Assignment 2

Statistical inference

Rasmus Hammar

Imports

```
import numpy as np
import matplotlib.pyplot as plt
```

Part A

A:1

a)

Generate data `D` with N = 20 from N(mu = 5, sigma = 2.7)

```
N = 20
mu = 5
sigma = 2.7
D = np.random.normal(mu, sigma, N)
```

b)

```
D_mean = np.mean(D)
D_sd = np.std(D)

print("D mean = ", D_mean)
print("D standard deviation = ", D_sd)
```

```
D mean =  5.183966036765218
D standard deviation =  3.2441924084485483
```

c)

```
N_b = 1000
B = []
for i in range(N_b):
    d_star = np.random.choice(D, size=20, replace=True)
```

```
    d_star_mean = np.mean(d_star)
    d_star_sd = np.std(d_star)
    B.append([d_star, d_star_mean, d_star_sd])
```

d)

```
B_mean_list = [b[1] for b in B]
b_ci_lower, b_ci_upper = np.percentile(B_mean_list, [2.5, 97.5])
print("B ci [{:.2f}, {:.2f}]".format(b_ci_lower, b_ci_upper))
```

```
B ci [3.83, 6.70]
```

e)

```
N = 2000
mu = 5
sigma = 2.7
D = np.random.normal(mu, sigma, N)

D_mean = np.mean(D)
D_sd = np.std(D)

print("D mean = ", D_mean)
print("D standard deviation = ", D_sd)

N_b = 1000
B = []
for i in range(N_b):
    b_star = np.random.choice(D, size=len(D), replace=True)
    b_star_mean = np.mean(b_star)
    b_star_sd = np.std(b_star)
    B.append([b_star, b_star_mean, b_star_sd])

B_mean_list = [b[1] for b in B]
b_ci_lower, b_ci_upper = np.percentile(B_mean_list, [2.5, 97.5])
print("B ci [{:.2f}, {:.2f}]".format(b_ci_lower, b_ci_upper))
```

```
D mean =  4.972103123903086
D standard deviation =  2.6934162130060253
B ci [4.86, 5.10]
```

The CI is narrower due to larger sample size making the distribution of the sample taller and narrower.

A:2

```
X = [13.2, 8.2, 10.9, 14.3, 10.7, 6.6, 9.5, 10.8, 8.8, 13.3]
Y = [14.0, 8.8, 11.2, 14.2, 11.8, 6.4, 9.8, 11.3, 9.3, 13.6]
```

a)

```
plt.scatter(X, Y)
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

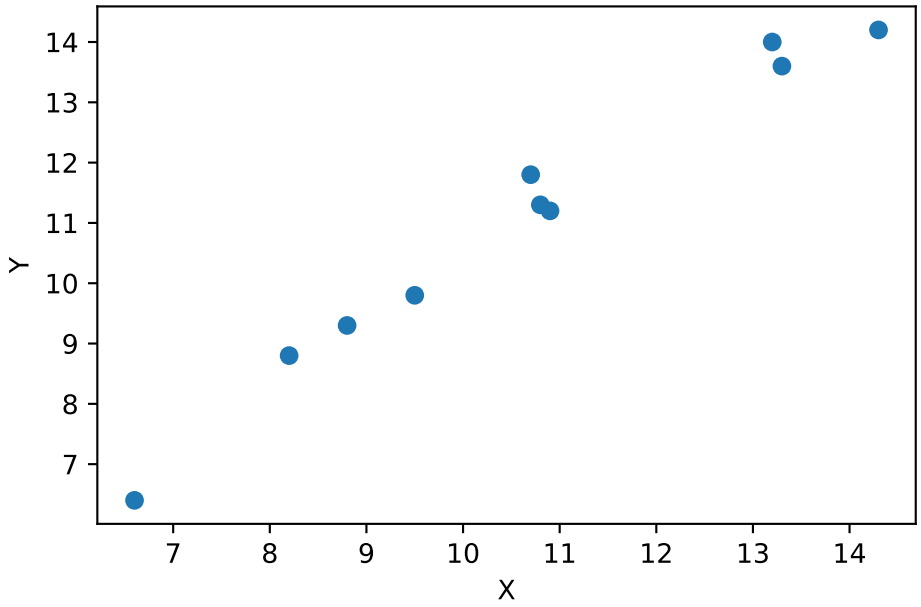

Figure 1: Scatterplot of X & Y graphically indicating a strong positive correlation.

They appear to be pretty strongly positively correlated.

b)

