

# CSCI3022 S22

## Homework 7: Bootstrapping and SLR

**Due Monday, April 18 at 11:59 pm to Canvas and Gradescope**

---

**Name:** Matthew Su

---

Your solutions to computational questions should include any specified Python code and results as well as written commentary on your conclusions. Remember that you are encouraged to discuss the problems with your classmates, but **you must write all code and solutions on your own.**

**NOTES:**

- Any relevant data sets should be available on Canvas. To make life easier on the graders if they need to run your code, do not change the relative path names here. Instead, move the files around on your computer.
- If you're not familiar with typesetting math directly into Markdown then by all means, do your work on paper first and then typeset it later. Here is a [reference guide](#) linked on Canvas on writing math in Markdown. **All** of your written commentary, justifications and mathematical work should be in Markdown. I also recommend the [wikibook](#) for LaTex.
- Because you can technically evaluate notebook cells in a non-linear order, it's a good idea to do **Kernel → Restart & Run All** as a check before submitting your solutions. That way if we need to run your code you will know that it will work as expected.
- It is **bad form** to make your reader interpret numerical output from your code. If a question asks you to compute some value from the data you should show your code output **AND** write a summary of the results in Markdown directly below your code.
- 45 points of this assignment are in problems. The remaining 5 are for neatness, style, and overall exposition of both code and text.
- This probably goes without saying, but... For any question that asks you to calculate something, you **must show all work and justify your answers to receive credit**. Sparse or nonexistent work will receive sparse or nonexistent credit.
- There is *not a prescribed API* for these problems. You may answer coding questions with whatever syntax or object typing you deem fit. Your evaluation will primarily live in the clarity of how well you present your final results, so don't skip over any interpretations! Your code should still be commented and readable to ensure you followed the given course algorithm.
- There are two ways to quickly make a .pdf out of this notebook for Gradescope submission.  
Either:

- Use File -> Download as PDF via LaTeX. This will require your system path find a working install of a TeX compiler
- Easier: Use File -> Print Preview, and then Right-Click -> Print using your default browser and "Print to PDF"

In [317...]

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
#regression stuff:
import statsmodels.api as sm
from patsy import dmatrices
%matplotlib inline
```

## Shortcuts: [Problem 1](#) | [Problem 2](#) |

[Back to top](#)

### Problem 1 (Small Samples; 20 points)

Consider the `hubble.csv` data set we used in nb15. In that example, we used the large-sample normal approximation and used it to calculate a confidence interval for the mean of a galaxy's distance from Earth in Mega parsecs (column `x` of the data).

Unfortunately, the data was only 24 observations! What could we have done instead?

In [318...]

```
# Path to the data - select the path that works for you
file_path = '../data/hubble.csv'

