DSC 80 winter 2025 midterm review

TA: Mizuho Fukuda
Tutors: Gabriel Cha, Ylesia Wu

midterm logistics

- tuesday, 2/11 in class
- 11:00 am - 12:20 pm in SOLIS 104
- see seating chart here
- 80 minutes paper exam
- 1 sheet of hand-written notes (front + back)
- lectures 1-8 (no HTML)

table of contents

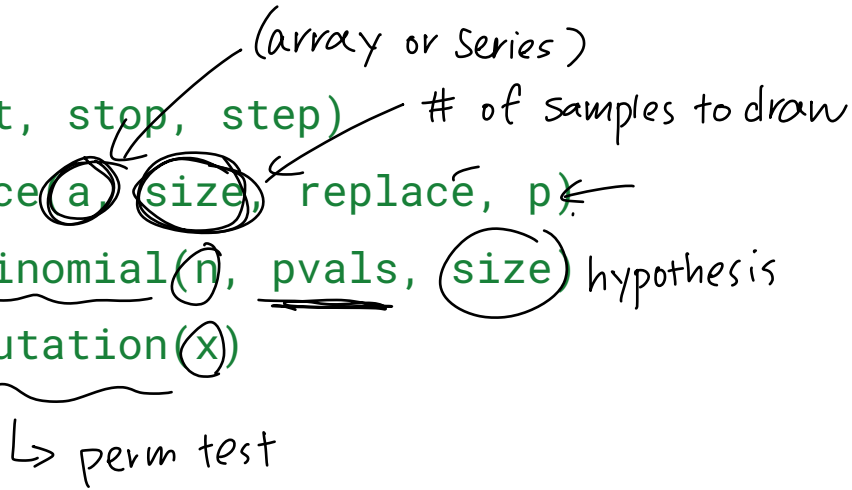
numpy and pandas basics

numpy basics

numpy supports vectorized operations (quicker than a loop)

📝methods to note

- `np.arange(start, stop, step)`
- `np.random.choice(a, size, replace, p)`
- `np.random.multinomial(n, pvals, size)`
- `np.random.permutation(x)`



pandas basics

📝 **`.loc` vs. `.iloc`**

`.loc`: access a group of rows and columns by labels or boolean arrays

→ `heroes.loc['A-Bomb', 'Gender']`

`.iloc`: access rows and columns by integer-location-based indexing.

→ `heroes.iloc[0, 0]` matrix

dataframe: `heroes`

name	Gender	Eye color	Race
A-Bomb	Male	yellow	Human
Abe Sapien	Male	blue	Icthyo Sapien
Abin Sur	Male	blue	Ungaran
Abomination	Male	green	Human / Radiation
Abraxas	Male	blue	Cosmic Entity
...	...	...	...
Yellowjacket II	Female	blue	Human
Ymir	Male	white	Frost Giant
Yoda	Male	brown	Yoda's species
Zatanna	Female	blue	Human
Zoom	Male	red	-

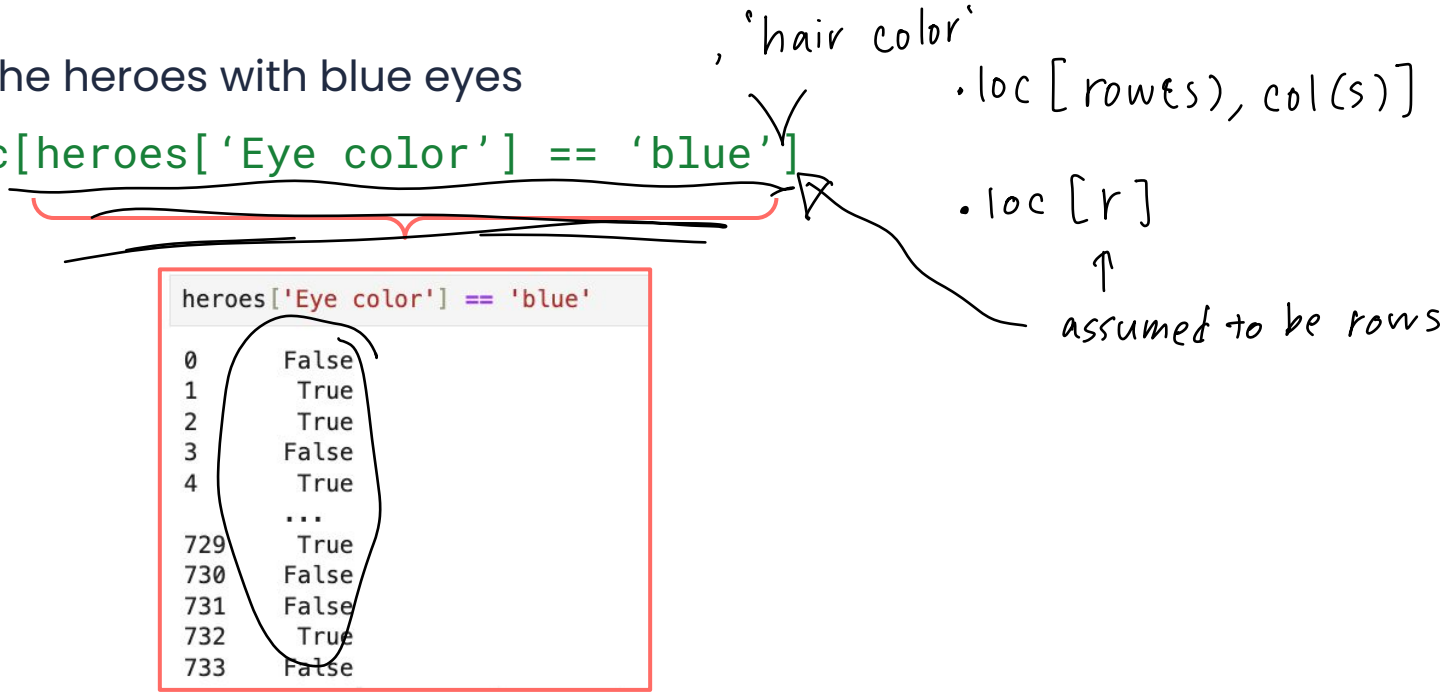
734 rows × 3 columns

pandas basics

query: a special case of `.loc`

e.g. get all the heroes with blue eyes

```
heroes.loc[heroes['Eye color'] == 'blue']
```