```
corr_xy = np.corrcoef(X, Y)[0, 1]
print("Pearson correlation between X and Y: {:.3f}".format(corr_xy))
```

```
Pearson correlation between X and Y: 0.988
```

The Pearson correlation indicates a correlation between X and Y.

c)

```
N_b = 1000
B = []
```

```
ind = [i for i in range(len(X))]
for i in range(N_b):
    ind_xy = np.random.choice(ind, size=len(ind), replace=True)
    B_x_star = [X[i] for i in ind_xy]
    B_y_star = [Y[i] for i in ind_xy]
    B_corr_xy = np.corrcoef(B_x_star, B_y_star)[0, 1]
    B.append([B_x_star, B_y_star, B_corr_xy])

B_corr_list = [b[2] for b in B]
b_ci_lower, b_ci_upper = np.percentile(B_corr_list, [2.5, 97.5])
print("B ci [{:.3f}, {:.3f}]".format(b_ci_lower, b_ci_upper))
```

```
B ci [0.976, 0.998]
```

```
plt.hist(B_corr_list, bins=30, edgecolor="black")
plt.vlines(b_ci_lower, ymin=0, ymax=100, colors="red", linestyles="dashed")
plt.vlines(b_ci_upper, ymin=0, ymax=100, colors="red", linestyles="dashed")
plt.xlabel("Bootstrap correlations")
plt.ylabel("Count")
plt.show()
```

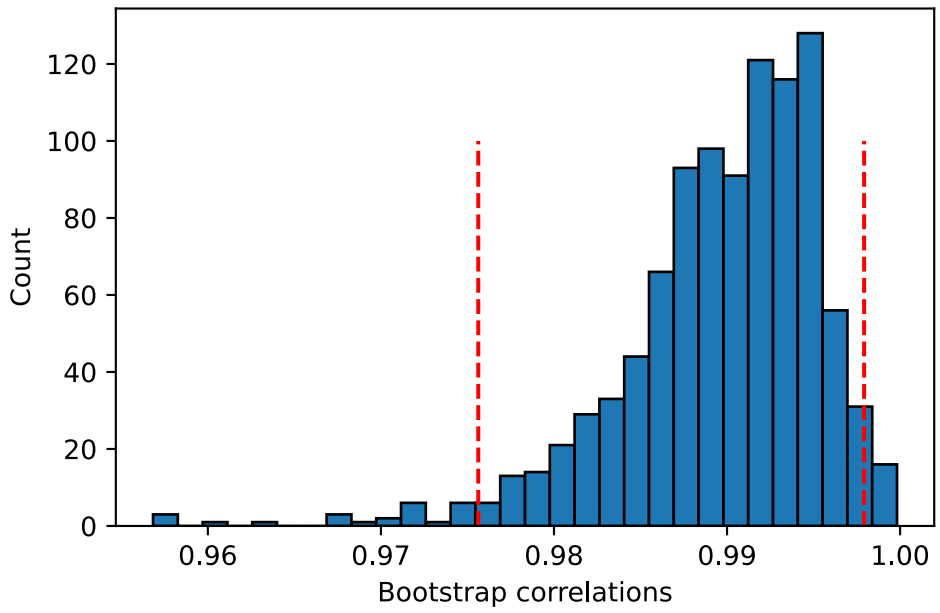

Figure 2: Correlations from bootstraping with 95 % confidence interval.

The CI does not include zero; variables X and Y are likely related.

A:3

i)

```
# Urn
values = np.asarray(["Red", "White"])
counts = np.asarray([14, 46])
urn_A3 = np.repeat(values, counts)

def rsamples(urn, n_samples):
    x = np.random.choice(urn, size=n_samples, p=None, replace=True)
    return x
```

ii)

```
N = 60
patient_samples = []
for i in range(1000):
    patient_sample = rsamples(urn_A3, N)
    patient_mutations = [True for m in patient_sample if m == "Red"]
    patient_samples.append([patient_sample, sum(patient_mutations)])
```

iii)

