IntelliR statistical pipeline is designed to be used together with the three proposed challenges. However, it can be easily adapted to run specific variables with any other IntelliCage design.

Here, we present an example on how to adapt the provided pipeline using the IntelliCage dataset from Wilke, J.B.H., Hindermann, M., et al., 2021 (https://doi.org/10.1038/s41380-021-01238-3). This study employed IntelliCage-based phenotyping using a different challenge configuration, including a sucrose preference paradigm. To show IntelliR's flexibility, the sucrose preference challenge will be used as our example in this small tutorial. **All scripts shown here are available in the "tutorial" folder of IntelliR**.

In this dataset, two cages were used simultaneously, so the datasets have to be merged. For this dataset, each individual day has its own folder, so the first step is to name them accordingly. Each set of cages was placed in its own folder (Sucrose1, Sucrose2) and to simplify the process, the internal IntelliCage folder was renamed Challenge_1. All other folders do not need to be changed. The structure is available below in Figure 1.

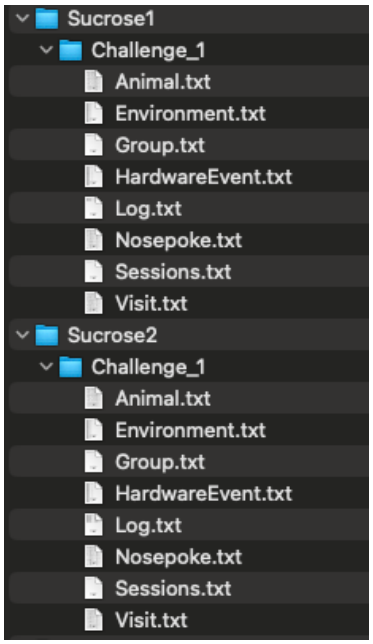

Figure 1 - Folder structure for merging the files.

The next step is to open the merging_intellicages script (matching your operational system). Run the code up to line 27, loading the files for the first cage. Then, jump to the "Second set" and repeat. The lines you need to keep are presented in Figure 2.

```
#### Loading first set of cages ####
## Select your Visit.txt file for Challenge 1
file.choose() -> project1

#### Loading second set of cages ####
## Select your Visit.txt file for Challenge 1
file.choose() -> project2p

#### What should be the new name of the folder with the merged cages? ####
project_merged <- "Sucrose"

#### Run everything below at once ####
project_folder <- gsub("(.*/)([^/]*)/Challenge_1.*", "\\1", project1)
project_folder <- gsub("/$", "", project_folder)
project <- gsub(".*/([^/]*)/Challenge_1.*", "\\1", project1)

#### Challenge 1 ####
read.delim(paste(project_folder,"/",project,"/","Challenge_1/Visit.txt",sep="")) -> C1_visits
paste(C1_visits$VisitID,"_set1",sep = "") -> C1_visits$VisitID
read.delim(paste(project_folder,"/",project,"/","Challenge_1/Nosepoke.txt",sep="")) -> C1_nosepokes
paste(C1_nosepokes$VisitID,"_set1",sep = "") -> C1_nosepokes$VisitID
```

Figure 2 - Initial lines of the merging_intellicages script.

Afterwards, it might be needed to adjust the dates in case you ran the experiments separately. In this example, the challenge ran for two days, so two corrections might be needed (Figure 3).

```
# How many dates appear after the command? These are the number of times you need to run the lines below. Here you will find what is ne
one day. You can comment out the additional days as needed and more if required.

C1_visits2[which(C1_visits2$StartDate == unique(C1_visits2$StartDate)[1]),"StartDate"] <- unique(C1_visits$StartDate)[1]
C1_visits2[which(C1_visits2$StartDate == unique(C1_visits2$StartDate)[2]),"StartDate"] <- unique(C1_visits$StartDate)[2]
#C1_visits2[which(C1_visits2$StartDate == unique(C1_visits2$StartDate)[3]),"StartDate"] <- unique(C1_visits$StartDate)[3]
#C1_visits2[which(C1_visits2$StartDate == unique(C1_visits2$StartDate)[4]),"StartDate"] <- unique(C1_visits$StartDate)[4]
#C1_visits2[which(C1_visits2$StartDate == unique(C1_visits2$StartDate)[5]),"StartDate"] <- unique(C1_visits$StartDate)[5]

C1_nosepokes2[which(C1_nosepokes2$StartDate == unique(C1_nosepokes2$StartDate)[1]),"StartDate"] <- unique(C1_nosepokes$StartDate)[1]
C1_nosepokes2[which(C1_nosepokes2$StartDate == unique(C1_nosepokes2$StartDate)[2]),"StartDate"] <- unique(C1_nosepokes$StartDate)[2]
#C1_nosepokes2[which(C1_nosepokes2$StartDate == unique(C1_nosepokes2$StartDate)[3]),"StartDate"] <- unique(C1_nosepokes$StartDate)[3]
#C1_nosepokes2[which(C1_nosepokes2$StartDate == unique(C1_nosepokes2$StartDate)[4]),"StartDate"] <- unique(C1_nosepokes$StartDate)[4]
#C1_nosepokes2[which(C1_nosepokes2$StartDate == unique(C1_nosepokes2$StartDate)[5]),"StartDate"] <- unique(C1_nosepokes$StartDate)[5]

56:1  In the case the experiments ran separately, it is necessary to adjust the dates to ensure the analysis runs properly

Console  Background Jobs
R 4.3.0 · ~/Documents/ownCloud/GitHub/IntelliR/tutorial/
> paste(C1_nosepokes2$VisitID,"_set2",sep = "") -> C1_nosepokes2$VisitID
> #### In the case the experiments ran separately, it is necessary to adjust the dates to ensure the analysis runs properly ####
> unique(C1_visits2$StartDate)
[1] "2020-03-16" "2020-03-17"
```

