BIO104 Lab A2: Mechanisms of Evolution

Goals for Lab 2

Be able to...

- Create code that you will understand in the future (i.e. by including comments)
- Calculate summary statistics for various subsets of data (using dplyr in R)
- Plot line graphs (using ggplot)
- Plot different subsets of data together (colors in ggplot)
- Compare two datasets (using a t-test in R)

1. Getting Started

Make a new project in a new directory and open a new script.

Begin by loading all the libraries you need. In this case, we will be using `ggplot2` to make our graphs. We are also entering data into Google Sheets, so we will need `gsheet` to pull that data into R.

- Copy the comment lines into your script
- Enter the necessary code based on the previous lab and your R script from Lab A1. **If there are no code instructions then look at your Lab A1 materials**

```
# Load the packages ggplot2, gsheet, and dplyr using the library command
```

2. Get your data into R

We will use the same procedure as the last lab to load data into R. Go to the Lab Sakai Site and scroll down like you did last week. Select the link for your lab section.

```
# Store the Google sheets link as the variable url
```

```
# Load the data stored in the variable url
# using the gsheet2tbl function and storing it as the variable snail_data
```

Once we have our variables, it is always good to check to make sure the data was imported correctly. Click on the variables listed in the R Environment (top right) and compare them to the Google Sheet data. Another way to look at the first few lines of your data is using the head function:

```
head(snail_data)
```

These are column names. You will use them to analyze and plot the data.

- `exp` is the experiment number
- `group` is your group number
- `generation` is the number of generations from the beginning (0,1,2,3)
- `snailcolor` is camouflaged or obvious (white or red)
- `snails` is the number of snails

You can also see the column names (i.e. the variables) by using the `colnames` function.

```
colnames(snail_data)
```

```
## [1] "exp"        "group"      "generation" "snailcolor" "snails"
```

3. Find the average snail count in groups of data

We are interested in differences in population size between red and white snails over time (`snailcolor`), between different generations (`generation`), and between experiments (`exp`). To graph the average snail population size over time, you first need to calculate the averages for each snail type at each generation time for each experiment. We want to take the average of the data for each experiment / generation / color combination.

We will use the `group_by` function from the `dplyr` package to create a new variable where each subset of data is labeled. Tell `group_by` the name of the data you want to divide into subsets, followed by the columns you want to include.

```
# Group the snail_data variable by exp, generation, and snailcolor
# using the group_by function
grouped_snail_data <- group_by(snail_data, exp, generation, snailcolor)
```

Next we calculate the mean for each group, and put this into a new variable `snail_data_means` using the `summarise` function. Here the name of the column containing the mean number of snails (calculated using the function `mean`) is "mean".

```
# Calculate the mean number of snails for each group using the summarise function
# by giving the name of the data you want to summarize (`grouped_snail_data`)
# and that you want the mean value of the snail counts (labeled as `snails`).

snail_data_means <- summarise(grouped_snail_data,mean=mean(snails))
```

Click on the `snail_data_means` variable in the in the R environment (top right). What are your mean values for the data? Do they make sense?

4. Plot your data from Experiment 1

4a. Subset your data from Experiment 1

Just like last time, we are going to use `ggplot` to graph the data. (Remember, `ggplot2` is a package you load and `ggplot` is the function you use for graphing).

