hw07

March 28, 2020

1 Homework 7: Testing Hypotheses

Reading: * Testing Hypotheses

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 7 is due **Thursday, 3/26 at 11:59pm** (after spring break). Start early so that you can come to office hours if you're stuck. Late work will not be accepted as per the course policies.

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.

[17]:
```
# 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)
```

1.1 1. Catching Cheaters

Suppose you are a casino owner, and your casino runs a very simple game of chance. The dealer flips a coin. The customer wins $$$9 from the casino if it comes up heads and loses $$$10 if it comes up tails.

Question 1.1. Assuming no one is cheating and the coin is fair, if a customer plays twice, what is the chance they make money?

[18]:
```
p_winning_after_two_flips = 0.25
```

A certain customer plays the game 20 times and wins 13 of the bets. You suspect that the customer is cheating! That is, you think that their chance of winning is higher than the normal chance of winning.

You decide to test your hunch using the outcomes of the 20 games you observed.

Question 1.2. Define the null hypothesis and alternative hypothesis for this investigation.

Null hypothesis: The customer is not cheating

Alternative hypothesis: The customer is cheating

Question 1.3. Given the outcome of 20 games, which of the following test statistics would be a reasonable choice for this hypothesis test?

Hint: For a refresher on choosing test statistics, check out this section on Test Statistics.

1. Whether there is at least one win.
2. Whether there is at least one loss.
3. The number of wins.
4. The number of wins minus the number of losses.
5. The total variation distance between the probability distribution of a fair coin and the observed distribution of heads and tails.
6. The total amount of money that the customer won.

Assign `reasonable_test_statistics` to a **list** of numbers corresponding to these test statistics.

[19]:
```
reasonable_test_statistics = np.arange(4,6)
```

Suppose you decide to use the number of wins as your test statistic.

Question 1.4. Write a function called `simulate` that generates exactly one simulation of your test statistic under the Null Hypothesis. It should take no arguments. It should return the number of wins in 20 games simulated under the assumption that the result of each game is sampled from a fair coin that lands heads or lands tails with 50% chance.

Hint: You may find the textbook section on the `sample_proportions` function to be useful.

[20]:
```
def simulate():
    eligible_population= [0.50,0.50]
    wins=sample_proportions(20,eligible_population)
    wins= wins[0]*20
    return wins


simulate()
```

[20]:
```
11.0
```

Question 1.5. Using 10,000 trials, generate simulated values of the number of wins in 20 games. Assign `test_statistics_under_null` to an array that stores the result of each of these trials.