pandas basics

query: a special case of `.loc`

e.g. get all the heroes with blue eyes

`heroes.loc[{True, False, True, … , False, False}]`

→ a subset of heroes with only rows that are True

	name	Gender	Eye color	Race
1	Abe Sapien	Male	blue	Icthyo Sapien
2	Abin Sur	Male	blue	Ungaran
4	Abraxas	Male	blue	Cosmic Entity
5	Absorbing Man	Male	blue	Human

pandas basics

query: a special case of `.loc`

e.g. get all the heroes with blue eyes

```
heroes.loc[heroes['Eye color'] == 'blue']
```

=

```
heroes[[heroes['Eye color'] == 'blue']]
```

pandas basics

query: a special case of `.loc`

get all the heroes with blue eyes **and** blond hair:

```
heroes.loc[(heroes['Eye color'] == 'blue') & (heroes['Hair color'] == 'blond')]
```

get all the heroes with blue eyes **or** blond hair:

```
heroes.loc[(heroes['Eye color'] == 'blue') | (heroes['Hair color'] == 'blond')]
```

- '&' is for 'and'
- '|' is for 'or'

pandas basics

📝more pandas methods

- `df.sort_values('column'or ['col1', 'col2'], ascending=True)`
- `df.set_index('column')`
- `df['col']` → Series; `df[['col1',...,'col2']]` → sub DataFrame
- `df.index` → index object (not a python list)
- `ser.unique()` → numpy array
- `ser.nunique()` → int
- `ser.value_counts()` → Series (index: unique values, values: counts)
- `ser.idxmax()`
- `ser.astype(some type)`

no NaNs ✓

pandas basics

creating or modifying a col

- `df.assign(column_name = some Series)`

  not in quotes

  → not in place; creates a copy of the df

- `df['column_name'] = some Series`

  → modifies the original df

pandas basics

- `.str` accessor: allows you to use string methods
  - `df['str_col'].str.lower()`
  - `df['str_col'].str.strip().str.replace(',', '')`

- `.dt` accessor: allows you to use datetime methods
  - `df['date_col'].dt.day`
  - `df['date_col'].dt.dayofweek`

aggregating

definition of aggregation

In pandas, **aggregation** refers to the process of applying a function to summarize or compute a single result from a set of values in a **Series** or **grouped DataFrame**.

Kind		PetID	Name	Kind	Gender	Age	OwnerID
Cat	1	Q0-2001	Roomba	Cat	male	9	5508
	2	M0-2904	Simba	Cat	male	1	3086
	6	Z4-5652	Priya	Cat	female	7	7343
	7	Z4-4045	Simba	Cat	male	0	2700
	8	M8-7852	Cookie	Cat	female	8	7606
Dog	0	J6-8562	Blackie	Dog	male	11	5168
	4	P2-7342	Cuddles	Dog	male	13	4378
	9	J2-3320	Heisenberg	Dog	male	3	1319
	11	U4-9376	Scout	Dog	female	2	7846
	12	H8-1429	Lily	Dog	female	3	7846
Parrot	3	R3-7551	Keller	Parrot	female	2	7908
	5	X0-8765	Vuitton	Parrot	female	11	7581
	16	H8-8856	Bandit	Parrot	male	11	6102
	19	Q0-3593	Oakley	Parrot	female	4	4989
	20	O8-2501	Bandit	Parrot	male	5	1899

Kind	PetID	Name	Kind	Gender	Age	OwnerID
Cat	some value	some value	some value	some value	some value	some value
Dog	some value	some value	some value	some value	some value	some value
Parrot	some value	some value	some value	some value	some value	some value

simpson's paradox

📝simpson's paradox is a result of **weighted averages**

unit price of each product at each store:

product	store A		store B
X	1	<	3
Y	5	<	7
Z	20	<	30

expect overall avg A < B

if each store sold 1 unit of each item:

Average Revenue of Store A (R_a)=
(1(1) + 1(5) + 1(20)) / 3 = 8.667

Average Revenue of Store B (R_b)=
(1(3) + 1(7) + 1(30)) / 3 = 13.333...

As expected, R_a < R_b. → no paradox

simpson’s paradox

📝simpson's paradox is a result of **weighted averages**

unit price of each product at each store:

product	store A		store B
X	1	<	3
Y	5	<	7
Z	20	<	30

now let’s say:
store A sells a lot of product Z
and store B sells a lot of product X

Average Revenue of Store A (R_a)=
(1(1) + 1(5) + 20(20)) / 22 = 18.455

Average Revenue of Store B (R_b)=
(20(3) + 1(7) + 1(30)) / 22 = 4.409
R_a > R_b → simpson’s paradox

For this question, we'll continue using the `df` and `foods` tables form Question 1. Dyaln and Giorgia want to compare their CO_2 emissions. They added a new column called `'bean'` to `df` that contains `True` if the food was a bean (e.g. "Pinto beans") and `False` otherwise. Then, they compute the following pivot table:

	Dylan's co2/kg	Giorgia's co2/kg
bean=True	5	10
bean=False	50	80

Each entry in the pivot table is the average CO_2 emissions for Dylan and Giorgia per kg of food they ate (CO_2/kg) for both bean and non-bean foods.

Suppose that overall, Dylan produced an average of 41 CO_2/kg of food he ate, while Giorgia produced an average of 38 CO_2/kg.

Problem 2.2

Dylan and Giorgia want to figure out exactly when Simpson's paradox occurs for their data. Suppose that 0.2 proportion of Dylan's food was bean foods. What range of proportions for Giorgia's bean food would cause Simpson's paradox to occur? Show your work in the space below, **then write your final answer in the blanks at the bottom of the page.** Your final answers should be between 0 and 1. Leave your answers as simplified fractions.

