Homework 10: Regression Inference, Diagnostics, and Optimization

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 17 at 7PM. You will receive an early submission bonus point if you turn it in by Wednesday, November 16 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. Part 3 of this assignment (on ice cream) will not be graded and is intended to give you extra practice with optimization.

Reading:

- Textbook chapter 13 (https://www.inferentialthinking.com/chapters/13/prediction.html)
- Textbook chapter 14 (https://www.inferentialthinking.com/chapters/14/inference-for-regression.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('hw10.ok')
```

Quantifying Sampling Errors in Regression

Recall the Old Faithful dataset from our lab on regression. The table contains two pieces of information about each eruption of the Old Faithful geyser in Yellowstone National Park:

1. The duration of the eruption, in minutes.
2. The time between this eruption and the next eruption (the "waiting time"), in minutes.

The dataset is plotted below along with its line of best fit.

In [6]:
```
faithful = Table.read_table('faithful_inference.csv')
faithful.scatter('duration', fit_line=True)
faithful
```

Last time we looked at this dataset, we noticed the apparent linear relationship betwen duration and wait, and we decided to use regression to predict wait in terms of duration. However, our data are just a sample of all the eruptions that have happened at Old Faithful. As we know, relationships can appear in a sample that don't really exist in the population from which the sample was taken.

Before we move forward using our linear model, we would like to know whether or not there truly exists a relationship between duration and wait time. If there is no relationship between the two, then we'd expect a correlation of 0, which would give us a slope of 0. In the language of hypothesis tests, we'd like to test the following hypotheses:

- **Null Hypothesis:** The true slope of the regression line that predicts wait from duration, computed using the population of all eruptions that have ever happened, is 0. If the slope of the regression line computed from our sample isn't 0, that's just the result of chance variation.
- **Alternate Hypothesis:** The true slope of the regression line is not 0.

We will use the method of confidence intervals to test this hypothesis.

Question 1

We'll warm up by implementing some familiar functions. You may use these functions throughout this assignment. Start by defining these two functions:

1. `standard_units` should take in an array of numbers and return an array containing those numbers converted to standard units.
2. `correlation` should take in a table with 2 columns and return the correlation between these columns.

In [7]:
```
def standard_units(arr):
    ...

def correlation(tbl):
    ...
```

Question 2

Using the functions you just implemented, create a function called `fit_line`. It should take a table as its argument. It should return an array containing the slope and intercept of the regression line that predicts the second column in the table using the first.

In [ ]:
```
def fit_line(tbl):
    ...
    slope = ...
    intercept = ...
    return make_array(slope, intercept)

fit_line(faithful)
```

Now we have all the tools we need in order to create a confidence interval quantifying our uncertainty about the true relationship between duration and wait time.

Question 3

Use the bootstrap to compute 1000 resamples from our dataset. For each resample, compute the slope of the best fit line. Put these slopes in the array `resample_slopes`, giving you the empirical distribution of regression line slopes in resamples.

In [9]:
```
resample_slopes = make_array()
for i in np.arange(1000):
    sample = ...
    resample_line = ...
    resample_slope = ...
    resample_slopes = ...

Table().with_column("Slope estimate", resample_slopes).hist()
```

Question 4

Use your resampled slopes to construct an approximate 95% confidence interval for the true value of the slope.

In [10]:
```
lower_end = ...
upper_end = ...
print("95% confidence interval for slope: [{:g},
{:g}]".format(lower_end, upper_end))
```

Question 5

Based on your confidence interval, would you accept or reject the null hypothesis that the true slope is 0? Why? What P-value cutoff are you using?

Write your answer here, replacing this text.

Suppose we're tourists at Yellowstone, and we'd like to know how long we'll have to wait for the next Old Faithful eruption. We decide to use our regression line to make some predictions for the waiting times. But just as we're uncertain about the slope of the true regression line, we're also uncertain about the predictions we'd make based on the true regression line.

Question 6

Define the function `fitted_value`. It should take 2 arguments:

1. A table with 2 columns. We'll be predicting the values in the second column using the first.
2. A number, the value of the predictor variable for which we'd like to make a prediction.

Make sure to use your `fit_line` function.

In [ ]:
```
def fitted_value(table, given_x):
    # The staff solution took 4 lines of code.
    ...

