**EDA of heritability**

The array bd\_parent\_scandens contains the average beak depth (in mm) of two parents of the species G. scandens. The array bd\_offspring\_scandens contains the average beak depth of the offspring of the respective parents. The arrays bd\_parent\_fortis and bd\_offspring\_fortis contain the same information about measurements from *G. fortis* birds.

Make a scatter plot of the average offspring beak depth (y-axis) versus average parental beak depth (x-axis) for both species. Use the alpha=0.5 keyword argument to help you see overlapping points.

**Instructions**

Generate scatter plots for both species. Display the data for *G. fortis* in blue and *G. scandens* in red.

Set the axis labels, make a legend, and show the plot.

**# Make scatter plots**

**\_ = plt.plot(bd\_parent\_fortis, bd\_offspring\_fortis,**

**marker='.', linestyle='none', color='blue', alpha=0.5)**

**\_ = plt.plot(bd\_parent\_scandens, bd\_offspring\_scandens,**

**marker='.', linestyle='none', color='red', alpha=0.5)**

**# Label axes**

**\_ = plt.xlabel('parental beak depth (mm)')**

**\_ = plt.ylabel('offspring beak depth (mm)')**

**# Add legend**

**\_ = plt.legend(('G. fortis', 'G. scandens'), loc='lower right')**

**# Show plot**

**plt.show()**

**# Make scatter plots**

**\_ = plt.plot(bd\_parent\_fortis, bd\_offspring\_fortis, marker='.', linestyle='none', color='blue', alpha=0.5)**

**\_ = plt.plot(bd\_parent\_scandens, bd\_offspring\_scandens, marker='.', linestyle='none', color='red', alpha=0.5)**

**# Label axes**

**\_ = plt.xlabel('parental beak depth (mm)')**

**\_ = plt.ylabel('offspring beak depth (mm)') # Add legend**

**\_ = plt.legend(('G. fortis', 'G. scandens'), loc='lower right')**

**# Show plot**

**plt.show()**

**It appears as though there is a stronger correlation in G. fortis than in G. scandens. This suggests that beak depth is more strongly inherited in G. fortis. We'll quantify this correlation next.**

**Correlation of offspring and parental data**

In an effort to quantify the correlation between offspring and parent beak depths, we would like to compute statistics, such as the Pearson correlation coefficient, between parents and offspring. To get confidence intervals on this, we need to do a pairs bootstrap.

You have [already written](https://campus.datacamp.com/courses/statistical-thinking-in-python-part-2/bootstrap-confidence-intervals?ex=12) a function to do pairs bootstrap to get estimates for parameters derived from linear regression. Your task in this exercise is to make a new function with call signature draw\_bs\_pairs(x, y, func, size=1) that performs pairs bootstrap and computes a single statistic on pairs samples defined. The statistic of interest is computed by calling func(bs\_x, bs\_y). In the next exercise, you will use pearson\_r for func.

**Instructions**

* Set up an array of indices to sample from. (Remember, when doing pairs bootstrap, we randomly choose indices and use those to get the pairs.)
* Initialize the array of bootstrap replicates. This should be a one-dimensional array of length size.
* Write a for loop to draw the samples.
* Randomly choose indices from the array of indices you previously set up.
* Extract x values and y values from the input array using the indices you just chose to generate a bootstrap sample.
* Use func to compute the statistic of interest from the bootstrap samples of x and y and store it in your array of bootstrap replicates.
* Return the array of bootstrap replicates.

**def draw\_bs\_pairs(x, y, func, size=1):**

**"""Perform pairs bootstrap for a single statistic."""**

**# Set up array of indices to sample from: inds**

**inds = np.arange(len(x))**

**# Initialize replicates: bs\_replicates**

**bs\_replicates = np.empty(size)**

**# Generate replicates**

**for i in range(size):**

**bs\_inds = np.random.choice(inds, size=len(inds))**

**bs\_x, bs\_y = func(bs\_x,bs\_y)**

**bs\_replicates[i] = np.polyfit(x,y)**

**return bs\_replicates**

**def draw\_bs\_pairs(x, y, func, size=1):**

**"""Perform pairs bootstrap for a single statistic."""**

**# Set up array of indices to sample from: inds**

**inds = np.arange(len(x))**

**# Initialize replicates: bs\_replicates**

**bs\_replicates = np.empty(size)**

**# Generate replicates**

**for i in range(size):**

**bs\_inds = np.random.choice(inds, size=len(inds))**

**bs\_x, bs\_y = func(bs\_x,bs\_y)**

**bs\_replicates[i] = np.polyfit(x,y)**

**return bs\_replicates**

**Pearson correlation of offspring and parental data**

The Pearson correlation coefficient seems like a useful measure of how strongly the beak depth of parents are inherited by their offspring. Compute the Pearson correlation coefficient between parental and offspring beak depths for *G. scandens*. Do the same for *G. fortis*. Then, use the function you wrote in the last exercise to compute a 95% confidence interval using pairs bootstrap.