```
patient_mutation_fractions = [s[1] / N for s in patient_samples]
```

iv)

```
patient_mutations_ci = np.percentile(patient_mutation_fractions, [2.5, 97.5])
print("Patient mutation fraction CI [{:.3f},
{:.3f}]".format(patient_mutations_ci[0], patient_mutations_ci[1]))
```

```
Patient mutation fraction CI [0.133, 0.350]
```

```
plt.hist(patient_mutation_fractions, bins=20, edgecolor="black")
plt.vlines(patient_mutations_ci[0], ymin=0, ymax=100, colors="red",
linestyles="dashed")
plt.vlines(patient_mutations_ci[1], ymin=0, ymax=100, colors="red",
linestyles="dashed")
plt.xlabel("proportions of mutations")
plt.ylabel("Count")
plt.show()
```

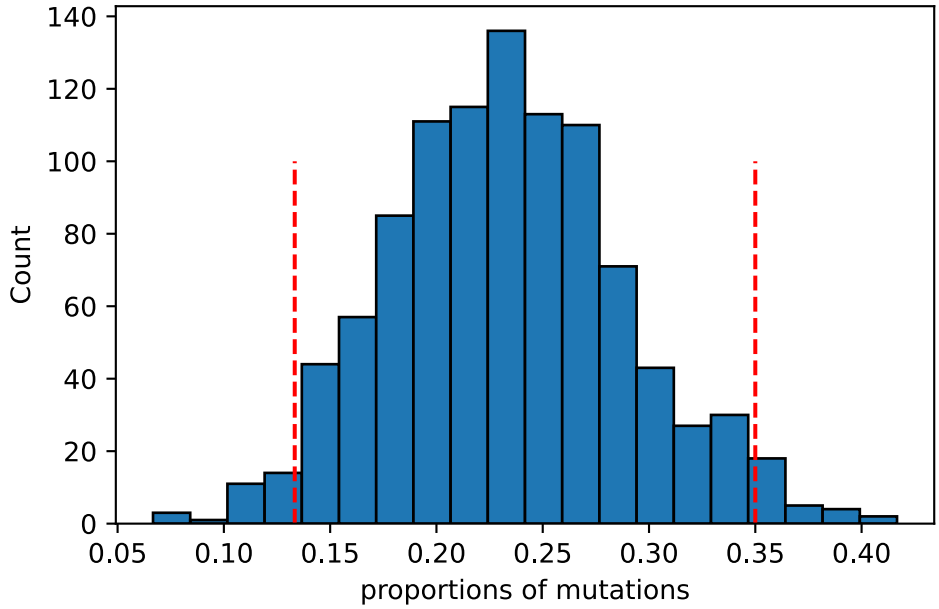

Figure 3: Bootstraping distribution of fraction mutations.

A:4

```
wells = 96
values = np.asarray(["Red", "White"])
counts = np.asarray([89, 7])
urn_A4 = np.repeat(values, counts)
```

```
plates_accuracy = []
for i in range(10000):
    plate = np.random.choice(urn_A4, size=wells, p=None, replace=True)
    plates_accuracy.append([sum([True for w in plate if w == "Red"])/wells])
plates_accuracy_ci = np.percentile(plates_accuracy, [2.5, 97.5])
print("Pipetting accuracy CI [{:.3f}, {:.3f}]".format(plates_accuracy_ci[0],
plates_accuracy_ci[1]))
```

```
Pipetting accuracy CI [0.875, 0.979]
```

Part B

B:1

```
# H_0 urn
values = np.asarray([1, 0])
```

```
counts = np.asarray([1, 1])
urn_B1_H0 = np.repeat(values, counts)
```

```
n = 20
guessing = []
for i in range(1000):
    guess = np.random.choice(urn_B1_H0, size=n, p=None, replace=True)
    guessing.append(sum(guess))
# guessing_ci = np.percentile(guessing, [2.5, 97.5])
guessing_pvalue = len([g for g in guessing if g <= 7]) / len(guessing)
```

