Final Exam - DSC 10, Winter 2024

Full Name:	
PID:	
Seat you are in:	

Instructions:

- This exam consists of 16 questions, worth a total of 175 points.
  ★ The two trickiest question subparts (in Janine's opinion) are marked with a star.
- 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 pts.

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

  ☐ A square box means you should **select all that apply**.
- Aside from the provided Reference Sheet, you may not refer to any other resources or technology during the exam (no notes, 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.

Important: Before proceeding, make sure to read the page called *Olympic Medalists*.

Question 1 (3 pts)

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

◯ `"Name"` ◯ `"Sport"` ◯ `"Team"` ◯ None of these.

Question 2 (9 pts)

Frank X. Kugler has Olympic medals in three sports (wrestling, weightlifting, and tug of war), which is more sports than any other Olympic medalist. Furthermore, his medals for all three of these sports are included in the `olympians` DataFrame. Fill in the blanks below so that the expression below evaluates to `"Frank X. Kugler"`.

```
(olympians.groupby(__(a)__).__(b)__
          .reset_index()
          .groupby(__(c)__).__(d)__
          .sort_values(by="Age", ascending=False)
          .index[0])
```

a: _____ , b: _____

c: _____ , d: _____

Question 3 (9 pts)

In `olympians`, `"Weight"` is measured in kilograms. There are 2.2 pounds in 1 kilogram. If we converted `"Weight"` to pounds instead, which of the following quantities would increase?

☐ The mean of the `"Weight"` distribution.

☐ The standard deviation of the `"Weight"` distribution.

☐ The proportion of `"Weight"` values within 3 standard deviations of the mean.

☐ The correlation between `"Height"` and `"Weight"`.

☐ The slope of the regression line predicting `"Weight"` from `"Height"`.

☐ The slope of the regression line predicting `"Height"` from `"Weight"`.

PID: _____

Question 4 (12 pts)

The Olympics are held every two years, in even-numbered years, alternating between the Summer Olympics and Winter Olympics. Summer Olympics are held in years that are a **multiple of 4** (such as 2024), and Winter Olympics are held in years that are not a multiple of 4 (such as 2022 or 2026).

We want to add a column to `olympics` that contains either `"Winter"` or `"Summer"` by applying a function called `season` as follows:

```
olympians.assign(Season=olympians.get("Year").apply(season))
```

Which of the following definitions of `season` is correct? Select all that apply.

Notes:

- We'll ignore the fact that the 2020 Olympics were rescheduled to 2021 due to Covid.
- Recall that the `%` operator in Python gives the remainder upon division. For example, `8 % 3` evaluates to `2`.

Way 1:

```
def season(year):
    if year % 4 == 0:
        return "Summer"
    return "Winter"
```

Way 2:

```
def season(year):
    return "Winter"
    if year % 4 == 0:
        return "Summer"
```

Way 3:

```
def season(year):
    if year % 2 == 0:
        return "Winter"
    return "Summer"
```

Way 4:

```
def season(year):
    if int(year / 4) != year / 4:
        return "Winter"
    return "Summer"
```

☐ Way 1 ☐ Way 2 ☐ Way 3 ☐ Way 4

Question 5 (16 pts)

In figure skating, skaters move around an ice rink performing a series of skills, such as jumps and spins. Ylesia has been training for the Olympics, and she has a set routine that she plans to perform.

Let's say that Ylesia performs a skill successfully if she does not fall during that skill. Each skill comes with its own probability of success, as some skills are harder and some are easier. Suppose that the probabilities of success for each skill in Ylesia's Olympic routine are stored in an array called `skill_success`.

For example, if Ylesia's Olympic routine happened to only contain three skills, `skill_success` might be the array with values 0.92, 0.84, 0.92. However, her routine can contain any number of skills.

a) (5 pts) Ylesia wants to simulate one Olympic routine to see how many times she might fall. Fill in the function `count_falls` below, which takes as input an array `skill_success` and returns as output the number of times Ylesia falls during her Olympic routine.

```
def count_falls(skill_success):
    falls = 0
    for p in skill_success:
        result = np.random.multinomial(1, __(a)__)
        falls = __(b)__
    return falls
```

a:

b:

b) (5 pts) Fill in the blanks below so that `prob_no_falls` evaluates to the exact probability of Ylesia performing her entire routine without falling.

```
prob_no_falls = __(a)__
for p in skill_success:
    prob_no_falls = __(b)__
prob_no_falls
```

a:

b:

c) (6 pts) Fill the blanks below so that `approx_prob_no_falls` evaluates to an estimate of the probability that Ylesia performs her entire routine without falling, based on 10,000 trials. Feel free to use the function you defined in part (a) as part of your solution.

```
results = np.array([])
for i in np.arange(10000):
    results = np.append(results, __(a)__)
approx_prob_no_falls = __(b)__
approx_prob_no_falls
```

a:

b:

Question 6 (9 pts)

Suppose we sample **400 rows** of `olympians` at random without replacement, then generate a 95% CLT-based confidence interval for the mean age of Olympic medalists based on this sample.

a) (3 pts) The CLT is stated for samples drawn with replacement, but in practice, we can often use it for samples drawn without replacement. What is it about this situation that makes it reasonable to still use the CLT despite the sample being drawn without replacement?