# Here's an example of how fitted_value is used.  This should
# compute the prediction for the wait time of an eruption that lasts
# two minutes .
two_minutes_wait = fitted_value(faithful, 2)
two_minutes_wait
```

Question 7

The park ranger tells us that the most recent eruption lasted 5 minutes. Using your function above, assign the variable `five_minutes_wait` to the predicted wait time.

In [33]:
```
five_minutes_wait = ...
five_minutes_wait
```

Juan, a fellow tourist, raises the following objection to your prediction:

> "Your prediction depends on your sample of 272 eruptions. Couldn't your prediction have been different if you had happened to have a different sample of eruptions?"

Having read section 14.3 (https://www.inferentialthinking.com/chapters/14/3/prediction-intervals.html) of the textbook, you know just the response!

Question 8

Define the function `bootstrap_lines`. It should take two arguments:

1. A table with two columns. As usual, we'll be predicting the second column using the first.
2. An integer, a number of bootstraps to run.

It should return a *table* whose first column, `"Slope"`, contains the given number of bootstrapped slopes, and whose second column, `"Intercept"`, contains the corresponding bootstrapped intercepts. Each slope and intercept should come from a regression line that predicts column 2 from column 1 of a resample of the given table. The table should have 1 row for each bootstrap replication.

Hint: Your code should look very similar to the code you wrote for question 3, with just a few key changes.

In [ ]:
```
def bootstrap_lines(tbl, num_bootstraps):
    ...

# When you're done, this code should produce the slopes
# and intercepts of 1000 regression lines computed from
# resamples of the faithful table.
regression_lines = bootstrap_lines(faithful, 1000)
regression_lines
```

Question 10

Create an array called `predictions_for_five`. It should contain 1000 numbers. Each number should be the predicted waiting time after an eruption with a duration of 5 minutes, using a different bootstrapped regression line.

In [16]:
```
predictions_for_five = ...

# This will make a histogram of your predictions:
table_of_predictions = Table().with_column('Predictions at eruptions=5',
 predictions_for_five)
table_of_predictions.hist('Predictions at eruptions=5', bins=20)
```

Question 11

Create a 95 percent confidence interval for these predictions.

In [14]:
```
lower_bound = ...
upper_bound = ...

print('95% Confidence interval for predictions for x=5: (',
lower_bound,",", upper_bound, ')')
```

Question 12

Look at the scatter plot of the data at the start of this exercise. Does your confidence interval cover around 95 percent of eruptions in `faithful` that had an eruption duration of 5 minutes? If not, what does this confidence interval mean?

Write your answer here, replacing this text.

2. Visual Diagnostics for Linear Regression

Linear regression isn't always the best way to describe the relationship between two variables. We'd like to develop techniques that will help us decide whether or not to use a linear model to predict one variable based on another.

We will use the insight that if a regression fits a set of points well, then the residuals from that regression line will show no pattern when plotted against the predictor variable.

The table below contains information about crime rates and median home values in suburbs of Boston. We will attempt to use linear regression to predict median home value in terms of crime rate.

About the dataset

All data are from 1970. Crime rates are per capita per year; home values are in thousands of dollars. The crime data come from the FBI, and home values are from the US Census Bureau. The original dataset can be found here (https://archive.ics.uci.edu/ml/datasets/Housing).

Run the next cell to load the data and see a scatter plot.

In [5]:
```
boston = Table.read_table('boston_housing.csv')
boston.scatter('Crime Rate')
```

Question 1

Write a function called `residuals`. It should take a single argument, a table. It should first compute the slope and intercept of the regression line that predicts the second column of that table (accessible as `tbl.column(1)`) using the first column (`tbl.column(0)`). `residuals` should return an array containing the *residuals* for that regression line. Recall that residuals are given by

$$\texttt{residual} = \texttt{observed value} - \texttt{regression estimate}.$$

In [ ]:
```
def residuals(tbl):
    ...
```

Question 2

Make a scatter plot of the residuals for the Boston housing dataset against crime rate. Crime rate should be on the horizontal axis.

In [10]:
```
...
```

Question 3

Does the plot of residuals look roughly like a formless cloud? Or is there some kind of pattern in them?

Write your answer here, replacing this text.

Question 4

Does it seem like a linear model is appropriate for describing the relationship between crime and median home value? Explain your reasoning.

Write your answer here, replacing this text.

Section 13.6 (https://www.inferentialthinking.com/chapters/13/6/numerical-diagnostics.html) of the textbook describes some mathematical facts that hold for all regression estimates, regardless of goodness of fit. One fact is that there is a relationship between the standard deviation of the residuals, the standard deviation of the response variable, and the correlation. Let us test this.

Below, we have imported a new table, the Old Faithful data.

In [22]:
```
old_faithful = Table.read_table('faithful.csv')
old_faithful
```

The following cell makes a residual plot for this new dataset.

In [23]:
```
Table().with_columns('Residual', residuals(old_faithful), 'Duration', fa
ithful.column('duration')).scatter('Duration')
```

Question 5

Directly compute the standard deviation of the residuals from the Boston data. Then compute the same quantity without using the residuals, using the formula described in section 13.6 instead.

In [18]:
```
boston_residual_sd = ...
boston_residual_sd_from_formula = ...