```
plt.hist(guessing, bins=20, edgecolor="black")
plt.vlines(7, ymin=0, ymax=150, colors="red", linestyles="dashed")
plt.text(4.5, 120, "p = {:.3f}".format(guessing_pvalue), color="black")
plt.xlabel("Number of errors by guessing")
plt.ylabel("Count")
plt.show()
```

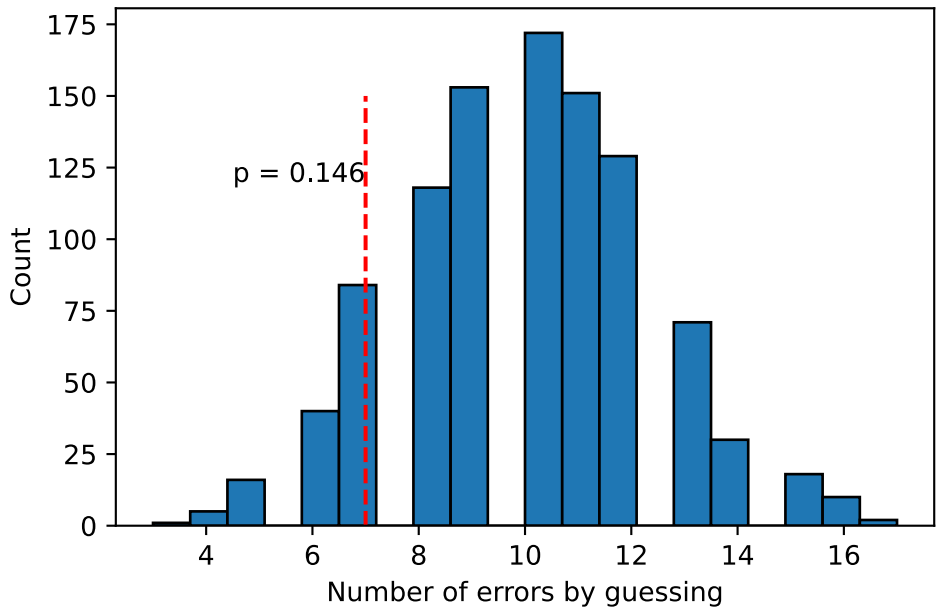

Figure 4: Distribution of number of errors.

With p = 0.146 I would reject the null hypothesis, but most likely consider the model subpar.

B:2

a)

```
# urn H0
values = np.asarray([1, 0]) # positive or negative test
counts = np.asarray([88, 12]) # H0 = 88% positive tests
urn_B2_H0 = np.repeat(values, counts)
```

```
# How often would I get 75% positive by chance if it should be 88%?
n = 1200
simulations = []
for i in range(1000):
    sim = np.random.choice(urn_B2_H0, size=n, p=None, replace=True)
    simulations.append(sum(sim) / n)  # fraction positive test
simulations_ci = np.percentile(simulations, [2.5, 97.5])
simulations_pvalue = (len([s for s in simulations if s <= 0.75])) /
len(simulations)
```

Based on the p-value being much smaller than any conventional threshold, I would reject the null hypothesis. The percentage of individuals in the population with antibodies for Epstein-Barr virus is reasonably lower than 88 % (Figure 5).

```
if simulations_pvalue < (1 / n):
    p_val = "p < {:.3f}".format((1 / n))
else:
    p_val = "p = {:.3f}".format(simulations_pvalue)

