Midterm Exam - DSC 10, Winter 2025

Full Name:

PID:

Exam Time: ◯ A (10AM) ◯ B (11AM)

Instructions:

- This exam consists of 6 questions, worth a total of 85 points.
- Write your PID in the top right corner of each page in the space provided.
- Please write **clearly** in the provided answer boxes; we will not grade work that appears elsewhere. Completely fill in bubbles and square boxes; if we cannot tell which option(s) you selected, you may lose points.

  ◯ A bubble means that you should only **select one choice**.

  ☐ A square box means you should **select all that apply**.
- For full credit, your solutions must use methods of the course.
- You may use one page of double-sided handwritten notes. Aside from this, you may not refer to any other resources or technology during the exam. No calculators!

By signing below, you are agreeing that you will behave honestly and fairly during and after this exam.

Signature:

Version A

Please do not open your exam until instructed to do so.

Dining Halls

In this exam, you'll work with a data set showcasing the different dining halls at UCSD and the foods served there. Each row represents a single menu item available at one of the UCSD dining halls.

The columns of `dining` are as follows:

- `"Dining Hall"` (`str`): The name of the dining hall.
- `"Item"` (`str`): The name of the menu item.
- `"Price"` (`str`): The cost of the menu item.
- `"Calories"` (`int`): The number of calories in the menu item.

The rows of `dining` are in no particular order. The first few rows are shown below, though `dining` has many more rows than pictured.

	Dining Hall	Item	Price	Calories
0	Café Ventanas	Crispy Chicken Sandwich	$9.50	1281
1	64 Degrees	Ancho Sesame Shrimp Fritas	$12.00	721
2	Club Med	Ahi Poke Bowl	$11.00	317
3	64 Degrees	Crispy Chicken Sandwich	$9.00	868
4	Pines	Ube Pancakes	$7.00	243

Assume that we have already run `import babypandas as bpd` and `import numpy as np`.

Important: Before proceeding, make sure to rip off the last page of this exam packet and read the data description.

Question 1 (4 pts)

Which of the following columns would be an appropriate index for the `dining` DataFrame?

◯ `"Dining Hall"`

◯ `"Item"`

◯ `"Price"`

◯ `"Calories"`

◯ None of these.

Question 2 (18 pts)

As a broke college student, you are on a mission to find the dining hall with the greatest number of affordable menu items.

a) (7 pts) To begin, you want a DataFrame with the same columns as `dining`, but with an additional column `"Affordability"` which classifies each menu item as follows:

- `"Cheap"`, for items that cost \$6 or less.
- `"Moderate"`, for items that cost more than \$6 and at most \$14.
- `"Expensive"`, for items that cost more than \$14.

Fill in the blanks below to assign this new DataFrame to the variable `with_affordability`.

```
def categorize(price):
    price_as_float = __(a)__
    if price_as_float __(b)__:
        return __(c)__
    elif price_as_float > 6:
        return "Moderate"
    else:
        return __(d)__
with_affordability = dining.assign(Affordability = __(e)__)
```

(a): [ ]

(b): [ ] (c): [ ] (d): [ ]

(e): [ ]

b) (6 pts) Now, you want to determine, for each dining hall, the number of menu items that fall into each affordability category. Fill in the blanks to define a DataFrame called `counts` containing this information. `counts` should have exactly three columns, named `"Dining Hall"`, `"Affordability"`, and `"Count"`.

```
counts = with_affordability.groupby(__(f)__).count().reset_index()
counts = counts.assign(Count=__(g)__).__(h)__
```

(f):

(g):

(h):

c) (5 pts) Suppose you determine that `"The Bistro"` is the dining hall with the most menu items in the `"Cheap"` category, so you will drag yourself there for every meal. Which of the following expressions must evaluate to the number of `"Cheap"` menu items available at `"The Bistro"`? **Select all that apply.**

- ☐ `counts.sort_values(by="Count", ascending=False).get("Count").iloc[0]`
- ☐ `counts.get("Count").max()`
- ☐ `(counts[counts.get("Affordability") == "Cheap"].sort_values(by="Count").get("Count").iloc[-1])`
- ☐ `counts[counts.get("Dining Hall") == "The Bistro"].get("Count").max()`
- ☐ `counts[(counts.get("Affordability") == "Cheap") & (counts.get("Dining Hall") == "The Bistro")].get("Count").iloc[0]`
- ☐ None of these.

