COSE471_2023sp_hw1

March 30, 2023

Table of Contents

1.8.1 Question 7a (1pt)

1.8.2 Question 7b (1pt)

1.8.3 Question 7c (1pt)

1.8.4 Question 7d (1pt)

1.8.5 Question 7e

1.8.6 Congratulations! You have completed HW1.

1 HW 1: Math Review and Plotting

1.1 Due Date: Fri 3/31, 11:59 PM

Collaboration Policy: You may talk with others about the homework, but we ask that you **write your solutions individually**. If you do discuss the assignments with others, please **include their names** in the following line.

Collaborators: *list collaborators here (if applicable)*

1.2 This Assignment

This homework is to help you diagnose your preparedness for the course. The rest of this course will assume familiarity with the programming and math concepts covered in this homework. Please consider reviewing prerequisite material if you struggle with this homework.

1.3 Score Breakdown

Question	Points
1	1
2a	1
2b	1
2c	1
2d	1
3a	4
3b	2
4a	2
4b	2
5	2
6a	2
6b	1
6c	1
7	5
Total	30

Here are some useful Jupyter notebook keyboard shortcuts. To learn more keyboard shortcuts, go to **Help -> Keyboard Shortcuts** in the menu above.

Here are a few we like: 1. `ctrl+return` : *Evaluate the current cell* 1. `shift+return`: *Evaluate the current cell and move to the next* 1. `esc` : *command mode* (may need to press before using any of the commands below) 1. `a` : *create a cell above* 1. `b` : *create a cell below* 1. `dd` : *delete a cell* 1. `m` : *convert a cell to markdown* 1. `y` : *convert a cell to code*

1.3.1 Initialize your environment

This cell should run without error if you have **set up your personal computer correctly**.

[2]:
```
import numpy as np
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')

from IPython.display import display, Latex, Markdown
```

1.4 Python

1.4.1 Question 1 (1 pt)

Recall the formula for population variance below:

$$\texttt{Mean}(\mathbf{x}) = \mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\texttt{Var}(\mathbf{x}) = \sigma^2 = \frac{1}{N}\sum_{i=1}^{N} (x_i - \mu)^2$$

Complete the functions below to compute the variance of `population`, an array of numbers. For this question, do not use built-in NumPy functions (i.e. `np.mean` and `np.var`); instead we will use NumPy to verify your code.

[3]:
```
def mean(population):
    """
    Compute the mean of population (mu).

    Args:
        population: a numpy array of numbers of shape [N,]
    Returns:
        the mean of population (mu).
    """
    # Calculate the mean of a population
    # BEGIN YOUR CODE
    # ------------------------
    return sum(population) / len(population)
    # ------------------------
    # END YOUR CODE
```

```
def variance(population):
    """
    Compute the variance of population (sigma squared).

    Args:
        population: a numpy array of numbers of shape [N,]
    Returns:
        the variance of population
    """
    # Calculate the variance of a population
    # BEGIN YOUR CODE
    # ------------------------
    mu = mean(population)
    return sum(map(lambda x:(x-mu)**2, population)) / len(population)
    # ------------------------
    # END YOUR CODE
```

[7]:
```
population_0 = np.random.randn(100)
assert np.isclose(mean(population_0), np.mean(population_0), atol=1e-6)
assert np.isclose(variance(population_0), np.var(population_0), atol=1e-6)
population_1 = 3 * np.random.randn(100) + 5
assert np.isclose(mean(population_1), np.mean(population_1), atol=1e-6)
assert np.isclose(variance(population_1), np.var(population_1), atol=1e-6)
```

1.5 NumPy

You should be able to understand the code in the following cells. If not, please review the following:

- UC Berkeley DS100 NumPy Review
- Stanford Condensed NumPy Review
- The Official NumPy Tutorial

Jupyter pro-tip: Pull up the docs for any function in Jupyter by running a cell with the function name and a `?` at the end:

[21]:
```
np.arange?
```

```
Docstring:
arange([start,] stop[, step,], dtype=None, *, like=None)

Return evenly spaced values within a given interval.

``arange`` can be called with a varying number of positional arguments:

* ``arange(stop)``: Values are generated within the half-open interval
  ``[0, stop)`` (in other words, the interval including `start` but
```

```
  excluding `stop`).
* ``arange(start, stop)``: Values are generated within the half-open
  interval ``[start, stop)``.
* ``arange(start, stop, step)`` Values are generated within the half-open
  interval ``[start, stop)``, with spacing between values given by
  ``step``.

For integer arguments the function is roughly equivalent to the Python
built-in :py:class:`range`, but returns an ndarray rather than a ``range``
instance.

When using a non-integer step, such as 0.1, it is often better to use
`numpy.linspace`.

See the Warning sections below for more information.

Parameters
----------
start : integer or real, optional
    Start of interval.  The interval includes this value.  The default
    start value is 0.
stop : integer or real
    End of interval.  The interval does not include this value, except
    in some cases where `step` is not an integer and floating point
    round-off affects the length of `out`.
step : integer or real, optional
    Spacing between values.  For any output `out`, this is the distance
    between two adjacent values, ``out[i+1] - out[i]``.  The default
    step size is 1.  If `step` is specified as a position argument,
    `start` must also be given.
dtype : dtype, optional
    The type of the output array.  If `dtype` is not given, infer the data
    type from the other input arguments.
like : array_like, optional
    Reference object to allow the creation of arrays which are not
    NumPy arrays. If an array-like passed in as ``like`` supports
    the ``__array_function__`` protocol, the result will be defined
    by it. In this case, it ensures the creation of an array object
    compatible with that passed in via this argument.

    .. versionadded:: 1.20.0

Returns
-------
arange : ndarray
    Array of evenly spaced values.

    For floating point arguments, the length of the result is
```

```
    ``ceil((stop - start)/step)``.  Because of floating point overflow,
    this rule may result in the last element of `out` being greater
    than `stop`.

Warnings
--------
The length of the output might not be numerically stable.

Another stability issue is due to the internal implementation of
`numpy.arange`.
The actual step value used to populate the array is
``dtype(start + step) - dtype(start)`` and not `step`. Precision loss
can occur here, due to casting or due to using floating points when
`start` is much larger than `step`. This can lead to unexpected
behaviour. For example::

  >>> np.arange(0, 5, 0.5, dtype=int)
  array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
  >>> np.arange(-3, 3, 0.5, dtype=int)
  array([-3, -2, -1,  0,  1,  2,  3,  4,  5,  6,  7,  8])

In such cases, the use of `numpy.linspace` should be preferred.

The built-in :py:class:`range` generates :std:doc:`Python built-in integers
that have arbitrary size <python:c-api/long>`, while `numpy.arange`
produces `numpy.int32` or `numpy.int64` numbers. This may result in
incorrect results for large integer values::

  >>> power = 40
  >>> modulo = 10000
  >>> x1 = [(n ** power) % modulo for n in range(8)]
  >>> x2 = [(n ** power) % modulo for n in np.arange(8)]
  >>> print(x1)
  [0, 1, 7776, 8801, 6176, 625, 6576, 4001]  # correct
  >>> print(x2)
  [0, 1, 7776, 7185, 0, 5969, 4816, 3361]  # incorrect

See Also
--------
numpy.linspace : Evenly spaced numbers with careful handling of endpoints.
numpy.ogrid: Arrays of evenly spaced numbers in N-dimensions.
numpy.mgrid: Grid-shaped arrays of evenly spaced numbers in N-dimensions.
:ref:`how-to-partition`

Examples
--------
>>> np.arange(3)
array([0, 1, 2])
```

```
>>> np.arange(3.0)
array([ 0.,  1.,  2.])
>>> np.arange(3,7)
array([3, 4, 5, 6])
>>> np.arange(3,7,2)
array([3, 5])
Type:      builtin_function_or_method
```

You can close the window at the bottom by pressing `esc` several times.

