Data C100/200, Midterm

Fall 2022

Name: _____

Email: _____@berkeley.edu

Student ID: _____

Name and SID of the person on your right: _____

Name and SID of the person on your left: _____

Instructions:

This midterm exam consists of **60 points** spread out over **5 questions** and the Honor Code and must be completed in the **110 minute** time period ending at **9:00 PM**, unless you have accommodations supported by a DSP letter.

Note that some questions have circular bubbles to select a choice. This means that you should only **select one choice**. Other questions have boxes. This means you should **select all that apply**. Please shade in the box/circle to mark your answer.
You must also write your Student ID at the top each page.

Honor Code [1 Pts]:

As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others. I am the person whose name is on the exam and I completed this exam in accordance with the Honor Code.

Signature: _____

This page has been intentionally left blank.

1 Monopoly Mistakes [15 Pts]

It's the annual Monopoly World Championship! The finalists: Shawn, Amanda, Neil, and Annie are playing Monopoly, a board game where players pay a price to buy properties, which can then generate income for them. Each property can be owned by only one player at a time. At the end of the game, the player with the most money wins.

Shawn wants to figure out which properties are most worth buying. He creates a DataFrame `income` with data on the current game state, shown on the **left**. He also finds a DataFrame `properties` with data on Monopoly properties, shown on the **right**.

Both tables have 28 rows. For brevity, only the first few rows of each DataFrame are shown.

	Player	Property	Income Generated
0	Shawn	Boardwalk	$425
1	Amanda	Park Place	$375
2	Neil	Marvin Gardens	$200
3	NaN	Kentucky Ave	NaN
4	Shawn	Pennsylvania Ave	$150
5	Annie	Oriental Ave	$50
6	Amanda	Baltic Ave	$60

`income`

	Property	Property Color	Purchase Price
0	Park Place	Dark Blue	350.0
1	Oriental Ave	Light Blue	100.0
2	Vermont Ave	Light Blue	100.0
3	Pacific Ave	Green	300.0
4	Boardwalk	Dark Blue	400.0
5	Illiniois Ave	Red	240.0
6	Atlantic Ave	Yellow	260.0

`properties`

- `Player` is the name of the player, as a **str**.
- `Property` is a property currently owned by the player, as a **str**.
- `Income Generated` is the amount of income a player has earned from that property so far, as a **str**.
- `Property` is the name of the property, as a **str**. There are 28 unique properties.
- `Property Color` is a color group that the property belongs to, as a **str**. There are 10 unique color groups, and each property belongs to a single group.
- `Purchase Price` is the price to buy the property, as a **float**.

Note: For properties that are not currently owned by any player, the `Player` and `Income Generated` columns in the `income` table have a NaN value.

(a) [1 Pt] What is the granularity of the `income` table?

Solution:
`Property`

(b) [2 Pts] Which of the following line(s) of code successfully returns a Series with the number of properties each player owns? **Select all that apply**.

- ☐ `income.groupby('Player').agg(pd.value_counts)`
- ☐ `income['Player'].value_counts()`
- ☐ `income['Player', 'Property'].groupby('Player').size()`
- ☐ `income.groupby('Player').size()`

Solution: Option A is wrong because it returns a DataFrame with multiple columns.
Options B and D are correct.
Option C will error because we attempt to select two column names with a single indexing operator, which will error.

(c) [5 Pts] He now decides to calculate the amount of profit from each property. He wants to store this in a column called `Profit` in the `income` DataFrame.

i. [2 Pts] To do this, he first has to transform the `Income Generated` column to be of a **float** datatype. **Write one line** of code to replace the old column with a new column, also called `Income Generated`, with the datatype modification described above. You may assume that each entry in `Income Generated` consists of a dollar sign ($) followed by a number, except for the NaN values.

Solution:
```
income['Income Generated'] = income['Income Generated']
.str[1:].astype(float)
```

ii. [3 Pts] Assuming that the answer to the last sub-part is correct, let's add a `Profit` column to the `income` DataFrame. **Fill in the following blanks to do this**, and please add arguments to function calls as you see appropriate.

Note: Profit is calculated by subtracting the purchase price from generated income.

```
combined_df = income.___A___(_____________B_____________)
income["Profit"] = _____________________C_____________________
```

α) [1 Pt] What goes in the blank indicated by the letter A?

β) [1 Pt] What goes in the blank indicated by the letter B?

γ) [1 Pt] What goes in the blank indicated by the letter C?

Solution:

```
combined_df = income.merge(properties, on = "Property")
income["Profit"] = combined_df["Income Generated"]
- combined_df["Purchase Price"]
```

(d) [2 Pts] Regardless of your answer to the previous sub-part, assume we've successfully created the `Profit` column. Let's help Shawn see how well he's doing by finding the average profit he's made on the properties he owns. Which of the following line(s) of code does this? **Select all that apply.**

- ☐ `income.groupby("Player")["Shawn"].agg(np.average)`
- ☐ **`income[income["Player"]=="Shawn"]["Profit"].mean()`**
- ☐ **`income.loc[income["Player"]=="Shawn", "Profit"].mean()`**
- ☐ `income.iloc[income["Player"]=="Shawn", "Profit"].mean()`

(e) [5 Pts] Oh no! He realizes he's lost more money than he's made. To solve this problem, he begins by writing some Pandas code to merge the `Property Color` column into the `income` DataFrame and drops all rows with NaN values. He calls this DataFrame `merged_df`. Shown below are the first few rows.

	Player	Property	Income Generated	Profit	Property Color
0	Shawn	Boardwalk	425.0	-25.0	Dark Blue
1	Amanda	Park Place	375.0	25.0	Dark Blue
2	Neil	Marvin Gardens	200.0	50.0	Yellow
3	Shawn	Pennsylvania Ave	150.0	-100.0	Green
4	Annie	Oriental Ave	50.0	0.0	Light Blue
5	Amanda	Baltic Ave	60.0	0.0	Purple

`merged_df`

Shawn decides he will now only buy properties from a color group that he deems "profitable." He deems a color group "profitable" if at least 50% of the properties in the group that are currently owned by players have made a positive (non-zero) profit for those players.

