Quantitative Colour Pattern Analysis (QCPA)

Example post-batch processing and analysis in R

Contents

1 Setup

The following R script uses the trichromatic example without UV (sea slugs seen by a triggerfish, *Rhinecanthus aculeatus*), but can similarly be applied to the tetrachromatic example with UV (spiders seen by a bluetit, *Parus major*). To make the worked examples readily accessible to users, we have prioritised using base R functions. As we will be working with primarily character-type data (i.e., species names, region of interest descriptors), we will introduce the user to many base functions that are for list-type data structures (e.g., `lapply()`). For clarity, we recommend that you, as the user, need to be comfortable with the different data types (e.g., numerical, character, factors) and data structures such as lists (collection of mixed data types) and data frames (special type of list). We also provide different methods to extract data, ideally introducing different coding methods and providing ways to increase the efficiency of data wrangling.

For the following R script to run, please ensure that you have implemented the QPCA batch script methods. In the following folder

`Worked_example_data/Example1_tri-chromat_no_UV/Test Data`

there should be two subfolders for the two species, *Aphelodoris varia* and *Aplysia sp.* Within these subdirectories, please confirm that data has been generated. As you proceed through the next few steps, we provide code that checks the data completeness. If you are uncertain that you uniformly named files, we have additionally provided script to check region of interest names (see section 2.3).

1.1 File naming conventions

It is important to note that the QCPA batch script folder and file names contain special characters and spaces, which can result in R code mishandling file names. We use this here as an example of poor naming conventions and provide code below to fix names, however it is best to be conscious of using consistent naming conventions prior to file generation. There are several naming conventions that can be used when coding (e.g., camel case and periods to replace spaces). In this tutorial, we use the snake case naming convention to improve readability when naming functions, variables, and arguments. Snake case replaces spaces with underscores and variable names are written in lowercase (e.g., 'Test data' becomes 'test_data,' Wickham 2014). We will rename four levels of folder names, including the species names. To run the following code, make sure the folders are not open on your desktop.

```
# Level 1:
# We first correct the file name of the parent directory 'Worked_example_data'
# Specify the old file name and the new file name
old_name <- "Worked_example_data"
new_name <- "example_data"
# Then use base function file.name to correct the file name to snake case
file.rename(old_name,new_name)
```

```
## [1] TRUE
```

```
# Level 2:
# We then change the UV and no UV example folder names within "example_data"
# Examine the files in example_data using list.files
list.files("example_data")
```

```
## [1] "Example1_tri-chromat_no_UV"     "Example2_tetra-chromat with UV"
```

```r
# Save the list of names to old_eg_names
old_eg_names <- list.files("example_data")

# Modify the names in old_eg_names to snake case and save to new_names
# Using a nested gsub we can replace the space, dash and 'Example'
new_eg_names <- gsub("-", "",# Remove the dashes
                 gsub(" ", "_",# Replace all spaces with underscores
                      gsub("Example", "eg", old_eg_names)))# Abbreviate Example to 'eg'

new_eg_names <- tolower(new_eg_names) # Convert to lowercase

# Add the directory level to the parent folder example_data
old_eg_names <- paste0("example_data/", old_eg_names)
new_eg_names <- paste0("example_data/", new_eg_names)

# Then replace the folder names
file.rename(old_eg_names,new_eg_names)
```

```
## [1] TRUE TRUE
```

```r
# Level 3:
# Here we correct the cone mapping and the test folder names
# using full.names = TRUE to save specifying directory pathways
old_test_names <- list.files(new_eg_names, full.names = TRUE)

new_test_names <- gsub(" ", "_", old_test_names)# Replace all spaces with underscores

new_test_names <- tolower(new_test_names) # Convert to lowercase

# And again replace the folder names
file.rename(old_test_names, new_test_names)
```

```
## [1] TRUE TRUE TRUE TRUE TRUE
```

```r
# Level 4:
# We finally fix the species names for eg1_trichromat_no_uv
list.files("example_data/eg1_trichromat_no_uv/test_data")# Examine folders
```

```
## [1] "Aphelodoris varia"  "Aplysia sp"         "Batch_QCPA_Log.txt"
```

```r
# To exclude the text file, filter species folders based on the "Ap" pattern
# in their names
old_species_names<- list.files("example_data/eg1_trichromat_no_uv/test_data",
                               pattern = "Ap", full.names = TRUE)

new_species_names <- gsub(" ", "_", old_species_names)# Replace spaces with underscores

new_species_names <- tolower(new_species_names) # Convert to lowercase

# And again replace the folder names
file.rename(old_species_names, new_species_names)
```

```
## [1] TRUE TRUE
```

Our new directory becomes:

`example_data/eg1_trichromat_no_uv/test_data`

2 Create the foundational dataset identifying unique animal IDs and regions of interest for multiple viewing distances

2.1 Iterate over folders to extract species, animals and viewing distances

First, we will specify the location of the data files with respect to where your working directory has been saved. Here, we have assumed that the R script was saved in the parent folder of `/example_data`. The next step will be to extract the names of the two focal species using the folder names within the `/test_data` folder.

```
# Construct the global data location path. This assumes the R script is in the
# project root
data_location <- file.path(getwd(), "example_data/eg1_trichromat_no_uv/test_data")

# List the species folders in the data location
# These could also be locations depending on your data set
species_folders <- list.files(path = data_location)

# Print the list of species folders and log file with QCPA processing details
print(species_folders)
```

```
## [1] "aphelodoris_varia"  "aplysia_sp"         "Batch_QCPA_Log.txt"
```

```
# To remove the "Batch_QCPA_Log.txt" from our list of folder names,
# we subset the list on the condition that file names do not end in
# "Log.txt" using the endsWith function

species <- species_folders[!endsWith(species_folders, "Log.txt")]

# Print the filtered list of species folders
print(species)
```

```
## [1] "aphelodoris_varia" "aplysia_sp"
```

Here, we will demonstrate the extraction of folder names using a for loop. We first create an empty data frame called `animal_info`. Then, for each i^{th} species (here $i = 1$ or 2), we create a path into the species folder and record in the object `indvidual_list` all the unique individual animal folder names using the `list.files()` function. We then establish a second, nested loop, which will then iterate over each individual (`individual_name`) in `indvidual_list` to extract all unique viewing distance folder names using `list.files()`. To distinguish the viewing distance folders from other files saved within the directory, use the argument `pattern = "cm"` in `list.files()`. Finally, we combine the unique species name, individual name and viewing distances into unique columns and add as a new row in `animal_info`.

```r
# Initialize an empty data frame
animal_info <- data.frame()

# Iterate over the species folders
for (species_name in species) {

  species_path <- file.path(data_location, species_name)
  individual_list <- list.files(path = species_path)

  # Iterate over the individuals in each species folder
  for (individual_name in individual_list) {

    individual_path <- file.path(species_path, individual_name)
    distances <- list.files(path = individual_path, pattern = "cm")

    # Add new rows to the animal_info data frame
    new_rows <- expand.grid(
      Species = species_name,
      Ind = individual_name,
      Dist = distances,
      stringsAsFactors = FALSE
    )
    animal_info <- rbind(animal_info, new_rows)
  }
}

# Rename the column names in animal_info
colnames(animal_info) <- c("Species", "Ind", "Dist")
```

By using the for loop here, the syntax of the code should make it clear as to how we systematically extract information and add rows of data to our 'growing' `animal_info` data frame. However, when using for loops in this manner, the loop uses increased amounts of memory and will perform poorly with larger amounts of data. As we proceed, we will use the functions `apply()`, `lapply()` and `sapply()` which iterate over data in a similar manner as a for loop but are more efficient.