Another Jupyter pro-tip: Pull up the docs for any function in Jupyter by typing the function name, then `<Shift>-<Tab>` on your keyboard. This is super convenient when you forget the order of the arguments to a function. You can press `<Tab>` multiple times to expand the docs and reveal additional information.

Try it on the function below:

[22]:
```
np.linspace
```

[22]:
```
<function numpy.linspace(start, stop, num=50, endpoint=True, retstep=False,
dtype=None, axis=0)>
```

Now, let's go through some linear algebra coding questions with NumPy. In this question, we'll ask you to use your linear algebra knowledge to fill in NumPy matrices. To conduct matrix multiplication in NumPy, you should write code like the following:

[8]:
```
# A matrix in NumPy is a 2-dimensional NumPy array
matA = np.array([
    [1, 2, 3],
    [4, 5, 6],
])

matB = np.array([
    [10, 11],
    [12, 13],
    [14, 15],
])

# The notation B @ v means: compute the matrix multiplication Bv
matA @ matB
```

[8]:
```
array([[ 76,  82],
       [184, 199]])
```

You can also use the same syntax to do matrix-vector multiplication or vector dot products. Handy!

[24]:
```
matA = np.array([
    [1, 2, 3],
    [4, 5, 6],
])
```

```
# A vector in NumPy is simply a 1-dimensional NumPy array
some_vec = np.array([ 10, 12, 14, ])
another_vec = np.array([ 10, 20, 30 ])

print(matA @ some_vec)
print(some_vec @ another_vec)
```

```
[ 76 184]
760
```

1.6 Question 2 (4 pt)

1.6.1 Question 2a

Joey, Deb, and Sam are shopping for fruit at K-Bowl. K-Bowl, true to its name, only sells fruit bowls. A fruit bowl contains some fruit and the price of a fruit bowl is the total price of all of its individual fruit.

Berkeley Bowl has apples for \$2.00, bananas for \$1.00, and cantaloupes for \$4.00 (expensive!). The price of each of these can be written in a vector:

$$v = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix}$$

K-Bowl sells the following fruit bowls:

1. 2 of each fruit
2. 5 apples and 8 bananas
3. 2 bananas and 3 cantaloupes
4. 10 cantaloupes

Create a 2-dimensional numpy array encoding the matrix B such that the matrix-vector multiplication

$$Bv$$

evaluates to a length 4 column vector containing the price of each fruit bowl. The first entry of the result should be the cost of fruit bowl #1, the second entry the cost of fruit bowl #2, etc.

[9]:
```
v = np.array([2,1,4])

# BEGIN YOUR CODE
# ----------------------
B = np.array([
    [2, 2, 2],
    [5, 8, 0],
    [0, 2, 3],
    [0, 0, 10]
```

```
    ])
# ----------------------
# END YOUR CODE

# The notation B @ v means: compute the matrix multiplication Bv
B @ v
```

[9]:
```
array([14, 18, 14, 40])
```

[11]:
```
assert B.shape == (4, 3)
assert np.allclose(B @ v, np.array([14, 18, 14, 40]))
```

1.6.2 Question 2b

Joey, Deb, and Sam make the following purchases:

- Joey buys 2 fruit bowl #1s and 1 fruit bowl #2.
- Deb buys 1 of each fruit bowl.
- Sam buys 10 fruit bowl #4s (he really like cantaloupes).

Create a matrix A such that the matrix expression

$$ABv$$

evaluates to a length 3 column vector containing how much each of them spent. The first entry of the result should be the total amount spent by Joey, the second entry the amount sent by Deb, etc.

Note that the tests for this question do not tell you whether your answer is correct. That's up to you to determine.

[12]:
```
A = np.array([
    [2, 1, 0, 0],
    # Finish this!
    # BEGIN YOUR CODE
    # ----------------------
    [1, 1, 1, 1],
    [0, 0, 0, 10]
    # ----------------------
    # END YOUR CODE
])

A @ B @ v
```

[12]:
```
array([ 46,  86, 400])
```

[14]:
```
assert A.shape == (3, 4)
assert np.allclose(A @ B @ v , np.array([ 46,  86, 400]))
```

1.6.3 Question 2c

Who spent the most money? Assign `most` to a string containing the name of this person.