Fill in the following lines of code to help him display a DataFrame with a subset of the rows in `merged_df`: the rows with properties that belong to profitable color groups. Your solution may use fewer lines of code than we provide.

```
def func(group):
    _____
    _____
    _____
    _____

merged_df._____
```

Solution:

```
def func(group):
```

```
    if sum(group['Profit'] > 0)/len(group['Profit'])>=0.5:
        return True
    return False

merged_df.groupby("Color Group").filter(func)
```

2 Dungeons and Data [13 Pts]

Grog the ogre owns a drugstore near the village of Slimestone. One day, in a local dungeon, he discovers an ancient recipe for a health potion.

To test the recipe, Grog heads into Slimestone to find the first 50 villagers he sees, knocks them out with his club, and drags them back to his lair. Each of them sustains a large amount of damage from the club, as measured in "health points" (HP). Grog gives each of them a free sample of his new potion and observes how many HP are restored. He records various attributes about each study participant, as well as the number of HP that were restored by his potion.

	age	**species**	**str**	**dex**	**con**	**int**	**wis**	**cha**	**restored**
0	22	human	15	14	12	13	9	11	6
1	25	dwarf	19	14	18	8	9	12	13
3	36	human	13	9	13	7	19	17	10
...	...	...	...	...	...	...	...	...	...
48	121	gnome	17	10	25	16	13	21	9
49	372	elf	9	24	12	16	18	12	8

Data for Grog's experiment

The variables are:

- `age`: in years, an **int**.
- `species`: dwarf, elf, gnome, or human, stored as a **str**.
- `str`, `dex`, `con`, `int`, `wis`, `cha`: strength, dexterity, constitution, intelligence, wisdom, and charisma. Each is a number is an **int** between 1 and 30. For this problem, it doesn't matter what they mean.
- `restored`: the amount of HP restored by the potion (response variable), an **int**.

Grog wants to figure out about how many health points are usually restored, and also use regression modeling to figure out whether some types of villagers can expect to gain more from the potion than others.

(a) [1 Pt] Grog's sampling frame is the set of all villagers in Slimestone. Which of the following are true about the sample that Grog gets? **Select all that apply.**

- ☐ **It is a convenience sample**
- ☐ It is a probability sample
- ☐ It is a simple random sample
- ☐ It is a random sample with replacement
- ☐ We can be reasonably confident that the sample is representative of the sampling frame

(b) [1 Pt] Grog's first idea is to use a constant model to predict `restored`. He decides to minimize the MSE. What will be the optimal constant prediction?

- ◯ The maximum of the `restored` column
- ◯ The minimum of the `restored` column
- ◯ **The mean of the `restored` column**
- ◯ The median of the `restored` column
- ◯ The mode of the `restored` column

(c) [1 Pt] What type of variable is the `species` column?

- ◯ Qualitative, ordinal
- ◯ Quantitative, continuous
- ◯ **Qualitative, nominal**
- ◯ Quantitative, discrete

(d) [2 Pts] Grog wants to run a multiple linear regression to predict the response, `restored`, from all the other variables, calculating the optimal parameters using the formula

$$\hat{\theta} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{Y},$$

where $\mathbb{Y}$ is the column vector (or 50×1 matrix) of responses, and $\mathbb{X}$ is a $50 \times d$ matrix of numerical features. He needs to do something about the variable `species`, which is currently represented as **str** instead of a numerical value. He plans to use one-hot encoding. Which of the following could describe a correct implementation of one-hot encoding for `species`? **Select all that apply**, remembering that the species are dwarf, elf, gnome, and human.

- ☐ Grog adds four columns representing the one-hot encoding of the `species` column, along with a `bias` column of all ones
- ☐ **Grog adds four columns representing the one-hot encoding of the `species` column, and does *not* include a `bias` column of all ones**
- ☐ **Grog adds three columns representing the one-hot encoding of the `species` column, along with a `bias` column of all ones**
- ☐ Each column that Grog adds to the matrix has a single entry equal to 1, and the other 49 entries in the column are all 0.

(e) [1 Pt] Grog writes a Python script to implement OLS linear regression from scratch, and uses it to run a regression with the full data set of 50 observations, using all of the other variables to predict the response `restored`.

He checks the MSE for his model's predictions, and he finds that it is even higher than the MSE for the constant model from part (b). **Which of the following do you think is the best explanation for what happened?**

Note: You may assume for this part that the $\mathbb{X}^T\mathbb{X}$ matrix is invertible.

- ◯ He probably overfit the data, and that might be why he got such a high MSE.
- ◯ He probably underfit the data, and he needs to add more features to the regression until his MSE goes down.
- ◯ **This should never happen with a correctly implemented OLS method. He must have a bug in his code.**

Solution: Option A: Grog uses the **full data set** to train both models and computed the MSE on the same data, so even if he is overfitting, we wouldn't know based on the training MSE.

Option B: The constant model is the simplest model there is; any OLS linear regression model would be more complex (and thus less underfitting) than the constant model. This also does not explain why the MSE for the OLS model is higher.

(f) [3 Pts] Grog is still a little confused about what happened in part (e) but he has heard that when you have a lot of predictor variables, you might want to consider using regularization, such as the LASSO or Ridge regression, still using the MSE loss. Which of the following considerations might be a reason to prefer LASSO? **Select all that apply.**

- ☐ **LASSO is more likely than ridge to give an answer with most coefficients set to 0, which might make the final prediction equation a little easier to understand.**
- ☐ Because LASSO uses L1 instead of L2, the LASSO solution $\hat{\theta}$ will not be affected much by outlier data points, even if they are extreme outliers.
- ☐ LASSO regression has a closed form solution but Ridge regression does not.

Solution: Option A: We talked about this in class and section. LASSO has the tendency to set many values to 0.

Option B: LASSO uses L1 for the regularization term, but the loss function is still squared loss ($(y_i - \hat{y}_i)^2$). Thus, LASSO will also be affected by outlier data points.

Option C: Ridge has a closed form solution, LASSO doesn't.