2.2 Inspect data

Check that the data has correctly been imported and reflects your experimental records. We converted the character data into factors, which is the appropriate format for running statistical tests in downstream analyses. We use the `lapply()` function to apply the function `factor()` to each column in our data frame. We can again use the `lapply()` function to apply the function `nlevels()` to our newly converted data frame, which will report the number of unique elements within each column of the data frame `animal_info`.

```r
# Check data
head(animal_info)
```

```
##             Species                            Ind Dist
## 1 aphelodoris_varia 3_AphelodorisVaria_10_NF_D_RAW 30cm
## 2 aphelodoris_varia 3_AphelodorisVaria_10_NF_D_RAW 50cm
## 3 aphelodoris_varia 3_AphelodorisVaria_11_NF_D_RAW 30cm
## 4 aphelodoris_varia 3_AphelodorisVaria_11_NF_D_RAW 50cm
## 5        aplysia_sp         4_Aplysiasp_1_NL_N_RAW 30cm
## 6        aplysia_sp         4_Aplysiasp_1_NL_N_RAW 50cm
```

```r
# Convert character columns to factors
animal_info[] <- lapply(animal_info, as.factor)

# Display structure of the data
str(animal_info)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ Species: Factor w/ 2 levels "aphelodoris_varia",..: 1 1 1 1 2 2
##  $ Ind    : Factor w/ 3 levels "3_AphelodorisVaria_10_NF_D_RAW",..: 1 1 2 2 3 3
##  $ Dist   : Factor w/ 2 levels "30cm","50cm": 1 2 1 2 1 2
##  - attr(*, "out.attrs")=List of 2
##   ..$ dim     : Named int [1:3] 1 1 2
##   .. ..- attr(*, "names")= chr [1:3] "Species" "Ind" "Dist"
##   ..$ dimnames:List of 3
##   .. ..$ Species: chr "Species=aphelodoris_varia"
##   .. ..$ Ind    : chr "Ind=3_AphelodorisVaria_10_NF_D_RAW"
##   .. ..$ Dist   : chr [1:2] "Dist=30cm" "Dist=50cm"
```

```r
# Check the number of unique species, individuals and viewing distances
sapply(animal_info, nlevels)
```

```
## Species     Ind    Dist
##       2       3       2
```

```r
# Aggregate data to count individuals by species and distance
aggregate_counts <- aggregate(Ind ~ Species + Dist, data = animal_info, FUN = length)

# Plot bar chart of individual counts by species and distance
barplot(Ind ~ Dist + Species,
  beside = TRUE,
  ylab = "Count",
  legend.text = TRUE,
  data = aggregate_counts
)
```

In this example, there are only three individuals. However, with a larger data set with missing data, this becomes an important step to determine a balanced experimental design.

2.3 Inspect regions of interest for inconsistent labelling (additional step)

The number of data files will depend on whether animal, background and/or animal+background options were selected; this will make up the prefix of the file name followed by an underscore and image analysis file type (e.g., `_Summary Results.csv`). For the following R code to work, regions of interest must be uniformly named and capitalised for all individuals analysed. Below is a method to ensure that these regions of interest (ROIs) have been correctly named by reporting all unique names across your data by examining all unique `.tif` file names. If you are confident that the naming was consistent, you may skip this step.

```r
# Generate individual animal sub-directories paths
animal_sub_dirs <- apply(animal_info, 1, function(x) paste(x, collapse = "/"))

# Prepend the global file directories to the sub-directory paths
```

Figure 1. Counts of individuals within species for each viewing distance.

```
full_animal_dirs <- file.path(data_location, animal_sub_dirs)

# Identify the unique regions of interest. If there are
# more than expected - look for naming inconsistencies.
tif_files <- list.files(full_animal_dirs, pattern = ".tif")
roi_names <- unique(unlist(strsplit(tif_files, "_"))[c(TRUE, FALSE)])
roi_names
```

```
## [1] "animal"            "animal+background" "background"
```

How the final lines works

The final lines achieve four things:

1. `list.files()` searches all the locations specified in the `full_animal_dirs` vector for files with the `.tif` extension and puts these file names into a new character vector.
2. Knowing that file names are composed of two bits of information separated by an underscore, `strsplit()` cuts the file names into two elements at the underscore and outputs a list containing these two new elements for each unique file.
3. `unlist()` puts all elements into a vector. As the region of interest are the prefix, it will be first element in the pair, which we can subset using: `[ c(TRUE,FALSE)]`.
4. Finally, the `unique()` function looks across all names used for each individual and retains only unique names, omitting duplicates.

2.4 Add regions of interest to animal information dataset

To proceed with the next step, the regions of interest (ROIs) must be uniformly named across all individual specimens (see above if uncertain). Here, we introduce two methods to add ROIs. The first method is the quickest and works if you are confident in your naming conventions.

However, we provide a second method that extracts your data's ROIs.

2.4.1 Simplified method to list ROIs

```
ROI <- c("animal", "animal+background", "background") # Add the names of ROIs
```

2.4.2 Extract ROIs from the data

Here, we will utilise the batch script naming convention of using the ROI as the prefix of the image analysis file name followed by an underscore and image analysis file type (e.g., '*_Summary Results.csv*'). See Table 1 below for all QCPA analyses and associated file names.

Table 1. QCPA image analysis and associated output files. See van den Berg & Troscianko et al. (2020) for a detailed description.

Image analysis	File name
CAA, VCA, BSA	_Summary Results.csv
GabRat	_GabRat_Results.csv
LEIA	_Local Edge Intensity Analysis.csv
Particle analysis	_Cluster Particle Analysis Summary Results.csv
Particle analysis + cluster properties	_Cluster Results.csv
Particle analysis + sub-cluster properties	_Individual Particle Results.csv
Particle analysis + transition matrix + cluster properties	_ROI Cluster Results.csv

We can then extract any particular image analysis `.csv` data files names using the `list.files()` function specifying the file name as the pattern to search for. In the following code, we have used '_Summary Results.csv' as our string pattern. Once we have all the file names with '_Summary Results.csv', we remove this phrase using `gsub()`.

```
# Here, we create a character vector determining whether animal,
# background and/or animal+background options were used
# Using the first row of animal_info to specify the location to examine
# the number and types of _Summary Results.csv files

# Use the first row of animal_info to specify the location to examine
first_animal_path <- file.path(data_location,
                               paste(unlist(animal_info[1, ]), collapse = "/"))

# Find all the ROI output files for '_Summary Results.csv'
roi_files <- list.files(first_animal_path, pattern = "_Summary Results.csv")

# Extract ROI names by removing the "_Summary Results.csv" suffix
rois <- gsub("_Summary Results.csv", "", roi_files)
rois
```

```
## [1] "animal"            "animal+background" "background"
```

```
# Note: In the QCPA batch script, three ROIs were investigated.
```

2.5 Merge the list of ROIs names with the animal ID data frame

```
# Expand each unique row of the animal_info data frame to included a unique ROI value
animal_info_roi <- merge(animal_info, rois)

colnames(animal_info_roi)[4] <- "ROI"  # Name column 4 to ROI

animal_info_roi$ROI <- factor(animal_info_roi$ROI)  # Convert to factor

head(animal_info_roi)  # Check data
```

```
##             Species                              Ind Dist    ROI
```

```
## 1 aphelodoris_varia 3_AphelodorisVaria_10_NF_D_RAW 30cm animal
## 2 aphelodoris_varia 3_AphelodorisVaria_10_NF_D_RAW 50cm animal
## 3 aphelodoris_varia 3_AphelodorisVaria_11_NF_D_RAW 30cm animal
## 4 aphelodoris_varia 3_AphelodorisVaria_11_NF_D_RAW 50cm animal
## 5        aplysia_sp         4_Aplysiasp_1_NL_N_RAW 30cm animal
## 6        aplysia_sp         4_Aplysiasp_1_NL_N_RAW 50cm animal
```

```
str(animal_info_roi)  # Check that ROI is a factor
```

```
## 'data.frame':    18 obs. of  4 variables:
##  $ Species: Factor w/ 2 levels "aphelodoris_varia",..: 1 1 1 1 2 2 1 1 1 1 ...
##  $ Ind    : Factor w/ 3 levels "3_AphelodorisVaria_10_NF_D_RAW",..: 1 1 2 2 3 3 1 1 2 2 ...
##  $ Dist   : Factor w/ 2 levels "30cm","50cm": 1 2 1 2 1 2 1 2 1 2 ...
##  $ ROI    : Factor w/ 3 levels "animal","animal+background",..: 1 1 1 1 1 1 2 2 2 2 ...
```

3 Create a function to import a .csv list from VCA, BSA, and CAA data, with options for particle analysis

As before, you could use a series of nested loops to extract the data in the .csv files (e.g., section 2.1). However, here we introduce how to define a function to determine the data location using the data listed in the `animal_info_roi` data frame and import the .csv file. By creating a function, we can then make our file importing dynamic, providing options to import different image analysis files. And, by using information called directly from the `animal_info_roi` data frame, we ensure the data extracted is correct and reduces errors that would have potentially occurred if we did a broad import. First, we present a simple example of a function to import the VCA,BSA,CAA data files, and then, secondly, we expand on our function to demonstrate how to define and import multiple types of image analysis data files.