[16]:
```
# BEGIN YOUR CODE
# ----------------------
most = "Sam"
# ----------------------
# END YOUR CODE
```

[17]:
```
assert most in ["Joey", "Deb", "Sam"]
```

1.6.4 Question 2d

Let's suppose K-Bowl changes their fruit prices, but you don't know what they changed their prices to. Joey, Deb, and Sam buy the same quantity of fruit baskets and the number of fruit in each basket is the same, but now they each spent these amounts:

$$x = \begin{bmatrix} 80 \\ 80 \\ 100 \end{bmatrix}$$

Use `np.linalg.inv` and the above final costs to compute the new prices for the individual fruits as a vector called `new_v`.

[28]:
```
# BEGIN YOUR CODE
# ----------------------
new_v = np.linalg.inv(A @ B) @ [80, 80, 100]
# ----------------------
# END YOUR CODE
new_v
```

[28]:
```
array([5.5       , 2.20833333, 1.        ])
```

[29]:
```
assert new_v.shape == (3,)
assert np.allclose(new_v, np.array([ 5.5,  2.20833333, 1.]))
```

1.7 Multivariable Calculus, Linear Algebra, and Probability

The following questions ask you to recall your knowledge of multivariable calculus, linear algebra, and probability. We will use some of the most fundamental concepts from each discipline in this class, so the following problems should at least seem familiar to you.

If you have trouble with these topics, we suggest reviewing:

- Khan Academy's Multivariable Calculus
- Khan Academy's Linear Algebra
- Khan Academy's Statistics and Probability

LaTeX For the following problems, you should use LaTeX to format your answer. If you aren't familiar with LaTeX, not to worry. It's not hard to use in a Jupyter notebook. Just place your math in between dollar signs:

\$ f(x) = 2x \$ becomes $ f(x) = 2x $.

If you have a longer equation, use double dollar signs to place it on a line by itself:

\
_{i=0}^n i^2 \
becomes:

$$\sum_{i=0}^{n} i^2$$

Here is some handy notation:

Output	Latex
x^{a+b}	`x^{a + b}`
x_{a+b}	`x_{a + b}`
$\frac{a}{b}$	`\frac{a}{b}`
$\sqrt{a+b}$	`\sqrt{a + b}`
$\{\alpha, \beta, \gamma, \pi, \mu, \sigma^2\}$	`\{ \alpha, \beta, \gamma, \pi, \mu, \sigma^2 \}`
$\sum_{x=1}^{100}$	`\sum_{x=1}^{100}`
$\frac{\partial}{\partial x}$	`\frac{\partial}{\partial x}`
$\begin{bmatrix} 2x + 4y \\ 4x + 6y^2 \end{bmatrix}$	`\begin{bmatrix} 2x + 4y \\ 4x + 6y^2 \\ \end{bmatrix}`

For more about basic LaTeX formatting, you can read this article.

1.7.1 Question 3a (4 pt)

Suppose we have the following scalar-valued function:

$$f(x, y) = x^2 + 4xy + 2y^3 + e^{-3y} + \ln(2y)$$

Compute the partial derivative $\frac{\partial}{\partial x} f(x, y)$:

Answer: $2x + 4y$

Now compute the partial derivative $\frac{\partial}{\partial y} f(x, y)$:

Answer: $4x + 6y^2 - 3e^{-3y} + \frac{1}{y}$

Finally, using your answers to the above two parts, compute $\nabla f(x, y)$. Also what is the gradient at the point (x, y) = (2, -1):

Note that ∇ represents the gradient.

Answer: $\begin{bmatrix} 2x+4y \\ 4x+6y^2-3e^{-3y}+\frac{1}{y} \end{bmatrix}$, $\begin{bmatrix} 0 \\ 13-3e^3 \end{bmatrix}$

1.7.2 Question 3b (2 pt)

Find the value(s) of x which minimizes the expression below. Justify why it is the minimum.