(g) [2 Pts] Grog decides to use Ridge regression instead, and wants to use 5-fold cross-validation to select the hyperparameter. He takes his folds to be contiguous blocks, using the rows 0 to 9 for the first fold, 10 to 19 for the second, and so on.

He notices that his data set is sorted by `age`, the first column, so the villagers in the first fold are much younger than the ones in the last fold. Which of the following ideas would allow him to resolve this problem and keep it from affecting the results? **Select all that apply.**

- ☐ Don't use `age` as a predictor variable in the regression
- ☐ Re-sort the data, in increasing order of the variable `restored`, and then use contiguous blocks for his folds
- ☐ **Shuffle the rows of the data frame randomly, and then use contiguous blocks for his folds**
- ☐ There is nothing he can do to fix it, he just needs to start over and collect a new data set

(h) [2 Pts] Assume Grog has fixed the issue from the previous part to his satisfaction, and he now has 5 folds that he is happy with. He uses 5-fold cross-validation to decide on the hyperparameter Q to use as the maximum squared L2 norm for the parameters. That is, any particular choice of Q means minimizing MSE(θ) such that $\sum_{j=1}^{d} \theta_j^2 \leq Q$.

Grog does 5-fold cross-validation to evaluate a grid of 100 different values for Q ranging all the way from tiny values to enormous values of Q. He is surprised to find that, as Q gets larger and larger, the cross-validation MSE stops changing, so the top 13 values of Q all give exactly the same cross-validation estimate of MSE. **What do you think could best explain this?**

- ◯ Above a certain value of Q, he is overfitting so much that he is making an exactly correct prediction for every single training point, and once he reaches this point he can't overfit any more.
- ◯ Above a certain value of Q, his answer is very close to the constant model, so the predictions are virtually identical.
- ◯ **Above a certain value of Q, his predictions will coincide exactly with the predictions for OLS linear regression.**
- ◯ Above a certain value of Q, his optimization algorithm is not able to converge to the optimal parameters.
- ◯ Above a certain value of Q, the optimal parameters are so large that numerical overflow will make the calculation fail.

Solution: Recall from lecture that Q is the radius of the ball that we restrict our coefficients in. It is inversely related to the α regularization parameter we use in the other formulation or `sklearn`.

As Q gets larger, the ball gets bigger and the restrictions get looser; and at one point the ball will include the OLS solution and we will have no regularization at all.

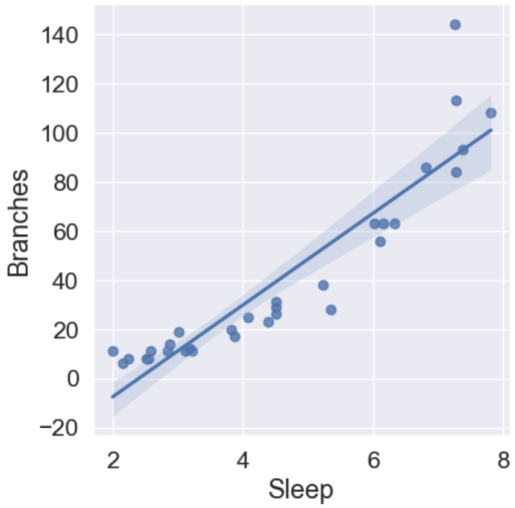
3 Beaver Bedtimes [9 Pts]

Brenda Beaver is building a dam, and she has been waking up earlier and earlier in the morning trying to build it faster. However, her friend Olivia Otter thinks Brenda is overworking herself, and that her dam production is suffering as a result.

To convince Brenda to get more sleep, Olivia created a DataFrame called `production`, shown below, with one row for each of the 31 days in the last month. For each day, Olivia recorded the amount of sleep Brenda got (in hours) and how many branches Brenda was able to add to her dam on that day. The first five rows of the DataFrame look like the left figure below:

	Sleep	Branches
0	4.5	29
1	6.3	63
2	2.0	11
3	3.8	20
4	2.9	14

`production`



Olivia's scatter plot

(a) [2 Pts] To help Olivia visualize this dataset, **write one line of code to create a scatter plot** of the two variables, with `Sleep` on the horizontal axis and `Branches` on the vertical axis. The scatter plot should also show a least-squares regression line and look similar to the plot above on the right.

```
import seaborn as sns
import matplotlib.pyplot as plt
```

Solution:

```
sns.lmplot(x = "Sleep",
y = "Branches",
data = production, fit_reg = True)
```

(b) [2 Pts] Olivia wants to know how the linear model performs on the data set. She calls the `Sleep` column x to remind herself it is the predictor variable, and calls the `Branches` column y to remind herself it is the response variable. Looking at the regression line from (a), **fill in the blanks by selecting the appropriate choices.**

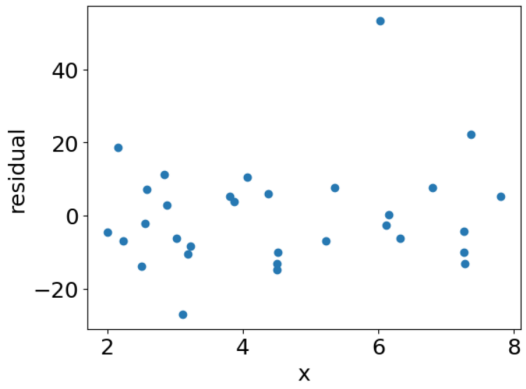
For small values of x, our linear model tends to _____(1)_____, y, and for large values of x, our linear model tends to _____(2)_____ y.

(1)
- ◯ **Underpredict**
- ◯ Overpredict

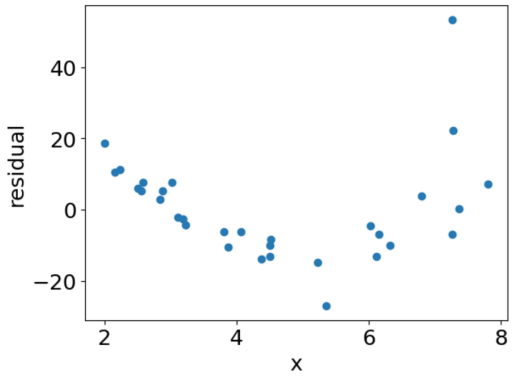
(2)
- ◯ **Underpredict**
- ◯ Overpredict

Solution: We ended up giving everyone points for the second blank because it can be argued that the "large values" are from 6 to 8, where the number of points above and below the lines are the same.