Between $\frac{39}{70}$ and $\frac{1}{\quad}$

[DSC80 FA23 Midterm Question 2]

	Dylan's co2/kg		Giorgia's co2/kg
bean=True	5	$<$	10
bean=False	50	$<$	80
overall co2	$D = 41$	$>$	G

remember, simpson's paradox is a result of weighted averages!

$$\text{Giorgia's total} = B(10) + B'(80) > 41$$

$B' \hookrightarrow (1 - B)$

$$10B + 80(1 - B) > 41$$

$$1 \geq B \geq \frac{39}{70}$$

B: bean prop.

DSC80 FA23 Midterm Question 2

Groupby.agg

Kind		PetID	Name	Kind	Gender	Age	OwnerID
Cat	1	Q0-2001	Roomba	Cat	male	9	5508
	2	M0-2904	Simba	Cat	male	1	3086
	6	Z4-5652	Priya	Cat	female	7	7343
	7	Z4-4045	Simba	Cat	male	0	2700
	8	M8-7852	Cookie	Cat	female	8	7606
Dog	0	J6-8562	Blackie	Dog	male	11	5168
	4	P2-7342	Cuddles	Dog	male	13	4378
	9	J2-3320	Heisenberg	Dog	male	3	1319
	11	U4-9376	Scout	Dog	female	2	7846
	12	H8-1429	Lily	Dog	female	3	7846
Parrot	3	R3-7551	Keller	Parrot	female	2	7908
	5	X0-8765	Vuitton	Parrot	female	11	7581
	16	H8-8856	Bandit	Parrot	male	11	6102
	19	Q0-3593	Oakley	Parrot	female	4	4989
	20	O8-2501	Bandit	Parrot	male	5	1899

input

aggregate functions are applied to
each column of each group
→ outputs a **single value** per column per group

```
pets.groupby('Kind')[['Age']].mean()
```

df

Kind	Age
Cat	7.322581
Dog	6.789474
Parrot	6.583333

output

functions like `.mean()` or `.count()` are built in agg functions.
this is the same thing as doing `.agg(mean)`

Groupby.agg

Kind		PetID	Name	Kind	Gender	Age	OwnerID
Cat	1	Q0-2001	Roomba	Cat	male	9	5508
	2	M0-2904	Simba	Cat	male	1	3086
	6	Z4-5652	Priya	Cat	female	7	7343
	7	Z4-4045	Simba	Cat	male	0	2700
	8	M8-7852	Cookie	Cat	female	8	7606
Dog	0	J6-8562	Blackie	Dog	male	11	5168
	4	P2-7342	Cuddles	Dog	male	13	4378
	9	J2-3320	Heisenberg	Dog	male	3	1319
	11	U4-9376	Scout	Dog	female	2	7846
	12	H8-1429	Lily	Dog	female	3	7846
Parrot	3	R3-7551	Keller	Parrot	female	2	7908
	5	X0-8765	Vuitton	Parrot	female	11	7581
	16	H8-8856	Bandit	Parrot	male	11	6102
	19	Q0-3593	Oakley	Parrot	female	4	4989
	20	O8-2501	Bandit	Parrot	male	5	1899

input

custom agg functions for more complex operations

series

```
def my_agg_func(x):
    return x.sum() / x.shape[0]

pets.groupby('Kind')[['Age']].agg(my_agg_func)
```

← scalar

Kind	Age
Cat	7.322581
Dog	6.789474
Parrot	6.583333

output

Groupby.agg vs Groupby.apply

The input:

function passed into **.agg** takes in a Series (each column of the original df for each group)

```
def my_agg_func(x):
    return x.shape

pets.groupby('Kind')[['Age', 'OwnerID']]\
.agg(my_agg_func)
```

Kind	Age	OwnerID
Cat	(31,)	(31,)
Dog	(57,)	(57,)
Parrot	(12,)	(12,)

the function passed into **.apply** takes in a DataFrame (all columns for each group)

```
def my_apply_func(x):
    return x.shape

pets.groupby('Kind')[['Age', 'OwnerID']]\
.apply(my_apply_func)
```

```
Kind
Cat       (31, 2)
Dog       (57, 2)
Parrot    (12, 2)
dtype: object
```

Groupby.agg vs Groupby.apply

The output:

function passed into .**agg** must return a **single value** per group

```
def my_agg_func(x):
    return x.mean()  ← scalar

pets.groupby('Kind')[['Age', 'OwnerID']]\
.agg(my_agg_func)
```

Kind	Age	OwnerID
Cat	7.322581	5660.000000
Dog	6.789474	5119.824561
Parrot	6.583333	6936.666667

.apply is much more flexible:
function passed into apply can output a **single value**, a **Series**, or a **DataFrame**.

df

```
def my_apply_func(x):
    return x.head(2)  ← DataFrame with shape (2,2)

pets.groupby('Kind')[['Age', 'OwnerID']]\
.apply(my_apply_func)
```

(2,2)

		Age	OwnerID
Kind			
Cat	1	9	5508
	2	1	3086
Dog	0	11	5168
	4	13	4378
Parrot	3	2	7908
	5	11	7581

Groupby.agg vs Groupby.apply

`.apply` can do the everything that `.agg` can and more!

why would i use `.agg` then?

RUNTIME!
`.agg` is optimized to handle aggregation (Series to scalar operation) so it runs much faster than `.apply`!

conclusion:
use `.agg` for simple aggregation. only use `.apply` when doing complex operations that `.agg` cannot handle.

