggplot() +
  geom_point(aes(year, team, size = n_games, color = wins))

# Trying to implement recommendation #3 but can't
# ggp <- win_props %>%
#   slice_max(wins, prop = 0.2) %>%
#   mutate(fct_reorder(wins))
#   ggplot() +
#   geom_col(aes(team,n_games,alpha = wins))+
#   theme(axis.text.x = element_text(angle = 45, vjust = 0.5, hjust=1))

ggp
```



Visual Redesign

```
library(shiny)
#A
mtcars$cyl <- as.factor(mtcars$cyl)
ggplot(mtcars, aes(x=hp, y=mpg, color=cyl, shape=cyl)) +
    geom_point(size=3)
```


35
30
25
20
15
10
mpg
100
200
300
hp
cyl
4
6
8

```
#B
# This plot reveal the relationship between mpg and hp, and the cylinder number has been conv
erted to factor and shaped. The take away matched the expectation. One design that might be h
ard to implement is to change the comparison features. For instance, if I want to find out th
e relationship between weight and hp, I will have to go back to the code and replace them man
ually.



#C
# The original plot is pretty easy to interpret where we can see the relationship between the
two features we are interested in.



#D
#I would like to apply shiny to improve the variability of this plot. Which I could choose an
y two different features and use the scatter plot to see their relationship, as well as addin
g a hue
scatterplot <- function( x_feature, y_feature, hue){
  #mtcars$cyl <- as.factor(mtcars$cyl)
  ggplot(mtcars, aes_string(x=x_feature, y=y_feature, color=hue, shape = as.factor(mtcars[,'c
yl']))) +
    geom_point(size=3)
}

ui  = fluidPage(
  fluidRow(
    column(width = 4, selectInput('dropdown_x', 'Select x', choice = colnames(mtcars))),
    column(width = 4, selectInput('dropdown_y', 'Select y', choices = colnames(mtcars))),
    column(width = 4, selectInput('dropdown_hue', 'Select hue', choice = colnames(mtcars)))
  ),


  # selectInput('dropdown_x', 'Select x', choice = colnames(mtcars)),
  # selectInput('dropdown_y', 'Select y', choices = colnames(mtcars)),
  # selectInput('dropdown_hue', 'Select hue', choice = colnames(mtcars)),
  plotOutput('scatterplot')
)

server = function(input, output){
  output$scatterplot = renderPlot({
    scatterplot(input$dropdown_x, input$dropdown_y, input$dropdown_hue)
  })
}

shinyApp(ui, server)
```

Select x

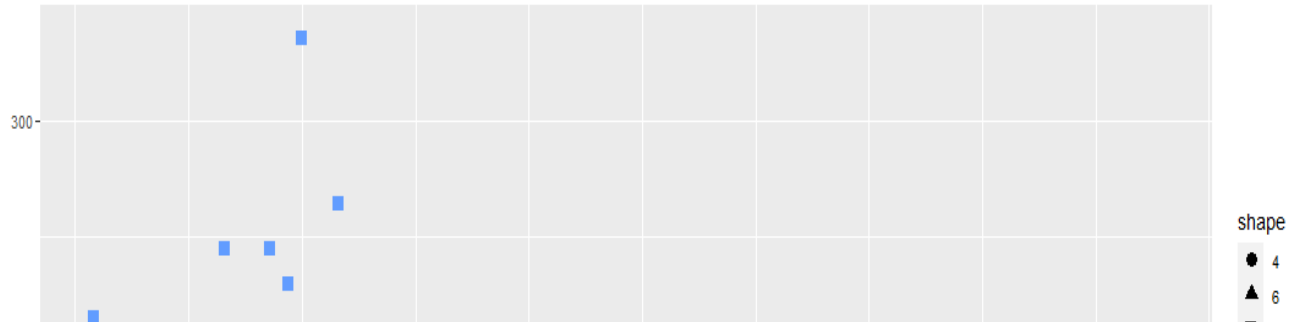
mpg

Select y

hp

Select hue

cyl



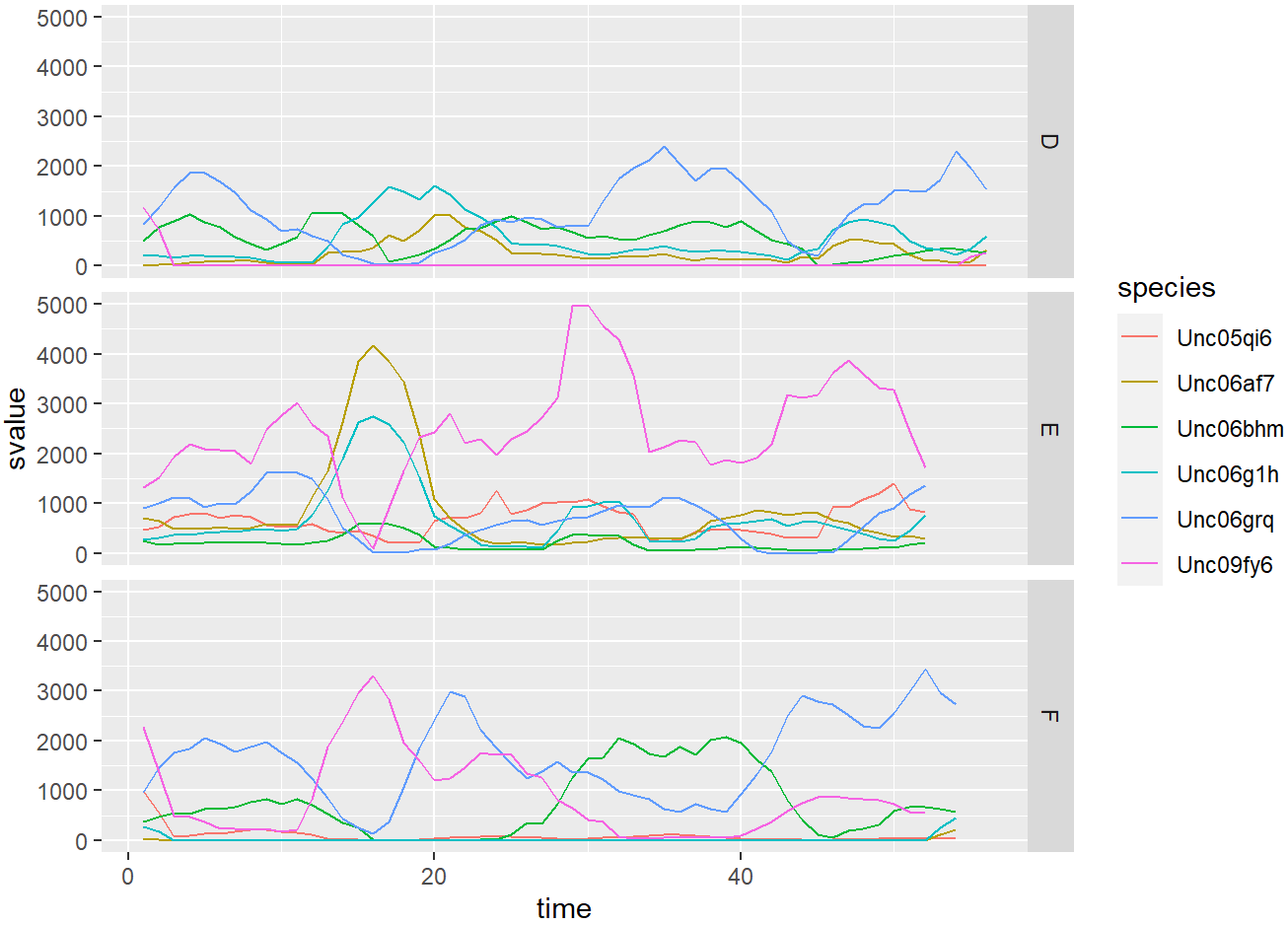
Antibiotics Comparison