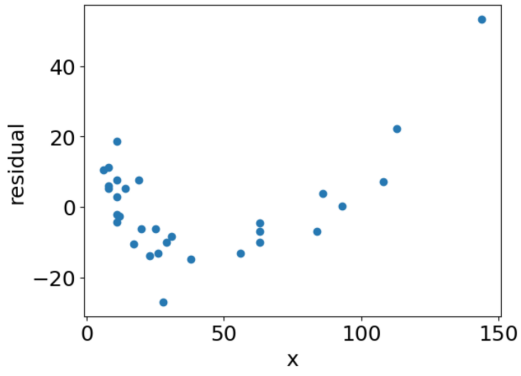
(c) [1 Pt] Olivia makes some more visualizations to evaluate the linear model. She makes a plot of the residuals $y - \hat{y}$ on the vertical axis against the predictor x on the horizontal axis. **Which of the plots below best matches her plot?**



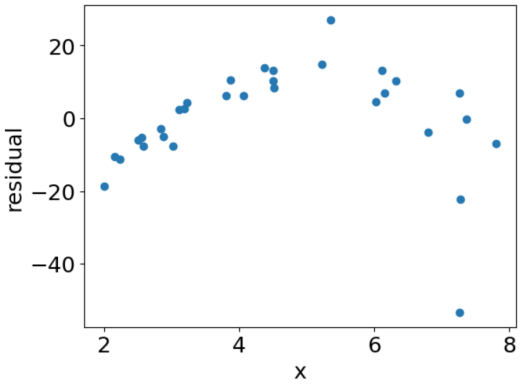
A



B



C



D

◯ A

◯ **B**

◯ C

◯ D

(d) [1 Pt] Olivia is thinking about using a transformation to make her model fit better. Which one transformation, from the following options, do you think is **most likely** to improve the fit?

◯ **Change the response variable to** $\log($`Branches`$)$

◯ Change the predictor variable to $\log($`Sleep`$)$

◯ Change the response variable to `Branches`^{2}

◯ Change the predictor variable to $60 \times$ `Sleep` (so sleep is measured in minutes)

Solution: The shape of the scatter plot resembles the bottom right of the Tukey-Mosteller Bulge Diagram, so we can apply any of the four transformations indicated there: $\sqrt{y}$, $\log(y)$, x^2, or x^3. The only option that fits is A.

(e) [2 Pts] Regardless of what you chose before, assume that Olivia comes to believe that the logarithms of x (`Sleep`) and y (`Branches`) are related, so she should actually use the prediction function

$$f_\theta(x) = \theta_0 + \theta_1 \log(x),$$

where $f_\theta(x)$ is the predicted value for $\log(y)$ (both logarithms are base e).

What is the prediction for y as a function of x, in terms of θ_0 and θ_1?

◯ $\hat{y} = e^{\theta_0} + x^{\theta_1}$

◯ $\hat{y} = e^{\theta_0} \cdot x^{\theta_1}$

◯ $\hat{y} = e^{\theta_0} \cdot \theta_1 x$

◯ $\hat{y} = e^{\theta_0} \cdot (\theta_1)^x$

◯ $\hat{y} = e^{\theta_0} + (\theta_1)^x$

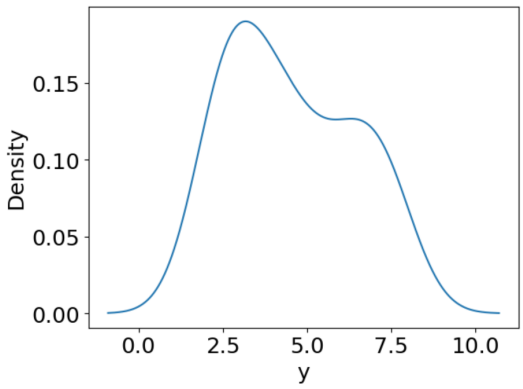
Solution: We have

$$\log(y) = \theta_0 + \theta_1 \log(x)$$

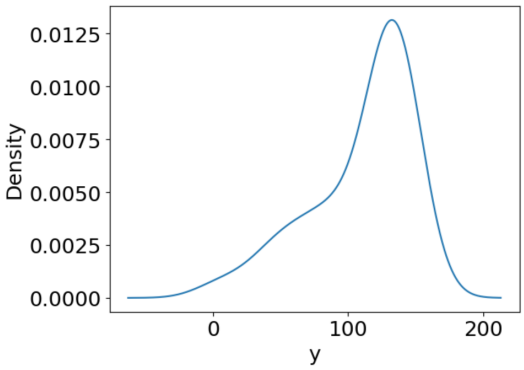
Raising e to the power of both sides:

$$\begin{aligned} e^{\log(y)} &= e^{\theta_0 + \theta_1 \log(x)} \\ y &= e^{\theta_0} \cdot e^{\theta_1 \log(x)} \\ y &= e^{\theta_0} \cdot e^{\log(x^{\theta_1})} \\ y &= e^{\theta_0} \cdot x^{\theta_1}. \end{aligned}$$

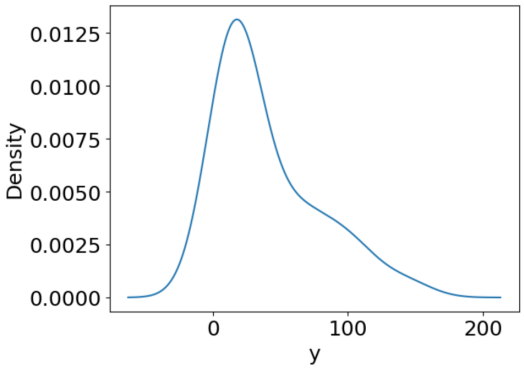
(f) [1 Pt] Olivia wants to take a closer look at the distribution of y, the `Branches` column. Which of the Kernel Density Plots best matches the distribution of y?