Groupby.filter

input:

- function passed into `.filter` must take in a **DataFrame**
- **one DataFrame per group**

output:

- function passed into `.filter` must return **a single boolean**
- **one boolean per group**

result:

- keep only the rows belonging to the group that are `True` based on the filter function

taking out entire groups

```
def my_filter_func(x):
    return x['Age'].mean() < 7

pets.groupby('Kind').filter(my_filter_func)
```

single boolean

	PetID	Name	Kind	Gender	Age	OwnerID
0	J6-8562	Blackie	Dog	male	11	5168
3	R3-7551	Keller	Parrot	female	2	7908
4	P2-7342	Cuddles	Dog	male	13	4378
5	X0-8765	Vuitton	Parrot	female	11	7581
9	J2-3320	Heisenberg	Dog	male	3	1319
...	...	...	...	...	...	...
91	U6-4890	Blackie	Dog	male	6	1546
93	F1-1855	Bandit	Parrot	male	2	9604
94	Z8-4419	Scooter	Dog	male	3	4464
95	U8-6473	Biscuit	Dog	female	3	1070
99	S5-5938	Taz	Dog	male	6	9427

Groupby.transform

input:

- function passed into `.transform` must take in a **Series**
- **one Series per column per group**

output:

- function passed into `.transform` must return **the same size Series**

result:

- the **same size DataFrame** as the original, with values transformed within each group

1 series per column per group

```
def my_transform_func(x):
    # calculate group-wise z-score for each column
    return (x - x.mean()) / x.std()

pets.groupby('Kind')[['Age', 'OwnerID']].transform(my_transform_func)
```

does not change shape of x

	Age	OwnerID
0	0.982026	0.019260
1	0.373599	-0.055659
2	-1.408181	-0.942540
3	-1.299121	0.375419
4	1.448488	-0.296578
...	...	...
95	-0.883823	-1.619102
96	-0.962736	0.615178
97	-0.071846	1.364011
98	1.487211	1.356687
99	-0.184130	1.721990

100 rows × 2 columns

unravels the groups after transformations are done

same size as before.
no rows or columns lost.

pivot table

- grouping by two columns → turning one group into columns

```
pets.groupby(['Kind', 'Gender'])[['Age']].mean()
```

		Age
Kind	Gender	
Cat	female	7.250000
	male	7.368421
Dog	female	5.909091
	male	7.342857
Parrot	female	6.714286
	male	6.400000

pivot

```
pets.pivot_table(
    index='Kind',
    columns='Gender',
    values='Age',
    aggfunc='mean'
)
```

Gender	female	male
Kind		
Cat	7.250000	7.368421
Dog	5.909091	7.342857
Parrot	6.714286	6.400000

df

	date	name	food	weight
0	2023-01-01	Sam	Ribeye	0.20
1	2023-01-01	Sam	Pinto beans	0.10
2	2023-01-01	Lauren	Mung beans	0.25
3	2023-01-02	Lauren	Lima beans	0.30
4	2023-01-02	Sam	Sirloin	0.30

exclude those that ate

Find all the unique people who did not eat any food containing the word "beans".

df

```
def foo(x):
    return x['food'].str.contains('beans').sum() == 0

df.groupby('name').filter(foo)['name'].unique()
```

series

count # of Trues

every element of x['food']

DSC80 FA23 Midterm Question 1

hid	number	street
1	7370	Torrey Pines Rd
2	960	Mission Blvd
3	5490	La Jolla Village Dr
4	5291	Gilman Dr
5	5834	Torrey Pines Rd

ints → str

Compute a DataFrame containing the proportion of 4-digit address numbers for each unique street in h.

← series (df if apply)

```
def foo(x):
    lengths = x.astype(str).str.len()   ← x['number'].astype
    return (lengths == 4).mean()        ← single scalar

h.groupby('street').agg(foo)
                    or
                    .apply
```

DSC80 SP24 Midterm Question 1

merging

merging

LEFT

	col1_left	col2_left
0	A	1
1	A	2
2	B	1
3	C	1
4	C	2
5	D	1

RIGHT

	col1_right	col2_right
0	A	1
1	B	1
2	C	1
3	C	2
4	C	3
5	E	1

inner merge

- one row for each match
- **does not** include rows with no match

same for all types of merges except cross join

```
LEFT.merge(
    RIGHT,
    how='inner',
    left_on='col1_left',
    right_on='col1_right'
)
```

LEFT

	col1_left	col2_left
0	A	1
1	A	2
2	B	1
3	C	1
4	C	2
5	D	1

RIGHT

	col1_right	col2_right
0	A	1
1	B	1
2	C	1
3	C	2
4	C	3
5	E	1

do not appear in merged df because there is not matching value in the other df.

	col1_left	col2_left	col1_right	col2_right
0	A	1	A	1
1	A	2	A	1
2	B	1	B	1
3	C	1	C	1
4	C	1	C	2
5	C	1	C	3
6	C	2	C	1
7	C	2	C	2
8	C	2	C	3

left merge

- one row for each match
- includes **all rows of the left** df even if there is no match on the right df.
- unmatched rows are filled with `np.nan`

LEFT

	col1_left	col2_left
0	A	1
1	A	2
2	B	1
3	C	1
4	C	2
5	D	1

RIGHT

	col1_right	col2_right
0	A	1
1	B	1
2	C	1
3	C	2
4	C	3
5	E	1

not

appears in merged df

```
LEFT.merge(
    RIGHT,
    how='left',
    left_on='col1_left',
    right_on='col1_right'
)
```

	col1_left	col2_left	col1_right	col2_right
0	A	1	A	1.0
1	A	2	A	1.0
2	B	1	B	1.0
3	C	1	C	1.0
4	C	1	C	2.0
5	C	1	C	3.0
6	C	2	C	1.0
7	C	2	C	2.0
8	C	2	C	3.0
9	D	1	NaN	NaN

right merge

- same logic as left merge but all rows of the **right df** are kept regardless of whether there is a match in the left df.

```
LEFT.merge(
    RIGHT,
    how='right',
    left_on='col1_left',
    right_on='col1_right'
)
```