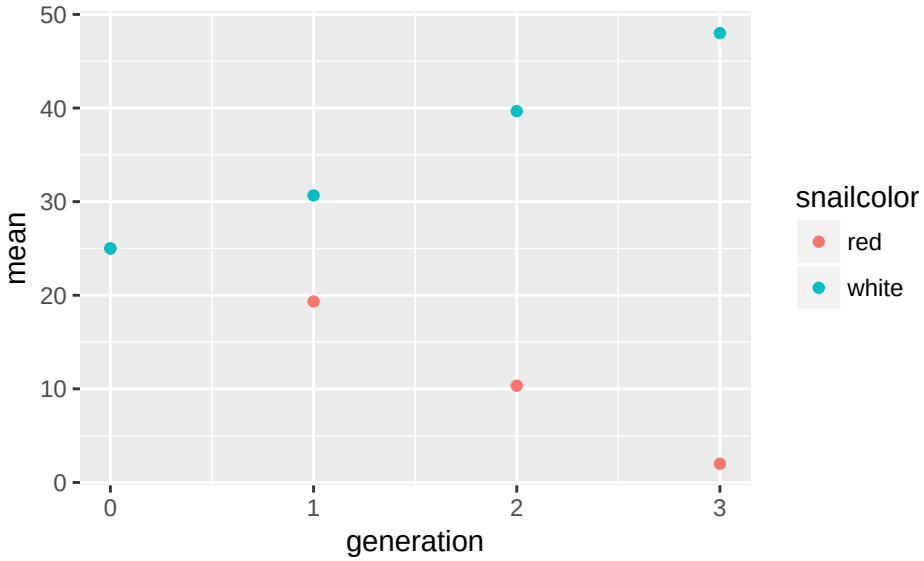
- Filter the mean data variable (`snail_data_means`) for just Experiment 1 (otherwise, you would graph both the data from all experiments, making a very confusing graph to look at).
- Assign the filtered data to a new variable named `snail_data_means_exp1`.

```
# Filter data to include only means for Experiment 1 (exp==1).
# Refer to Lab A1, Section 5d for the commands.
```

4b. Create the base layer of your plot and add points

- Use your new filtered dataset (`snail_data_means_exp1`)
- Add `color` to the `aes` function. The `color` variable separates the data based on `snailcolor` and then plots these separately on the same graph (in different colors!).

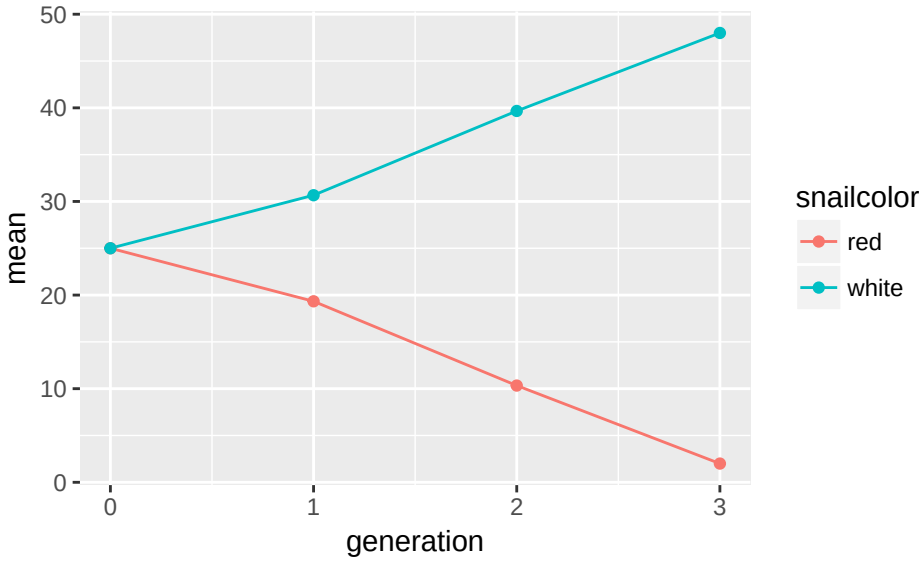
```
# Make your plot for the snail_data_means_exp1 data
ggplot(snail_data_means_exp1, aes(x=generation, y=mean, color=snailcolor)) + geom_point()
```



4c. Add a line to your plot to connect points

The function to add a line is `geom_line()` and it is added to the ggplot command just like you add `geom_point()`.

```
# Add a line to the plot
```