◯ The sample is much smaller than the population.

◯ The statistic is the sample mean.

◯ The CLT is less computational than bootstrapping, so we don't need to sample with replacement like we would for bootstrapping.

◯ The population is normally distributed.

◯ The sample standard deviation is similar to the population standard deviation.

b) (6 pts) Suppose our 95% CLT-based confidence interval for the mean age of Olympic medalists is $[24.9, 26.1]$. What was the mean and standard deviation of ages in our sample?

mean = _____, standard deviation= _____

Question 7 (14 pts)

In our sample, we have data on **1210 medals** for the sport of gymnastics. Of these, **126** were awarded to American gymnasts, **119** were awarded to Romanian gymnasts, and the remaining **965** were awarded to gymnasts from other nations.

We want to do a hypothesis test with the following hypotheses.

Null: American and Romanian gymnasts win an equal share of Olympic gymnastics medals.

Alternative: American gymnasts win more Olympic gymnastics medals than Romanian gymnasts.

a) (3 pts) Which test statistic could we use to test these hypotheses?

◯ total variation distance between the distribution of medals by country and the uniform distribution

◯ proportion of medals won by American gymnasts

◯ difference in the number of medals won by American gymnasts and the number of medals won by Romanian gymnasts

◯ absolute difference in proportion of medals won by American gymnasts and proportion of medals won by Romanian gymnasts

b) (8 pts) Below are four different ways of testing these hypotheses. In each case, fill in the calculation of the observed statistic in the variable `observed`, such that `p_val` represents the p-value of the hypothesis test.

Way 1:

```
many_stats = np.array([])
for i in np.arange(10000):
    result = np.random.multinomial(245, [0.5, 0.5]) / 245
    many_stats = np.append(many_stats, result[0] - result[1])
observed = __(a)__
p_val = np.count_nonzero(many_stats >= observed)/len(many_stats)
```

a:

Way 2:

```
many_stats = np.array([])
for i in np.arange(10000):
    result = np.random.multinomial(245, [0.5, 0.5]) / 245
    many_stats = np.append(many_stats, result[0] - result[1])
observed = __(b)__
p_val = np.count_nonzero(many_stats <= observed)/len(many_stats)
```

b:

Way 3:

```
many_stats = np.array([])
for i in np.arange(10000):
    result = np.random.multinomial(245, [0.5, 0.5]) / 245
    many_stats = np.append(many_stats, result[0])
observed = __(c)__
p_val = np.count_nonzero(many_stats >= observed)/len(many_stats)
```

c:

Way 4:

```
many_stats = np.array([])
for i in np.arange(10000):
    result = np.random.multinomial(245, [0.5, 0.5]) / 245
    many_stats = np.append(many_stats, result[0])
observed = __(d)__
p_val = np.count_nonzero(many_stats <= observed)/len(many_stats)
```

d:

c) (3 pts) The four p-values calculated in Ways 1 through 4 are:

◯ exactly the same
◯ similar, but not necessarily exactly the same
◯ not necessarily similar

Question 8 (9 pts)

a) (3 pts) In Olympic hockey, the number of goals a team scores is linearly associated with the number of shots they attempt. In addition, the number of goals a team scores has a mean of 10 and a standard deviation of 5, whereas the number of attempted shots has a mean of 30 and a standard deviation of 10.

Suppose the regression line to predict the number of goals based on the number of shots predicts that for a game with 20 attempted shots, 6 goals will be scored. What is the correlation between the number of goals and the number of attempted shots? Give your answer as an exact fraction or decimal.

b) (6 pts) In Olympic baseball, the number of runs a team scores is linearly associated with the number of hits the team gets. The number of runs a team scores has a mean of 8 and a standard deviation of 4, while the number of hits has a mean of 24 and a standard deviation of 6. Consider the regression line that predicts the number of runs scored based on the number of hits.

(i) (3 pts) What is the maximum possible predicted number of runs for a team that gets 27 hits?

(ii) (3 pts) What is the correlation coefficient in the case where the predicted number of runs for a team with 25 hits is as large as possible?

PID: _____

Question 9 (12 pts)

In 2024, the Olympics will include breaking (also known as breakdancing) for the first time. The breaking competition will include **16 athletes**, who will compete in a single-elimination tournament.

In the first round, all 16 athletes will compete against an opponent in a face-to-face "battle". The 8 winners, as determined by the judges, will move on to the next round. Elimination continues until the final round contains just 2 competitors, and the winner of this final battle wins the tournament.

The table below shows how many competitors participate in each round:

Round	Competitors
1	16
2	8
3	4
4	2

After the 2024 Olympics, suppose we make a DataFrame called `breaking` containing information about the performance of each athlete during each round. `breaking` will have one row for each athlete's performance in each round that they participated. Therefore, there will be $16 + 8 + 4 + 2 = \mathbf{30}$ **rows** in `breaking`.