plt.hist(simulations, bins=25, edgecolor="black")
plt.vlines(simulations_ci[0], ymin=0, ymax=100, colors="blue",
linestyles="dashed")
plt.vlines(simulations_ci[1], ymin=0, ymax=100, colors="blue",
linestyles="dashed")
plt.vlines(0.75, ymin=0, ymax=100, colors="red", linestyles="dashed")
plt.text(0.76, 90, p_val, color="black")
plt.xlabel("Positive tests per 1200 individuals")
plt.ylabel("Count simulations")
plt.show()
```

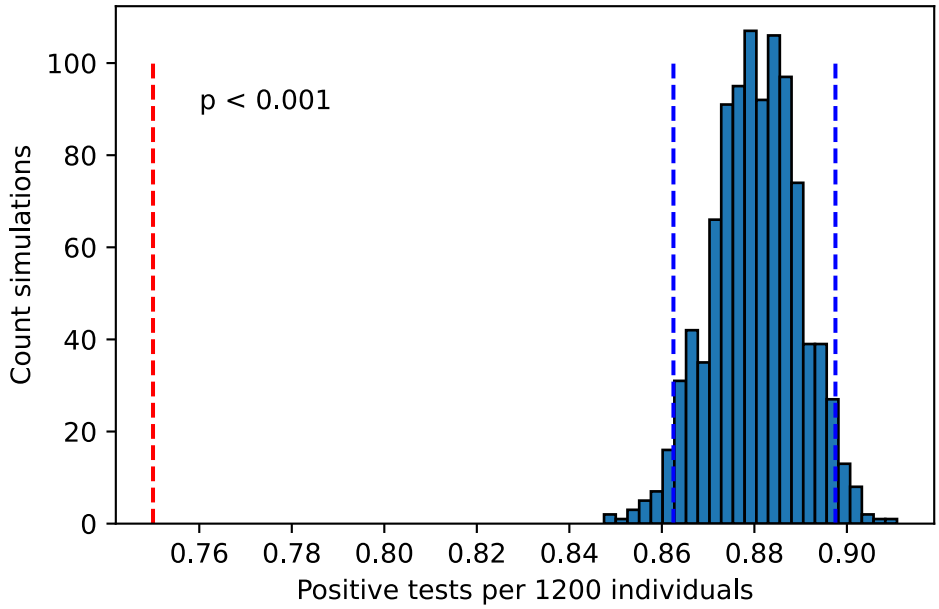

Figure 5: Distribution of positive tests per 1200 individuals over 1000 simulations.

b)

Doing the same analysis again, but with sample size n = 12.

```
# How often would I get 75% positive by chance if it should be 88%?
n = 12
simulations = []
for i in range(1000):
    sim = np.random.choice(urn_B2_H0, size=n, p=None, replace=True)
    simulations.append(sum(sim) / n)  # n positive test
simulations_ci = np.percentile(simulations, [2.5, 97.5])
simulations_pvalue = (len([s for s in simulations if s <= 0.75])) /
len(simulations)
```

However, with a smaller sample size, n = 12, the null hypothesis can not be rejected (Figure 6).

```
if simulations_pvalue < (1 / n):
    p_val = "p < {:.3f}".format((1 / n))
else:
    p_val = "p = {:.3f}".format(simulations_pvalue)

plt.hist(simulations, bins=10, edgecolor="black")
plt.vlines(simulations_ci[0], ymin=0, ymax=300, colors="blue")
plt.vlines(simulations_ci[1], ymin=0, ymax=300, colors="blue")
plt.vlines(0.75, ymin=0, ymax=300, colors="red", linestyles="dashed")
plt.text(0.76, 300, p_val, color="black")
plt.xlabel("Positive tests per 1200 individuals")
```

```
plt.ylabel("Count simulations")
plt.show()
```

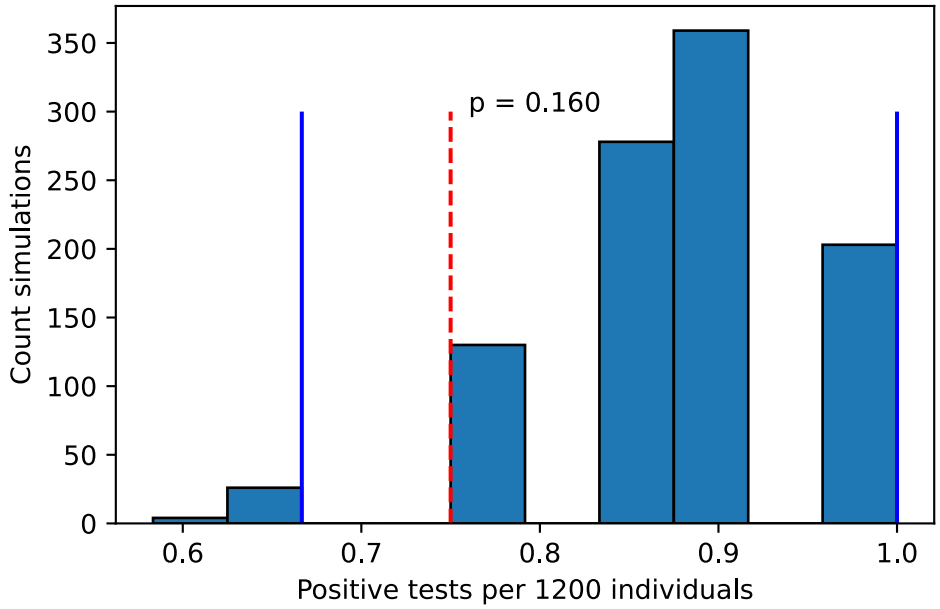

Figure 6: Distribution of positive tests per 1200 individuals over 1000 simulations.

B:3