3.1 Simplified function example to read in image analysis files

In the following function, we walk through the steps for a function that only imports VCA, BSA, CAA image analysis data but will lay the foundations for when we create the function that can input any of the image analyses.

There are three parts:

1. The first section will specify what type of image analysis we are interested in
2. The second section extracts the data using `animal_info_roi` data frame and stores the data as a list in the object `Sum_Res_list`
3. In the third section, we use the `do.call()` function to extract all the list elements in `Sum_Res_list` and combine them into a data frame using `rbind()`. From Table 1, the file suffix for this analysis is '*_Summary Results.csv*'.

```
# Part A:
# Here, we specify what image analysis so that we can use the correct file extension
exnfile <- "_Summary Results.csv"

ani_id_dat <- animal_info_roi # Assign our animal ID data frame

path <- data_location # This is the global location set in section 2.1
```

```r
# Part B:
# Once the file extension condition is met, we use lapply to extract the data,
# using a nested function within lapply
Sum_Res_list <- lapply(
  1:nrow(ani_id_dat), # for each row in ani_id_dat...
  function(x) { # start of our nested function

    # Extract the individual row information
    tmpLoc <- paste(unlist(ani_id_dat[x, ]), collapse = "/")

    # Create a directory string
    tmpLoc <- paste0(paste(path, tmpLoc, sep = "/"), exnfile)

    # Read in the data using the newly specified location
    data <- read.csv(tmpLoc)

    # Add the specific animal information to the data
    data <- merge(ani_id_dat[x, ], data)

    data # Returns data
  }
)

# Part C:
# We then collapse the Sum_Res_list list into a data frame by
# calling rbind in the function do.call
Sum_Res_df <- do.call(rbind, Sum_Res_list)

# Check the imported data
dim(Sum_Res_df) # 18 rows by 133 columns
```

```
## [1]  18 133
```

```r
# Finally, to prevent any errors we remove the objects ani_id_dat, exnfile, path
# from the global environment because we'll be recycling them in the second function
rm(exnfile)
rm(ani_id_dat)
rm(path)
```

3.2 Dynamic function to read in various image analysis files

To dynamically import data, we are going to create the function `read_qcpa()`, which will take an animal ID data frame (`ani_id_dat`), the specific image analysis that we are interested in (`filetype`), and a folder directory (where we set the default `path` to the `data_location` defined in section 1.2). For Part A of the function, we can conditionally set what image analysis we are interested in and set the `exnfile` object to the correct file extension. We have listed each image analysis and the argument name to be specified as the filetype in Table 2. As the last conditional (at the `else` level), we have used the `stop()` function which will occur if `filetype` has not been specified, and will report an error. Part B is the same as in section 3.1 above however it is worth noting that we used `merge()` here instead of `rbind()` (section 2.1) because this allows our function to handle data in both long and wide format. Then, in Part C, we use the `return()` function to output our newly imported data.

Table 2. QCPA image analysis and the different filetype arguments

Image analysis	Filetype argument
VCA,BSA,CAA	"VCA"
Cluster Particle Analysis	"PartAn"
Cluster Results	"Clust"
Individual Particle	"IndParticle"

```
read_qcpa <- function(ani_id_dat, filetype = "NA", path = data_location) {
  # Default location is assigned, but can be changed

  # Part A: Specify arguments for different image analyses
  file_extensions <- list(
    VCA = "_Summary Results.csv",
    PartAn = "_Cluster Particle Analysis Summary Results.csv",
    Clust = "_Cluster Results.csv",
    IndParticle = "_Individual Particle Results.csv"
  )

  if (!filetype %in% names(file_extensions)) {
    stop('Specify analysis output type:
         filetype = "VCA", "PartAn", "Clust", or "IndParticle"')
  }

  exnfile <- file_extensions[[filetype]]

  # Part B: Extract data using lapply
  data_list <- lapply(seq_len(nrow(ani_id_dat)), function(x) {
    # Extract row information and create directory string
    dir_components <- unlist(ani_id_dat[x, ])
    tmpLoc <- file.path(path, paste(dir_components[1:3], collapse = "/"),
                        paste(dir_components[4], exnfile, sep = ""))

    # Check if file exists before trying to read it
    if (!file.exists(tmpLoc)) {
      stop("File does not exist: ", tmpLoc)
    }

    # Read data and add animal information
    data <- read.csv(tmpLoc)
    data <- merge(ani_id_dat[x, ], data)

    return(data)
  })

  # Part C: Combine list into a data frame
  data_df <- do.call(rbind, data_list)

  return(data_df)
}
```

We can then import our data using our function and do some data cleaning by removing unnecessary columns.

```r
# VCA,BSA,CAA analysis:
# Read in and assign analysis
VCA_analysis <- read_qcpa(animal_info_roi, filetype = "VCA")

# Subset data excluding the column Image and X
VCA_analysis <- VCA_analysis[, !colnames(VCA_analysis) %in% c("X", "Image")]

dim(VCA_analysis) # 18 rows, by 131 columns
```

```
## [1]  18 131
```

```r
VCA_analysis[1:2, 1:6] # Observe the first two rows and six columns of data
```

```
##             Species                              Ind Dist    ROI   CAA.Sc    CAA.Jc
## 1 aphelodoris_varia 3_AphelodorisVaria_10_NF_D_RAW 30cm animal 2.895159 0.9650530
## 2 aphelodoris_varia 3_AphelodorisVaria_10_NF_D_RAW 50cm animal 1.985642 0.9928209
```

```r
# Cluster Particle Analysis:
# Read in and assign analysis
Particle_analysis <- read_qcpa(animal_info_roi, filetype = "PartAn")

dim(Particle_analysis) # 62 rows, by 20 columns
```

```
## [1] 62 20
```

```r
colnames(Particle_analysis)[5]
```

```
## [1] "Cluster.ID"
```

```r
# Change column name to be consistent with Cluster analysis
colnames(Particle_analysis)[5] <- "ClusterID"

# Cluster Results:
# Read in and assign analysis
Cluster_analysis <- read_qcpa(animal_info_roi, filetype = "Clust")

# Subset data excluding the column Image and X
Cluster_analysis <- Cluster_analysis[, !colnames(Cluster_analysis) %in% c("X", "Image")]

dim(Cluster_analysis) # 62 rows, by 21 columns
```

```
## [1] 62 21
```

```r
# Individual Particle Analysis:
# Read in and assign analysis
IndParticle_analysis <- read_qcpa(animal_info_roi, filetype = "IndParticle")

# Subset data excluding the column X.1
IndParticle_analysis <- IndParticle_analysis[, colnames(IndParticle_analysis) != "X.1"]

dim(IndParticle_analysis) # 214 rows, by 23 columns
```

```
## [1] 214  23
```

4 Get a .csv list from LEIA output

Here, we implement a function similar to section 3.1. However, because QCPA batch script exports LEIA analysis into a subdirectory, we have to add two additional folders to our location, `LEIA` and the ROI as a folder. For the LEIA folder, as we iterate over each row from `animal_info_roi`, we convert the row into a character vector and add 'LEIA' as an element. We include a backslash to treat the ROI as a folder before specifying the image analysis file extension, `_Local Edge Intensity Analysis.csv` (Table 1).

```
# Define a function to generate file paths, read data, and merge animal information
read_leia <- function(index, animal_info, base_location) {
  # Extract row as a character vector
  row_values <- as.character(unlist(animal_info[index, ]))

  # Construct the file path
  file_path <- file.path(
    base_location,
    row_values[1],
    row_values[2],
    row_values[3],
    "LEIA",
    row_values[4],
    "_Local Edge Intensity Analysis.csv"
  )

  # Read data
  data <- read.csv(file_path)

  # Add animal information to the data
  data <- merge(animal_info[index, ], data)

  return(data)
}

# Read LEIA data using lapply
LEIA_Res_list <- lapply(
  seq_len(nrow(animal_info_roi)),
  function(x) read_leia(x, animal_info_roi, data_location)
)

# Combine list into a data frame
LEIA_Res_analysis <- do.call(rbind, LEIA_Res_list)

# Subset data to exclude specific columns
LEIA_Res_analysis <- LEIA_Res_analysis[, !(colnames(LEIA_Res_analysis)
                                           %in% c("X", "Image"))]

# Check the resulting data
dim(LEIA_Res_analysis) # Print dimensions of the data
```

```
## [1] 18 37
```

```
names(LEIA_Res_analysis)
```

```
##  [1] "Species"          "Ind"              "Dist"             "ROI"              "Col.mean"
##  [8] "Col.skew"         "Col.kurtosis"     "Lum.mean"         "Lum.sd"           "Lum.CoV"
## [15] "Col.mean.hrz"     "Col.sd.hrz"       "Col.CoV.hrz"      "Col.skew.hrz"     "Col.kurtosis.hrz"
## [22] "Col.CoV.vrt"      "Col.skew.vrt"     "Col.kurtosis.vrt" "Lum.mean.hrz"     "Lum.sd.hrz"
## [29] "Lum.kurtosis.hrz" "Lum.mean.vrt"     "Lum.sd.vrt"       "Lum.CoV.vrt"      "Lum.skew.vrt"
## [36] "Col.thresh"       "Lum.thresh"
```

