Homework 9: Correlation, Regression, and Least Squares

Please complete this notebook by filling in the cells provided. When you're done, follow the instructions in this short explainer video (https://www.youtube.com/watch?v=gMt_Rq43y_4&ab_channel=FahadKamran) to submit your homework.

If you cannot submit online, come to office hours for assistance. The office hours schedule appears on data8.org/fa16/weekly.html (http://data8.org/fa16/weekly.html).

This assignment is due Thursday, November 10 at 7PM. You will receive an early submission bonus point if you turn it in by Wednesday, November 3 at 7PM. Directly sharing answers is not okay, but discussing problems with course staff or with other students is encouraged.

Important note: Only Parts 1 and 2 of this assignment will be graded. Parts 3 and 4 of this assignment will not be graded and are intended to give you extra practice.

Reading:

- Textbook chapter 12 (https://www.inferentialthinking.com/chapters/12/why-the-mean-matters.html) (for review)
- Textbook chapter 13 (https://www.inferentialthinking.com/chapters/13/prediction.html)

Run the cell below to prepare the notebook.

In [ ]:
```
# Run this cell to set up the notebook, but please don't change it.
import numpy as np
from datascience import *

# These lines do some fancy plotting magic.
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import warnings
warnings.simplefilter('ignore', FutureWarning)
from matplotlib import patches
from ipywidgets import interact, interactive, fixed
import ipywidgets as widgets

from client.api.assignment import load_assignment
tests = load_assignment('hw09.ok')
```

Evaluating NBA Game Predictions

A brief introduction to sports betting

In a basketball game, each team scores some number of points. Conventionally, the team playing at its own arena is called the "home team," and the other team is called the "away team." The winner, of course, is the team with the most points. So we could summarize what happened in a game by the following number:

$$\text{outcome} = \text{points scored by the away team} - \text{points scored by the home team}$$

If this number is positive, the away team won. If it's negative, the home team won. For brevity, we'll use the shorthand **"outcome"** for **the away team's score minus the home team's score**.

Casinos in Las Vegas offer bets on the outcomes of NBA games. One kind of bet works like this:

1. The casino decides on a "spread."
2. You can bet $11 that the outcome will be above the spread, or $11 that the outcome will be below the spread.
3. After the game, you end up with $21 if you guessed correctly, and $0 if you guessed incorrectly.

The analysts at the casino try to choose the spread so that (according to their analysis of the teams) there is a 50% chance that the outcome will be below that amount, and a 50% chance that the outcome will be above that amount.

tl;dr (https://en.wikipedia.org/wiki/Wikipedia:Too_long;_didn%27t_read): The spread is the casino's best guess at the outcome (the away team's score minus the home team's score).

The table `spreads` contains spreads from the betting website Covers (http://www.covers.com) from every game in the 2014 NBA season, plus actual game outcomes.

```
In [3]: spreads = Table.read_table("spreads.csv")
        spreads
```

Question 1

Make a scatter plot of the outcomes and spreads, with the spreads on the horizontal axis.

```
In [1]: ...
```

You might notice that spreads and outcomes are (almost) never 0. It's because a game of basketball never ends in a tie; one team has to win.

Let's investigate how well the casinos are predicting game outcomes.

One question we can ask is: Is the casino's prediction correct on average? In other words, for every value of the spread, is the average outcome of games assigned that spread equal to the spread? If not, the casino would apparently be making a systematic error in its predictions.

Question 2

Among games with a spread around 5 (concretely: in the range $[3.5, 6.5]$), what was the average outcome?

```
In [10]: spreads_around_5 = ...
         spread_5_outcome_average = ...
         print("Average outcome:", spread_5_outcome_average)
```

Instead of doing that for each possible spread, we can use linear regression to predict an outcome for any spread.

Question 3

If the average outcome for games with each spread is roughly equal to that spread, what would you expect the slope and intercept of the linear regression line to be? Or is it impossible to say? If it's impossible, use the cell below to write a comment explaining why. Otherwise, write the expected slope and intercept.

```
In [ ]: expected_slope = ...
        expected_intercept = ...
```

Let's compute the regression line and find out if that's the case. This takes a few steps.

Question 4

Define a function called `standard_units`. It should take an array of numbers as its argument and return an array of those numbers in standard units.

```
In [23]: def standard_units(nums):
             """Converts an array of numbers to standard units."""
             ...
```

Question 5

Compute the correlation between outcomes and spreads.