```
# pg/L
women = [60, 62, 73, 81, 27, 72, 61, 59, 42, 39, 71, 74]
men = [74, 79, 80, 89, 71, 72, 92, 67, 91, 83, 43]
```

```
# Urn H0
pool = women + men
values = np.asarray(pool)
counts = np.asarray([1 for x in pool])
urn_B3_H0 = np.repeat(values, counts)
```

```
H1_diff = np.mean(women) - np.mean(men)
B_mean_diff = []
for i in range(10000):
    women_star = np.random.choice(urn_B3_H0, size=len(women), p=None,
replace=True)
    men_star = np.random.choice(urn_B3_H0, size=len(men), p=None,
replace=True)
    B_mean_diff.append(np.mean(women_star) - np.mean(men_star))
B_mean_diff_ci = np.percentile(B_mean_diff, [2.5, 97.5])
```

```
B_mean_diff_pvalue = len([d for d in B_mean_diff if abs(d) >= abs(H1_diff)]) /
len(
    B_mean_diff
)
```

The observed mean is outside the confidence interval, and the p-value is less than alpha = 0.05 (Figure 7). I would reject the null hypothesis.

Here we should definetly use a two-tailed test because we don't know if the hormone levels will be higher or lower depending on how the groups are compared. You should almost always use a two-tailed test, regardless.

```
plt.hist(B_mean_diff, bins=30, edgecolor="black")
plt.vlines(B_mean_diff_ci[0], ymin=0, ymax=800, colors="blue",
linestyles="dashed")
plt.vlines(B_mean_diff_ci[1], ymin=0, ymax=800, colors="blue",
linestyles="dashed")
plt.vlines(H1_diff, ymin=0, ymax=800, colors="red", linestyles="dashed")
plt.text(-25, 700, "p = {:.3f}".format(B_mean_diff_pvalue), color="black")
plt.xlabel("Mean difference in hormone levels")
plt.ylabel("Count simulations")
plt.show()
```

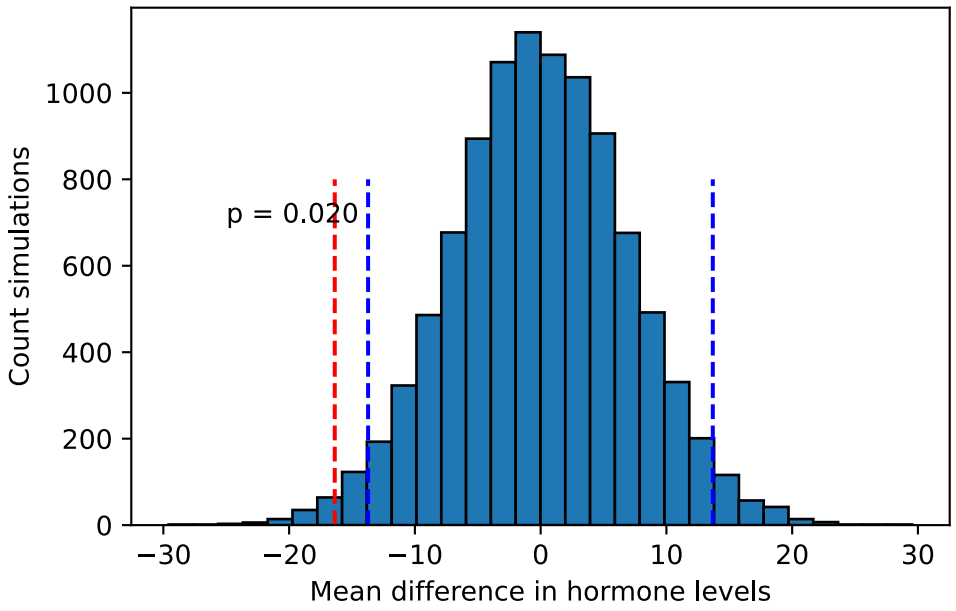

Figure 7: Distribution of mean differences in new hormone levels between men and women.

B:4