In the `"name"` column of `breaking`, we will record the athlete's name (which we'll assume to be unique), and in the other columns we'll record the judges' scores in the categories on which the athletes will be judged (creativity, personality, technique, variety, performativity, and musicality).

a) (3 pts) How many rows of `breaking` correspond to the winner of the tournament? Give your answer as an integer.

b) (3 pts) How many athletes' names appear exactly twice in the `"name"` column of `breaking`? Give your answer as an integer.

c) (3 pts) ★ If we merge `breaking` with itself on the `"name"` column, how many rows will the resulting DataFrame have? Give your answer as an integer.

Hint: Parts (a) and (b) of this question are relevant to part (c).

d) (3 pts) Recall that the number of competitors in each round is $16, 8, 4, 2$. Write one line of code that evaluates to the array `np.array([16, 8, 4, 2])`. You **must use** `np.arange` in your solution, and you **may not use** `np.array` or the DataFrame `breaking`.

Question 10 (20 pts)

We want to use the sample of data in `olympians` to estimate the mean age of Olympic beach volleyball players.

a) (3 pts) Which of the following distributions must be normally distributed in order to use the Central Limit Theorem to estimate this parameter?

◯ The age distribution of all Olympic athletes.
◯ The age distribution of Olympic beach volleyball players.
◯ The age distribution of Olympic beach volleyball in our sample.
◯ None of the above.

b) (10 pts) Next we want to use bootstrapping to estimate this parameter. Which of the following code implementations correctly generates an array called `sample_means` containing $10,000$ bootstrapped sample means?

Way 1:

```
sample_means = np.array([])
for i in np.arange(10000):
    bv = olympians[olympians.get("Sport") == "Beach Volleyball"]
    one_mean = (bv.sample(bv.shape[0], replace=True)
                  .get("Age").mean())
    sample_means = np.append(sample_means, one_mean)
```

Way 2:

```
sample_means = np.array([])
for i in np.arange(10000):
    bv = olympians[olympians.get("Sport") == "Beach Volleyball"]
    one_mean = (olympians.sample(olympians.shape[0], replace=True)
                         .get("Age").mean())
    sample_means = np.append(sample_means, one_mean)
```

Way 3:

```
sample_means = np.array([])
for i in np.arange(10000):
    resample = olympians.sample(olympians.shape[0], replace=True)
    bv = resample[resample.get("Sport") == "Beach Volleyball"]
    one_mean = bv.get("Age").mean()
    sample_means = np.append(sample_means, one_mean)
```

Way 4:

```
sample_means = np.array([])
bv = olympians[olympians.get("Sport") == "Beach Volleyball"]
for i in np.arange(10000):
    one_mean = (bv.sample(bv.shape[0], replace=True)
                  .get("Age").mean())
    sample_means = np.append(sample_means, one_mean)
```

Way 5:

```
sample_means = np.array([])
bv = olympians[olympians.get("Sport") == "Beach Volleyball"]
one_mean = (bv.sample(bv.shape[0], replace=True)
              .get("Age").mean())
for i in np.arange(10000):
    sample_means = np.append(sample_means, one_mean)
```

☐ Way 1 ☐ Way 2 ☐ Way 3 ☐ Way 4 ☐ Way 5

c) (4 pts) For most of the answer choices in part (b), we do not have enough information to predict how the standard deviation of `sample_means` would come out. There is one answer choice, however, where we do have enough information to compute the standard deviation of `sample_means`. Which answer choice is this, and what is the standard deviation of `sample_means` for this answer choice?

◯ Way 1 ◯ Way 2 ◯ Way 3 ◯ Way 4 ◯ Way 5

standard deviation = ☐

d) (3 pts) There are **68 rows** of `olympians` corresponding to beach volleyball players. Assume that in part (b), we correctly generated an array called `sample_means` containing 10,000 bootstrapped sample mean ages based on this original sample of 68 ages. The standard deviation of the original sample of 68 ages is approximately how many times larger than the standard deviation of `sample_means`? Give your answer to the nearest integer.

☐

Question 11 (12 pts)

Aladár Gerevich is a Hungarian fencer who is one of only two men to win Olympic medals 28 years apart. He earned 10 Olympic medals in total throughout his career: 7 gold, 1 silver, and 2 bronze. The table below shows the distribution of medal types for Aladár Gerevich, as well as a few other athletes who also earned 10 Olympic medals.

Athlete	**Gold**	**Silver**	**Bronze**
Aladár Gerevich	0.7	0.1	0.2
Katie Ledecky	0.7	0.3	0
Alexander Dityatin	0.3	0.6	0.1
Franziska van Almsick	0	0.4	0.6

a) (3 pts) Which type of data visualization is most appropriate to compare two athlete's medal distributions?

◯ overlaid histogram
◯ overlaid bar chart
◯ overlaid line plot

b) (3 pts) Among the other athletes in the table above, whose medal distribution has the largest total variation distance (TVD) to Aladár Gerevich's distribution?

◯ Katie Ledecky
◯ Alexander Dityatin
◯ Franziska van Almsick

c) (3 pts) Suppose Pallavi earns 10 Olympic medals in such a way that the TVD between Pallavi's medal distribution and Aladár Gerevich's medal distribution is as large as possible. What is Pallavi's medal distribution?

Athlete	**Gold**	**Silver**	**Bronze**
Aladár Gerevich	0.7	0.1	0.2
Pallavi	x	y	z

$x =$ ____ $y =$ ____ $z =$ ____

d) (3 pts) More generally, suppose `medal_dist` is an array of length three representing an athlete's medal distribution. Which of the following expressions gives the maximum possible TVD between `medal_dist` and any other distribution?

