hw12_student_solutions

December 11, 2019

1 Homework 12: Linear Regression

Reading: * Linear Regression * Least Squares Regression

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to load the provided tests. Each time you start your server, you will need to execute this cell again to load the tests.

Homework 12 is due **Thursday, 12/05 at 11:59pm**. You will receive an early submission bonus point if you turn in your final submission by Wednesday, 12/04 at 11:59pm. Start early so that you can come to office hours if you're stuck. Check the website for the office hours schedule. Late work will not be accepted as per the policies of this course.

Directly sharing answers is not okay, but discussing problems with the course staff or with other students is encouraged. Refer to the policies page to learn more about how to learn cooperatively.

For all problems that you must write our explanations and sentences for, you **must** provide your answer in the designated space. Moreover, throughout this homework and all future ones, please be sure to not re-assign variables throughout the notebook! For example, if you use `max_temperature` in your answer to one question, do not reassign it later on.

[ ]:
```python
# Don't change this cell; just run 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 client.api.notebook import Notebook
ok = Notebook('hw12.ok')
```

1.1 1. 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 (a 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 level in the two sports.

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

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

```
[9]: triple | vertical
383    | 33
781    | 71.1
561.62 | 62.25
624.52 | 61.33
446.24 | 40.19
515.3  | 38.96
449.22 | 39.69
560.91 | 46.51
519.12 | 37.68
595.38 | 53.48
… (30 rows omitted)
```

Question 1

Create the function `standard_units` so that it converts the values in the array `data` to standard units.

```
BEGIN QUESTION
name: q1_1
manual: false
```

```
[10]: def standard_units(data):
    return (data - np.mean(data)) / np.std(data) #SOLUTION
```

Question 2

Now, using the `standard units` function, define the function `correlation` which computes the correlation between `x` and `y`.

```
BEGIN QUESTION
name: q1_2
manual: false
```

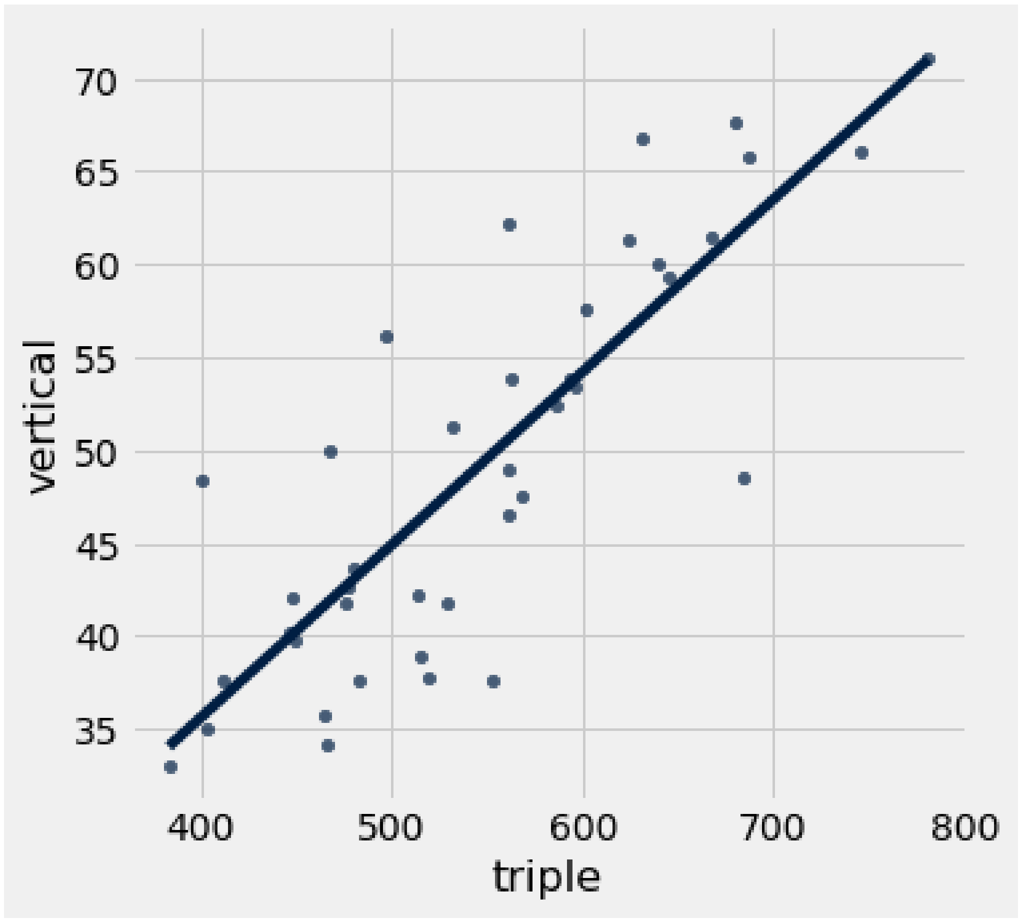
```
[13]: def correlation(x, y):
    return np.mean(standard_units(x) * standard_units(y)) #SOLUTION
```

Question 3 Before running a regression, it's important to see what the data looks like, because our eyes are good at picking out unusual patterns in data. Draw a scatter plot, **that includes the regression line**, with the triple jump distances on the horizontal axis and the vertical jump heights on vertical axis.