Figure 3 - Initial lines of the merging_intellicages script.

Following the change to the dates, you can proceed with the script, binding the files together for the rest of the analysis. The script "merging_intellicages_example.R" covers what was done in the example above.

After saving the files, we need to set up the variables and load the packages for IntelliR. The shiny interface requires that you follow the challenge structure proposed in this paper, so you need to manually set it up for other challenges. First, open "initial_intellicages.R" and run the code from the beginning (Figure 4).

```r
#### IntelliCage data processing ####
## Author: Vinicius Daguano Gastaldi
## This script is required to start the file processing. A new script will be called depending on the operational system.
# Modified: 13.08.2024
#### Required packages ####
if (grepl("Windows", Sys.info()["sysname"])) {
  required_packages <- c("dplyr","openxlsx","ggplot2","reshape2","tcltk","multcomp","coin","dunn.test","car","tidyr","effectsize","emmeans","boot","rstatix"
,"conover.test")
  new_packages <- required_packages[!(required_packages %in% installed.packages()[,"Package"])]
  if(length(new_packages)) install.packages(new_packages)
  if (packageVersion("ggplot2") < "3.4.2") {
    install.packages("ggplot2")
  }
  library(openxlsx)
  library(ggplot2)
  library(reshape2)
  library(multcomp)
  library(coin)
  library(dunn.test)
  library(car)
  library(dplyr)
  library(tcltk)
  library(shiny)
  library(tidyr)
  library(effectsize)
  library(emmeans)
  library(boot)
  library(rstatix)
  library(conover.test)
  rm(new_packages,required_packages)
} else {
  required_packages <- c("dplyr","openxlsx","ggplot2","reshape2","tcltk","multcomp","coin","dunn.test","car","tidyr","effectsize","emmeans","boot","rstatix"
```

Figure 4 - Initial lines of the initial_intellicages script.

This should load (and install if needed) all required packages. If a shiny webpage opens up, please close it and stop the script using ESC (Figure 5).

```
R 4.3.0 · ~/
+       project_folder
+     })
+
+     output$results_folder_path <- renderText({
+       results_folder
+     })
+
+     observeEvent(c(input$enter_project, input$project_folder, input$results_folder), {
+       if (!is.null(project) && nchar(project) > 0 && !is.null(project_folder) && !is.null(results_folder)) {
+         stopApp()
+         session$close()
+       }
+     })
+   }
+   runApp(shinyApp(ui = ui, server = server, options = list(launch.browser = TRUE)))
+ }

Listening on http://127.0.0.1:4401
```

Figure 5 - R Studio console when running the whole initial_intellicages script.

Then open the “set_up_IC.R” that matches your operational system (the one included in the tutorial folder matches Unix) and load your unblinding file, write your group names as in the unblinding file, choose your colors, and assign paths to the project and results. Figure 6 shows the script ready to run the example files.

```r
#### Manual setup for IntelliR ####
# The lines below give the necessary variables for IntelliR to work
read.delim(file.choose()) -> unblinding
group1_name <- "A"
group2_name <- "B"
group3_name <- "C"
group4_name <- "D"
group1_color <- "red"
group2_color <- "blue"
group1_color <- "green"
group2_color <- "purple"
project <- "Sucrose" # name of the folder where your files are
project_folder <- "/Users/gastaldi/Documents/ownCloud/GitHub/IntelliR/tutorial/data_files" # put your path here
results_folder <- "/Users/gastaldi/Documents/ownCloud/GitHub/IntelliR/tutorial/data_files/results" # put your path here
```

Figure 6 - set_up_IC script ready to run the tutorial example.

