Midterm Exam - DSC 10, Summer Session I 2025

Full Name:	Solutions
PID:	A12345678

Instructions:

- This exam consists of 6 questions, worth a total of 49 points. You will have 80 minutes to complete this exam.
- Write your answers on a separate piece of a paper and draw a square around your final answer. Make sure your answers are legible. If we are unable to read your answer, we will not grade your answer.
- As this exam is conducted remotely, we expect you to have your camera on throughout the exam. We expect your hands to be visible during the exam. You are **prohibited** from using online resources or calculators during this exam. In particular, if we see evidence of generative AI usage, we will automatically give you a zero. A reference sheet for babypandas and other miscellaneous python functions/methods is included in this exam.
- After you finish your exam, you are expected to either take **clear** pictures of your exam responses or scan the responses to pdf and then upload them to Gradescope. **You will have up to 15 minutes after the end of the exam to upload your exams. Exams submitted after the 15 minute deadline will not be graded.**
- **Before beginning your exam:** Write the phrase "I agree to behave honestly and fairly during and after this exam" once at the top of your exam response paper and sign your name next to it. **Exams without this pledge will not be graded.**

Reference Sheet/Documentation

DataFrame Methods

- `df.groupby(col)` – group dataframe by column; followed by common aggregation methods: `mean`, `median`, `count`, `sum`, `min`, `max`.
- `df.plot(kind = ...)` – quick plots (`scatter`, `line`, `hist`, etc.).
- `df.get(col)` – single column `col` as a `Series`.
- `df.index[i]` – row label at position `i`.
- `df.take([i_1,...,i_k])` – select rows by integer position.
- `df[<condition>]` – query to keep only the rows in `df` that satisfy the condition(s).
- `df.assign(new_col = values)` – add or replace a column.
- `df.drop(columns = [...])` – drop one or more columns.
- `df.set_index(col)` – move a column to the index.
- `df.reset_index()` – move the index back to a column.
- `df.sort_values(by = col, ascending = ...)` – sort rows by a column.
- `df.get(col).apply(func)` - apply the function `func` to all values in the column `col`.
- `left.merge(right, left_on = ..., right_on = ...)` – merges the DataFrames `left` and `right` by the specified columns.

Miscellaneous Python Methods

- `s.upper()` – all uppercase.
- `s.lower()` – all lowercase.
- `s.split(<separator>)` – splits string into a list based on the separator.
- `arr[index]` - the element at position `index` in array `arr`. The first element is at position 0.
- `np.array(<lst>)` - casts list to arrays.
- `np.arange(<start>, <stop>, <step>)` - creates an array starting from `<start>`, ending at `<stop>`, with step size `<step>`.

PID: _____

Course Grades

The DataFrame `grades` contains the following information about assignment and exam scores for students in a previous offering of DSC 10:

- `"pid" (str)`: The PID of the student.
- `"assignment" (str)`: The name of the assignment (`"hw1"`, `"lab1"`, `"midterm"`, etc.). There are 6 homeworks, 7 labs, 1 midterm, and 1 final.
- `"category"`: The category of the assignment (either `"homework"`, `"lab"`, or `"exam"`).
- `"raw_score" (float)`: The student's raw score on the assignment, as a percentage from 0 to 100.
- `"due_date" (str)`: The due date of the assignment, formatted as `"YYYY-MM-DD"`.
- `"turn_in_date" (str)`: The date the student turned in the assignment, formatted as `"YYYY-MM-DD"`.

The first few rows of `grades` are shown below, though `grades` has many more rows than pictured. Note that the rows of `grades` are not sorted in any particular order.

	pid	assignment	category	raw_score	due_date	turn_in_date
0	A12345678	hw1	homework	95.50	2024-04-16	2024-04-16
1	A12345678	hw2	homework	87.00	2024-04-23	2024-04-25
2	A25412369	midterm	exam	75.70	2024-05-03	2024-05-03
3	A22512368	lab6	lab	100.00	2024-05-23	2024-05-24
4	A25412369	final	exam	92.28	2024-06-08	2024-06-08

Notes:

- Throughout this exam, assume we have already run `import babypandas as bpd` and `import numpy as np`.
- At any point, feel free to use functions and variables that you defined in earlier subparts of the same question.

Question 1 (3 points)

a) (2 points) Assign the variables `num_rows` and `num_cols` to the number of rows and columns in `grades`, respectively.

Solution: `num_rows = grades.shape[0], num_cols = grades.shape[1]`

b) (1 point) Explain what `grades.get("raw_score") + 2` does.

Solution: This line adds 2 to each value in the `"raw_score"` column of `grades`.

Question 2 (9 points)

a) (2 points) Fill in the blank below to add a new column, `"late"`, to `grades` that is `True` if the assignment was submitted late and `False` otherwise.

```
grades = grades.assign(late = __(a)__)
```

Hint: both `"0" < "1"` and `"02" < "03"` evaluate to `True`.