See the documentation on `scatter` here for instructions on how to have Python draw the regression line automatically.

```
BEGIN QUESTION
name: q1_3
manual: true
image: true
```

```
[16]: jumps.scatter('triple', 'vertical', fit_line=True) #SOLUTION
```



Question 4 Does the correlation coefficient `r` look closest to 0, .5, or -.5? Explain.

```
BEGIN QUESTION
name: q1_4
manual: true
```

SOLUTION: It definitely looks closest to .5. The two variables are positively associated, so it's not -.5. The data roughly follow a line, so the correlation is probably closer to .5 than to 0.

Question 5 Create a function called `parameter_estimates` that takes in the argument `tbl`, a two-column table where the first column is the x-axis and the second column is the y-axis. It should return an array with three elements: the **(1) correlation coefficient** of the two columns and the **(2) slope** and **(3) intercept** of the regression line that predicts the second column from the first, in original units (centimeters).

Hint: This is a rare occasion where it's better to implement the function using column indices instead of column names, in order to be able to call this function on any table. If you need a reminder about how to use column indices to pull out individual columns, you can refer to the textbook.

```
BEGIN QUESTION
name: q1_5
manual: false
```

```
[29]: def parameter_estimates(tbl):
          # BEGIN SOLUTION
          y_mean = np.mean(tbl.column(1))
          y_sd = np.std(tbl.column(1))
          x_mean = np.mean(tbl.column(0))
          x_sd = np.std(tbl.column(0))
          r = np.mean((tbl.column(1) - y_mean)/y_sd * (tbl.column(0) - x_mean)/x_sd)
          slope = r * (y_sd / x_sd)
          intercept = slope * (-x_mean) + y_mean
          # END SOLUTION
          return make_array(r, slope, intercept)

      parameters = parameter_estimates(jumps) #SOLUTION
      print('r:', parameters.item(0), '; slope:', parameters.item(1), '; intercept:',␣
       ↪parameters.item(2))
```

```
r: 0.8343076972837598 ; slope: 0.09295728160512184 ; intercept:
-1.566520972963474
```

Question 6

Now suppose you want to go the other way: to predict a triple jump distance given a vertical jump distance. What would the regression parameters of this linear model be? How do they compare to the regression parameters from the model where you were predicting vertical jump distance given a triple jump distance (in question 5)?

Set `regression_changes` to an array of 3 elements, with each element corresponding to whether or not the corresponding item returned by `parameter_estimates` changes when switching vertical and triple as x and y. For example, if r changes, the slope changes, but the intercept wouldn't change, the array would be `make_array(True, True, False)`.

```
BEGIN QUESTION
```

```
name: q1_6
manual: false
```

```
[32]: regression_changes = make_array(False, True, True) #SOLUTION
      regression_changes
```

```
[32]: array([0, 1, 1], dtype=int64)
```

Question 7 Let's use `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 is the prediction for Edward's vertical jump using this line?

Hint: Make sure to convert from meters to centimeters!

```
BEGIN QUESTION
name: q1_7
manual: false
```

```
[24]: triple_record_vert_est = parameters.item(1) * 1829 + parameters.item(2)␣
       ↪#SOLUTION
      print("Predicted vertical jump distance: {:f} centimeters".
       ↪format(triple_record_vert_est))
```

```
Predicted vertical jump distance: 168.452347 centimeters
```

Question 8 Do you think it makes sense to use this line to predict Edward's vertical jump?

Hint: Compare Edwards' triple jump distance to the triple jump distances in `jumps`. Is it relatively similar to the rest of the data?

```
BEGIN QUESTION
name: q1_8
manual: true
```

SOLUTION: No; we have absolutely no information on the triple jump distances in any remote region near 18.29 meters, so it's not smart to make an estimate for it based on this data that is outside our observed range. In fact, this is around 7 cm higher than the current world record according to Guinness. That's not totally implausible, but it seems unlikely.

1.2 2. Cryptocurrencies

Imagine you're an investor in December 2017. Cryptocurrencies, online currencies backed by secure software, are becoming extremely valuable, and you want in on the action!

The two most valuable cryptocurrencies are Bitcoin (BTC) and Ethereum (ETH). Each one has a dollar price attached to it at any given moment in time. For example, on December 1st, 2017, one BTC costs $$$10859.56 and one ETH costs $$$424.64.