With this you are ready to start your analysis. Open the appropriate "processing_intellicages" script according to your operational system. Here we can skip loading files for Challenges 2 and 3, but all other steps until line 98 should be followed. On line 88 (Figure 7), you can define the times where the light setting of your room changes. It is currently configured for 12 hours of light, from 6h00 to 17h59.

```
84   # We need to differentiate between Light and Dark for several of the measurements
85   nosepokes_df$Light_Status <- NA
86
87   nosepokes_df <- nosepokes_df %>% mutate(Light_Status = case_when(
88     (Cage == 1 | Cage == 2) & (Start_Hour >= 6 & Start_Hour <= 17) ~ "Light"))
89
90   nosepokes_df[is.na(nosepokes_df$Light_Status),ncol(nosepokes_df)] <- "Dark"
```

Figure 7 – Defining light and dark cycles.

From lines 100 to 159 (Figure 8), IntelliR divides the original challenges into their subchallenges (e.g., Place Learning, Reversal Learning, Multiple Reversal Learning for Challenge 1), which can be skipped in this example.

```
100 #### Dividing into the individual challenges ####
101 rbind(C1_visits[which(C1_visits$StartDate == unique(C1_visits$StartDate)[1]),],C1_visits[which(C1_visits$StartDate == unique(C1_visits$StartDate)[2] &
    C1_visits$Start_Hour <= 5),]) -> Habituation_visits
102
103 rbind(C1_visits[which(C1_visits$StartDate == unique(C1_visits$StartDate)[2] & C1_visits$Start_Hour >= 6),],C1_visits[which(C1_visits$StartDate == unique
    (C1_visits$StartDate)[3] & C1_visits$Start_Hour <= 5),]) -> PL_visits
104
105 rbind(C1_visits[which(C1_visits$StartDate == unique(C1_visits$StartDate)[3] & C1_visits$Start_Hour >= 6),],C1_visits[which(C1_visits$StartDate == unique
    (C1_visits$StartDate)[4] & C1_visits$Start_Hour <= 5),]) -> RL_visits
106
107 rbind(C1_visits[which(C1_visits$StartDate == unique(C1_visits$StartDate)[4] & C1_visits$Start_Hour >= 6),],C1_visits[which(C1_visits$StartDate == unique
    (C1_visits$StartDate)[5] & C1_visits$Start_Hour <= 5),]) -> MRL_visits
```

Figure 8 – Dividing the input files into multiple challenges.

After this stage, all general variables can be calculated for any dataset. Specific ones such as Challenge Error, Time Error, and Direction Error cannot be used without planning. That is, it requires that your design shares the idea behind these measurements as explained in the main manuscript. As the majority of the calculated errors are not available, the Cognition Index (and the Overall Error Index) cannot be calculated.

From line 161 (Figure 9), IntelliR starts to calculate variables, but after skipping lines 100 to 159 you will have two objects loaded, C1_visits and C1_nosepokes. To avoid issues, you need to comment out line 165:

```
161 #### Getting total visits, nosepokes, and licks ####
162 # Get the names of all objects that will be used for these variables
163 visits_objects <- ls(pattern = "_visits$")
164 # Reorder to follow the challenges
165 visits_objects <- visits_objects[c(15,17,18,16,13,6,5,8,7,10,9,12,11,14,3,4,1,2)]
166 visits_objects[!is.na(visits_objects)] -> visits_objects
167 # Create a vector of column names for the results data frame
168 column_names <- c("Animal", paste0(rep(gsub("_visits","",visits_objects), each = 3), c("_Total_Licks","_Total_Visits", "_Total_Nosepokes")))
169
170 # Create the results data frame
171 results <- setNames(data.frame(matrix(ncol = length(column_names), nrow = 0)), column_names)
```

Figure 9 – Example variable where it is demonstrated one line of code that needs extra attention.

You either need to do this for all lines that do this (e.g., lines 211 and 257; Figure 10) or simply skip it when running the code.

```
#### Getting total visits, nosepokes, and licks ####
# Get the names of all objects that will be used for these variables
visits_objects <- ls(pattern = "_visits$")
# Reorder to follow the challenges
#visits_objects <- visits_objects[c(15,17,18,16,13,6,5,8,7,10,9,12,11,14,3,4,1,2)]
visits_objects[!is.na(visits_objects)] -> visits_objects
# Create a vector of column names for the results data frame
column_names <- c("Animal", paste0(rep(gsub("_visits","",visits_objects), each = 3), c("_Total_Licks","_Total_Visits", "_Total_Nosepokes")))

# Create the results data frame
results <- setNames(data.frame(matrix(ncol = length(column_names), nrow = 0)), column_names)
```

Figure 10 – The line referred to in Figure 9 after being commented out.

This command reorganizes the objects to calculate the table in the order in which the experiments occurred and is not necessary if you do not have multiple challenges.