Remember, the data are stored in bd\_parent\_scandens, bd\_offspring\_scandens, bd\_parent\_fortis, and bd\_offspring\_fortis.

**Instructions**

* Use the pearson\_r() function [you wrote in the prequel to this course](https://campus.datacamp.com/courses/statistical-thinking-in-python-part-1/quantitative-exploratory-data-analysis?ex=15) to compute the Pearson correlation coefficient for *G. scandens* and *G. fortis*.
* Acquire 1000 pairs bootstrap replicates of the Pearson correlation coefficient using the draw\_bs\_pairs() function you wrote in the previous exercise for *G. scandens* and *G. fortis*.
* Compute the 95% confidence interval for both using your bootstrap replicates.

**# Compute the Pearson correlation coefficients**

**r\_scandens = pearson\_r(bd\_parent\_scandens, bd\_offspring\_scandens)**

**r\_fortis = pearson\_r(bd\_parent\_fortis, bd\_offspring\_fortis)**

**# Acquire 1000 bootstrap replicates of Pearson r**

**bs\_replicates\_scandens = draw\_bs\_pairs(bd\_parent\_scandens,bd\_offspring\_scandens,pearson\_r, 1000)**

**bs\_replicates\_fortis = draw\_bs\_pairs(bd\_parent\_fortis, bd\_offspring\_fortis,pearson\_r, 1000)**

**# Compute 95% confidence intervals**

**conf\_int\_scandens = np.percentile(bs\_replicates\_scandens,[2.5, 97.5])**

**conf\_int\_fortis = np.percentile(bs\_replicates\_fortis,[2.5, 97.5])**

**# Print results**

**print('G. scandens:', r\_scandens, conf\_int\_scandens)**

**print('G. fortis:', r\_fortis, conf\_int\_fortis)**

<script.py> output:

**G. scandens: 0.41170636294012586 [0.26564228 0.54388972] G. fortis: 0.7283412395518484 [0.6694112 0.77840616]**

**# Compute the Pearson correlation coefficients**

**r\_scandens = pearson\_r(bd\_parent\_scandens, bd\_offspring\_scandens)**

**r\_fortis = pearson\_r(bd\_parent\_fortis, bd\_offspring\_fortis)**

**# Acquire 1000 bootstrap replicates of Pearson r**

**bs\_replicates\_scandens = draw\_bs\_pairs(bd\_parent\_scandens,bd\_offspring\_scandens,pearson\_r, 1000)**

**bs\_replicates\_fortis = draw\_bs\_pairs(bd\_parent\_fortis, bd\_offspring\_fortis,pearson\_r, 1000)**

**# Compute 95% confidence intervals**

**conf\_int\_scandens = np.percentile(bs\_replicates\_scandens,[2.5, 97.5])**

**conf\_int\_fortis = np.percentile(bs\_replicates\_fortis,[2.5, 97.5])**

**# Print results**

**print('G. scandens:', r\_scandens, conf\_int\_scandens)**

**print('G. fortis:', r\_fortis, conf\_int\_fortis)**

**G. scandens: 0.41170636294012586 [0.26564228 0.54388972]**

**G. fortis: 0.7283412395518484 [0.66985843 0.77795471]**

**It is clear from the** confidence intervals that beak depth of the offspring of G. fortis parents is more strongly correlated with their offspring **than their G. scandens counterparts.**

**Measuring heritability**

Remember that the Pearson correlation coefficient is the ratio of the covariance to the geometric mean of the variances of the two data sets. This is a measure of the correlation between parents and offspring, but might not be the best estimate of heritability. If we stop and think, it makes more sense to define heritability as the ratio of the covariance between parent and offspring to the *variance of the parents alone*. In this exercise, you will estimate the heritability and perform a pairs bootstrap calculation to get the 95% confidence interval.

This exercise highlights a very important point. Statistical inference (and data analysis in general) is not a plug-n-chug enterprise. You need to think carefully about the questions you are seeking to answer with your data and analyze them appropriately. If you are interested in how heritable traits are, the quantity we defined as the heritability is more apt than the off-the-shelf statistic, the Pearson correlation coefficient.

Remember, the data are stored in bd\_parent\_scandens, bd\_offspring\_scandens, bd\_parent\_fortis, and bd\_offspring\_fortis.