In [24]:
```
spread_r = ...
spread_r
```

Question 6

Compute the slope and intercept of the least-squares linear regression line that predicts outcomes from spreads.

In [25]:
```
spread_slope = ...
spread_intercept = ...
print("predicted outcome = {:f}*spread + {:f}".format(spread_slope, spre
ad_intercept))
```

Question 7

For each game in `spreads`, compute the predicted outcome using your regression line. Add these to `spreads` as a column called `"Predicted outcome"`, naming the resulting table `with_predictions`.

In [30]:
```
with_predictions = ...
with_predictions
```

Here's a plot of the predictions.

In [31]:
```
with_predictions.scatter("Spread", make_array("Outcome", "Predicted outc
ome"))
```

Question 8

Is it true that the average outcome for games with each spread is around that spread?

Write your answer here, replacing this text.

Question 9

Do you think the casino predicted game outcomes *accurately*? What number would you use to quantify that?

Write your answer here, replacing this text.

Finding the Least Squares Regression Line

In this exercise, you'll work with a small, abstract dataset of 5 x/y pairs. You'll see 3 ways to find the least-squares regression line.

Run the next cell to generate the dataset `d` and see a scatter plot.

```
In [2]: d = Table().with_columns(
            'x', make_array(0,  1,  2,  3,  4),
            'y', make_array(1, .5, -1,  2, -3))
        d.scatter('x')
```

When you run it, the next cell generates sliders that control the slope and intercept of a potential line. When you adjust a slider, the line will move.

Question 1

By moving the line around, make your best guess at the least-squares regression line. (It's okay if your line isn't exactly right, as long as it's reasonable.)

Note: Python will probably take about a second to redraw the plot each time you adjust the slider. We suggest clicking the place on the slider you want to try and waiting for the plot to be drawn; dragging the slider handle around will cause a long lag.

In [3]:
```
def plot_line(slope, intercept):
    plt.figure(figsize=(5,5))

    endpoints = make_array(-2, 7)
    p = plt.plot(endpoints, slope*endpoints + intercept, color='orange',
 label='Proposed line')

    plt.scatter(d.column('x'), d.column('y'), color='blue', label='Point
s')

    plt.xlim(-4, 8)
    plt.ylim(-6, 6)
    plt.gca().set_aspect('equal', adjustable='box')

    plt.legend(bbox_to_anchor=(1.8, .8))

interact(plot_line, slope=widgets.FloatSlider(min=-4, max=4, step=.1), i
ntercept=widgets.FloatSlider(min=-4, max=4, step=.1));
```

You can probably find a reasonable-looking line by just eyeballing it. But remember: the least-squares regression line minimizes the mean of the squared errors made by the line for each point. Your eye might not be able to judge squared errors very well.

A note on mean and total squared error

Before we move on, a note is in order.

It is common to think of the least-squares line as the line with the least *mean* squared error (or the square root of the mean squared error), as the textbook does.

But it turns out that it doesn't matter whether you minimize the mean squared error or the *total* squared error. You'll get the same best line in either case.

That's because the total squared error is just the mean squared error multipled by the number of points (`d.num_rows`). So if one line gets a better total squared error than another line, then it also gets a better mean squared error. In particular, the line with the smallest total squared error is also better than every other line in terms of mean squared error. That makes it the least squares line.

Question 2

The next cell produces a more useful plot. Use it to find a line that's closer to the least-squares regression line, keeping the above note in mind.

In [4]:
```
def plot_line_and_errors(slope, intercept):
    plt.figure(figsize=(5,5))
    points = make_array(-2, 7)
    p = plt.plot(points, slope*points + intercept, color='orange', 
label='Proposed line')
    ax = p[0].axes

    predicted_ys = slope*d.column('x') + intercept
    diffs = predicted_ys - d.column('y')
    for i in np.arange(d.num_rows):
        x = d.column('x').item(i)
        y = d.column('y').item(i)
        diff = diffs.item(i)

        if diff > 0:
            bottom_left_x = x
            bottom_left_y = y
        else:
            bottom_left_x = x + diff
            bottom_left_y = y + diff

        ax.add_patch(patches.Rectangle(make_array(bottom_left_x, bottom_
left_y), abs(diff), abs(diff), color='red', alpha=.3, label=('Squared er
ror' if i == 0 else None)))
        plt.plot(make_array(x, x), make_array(y, y + diff), color='red',
 alpha=.6, label=('Error' if i == 0 else None))

    plt.scatter(d.column('x'), d.column('y'), color='blue', label='Point
s')

    plt.xlim(-4, 8)
    plt.ylim(-6, 6)
    plt.gca().set_aspect('equal', adjustable='box')

    plt.legend(bbox_to_anchor=(1.8, .8))

interact(plot_line_and_errors, slope=widgets.FloatSlider(min=-4, max=4, 
step=.1), intercept=widgets.FloatSlider(min=-4, max=4, step=.1));
```

Question 3

Describe the visual criterion you used to find a line in question 2. (For example, a possible (but incorrect) answer is, "I tried to make the red line for the bottom-right point as small as possible.")