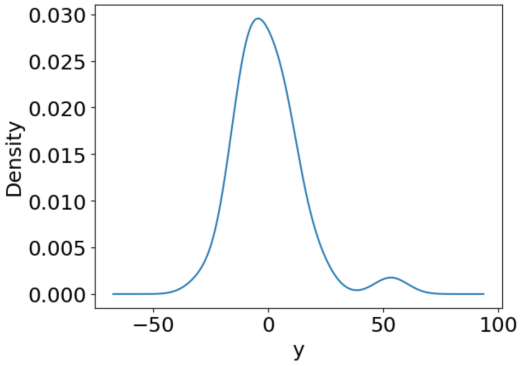
A



B



C



D

- ◯ A
- ◯ B
- ◯ **C**
- ◯ D

4 Booo-ba [11 Pts]

Casper, a ghost who loves milk tea boba, and Winnie, a witch who loves fruit tea boba, are partners in UC Berkeley's Boo 100 class. As a part of their class, they must conduct a research project that addresses some question they have about the UC Berkeley student population.

They are interested in whether UC Berkeley students prefer milk tea boba or fruit tea boba. They decide to collect a sample by publicly posting a poll on Reddit, an anonymous online forum that anyone can visit. Winnie posts the following question for the people on the forum to respond to:

Who thinks Fruit tea is better than Milk tea?

u/GoinGhost247

Hi all. My friend and I are interested in what the tea flavor preferences are throughout the school. It would be a great help if you could respond to the poll! Optionally, please feel free to also comment responding to the question!

Poll

- Fruit tea
- Milk tea

Question: Why Fruit / Milk tea and which flavors are your favorite?

Comments ...

After responding to the poll, users may optionally leave a comment, and other users can optionally give the comment "upvotes" or "downvotes" depending on whether they like or dislike the comment. Casper and Winnie store the data from their Reddit poll in a DataFrame, `responses`. For brevity, only the first few rows of the DataFrame are shown.

	Preference	Comment
0	Fruit	u/FruityBox: 'I love Fruit tea because its refreshing!' vote: 2
1	Milk	u/MilkyDay: 'Milk tea because taro is my favorite. Taro tea for life!' vote: 12
2	Fruit	u/Kulture: 'Did you know that Steave jobs has Sigma?' vote: -2
3	Milk	u/CluelessAngel1999: 'no tears left to cry.' vote: 0
4	Fruit	u/LemonTree: 'I had 7 orange fruit tea today. What is wrong with me?' vote: 9
5	Milk	u/ALLCAPSKING: 'I LOVE MILK TEA!' vote: 8
6	Milk	None

`responses`

The `Comment` column has a text string in three parts: the commenter's user name in the format `u/username`, the actual comment in single quotes, and the "net vote," upvotes minus downvotes, the comment got (so 5 upvotes and 7 downvotes results in `"vote: -2"` at the end). 30% of respondents did not comment, and have a value of `None` in the `Comment` column.

Note on edge cases: For all parts of this question dealing with regular expressions, you **don't** need to consider any edge cases beyond the example comments shown in the first six rows of `responses`. As long as your regex pattern works on the `Comment` strings shown in the DataFrame, and it is not trivializing the problem, you will get full credit.

(a) [1 Pt] What is their population of interest?

- ◯ All students in Boo 100
- ◯ **All students at UC Berkeley**
- ◯ All students at UC Berkeley on Reddit
- ◯ All people on Reddit

(b) [1 Pt] Is their sampling frame the same as their population of interest?

- ◯ Yes
- ◯ **No**

(c) [1 Pt] Assuming we want to learn more about users' boba preferences, which one of the following is a valid way of handling the missing data?

- ◯ **Fill all missing comments with the empty string ""**
- ◯ Delete all rows with a None value

(d) [2 Pts] First, they want to extract the comments (without the usernames or votes) and store them in a new column called `Text`. **Write a Regex pattern to extract only the comments, enclosed in, but not including, the single quotes**.

```
text_pattern = r"_____"

responses["Text"] = responses["Comment"].str.extract(text_pattern)
```

Solution:

```
text_pattern = r"'([^']+)'"
```

or

```
text_pattern = r"'(.*)'"
```

(e) [2 Pts] To assess each comment's popularity, they want to extract the net vote from the `Comment` column. They write a Regex pattern to create a new column called `Vote` with the net vote that each comment got. There is a space between `vote:` and the number. **Which of the following options will correctly extract the number of votes? Select all that apply.**

```
vote_pattern = r"_____"

responses["Vote"] = responses["Comment"].str.extract(vote_pattern)
```

☐ `vote_pattern = r"\s([-\d]+)"`

☐ **`vote_pattern = r"vote:\s([-0-9]+)"`**

☐ **`vote_pattern = r"([-\d]+)$"`**

☐ `vote_pattern = r"vote:\s([\d]+)"`

Solution: The first option is wrong because it will capture digits/numbers in usernames or the comment texts: the 4th row will give 7 not 9.

The second and third options work.

The fourth option does not work because it does not match negative numbers—the 2nd row will have no match.

(f) [4 Pts] After creating the `Text` column in part e), Casper and Winnie decide to drop the `Comment` column. Additionally, they decide to limit their dataset to only rows with valid text strings. In other words, you may assume there are no `None` values in the `Text` column.

The first few rows of `responses` now look like the following:

	Preference	Vote	Text
0	Fruit	2	I love Fruit tea because its refreshing!
1	Milk	12	Milk tea because taro is my favorite. Taro tea for life!
2	Fruit	-2	Did you know that Steave jobs has Sigma?
3	Milk	0	no tears left to cry.
4	Fruit	9	I had 7 orange fruit tea today. What is wrong with me?
5	Milk	8	I LOVE MILK TEA!