# Load the data into a DataFrame
df = pd.read_csv(file_path)
df.head(10)
```

Out[318...]

|   | Galaxy   | y    | x     |
|---|----------|------|-------|
| 0 | NGC0300  | 133  | 2.00  |
| 1 | NGC0925  | 664  | 9.16  |
| 2 | NGC1326A | 1794 | 16.14 |
| 3 | NGC1365  | 1594 | 17.95 |
| 4 | NGC1425  | 1473 | 21.88 |
| 5 | NGC2403  | 278  | 3.22  |
| 6 | NGC2541  | 714  | 11.22 |

| Galaxy | y       | x         |
|--------|---------|-----------|
| 7      | NGC2090 | 882 11.75 |
| 8      | NGC3031 | 80 3.63   |
| 9      | NGC3198 | 772 13.80 |

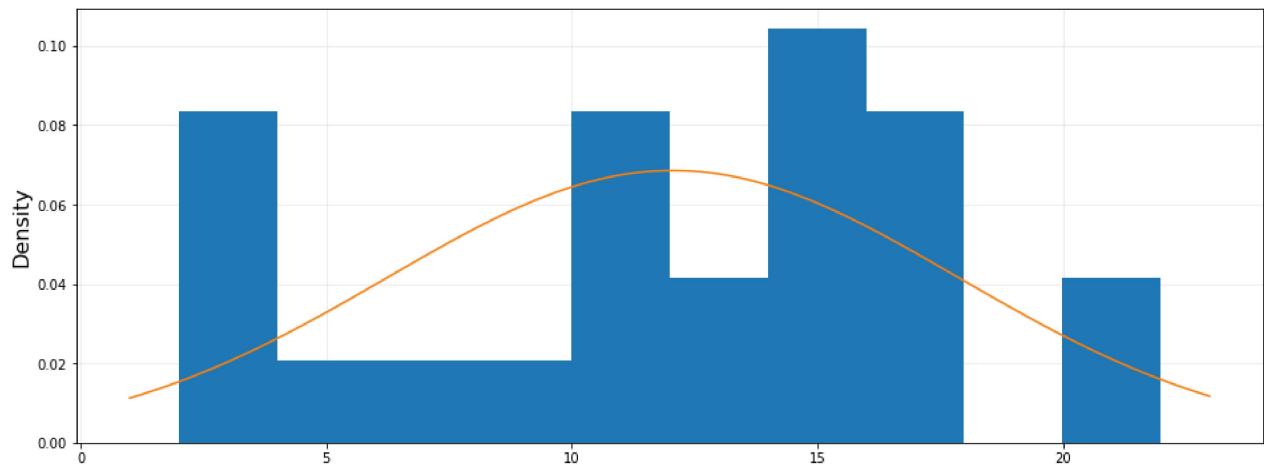
**Part A:**

Make a histogram of the data and overlay the theoretical density of the normal with mean  $\bar{X}$  and variance  $s^2$ . How closely do the data follow the histogram?

```
In [319...]: fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(16,6))
xbars = df['x']
# df.plot(x = 'x', y = 'y', kind = "hist" density = true)
ax.hist(x = df['x'], density = True, bins = 10)
ax.grid(alpha=0.25)
ax.set_axisbelow(True)
ax.set_ylabel("Density", fontsize=16)
# df['x']

# Overlay mean X and variance s^2
min, max = plt.xlim()
minmax = np.linspace(min,max,100)
pdf = stats.norm.pdf(minmax,df['x'].mean(),df['x'].std())
ax.plot(minmax,pdf)
```

Out[319...]: <matplotlib.lines.Line2D at 0x1b0823604f0>

**Part B:**

Things may not look perfectly normal in part A... but let's perform the tests that require normality anyways:

For Part B only, assume each observation is independent and identically distributed as a normal random variable.

- Create a 95% confidence interval for the mean of a galaxy's distance from Earth in Mega parsecs.

- Create a 95% confidence interval for the standard deviation of a galaxy's distance from Earth in Mega parsecs.

In [320...]

```
n=len(df)
xbar=np.sum(df['x'])/n
svar=np.sum((xi-xbar)**2/(n-1) for xi in df['x'])
sd=np.sqrt(svar)
print("Mean 95% CI: ",stats.norm.interval(.95, loc=xbar, scale=sd/np.sqrt(n)), "SD 95%"
np.sqrt(((n-1)*svar)/stats.chi2.ppf(0.975,n-1)), np.sqrt(((n-1)*svar)/stats.chi2.ppf(0.
```

Mean 95% CI: (9.728282115953515, 14.380884550713152) SD 95% CI: 4.519223941819863 8.15  
6560101133485

<ipython-input-320-670689cd945c>:3: DeprecationWarning: Calling np.sum(generator) is deprecated, and in the future will give a different result. Use np.sum(np.fromiter(generator)) or the python sum builtin instead.

```
svar=np.sum((xi-xbar)**2/(n-1) for xi in df['x'])
```

### Part C:

For part C only, assume each observation is independent and identically distributed but is **not** normally distributed. Instead, use bootstrapping to:

- Create a 95% confidence interval for the mean of a galaxy's distance from Earth in Mega parsecs.
- Create a 95% confidence interval for the standard deviation of a galaxy's distance from Earth in Mega parsecs.
- Create a 95% confidence interval for the interquartile range (both lower and upper quartiles) of a galaxy's distance from Earth in Mega parsecs.

Use 10,000 bootstrapped samples to make the statistics above.

In [321...]