◯ `medal_dist.max()`
◯ `medal_dist.min()`
◯ `1 - medal_dist.max()`
◯ `1 - medal_dist.min()`
◯ `np.abs(1 - medal_dist).sum()/2`

PID: _____

Question 12 (6 pts)

Consider the DataFrame `olympians.drop(columns=["Medal", "Year", "Count"])`.

a) (2 pts) State a question we could answer using the data in this DataFrame and a permutation test.

b) (4 pts) State the null and alternative hypotheses for this permutation test.

Question 13 (17 pts)

In our sample, we have data on **163 medals** for the sport of table tennis. Based on our data, China seems to really dominate this sport, earning **81** of these medals.

That's nearly half of the medals for just one country! We want to do a hypothesis test with the following hypotheses to see if this pattern is true in general, or just happens to be true in our sample.

Null: China wins half of Olympic table tennis medals.

Alternative: China does not win half of Olympic table tennis medals.

a) (3 pts) Why can these hypotheses be tested by constructing a confidence interval?

◯ Since proportions are means, so we can use the CLT.

◯ Since the test aims to determine whether a parameter is equal to a fixed value.

◯ Since we need to get a sense of how other samples would come out by bootstrapping.

◯ Since the test aims to determine if our sample came from a known population distribution.

b) (6 pts) Suppose we construct a 95% bootstrapped CI for the proportion of Olympic table tennis medals won by China. Select all true statements.

☐ The true proportion of Olympic table tennis medals won by China has a 95% chance of falling within the bounds of our interval.

☐ If we resampled our original sample and calculated the proportion of Olympic table tennis medals won by China in our resample, there is approximately a 95% chance our interval would contain this number.

☐ 95% of Olympic table tennis medals are won by China.

☐ None of the above.

c) (2 pts) True or False: In this scenario, it would also be appropriate to create a 95% CLT-based confidence interval.

◯ True ◯ False

d) (2 pts) True or False: If our 95% bootstrapped CI came out to be $[0.479, 0.518]$, we would reject the null hypothesis at the 0.05 significance level.

◯ True ◯ False

e) (2 pts) True or False: If we instead chose to test these hypotheses at the 0.01 significance level, the confidence interval we'd create would be wider.

◯ True ◯ False

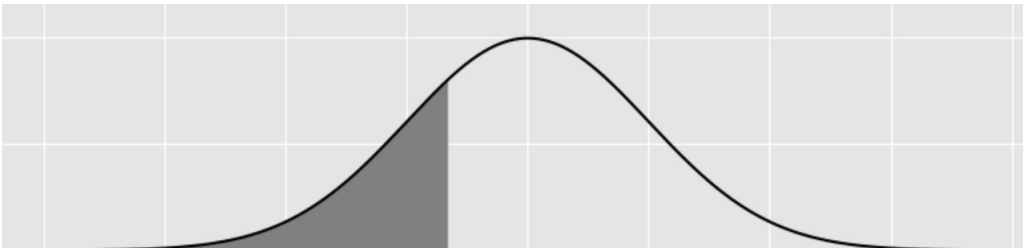
f) (2 pts) True or False: If we instead chose to test these hypotheses at the 0.01 significance level, we would be more likely to conclude a statistically significant result.

◯ True ◯ False

PID: _____

Question 14 (9 pts)

a) (3 pts) Suppose that in Olympic ski jumping, ski jumpers jump off of a ramp that's shaped like a portion of a normal curve. Drawn from left to right, a full normal curve has an inflection point on the ascent, then a peak, then another inflection point on the descent. A ski jump ramp stops at the point that is one third of the way between the inflection point on the ascent and the peak, measured horizontally. Below is an example ski jump ramp, along with the normal curve that generated it.



Fill in the blank below so that the expression evaluates to the area of a ski jump ramp, if the area under the normal curve that generated it is 1.

```
from scipy import stats
stats.norm.cdf(______)
```

What goes in the blank?

b) (3 pts) Suppose that in Olympic downhill skiing, skiers compete on mountains shaped like normal distributions with mean 50 and standard deviation 8. skiers start at the peak and ski down the right side of the mountain, so their x-coordinate increases.

Keenan is an Olympic downhill skier, but he's only been able to practice on a mountain shaped like a normal distribution with mean 65 and standard deviation 12. In his practice, Keenan always crouches down low when he reaches the point where his x-coordinate is 92, which helps him ski faster. When he competes at the Olympics, at what x-coordinate should he crouch down low, corresponding to the same relative location on the mountain?

c) (3 pts) Aaron is another Olympic downhill skier. When he competes on the normal curve mountain with mean 50 and standard deviation 8, he crouches down low when his x-coordinate is 54. If the total area of the mountain is 1, approximately how much of the mountain's area is ahead of Aaron at the moment he crouches down low?

◯ 0.1 ◯ 0.2 ◯ 0.3 ◯ 0.4

Question 15 (9 pts)

Birgit Fischer-Schmidt is a German canoe paddler who set many records, including being the only woman to win Olympic medals 24 years apart.