LEFT

	col1_left	col2_left
0	A	1
1	A	2
2	B	1
3	C	1
4	C	2
5	D	1

RIGHT

	col1_right	col2_right
0	A	1
1	B	1
2	C	1
3	C	2
4	C	3
5	E	1

appears in merged df

	col1_left	col2_left	col1_right	col2_right
0	A	1.0	A	1
1	A	2.0	A	1
2	B	1.0	B	1
3	C	1.0	C	1
4	C	2.0	C	1
5	C	1.0	C	2
6	C	2.0	C	2
7	C	1.0	C	3
8	C	2.0	C	3
9	NaN	NaN	E	1

outer merge

- still one row per match
- unmatched rows from both dfs appear in the merged df.

```
LEFT.merge(
    RIGHT, how='outer',
    left_on='col1_left',
    right_on='col1_right'
)
```

LEFT

	col1_left	col2_left
0	A	1
1	A	2
2	B	1
3	C	1
4	C	2
5	D	1

RIGHT

	col1_right	col2_right
0	A	1
1	B	1
2	C	1
3	C	2
4	C	3
5	E	1

	col1_left	col2_left	col1_right	col2_right
0	A	1.0	A	1.0
1	A	2.0	A	1.0
2	B	1.0	B	1.0
3	C	1.0	C	1.0
4	C	1.0	C	2.0
5	C	1.0	C	3.0
6	C	2.0	C	1.0
7	C	2.0	C	2.0
8	C	2.0	C	3.0
9	D	1.0	NaN	NaN
10	NaN	NaN	E	1.0

both appear in merged df

tasks:

	category	completed	minutes	urgency	client
0	work	False	NaN	2.0	NaN
1	work	False	NaN	1.0	NaN
2	work	True	13.5	2.0	NaN
3	work	False	NaN	1.0	NaN
4	relationship	True	5.3	NaN	NaN
...	...	...	...	...	...
9831	consulting	True	71.7	2.0	San Diego Financial Analysts

clients:

index	rate	active
San Diego Financial Analysts	55.00	True
ABC LLC	95.25	True
SDUSD	45.00	False
NASA	75.00	True
Grandma	1.00	False

Fill in the code below so that it produces a DataFrame which has all of the columns that appear in tasks, but with two additional columns, rate and activity, listing the pay rate for each task and whether the client being consulted for is still active. The number of rows in your resulting DataFrame should be equal to the number of rows in tasks for which the value in 'client' is in clients.

only include rows from tasks if I can find a match in clients

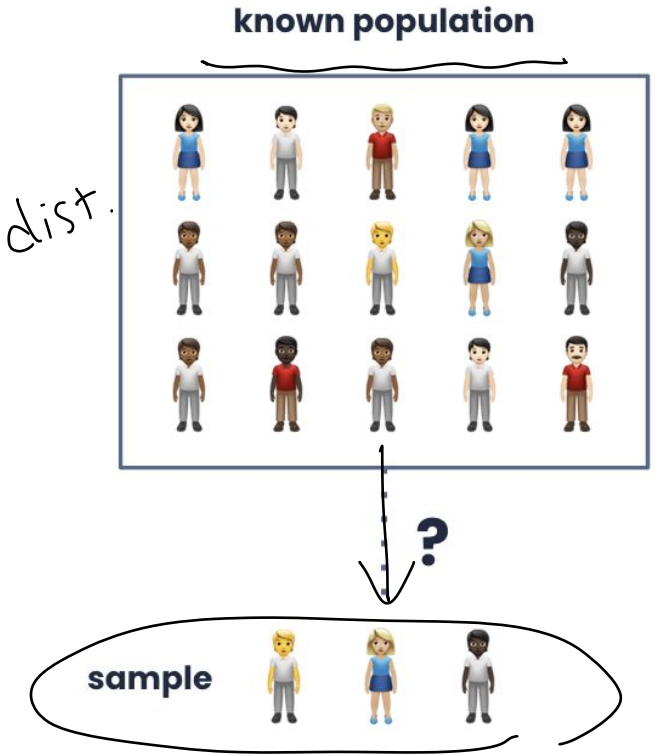
```
tasks.merge(clients, how=_____, _____, _____)
```

`'inner'`, `left_on='client'`, `right_index=True`

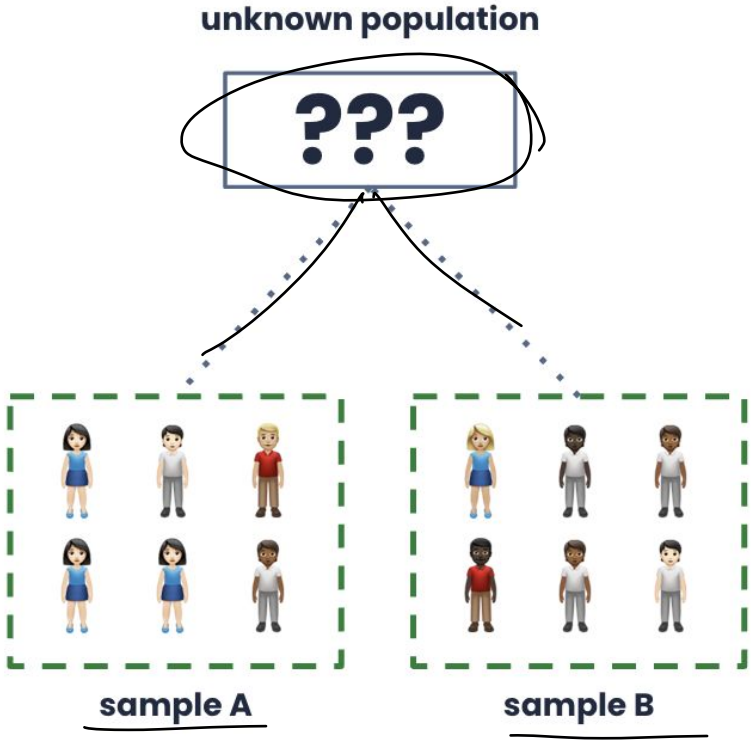
DSC80 SP22 Midterm Question 9

hypothesis & permutation tests

hypothesis test



permutation test



hypothesis test

I have a sample S.
I also have a population P.

Question:
Does sample S look like it is drawn from population P?

permutation test

I have two samples: A and B.
I don't know anything about the populations they come from.