```
# Adapted from notebook 18, modified for this problem

def bootstrapped_mean(sample, num_boots=10000):

    # np.mean of df['x']
    meanci = np.array([np.mean(np.random.choice(sample, replace=True, size=len(sample)))
    # 95% -> 2.5% --- 97.5%
    L, U = np.percentile(meanci, 2.5), np.percentile(meanci, 97.5)
    # Place Lower/upper into array and return
    CI = np.array([L, U])
    return CI

def bootstrapped_std(sample, num_boots=10000):

    # np.std of df['x']
    sdci = np.array([np.std(np.random.choice(sample, replace=True, size=len(sample))) for
    L, U = np.percentile(sdci, 2.5), np.percentile(sdci, 97.5)
    CI = np.array([L, U])

    return CI

def bootstrapped_iqr(sample, num_boots=10000):

    # Upper and Lower quartiles
```

```

lower = stats.iqr(np.array([np.std(np.random.choice(sample, replace=True, size=len(
upper = stats.iqr(np.array([np.std(np.random.choice(sample, replace=True, size=len(
L, U = np.percentile(lower, 2.5), np.percentile(upper, 97.5)
CI = np.array([L, U])

return CI

```

In [322...]

```

print("Bootstrap Mean CI : ", bootstrapped_mean(df['x']), " Bootstrap Std CI : ", bootst
Bootstrap Mean CI : [ 9.74955208 14.2879375 ] Bootstrap Std CI : [4.20331524 6.750150
74] Bootstrap UP/LOW IQR CI : [2.65862725 2.03305357]

```

[Back to top](#)

## (25 pts) Problem 2 (Theory): Practice with SLR and Inference

In this problem we will explore the `mortality.csv` data introduced in notebook21. In that notebook we found some linear regression coefficients and did a related hypothesis test. In this exercise we will "finish" analysis on that data set by exploring and visualizing the underlying assumptions, and performing some more nuanced inference.

This data set has two variables:

- Rate ; the annual mortality rate (as a percentage) over the years 1958–1964 of 61 towns in England and Wales.
- Calc ; the recorded calcium concentration in the drinking water supply for those towns, in parts-per-million

In [323...]

```

# Path to the data - select the path that works for you
file_path = '../data/mortality.csv'

# Load the data into a DataFrame
df = pd.read_csv(file_path)
print(len(df))
df.head(10)

```

61

Out[323...]

|   | Rate  | Calc |
|---|-------|------|
| 0 | 1.247 | 105  |
| 1 | 1.392 | 73   |
| 2 | 1.260 | 21   |
| 3 | 1.259 | 133  |
| 4 | 1.236 | 101  |
| 5 | 1.627 | 53   |
| 6 | 1.581 | 14   |

|   | Rate  | Calc |
|---|-------|------|
| 7 | 1.609 | 18   |
| 8 | 1.755 | 12   |
| 9 | 1.723 | 44   |

**A)** Perform a simple linear regression model using the `OLS` method from `statsmodels`. Print the resulting `summary` table and then as a print statement or in a markdown cell typeset the actual line of best fit, using the coefficients from your fit. (See: `.params`.)

In [324...]