The script processing_intellicages starts by calculating the variables total licks, total visits, and total nose pokes. All calculated variables are saved in the object "results_df", which is then used to calculate statistics (Figure 11) starting from line 1966. It supports 2 to 4 groups and can be used for any type of variable, with the only requisite being that they follow the standard format from the object.

```
#### Statistical Analysis ####
# The IntelliR script currently supports analysis for 2, 3 or 4 groups
# It detects how many groups are being used through the variables group"NUMBER"_name
# The script doesn't require any special formatting and can be applied to different variables without issues
# You only need the table to start with a column named ID and second named Group.

if(length(grep("^group[0-9]+_name$", ls(envir = globalenv()))) == 2){
  # Create a vector of column names for the results data frame
  column_names <- c("Variable",paste("Number_Animals_",group1_name,sep = ""),paste("Number_Animals_",group2_name,sep = ""),paste("Mean_",group1_name,sep = ""),paste("SD_",group1_name,sep = ""),paste("Mean_",group2_name,sep = ""),paste("SD_",group2_name,sep = ""),paste("Shapiro_",group1_name,sep = ""),paste("Shapiro_",group2_name,sep = ""),"Levene","Statistical test","p-value","Statistic name","Statistic Value","Effect size test","Effect size","CI")
  factor(results_df[,"Group"],levels = c(group1_name,group2_name)) -> group
  factor(results_df[,"Group"],levels = c(group1_name,group2_name)) -> results_df[,"Group"]
  # Create the results data frame
  results_stats <- setNames(data.frame(matrix(ncol = length(column_names), nrow = 0)), column_names)
  for (variable in 3:ncol(results_df)){
```

Figure 11 – Start of the statistical analysis section of IntelliR.

After general statistics are calculated, IntelliR also calculates simple p-values for corner preference (Figure 12). The columns used are stated on line 2886, but they would change according to your challenges. Please be aware of this in case you are interested in the calculation of statistics for your challenges.

```
#### Statistics for corner preference ####
corner_stats <- NULL

for (variable in seq(3,110,4)){
  results_corner[,c(1,2,variable,variable+1,variable+2,variable+3)] -> subset_corner
  colnames(subset_corner)[3:6] <- c("Corner1","Corner2","Corner3","Corner4")
  if(sum(is.na(subset_corner[,3])) == nrow(subset_corner)){
    next()
  } else {
    df_long <- pivot_longer(subset_corner, cols = c("Corner1", "Corner2", "Corner3", "Corner4"), names_to = "Corner", values_to = "Proportion")
    df_long %>%
      group_by(Group) %>%
      summarise(p.value = friedman.test(Proportion ~ Corner | ID)$p.value) -> hold_corner_stats
    hold_corner_stats$Variable <- gsub("Corner[0-9]","Corner_",colnames(results_corner)[variable])
    rbind(corner_stats,hold_corner_stats) -> corner_stats
  }
}
write.xlsx(corner_stats,paste("corner_statistics_",project,".xlsx",sep=""),colNames = T,rowNames = F,keepNA = F)
```

Figure 12 – Statistical analysis for corner preference.

The last measurements calculated in IntelliR are the values for the learning curves presented in the paper (Figure 13). The usage of this section assumes you are interested in the drinking attempts variable calculated by IntelliR. If this variable was calculated, this part of the code should run without issues.

```
2902 #### Learning Curves ####
2903 DA_progression <- NULL
2904 visits_objects <- ls(pattern = "_visits$")
2905
2906 for (animal in all_animals) {
2907   # Iterate through all the challenges
2908   for (visits_object in visits_objects) {
2909     # Get the current challenge
2910     visits_df <- get(visits_object)
2911
2912     # Subset the data frame based on each individual animal
2913     subset_df <- subset(visits_df, Animal == animal & NosepokeNumber > 0 & Light_Status == "Dark")
2914
2915     # Progression
2916     if(nrow(subset_df) > 0){
2917       row.names(subset_df) <- NULL
2918       df <- data.frame(ID = animal, Number_DAs = 1, Success_DAs = 0, Challenge = gsub("_visits","",visits_object))
2919       if(subset_df[which(row.names(subset_df) == 1),"CornerCondition"] == "Correct"){
2920         df$Success_DAs <- 1
```

Figure 13 – Start of the section dedicated to the calculation of learning curves.

Finally, IntelliR plots bar graphs for all calculated variables, simple bar graphs for corner preference, and the learning curves. Plotting for the general statistics should work without issues regardless of what is used, with the same applying to the learning curves. However, for corner preference you once again have to be aware of the column numbers (line 3286).

A full set of scripts adapted to run the sucrose challenge with a few calculated variables is provided together with this PDF file. They are labeled with “tutorial_” in the beginning of the script name and contain everything needed to run the analysis. These were done on MacOS/Unix, so changes to the path to run it on Windows are necessary (change “/” to “\\” when loading files). The results are also made available here.

We hope this tutorial was useful and remain available in case of questions!