Question:
Do samples A and B look like they are from the same distribution? In other words, do these samples look similar?

null and alternative hypotheses

null hypothesis ☝🏻

- must be an **exact statement**
- serves as your assumed ground truth when simulating empirical distribution of test statistics
- e.g. exactly 4% of cookies from the store are burnt

alternative hypothesis🤔

- what you suspect may be the case based on what you observe
- could be >, < or ≠
- e.g. more than 4% of cookies from the store are burnt

15/256 burnt

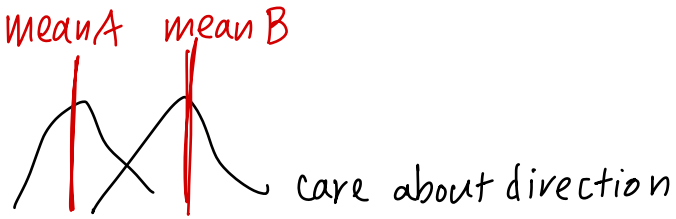
i suspect

test statistics

- a single summary statistic
  - e.g. proportion of cookies burnt
- difference in means
  - mean_sampleA - mean_sampleB
- absolute difference in means
  - | mean_sampleA - mean_sampleB |
- TVD

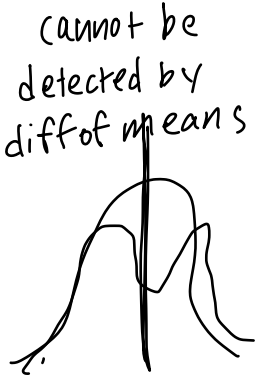
$$\text{TVD}(A, B) = \frac{1}{2} \sum_{i=1}^{k} |a_i - b_i|$$

  - measures similarity between two **categorical distributions**
- K-S test statistic (Kolmogorov-Smirnov)
  - `scipy.stats.ks_2samp(A, B).statistic`
  - measures similarity between two **numerical distributions**



not directional

	A	B
cat 1	0.3	0.4
cat 2	0.2	0.3
cat 3	0.5	0.3



putting everything together...

general workflow for hypothesis test:

1. decide on a test statistic.
2. compute test statistic for the the sample (this is your **observed test statistic**)
3. state null and alternative hypotheses
4. simulate test statistics based on null distribution → list of test statistics
5. calculate p-value:
   proportion of the simulated test statistics that are at least as extreme as the observed test statistic

mean(S) > mean(P)

diff in means (mean(S) − mean(P))

general workflow for hypothesis test (using the cookies example from lab4):

1. test statistic:
   proportion of burnt cookies
2. compute observed test statistic:
   proportion of burnt cookies you observe = 15/250 = 0.06
3. state hypotheses:
4. **null:** proportion of burnt cookies = 0.04 (supposed ground truth)
   alternative: proportion of burnt cookies > 0.04 (because i observed 0.06)
5. simulate test statistics based on null distribution
   `np.random.multinomial(250, [0.96, 0.04], N)`
   → assuming the store is telling the truth, i simulate N batches of 250 cookies
6. calculate p-value:

```
num_burnt = simulations[:, 1]
p-value = np.count_nonzero(num_burnt >= 15) / N
```

general workflow for permutation test:

1. decide on a test statistic.
2. compute test statistic for the the two samples (**observed test statistic**)
3. state null and alternative hypotheses
4. simulate test statistics by shuffling the labels
5. calculate p-value:

   proportion of the simulated test statistics that are at least as extreme as the observed test statistic

general workflow for permutation test (using the skittles example from lab4):

1. test statistic:

   TVD (comparing two categorical distributions)

2. compute test statistic:

   TVD between color distribution of Waco and Yorkville

3. state null and alternative hypotheses

   null: there is no difference between the two factories' color distributions

   alternative: there is a difference between the two factories' color distributions

4. simulate test statistics

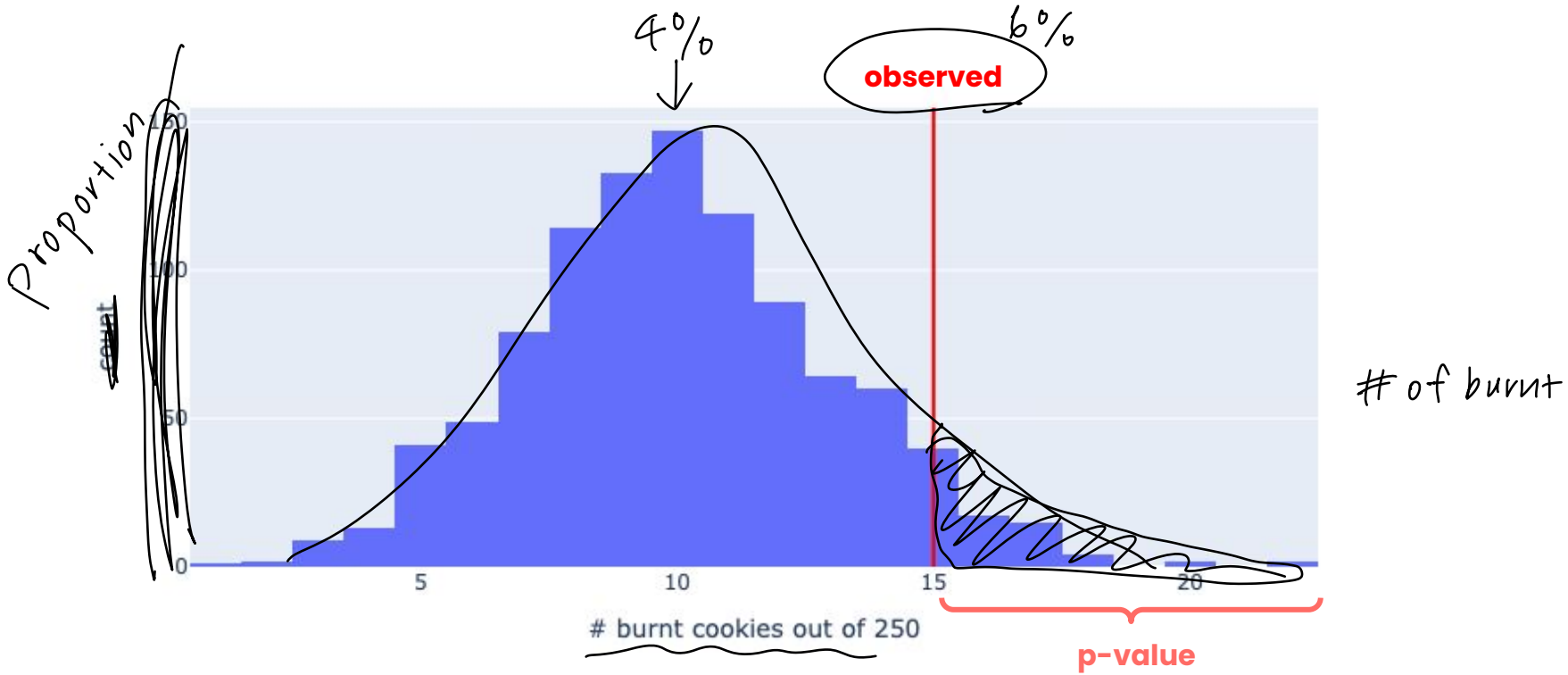
   ```
   sk['Factory'] = np.random.permutation(sk['Factory'])
   simulated_tvds.append(tvd(dg))
   ```