```
# In notebook example, Calc is treated as x, Rate is treated as y
print((sm.regression.linear_model.OLS(df['Rate'], sm.add_constant(df['Calc'])).fit()).sum-
model = sm.regression.linear_model.OLS(df['Rate'], sm.add_constant(df['Calc'])).fit()
```

| OLS Regression Results |                  |                     |          |       |        |        |
|------------------------|------------------|---------------------|----------|-------|--------|--------|
| Dep. Variable:         | Rate             | R-squared:          | 0.429    |       |        |        |
| Model:                 | OLS              | Adj. R-squared:     | 0.419    |       |        |        |
| Method:                | Least Squares    | F-statistic:        | 44.30    |       |        |        |
| Date:                  | Mon, 18 Apr 2022 | Prob (F-statistic): | 1.03e-08 |       |        |        |
| Time:                  | 23:22:48         | Log-Likelihood:     | 33.088   |       |        |        |
| No. Observations:      | 61               | AIC:                | -62.18   |       |        |        |
| Df Residuals:          | 59               | BIC:                | -57.96   |       |        |        |
| Df Model:              | 1                |                     |          |       |        |        |
| Covariance Type:       | nonrobust        |                     |          |       |        |        |
|                        | coef             | std err             | t        | P> t  | [0.025 | 0.975] |
| const                  | 1.6764           | 0.029               | 57.217   | 0.000 | 1.618  | 1.735  |
| Calc                   | -0.0032          | 0.000               | -6.656   | 0.000 | -0.004 | -0.002 |
| Omnibus:               | 0.570            | Durbin-Watson:      | 1.624    |       |        |        |
| Prob(Omnibus):         | 0.752            | Jarque-Bera (JB):   | 0.694    |       |        |        |
| Skew:                  | 0.112            | Prob(JB):           | 0.707    |       |        |        |
| Kurtosis:              | 2.528            | Cond. No.           | 96.7     |       |        |        |

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

**Part B: Visualize.** Make a figure containing 3 side-by-side plots. Lefthand plot: scatter plot of the data set with overlay the line of best least-squares fit. Middle plot: histogram of *residuals* of best least-squares fit are the y-axis. Righthand plot: scatter plot where the original x-values are the x-axis and the *residuals* are the y-axis.

In [325...]

```
fig = plt.figure(figsize=(32,20))
fig.subplots_adjust(hspace=0.4, wspace=0.4)

ax1 = fig.add_subplot(331)
ax1.scatter(df["Calc"], df["Rate"], color="steelblue")
ax1.set_xlim([0,3])
ax1.set_xlabel("Calcium concentration")
ax1.set_ylabel("Mortality rate")
# Line of best Least square fit
bhat, ahat, rval, pval, stderr = stats.linregress(df['Calc'], df['Rate'])
```

```

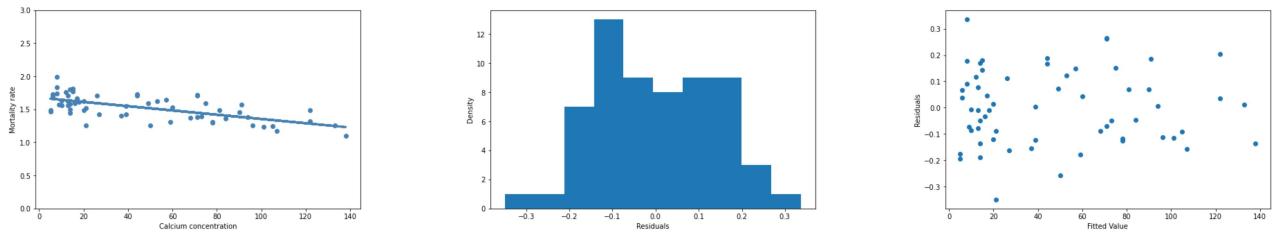
ax1.plot(df['Calc'], ahat + bhat*df['Calc'], color="steelblue", lw=3)

ax2 = fig.add_subplot(332)
ax2.hist(model.resid)
ax2.set_xlabel("Residuals")
ax2.set_ylabel("Density")