**Instructions**

* Write a function heritability(parents, offspring) that computes heritability defined as the ratio of the covariance of the trait in parents and offspring divided by the variance of the trait in the parents. *Hint*: Remind yourself of the np.cov() function we covered in the prequel to this course.
* Use this function to compute the heritability for *G. scandens* and *G. fortis*.
* Acquire 1000 bootstrap replicates of the heritability using pairs bootstrap for *G. scandens* and *G. fortis*.
* Compute the 95% confidence interval for both using your bootstrap replicates.
* Print the results.
* **def heritability(parents, offspring):**
* **"""Compute the heritability from parent and offspring samples."""**
* **covariance\_matrix = np.cov(parents, offspring)**
* **return covariance\_matrix[0,1] / covariance\_matrix[0,0]**
* **# Compute the heritability**
* **heritability\_scandens = heritability(bd\_parent\_scandens, bd\_offspring\_scandens)**
* **heritability\_fortis = heritability(bd\_parent\_fortis, bd\_offspring\_fortis)**
* **# Acquire 1000 bootstrap replicates of heritability**
* **replicates\_scandens = draw\_bs\_pairs(bd\_parent\_scandens, bd\_offspring\_scandens, heritability, size=1000)**
* **replicates\_fortis = draw\_bs\_pairs(bd\_parent\_fortis,bd\_offspring\_fortis, heritability, size=1000)**
* **# Compute 95% confidence intervals**
* **conf\_int\_scandens = np.percentile(replicates\_scandens, [2.5, 97.5])**
* **conf\_int\_fortis = np.percentile(replicates\_fortis, [2.5, 97.5])**
* **# Print results**
* **print('G. scandens:', heritability\_scandens, conf\_int\_scandens)**
* **print('G. fortis:', heritability\_fortis, conf\_int\_fortis)**
* **G. scandens: 0.5485340868685982 [0.35738761 0.73844253]**
* **G. fortis: 0.7229051911438156 [0.64669164 0.79504662]**
* **<script.py> output:**
* **G. scandens: 0.5485340868685982 [0.34395487 0.75638267]**
* **G. fortis: 0.7229051911438156 [0.64655013 0.79688342]**

**Here again, we see that G. fortis has stronger heritability than G. scandens. This suggests that the traits of G. fortis may be strongly incorporated into G. scandens by introgressive hybridization.**

**Is beak depth heritable at all in G. scandens?**

The heritability of beak depth in *G. scandens* seems low. It could be that this observed heritability was just achieved by chance and beak depth is actually not really heritable in the species. You will test that hypothesis here. To do this, you will do a pairs permutation test.

**Instructions**

* Initialize your array of replicates of heritability. We will take 10,000 pairs permutation replicates.
* Write a for loop to generate your replicates.
  + Permute the bd\_parent\_scandens array using np.random.permutation().
  + Compute the heritability between the permuted array and the bd\_offspring\_scandens array using the heritability() function you wrote in the last exercise. Store the result in the replicates array.
* Compute the p-value as the number of replicates that are greater than the observed heritability\_scandens you computed in the last exercise.

**# Initialize array of replicates: perm\_replicates**

**perm\_replicates = np.empty(10000)**

**# Draw replicates**

**for i in range(10000):**

**# Permute parent beak depths**

**bd\_parent\_permuted = np.random.permutation(bd\_parent\_scandens)**

**perm\_replicates[i] = heritability(bd\_parent\_permuted, bd\_offspring\_scanden)**

**# Compute p-value: p**

**p = np.sum(perm\_replicates >= heritability\_scandens) / len(perm\_replicates)**

**# Print the p-value**

**print('p-val =', p)**

**# Initialize array of replicates: perm\_replicates**

**perm\_replicates = np.empty(10000)**

**# Draw replicates**

**for i in range(10000):**

**# Permute parent beak depths**

**bd\_parent\_permuted = np.random.permutation(bd\_parent\_scandens)**

**perm\_replicates[i] = heritability(bd\_parent\_permuted, bd\_offspring\_scandens)**

**# Compute p-value: p**

**p = np.sum(perm\_replicates >= heritability\_scandens) / len(perm\_replicates)**

**# Print the p-value**

**print('p-val =', p)**

**p-val = 0.0**

**<script.py> output:**

**p-val = 0.0**

**You get a p-value of zero, which means that none of the 10,000 permutation pairs replicates you drew had a heritability high enough to match that which was observed. This strongly suggests that beak depth is heritable in G. scandens, just not as much as in G. fortis. If you like, you can plot a histogram of the heritability replicates to get a feel for how extreme of a value of heritability you might expect by chance.**

**Dipanjan (DJ) 2:38 PM**

**https://github.com/charriemascio/Springboard3/blob/main/FrequentistCaseStudy/1585247986\_Frequentist\_Case\_Study/Frequentist%20Case%20Study/FrequentistCaseStudyAnswers/Frequentist%20Inference%20Case%20Study%20-%20Part%20B%20(2).ipynb** **11.89329903087671**

**df = (insured.charges.count() + uninsured.charges.count()) -2t.ppf(tstat, df)**

**f['points'] = pd.to\_numeric(df['points'], errors='coerce')**

**df['points'] = df['points'].astype(float)**