You want to predict the price of ETH at some point in time based on the price of BTC. Below, we load two tables called `btc` and `eth`. Each has 5 columns: * `date`, the date * `open`, the value of the currency at the beginning of the day * `close`, the value of the currency at the end of the day * `market`, the market cap or total dollar value invested in the currency * `day`, the number of days since the start of our data

[3]:
```
btc = Table.read_table('btc.csv')
btc
```

[3]:
```
date       | open   | close  | market     | day
2015-09-29 | 239.02 | 236.69 | 3505090000 | 1
2015-09-30 | 236.64 | 236.06 | 3471280000 | 2
2015-10-01 | 236    | 237.55 | 3462800000 | 3
2015-10-02 | 237.26 | 237.29 | 3482190000 | 4
2015-10-03 | 237.2  | 238.73 | 3482100000 | 5
2015-10-04 | 238.53 | 238.26 | 3502460000 | 6
2015-10-05 | 238.15 | 240.38 | 3497740000 | 7
2015-10-06 | 240.36 | 246.06 | 3531230000 | 8
2015-10-07 | 246.17 | 242.97 | 3617400000 | 9
2015-10-08 | 243.07 | 242.3  | 3572730000 | 10
… (825 rows omitted)
```

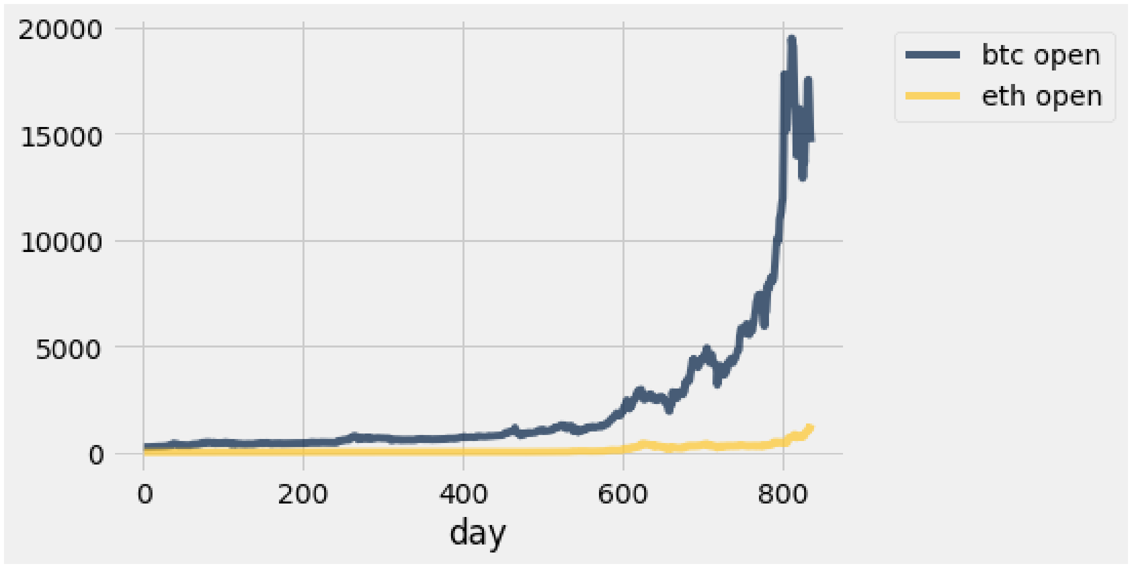
[4]:
```
eth = Table.read_table('eth.csv')
eth
```

[4]:
```
date       | open     | close    | market   | day
2015-09-29 | 0.579414 | 0.661146 | 42607700 | 1
2015-09-30 | 0.661192 | 0.738644 | 48636600 | 2
2015-10-01 | 0.734307 | 0.690215 | 54032300 | 3
2015-10-02 | 0.683732 | 0.678574 | 50328700 | 4
2015-10-03 | 0.678783 | 0.687171 | 49981900 | 5
2015-10-04 | 0.686343 | 0.668379 | 50556000 | 6
2015-10-05 | 0.666784 | 0.628643 | 49131600 | 7
2015-10-06 | 0.622218 | 0.650645 | 45863300 | 8
2015-10-07 | 0.650515 | 0.609388 | 47964700 | 9
2015-10-08 | 0.609501 | 0.621716 | 44955900 | 10
… (825 rows omitted)
```

Question 1 In the cell below, create a line plot that visualizes the BTC and ETH open prices as a function of time. Both btc and eth open prices should be plotted on the same graph.

```
BEGIN QUESTION
name: q2_1
manual: true
image: true
```

```
[5]: # BEGIN SOLUTION NO PROMPT
     btc_only = btc.select('day', 'open').relabeled('open', 'btc open')
     both = btc_only.with_column('eth open', eth.column('open'))
     both.plot('day')
     # END SOLUTION
     """ # BEGIN PROMPT
     # Create a line plot of btc and eth open prices as a function of time
     ...
     """; # END PROMPT
```



Question 2 Now, calculate the correlation coefficient between the opening prices of BTC and ETH using the `correlation` function you defined earlier.

```
BEGIN QUESTION
name: q2_2
manual: false
```

```
[6]: r = correlation(both.column('eth open'), both.column('btc open')) #SOLUTION
     r
```

```
[6]: 0.9250325764148278
```

Question 3 Regardless of your conclusions above, write a function `eth_predictor` which takes an opening BTC price and predicts the opening price of ETH. Again, it will be helpful to use the function `parameter_estimates` that you defined earlier in this homework.