$\sum_{i=1}^{10}(i-x)^2$

Answer: $\frac{d}{dx}\sum_{i=1}^{10}(i-x)^2 = 10x^2 - 110$. It gets to zero on $\frac{11}{2}$.
Since the given function is a quadratic function with respect to x, this is the lowest value.

1.7.3 Question 4a (2 pt)

Let $\sigma(x) = \dfrac{1}{1+e^{-x}}$.

Show that $\sigma(-x) = 1 - \sigma(x)$.

Answer: $\sigma(-x) = \frac{1}{1+e^x} = \frac{1+e^x-e^x}{1+e^x} = 1 - \frac{e^x}{1+e^x} = 1 - \frac{1}{(1+e^x)e^{-x}} = 1 - \frac{1}{1+e^{-x}} = 1 - \sigma(x)$

1.7.4 Question 4b (2 pt)

Show that the derivative can be written as:

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x))$$

Answer: $\frac{d}{dx}\sigma(x) = \frac{0+e^{-x}}{(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2} = \frac{1}{(1+e^{-x})} - \frac{1}{(1+e^{-x})^2} = \sigma(x)(1-\sigma(x))$
By quotient rule.

1.7.5 Question 5 (2 pt)

Consider the following scenario:

Only 1% of 40-year-old women who participate in a routine mammography test have breast cancer. 80% of women who have breast cancer will test positive, but 9.6% of women who don't have breast cancer will also get positive tests.

Suppose we know that a woman of this age tested positive in a routine screening. What is the probability that she actually has breast cancer?

Hint: Use Bayes' rule.

Answer: Bayes' rule is $P(B|A) = \frac{P(A|B)P(B)}{P(A)}$.
In this case, getting positive in test is A, having breast cancer is B.
$P(B|A) = \frac{0.8 \cdot 0.01}{0.01 \cdot 0.8 + 0.99 \cdot 0.096} = \frac{0.008}{0.10304} \sim 0.078$
Therefore, answer is 7.8%

1.7.6 Question 6

Consider (once again) a sample of size n drawn at random with replacement from a population in which a proportion p of the individuals are called successes.

Let S be the random variable that denotes the number of successes in our sample. Then, the probability that the number of successes in our sample is **at most** s (where $0 \leq s \leq n$) is

$$P(S \leq s) = P(S = 0) + P(S = 1) + ... + P(S = s) = \sum_{k=0}^{s} \binom{n}{k} p^k (1-p)^{n-k}$$

We obtain this by summing the probability that the number of successes is exactly k, for each value of $k = 0, 1, 2, ..., s$.

1.7.7 Question 6a (2pt)

Please fill in the function `prob_at_most` which takes n, p, and s and returns $P(S \leq s)$ as defined above. If the inputs are invalid: for instance, if p > 1 or s > n then return 0."

Hint: One way to compute the binomial coefficients is to use SciPy module, which is a collection of Python-based software for math, probability, statistics, science, and enginnering. Feel free to use scipy.special.comb**

[25]:
```
from scipy import special

def prob_at_most(n, p, s):
    """
    returns the probability of S <= s
    Input n: sample size; p : proportion; s: number of successes at most
    """
    if p > 1 or s > n:
        return 0

    # BEGIN YOUR CODE
    # ------------------------
    all_probs = []

    for k in range(s+1):
        all_probs.append(special.comb(n, k, exact=True) * (p ** k) * ((1 - p)␣
 ↪** (n - k)))
    # ------------------------
    # END YOUR CODE

    return sum(all_probs[:s+1])
```

[26]:
```
assert prob_at_most(3, 0.4, 1) >= 0
assert prob_at_most(5, 0.6, 3) <= 1
assert prob_at_most(2, 3, 4) == 0
```

1.7.8 Question 6b (1pt)

In an election, supporters of Candidate C are in a minority. Only 45% of the voters in the population favor the candidate.

Suppose a survey organization takes a sample of 200 voters at random with replacement from this population. Use `prob_at_most` to write an expression that evaluates to the chance that a majority (more than half) of the sampled voters favor Candidate C.