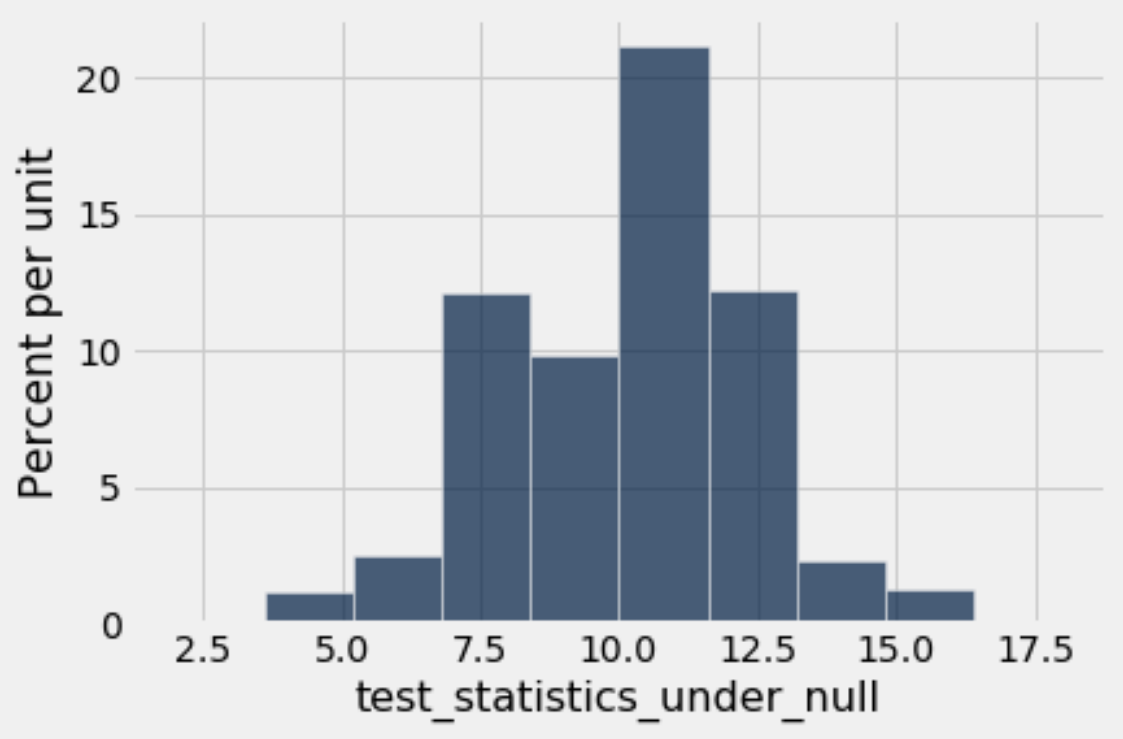
Hint: Feel free to use the function you defined in Question 4.

[21]:
```
test_statistics_under_null = make_array()
repetitions = 10000
for i in range(repetitions):
    test_statistics_under_null = np.append(test_statistics_under_null,␣
 ↪simulate())

test_statistics_under_null
```

[21]:
```
array([12., 11.,  5., ..., 10.,  9., 11.])
```

Question 1.6. Using the results from Question 5, generate a histogram of the empirical distribution of the number of wins in 20 games.

[22]:
```
Table().with_column('test_statistics_under_null', test_statistics_under_null).
 ↪hist()

#plt.hist(test_statistics_under_null,density=True,bins=20)
#plt.ylabel('test_statistics_under_null')
```


Percent per unit
20
15
10
5
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
test_statistics_under_null

Question 1.7. Compute an empirical P-value for this test.

Hint: Which values of our test statistic are in the direction of the alternative hypothesis?

[23]:
```
p_value = np.count_nonzero(test_statistics_under_null <= 10.0) / 10000
p_value
```

[23]:
```
0.5798
```

Question 1.8. Suppose you use a P-value cutoff of 1%. What do you conclude from the hypothesis test? Why?

We fail to reject the null hypothesis that the customer is not cheating because it's a fair coin.

Question 1.9. Is `p_value` the probability that the customer cheated, or the probability that the customer didn't cheat, or neither? If neither, what is it?

It's neither. The p-value shows the variance between an expected and observed values.

Question 1.10. Is 1% (the P-value cutoff) the probability that the customer cheated, or the probability that the customer didn't cheat, or neither? If neither, what is it?

1% is neither. It is the alpha value i.e., the probability that our test will produce a false positive.

Question 1.11. Suppose you run this test for 400 different customers after observing each customer play 20 games. When you reject the null hypothesis for a customer, you accuse that customer of cheating. If no customers were actually cheating, can we compute how many we will incorrectly accuse of cheating? If so, what is the number? Explain your answer. Assume a 1% P-value cutoff.