Note: Make sure that your `eth_predictor` is using least squares linear regression.

```
BEGIN QUESTION
```

```
name: q2_3
manual: false
```

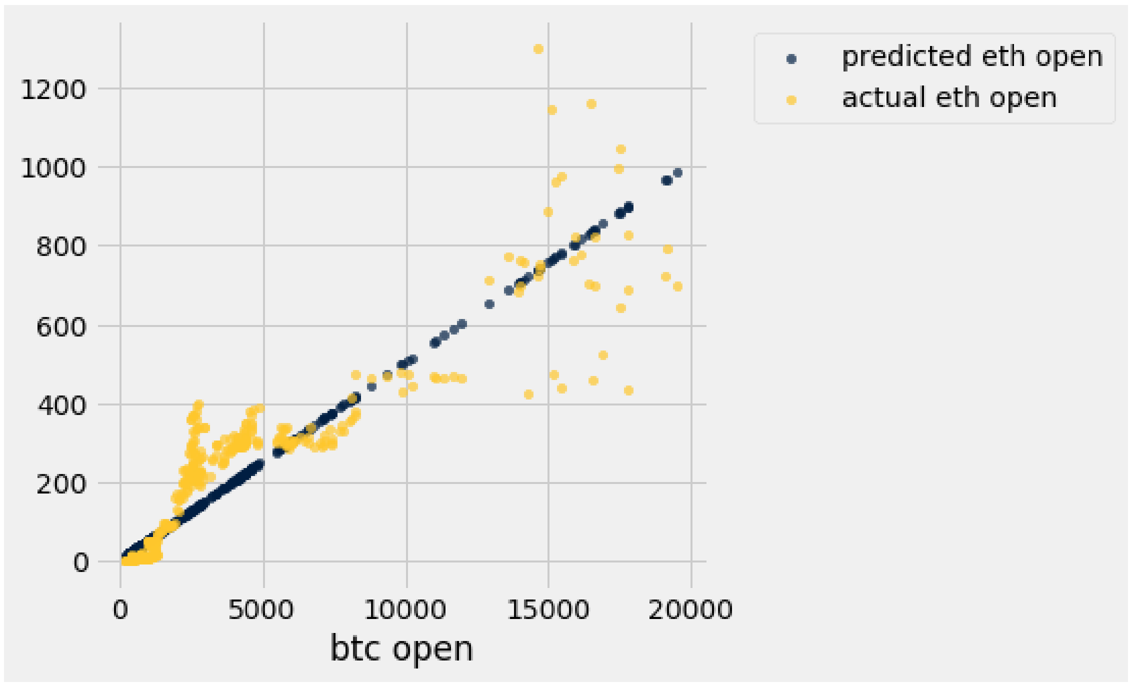
```
[10]: def eth_predictor(btc_price):
          parameters = parameter_estimates(Table().with_columns("btc", btc.
       ↪column("open"), "eth", eth.column("open"))) #SOLUTION
          slope = parameters.item(1) #SOLUTION
          intercept = parameters.item(2) #SOLUTION
          return slope * btc_price + intercept #SOLUTION
```

Question 4 Now, using the `eth_predictor` you defined in the previous question, make a scatter plot with BTC prices along the x-axis and both real and predicted ETH prices along the y-axis. The color of the dots for the real ETH prices will be different from the color for the predicted ETH prices.

Hints: * An example of such a scatter plot is generated here. * Think about the table that must be produced and used to generate this scatter plot. What data should the columns represent? Based on the data that you need, how many columns should be present in this table? Also, what should each row represent? Constructing the table will be the main part of this question; once you have this table, generating the scatter plot should be straightforward as usual.

```
BEGIN QUESTION
name: q2_4
manual: true
image: true
```

```
[13]: btc_open = btc.select('open') # SOLUTION
      eth_pred = btc_open.with_column("predicted eth open", btc.apply(eth_predictor,␣
       ↪"open")) # SOLUTION
      eth_pred_actual = eth_pred.with_column("actual eth open", eth.column("open")) #␣
       ↪SOLUTION
      eth_pred_actual.relabeled("open", "btc open").scatter('btc open') # SOLUTION
```



Question 5 Considering the shape of the scatter plot of the true data, is the model we used reasonable? If so, what features or characteristics make this model reasonable? If not, what features or characteristics make it unreasonable?

```
BEGIN QUESTION
name: q2_5
manual: true
```

SOLUTION: This is not a great model for this particular data, as the true data are not even close to being linear.