[53]:
```
# BEGIN YOUR CODE
# ------------------------
p_majority = prob_at_most(200, 0.55, 99)
# prob_at_most(200, 0.55, 99) == 1-prob_at_most(200, 0.45, 100))
# ------------------------
# END YOUR CODE
p_majority
```

[53]:
```
0.06807524986274854
```

[6]:
```
assert p_majority >= 0 and p_majority <= 1
```

1.7.9 Question 6c (1pt)

Suppose each of five survey organizations takes a sample of voters at random with replacement from the population of voters in Part **b**, independently of the samples drawn by the other organizations.

- Three of the organizations use a sample size of 200
- One organization uses a sample size of 300
- One organization uses a sample size of 400

Write an expression that evaluates to the chance that in at least one of the five samples the majority of voters favor Candidate C. You can use any quantity or function defined earlier in this exercise.

[56]:
```
# BEGIN YOUR CODE
# ------------------------
a, b, c = prob_at_most(200, 0.45, 100), prob_at_most(300, 0.45, 150),␣
 ↪prob_at_most(400, 0.45, 200)
prob_6c = 1 - a * a * a * b * c
# ------------------------
# END YOUR CODE
prob_6c
```

[56]:
```
0.23550361568476286
```

[54]:
```
assert prob_6c >= 0 and prob_6c <= 1
```

1.8 The US Presidential Election

The US president is chosen by the Electoral College, not by the popular vote. Each state is alotted a certain number of electoral college votes, as a function of their population. Whomever wins in the state gets all of the electoral college votes for that state.

There are 538 electoral college votes (hence the name of the Nate Silver's site, FiveThirtyEight).

Pollsters correctly predicted the election outcome in 46 of the 50 states. For these 46 states Trump received 231 and Clinton received 232 electoral college votes.

The remaining 4 states accounted for a total of 75 votes, and whichever candidate received the majority of the electoral college votes in these states would win the election.

These states were Florida, Michigan, Pennsylvania, and Wisconsin.

State	Electoral College Votes
Florida	29
Michigan	16
Pennsylvania	20
Wisconsin	10

For Donald Trump to win the election, he had to win either: * Florida + one (or more) other states * Michigan, Pennsylvania, and Wisconsin

The electoral margins were very narrow in these four states, as seen below:

State	Trump	Clinton	Total Voters
Florida	49.02	47.82	9,419,886
Michigan	47.50	47.27	4,799,284
Pennsylvania	48.18	47.46	6,165,478
Wisconsin	47.22	46.45	2,976,150

Those narrow electoral margins can make it hard to predict the outcome given the sample sizes that the polls used.

1.8.1 Question 7a (1pt)

For your convenience, the results of the vote in the four pivotal states is repeated below:

State	Trump	Clinton	Total Voters
Florida	49.02	47.82	9,419,886
Michigan	47.50	47.27	4,799,284
Pennsylvania	48.18	47.46	6,165,478
Wisconsin	47.22	46.45	2,976,150

Using the table above, write a function `draw_state_sample(N, state)` that returns a sample with replacement of N voters from the given state. Your result should be returned as a list, where the first element is the number of Trump votes, the second element is the number of Clinton votes, and the third is the number of Other votes. For example, `draw_state_sample(1500, "florida")` could return `[727, 692, 81]`. You may assume that the state name is given in all lower case.

You might find `np.random.multinomial` useful.

[3]:
```
def draw_state_sample(N, state):
    # BEGIN YOUR CODE
    # ----------------------
```

```
    state_dict_t = {'florida': 49.02, 'michigan': 47.50, 'pennsylvania': 48.18,␣
↪'wisconsin': 47.22}
    state_dict_c = {'florida': 47.82, 'michigan': 47.27, 'pennsylvania': 47.46,␣
↪'wisconsin': 46.45}
    return np.random.multinomial(N, [state_dict_t[state]/100,␣
↪state_dict_c[state]/100, 1 - state_dict_t[state]/100 - state_dict_c[state]/
↪100])
    # -----------------------
    # END YOUR CODE
```