Below is a DataFrame with information about all **12 Olympic medals** she has won. There are only 10 rows but 12 medals represented, because there are some years in which she won more than one medal of the same type.

	Name	Sex	Age	Height	Weight	Sport	Team	Medal	Year	Count
3022	Birgit Fischer-Schmidt	F	18	172.0	69.0	Canoeing	East Germany	Gold	1980	1
3023	Birgit Fischer-Schmidt	F	26	172.0	69.0	Canoeing	East Germany	Gold	1988	2
3028	Birgit Fischer-Schmidt	F	26	172.0	69.0	Canoeing	East Germany	Silver	1988	1
3024	Birgit Fischer-Schmidt	F	30	172.0	69.0	Canoeing	Germany	Gold	1992	1
3029	Birgit Fischer-Schmidt	F	30	172.0	69.0	Canoeing	Germany	Silver	1992	1
3025	Birgit Fischer-Schmidt	F	34	172.0	69.0	Canoeing	Germany	Gold	1996	1
3030	Birgit Fischer-Schmidt	F	34	172.0	69.0	Canoeing	Germany	Silver	1996	1
3026	Birgit Fischer-Schmidt	F	38	172.0	69.0	Canoeing	Germany	Gold	2000	2
3027	Birgit Fischer-Schmidt	F	42	172.0	69.0	Canoeing	Germany	Gold	2004	1
3031	Birgit Fischer-Schmidt	F	42	172.0	69.0	Canoeing	Germany	Silver	2004	1

a) (3 pts) Suppose we randomly select one of Birgit's Olympic medals, and see that it is a gold medal. What is the probability that the medal was earned in the year 2000? Give your answer as a fully simplified fraction.

b) (3 pts) Suppose we randomly select one of Birgit's Olympic medals. What is the probability it is gold or earned while representing East Germany? Give your answer as a fully simplified fraction.

c) (3 pts) Suppose we randomly select two of Birgit's Olympic medals, without replacement. What is the probability both were earned in 1988? Give your answer as a fully simplified fraction.

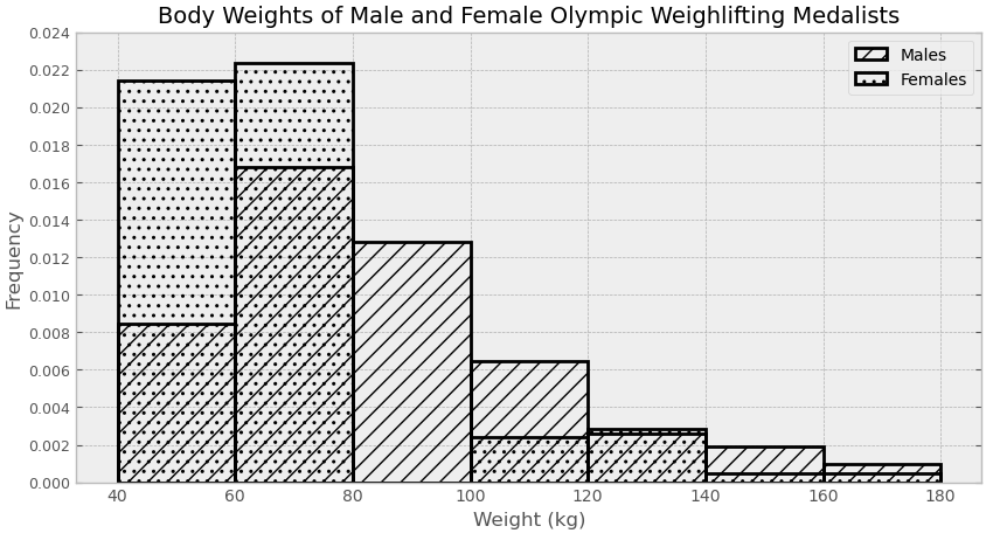
PID: _____

Question 16 (9 pts)

Suppose `males` is a DataFrame of all male Olympic weightlifting medalists with a column called `"Weight"` containing their body weight. Similarly, `females` is a DataFrame of all female Olympic weightlifting medalists. It also has a `"Weight"` column with body weights.

The `males` DataFrame has **425 rows** and the `females` DataFrame has **105 rows**, since women's weightlifting became an Olympic sport much later than men's.

Below, density histograms of the distributions of `"Weight"` in `males` and `females` are shown on the same axes:



a) (3 pts) Estimate the number of males included in the third bin (from 80 to 100). Give your answer as an integer, rounded to the nearest multiple of 10.

b) (3 pts) Using the `males` DataFrame, write one line of code that evaluates to the exact number of males included in the third bin (from 80 to 100).

c) (3 pts) ★ Among Olympic weightlifting medalists who weigh less than 60 kilograms, what proportion are male?

◯ less than 0.25
◯ between 0.25 and 0.5
◯ between 0.5 and 0.75
◯ more than 0.75

Feel free to draw us a picture or tell us about your fondest memory from DSC 10 this quarter.

Before turning in your exam, please make sure that your PID is on every page.

Congratulations on finishing DSC 10!