1.3 3. 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 is the team with more points.

We can summarize what happened in a game by the "**outcome**", defined as the **the away team's score minus the home team's score**:

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

If this number is positive, the away team won. If it's negative, the home team won.

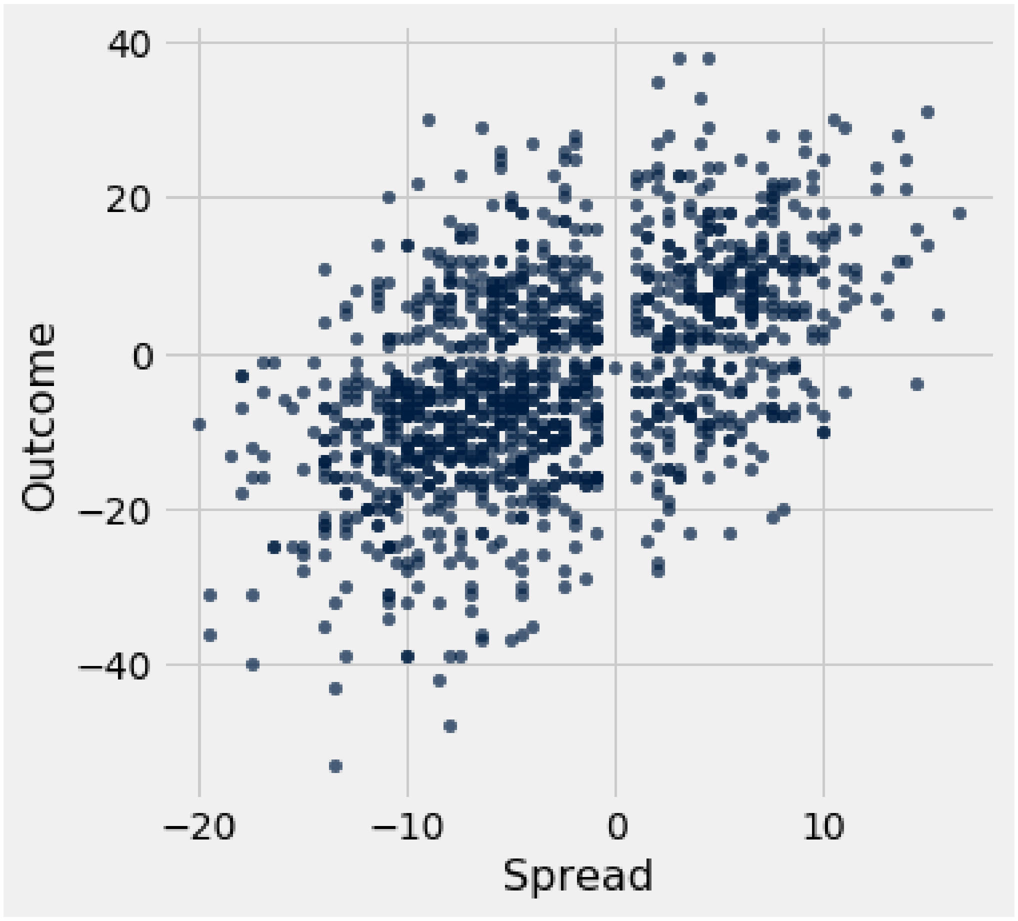
In order to facilitate betting on games, analysts at casinos try to predict the outcome of the game. This prediction of the outcome is called the **spread.**

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

```
[2]: Date       | Home Team     | Away Team   | Home Points | Away Points | Outcome |
     Spread
     4/10/2015  | Utah          | Memphis     | 88          | 89          | 1       |
     2.5
     3/10/2015  | Utah          | New York    | 87          | 82          | -5      |
     -13
     11/19/2014 | Indiana       | Charlotte   | 88          | 86          | -2      |
     -2
     11/15/2014 | Chicago       | Indiana     | 90          | 99          | 9       |
     -9
     3/25/2015  | Utah          | Portland    | 89          | 92          | 3       |
     -2
     3/3/2015   | Memphis       | Utah        | 82          | 93          | 11      |
     -7
     3/18/2015  | Utah          | Washington  | 84          | 88          | 4       |
     -3
     3/16/2015  | Utah          | Charlotte   | 94          | 66          | -28     |
     -4.5
     1/24/2015  | Charlotte     | New York    | 76          | 71          | -5      |
     -9
     11/7/2014  | Oklahoma City | Memphis     | 89          | 91          | 2       |
     7
     … (1220 rows omitted)
```

Here's a scatter plot of the outcomes and spreads, with the spreads on the horizontal axis.

```
[3]: spreads.scatter("Spread", "Outcome")
```



From the scatter plot, you can see that the spread and outcome are almost never 0, aside from 1 case of the spread being 0. This is because a game of basketball never ends in a tie. One team has to win, so the outcome can never be 0. The spread is almost never 0 because it's chosen to estimate the outcome.