ax3 = fig.add_subplot(333)
ax3.scatter(df['Calc'],model.resid)
ax3.set_xlabel("Fitted Value")
ax3.set_ylabel("Residuals")

```

Out[325... Text(0, 0.5, 'Residuals')



**Part C:** For each of the 4 major assumptions of the simple linear regression model, use your plots in part B to decide whether or not the data set meets those assumptions.

### Solution:

- Linear Relationship (Are x and y related in any way?): Based on the leftmost graph 1, we can assume that there seems to be a fairly strong relationship between Calcium levels and the mortality rate.
- Independence (Is this data following a time specific pattern (ie seasonal, biannual, etc.)?): Our Durbin-Watson test returned a result of 1.624. Being within range of 1.5 and 2.5, that means that we don't have a serious autocorrelation issue in our data and is not a concern. We can safely assume that our results are independent and are not part of a time related pattern.
- Homoscedasticity (Is our variance constant at every level of x?): The rightmost graph 3 gives us a good idea of if our data suffers from heteroscedasticity, making it harder to trust. Fortunately, our values deviate on a scale of tenths, so if adjusted to larger y values, we would see very little deviation from 0. Thus, we can assume that our data reflected homoscedasticity and will not be problematic.
- Normality (Are the residuals normally distributed?): Our Jarque-Bera test resulted in a value of 0.694, which is relatively close to zero. That means that we can assume that the dataset and the residuals are distributed normally and won't be a problem. We can also see in the middle graph 2 that our distribution is fair, not the greatest, but with no glaring issues either. This is reflected in the earlier mentioned JB value.

**D:** In the rest of this problem we will draw specific conclusions from our analysis and linear model. For each prompt, you should provide both a numerical answer and a full sentence explaining the conclusion.

**D1):** How much of the variance in the mortality rates could be explained by the calcium in the drinking water? Does this seem like a high value?

```
In [326...]: print("The R-squared value from our OLS calculation comes out to ", model.rsquared, " w
```

The R-squared value from our OLS calculation comes out to 0.42882671931249716 which means that roughly 43%' of the variance that we saw can be explained by the input variables. This is notable, as it means that Calcium does have an effect on the variance that we saw for the mortality rates of citizens of all towns.

**D2):** A skeptic insists that "there's no way water quality is actually affecting death rates." You decide that your linear model demonstrates otherwise. State the null and alternative hypotheses, and use the values on your summary table at a significance of  $\alpha = 0.01$  to argue against this skeptic.

```
In [327...]:
```

```
print(model.pvalues)
```

```
const      2.165827e-53
Calc       1.033134e-08
dtype: float64
```

$$H_0 : \beta = 0$$

$$H_1 : \beta \neq 0$$

We state our null and alternate hypothesis as follows above. We can gather from the summary that our p value is extremely close to 0 and less than  $\alpha = 0.01$ . Therefore the idea that water quality is affecting mortality rates is backed up by our data.

**D3):** The government decides they believe your analysis in **D2)** that Calc values matter, but augmenting the drinking supply with Calcium is costly. They insist that they don't actually want to take any actions unless you can demonstrate that the effects of Calc concentration on mortality Rate are *large enough*. In particular, they will implement a calcification campaign if and only if you can demonstrate that at a significance of  $\alpha = 0.01$  one part-per-million of Calcium decreases mortality rate by more than 0.0025 percentage.

State the null and alternative hypotheses, and use the values on your summary table to recommend an action to the government (you may want to check out .bse of your linear model)

```
In [328...]:
```

```
# Var taken from nb21

bhat, ahat, rval, pval, stderr = stats.linregress(df['Calc'], df['Rate'])

# For SEbeta
yhat = ahat + bhat*df['Calc']
SSE = np.sum((df['Rate']-yhat)**2)
varhat = SSE / (len(df['Calc'])-2)

SEbeta = np.sqrt(varhat / np.sum((df['Calc']-np.mean(df['Calc']))**2))

Tb = (bhat - 0)/SEbeta
print("test statistic Tb = {}".format(Tb))

tpval = 2*(stats.t.cdf(Tb, df=len(df['Calc'])-2))
print("p-value = {}".format(tpval))

test statistic Tb = -6.6555351748006455
p-value = 1.0331339260349543e-08
```

$$H_0 : \beta = -0.0025$$

$$H_1 : \beta > 0.0025$$

A hypothesis test was done to check and validate the values of the summary in part B. The p value that we generated from the test is significantly less than  $\alpha = 0.01$ , signifying that the results of our data are statistically significant and thus supports our null hypothesis. We can therefore assure the proven notion that water quality indeed affects mortality rate at a large enough rate as proven by our data.

**D4):** The town of Manchester and their football clubs are loathsome. While the town wasn't originally tested, we are told that the calcium concentration in their drinking water is exactly 100 ppm.

What is the 95% **prediction** band for the mortality rate in Manchester? In other words, given our linear model, what is the interval that we think should contain the actual mortality rate for Manchester?

What is the 95% **confidence** band for the mortality rate in Manchester? In other words, what is the interval that we think should contain the **mean** mortality rate for any town with a Calcium concentration of 100 ppm?

*Hint:* These are the things computed in the SLR Confidence and Prediction.ipynb ... make sure to include an intercept!

In [329...]