Write your answer here, replacing this text.

Question 4

Does the point at (3, 2) have more or less influence than any other point on the location of the line?

Write your answer here, replacing this text.

Now, let's have Python find this line for us. When we use `minimize`, Python goes through a process similar to the one you might have used in question 2.

But Python can't look at a plot that displays errors! Instead, we tell it how to find the total squared error for a line with a given slope and intercept.

Question 5

Define a function called `total_squared_error`. It should take two numbers as arguments:

1. the slope of some potential line
2. the intercept of some potential line

It should return the total squared error when we use that line to make predictions for the dataset `d`.

In [5]:
```
def total_squared_error(slope, intercept):
    # Hint: The staff answer computed an array called predictions
    # and an array called errors first.
    predictions = ...
    errors = ...
    ...
```

In [6]:
```
_ = tests.grade('q2_5')
```

Question 6

What is the total squared error for the line you found by "eyeballing" the errors in the first question? What about the question after that, where you had a better visual aid? (It's okay if the error went up!)

In [8]:
```
eyeballed_error = ...
aided_error = ...
print("Eyeballed error:", eyeballed_error, "\nAided error:",
aided_error)
```

Question 7

Use `minimize` to find the actual slope and intercept of the least-squares regression line.

Note: `minimize` will return a single array containing the slope as the first element and intercept as the second.

```
In [9]: # The staff solution used 1 line of code above here.
        slope_from_minimize = ...
        intercept_from_minimize = ...
        print("Least-squares regression line: predicted_y = {:f}*x + {:f}".forma
        t(slope_from_minimize, intercept_from_minimize))
```

Question 8

What was the total squared error for that line?

```
In [13]: best_total_squared_error = ...
         best_total_squared_error
```

Run the following cell to plot this line and its errors:

```
In [14]: plot_line_and_errors(slope_from_minimize, intercept_from_minimize)
```

Question 9

Compute the correlation between the `"x"` and `"y"` columns, then use the formula in 13.2 (https://www.inferentialthinking.com/chapters/13/2/regression-line.html) to find the slope and intercept of the least-squares regression line.

```
In [15]: # The staff solution used 4 lines of code before this.
         d_r = ...
         slope_from_r = ...
         intercept_from_r = ...
         print("Regression line computed from the correlation: predicted_y =
         {:f}*x + {:f}".format(slope_from_r, intercept_from_r))
```

Compare this with your answer to question 7 to verify that they're both correct. They will be a little bit different, because `minimize` is by default accurate only to within 0.01. (If they're not roughly the same, try computing the total squared error for both; the one with the smaller error is more likely correct!)

Triple Jump Distances vs. Vertical Jump Heights

Does skill in one sport imply skill in a related sport? The answer might be different for different activities. Let us find out whether it's true for the triple jump (https://en.wikipedia.org/wiki/Triple_jump) (an horizontal jump similar to a long jump) and the vertical jump. Since we're learning about linear regression, we will look specifically for a *linear* association between skill in the two sports.

The following data was collected by observing 40 collegiate level soccer players. Each athlete's distance in both jump activities was measured (in centimeters). Run the cell below to load the data.

```
In [2]: # Run this cell to load the data
        jumps = Table.read_table('triple_vertical.csv')
        jumps
```

Question 1

Before running a regression, it's important to see what the data look like, because our eyes are good at picking out unusual patterns in data. Draw a scatter plot with the triple jump distances on the horizontal axis and the vertical jump heights on vertical axis.

```
In [3]: ...
```

Question 2 Does the correlation coefficient `r` look closer to 0, .5, or -.5? Explain.

Write your answer here, replacing this text.

Question 3

Create a function called `regression_parameters`. It takes as its argument a table with two columns. The first column is the x-axis, and the second column is the y-axis. It should compute the correlation between the two columns, then compute the slope and intercept of the regression line that predicts the second column from the first. It should return an array with three elements: the correlation coefficient of the two columns, the slope of the regression line, and the intercept of the regression line.