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 1 Compute the correlation coefficient between outcomes and spreads.

Note: It might be helpful to use the `correlation` function.

```
BEGIN QUESTION
name: q3_1
manual: false
```

```
[4]: spread_r = correlation(spreads.column("Outcome"), spreads.column("Spread"))␣
      ↪#SOLUTION
     spread_r
```

```
[4]: 0.49181413688314235
```

Question 2 Among games with a spread between 3.5 and 6.5 (including both 3.5 and 6.5), what was the average outcome?

Hint: Read the documentation for the predicate `are.between_or_equal_to` here.

```
BEGIN QUESTION
name: q3_2
manual: false
```

```
[7]: spreads_around_5 = spreads.where("Spread", are.between_or_equal_to(3.5, 6.5))␣
      ↪#SOLUTION
     spread_5_outcome_average = np.mean(spreads_around_5.column("Outcome")) #SOLUTION
     print("Average outcome for spreads around 5:", spread_5_outcome_average)
```

```
Average outcome for spreads around 5: 4.9941176470588236
```

Question 3 Compute the slope and intercept of the least-squares linear regression line that predicts outcomes from spreads, in original units.

```
BEGIN QUESTION
name: q3_3
manual: false
```

```
[10]: spread_slope = spread_r * np.std(spreads.column("Outcome")) / np.std(spreads.
       ↪column("Spread")) #SOLUTION
      spread_intercept = np.mean(spreads.column("Outcome")) - spread_slope * np.
       ↪mean(spreads.column("Spread")) # SOLUTION
      print("Slope:", round(spread_slope, 3))
      print("Intercept", round(spread_intercept, 3))
```

```
Slope: 0.954
Intercept 0.22
```

Question 4 Suppose that we create another predictor that simply predicts the average outcome regardless of the value for spread. Does this new predictor minimize least squared error?

```
BEGIN QUESTION
name: q3_4
manual: true
```

SOLUTION: The new predictor is a horizontal **line** that passes through the average value for outcome. Therefore. it does not minimize least squared error, as only the regression line is the unique straight line that minimizes least squared error among all straight lines.

1.3.1 Fitting a least-squares regression line

Recall that the least-square regression line is the unique straight line that minimizes root mean squared error (RMSE) among all possible fit lines. Using this property, we can find the equation of the regression line by finding the pair of slope and intercept values that minimize root mean squared error.

Question 5 Define a function called `errors`. It should take three arguments: 1. a table like `spreads` (with the same column names and meanings, but not necessarily the same data) 2. the slope of a line (a number) 3. the intercept of a line (a number).

It should return an array of the errors made when a line with that slope and intercept is used to predict outcome from spread for each game in the given table.

```
BEGIN QUESTION
name: q3_5
```

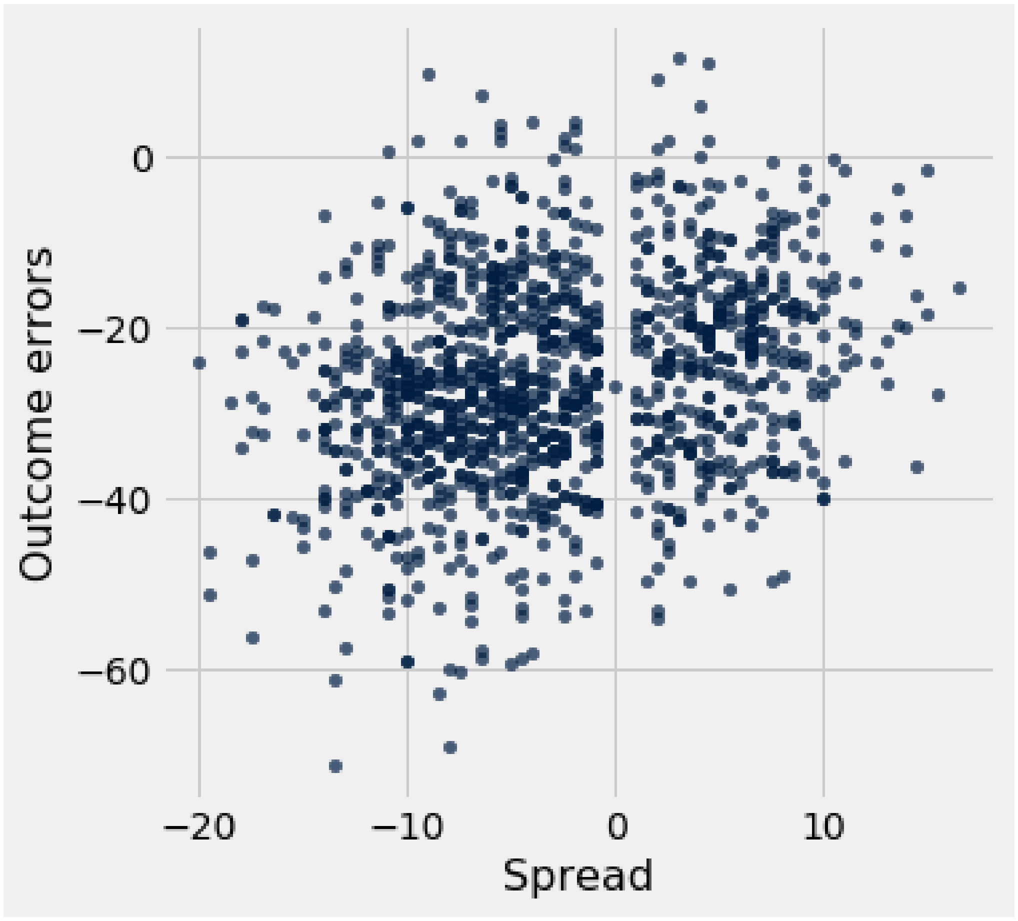
```
[15]: def errors(tbl, slope, intercept):
          predictions = slope * tbl.column("Spread") + intercept # SOLUTION NO PROMPT
          return tbl.column("Outcome") - predictions # SOLUTION
```