```
A = [13.2, 8.2, 10.9, 14.3, 10.7, 6.6, 9.5, 10.8, 8.8, 13.3]
B = [14.0, 8.8, 11.2, 14.2, 11.8, 6.4, 9.8, 11.3, 9.3, 13.6]
```

i)

```
def iqr(data):
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)
    return q3 - q1


d_mean_IQR = iqr(A) - iqr(B)
print("Difference in mean IQR between species A and B:
{:.3f}".format(d_mean_IQR))
```

```
Difference in mean IQR between species A and B: -0.075
```

ii)

```
# Urn H0
pool = A + B
values = np.asarray(pool)
counts = np.asarray([1 for x in pool])
urn_B4_H0 = np.repeat(values, counts)
```

iii)

```
B_mean_diff = []
for i in range(10000):
    A_star = np.random.choice(urn_B4_H0, size=len(A), p=None, replace=True)
    B_star = np.random.choice(urn_B4_H0, size=len(B), p=None, replace=True)
    B_mean_diff.append(iqr(A_star) - iqr(B_star))
B_mean_diff_ci = np.percentile(B_mean_diff, [2.5, 97.5])
B_mean_diff_pvalue = len([d for d in B_mean_diff if abs(d) >=
abs(d_mean_IQR)]) / len(B_mean_diff)
```

iv)

```
plt.hist(B_mean_diff, bins=30, edgecolor="black")
plt.vlines(B_mean_diff_ci[0], ymin=0, ymax=800, colors="blue")
plt.vlines(B_mean_diff_ci[1], ymin=0, ymax=800, colors="blue")
plt.vlines(d_mean_IQR, ymin=0, ymax=800, colors="red", linestyles="dashed")
plt.text(2, 900, "p = {:.3f}".format(B_mean_diff_pvalue), color="black")
plt.xlabel("Difference in mean IQR")
```

```
plt.ylabel("Count simulations")
plt.show()
```

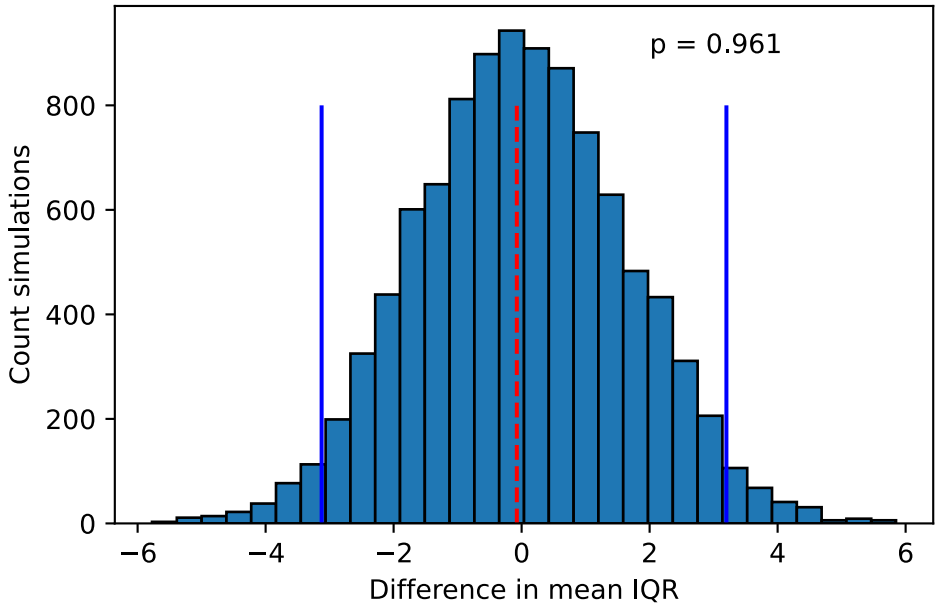

Figure 8: Distribution of difference in mean IQR between species A and B.

v)

With a p-value larger than alpha = 0.05, I would accept the null hypothesis. There is no difference in mean IQR between the two species.