print("Residual SD: {0}".format(boston_residual_sd))
print("Residual SD from the formula: {0}".format(boston_residual_sd_from
_formula))
```

Question 6

Repeat the procedure from Question 5 for the `old_faithful` dataset.

In [24]:
```
faithful_residual_sd = ...
faithful_residual_sd_from_formula = ...

print("Residual SD: {0}".format(faithful_residual_sd))
print("Residual SD from the formula: {0}".format(faithful_residual_sd_fr
om_formula))
```

3. Better Ice Cream Sales through Minimization

In this exercise, we'll use `minimize` to find an optimal location for an ice cream truck. Minimization is useful in a vast array of applications - it's not just for finding the best line through a scatter plot!

You'll see 3 different ways to do minimization:

1. Using a slider to find the best location manually
2. Trying a bunch of locations using `apply` and finding the best one using `sort`
3. Using `minimize`

Data 8 is poised to disrupt the ice cream market. We're catering to San Francisco hipsters, so we operate a truck that sells our locally-sourced organic Sriracha-Kale ice cream. Today we have driven our truck to Ocean Beach, a long, narrow beach on the western coast of the city.

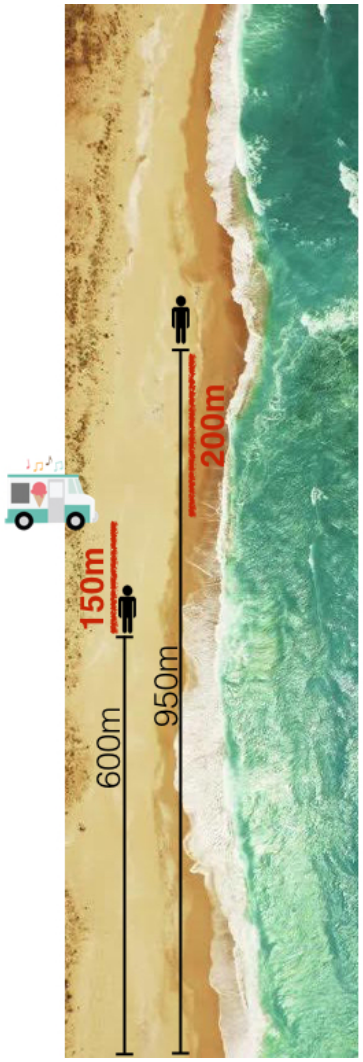
Upon arriving, we find that our potential customers are spread out along the beach. We decide we want to park our truck in the location that's closest *on average* to all the customers. That way, customers will be more likely to come to our truck.

(This may not be a great way to choose our truck's location. Maybe you can think of a better way to decide on a location.)

We canvas the beach and record the location of each beachgoer in a table called `customers`. The beach is oriented roughly North/South, and it's narrow, so we ignore how close each beachgoer is to the water. We record only how far north each person is from the southern end of the beach.

Suppose there are 2 people on the beach, at 600 meters and 950 meters from the Southern end, respectively. If we park our truck at 750 meters, the average distance from our truck to customers is:

$$\frac{|600 - 750| + |950 - 750|}{2}.$$



By now, the Python code that computes this might look a little familiar:

In [70]:
```
# The customer locations:
two_customer_locations = make_array(600, 950)

first_truck_location = 750

two_customers_mean_distance_from_750 = np.mean(np.abs(two_customer_locat
ions - first_truck_location))
two_customers_mean_distance_from_750
```

Question 1

A new person shows up on the beach, so the new customer locations are 600, 950, and 1,150 meters from the southern end. If we park our ice cream truck at the *mean* of those locations, what is the average distance from our truck to customers?

In [4]:
```
three_customer_locations = make_array(600, 950, 1150)