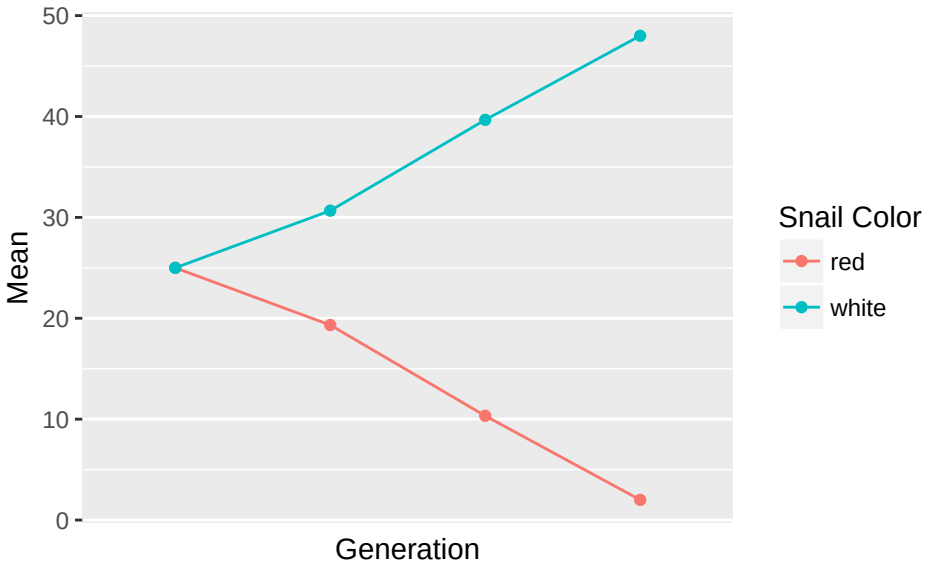


Note: `geom_line()` and `geom_point()` are followed by empty parentheses because they are functions. Functions must be able to accept arguments (e.g. the name of a dataset). These arguments need to go in the parentheses associated with the function. In this case, these functions do not have additional arguments, but we will see ones that do.

4d. Add labels to your plot and change axis titles

Label the axes using `scale_x_discrete` and `scale_y_continuous` like you did in Lab 1, Section 5b. Use `name=" "` to name the x and y axes. Label the legend using `+ labs(color="Snail Color")`.

```
#Label your plot and change axes titles
```



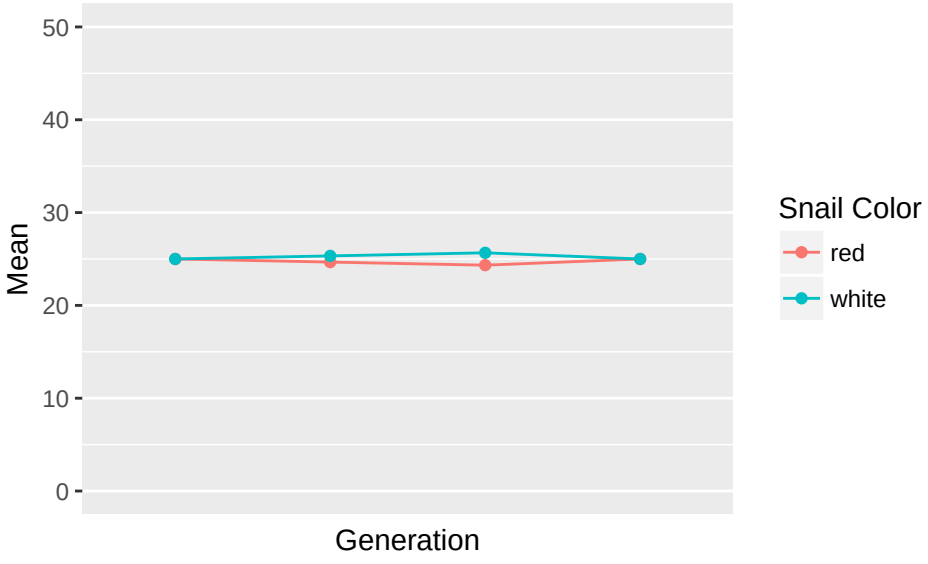
Save this plot as Lab1_Experiment1 so that you can turn it in with your lab report.