Olympic Medalists

The Olympic Games are the world's leading international sporting event, dating back to ancient Greece. Today, we'll explore data on modern Olympic medalists. Each row in the DataFrame `olympians` corresponds to a type of medal earned by one Olympic athlete in one year. The columns of `olympians` are as follows:

- `"Name" (str)`: The name of the athlete. Unique for each athlete.
- `"Sex" (str)`: The sex of the athlete, denoted by `"M"` or `"F"`.
- `"Age" (int)`: The age of the athlete at the time of the Olympics.
- `"Height" (float)`: The height of the athlete in centimeters.
- `"Weight" (float)`: The weight of the athlete in kilograms.
- `"Sport" (str)`: The sport in which the athlete competed.
- `"Team" (str)`: The team or country the athlete represented.
- `"Medal" (str)`: The type of medal won (`"Gold"`, `"Silver"`, or `"Bronze"`).
- `"Year" (int)`: The year of the Olympic Games.
- `"Count" (int)`: The number of medals of this type earned by this athlete in this year.

The first few rows of `olympians` are shown below, though `olympians` has many more rows than pictured. The data in `olympians` is only a sample from the much larger population of **all** Olympic medalists.

	Name	Sex	Age	Height	Weight	Sport	Team	Medal	Year	Count
0	Michael Fred Phelps, II	M	23	193.0	91.0	Swimming	United States	Gold	2008	8
1	Michael Fred Phelps, II	M	19	193.0	91.0	Swimming	United States	Gold	2004	6
2	Motiullah Khan	M	26	181.0	69.0	Hockey	Pakistan	Silver	1964	1
3	Park Seong-Hyeon	F	21	170.0	72.0	Archery	South Korea	Gold	2004	2
4	Meryl Elizabeth Davis	F	27	160.0	49.0	Figure Skating	United States	Bronze	2014	1
5	Witold Woyda	M	29	168.0	63.0	Fencing	Poland	Bronze	1968	1

Throughout this exam, assume that we have already run `import babypandas as bpd` and `import numpy as np`.

DSC 10 Reference Sheet

Below, `df` is a DataFrame, `ser` is a Series, `arr` is an array, `babypandas` has been imported as `bpd`, and `numpy` has been imported as `np`.

Building and Organizing DataFrames

Each function/method below creates a new DataFrame. Remember to save it!

`bpd.DataFrame()`
: Creates an empty DataFrame.

`bpd.read_csv(path_to_file)`
: Creates a DataFrame by reading from a CSV file.

`df.assign(name_of_column=column_data)`
: Adds/replaces a column. `name_of_column` should not have quotes or spaces.

`df.drop(columns=column_name or [col_1_name, ..., col_k_name])`
: Drops a single column, or every column in a list of column names.

`df.set_index(column_name)`
: Moves a column to the index.

`df.reset_index()`
: Moves the index to a column.

`df.sort_values(by=column_name, ascending=True)`
: Sorts the entire DataFrame in ascending order by the values in a column. `ascending` can be omitted, as its default value is `True`.

`left.merge(right, left_on=left_column, right_on=right_column)`
: Merges the DataFrames `left` and `right` by the specified columns. Can use `on` instead of `left_on` and `right_on` if the column names are the same.

`left.merge(right, left_index=True, right_on=right_column)`
: Merges using `left`'s index instead of a column. Can also be done with `right_index=True`.

Arrays and NumPy

`arr[index]`
: The element at position `index` in the array `arr`. The first element is `arr[0]`.

`np.append(arr, value)`
: A copy of `arr` with `value` appended to the end. This does not change `arr` unless you store the result in `arr`.

`np.count_nonzero(arr)`
: The number of non-zero entries in `arr`. `True` counts as 1, `False` counts as 0.

`np.arange(start, stop, step)`
: An array of numbers starting with `start`, increasing/decreasing in increments of `step`, and stopping before (excluding) `stop`. If `start` or `step` are omitted, the default values are 0 and 1, respectively.

`np.percentile(arr, p)`
: The pth percentile of the numbers in `arr`.

These also work if `arr` is a Series, list, or some other type of sequence.

Plotting

`df.plot(kind=kind, x=col_x, y=col_y)`
: Draws a plot. `kind` may be `'scatter'`, `'line'`, `'bar'`, or `'barh'`. If `x` is omitted, the index is used. For most kinds of plots, if `y` is omitted, all columns are plotted on shared axes.

`df.plot(kind='hist', y=data_col, bins=the_bins, density=True)`
: Plots a density histogram of the numerical data in `data_col`. `the_bins` can be a number of bins, or a sequence specifying bin endpoints. Scaled so that the total area is 1.

Accessing Data

`df.shape[0]` and `df.shape[1]`
: The number of rows and the number of columns, respectively.

`df.get(column_name)`
: One column, as a Series.

`df.get([col_1_name, ..., col_k_name])`
: Several columns, as a DataFrame.

`ser.loc[label]`
: An element of `ser`, accessed by its row label. Often `ser` comes from `df.get(column_name)`.

`ser.iloc[position]`
: An element of `ser`, accessed by its integer position. Positions start at 0. Often `ser` comes from `df.get(column_name)`.

`df.index[position]`
: An element in the index, accessed by its integer position. Positions start at 0.

`df.take([position_1, ..., position_k])`
: A DataFrame of specific rows, accessed by integer position. Often used with `np.arange`.