5 Get a .csv list from GabRat output

For GabRat analyses, there are no specific ROIs, so we only need the first three columns from `animal_info_roi`. Similar to LEIA (section 4, above), QCPA batch script also exports GabRat analysis into a subdirectory. We will again iterate over each row of `animal_info_roi` and convert the row into a character vector, adding 'GabRat' as an element. From Table 1, the GabRat file extension is: `_GabRat_Results.csv`, where we place a backslash before the file name to specify that it is located within the GabRat folder.

For this work example, we are only interested in the GabRat values `dbl` (the luminance channel of the visual system considered in the analysis), so we will then extract the rows with only these values. We then extract the ROI information in a method similar to section 2.3.

```
# Because we are not using ROI which adds three levels (rows),
# we trim data frame using unique()
# For each the 6 rows in animal_info_roi

# Import GabRat results and process data

# Function to read and process GabRat results
read_gabrat <- function(data_location, animal_info) {
  gabrat_res_list <- lapply(
    1:nrow(unique(animal_info[, 1:3])),
    function(row_index) {
      # Convert row into a character vector
      tmp_loc <- as.character(unlist(animal_info[row_index, 1:3]))
      tmp_loc <- c(tmp_loc, "GabRat") # Add the new sub folder GabRat
      dir_path <- file.path(data_location,
                            paste(tmp_loc, collapse = "/"),
                            "_GabRat_Results.csv")

      # Read in the data using the newly specified location
      data <- read.csv(dir_path)

      # Add the specific animal information to the data
      data <- merge(animal_info[row_index, 1:3], data)
      data
    }
  )

  # Collapse the gabrat_res_list list into a data frame
  gabrat_res_analysis <- do.call(rbind, gabrat_res_list)

  # Filter data for rows containing "_dbl" in ID
  gabrat_res_analysis <- gabrat_res_analysis[endsWith(gabrat_res_analysis$ID, "_dbl"), ]
  rownames(gabrat_res_analysis) <- NULL
```

```
  # Add ROI column by extracting substring after "WholeImage_"
  gabrat_res_analysis$ROI <- sapply(
    strsplit(gabrat_res_analysis$ID, "WholeImage_"), `[`, 2
  )

  # Remove '_dbl' suffix from ROI column
  gabrat_res_analysis$ROI <- gsub("_dbl", "", gabrat_res_analysis$ROI)

  # Remove unnecessary columns
  gabrat_res_analysis <- gabrat_res_analysis[, !colnames(gabrat_res_analysis) %in%
                                               c("X", "ID", "Sigma")]

  gabrat_res_analysis
}

# Call the function to read and process GabRat results
gabrat_res_analysis <- read_gabrat(data_location, animal_info_roi)

# Check the processed data
dim(gabrat_res_analysis) # Should show 18 rows and 5 columns
```

```
## [1] 18  5
```

```
print(head(gabrat_res_analysis)) # Displays first six rows of data
```

```
##             Species                              Ind Dist    GabRat               ROI
## 1 aphelodoris_varia 3_AphelodorisVaria_10_NF_D_RAW 30cm 0.1682120            animal
## 2 aphelodoris_varia 3_AphelodorisVaria_10_NF_D_RAW 30cm 0.5332068 animal+background
## 3 aphelodoris_varia 3_AphelodorisVaria_10_NF_D_RAW 30cm 0.4139920        background
## 4 aphelodoris_varia 3_AphelodorisVaria_10_NF_D_RAW 50cm 0.1624157            animal
## 5 aphelodoris_varia 3_AphelodorisVaria_10_NF_D_RAW 50cm 0.4468272 animal+background
## 6 aphelodoris_varia 3_AphelodorisVaria_10_NF_D_RAW 50cm 0.3590715        background
```

6 Combine data sets and data preparation

6.1 Merging data sets

To merge files, we will again use the `merge()` function.

6.1.1 Particle analyses

```
# Merging two files together
Cluster_Particle_analysis <- merge(Cluster_analysis, Particle_analysis,
  by = c("Species", "Ind", "Dist", "ROI", "ClusterID")
)

dim(Cluster_Particle_analysis) # 62 rows and 36 columns
```

```
## [1] 62 36
```

```
names(Cluster_Particle_analysis) # Inspect data
```

```
##  [1] "Species"           "Ind"               "Dist"             "ROI"              "ClusterID"
##  [7] "lw_mean"           "lw_sd"             "mw_mean"          "mw_sd"            "sw_mean"
## [13] "dbl_mean"          "dbl_sd"            "Area"             "Coverage..pc."    "RNL.X"
## [19] "RNL.Saturation"    "Dmax.Chromaticity" "Dmax.channel"     "Count"            "Total.Area"
## [25] "X.Area"            "Perim."            "Major"            "Minor"            "Angle"
## [31] "Solidity"          "Feret"             "FeretX"           "FeretY"           "FeretAngle"
```