5. Plot your data from Experiment 2

- Repeat everything you did for Experiment 1 for Experiment 2.
- Make sure your axis and legend titles are correct.
- In your `scale_y_continuous` add an argument `limits = c(0,50)` to make sure your y-axis starts at 0. Note, the maximum for your y-axis may not be 50.

```
# Filter data to include only means for Experiment 2 and name this variable snail_data_means_exp2

# Graph Experiment 2 data using the same commands you used in Section 4.
```



Save this plot as Lab1_Experiment2 so that you can turn it in with your lab report.

Take a look at the scale of your plots. In many cases, your Experiment 2 data will have a much larger y-axis

than your Experiment 1 data. Why do you think this is? Note that these graphs do not show the variance in the data. We will show you how to add error bars in future labs.

6. Compare number of snails in different groups

Now that we have examined our data visually, we are interested in knowing if the number of snails of each color differs at the end of the experiment. To compare the number of each color, you will run a *t*-test. A *t*-test is a statistical analysis that compares the means of two groups to see if they are statistically different from one another. To run a *t*-test in R, we will go back to the original data set (`snail_data`).

First we filter our white and red snails from Generation 3 in both the Experiments 1 and 2. Use the command from `Red1` for `Red2`, `White1`, and `White2`.

```
# Subset snail data by generation, experiment and snail color and store it in the Red1 variable.
# Red1 contains just data from exp 1 red snails
Red1 <- filter(snail_data, exp==1 & generation==3 & snailcolor=="red")

# White1 contains just data from exp 1 white snails
# (write the command for White1 here)

# Red2 contains just data from exp 2 red snails

# White2 contains just data from exp 2 white snails
```

Now that we have separate variables for each subset of data, we can run our *t*-tests. Input the two groups you want to compare, and then select the type of *t*-test to use. For this class, we will be using the two-sample *t*-test because we have two samples (a one sample test compares your data to some expected value).

What is your null hypothesis for the Experiment 1 data?

```
# My null hypothesis is that
# red and white snail numbers are the same at the third generation of experiment 1.

# Oystercatcher_ttest
t.test(Red1$snails, White1$snails)
```

```
##
##  Welch Two Sample t-test
##
## data:  Red1$snails and White1$snails
## t = -56.338, df = 4, p-value = 5.943e-07
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -48.26696 -43.73304
## sample estimates:
## mean of x mean of y
##         2        48
```

What is your null hypothesis for the Experiment 2 data?

```
# My null hypothesis is:

# Driftlog_ttest
t.test(Red2$snails, White2$snails)
```

```
##
##  Welch Two Sample t-test
```

```
##
## data:  Red2$snails and White2$snails
## t = 0, df = 4, p-value = 1
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -8.173633  8.173633
## sample estimates:
## mean of x mean of y
##        25        25
```

The output from `t.test` gives you a lot of information. For this class, we are going to focus on the p-value. A p-value is the probability of getting data different from the observed data given that the null hypothesis is true.

Imagine you are willing to accept a 5% probability that you reject the null hypothesis if it is really true. That's like saying for 20 experiments where the null is true, one of them will probably appear as if the null is false ($1/20 = 5\%$). Because you have some variance in your data you need to allow for some probability of being wrong. If your p-value is less than 0.05 you will reject the null hypothesis. If it is greater than 0.05 you cannot reject the null hypothesis.

Lab 2 Report Submission

Turn in a hard copy of your lab report that includes the following:

1. Summary table for your section's data
2. Line graph of Experiment 1 means from Step 4d
3. Line graph of Experiment 2 means from Step 5
4. Results of your t-tests for both experiments from Step 6
5. Explain why you rejected or failed to reject your null hypotheses based on your t-test results and explain the processes that influenced both experiments.
6. Your code. Code should be organized, include good comments, and include only code that works and does not produce errors.