```
antibiotic <- read_csv("https://uwmadison.box.com/shared/static/5jmd9pku62291ek20lioevsw1c588
ahx.csv")
```

```
## Rows: 972 Columns: 7
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): species, sample, ind, antibiotic
## dbl (3): value, time, svalue
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
#antibiotic

#Approach 3
ggplot()+
  geom_line(antibiotic, mapping = aes(time,  svalue, color = species))+
  facet_grid(ind~.)
```

```
## Warning: Removed 4 row(s) containing missing values (geom_path).
```



Name App Bugs (A)

Error #1: In the server function, we can't assign variable to input because it is for read only. To rewrite, we have: output\(printed_name <- renderText({paste0("Hello ", input\)name, "!")})

```
library(shiny)
ui <- fluidPage(
  titlePanel("Hello!"),
  textInput("name", "Enter your name"),
  textOutput("printed_name")
)

server <- function(input, output) {
  #Original code (Not runnable)
  # input$name <- paste0("Welcome to shiny, ", input$name, "!")
  # output$printed_names <- renderText({ input$name })

  #Modified code (runnable)
  output$printed_name <- renderText({paste0("Hello ", input$name, "!")})
}
app <- shinyApp(ui, server)
app
```

Hello!

Enter your name

shun

Hello shun!

Name App Bugs (B) Error #1: The name of the output function has an extra ‘s’

```
library(shiny)
ui <- fluidPage(
  titlePanel("Hello!"),
  textInput("name", "Enter your name"),
  textOutput("printed_name")
)

server <- function(input, output) {
  #Original code(Not showing the right content)
  # output$printed_names <- renderText({
  #   paste0("Welcome to shiny, ", input$name, "!")
  # })

  #Modified code(runnable)
  output$printed_name <- renderText({
    paste0("Welcome to shiny, ", input$name, "!")
  })
}

app <- shinyApp(ui, server)
app
```

Hello!

Enter your name

Shunyi

Welcome to shiny, Shunyi!

Name App Bugs (C) Error #1: the output in the server function used the wrong format, the correct one should be renderText

```
library(shiny)
ui <- fluidPage(
  titlePanel("Hello!"),
  textInput("name", "Enter your name"),
  textOutput("printed_name")
)

server <- function(input, output) {
  #Original code (not runnable)
  # output$printed_name <- renderDataTable({
  #   paste0("Welcome to shiny, ", input$name, "!")
  # })

  #Modefied code (runnable)
  output$printed_name <- renderText({
    paste0("Welcome to shiny, ", input$name, "!")
  })
}
app <- shinyApp(ui, server)
app
```

Hello!

Enter your name

Shunyi Huang

Welcome to shiny, Shunyi Huang!

Name App Bugs (D) Error #1: no comma to separate each panel in the ui function

```
library(shiny)
ui <- fluidPage(
  #Original code (not runnable)
  # titlePanel("Hello!")
  # textInput("name", "Enter your name")
  # textOutput("printed_name")

  #Modified code(runnable)
  titlePanel("Hello!"),
  textInput("name", "Enter your name"),
  textOutput("printed_name")
)


server <- function(input, output) {
  output$printed_name <- renderText({
    paste0("Welcome to shiny, ", input$name, "!")
})
}
app <- shinyApp(ui, server)
```