6.1.2 VCA,BSA,CAA data, LEIA output and GabRat analyses

Here, we demonstrate how to collate data across the VCA, BSA, CAA data, LEIA output, and GabRat output. We have excluded the particle analysis outcome from this step.

```
# Merge the first two data frames (VCA_analysis and LEIA_Res_analysis) together
VCA_LEIA_GabRat_analysis <- merge(VCA_analysis, LEIA_Res_analysis,
  by = c("Species", "Ind", "Dist", "ROI")
)

dim(VCA_LEIA_GabRat_analysis) # 18 rows and 164 columns
```

```
## [1]  18 164
```

```
# Merge the new data frame (VCA_LEIA_GabRat_analysis) with gabrat_res_analysis
VCA_LEIA_GabRat_analysis <- merge(VCA_LEIA_GabRat_analysis, gabrat_res_analysis,
  by = c("Species", "Ind", "Dist", "ROI")
)

dim(VCA_LEIA_GabRat_analysis) # 18 rows and 165 columns
```

```
## [1]  18 165
```

```
names(VCA_LEIA_GabRat_analysis) # Inspect data structure
```

```
##   [1] "Species"          "Ind"              "Dist"             "ROI"              "CAA.Sc"
##   [8] "CAA.Jt"           "CAA.Hc"           "CAA.Qc"           "CAA.Ht"           "CAA.Qt"
##  [15] "CAA.C"            "CAA.Sc.Hrz"       "CAA.Sc.Vrt"       "CAA.Jc.Hrz"       "CAA.Jc.Vrt"
##  [22] "CAA.Jt.Hrz"       "CAA.Jt.Vrt"       "CAA.Hc.Hrz"       "CAA.Hc.Vrt"       "CAA.Qc.Hrz"
##  [29] "CAA.Ht.Vrt"       "CAA.Qt.Hrz"       "CAA.Qt.Vrt"       "CAA.Scpl.Hrz"     "CAA.Scpl.Vrt"
##  [36] "CAA.C.Hrz"        "CAA.C.Vrt"        "CAA.PT"           "CAA.PT.Hrz"       "CAA.PT.Vrt"
##  [43] "VCA.sL"           "VCA.CVL"          "VCA.MDmax"        "VCA.sDmax"        "VCA.CVDmax"
##  [50] "VCA.CVSsat"       "VCA.MSL"          "VCA.sSL"          "VCA.CVSL"         "VCA.MS"
##  [57] "VCA.ML.Hrz"       "VCA.sL.Hrz"       "VCA.CVL.Hrz"      "VCA.ML.Vrt"       "VCA.sL.Vrt"
##  [64] "VCA.sDmax.Hrz"    "VCA.CVDmax.Hrz"   "VCA.MDmax.Vrt"    "VCA.sDmax.Vrt"    "VCA.CVDmax.Vrt"
##  [71] "VCA.CVSsat.Hrz"   "VCA.MSsat.Vrt"    "VCA.sSsat.Vrt"    "VCA.CVSsat.Vrt"   "VCA.MSL.Hrz"
##  [78] "VCA.MSL.Vrt"      "VCA.sSL.Vrt"      "VCA.CVSL.Vrt"     "VCA.MS.Hrz"       "VCA.sS.Hrz"
##  [85] "VCA.sS.Vrt"       "VCA.CVS.Vrt"      "BSA.BML"          "BSA.BsL"          "BSA.BCVL"
##  [92] "BSA.BCVDmax"      "BSA.BMSsat"       "BSA.BsSsat"       "BSA.BCVSsat"      "BSA.BMSL"
```

```
##  [99] "BSA.BMS"           "BSA.BsS"           "BSA.BCVS"          "BSA.BML.Hrz"       "BSA.BsL.Hrz"
## [106] "BSA.BsL.Vrt"       "BSA.BCVL.Vrt"      "BSA.BMDmax.Hrz"    "BSA.BsDmax.Hrz"    "BSA.BCVDmax.Hrz"
## [113] "BSA.BCVDmax.Vrt"   "BSA.BMSsat.Hrz"    "BSA.BsSsat.Hrz"    "BSA.BCVSsat.Hrz"   "BSA.BMSsat.Vrt"
## [120] "BSA.BMSL.Hrz"      "BSA.BsSL.Hrz"      "BSA.BCVSL.Hrz"     "BSA.BMSL.Vrt"      "BSA.BsSL.Vrt"
## [127] "BSA.BsS.Hrz"       "BSA.BCVS.Hrz"      "BSA.BMS.Vrt"       "BSA.BsS.Vrt"       "BSA.BCVS.Vrt"
## [134] "Col.CoV"           "Col.skew"          "Col.kurtosis"      "Lum.mean"          "Lum.sd"
## [141] "Lum.kurtosis"      "Col.mean.hrz"      "Col.sd.hrz"        "Col.CoV.hrz"       "Col.skew.hrz"
## [148] "Col.sd.vrt"        "Col.CoV.vrt"       "Col.skew.vrt"      "Col.kurtosis.vrt"  "Lum.mean.hrz"
## [155] "Lum.skew.hrz"      "Lum.kurtosis.hrz"  "Lum.mean.vrt"      "Lum.sd.vrt"        "Lum.CoV.vrt"
## [162] "Transform"         "Col.thresh"        "Lum.thresh"        "GabRat"
```

6.2 Data cleaning

In this tutorial, we are using a minimal working example, where the data used does not have proper replication (i.e., we have 3 individuals in total). For those considering using linear mixed effects models (LMMs) in R for downstream analyses, we recommend reading Harrison et al. (2018). Please consider the number of data points (n) to parameters (k); where ideally $n/k > 3$ (but see discussion in Harrison et al. 2018).

6.2.1 Omit rows based on values

To demonstrate some data cleaning methods, we have created a smaller data frame `v.l.gabrat_sub_analysis`, subset from `VCA_LEIA_GabRat_analysis` by selecting the column names. In this data set, there are both `NaN` (not a number) and `Inf` (infinite) entries. Using `summary()` we can see that by retaining `NaN` and `Infs`, functions like `mean()` and `max()` can not calculated values correctly. To remove `Inf` values, we first convert them to `NA`. To do so, we can use the `is.infinite()` function to detect `Inf` values. However, this only works on vectors (i.e., data frame columns), so we can loop over our data frame using a variant of `lapply()`, the `sapply()` function, which returns a vector (not list).

```
# Column names specification
column_names <- c(
  "Species", "Ind", "Dist", "ROI", "BSA.BsL.Hrz", "Lum.mean",
  "CAA.Scpl", "Lum.CoV", "BSA.BML", "Col.mean", "GabRat"
)

# Subset the data based on the specified column names
v.l.gabrat_sub_analysis <- VCA_LEIA_GabRat_analysis[, column_names]

# Inspect the dimensions and provide an initial summary
cat("Dimensions before cleaning:", dim(v.l.gabrat_sub_analysis), "\n")
```

```
## Dimensions before cleaning: 18 11
```

```
# Replace infinite values with NA
infinite_indices <- sapply(v.l.gabrat_sub_analysis, is.infinite)
v.l.gabrat_sub_analysis[infinite_indices] <- NA

# Remove NA/NaNs
v.l.gabrat_sub_analysis <- na.omit(v.l.gabrat_sub_analysis)

# Inspect the cleaned data frame
cat("Dimensions after cleaning:", dim(v.l.gabrat_sub_analysis), "\n")
```

```
## Dimensions after cleaning: 14 11
```

```
# Examine variance for specified columns
variance_results <- apply(v.l.gabrat_sub_analysis[, 5:11], 2, var)
variance_results
```

```
## BSA.BsL.Hrz    Lum.mean    CAA.Scpl     Lum.CoV     BSA.BML    Col.mean      GabRat
## 0.001451083 0.084575106 0.019484022 0.083076179 0.007031222 0.001558206 0.014983401
```

6.2.2 Testing for normality and univaraite outliers

Again, with our minimal example, our estimates do not deviate from normality but here we have provided code to test this. Using a combination of the Shapiro-Wilk test for normality (Part A) and histograms (Part B) we can determine if our data is normal. For the Shapiro-Wilk test, the null hypothesis is that data is normal (i.e. p-value less than 0.05 are not normally distributed).

```
# Subset column names for analysis
colnames_subset <- colnames(v.l.gabrat_sub_analysis[, 5:11])

# Part A: Test for normality using the Shapiro-Wilk test
normality_results <- lapply(v.l.gabrat_sub_analysis[colnames_subset], shapiro.test)
print(normality_results)
```

```
## $BSA.BsL.Hrz
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.9755, p-value = 0.9403
##
##
## $Lum.mean
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.88774, p-value = 0.075
##
##
## $CAA.Scpl
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.93046, p-value = 0.3097
##
##
## $Lum.CoV
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
```

```
## W = 0.96054, p-value = 0.732
##
##
## $BSA.BML
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.90755, p-value = 0.1452
##
##
## $Col.mean
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.88052, p-value = 0.05911
##
##
## $GabRat
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.92572, p-value = 0.2655
```

```r
# Part B: Visualize distribution of the data using histograms
# Set up plot layout to display histograms as 3 rows by 3 columns
par(mfrow = c(3, 3))

# Create histograms for each column in the subset
for (column in colnames_subset) {
  hist(v.l.gabrat_sub_analysis[, column],
       breaks = 10,
       main = column,
       xlab = "")
}
```

To determine the degree of association between our colour variables, we can use both correlations and scatter plots. As our data is normally distributed we can use Pearson's correlation.

```r
# Define function for correlation panel
panel_correlation <- function(x, y) {
  # Set up the plotting area
  par(usr = c(0, 1, 0, 1))

  # Calculate and format the Pearson correlation
  correlation <- round(cor(x, y, method = "pearson"), 2)
  correlation_text <- paste0("R = ", correlation)

  # Display the correlation on the panel
  text(0.5, 0.5, correlation_text, cex = 1)
}
```