Question 3 (21 pts)

Suppose we have access to another DataFrame called `orders`, containing all student dining hall orders from the past three years. `orders` includes the following columns, among others:

- `"Dining Hall"` (`str`): The dining hall where the order was placed.
- `"Start"` (`str`): The time the order was placed.
- `"End"` (`str`): The time the order was completed.

All times are expressed in 24-hour military time format (HH:MM). For example, `"13:15"` indicates 1:15 PM. All orders were completed on the same day as they were placed, and `"End"` is always after `"Start"`.

a) (5 pts) Fill in the blanks in the function `to_minutes` below. The function takes in a string representing a time in 24-hour military time format (HH:MM) and returns an int representing the number of minutes that have elapsed since midnight. Example behavior is given below.

```
>>> to_minutes("02:35")
155

>>> to_minutes("13:15")
795

def to_minutes(time):
    separate = time.__(a)__
    hour = __(b)__
    minute = __(c)__
    return __(d)__
```

(a): [ ] (b): [ ]

(c): [ ] (d): [ ]

b) (4 pts) Fill in the blanks below to add a new column called `"Wait"` to `orders`, which contains the number of minutes elapsed between when an order is placed and when it is completed. Note that the first two blanks both say `(e)` because they should be filled in with the same value.

```
start_min = orders.get("Start").__(e)__
end_min = orders.get("End").__(e)__
orders = orders.assign(Wait = __(f)__)
```

(e): [ ] (f): [ ]

c) (2 pts) You were told that `"End"` is always after `"Start"` in `orders`, but you want to verify if this is correct. Fill the blank below so that the result is `True` if `"End"` is indeed always after `"Start"`, and `False` otherwise.

```
(orders.get("Wait") >= 0).sum() == __(g)__
```

`(g)`:

d) (6 pts) Fill in the blanks below so that `ranked` evaluates to an array containing the names of the dining halls in `orders`, sorted in descending order by **average wait time**.

```
ranked = np.array(orders.__(h)__
                         .sort_values(by="Wait", ascending=False)
                         .__(i)__)
```

`(h)`:

`(i)`:

e) (4 pts) What would be the most appropriate type of data visualization to compare dining halls by average wait time?

◯ scatter plot

◯ line plot

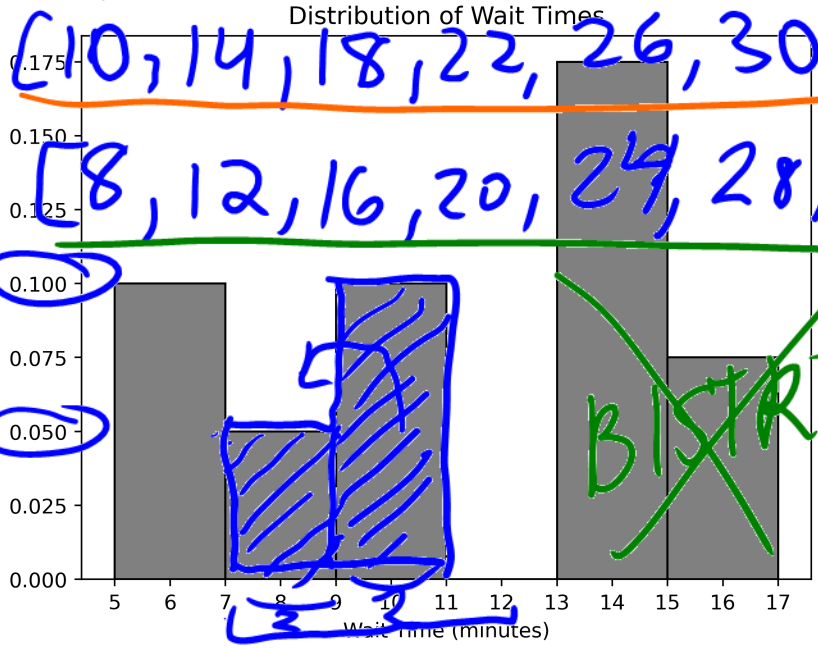
◯ bar chart

◯ histogram

◯ overlaid plot

Question 4 (14 pts)

Below is a density histogram displaying the distribution of wait times for orders from UCSD dining halls.