Question 6 Using `errors`, compute the errors for the line with slope `0.5` and intercept `25` on the `spreads` dataset. Name that array `spread_errors`. Then make a scatter plot of the errors.

Hint: To make a scatter plot of the errors, plot the error for each outcome in the dataset. Put the actual spread on the horizontal axis and the outcome error on the vertical axis.

```
BEGIN QUESTION
name: q3_6
```

```
[19]: spread_errors = errors(spreads, 0.5, 25) # SOLUTION
      # BEGIN SOLUTION
      spreads.with_column("Outcome errors", spread_errors)\
                  .scatter("Spread", "Outcome errors")
      # END SOLUTION
```



You should find that the errors are almost all negative. That means our line is not the regression line. Let's find a better one.

Question 7 Define a function called `fit_line`. It should take a table like `spreads` (with the same column names and meanings) as its argument. It should return an array containing the slope (as item 0) and intercept (as item 1) of the least-squares regression line predicting outcome from spread for that table.

Hint: Define a function `rmse` within `fit_line` that takes a slope and intercept as its arguments. `rmse` will use the table passed into `fit_line` to compute predicted outcomes and then return the mean squared error between the predicted and actual outcomes. Within `fit_line`, you can call `rmse` the way you would any other function.

If you haven't tried to use the `minimize` function yet, now is a great time to practice. Here's an example from the textbook.

```
BEGIN QUESTION
name: q3_7
```

[21]:
```
# BEGIN SOLUTION NO PROMPT
def fit_line(tbl):
    def rmse(slope, intercept):
        return (np.mean(errors(tbl, slope, intercept)**2)**0.5)
    return minimize(rmse)
# END SOLUTION
""" # BEGIN PROMPT
def fit_line(tbl):
    # Your code may need more than 1 line below here.
    def rmse(..., ...):
        return ...
    return ...
"""; # END PROMPT

# Here is an example call to your function.  To test your function,
# figure out the right slope and intercept by hand.
example_table = Table().with_columns(
    "Spread", make_array(0, 1),
    "Outcome", make_array(1, 3))
fit_line(example_table)
```

[21]:
```
array([2., 1.])
```

Question 8 Use `fit_line` to fit a line to `spreads`. Assign the output to `best_line`. Assign the first and second elements in `best_line` to `best_line_slope` and `best_line_intercept`, respectively.

Then, set `new_errors` equal to the errors that we get by calling `errors` with our new line. The following line will graph the corresponding residual plot with a best fit line.