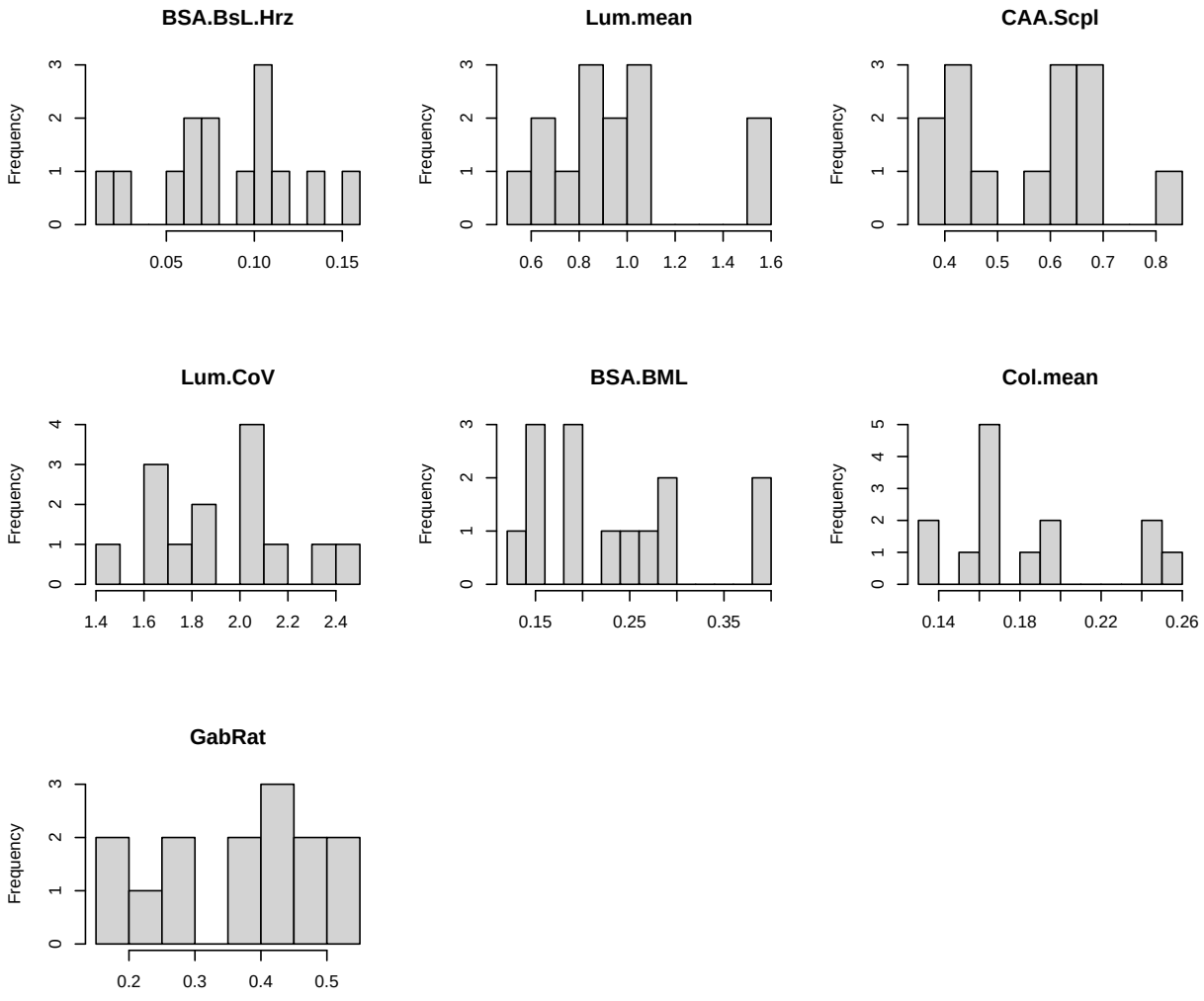

Figure 2. Histograms of seven colour variables.

```r
# Define function for scatter plot panel
panel_scatter <- function(x, y) {
  points(x, y,
         col = my_cols[v.l.gabrat_sub_analysis$ROI],
         cex = 1.1, pch = 19)
}

# Define colors
my_cols <- c("#00AFBB", "#E7B800", "#FC4E07")

# Create scatter plot matrix for the specified columns
pairs(v.l.gabrat_sub_analysis[, 5:11],
      main = "Seven colour variables for two viewing distances",
      upper.panel = panel_correlation,
      lower.panel = panel_scatter,
      oma = c(5, 5, 6, 18), gap = 0)

# Create legend data
legend_data <- unique(data.frame(
  ROI_Label = as.character(v.l.gabrat_sub_analysis$ROI),
  Color = my_cols[v.l.gabrat_sub_analysis$ROI]
))

# Display the legend
legend("right", legend_data$ROI_Label,
       xpd = TRUE,
       pch = 19, cex = 1.1, bty = "n", col = legend_data$Color)
```

From the scatter plots in Figure 3, there doesn't appear to be any outliers in each of the colour variables. However, we can confirm this using a boxplot (Part A) and then calculate the z-scores for each estimate (Part B) and look for outliers at greater than 3 standard deviations.

```r
# Convert data to long format using base R's 'reshape'
v.l.gabrat_sub_analysis_long <- reshape(v.l.gabrat_sub_analysis,
                            varying = list(names(v.l.gabrat_sub_analysis)[5:11]),
                            v.names = "value",
                            timevar = "variable",
                            idvar = c("Species", "Ind", "Dist", "ROI"),
                            direction = "long")

# Reset plot parameters for a single plot
par(mfrow = c(1, 1), oma = c(0, 0, 0, 0))

# Extract variable names for the x-axes labels
box_xlab <- names(v.l.gabrat_sub_analysis)[5:11]

# Create boxplots to inspect for outliers
boxplot(value ~ variable,
        data = v.l.gabrat_sub_analysis_long,
        pch = 16, boxwex = 0.5,
        xlab = "Colour variable",
        names = box_xlab)
```

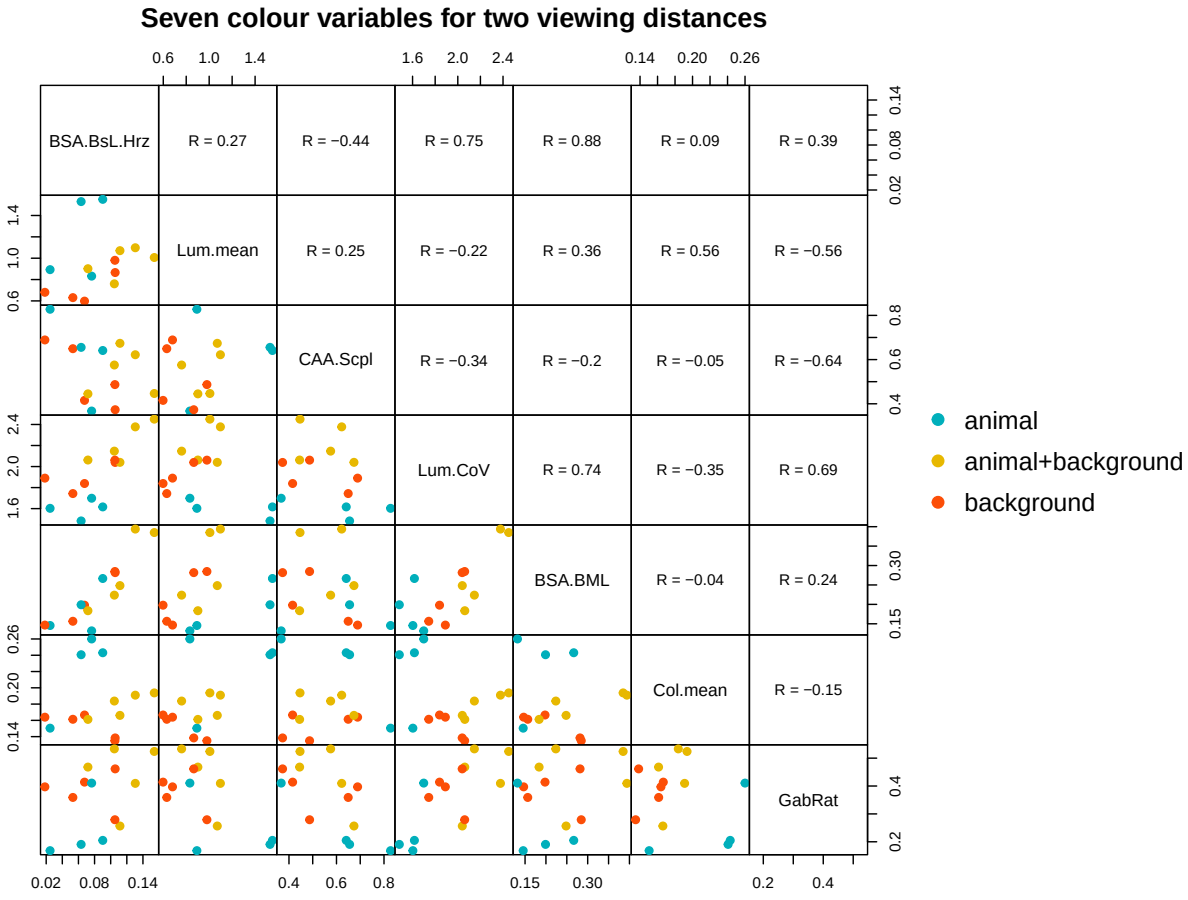

Figure 3. Paired scatter plots of seven colour variables.Lower off-diagonal are the paired scatter plots where coloured points representregions of interest (animal+background is yellow, background is red and animalis blue). Upper off-diagonal represents Pearson's correlation betweenpaired colour variables.