# Compute this.
three_customers_mean_distance_from_mean = ...
three_customers_mean_distance_from_mean
```

In [5]:
```
_ = tests.grade('q3_1')
```

Question 2

The mean is 900 meters. If we park our truck at 925 meters instead, what's the average distance from our truck to a customer?

In [73]:
```
# Fill in three_customers_mean_distance_from_925.  Use code to compute i
t.
three_customers_mean_distance_from_925 = ...
three_customers_mean_distance_from_925
```

In [7]:
```
_ = tests.grade('q3_2')
```

The average distance went down! Despite what your intuition might say, the mean of the customer locations isn't the best location to pick.

Use the slider created by the next cell to find approximately the best location for the `three_customer_locations` dataset. (You'll only be able to get within 5 of the best location. It's okay if your submission doesn't display the slider.)

In [16]:
```
def three_customers_distance(location):
    return np.mean(np.abs(three_customer_locations - location))

interact(three_customers_distance, location=widgets.FloatSlider(min=700,
 max=1300, step=5, value=900, msg_throttle=1));
```

Question 3

What location did you find? What was the average distance to customers from that location? Is that location around the same as any familiar statistic of the data?

Write your answer here, replacing this text.

The full dataset

Now let's look at the full customer dataset. In this dataset, there are 1,000 people on the beach. The next cell displays a histogram of their locations.

In [5]:
```
# Just run this cell.
customers = Table.read_table("customers.csv")
customers.hist(bins=np.arange(0, 2001, 100))
customers
```

Let's think very precisely about what we're trying to optimize. Given these customer locations, we want to find a *single location*. If we park our truck at that location, we want it to result in the smallest *average distance from our truck to customers*.

Question 4

Write a function called `average_distance`. It should take a single number as its argument (a truck location) and return the average distance from that location to the customers in the `customers` table.

In [8]:
```
def average_distance(location):
    # Fill in the function definition here.
    ...

# An example call to your function:
average_distance(1000)
```

In [12]:
```
_ = tests.grade('q3_4')
```

`average_distance` tells us how badly we're meeting our objective. A mathematician would call this an *objective function*. We want to find the distance that produces the smallest value of this objective function.

Use the slider created by the next cell to find approximately the best location for the `customers` dataset. (You'll only be able to get within 5 of the best location.)

In [9]:
```
interact(average_distance, location=widgets.FloatSlider(min=700, max=130
0, step=5, value=800, msg_throttle=1));
```

Question 5

What location did you find, and what was the average distance to customers from that location?

Write your answer here, replacing this text.

Question 6

Create a table called `average_distances` with two columns:

1. `"location"`, a truck location. The smallest location should be 700 and the largest should be 1300, and they should go up in increments of 1.
2. `"average distance to customers"`. The average distance from customers (in the `customers` table) to that location.

Then, sort the table to find the location with the smallest average distance to customers. Name the sorted table `sorted_average_distances`, and name the best location (a number) `best_location_by_sorting`.

Hint: The staff solution used the table method `apply`. If you don't, you'll need to use a `for` loop, and your code will be longer than the skeleton suggests.

In [86]:
```
locations = Table().with_column("location", np.arange(700, 1300+1, 1))

average_distances = locations.with_column("average distance to customer
s", ...)

sorted_average_distances = ...
sorted_average_distances.show(5)

best_location_by_sorting = ...
best_location_by_sorting
```

The function `minimize` does basically the same thing you just did.

It takes as its argument a *function*, the objective function. It returns the input (that is, the argument) that produces the smallest output value of the objective function. If the objective function takes several arguments, it returns the arguments that produce the smallest output value of the objective function, all together in one array.

Question 7

Use `minimize` to find the best location for our ice cream truck.

In [85]:
```
# Write code to compute the best location, using minimize.
best_location = ...
best_location
```

Your answer should match `best_location_by_sorting` up to a few decimal places.

Later in the day, the distribution of potential customers along the beach has changed. `customers2.csv` contains their new locations.

In [12]:
```
customers2 = Table.read_table('customers2.csv')
customers2.hist(bins=np.arange(0, 2000+100, 100))
```

Question 8

Find the new best location for our ice cream truck.

In [15]:
```
# Hint: The staff solution defined a function called average_distance2.
# We recommend doing that.
def average_distance2(location):
    ...

new_best_location = ...
new_best_location
```

If you'd like to check your answer, try doing what you did in question 6. Your answer to question 3 may also be useful.

In [ ]:
```
# For your convenience, you can run this cell to run all the tests at on
ce!
import os
print("Running all tests...")
_ = [tests.grade(q[:-3]) for q in os.listdir("tests") if
q.startswith('q')]
print("Finished running all tests.")
```