241111

Hana Kwon

2024-11-14

“‘{#==========
#====== *Nov 11, 2024* ==========
#==========
#====== *Code Revision After Meeting* ==========
#==========

DATA PREPARATION: PREPROCESSING

Load necessary libraries

library(dplyr)

Load the data

Read the file as a tab-separated file

I changed the file path to our DropBox folder name. Please check if it works right! data <- read.table(“~/Columbia Dropbox/Kwon Hana/Prof. W.Bentley MacLeod & Hana Kwon/Data/Embrey_2018a_new_data.tx
header = TRUE, sep = “”, stringsAsFactors = FALSE)

Before the arranging, we can check the structure of the data to verify column names and data types

head(data) str(data) View(data) # View the data in a table format

#* Arrange data ==========

#========== *1. Arrange columns* ==========
Sort data by session, player ID, supergame, and round sorted_data <- data %>% arrange(session, id, supergame, round) # Reorder columns to place ‘session’ at the front sorted_data <- sorted_data %>% select(session, everything()) # Check if the columns are arranged as intended View(sorted_data)

#========== *2. Add variables/columns* ==========

#========== *2.1. Add opponent’s cooperation variable (ocoop)* ==========
Add the opponent’s cooperation info (ocoop) by joining the data to itself on id and oid #** This way, each player’s row gets matched with the correct opponent’s cooperation data for every round in the same game sorted_data <- sorted_data %>% left_join(sorted_data %>% select(session, supergame, round, id

= oid, ocoop = coop), # Selects opponent's cooperation (coop) for matching by = c("session", "supergame", "round", "id") # Joins on session, supergame, round, and id to map correct opponent data)

#========== *2.2. Add Always Cooperate (AC) and Always Defect (AD) strategy variables* ==========# # AC: Always cooperate in all rounds # AD: Always defect in all rounds sorted_data <- sorted_data %>% mutate(AC = ifelse(coop == 1, 1, 0), # 1 if the player cooperates in all rounds AD = ifelse(coop == 0, 1, 0) # 1 if the player defects in all rounds)

#========== *2.3. Add tit-for-tat (TFT) strategy variable* ==========
TFT starts with cooperation in the first round and mimics the opponent's previous action in subsequent rounds sorted_data <- sorted_data %>% mutate(TFT = ifelse(round == 1, 1, lag(ocoop)))

#========== *2.4. Add Grim (G) Strategy variables (G0, G1, G2, G3)* ==========
Add Grim strategy variables (G0, G1, G2, G3) based on the opponent's cooperation history # G0 cooperates in the first round and defects if the opponent defected in the previous round # G1 defects in the last round; otherwise, it cooperates unless G0 or the opponent defected in the previous round # G2 defects in the last two rounds; otherwise, it cooperates unless G1 or the opponent defected in the previous round # G3 defects in the last three rounds; otherwise, it cooperates unless G2 or the opponent defected in the previous round sorted_data <- sorted_data %>% mutate(G0 = ifelse(round == 1, 1, ifelse(lag(ocoop) == 0, 0, 1)), G1 = ifelse(round == horizon, 0, ifelse(lag(G0) == 0 | lag(ocoop) == 0, 0, 1)), G2 = ifelse(round >= horizon - 1, 0, ifelse(lag(G1) == 0 | lag(ocoop) == 0, 0, 1)), G3 = ifelse(round >= horizon - 2, 0, ifelse(lag(G2) == 0 | lag(ocoop) == 0, 0, 1)))

#========== *2.5. Add payoff calculation* ==========
Assuming that cooperation leads to a certain payoff, we calculate the payoff based on coop and ocoop values. # Customize this formula based on the specific payoff rules of your game. sorted_data <- sorted_data %>% mutate(payoff = case_when(coop == 1 & ocoop == 1 ~ r, # Both cooperate coop == 1 & ocoop == 0 ~ s, # Player cooperates, opponent defects coop == 0 & ocoop == 1 ~ t, # Player defects, opponent cooperates coop == 0 & ocoop == 0 ~ p # Both defect))