`responses` after processing

Casper and Winnie realize that not all comments answer the original question they posed. Naturally, they wants to filter out the rows that do not have a relevant comment. Here, a "relevant comment" is one that contains the word "tea", but not as a substring of another word ("tears" and "teapot" do not count). It should match regardless of cases (e.g. "Tea", "tEa", "TEA", and "teA" all count as the word "tea").

Complete the following lines of code so that `responses` now contains only the rows with relevant comments (i.e. rows 2 and 3 should be filtered out).

Hint: `valid_rows` should be a boolean series indicating if each row contains a relevant comment.

```
valid_rows = responses["Text"]_____A_____
```

```
responses = _____A_____B_____
```

i. [3 Pts] What goes in the blank indicated by the letter A?

ii. [1 Pt] What goes in the blank indicated by the letter B?

Solution:

```
valid_rows = responses["Text"].str.lower()
                .str.contains(r"[^\w]tea[^\w]")
responses = responses[valid_rows]
```

5 A lucky strike [11 Pts]

We have discovered rich stores of the valuable and rare mineral unobtanium. There are two different types of unobtanium: unobtanium-A, which is stable, and unobtanium-B, which is highly unstable. We carry out an experiment by starting a reaction that has no effect on unobtanium-A, but causes unobtanium-B to rapidly degrade with a "half-life" of γ milliseconds (ms) as soon as the reaction starts. That is, only half of the initial amount of unobtanium-B remains after γ ms, only one quarter remains after 2γ ms, and so on. If β_0 is the initial amount of unobtanium-A, and β_1 is the initial amount of unobtanium-B, then the total amount of unobtanium remaining after x milliseconds is given by $f_\theta(x)$ the formula:

$$f_\theta(x) = \beta_0 + \beta_1 2^{-x/\gamma}$$

Here $\theta = (\beta_0, \beta_1, \gamma)$, encoding all three model parameters.

We start the reaction, and at ten-millisecond intervals we take a noisy measurement of the total amount of unobtanium remaining. This gives us a data set with five data points, shown in the table below. x is the time since the reaction began, measured in milliseconds (ms), and y is the measured amount of unobtanium remaining x ms after the reaction begins, measured in milligrams (mg).

x	y
0	19
10	13
20	10
30	5
40	5

Unobtanium reaction data

For easy reference, the first four negative powers of 2 are given below:

$$2^{-1} = \frac{1}{2} = 0.5 \qquad 2^{-3} = \frac{1}{8} = 0.125$$

$$2^{-2} = \frac{1}{4} = 0.25 \qquad 2^{-4} = \frac{1}{16} = 0.0625$$

(a) [3 Pts] From briefly inspecting the data, we come up with a rough guess that the parameters are close to $\beta_0 = 4$, $\beta_1 = 16$, and $\gamma = 10$. **What is the MSE** for these parameters, in units of mg^2? Remember that the mean squared error is defined as

$$\text{MSE}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

where n is the number of observations, and $\hat{y}_i = f_\theta(x_i)$ is the prediction we would have made for y_i if we used the parameters $\theta = (\beta_0, \beta_1, \gamma)$.

- ◯ 0.2
- ◯ 0.6
- ◯ 1
- ◯ **1.4**
- ◯ 2

You will get full credit for choosing the right answer, but you can show your work in the box below if you want to get partial credit in case your answer is wrong:

Solution: The predictions would be $\hat{y} = (20, 12, 8, 6, 5)$, so the residuals are $e = (-1, 1, 2, -1, 0)$. The mean squared error is $(1 + 1 + 4 + 1 + 0)/5 = 7/5 = 1.4\,\text{mg}^2$.

(b) [3 Pts] Because our formula is not a linear model, we cannot use our closed-form expression for the MSE-minimizing model parameters. Instead, we could use gradient descent to search for the optimal model parameters, starting at our guess $\beta_0^{(0)} = 4, \beta_1^{(0)} = 16$, and $\gamma^{(0)} = 10$. **What will be the value** of $\beta_0^{(1)}$, the intercept parameter after one step of gradient descent? Assume we use learning rate $\alpha = 0.5$.

Hint: You will save some time if you don't calculate more derivatives than you need to.

Solution: Again, the residuals are $e = (-1, 1, 2, -1, 0)$, and we can calculate the gradient by taking partial derivatives:

$$\begin{aligned}\frac{\partial}{\partial \beta_0} \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 e^{-\beta_2 x_i})^2 &= \frac{-2}{n} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 e^{-\beta_2 x_i}) \\ &= \frac{-2}{n} \sum_{i=1}^{n} e_i \\ &= \frac{-2}{5}\end{aligned}$$

If we use the learning rate 0.5, then $\beta_0^{(1)} = \beta_0^{(0)} - 0.5(-\frac{2}{5}) = 4.2$.

(c) [2 Pts] Now suppose that, before we fit our model, we learn that another research group has discovered that the half-life of unstable unobtanium is exactly ten milliseconds ($\gamma = 10$). Now, instead of estimating γ, we can just plug in $\gamma = 10$. Better still, we don't have to do gradient descent anymore; we can estimate $\beta = (\beta_0, \beta_1)$ using linear regression.

If we construct the right matrix $\mathbb{X}$, we will get a closed form for our estimate:

$$\hat{\beta} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{Y},$$

where $\mathbb{Y}$ is the column vector (or $n \times 1$ matrix) representing the response. Which of the following choices is the correct matrix $\mathbb{X}$?

◯

0
10
20
30
40

◯

1	0
1	10
1	20
1	30
1	40

◯

1	0	0
1	10	100
1	20	400
1	30	900
1	40	1600

◯

0	1
10	0.5
20	0.25
30	0.125
40	0.0625

◯

1	**1**
1	**0.5**
1	**0.25**
1	**0.125**
1	**0.0625**

◯

1	0	1
1	10	0.5
1	20	0.25
1	30	0.125
1	40	0.0625

Solution: With $\gamma = 10$, our model is $f_\theta(x) = \beta_0 + \beta_1 2^{-x/10}$. Our design matrix would therefore have two columns: one bias/intercept column of all ones, one column with our feature $2^{-x/10} : \{2^0, 2^{-1}, \ldots, 2^{-4}\}$.