   → repeat N times

5. calculate p-value: `(np.array(simulated_tvds) >= observed).mean()`

of test statistic

visualizing empirical distribution & p-value



Power outages

The first few rows of the `o` DataFrame are shown below. For this problem, assume that some of the `duration` values are missing.

For each test, select the one correct procedure to simulate a single sample under the null hypothesis, and select all test statistics that can be used for the hypothesis test among the choices given.

oid	time	duration	hour	is_morning
1	2024-04-07 03:21:00	3	3	True
2	2024-04-20 16:35:00	70	16	False

sample

population

prob. power out ~ uniformly distributed through of 24 hours of the day

Problem 5.1

Null Hypothesis: Every hour of the day (0, 1, 2, etc.) has an equal probability of having a power outage.

Alternative Hypothesis: At least one hour is more prone to outages than others.

hypothesis

Simulation procedure:

- ○ `np.random.multinomial(100, [1/2] * 2)` ✗
- ○ `np.random.multinomial(100, [1/24] * 24)` ✓
- ○ `o['hour'].sample(100)` not valid
- ○ `np.random.permutation(o['duration'])` ✗

Test statistic:

- ☐ Difference in means
- ☐ Absolute difference in means
- ☐ Total variation distance ✗ 24 categories
- ☐ K-S test statistic ✗ discrete numerical

[DSC80 SP24 Midterm Question 5]

The first few rows of the `o` DataFrame are shown below. For this problem, assume that some of the `duration` values are missing.

For each test, select the one correct procedure to simulate a single sample under the null hypothesis, and select all test statistics that can be used for the hypothesis test among the choices given.

oid	time	duration	hour	is_morning
1	2024-04-07 03:21:00	3	3	True
2	2024-04-20 16:35:00	70	16	False

MAR analysis

Problem 5.2

Permutation test

Null: The proportion of outages that happen in the morning is the same for both recorded durations and missing durations.

Alternative: The outages are more likely to happen in the morning for missing durations than for recorded durations.

Simulation procedure:

- ○ `np.random.multinomial(100, [1/2] * 2)`
- ○ `np.random.multinomial(100, [1/24] * 24)`
- ○ `o['hour'].sample(100)`
- ✗ `np.random.permutation(o['duration'])`

Test statistic:

- ✗ Difference in means
- ☐ Absolute difference in means
- ☐ Total variation distance
- ☐ K-S test statistic

[DSC80 SP24 Midterm Question 5]

The first few rows of the `o` DataFrame are shown below. For this problem, assume that some of the `duration` values are missing.

For each test, select the one correct procedure to simulate a single sample under the null hypothesis, and select all test statistics that can be used for the hypothesis test among the choices given.

oid	time	duration	hour	is_morning
1	2024-04-07 03:21:00	3	3	True
2	2024-04-20 16:35:00	70	16	False

Problem 5.3

permutation test (MAR analysis)

Null: The distribution of hours is the same for both recorded durations and missing durations.

Alternative: The distribution of hours is different for recorded durations and missing durations.

Simulation procedure:

- ○ `np.random.multinomial(100, [1/2] * 2)`
- ○ `np.random.multinomial(100, [1/24] * 24)`
- ○ `o['hour'].sample(100)`
- ✗ `np.random.permutation(o['duration'])`

Test statistic:

- ☐ Difference in means ←
- ✗ Absolute difference in means
- ✗ Total variation distance
- ✗ K-S test statistic

DSC80 SP24 Midterm Question 5

missingness & imputation

missingness mechanisms

- **missing by design (MD)**
  when data is intentionally left out; you know exactly what a missing value in that column represents
- **not missing at random (NMAR)**
  when the chance of a value being missing is dependent on the missing value itself
- **missing at random (MAR)**
  when the chance of a value being missing is dependent on other columns
- **missing completely at random (MCAR)**
  when the chance of a value being missing is completely due to chance