```
#Proper model:

# Taken from SLR Confi/Predict
y = df['Rate']
x = df['Calc']
lm4=model
xgrd=np.linspace(0,150,150)
newpred=lm4.get_prediction(sm.add_constant(xgrd))
predmat=newpred.summary_frame(alpha=0.05)

# arr = predmat.mean_ci_lower
# print(predmat.mean_ci_lower[100])

print("CI lower,upper interval: ",(predmat.mean_ci_lower[100]), (predmat.mean_ci_upper[100]))
print("Pred lower,upper interval: ",(predmat.obs_ci_lower[100]),(predmat.obs_ci_upper[100]))

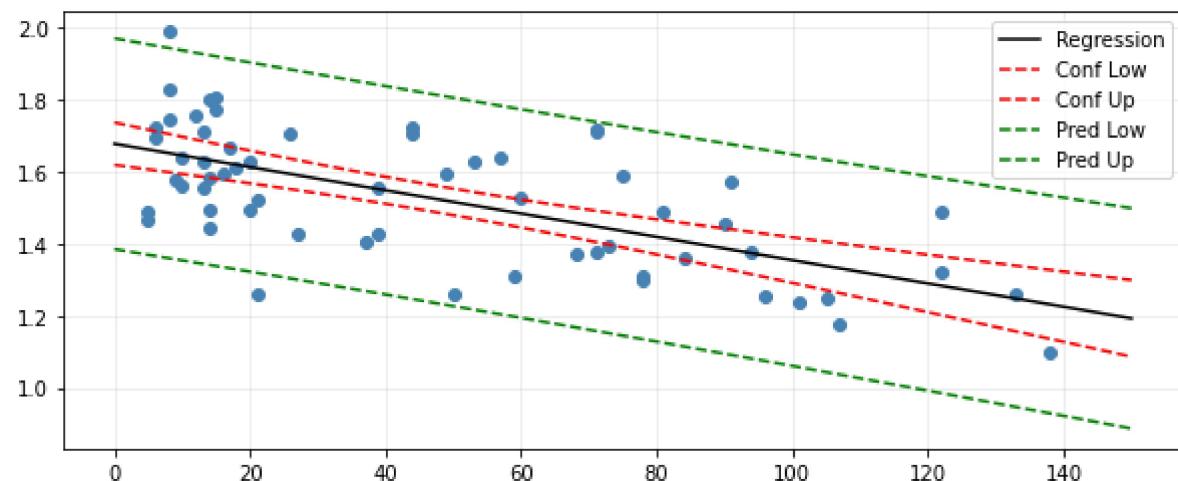
#plot it all

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(10,4))
ax.grid(alpha=0.25)
ax.set_axisbelow(True)
ax.scatter(df["Calc"], df["Rate"], color="steelblue")

#Plot each column
plt.plot(xgrd,(predmat.mean_ci_lower+predmat.mean_ci_upper)/2, color='black', label='Re')
plt.plot(xgrd,predmat.mean_ci_lower, 'r--', label='Conf Low')
plt.plot(xgrd,predmat.mean_ci_upper, 'r--', label='Conf Up')
plt.plot(xgrd,predmat.obs_ci_lower, 'g--', label='Pred Low')
plt.plot(xgrd,predmat.obs_ci_upper, 'g--', label='Pred Up')
plt.legend()
```

CI lower,upper interval: 1.2880629603694476 1.415099595480355  
Pred lower,upper interval: 1.0584161943058488 1.6447463615439537

Out[329... <matplotlib.legend.Legend at 0x1b08283c250>



[Back to top](#)