We can compute how many we will incorrectly accuse of cheating. The number is 4(alpha value i.e. 0.01 number of customers)*

1.2 2. Landing a Spacecraft

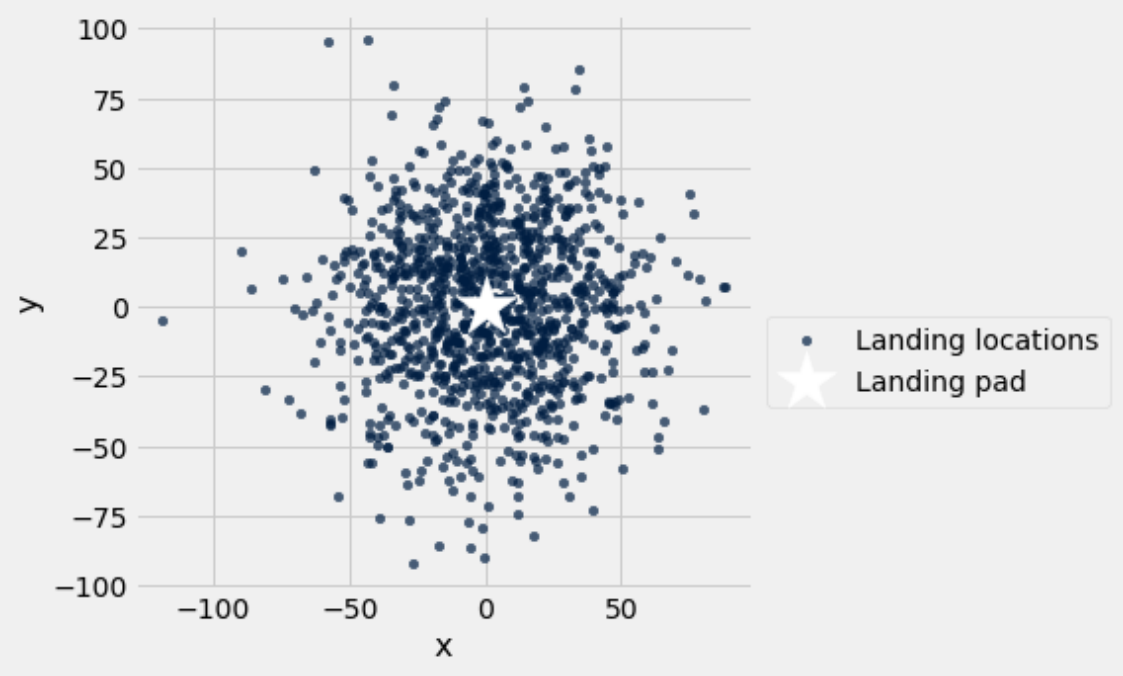
(Note: This problem describes something that's close to a real story with a very exciting video, but the details have been changed somewhat.)

SpaceY, a company that builds and tests spacecraft, is testing a new reusable launch system. Most spacecraft use a "first stage" rocket that propels a smaller payload craft away from Earth, then falls back to the ground and crashes. SpaceY's new system is designed to land safely at a landing pad at a certain location, ready for later reuse. If it doesn't land in the right location, it crashes, and the very, very expensive vehicle is destroyed.

SpaceY has tested this system over 1000 times. Ordinarily, the vehicle doesn't land exactly on the landing pad. For example, a gust of wind might move it by a few meters just before it lands. It's reasonable to think of these small errors as random. That is, the landing locations are drawn from some distribution over locations on the surface of Earth, centered around the landing pad.

Run the next cell to see a plot of those locations.