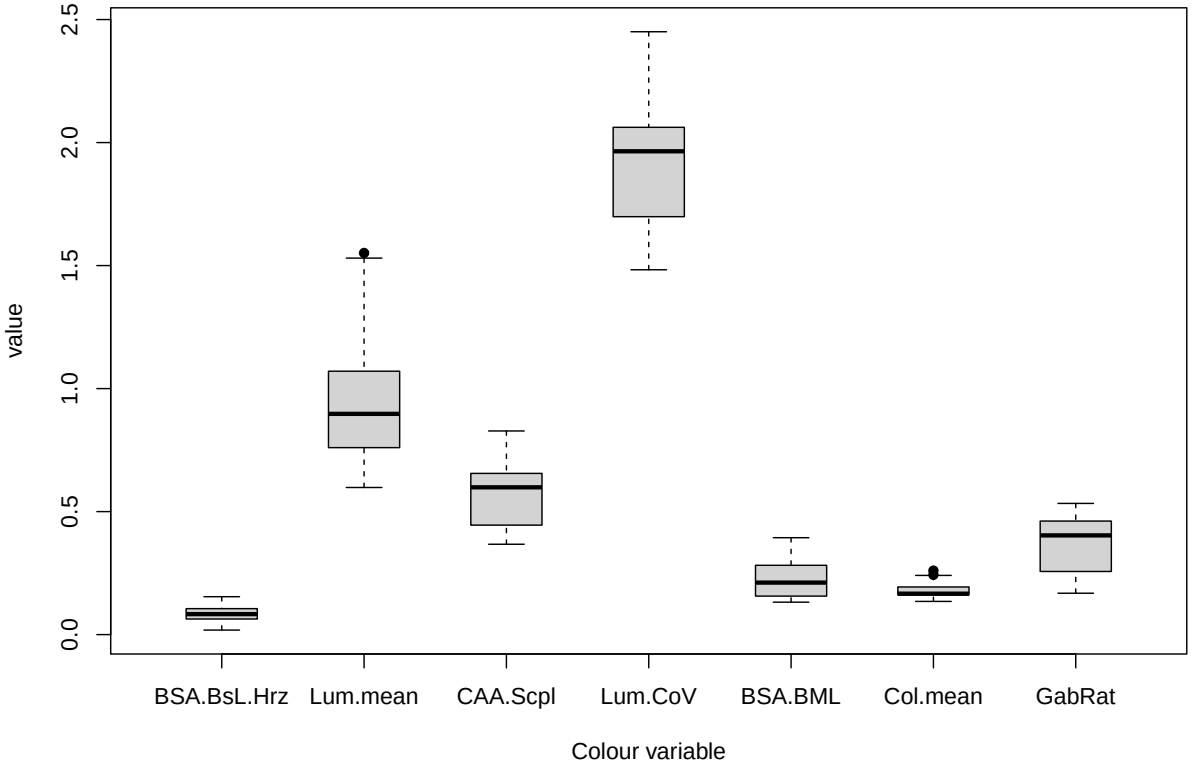

Figure 4. Box plots of the seven colour variables.

```r
# Calculate z-scores and look for outliers
apply(v.l.gabrat_sub_analysis[, 5:11], 2, function(x) (x - mean(x)) / sd(x))
```

```
##    BSA.BsL.Hrz    Lum.mean    CAA.Scpl    Lum.CoV     BSA.BML    Col.mean      GabRat
## 1   -1.5546385 -0.22055042  1.90379871 -1.1424503 -1.02808349 -0.81282404 -1.58807203
## 2    0.5370325 -0.67748164  0.09643282  0.7440674 -0.09602037  0.03116681  1.39374839
## 3   -0.4333418 -1.23452112 -1.04991729 -0.3237107 -0.40399429 -0.40340463  0.41982447
## 6   -0.8127940 -1.12221274  0.62598940 -0.6551888 -0.89799840 -0.53666071 -0.02884793
## 7    0.1599421  2.04325079  0.56684101 -1.0957471  0.41524161  1.53444596 -1.28720435
## 8    1.8375163  0.16987766 -0.82657835  1.7985505  1.82603561  0.28466379  1.31882541
## 9   -1.7238801 -0.95088812  0.90929480 -0.1461752 -1.01501221 -0.47110008  0.28130451
## 10  -0.5438505  1.97169020  0.66864937 -1.5580123 -0.39048437  1.47264174 -1.40479912
## 11   1.2192275  0.48433543  0.42992583  1.5449501  1.93280350  0.21421862  0.38333872
## 13  -0.2019337 -0.43002682 -1.39494368 -0.8094556 -1.19139269  1.96307457  0.39153293
## 14  -0.3228323 -0.19023671 -0.83845384  0.4506500 -0.57371014 -0.54001985  0.86062112
## 15   0.5639175 -0.31408682 -1.35293926  0.3711017  0.59554606 -1.11684897  0.80812224
## 17   0.7201389  0.39082017  0.79996562  0.3743901  0.19712480 -0.41485889 -0.86606705
## 18   0.5554960  0.08003014 -0.53806513  0.4470303  0.62994437 -1.20449430 -0.68232730
```

Although in Figure 4, there are points that exceed 1.5 times the interquartile range (the outer whiskers of the boxplot), none of the points exceed 3 standard deviations.

6.2.3 Standardising data

When implementing multivariate models, it is best to standardise data to facilitate model convergence. Here, we standardised the variance (mean = 0, standard deviation = 1) in each colour pattern variable across both viewing distances and ROIs using the `scale()` function.

```r
# Function to standardize and inspect data
standardize_data <- function(data) {
  # Standardize data
  standardized_data <- round(scale(data, center = TRUE, scale = TRUE), 3)

  # Check and print mean and variance
  cat("Column Means (should be near zero):\n")
  print(colMeans(standardized_data))
  cat("\nColumn Variances (should be 1):\n")
  print(apply(standardized_data, 2, var))
  standardized_data
}

# Update the data frame with standardized values and inspect
v.l.gabrat_sub_analysis[, 5:11] <- standardize_data(v.l.gabrat_sub_analysis[, 5:11])
```

```
## Column Means (should be near zero):
##   BSA.BsL.Hrz      Lum.mean      CAA.Scpl       Lum.CoV       BSA.BML      Col.mean        GabRat
## -7.142857e-05 -4.956353e-18 -1.982541e-18  7.142857e-05  7.142857e-05  1.586033e-17  7.142857e-05
##
## Column Variances (should be 1):
## BSA.BsL.Hrz    Lum.mean    CAA.Scpl     Lum.CoV     BSA.BML    Col.mean      GabRat
##   1.0001864   1.0000266   1.0000992   1.0000178   0.9999859   0.9999195   1.0000735
```

6.2.4 Multivariate outliers

For users who will be investigating colours multivariately, we would recommend using Mahalanobis distance to determine if there are any outliers in multivariate space. To determine p-values for Mahalanobis distance, you can use the Chi-Square statistic with k-1 degrees of freedom, where k is the number of variables (here k = 7), and p-values less than 0.05 outliers.

```
# Using R's Mahalanobis function, add a new column to your data
v.l.gabrat_sub_analysis$mahalanobis <- mahalanobis(
  v.l.gabrat_sub_analysis[, 5:11],

  # Get a vector of column means
  colMeans(v.l.gabrat_sub_analysis[, 5:11]),

  # Get a covaraince matrix
  cov(v.l.gabrat_sub_analysis[, 5:11])
)

# Calculate p-values using k-1 df = 6
v.l.gabrat_sub_analysis$pvalue <- pchisq(v.l.gabrat_sub_analysis$mahalanobis,
  df = 6, lower.tail = FALSE
)

# Inspect Mahalanobis distances and p-values
v.l.gabrat_sub_analysis[, colnames(v.l.gabrat_sub_analysis) %in%
                          c("mahalanobis", "pvalue")]
```

```
##    mahalanobis    pvalue
## 1     5.293433 0.5067658
## 2     7.356283 0.2891509
## 3     4.906830 0.5558174
## 6     4.588844 0.5975186
## 7     5.394414 0.4943090
## 8     4.548548 0.6028722
## 9     7.448557 0.2813473
## 10    5.847250 0.4405172
## 11    6.977108 0.3229695
## 13   10.099588 0.1205203
## 14    8.049307 0.2345132
## 15    7.024990 0.3185423
## 17    7.314508 0.2927383
## 18    6.150340 0.4065616
```

7 Export files

Depending on what image analysis sets you are using, we have provided multiple ways to save the image analysis data files as .csvs in the folder `/output`. First, we use `dir.create()` to create the folder `/output` in the `/test_data` folder, and save this new folder location as `out_path`. To save a single file, you can use `write.csv()`, where we have used the argument `row.names = FALSE` to make sure that the row numbers in the data frame aren't included in the .csv file.