(d) [3 Pts] Continue to assume that we know $\gamma = 10$ so we are only estimating β_0 and β_1. Even before we estimate the model, **what can we be sure will be true** about the fitted values $\hat{y}$ and the residuals $e = y - \hat{y}$ for the estimated model? **Select all that apply.**

Note: In the following options, x represents time in milliseconds.

- ☐ If we plot the residuals e against the time variable x, we will not see any clear pattern in the plot.
- ☐ **The residuals will add up to zero.**
- ☐ The residuals will be uncorrelated with x.
- ☐ The residuals will be highly correlated with the fitted values $\hat{y}$.

☐ The residuals in the left half of the plot will not be as spread out as the ones in the right half of the plot.

☐ **The fitted values $\hat{y}$ will have smaller variance than the original y values.**

Solution: Again, our model is $f_\theta(x) = \beta_0 + \beta_1 2^{-x/10}$. Note the only feature is $2^{-x/10}$ and the time x itself is not a feature.

Option A is wrong because x is not a feature; we can't be sure if there will be a pattern in the residual plot. Even if x is a feature, there will still be patterns in the residual plot if the relationship between the response and the predictor is not linear.

Option B is a fact from OLS.

Option C is wrong. It is a fact that in OLS the residuals are uncorrelated with the predictor variables (proved in HW6), but here the predictor variable is not x (it's $2^{-x/10}$) so we can't be sure about the correlation between the residuals and x.

Option D is wrong. Since the fitted values $\hat{y}$ are in the column space of the design matrix, the residuals will be orthogonal to the fitted values and therefore uncorrelated.

Option E is wrong. We can't be sure of the shape and spread of the residual plot before fitting the model.

Option F is correct. We showed that the variance of the original y values are the sum of the variance of the fitted values $\hat{y}$ and the variance of the residuals ($\sigma_y^2 = \sigma_{\hat{y}}^2 + \sigma_e^2$). Therefore, $\sigma_y^2 > \sigma_{\hat{y}}^2$.

You are done with the Midterm! Congratulations!

Use this page to draw your Halloween costume!

Fall 2022 Data C100/C200 Midterm Reference Sheet

Pandas

Suppose `df` is a DataFrame; `s` is a Series. `pd` is the Pandas package.

Function	Description
`df[col]`	Returns the column labeled `col` from `df` as a Series.
`df[[col1, col2]]`	Returns a DataFrame containing the columns labeled `col1` and `col2`.
`s.loc[rows] / df.loc[rows, cols]`	Returns a Series/DataFrame with rows (and columns) selected by their index values.
`s.iloc[rows] / df.iloc[rows, cols]`	Returns a Series/DataFrame with rows (and columns) selected by their positions.
`s.isnull() / df.isnull()`	Returns boolean Series/DataFrame identifying missing values
`s.fillna(value) / df.fillna(value)`	Returns a Series/DataFrame where missing values are replaced by `value`
`df.drop(labels, axis)`	Returns a DataFrame without the rows or columns named `labels` along `axis` (either 0 or 1)
`df.rename(index=None, columns=None)`	Returns a DataFrame with renamed columns from a dictionary `index` and/or `columns`
`df.sort_values(by, ascending=True)`	Returns a DataFrame where rows are sorted by the values in columns `by`
`s.sort_values(ascending=True)`	Returns a sorted Series.
`s.unique()`	Returns a NumPy array of the unique values
`s.value_counts()`	Returns the number of times each unique value appears in a Series
`pd.merge(left, right, how='inner', on='a')`	Returns a DataFrame joining DataFrames `left` and `right` on the column labeled `a`; the join is of type `inner`
`left.merge(right, left_on=col1, right_on=col2)`	Returns a DataFrame joining DataFrames `left` and `right` on columns labeled `col1` and `col2`.
`df.pivot_table(index, columns, values=None, aggfunc='mean')`	Returns a DataFrame pivot table where columns are unique values from `columns` (column name or list), and rows are unique values from `index` (column name or list); cells are collected `values` using `aggfunc`. If `values` is not provided, cells are collected for each remaining column with multi-level column indexing.
`df.set_index(col)`	Returns a DataFrame that uses the values in the column labeled `col` as the row index.
`df.reset_index()`	Returns a DataFrame that has row index 0, 1, etc., and adds the current index as a column.
`s.astype(dtype)`	Converts the Series `s` to of data type `dtype` in place.

Let `grouped = df.groupby(by)` where `by` can be a column label or a list of labels.

Function	Description
`grouped.count()`	Return a Series/DataFrame containing the size of each group, excluding missing values
`grouped.size()`	Return a Series containing size of each group, including missing values
`grouped.mean()`/`grouped.min()`/`grouped.max()`	Return a Series/DataFrame containing mean/min/max of each group for each column, excluding missing values
`grouped.filter(f)` `grouped.agg(f)`	Filters or aggregates using the given function f

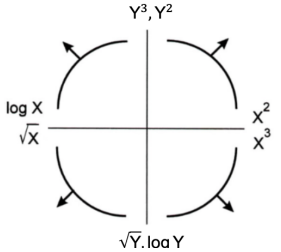
Function	Description
`s.str.len()`	Returns a Series containing length of each string
`s.str.lower()`/`s.str.upper()`	Returns a Series containing lowercase/uppercase version of each string
`s.str.replace(pat, repl)`	Returns a Series after replacing occurences of substrings matching regular expression `pat` with string `repl`
`s.str.contains(pat)`	Returns a boolean Series indicating whether a substring matching the regular expression `pat` is contained in each string
`s.str.extract(pat)`	Returns a Series of the first subsequence of each string that matches the regular expression `pat`. If `pat` contains one group, then only the substring matching the group is extracted

Visualization

Matplotlib: `x` and `y` are sequences of values.

Function	Description
`plt.plot(x, y)`	Creates a line plot of `x` against `y`
`plt.scatter(x, y)`	Creates a scatter plot of `x` against `y`
`plt.hist(x, bins=None)`	Creates a histogram of `x`; `bins` can be an integer or a sequence
`plt.bar(x, height)`	Creates a bar plot of categories `x` and corresponding heights `height`

Tukey-Mosteller Bulge Diagram.