`(a)`: `grades.get("turn_in_date") > grades.get("due_date")`

For the remaining subparts, assume the `"late"` column was added to `grades` correctly in part (a).

b) (4 points) Write a line of code that assigns `hw1_late` to the total number of students who turned in Homework 1 late.

Solution:
```
hw1_late = grades[(grades.get("assignment") == "hw1")
& (grades.get("late"))].shape[0]
```

c) (3 points) Now, suppose we have run the following line of code:

```
hw = grades[grades.get("category") == "homework"]
```

Fill in the blanks below so that `submission_counts` evaluates to a DataFrame with a single column, `count`, which containing the total number of late and on-time homework submissions in `hw`.

```
hw_ct_df = hw.groupby(__(b)__).count()

submission_counts = hw_ct_df.assign(
    count = hw_ct_df.__(c)__(__(d)__)
).get(["count])
```

(b): `"late"` , (c): `get`

(d): any column other than `"late"`

Question 3 (13 points)

Suppose the course had a late policy stating that assignments lose 5% of their score for each day they are submitted late, up to a maximum of 2 days. In this section, we will adjust each student's raw score according to how many days late their submission was.

To simplify calculations, assume the following:

- All turn-in dates in `grades` are in the same month as their corresponding due dates.
- There are no late submissions for exams. That is, all students have the same due date and turn-in date for assignments in the `"exam"` category.

a) (3 points) To start, we need to extract the day from each `"due_date"` and `"turn_in_date"` in order to calculate how many days late each submission was.

Fill in the blanks in the function below to extract the day portion of each date as an integer. Example behavior is shown below.

```
>>> extract_day("2025-07-18")
18

def extract_day(date):
    return __(e)__
```

(e): `int(date.split("-")[2])`

b) (4 points) Assuming the `extract_day` function works correctly, complete the code below to create a new column, `"days_late"`, in `grades` that contains the number of days each assignment was submitted late. If a student turned in an assignment before the due date, the value in `"days_late"` should be negative.

```
grades = grades.assign(days_late = __(f)__)
```

(f): `grades.get("turn_in_date").apply(extract_day) - grades.get("due_date").apply(extract_day)`

c) (3 points) Recall that an assignment loses 5% of its score for each day it is submitted late, up to a maximum of 2 days. If an assignment is submitted more than 2 days late, the score is 0.

Complete the function `mult_factor`, which takes in the number of days late and returns the factor by which the original score should be multiplied. Submissions turned in early (negative `days_late`) receive full credit. Example behavior is shown below.

```
>>> mult_factor(2)
0.9
>>> mult_factor(3)
0
def mult_factor(days_late):
    if days_late < 0:
        return 1
    elif __(g)__:
        return __(h)__
    else:
        return __(i)__
```

(g): `days_late <= 2`, (h): `1 - 0.05 * days_late`

(i): `0`

d) (3 points) Finally, fill in the code below to compute each student's adjusted score for every assignment, based on the number of days late it was submitted, and store the result in a new `"score"` column in `grades`.

```
grades = grades.assign(score = __(j)__)
```

(j): `grades.get("raw_score") * grades.get("days_late").apply(mult_factor)`

PID: _____

Question 4 (11 points)

Now that we have determined the adjusted scores for all assignments, we want to calculate each student's overall course grade using the following category weights:

- `"homework"`: 30%
- `"lab"`: 20%
- `"exam"`: 50%

Recall that there are 6 homeworks, 7 labs, and 2 exams. Assume that each student has exactly 15 rows in `grades` – one for each assignment – even if they did not submit the assignment (in which case, the `"score"` for that assignment would be 0).

Also assume that all assignments are weighted equally within their respective categories.

a) (4 points) Complete the following code to add a new column, `"weight"`, to `grades`, containing the weight (as a proportion) that each **individual** assignment contributes to the overall course grade.

```
weights_arr = np.array([])
categories = grades.get("category")
for i in np.arange(__(k)__):
    if __(l)__:
        weight = 0.3 / 6
    elif __(m)__:
        weight = 0.2 / 7
    else:
        weight = 0.5 / 2
    weights_arr = __(n)__

grades = grades.assign(weight = weights_arr)
```

(k): `len(categories)`, (l): `categories.iloc[i] == "homework"`

(m): `categories.iloc[i] == "lab"`, (n): `np.append(weights_arr, weight)`

b) (3 points) Now that we've added a `"weight"` column representing the proportion each assignment contributes to the overall course grade, we can calculate each student's overall course grade. To do this, we'll use a **weighted average**, where each assignment's score is multiplied by its weight, and these values are summed across all assignments for each student.

For example, if a student scores 95 on a lab that's worth 0.0286 (i.e. 2.86% of the total grade), then that lab contributes $95 * 0.0286 = 2.717$ points to their overall course grade (out of 100 points).

Which of the following correctly sets `cg` to a DataFrame indexed by `"pid"` with a column called `"overall"` that contains the overall course grade as a percentage for each student? Select all that apply.