a) (6 pts) Fill in the blanks to define a variable `bins` that could have been used as the `bins` argument in the call to `plt.hist` that generated the histogram above.

```
bins = 0.5 * (2 + np.arange(__(a)__, __(b)__, __(c)__))
```

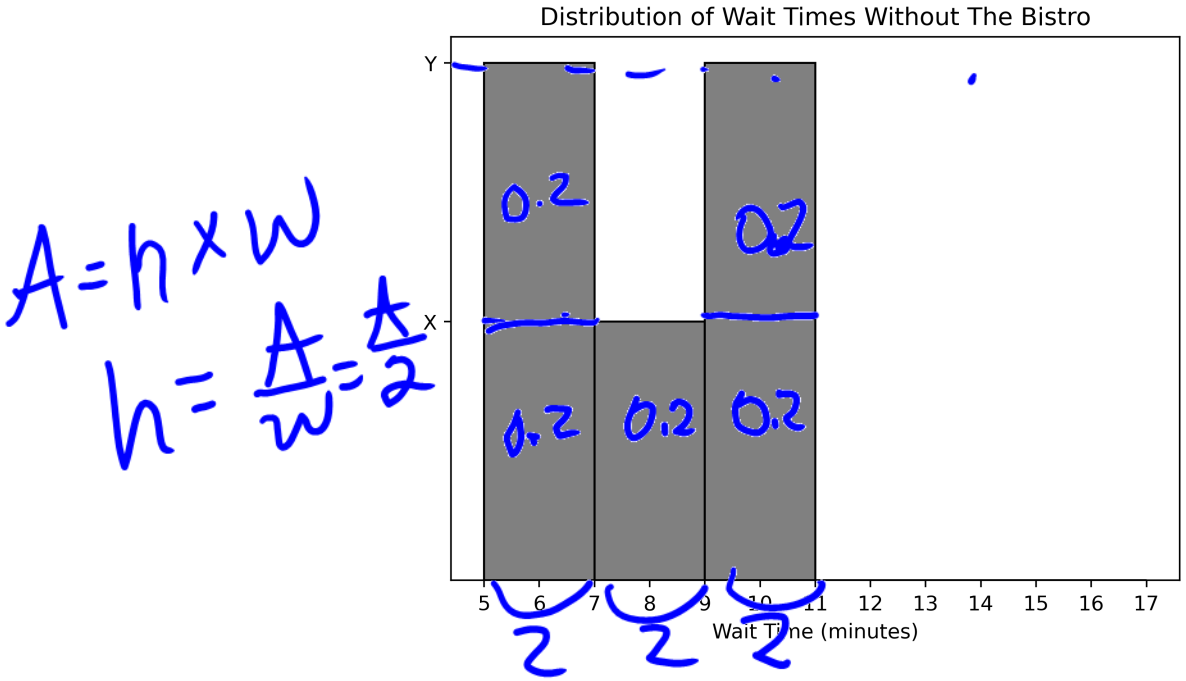
(a): 8 (b): 33 (c): 4

b) (4 pts) What proportion of orders took at least 7 minutes but less than 12 minutes to complete?

0.3

PID: _____

c) (4 pts) After some investigation, HDH discovers that every order that took longer than 12 minutes to complete was ordered from The Bistro. As a result, they fire all the employees at The Bistro and permanently close the dining hall. With this update, we generate a new density histogram displaying the distribution of wait times for orders from the other UCSD dining halls.



$A = h \times w$

$h = \frac{A}{w} = \frac{\Delta}{2}$

What are the values of X and Y along the y-axis of this histogram?

X: 0.1 Y: 0.2

Question 5 (12 pts)

It's the grand opening of UCSD's newest dining attraction: The Bread Basket! As a hardcore bread enthusiast, you celebrate by eating as much bread as possible. There are only a few menu items at The Bread Basket, shown with their costs in the table below:

Bread	Cost
Sourdough	2
Whole Wheat	3
Multigrain	4

Suppose you are given an array `eaten` containing the names of each type of bread you ate. For example, `eaten` could be defined as follows:

```
eaten = np.array(["Whole Wheat", "Sourdough", "Whole Wheat",
                  "Sourdough", "Sourdough"])
```

In this example, `eaten` represents five slices of bread that you ate, for a total cost of $12. Pricey!