To save multiple files, we utilise the fact that our naming convention in this R tutorial specifies image analysis data files with `_analysis`. Using the `ls()` function with the pattern argument set to `_analysis`, we can

create a list of all our image analysis data frames. Then, using a for loop, we iterate over each data frame, rename it using the current date, and then output the .csv file. Finally, we use `list.files()` to check that the files have been created.

```
# Function to create and return a directory path
create_directory <- function(path_components) {
  directory_path <- paste(path_components, collapse = "/")
  if (!dir.exists(directory_path)) {
    dir.create(directory_path)
  }
  return(directory_path)
}

# Create and get the output directory path
out_path <- create_directory(c(data_location, "output"))

# Save a single data frame to CSV
save_to_csv <- function(data, filename) {
  file_path <- paste0(out_path, "/", filename)
  write.csv(data, file = file_path, row.names = FALSE)
}

# Save the gabrat_res_analysis data frame
save_to_csv(gabrat_res_analysis, "gabrat_res_analysis.csv")

# Get the list of objects with "analysis" in their names
savefile_list <- ls(pattern = "analysis")

# Save each data frame in the list to a CSV file
for (data_name in savefile_list) {
  # Generate file name with date and custom format
  filename <- paste0(
    gsub("_analysis", "_data", data_name),
    format(Sys.time(), "_%d_%b_%Y"),
    ".csv"
  )

  save_to_csv(get(data_name), filename)
}
```

8 Fast import for large data sets using data.table (optional - advanced)

For very large data sets we would recommend importing data using the `data.table` package (Dowle and Srinivasan 2023). The `data.table` format can be used in parallel and is optimised for fast `.csv` reading/writing. For further information and cheat sheets, visit the `data.table` GitHub page (https://github.com/Rdatatable/data.table). The *data.table* has the following structure:

```
FROM[WHERE, SELECT, GROUP BY]
DT  [i,     j,      by]
```

Here we will provide a quick demonstration by importing the LEIA QCPA batch script output. First,

we convert our `animal_info_roi` into `data.table` format using the function `as.data.table()`. Then we will input our data using the dedicated `data.table` read function `fread()`. Finally, we can output large data.tables fast using the `fwrite()` function.

```
library(data.table)

# Convert animal_info_roi into data.table format
animal_info_roi_DT <- as.data.table(animal_info_roi)

# Create a function to generate the file path for LEIA analysis
generate_filepath <- function(row) {
  paste(data_location, row$Species, row$Ind, row$Dist, "LEIA", row$ROI,
        "_Local Edge Intensity Analysis.csv", sep = "/")
}

# Read data for each row in animal_info_roi_DT
LEIA_analysis_DT <- animal_info_roi_DT[, {
  file_path <- generate_filepath(.SD)
  fread(file_path)
}, by = 1:NROW(animal_info_roi_DT)]

# Remove unwanted columns
unwanted_columns <- c("NROW", "V1", "Image", "Transform")
LEIA_analysis_DT[, (unwanted_columns) := NULL]

# Combine the original and new data
LEIA_analysis_DT <- cbind(animal_info_roi_DT, LEIA_analysis_DT)

# Inspect the combined data
head(LEIA_analysis_DT)
```

```
##              Species                              Ind Dist    ROI  Col mean    Col sd  Col CoV  Col sk
## 1: aphelodoris_varia 3_AphelodorisVaria_10_NF_D_RAW 30cm animal 0.1504216 0.2447160 1.626867 2.31024
## 2: aphelodoris_varia 3_AphelodorisVaria_10_NF_D_RAW 50cm animal 0.1425384 0.2255096 1.582097 2.08174
## 3: aphelodoris_varia 3_AphelodorisVaria_11_NF_D_RAW 30cm animal 0.2430781 0.3859889 1.587922 2.02882
## 4: aphelodoris_varia 3_AphelodorisVaria_11_NF_D_RAW 50cm animal 0.2406384 0.3585004 1.489789 1.65948
## 5:        aplysia_sp         4_Aplysiasp_1_NL_N_RAW 30cm animal 0.2599978 0.4975240 1.913570 1.03517
## 6:        aplysia_sp         4_Aplysiasp_1_NL_N_RAW 50cm animal 0.1809975 0.3314044 1.830989 0.92637
##     Lum CoV  Lum skew Lum kurtosis Col mean hrz Col sd hrz Col CoV hrz Col skew hrz Col kurtosis hrz
## 1: 1.602808 2.2559735     9.558368   0.07536838  0.1581647    2.098556     3.932707         23.11730
## 2: 1.536144 2.0603133     8.282279   0.06958630  0.1449846    2.083522     3.640064         19.88286
## 3: 1.616269 2.5463566    12.453982   0.13050419  0.2706214    2.073661     3.607496         20.25214
## 4: 1.483030 2.0211324     8.743586   0.12954293  0.2590665    1.999850     3.248433         15.98547
## 5: 1.698786 0.4658983     2.619744   0.12837917  0.2877816    2.241653     2.211416         11.50972
## 6: 1.741279 0.6844910     3.147853   0.09196166  0.1943035    2.112876     2.338302         13.64586
##    Col skew vrt Col kurtosis vrt Lum mean hrz Lum sd hrz Lum CoV hrz Lum skew hrz Lum kurtosis hrz Lu
## 1:     3.877295         21.68659    0.4355562  0.8958863    2.056879     3.612730        20.015200
## 2:     3.585899         19.15019    0.4406635  0.8761384    1.988225     3.493638        19.455513
## 3:     3.405425         17.69506    0.8198832  1.7714397    2.160600     4.465145        31.933329
## 4:     2.908382         14.04169    0.8071571  1.6670532    2.065339     3.871669        23.224363
## 5:     2.321757         11.03054    0.4030806  0.7863049    1.950739     1.070465         4.311739
## 6:     2.322545         11.96114    0.4673162  0.9458438    2.023991     1.530686         7.331437
##    Lum skew vrt Lum kurtosis vrt Col thresh Lum thresh
## 1:     3.852029         22.42973          0          0
```

```
## 2:     3.690217         20.01364          0          0
## 3:     3.836817         23.39465          0          0
## 4:     3.191745         18.53788          0          0
## 5:     1.771279         10.55201          0          0
## 6:     2.801707         15.67831          0          0
```

```r
# Define the file name for saving
fname <- paste0(
  out_path, "/", gsub("_analysis", "", "LEIA_analysis_DT"),
  "_data", format(Sys.time(), "_%d_%b_%Y"), ".csv"
)

# Save the data to a CSV file
fwrite(LEIA_analysis_DT, file = fname, row.names = FALSE)
```

References

Dowle, Matt, and Arun Srinivasan. 2023. *Data.table: Extension of 'Data.frame'*. https://CRAN.R-project.org/package=data.table.

Harrison, Xavier A, Lynda Donaldson, Maria Eugenia Correa-Cano, Julian Evans, David N Fisher, Cecily ED Goodwin, Beth S Robinson, David J Hodgson, and Richard Inger. 2018. "A Brief Introduction to Mixed Effects Modelling and Multi-Model Inference in Ecology." *PeerJ* 6: e4794.

van den Berg, Cedric P, Jolyon Troscianko, John A Endler, N Justin Marshall, and Karen L Cheney. 2020. "Quantitative Colour Pattern Analysis (QCPA): A Comprehensive Framework for the Analysis of Colour Patterns in Nature." *Methods in Ecology and Evolution* 11 (2): 316–32.

Wickham, Hadley. 2014. *Advanced r*. 1st ed. The r Series. Philadelphia, PA: Chapman; Hall/CRC.