#========== *2.6. Add Agreement(Consistency) Check Variables (G1A and TFTA)* ==========# # ACA: Check if AC (Always Cooperate) strategy matches the actual cooperation (coop) behavior # ADA: Check if AD (Always Defect) strategy matches the actual cooperation (coop) behavior # G0A, G1A, G2A, G3A: Check if Grim strategies (G0, G1, G2, G3) match the actual cooperation behavior # TFTA: Check if TFT strategy matches the actual cooperation (coop) behavior sorted_data <- sorted_data %>% mutate(ACA = ifelse(coop == 1, 1, 0), # 1 if Always Cooperate strategy is followed ADA = ifelse(coop == 0, 1, 0), # 1 if Always Defect strategy is followed G0A = ifelse(G0 == coop, 1, 0), # 1 if G0 matches actual behavior G1A = ifelse(G1 == coop, 1, 0), # 1 if G1 matches actual behavior G2A = ifelse(G2 == coop, 1, 0), # 1 if G2 matches actual behavior G3A = ifelse(G3 == coop, 1, 0), # 1 if G3 matches actual behavior TFTA = ifelse(TFT == coop, 1, 0) # 1 if TFT matches actual behavior)

#========== *2.7: Filter out cases where both id and oid are Always Cooperate (AC)* ==========# ### Nov 11: Updated Part # Join the data on id and oid to identify rounds where both players are AC filtered_data <- sorted_data %>% left_join(sorted_data %>% select(session, supergame, round, id, AC) %>% rename(opponent_id = id, opponent_AC = AC), by = c("session", "supergame", "round", "oid" = "opponent_id")) %>% filter(!(AC == 1 & opponent_AC == 1)) # Remove cases where both id and oid are AC

#========== *3. Collapse Data to Game-Level* ==========
Now use `filtered_data` in place of `sorted_data` for further analysis # Collapse Data to Game-Level with filtered data collapsed_data <- filtered_data %>% group_by(session, id, supergame) %>% summarise(mean_ACA = mean(ACA, na.rm = TRUE), mean_ADA = mean(ADA, na.rm = TRUE), mean_G0A = mean(G0A, na.rm = TRUE), mean_G1A = mean(G1A, na.rm = TRUE), mean_G2A = mean(G2A, na.rm = TRUE), mean_G3A = mean(G3A, na.rm = TRUE), mean_TFTA = mean(TFTA, na.rm = TRUE), avg_payoff = mean(payoff, na.rm = TRUE), .groups = "drop")

#========== *4. Summary Table* ==========
Summary Table with filtered data summary_table <- collapsed_data %>% summarise(min_ACA

= min(mean_ACA, na.rm = TRUE), max_ACA = max(mean_ACA, na.rm = TRUE), avg_ACA = mean(mean_ACA, na.rm = TRUE), avg_payoff_ACA_1 = mean(avg_payoff[mean_ACA == 1], na.rm = TRUE), # avg payoff when ACA is fully followed

```
min_ADA = min(mean_ADA, na.rm = TRUE),
max_ADA = max(mean_ADA, na.rm = TRUE),
avg_ADA = mean(mean_ADA, na.rm = TRUE),
avg_payoff_ADA_1 = mean(avg_payoff[mean_ADA == 1], na.rm = TRUE),

min_G1A = min(mean_G1A, na.rm = TRUE),
max_G1A = max(mean_G1A, na.rm = TRUE),
avg_G1A = mean(mean_G1A, na.rm = TRUE),
avg_payoff_G1A_1 = mean(avg_payoff[mean_G1A == 1], na.rm = TRUE),

min_G2A = min(mean_G2A, na.rm = TRUE),
max_G2A = max(mean_G2A, na.rm = TRUE),
avg_G2A = mean(mean_G2A, na.rm = TRUE),
avg_payoff_G2A_1 = mean(avg_payoff[mean_G2A == 1], na.rm = TRUE),

min_G3A = min(mean_G3A, na.rm = TRUE),
max_G3A = max(mean_G3A, na.rm = TRUE),
avg_G3A = mean(mean_G3A, na.rm = TRUE),
avg_payoff_G3A_1 = mean(avg_payoff[mean_G3A == 1], na.rm = TRUE),

min_TFTA = min(mean_TFTA, na.rm = TRUE),
max_TFTA = max(mean_TFTA, na.rm = TRUE),
avg_TFTA = mean(mean_TFTA, na.rm = TRUE),
avg_payoff_TFTA_1 = mean(avg_payoff[mean_TFTA == 1], na.rm = TRUE)
```

)

View the updated summary table

print(summary_table) View(summary_table)

#============== 5. Display and Verify Key Summary Statistics ==========
Display key summary statistics to confirm data structure and verify sorting summary(sorted_data) # Summary statistics for sorted_data head(sorted_data) # First few rows to visually confirm sorting str(sorted_data) # Structure of sorted_data to check variable types View(sorted_data) # View data in a table format for a final verification "'