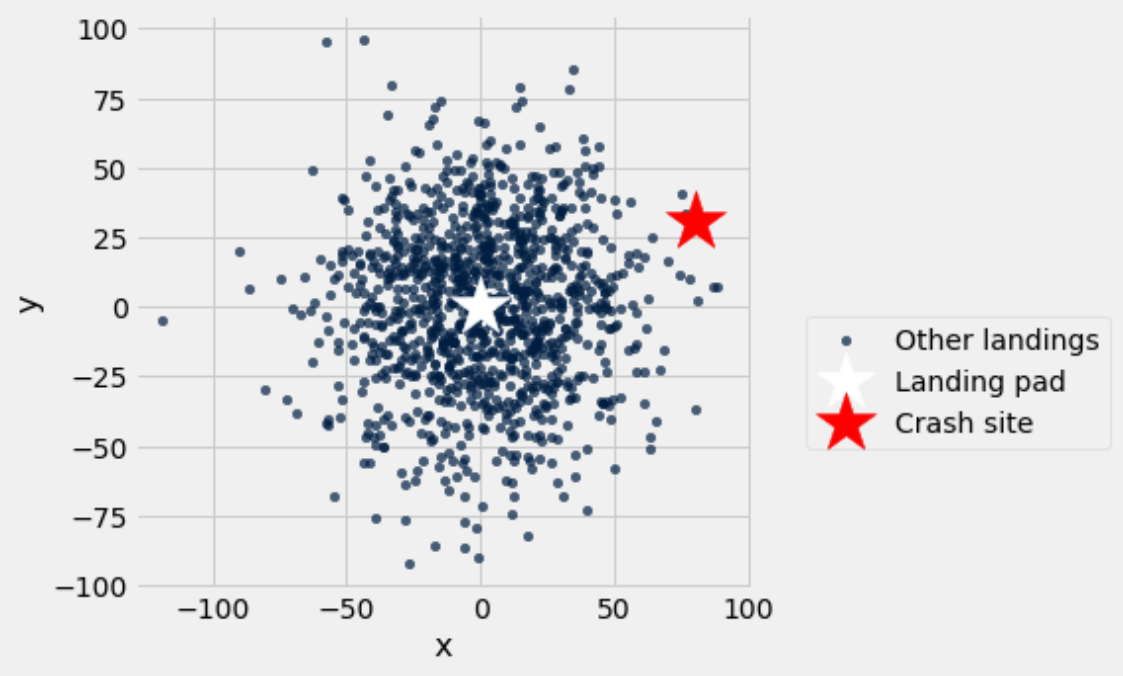
[24]:
```
ordinary_landing_spots = Table.read_table("ordinary_landing_spots.csv")
ordinary_landing_spots.scatter("x", label="Landing locations")
plt.scatter(0, 0, c="w", s=1000, marker="*", label="Landing pad")
plt.legend(scatterpoints=1, bbox_to_anchor=(1.6, .5));
```



During one test, the vehicle lands far away from the landing pad and crashes. SpaceY investigators suspect there was a problem unique to this landing, a problem that wasn't part of the ordinary pattern of variation in landing locations. They think a software error in the guidance system caused the craft to incorrectly attempt to land at a spot other than the landing pad. The guidance system engineers think there was nothing out of the ordinary in this landing, and that there was no special problem with the guidance system.

Run the cell below to see a plot of the 1100 ordinary landings and the crash.

[25]:
```
landing_spot = make_array(80.59, 30.91)
ordinary_landing_spots.scatter("x", label="Other landings")
plt.scatter(0, 0, c="w", s=1000, marker="*", label="Landing pad")
plt.scatter(landing_spot.item(0), landing_spot.item(1), marker="*", c="r",␣
↪s=1000, label="Crash site")
plt.legend(scatterpoints=1, bbox_to_anchor=(1.6, .5));
```



Question 2.1. Suppose we'd like to use hypothesis testing to shed light on this question. We've written down an alternative hypothesis below. What is a reasonable null hypothesis?

Null hypothesis: This landing was not special; its location was not a draw from some other distribution, but the distribution from which the other 1100 landing locations were drawn.

Alternative hypothesis: This landing was special; its location was a draw from some other distribution, not the distribution from which the other 1100 landing locations were drawn.

Question 2.2. What's a good test statistic for this hypothesis test?

Hint: A test statistic can be almost anything, but a *good* test statistic varies informatively depending on whether the null is true. So for this example, we might think about a test statistic that would be small if the null is true, and large otherwise. If we want to compare landings, we might want to see *how far* each landing is from some *reference point*, so we can compare all landings from the same vantage point.

Test statistic: compare the total variation distance between the probability distribution of the x and y coordinates of the ordinary landings and the X and Y coordinates of crash landings.