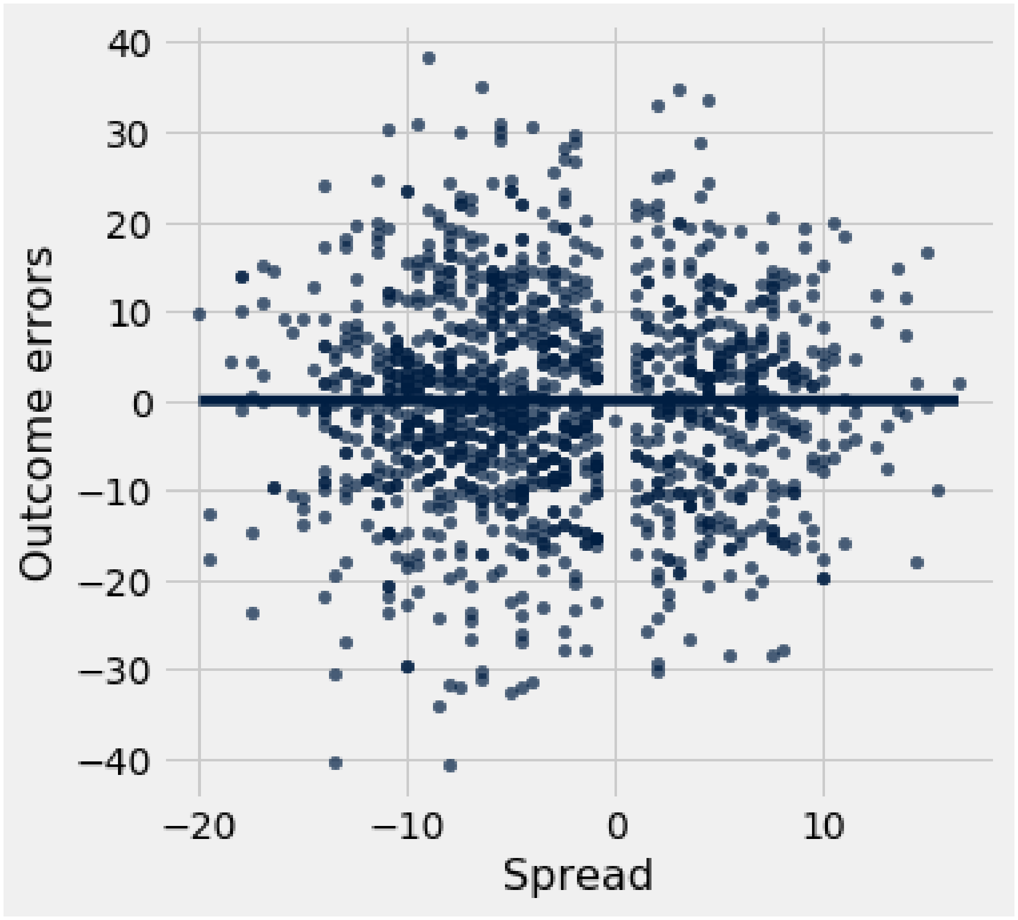
Make sure that the residual plot makes sense. (Hint: what qualities should the best fit line of a residual plot have?)

```
BEGIN QUESTION
name: q3_8
```

[25]:
```
best_line = fit_line(spreads) #SOLUTION
best_line_slope = best_line.item(0) #SOLUTION
best_line_intercept = best_line.item(1) #SOLUTION

new_errors = errors(spreads, best_line_slope, best_line_intercept) #SOLUTION

# This code displays the residual plot, given your values for the␣
 ↪best_line_slope and best_line_intercept
Table().with_columns("Spread",
                     spreads.column("Spread"),
                     "Outcome errors",
                     new_errors
```

```
                    ).scatter("Spread", "Outcome errors", fit_line=True)

# This just prints your slope and intercept
"Slope: {:g} | Intercept: {:g}".format(best_line_slope, best_line_intercept)
```

```
[25]: 'Slope: 0.953816 | Intercept: 0.217835'
```



Question 9 The slope and intercept pair you found in Question 8 should be very similar to the values that you found in Question 3. Why were we able to minimize RMSE to find the same slope and intercept from the previous formulas?

```
BEGIN QUESTION
name: q3_9
manual: true
```

SOLUTION: The regression line is the unique straight line (in other words, the unique slope/intercept pair) that minimizes RMSE. Therefore, we can also find the regression line by finding the slope and intercept values that minimize RMSE.

1.4 4. Final-Semester Survey

You can find the end of semester feedback form here. Please take some time to fill the survey out! Data 8 is still a relatively new class, and your feedback helps the class get better every semester!

As incentive, if 80% of the course fills out this feedback form **and** the official Berkeley Course Evaluations (which will be released sometime in the next couple of weeks) for Data 8, everyone will receive two points of extra credit!

Question 1. Fill out the end of semester feedback form linked above. Once you have submitted, a secret word will be displayed. Set `secret_word` to the secret string at the end of the form.

```
BEGIN QUESTION
name: q4_1
manual: false
```

[1]:
```
secret_word = "bayes" # SOLUTION
```

1.5 5. Submission

Once you're finished, select "Save and Checkpoint" in the File menu and then execute the `submit` cell below. The result will contain a link that you can use to check that your assignment has been submitted successfully. If you submit more than once before the deadline, we will only grade your final submission. If you mistakenly submit the wrong one, you can head to okpy.org and flag the correct version. To do so, go to the website, click on this assignment, and find the version you would like to have graded. There should be an option to flag that submission for grading!

[ ]:
```
_ = ok.submit()
```

[ ]:
```
# For your convenience, you can run this cell to run all the tests at once!
import os
print("Running all tests...")
_ = [ok.grade(q[:-3]) for q in os.listdir("tests") if q.startswith('q') and␣
 ↪len(q) <= 10]
print("Finished running all tests.")
```