Hint: You did a similar thing in lab, though it was spread out across several questions.

```
In [11]: def regression_parameters(tbl):
             ...
             # Our solution had 4 lines above this one
             r = ...
             slope = ...
             intercept = ...
             return make_array(r,slope,intercept)

         # When your function is finished, the next lines should
         # perform a regression predicting vertical jump distances
         # from triple jump distances.
         parameters = regression_parameters(jumps)
         jumps_r = parameters.item(0)
         jumps_slope = parameters.item(1)
         jumps_intercept = parameters.item(2)
```

Question 4

Let's use `regression_parameters` to predict what certain athletes' vertical jump heights would be given their triple jump distances.

The world record for the triple jump distance is 18.29 *meters* by Johnathan Edwards. What's our prediction for what Edwards' vertical jump would be?

```
In [15]: triple_record_vert_est = ...
         print("Predicted vertical jump distance: {:f} centimeters".format(triple
         _record_vert_est))
```

Question 5

Do you trust this estimate? Why or why not?

Write your answer here, replacing this text.

The Bootstrap and The Normal Curve

In this exercise, we will explore a dataset that includes the safety inspection scores for restauraunts in the city of Austin, Texas. We will be interested in determining the average restaurant score (out of 100) for the city from a random sample of the scores. We'll compare two methods for computing a confidence interval for that quantity: the bootstrap resampling method, and an approximation based on the Central Limit Theorem.

```
In [3]: # Just run this cell.
        pop_restaurants = Table.read_table('restaurant_inspection_scores.csv').d
        rop(5,6)
        pop_restaurants
```

Question 1

Plot a histogram of the scores.

```
In [4]: # Write your code here.
        ...
```

This is the population mean:

```
In [5]: pop_mean = np.mean(pop_restaurants.column(3))
        pop_mean
```

Often it is impossible to find complete datasets like this. Imagine we instead had access only to a random sample of 100 restaurants, called `restaurant_sample`. That table is created below. We are interested in using this sample to estimate the population mean.

```
In [4]: restaurant_sample = pop_restaurants.sample(100, with_replacement=False)
        restaurant_sample
```

Question 2

Plot the histogram of the **sample** scores.

```
In [5]: # Write your code here:
        ...
```

This is the **sample mean**:

```
In [6]: sample_mean = np.mean(restaurant_sample.column(3))
        sample_mean
```

Question 3

Complete the function `bootstrap_scores` below. It should take no arguments. It should simulate drawing 5000 resamples from `restaurant_sample` and computing the mean restaurant score in each resample. It should return an array of those 5000 resample means.

In [7]:
```
def bootstrap_scores():
    resampled_means = ...
    for i in range(5000):
        resampled_mean = ...
        resampled_means = ...
    ...

resampled_means = bootstrap_scores()
resampled_means
```

Question 4

Make a histogram of the **resampled means**. What sort of a distribution do they look like they follow?

In [11]:
```
# Write your code here:
...
```

Question 5

Compute a 95 percent confidence interval for the average restaurant score.

In [12]:
```
lower_bound = ...
upper_bound = ...
print("95% confidence interval for the average restaurant score, compute
d by bootstrapping:\n(",lower_bound, ",", upper_bound, ")")
```

Question 6

Throughout this section, we've looked at histograms of three distributions: the population, the sample, and the means of resamples taken from the sample. Do the population and sample distributions look similar to each other? What about the population and the means of resamples? Which, if any, look normally distributed?

Write your answer here, replacing this text.

For the last question, you'll need to recall two facts.

1. If a group of numbers has a normal distribution, around 95% of them lie within 2 standard deviations of their mean.
2. The Central Limit Theorem tells us the quantitative relationship between
    - the standard deviation of an array of numbers and
    - the standard deviation of an array of means of samples taken from those numbers.

Question 7

Without referencing the array `resampled_means` or performing any new simulations, calculate an interval that covers approximately 95% of the numbers in that array. **You may access `restaurant_sample`.**

```
In [14]: # The staff solution included a few lines of code before this one.
         lower_bound_normal = ...
         upper_bound_normal = ...
         print("95% confidence interval for the average restaurant score, compute
         d by a normal approximation:\n(",lower_bound_normal, ",", upper_bound_no
         rmal, ")")
```

This confidence interval should look very similar to the one you computed in question 5. If not, try calculating the inner 95 percent using 1.96 standard deviations instead of 2, for a more precise calculation. If they are still very different, there may be an error in your code.