Question 2.3. Write a function called `landing_test_statistic`. It should take two arguments: an "x" location and a "y" location (both numbers). It should return the value of your test statistic for a landing at those coordinates.

[26]:
```
def landing_test_statistic(x_coordinate, y_coordinate):
    x_values= ordinary_landing_spots.column(0)
    y_values= ordinary_landing_spots.column(1)
    xp_value = np.count_nonzero(x_values <= x_coordinate ) / 1100
    yp_value = np.count_nonzero(y_values <= y_coordinate ) / 1100
    return min(xp_value,yp_value)
```

Question 2.4. The next three cells compute a P-value using your test statistic. Describe the test procedure in words. Is there a simulation involved? If so, what is being simulated? If not, why

not? Where are we getting the data from? What kind of calculations are being performed? How are we calculating our p-value?

Hint: Think about what a simulation actually consists of.

[27]:
```
observed_test_stat = landing_test_statistic(
    landing_spot.item(0),
    landing_spot.item(1))

observed_test_stat
```

[27]:
```
0.8418181818181818
```

[28]:
```
null_stats = make_array()
repetitions = ordinary_landing_spots.num_rows

for i in np.arange(repetitions):
    null_stat = landing_test_statistic(
        ordinary_landing_spots.column('x').item(i),
        ordinary_landing_spots.column('y').item(i))
    null_stats = np.append(null_stats, null_stat)

null_stats
```

[28]:
```
array([0.39363636, 0.10545455, 0.67727273, ..., 0.55727273, 0.26727273,
       0.28181818])
```

[29]:
```
p_value = np.count_nonzero(null_stats >= observed_test_stat) / len(null_stats)
p_value
```

[29]:
```
0.028181818181818183
```

There is no simulation involved because we have all the data we need in the table.In order to get the test statistic, we have to compute the distance between the point of landingand the landing pad. So, we can caluclate the p-value by taking all the values in statistics thatare larger or equal to the observed test statistics divided by the number of data point.

1.3 3. Testing Dice

Students in a Data Science class want to figure out whether a six-sided die is fair or not. On a fair die, each face of the die appears with chance 1/6 on each roll, regardless of the results of other rolls. Otherwise, a die is called unfair. We can describe a die by the probability of landing on each face. This table describes an example of a die that is unfairly weighted toward 1:

Face	Probability
1	.5
2	.1
3	.1
4	.1
5	.1
6	.1

Question 3.1. Define a null hypothesis and an alternative hypothesis to test whether a six-sided die is fair or not.

Hint: Remember that an unfair die is one for which each face does not have an equal chance of appearing.

Null hypothesis: There is an equal chance of each face appearing

Alternative hypothesis: There is not an equal chance of each face appearing.

We decide to test the die by rolling it 5 times. The proportions of the 6 faces in these 5 rolls are stored in a table with 6 rows. For example, here is the table we'd make if the die rolls ended up being 1, 2, 3, 3, and 5:

Face	Proportion
1	.2
2	.2
3	.4
4	.0
5	.2
6	.0

The function `mystery_test_statistic`, defined below, takes a single table like this as its argument and returns a number (which we will use as a test statistic).

[30]:
```
# Note: We've intentionally used unhelpful function and
# variable names to avoid giving away answers.  It's rarely
# a good idea to use names like "x" in your code.

def mystery_test_statistic(sample):
    x = np.ones(1) * (1/6)
    y = (sample.column('Proportion') - x)
    return np.mean(y**2)
```

[ ]:
```

```

Question 3.2. Describe in English what the test statistic is. Is it equivalent to the total variation distance between the observed face distribution and the fair die distribution?

It is equivalent to the total variation distance between the observed face distribution and the fair die distribution

The function `simulate_observations_and_test` takes as its argument a table describing the probability distribution of a die. It simulates one set of 5 rolls of that die, then tests the null hypothesis about that die using our test statistic function above. It returns `False` if it *rejects* the null hypothesis about the die, and `True` otherwise.