`df[bool_arr]`
: A DataFrame of specific rows, usually where some condition is satisfied. See: Querying.

Series Methods

Series have the following methods:

`.count()`, `.max()`, `.min()`, `.sum()`, `.mean()`, `.median()`, `.unique()`

Querying

Querying (also called filtering or Boolean indexing) selects a subset of a DataFrame's rows. We need a sequence of Boolean values, `condition`, with length equal to the number of rows of the DataFrame. The expression `df[condition]` results in a DataFrame containing only those rows whose corresponding element in `condition` is `True`.

Boolean sequences are easily constructed by comparing an array, index, or Series to a value using the comparison operators: `==`, `!=`, `<`, `>`, `<=`, `>=`.

`(bool_arr_1) & (bool_arr_2)`
: The "and" of two Boolean arrays. Contains `True` where both Boolean arrays contain `True`, otherwise contains `False`.

`(bool_arr_1) | (bool_arr_2)`
: The "or" of two Boolean arrays. Contains `True` where at least one of the Boolean arrays contains `True`, otherwise contains `False`.

`df[df.get(column_name) > 42]`
: Retrieves all rows for which the given column is bigger than 42.

`df[(df.get(column_name) > 42) & (df.get(column_name) < 100)]`
: Retrieves all rows for which the given column is between 42 and 100. Parentheses are important!

`df[df.get(column_name).str.contains(pattern)]`
: Retrieves all rows for which the given column contains the string `pattern`.

`df[df.index > 2]`
: Retrieves all rows for which the index is greater than 2.

Grouping

Use `df.groupby(column_name)`, followed by one of these aggregation methods:

`.mean()`, `.median()`, `.count()`, `.max()`, `.min()`, `.sum()`

This results in a DataFrame indexed by the group names. Only those columns whose data type permits the selected aggregation method are kept – for instance, `.sum()` will drop columns containing strings.

`df.groupby([col_1_name, ..., col_k_name])` creates subgroups, first grouping by `col_1_name`, then, within each group, grouping by `col_2_name`, and so on. The resulting DataFrame has one row for every combination of values in the given columns. Typically, grouping with subgroups is followed by `.reset_index()`.

Writing Functions

```
def function_name(argument_1, ..., argument_k):
  <function body>
```

For example, this function squares a number:

```
def square_a_number(number):
  return number**2
```

Applying Functions

`df.get(column_name).apply(function_name)`
: Applies a function of one parameter to every entry in the column. Returns a Series of the same size containing the results.

`if`-statements

```
if <condition>:
  <if body>
elif <second_condition>:
  <elif body>
elif <third_condition>:
  <elif body>
...
else:
  <else body>
```

Conditionally execute code. The `elif` and `else` blocks are optional.

Statistics and Hypothesis Testing

A **sample** is a subset of a **population**. A **statistic** is a number computed using the sample. The field of statistics is about using a sample to say something about the population, which is called **inference**.

An **experiment** is a process whose outcome is random; for example, flipping 100 coins. An **observed statistic** is a statistic computed from the outcome of an experiment; for example, the number of heads observed. A **model** is a set of assumptions about how the data was generated. For example: the result of a coin flip is equally likely to be heads or tails. **Hypothesis testing** is the process of testing the validity of a model based on simulating data with the model and comparing it to observed data.

To perform a **hypothesis test**, we first establish a **null hypothesis**: this is a precise assumption about how the data was generated. For instance: the coin is fair. Then we state an **alternative hypothesis**, such as: the coin is not fair.

To **test** the hypothesis, we compute the probability of seeing an outcome at least as extreme as the observed statistic under the assumptions of the null hypothesis; this is called the **p-value**. In practice, we do this by simulating many of outcomes using the null hypothesis and counting how many times the outcome is equal to or more extreme than what was originally observed.

`for`-loops

```
for <loop variable> in <sequence>:
  <loop body>
```

Performs the loop body for every element of the sequence. For example, to print the squares of the numbers 0 through 9:

```
for i in np.arange(10):
  print(i**2)
```

Random Sampling

`np.random.choice(arr, size, replace=True, p=[p_0, p_1, ...])`
: An array of elements chosen from `arr` at random, with replacement, such that `array[i]` is selected with probability `p_i`. `replace` can be omitted, as its default value is `True`. If `size` is omitted, the result is a single randomly chosen element instead of an array of length `size`.

`np.random.multinomial(n, [p_0, p_1, p_2, ...])`
: An array where each element contains the number of occurrences of an event, where events have probabilities `p_0`, `p_1`, `p_2`, ... For instance, if each M&M is red with probability 0.2, green with probability 0.5, and brown with probability 0.3, then a random selection of 100 M&Ms is given by:

`np.random.multinomial(100, [0.2, 0.5, 0.3]).`

The result might be `[22, 45, 33]`, representing the number of red, green, and brown M&Ms, respectively.

`np.random.permutation(arr)`
: A random shuffling/reordering of the input.

`df.sample(n, replace=False)`
: A random sample of n rows from `df`, selected without replacement. `replace` can be omitted, as its default value is `False`.