[4]:
```
assert len(draw_state_sample(1500, "florida")) == 3
assert sum(draw_state_sample(1500, "michigan")) == 1500
q7a_penn = draw_state_sample(1500, "pennsylvania")
trump_win_penn = (q7a_penn[0] - q7a_penn[1]) / 1500
abs(trump_win_penn - 0.007) <= 0.12
```

[4]:
```
True
```

1.8.2 Question 7b (1pt)

Now, create a function `trump_advantage` that takes in a sample of votes (like the one returned by `draw_state_sample`) and returns the difference in the proportion of votes between Trump and Clinton. For example `trump_advantage([100, 60, 40])` would return `0.2`, since Trump had 50% of the votes in this sample and Clinton had 30%.

[5]:
```
def trump_advantage(voter_sample):
    # BEGIN YOUR CODE
    # -----------------------
    trump, clinton, _ = voter_sample
    n = sum(voter_sample)

    return (trump - clinton) / n
    # -----------------------
    # END YOUR CODE
```

[6]:
```
assert -1 < trump_advantage(draw_state_sample(1500, "wisconsin")) < 1
assert np.isclose(trump_advantage([100, 60, 40]), 0.2)
assert np.isclose(trump_advantage([10, 30, 10]), -0.4)
```

1.8.3 Question 7c (1pt)

Simulate Trump's advantage across 100,000 samples of 1500 voters for the state of Pennsylvania and store the results of each simulation in a list called `simulations`.

That is, `simulations[i]` should be Trump's proportion advantage for the `i+1`th simple random sample.

[7]:
```
# BEGIN YOUR CODE
# ----------------------
simulations = [trump_advantage(draw_state_sample(1500, 'pennsylvania')) for _␣
 ↪in range(100_000)]
# ----------------------
# END YOUR CODE
```

[8]:
```
assert len(simulations) == 100000
assert sum([-1 < x < 1 for x in simulations]) == len(simulations)
assert abs(np.mean(simulations) - 0.007) <= 0.016
```

1.8.4 Question 7d (1pt)

Now write a function `trump_wins(N)` that creates a sample of N voters for each of the four crucial states (Florida, Michigan, Pennsylvania, and Wisconsin) and returns 1 if Trump is predicted to win based on these samples and 0 if Trump is predicted to lose.

Recall that for Trump to win the election, he must either: * Win the state of Florida and 1 or more other states * Win Michigan, Pennsylvania, and Wisconsin

[10]:
```
def trump_wins(N):
    # BEGIN YOUR CODE
    # ----------------------
    fl = draw_state_sample(N, 'florida')
    mi = draw_state_sample(N, 'michigan')
    pe = draw_state_sample(N, 'pennsylvania')
    wi = draw_state_sample(N, 'wisconsin')

    f = fl[0] > max(fl[1], fl[2])
    m = mi[0] > max(mi[1], mi[2])
    p = pe[0] > max(pe[1], pe[2])
    w = wi[0] > max(wi[1], wi[2])

    if f and True in (m, p, w):
        return 1
    if m == True and p == True and w == True:
        return 1
    return 0
    # ----------------------
    # END YOUR CODE
```

[11]:
```
assert trump_wins(1000) in [0, 1]
```

1.8.5 Question 7e

If we repeat 100,000 simulations of the election, i.e. we call `trump_wins(1500)` 100,000 times, what proportion of these simulations predict a Trump victory? Give your answer as `proportion_trump`.

This number represents the percent chance that a given sample will correctly predict Trump's victory *even if the sample was collected with absolutely no bias.*

[17]:
```
# BEGIN YOUR CODE
# ----------------------
proportion_trump = sum([trump_wins(1500) for _ in range(100_000)]) / 100_000
# ----------------------
# END YOUR CODE
proportion_trump
```

[17]:
```
0.69446
```

[15]:
```
assert 0 < proportion_trump < 1
assert abs(proportion_trump - 0.695) <= 0.02
```

1.8.6 Congratulations! You have completed HW1.

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output.,

Please save before submitting!

Please generate pdf as follows and submit it to Gradescope.

File > Print Preview > Print > Save as pdf