[40]:
```
# The probability distribution table for a fair die:
fair_die = Table().with_columns(
        "Face", np.arange(1, 6+1),
        "Probability", [1/6, 1/6, 1/6, 1/6, 1/6, 1/6])

def simulate_observations_and_test(actual_die):
    """Simulates die rolls from actual_die and tests the hypothesis that the␣
 ↪die is fair.
```

```
    Returns False if that hypothesis is rejected, and True otherwise.

    """

    sample_size = 5
    p_value_cutoff = .2
    num_simulations = 500 #250

    # Compute the observed value of the test statistic.
    observation_set = sample_proportions(sample_size, actual_die.
↪column("Probability"))
    observation_props_table = Table().with_columns('Face', actual_die.
↪column('Face'), 'Proportion', observation_set)
    observed_statistic = mystery_test_statistic(observation_props_table)

    # Simulate the test statistic repeatedly to get an
    # approximation to the probability distribution of
    # the test statistic, as predicted by the model in
    # the null hypothesis. Store the simulated values
    # of the test statistic in an array.
    simulated_statistics = make_array()
    for _ in np.arange(num_simulations):
        one_observation_set_under_null = sample_proportions(sample_size,␣
↪fair_die.column("Probability"))
        simulated_props_table = Table().with_columns('Face', fair_die.
↪column('Face'), 'Proportion', one_observation_set_under_null)
        simulated_statistic = mystery_test_statistic(simulated_props_table)
        simulated_statistics = np.append(simulated_statistics,␣
↪simulated_statistic)

    # Compute the P-value
    p_value = np.count_nonzero(simulated_statistics >= observed_statistic) /␣
↪num_simulations

    # If the P-value is below the cutoff, reject the
    # null hypothesis and return False. Otherwise,
    # return True.
    return p_value >= p_value_cutoff

# Calling the function to simulate a test of a fair die:
simulate_observations_and_test(fair_die)
```

[40]: `True`

Question 3.3. Use your knowledge of hypothesis tests and interpretation of the code above to compute the probability that `simulate_observations_and_test` returns `False` when its argument is `fair_die` (which is defined above the function). In other words, what is the probability

that we reject the Null Hypothesis if the die is actually fair.

You can call the function a few times to see what it does, but **don't** perform a simulation to compute this probability. Use your knowledge of hypothesis tests. You shouldn't have to write any code to answer this question.

[41]:
```
probability_of_false = 0.2
```

Question 3.4. Why is your answer to Question 3 the correct probability?

because it's the alpha value which is the probability of rejecting the null hypothesis when it is true

Question 3.5. Simulate the process of running `simulation_observations_and_test` 300 times. Assign `test_results` to an array that stores the result of each of these trials.

Note: This will be a little slow. 300 repetitions of the simulation should require a minute or so of computation, and should suffice to get an answer that's roughly correct.

[52]:
```
test_results = make_array()
num_test_simulations = 300

for i in range(num_test_simulations):
    test_results = np.append(test_results,␣
 ↪simulate_observations_and_test(fair_die))

# Don't change the following line.
test_results.astype(bool)
```

[52]:
```
array([ True,  True,  True, False,  True,  True,  True,  True,  True,
       False,  True,  True,  True,  True,  True, False, False,  True,
        True,  True,  True,  True,  True, False,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True, False,  True,  True,  True,  True, False,  True,
        True,  True,  True, False,  True,  True,  True, False, False,
       False,  True, False, False,  True,  True, False,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
       False,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True, False, False,  True,  True,  True,
       False, False,  True,  True,  True,  True,  True,  True,  True,
       False,  True,  True,  True, False,  True,  True,  True,  True,
       False, False,  True,  True,  True,  True,  True,  True,  True,
        True, False, False,  True,  True,  True,  True,  True,  True,
       False,  True,  True, False, False,  True, False, False,  True,
        True,  True,  True,  True,  True, False, False,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True, False,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True, False,  True,  True, False,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True, False,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True, False,
```