Bootstrapping and Confidence Intervals

When we compute a statistic from a sample, such as the median salary of San Diego city employees, since our sample is random, our statistic could have been different if we'd had a different sample. Bootstrapping allows us to answer: "how different could it have been?" by giving us an approximation of the distribution of the sample statistic.
Suppose `salaries` contains a column called "Salary" containing the salary of each employee in a sample. The observed median salary is:

```
observed = salaries.get('Salary').median()
```

To make a 95% confidence interval for the median salary, we bootstrap by repeatedly resampling the data in our sample, with replacement:

```
boot_medians = np.array([])
for i in np.arange(10000):
  # 1. Resample the data.
  resample = salaries.sample(salaries.shape[0], replace=True)

  # 2. Compute the statistic on the bootstrap resample.
  boot_median = resample.get('Salary').median()

  # 3. Save the result.
  boot_medians = np.append(boot_medians, boot_median)
```

The endpoints of a 95% bootstrapped confidence interval are:

```
left = np.percentile(boot_medians, 2.5)
right = np.percentile(boot_medians, 97.5)
```

This interval gives a range of reasonable values where we'd guess the population median would fall.

Permutation Testing

We use a permutation test to test whether two samples were drawn from the same population. For instance, suppose we administer an exam with versions A and B and record the versions and scores for each student in a DataFrame `scores` with columns "Version" and "Score". We notice that students who took version A had a higher average score than students who took version B. We can test whether version A was indeed significantly harder than version B with a permutation test.

First, we decide on a test statistic, such as the difference between the mean scores for each version:

```
def difference_in_means(scores):
  means = scores.groupby('Version').mean()
  return (means.get('Score').loc['A'] -
          means.get('Score').loc['B'])
```

For the permutation test, we repeatedly shuffle one column to generate two random samples from our data, and compute the test statistic on each pair of random samples.

```
statistics = np.array([])
for i in np.arange(5000):
  # 1. Shuffle the versions.
  shuffled = scores.assign(Version=
  np.random.permutation(scores.get('Version')))

  # 2. Compute the test statistic.
  statistic = difference_in_means(shuffled)

  # 3. Save the result.
  statistics = np.append(statistics, statistic)
```

We then compute the observed test statistic and compute how often we saw a difference in means greater than or equal to our observed difference.

```
observed = difference_in_means(scores)
p_value = np.count_nonzero(statistics >= actual) / 5000
```

Standard Units, Correlation, Regression

To convert a value x_i from a column x to **standard units**:

$$x_{i\,(\text{su})} = \frac{x_i - \text{mean of } x}{\text{SD of } x}$$

Standard units represent "the number of SDs above the mean."

The **correlation coefficient**, r, is the average value of the product of two variables, when both variables are measured in standard units. r measures the strength of the linear association between the two variables, and is always between -1 and 1.

In standard units, the regression line has slope r and intercept 0. To make predictions:

$$\text{predicted } y_{i\,(\text{su})} = r \cdot x_{i\,(\text{su})}$$

In original units, the slope m and intercept b of the regression line are:

$$m = r \cdot \frac{\text{SD of } y}{\text{SD of } x}$$
$$b = (\text{mean of } y) - m \cdot (\text{mean of } x)$$

To make predictions:

$$\text{predicted } y_i = m \cdot x_i + b$$

Spread of a Distribution

For a data set with n observations, the variance is the average squared deviation from the mean. The standard deviation is the square root of the variance.

$$\text{SD} = \sqrt{\frac{(\text{value}_1 - \text{mean})^2 + \ldots + (\text{value}_n - \text{mean})^2}{n}}$$

Chebyshev's Inequality states that in any dataset, at least $1 - 1/z^2$ of the data falls within z SDs of the mean.

Range	All Distributions	Normal Distribution
mean ± 1 SD	at least 0%	about 68%
mean ± 2 SDs	at least 75%	about 95%
mean ± 3 SDs	at least 88.8%	about 99.73%

The Standard Normal Distribution

The standard normal distribution has mean 0, standard deviation 1, and inflection points at ± 1.

To compute the area under the standard normal curve from $-\infty$ to z, use

```
scipy.stats.norm.cdf(z)
```

To compute the area under the standard normal curve between z_1 and z_2, where $z\_1 < z\_2$:

```
scipy.stats.norm.cdf(z_2) - scipy.stats.norm.cdf(z_1)
```

The Central Limit Theorem

Consider drawing a large random sample, with replacement, from some population. Imagine all the ways the sample mean could turn out. The sample mean is a random quantity, which has some distribution. The CLT says that "the distribution of possible sample means is approximately normal, no matter the distribution of the population."

The mean of the distribution of possible sample means is the population mean. The standard deviation is

$$\text{SD of Distribution of Possible Sample Means} = \frac{\text{Population SD}}{\sqrt{\text{sample size}}}$$

We often use the sample mean and SD instead of the population mean and SD, since we have this information for a sample, but not the population.

A CLT-based 95% confidence interval for the sample mean is:

$$\left[\text{sample mean} - 2 \cdot \frac{\text{sample SD}}{\sqrt{\text{sample size}}},\quad \text{sample mean} + 2 \cdot \frac{\text{sample SD}}{\sqrt{\text{sample size}}}\right]$$

For all of this to work, the sample has to be randomly drawn with replacement.