☐
```
cg = cg.groupby("pid").mean()
cg = cg.assign(overall = cg.get("score") * cg.get("weight")).get(["overall"])
```

☐
```
cg = cg.groupby("pid").sum()
cg = cg.assign(overall = cg.get("score") * cg.get("weight")).get(["overall"])
```

☐
```
cg = cg.assign(overall = cg.get("score") * cg.get("weight"))
cg = cg.groupby("pid").mean().get(["overall"])
```

■
```
cg = cg.assign(overall = cg.get("score") * cg.get("weight"))
cg = cg.groupby("pid").sum().get(["overall"])
```

☐
```
cg = cg.assign(overall = cg.get("score") * cg.get("weight"))
cg = cg.groupby(["pid", "assignment"]).mean().get(["overall"])
```

☐
```
cg = cg.assign(overall = cg.get("score") * cg.get("weight"))
cg = cg.groupby(["pid", "assignment"]).sum().get(["overall"])
```

c) (4 points) Assume that `cg` is assigned correctly in part (b). Suppose we have access to another DataFrame, `students`, with information about each student in `grades`. `students` is indexed by `"pid" (str)`, contains one column, `"name" (str)`, which stores each student's name.

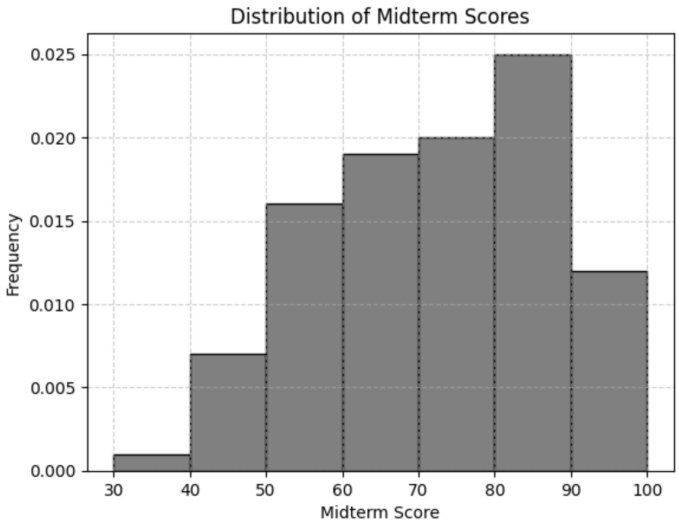
Write one line of code to find the name of the student who has the highest overall grade in the course. You may assume that the highest score is unique.

Solution:
```
cg.merge(students, left_index = True, right_index = True)
.sort_values(by = "overall", ascending = False).get("name").iloc[0]
```

PID: _____

Question 5 (7 points)

The density histogram below shows the distribution of midterm scores in `grades`.



a) (1 point) What is the total area of histogram?

1

b) (3 points) Suppose we randomly select one student from the class. What is the probability that they scored between 70% and 90% on the midterm? Give your answer as a fraction or decimal, or write "not enough information" if the answer cannot be determined from the plot. You may leave your answer unsimplified.

$$[70, 80) : (10 * 0.02) = 0.2,$$
$$[80, 90) : (10 * 0.025) = 0.25$$
$$=> \ [70, 90) \ :$$
$$0.2 + 0.25 \ = \ 0.45$$

c) (2 points) Imagine we changed our histogram to have just one bin, from 30% to 100%. What would the height of this bin be? Give your answer as a fraction or decimal, or write "not enough information" if the answer cannot be determined from the plot. You may leave your answer unsimplified.

$$area \ = \ w * h \ =>$$
$$1 \ = \ 70 * h \ => \ \frac{1}{70}$$

d) (1 point) What kind of plot should be used to visualize the average score for each assignments.

◯ scatter plot ◯ line plot ◯ histogram ● bar plot ◯ none of the above

Question 6 (6 points)

In addition to DSC 10, suppose that each student in the `grades` DataFrame is enrolled in **exactly one** elective course from the table below. The table shows the probability that a randomly selected student is enrolled in each elective.

Department	**Name**	**Probability**
Film Studies	Indigenous Cinema	0.12
Film Studies	Queer Film Theory	0.10
Film Studies	Film Noir	0.16
Mathematics	Real Analysis	0.17
Mathematics	Abstract Algebra	0.10
Religion	Comparative Religions	0.05
Religion	Introduction to Religion	0.15
Religion	Religions of Antiquity	0.15

Assume that each student's elective enrollment is independent of others. For the following parts, you may leave your answer unsimplified.

a) (2 points) Suppose you randomly select a student. Given that the student is taking a Religion course as their elective, what is the probability that they are taking *Comparative Religions*?

$$\frac{0.05}{0.35} = \frac{1}{7}$$

b) (2 points) Suppose you randomly select two students. What is the probability that they are both taking *Film Noir* as their elective?

$$(0.16)^2$$

c) (2 points) Suppose you randomly select three students. What is the probability that at least one of them is taking a Mathematics course as their elective?

$$1 - P(\text{none taking math}) = 1 - (1 - (0.27))^3$$