In this problem, you'll calculate the total cost of your bread-eating extravaganza in various ways. In all cases, your code must calculate the total cost for an arbitrary `eaten` array, which might not be exactly the same as the example shown above.

a) (6 pts) One way to calculate the total cost of the bread in the `eaten` array is outlined below. Fill in the missing code.

```
breads = ["Sourdough", "Whole Wheat", "Multigrain"]
prices = [2, 3, 4]
total_cost = 0
for i in __(c)__:
    total_cost = (total_cost +
                  np.count_nonzero(eaten == __(d)__) * __(e)__)
```

(c): `[0,1,2]`

(d): `breads[i]`

(e): `prices[i]`

PID: _____

b) (6 pts) Another way to calculate the total cost of the bread in the `eaten` array uses the `merge` method. Fill in the missing code below.

```
available = bpd.DataFrame().assign(Type = ["Sourdough",
            "Whole Wheat", "Multigrain"]).assign(Cost = [2, 3, 4])
consumed = bpd.DataFrame().assign(Eaten = eaten)
combined = available.merge(consumed, left_on = __(f)__,
                           right_on = __(g)__)
total_cost = combined.__(h)__
```

(f): `'Type'`

(g): `'Eaten'`

(h): `get('Cost').sum()`

available

	Type	Cost
0	Sourdough	2
1	~~WW~~	3
2	Multi	4

consumed

	Eaten
0	Multi
1	WW
2	Multi
3	Sourdough
:	

might not be exactly this

Question 6 (16 pts)

At OceanView Terrace, you can make a custom pizza. There are 6 toppings available. Suppose you include each topping with a probability of 0.5, independently of all other toppings.

a) (4 pts) What is the probability you create a pizza with no toppings at all? Give your answer as a **fully simplified fraction**.

$\frac{1}{2} * \frac{1}{2} * \frac{1}{2} * \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = 1/64$

b) (8 pts) What is the probability that you create a pizza with exactly three toppings? Fill in the blanks in the code below so that `toppa` evaluates to an estimate of this probability.

```
tiptop = 0
num_trials = 10000

for i in np.arange(num_trials):
    pizza = np.random.choice([0, 1], __(a)__)
    if np.__(b)__ == 3:
        tiptop = __(c)__

toppa = __(d)__
```

(a): 6, replace = True

(b): sum(pizza) or count_nonzero(pizza)

(c): tiptop + 1

(d): tiptop / 10000

c) (4 pts) What is the meaning of `tiptop` after the code has finished running?

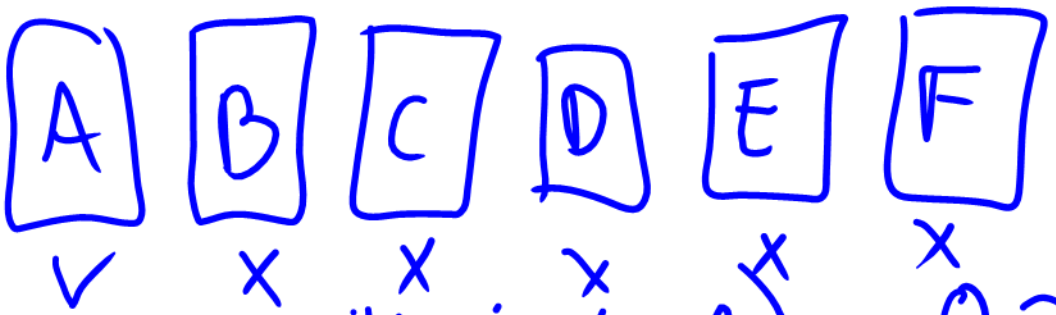
- ◯ The number of repetitions in our simulation.
- ◯ The size of our sample.
- ◯ The number of times we randomly selected three toppings.
- ◯ The proportion of times we randomly selected three toppings.
- ◯ The number of toppings we randomly selected.
- ◯ None of these answers is what `tiptop` represents.

prob including each topping is 0.2 (all independent)

just answered: P(no toppings)

next: P(exactly one topping)

$$0.2 * (0.8)^5 * 6$$



$$P(\text{pizza with just A}) = 0.2 * (0.8)^5$$

$$P(\text{pizza with just B}) = 0.2 * (0.8)^5$$

6 cases: add

$$0.2 * (0.8)^5 * 6$$