testing MAR of column X dependent on Y

permutation test

group A: X is missing

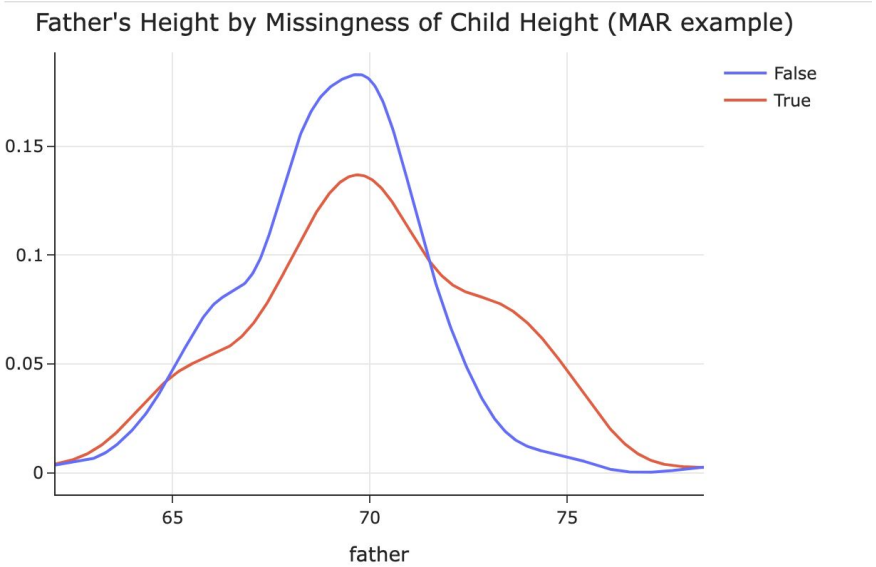
group B: X is not missing

null:

the distribution of variable Y is the same for group A and B

alternative:

the distribution of variable Y is different for groups A and B



from lecture 8

At the Estancia La Jolla, the hotel manager enters information about each reservation in the DataFrame `guests`, after guests check into their rooms. Specifically, `guests` has the columns:

- `"id" (str)`: The booking ID (e.g. `"SN1459"`).
- `"age" (int)`: The age of the primary occupant (the person who made the reservation).
- `"people" (int)`: The total number of occupants.
- `"is_business" (str)`: Whether or not the trip is a business trip for the primary occupant (possible values: `"yes"`, `"no"`, and `"partially"`).
- `"company" (str)`: The company that the primary occupant works for, if this is a business trip.
- `"loyalty" (int)`: The loyalty number of the primary occupant. Note that most business travelers have a loyalty number.

Some of the values in `guests` are missing.

What is the most likely missingness mechanism of the `"loyalty"` column?

A. Missing by design
B. Missing at random
C. Not missing at random
D. Missing completely at random

DSC80 WI24 Final Question 2

At the Estancia La Jolla, the hotel manager enters information about each reservation in the DataFrame `guests`, after guests check into their rooms. Specifically, `guests` has the columns:

- `"id" (str)`: The booking ID (e.g. `"SN1459"`).
- `"age" (int)`: The age of the primary occupant (the person who made the reservation).
- `"people" (int)`: The total number of occupants.
- `"is_business"` `(str)`: Whether or not the trip is a business trip for the primary occupant (possible values: `"yes"`, `"no"`, and `"partially"`).
- `"company" (str)`: The company that the primary occupant works for, if this is a business trip.
- `"loyalty" (int)`: The loyalty number of the primary occupant. Note that most business travelers have a loyalty number.

Some of the values in `guests` are missing.

What is the most likely missingness mechanism of the `"company"` column?

A. Missing by design
B. Missing at random
C. Not missing at random
D. Missing completely at random

DSC80 WI24 Final Question 2

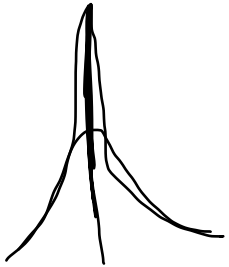
Fill in the blanks: To assess whether the missingness of `"is_business"` depends on `"age"`, we should perform a `__(i)__` with `__(ii)__` as the test statistic.

1. What goes in blank (i)?
   a. standard hypothesis test
   b. permutation test

2. What goes in blank (ii)?
   a. the total variation distance
   b. the sample mean
   c. the (absolute) difference in means
   d. the K-S statistic
   e. either the (absolute) difference in means or the K-S statistic, depending on the shapes of the observed distributions

DSC80 WI24 Final Question 2

imputation



- **Imputation with a single value:** e.g. mean, median, mode

  mean imputation - fill in missing values with the mean of that column

  pros: preserves the mean of the observed data

  cons: decreases the variance of the data;

  creates a biased estimate of the true mean if the data are not MCAR

  → **within-group (conditional) mean imputation**

  using different mean for each group of the column missingness is dependent on

- **Probabilistic imputation -** fill in missing values by drawing from the distribution of the non-missing data

  pros: preserves the original data's distribution

  cons: random each time (best to do multiple imputations and aggregate the results)

Doris wants to use multiple imputation to fill in the missing values in `'WeightAlt'`. She knows that `'WeightAlt'` is MAR conditional on `'BCS'` and `'Age'`, so she will perform multiple imputation conditional on `'BCS'` and `'Age'` - each missing value will be filled in with values from a random `'WeightAlt'` value from a donkey with the same `'BCS'` and `'Age'`. Assume that all `'BCS'` and `'Age'` combinations have observed `WegihtAlt` values. Fill in the blanks in the code below to estimate the median of `'WeightAlt'` using multiple imputation conditional on `'BCS'` and `'Age'` with 100 repetitions. A function `impute` is also partially filled in for you, and you should use it in your answer.

```
def impute(col):
    col = col.copy()
    n = col.isna().sum()
    fill = np.random.choice(col.dropna(), n)
    col[col.isna()] = fill
    return col

results = []
for i in range(100):
    imputed = (donkeys.groupby(['BCS', 'Age'])['WeightAlt'].transform(impute))
    results.append(imputed.median())
```

donkeys DataFrame:

	id	BCS	Age	Weight	WeightAlt
0	d01	3.0	<2	77	NaN
1	d02	2.5	<2	100	NaN
2	d03	1.5	<2	74	NaN

`id`	A unique identifier for each donkey (`d01`, `d02`, etc.).
`BCS`	Body condition score: from 1 (emaciated) to 3 (healthy) to 5 (obese) in increments of 0.5.
`Age`	Age in years: <2, 2–5, 5–10, 10–15, 15–20, and over 20 years.
`Weight`	Weight in kilograms.
`WeightAlt`	Second weight measurement taken for 30 donkeys. NaN if the donkey was not reweighed.

DSC80 FA23 Midterm Question 4

questions?

good luck tomorrow!