Seaborn: `x` and `y` are column names in a DataFrame `data`.

Function	Description
`sns.countplot(data, x)`	Create a barplot of value counts of variable `x` from `data`
`sns.histplot(data, x, kde=False)` `sns.displot(x, data, rug = True, kde = True)`	Creates a histogram of `x` from `data`; optionally overlay a kernel density estimator. `displot` is similar but can optionally overlay a rug plot.
`sns.boxplot(data, x=None, y)` `sns.violinplot(data, x=None, y)`	Create a boxplot of y, optionally factoring by categorical `x`, from `data`. `violinplot` is similar but also draws a kernel density estimator of `y`.
`sns.scatterplot(data, x, y)`	Create a scatterplot of `x` versus `y` from `data`
`sns.lmplot(x, y, data, fit_reg=True)`	Create a scatterplot of `x` versus `y` from `data`, and by default overlay a least-squares regression line
`sns.jointplot(x, y, data, kind)`	Combine a bivariate scatterplot of `x` versus `y` from `data`, with univariate density plots of each variable overlaid on the axes; `kind` determines the visualization type for the distribution plot, can be `scatter`, `kde` or `hist`

Regular Expressions

List of all metacharacters: `. ^ $ * + ? ] [ \ | ( ) { }`

Operator	Description	Operator	Description
`.`	Matches any character except **\n**	`*`	Matches preceding character/group zero or more times
`\\`	Escapes metacharacters	`?`	Matches preceding character/group zero or one times
`|`	Matches expression on either side of expression; has lowest priority of any operator	`+`	Matches preceding character/group one or more times
`\d`, `\w`, `\s`	Predefined character group of digits (0-9), alphanumerics (a-z, A-Z, 0-9, and underscore), or whitespace, respectively	`^`, `$`	Matches the beginning and end of the line, respectively
`\D`, `\W`, `\S`	Inverse sets of `\d`, `\w`, `\s`, respectively	`( )`	Capturing group used to create a sub-expression
`{m}`	Matches preceding character/group exactly `m` times	`[ ]`	Character class used to match any of the specified characters or range (e.g. `[abcde]` is equivalent to `[a-e]`)
`{m, n}`	Matches preceding character/group at least `m` times and at most `n` times if either `m` or `n` are omitted, set lower/upper bounds to 0 and ∞, respectively	`[^ ]`	Invert character class; e.g. `[^a-c]` matches all characters except `a`, `b`, `c`

Function	Description
`re.match(pattern, string)`	Returns a match if zero or more characters at beginning of `string` matches `pattern`, else None
`re.search(pattern, string)`	Returns a match if zero or more characters anywhere in `string` matches `pattern`, else None
`re.findall(pattern, string)`	Returns a list of all non-overlapping matches of `pattern` in `string` (if none, returns empty list)
`re.sub(pattern, repl, string)`	Returns `string` after replacing all occurrences of `pattern` with `repl`

Modified lecture example for a single capturing group:

```
lines = '169.237.46.168 - - [26/Jan/2014:10:47:58 -0800] "GET ... HTTP/1.1"'
re.findall(r'\[\d+\/(\w+)\/\d+:\d+:\d+:\d+ .+\]', line) # returns ['Jan']
```

Modeling

Concept	Formula	Concept	Formula
Variance σ_x^2	$\frac{1}{n}\sum_i^n (x_i - \bar{x})^2$	Correlation r	$r = \frac{1}{n}\sum_{i=1}^n \frac{x_i - \bar{x}}{\sigma_x}\frac{y_i - \bar{y}}{\sigma_y}$
L_1 loss	$L_1(y, \hat{y}) = \mid y - \hat{y} \mid$	Linear regression prediction of y	$\hat{y} = a + bx$
L_2 loss	$L_2(y, \hat{y}) = (y - \hat{y})^2$	Least squares linear regression, slope $\hat{b}$	$\hat{b} = r\frac{\sigma_y}{\sigma_x}$
Empirical risk with loss L	$R(\theta) = \frac{1}{n}\sum_{i=1}^n L(y_i, \hat{y}_i)$	Least squares linear regression, intercept $\hat{a}$	$\hat{a} = \bar{y} - \hat{b}\bar{x}$

Ordinary Least Squares

Multiple Linear Regression Model: $\hat{\mathbb{Y}} = \mathbb{X}\theta$ with design matrix $\mathbb{X}$, response vector $\mathbb{Y}$, and predicted vector $\hat{\mathbb{Y}}$. If there are d features plus a bias/intercept, then the vector of parameters $\theta = [\theta_0, \theta_1, \ldots, \theta_d]^T \in \mathbb{R}^{d+1}$. The vector of estimates $\hat{\theta}$ is obtained from fitting the model to the sample $(\mathbb{X}, \mathbb{Y})$.

Concept	Formula	Concept	Formula
Mean squared error	$R(\theta) = \frac{1}{n}\|\|\mathbb{Y} - \mathbb{X}\theta\|\|_2^2$	Normal equation	$\mathbb{X}^T\mathbb{X}\hat{\theta} = \mathbb{X}^T\mathbb{Y}$
Least squares estimate, if $\mathbb{X}$ is full rank	$\hat{\theta} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{Y}$	Residual vector, e	$e = \mathbb{Y} - \hat{\mathbb{Y}}$
		Multiple R^2 (coefficient of determination)	$R^2 = \frac{\text{variance of fitted values}}{\text{variance of } y}$
Ridge Regression L2 Regularization	$\frac{1}{n}\|\|\mathbb{Y} - \mathbb{X}\theta\|\|_2^2 + \alpha\sum_{i=1}^d \theta_i^2$	Ridge Regression (constrained form)	$\frac{1}{n}\|\|\mathbb{Y} - \mathbb{X}\theta\|\|_2^2$ such that $\sum_{i=1}^d \theta_i^2 \leq Q$
Ridge regression estimate (closed form)	$\hat{\theta}_{\text{ridge}} = (\mathbb{X}^T\mathbb{X} + n\alpha I)^{-1}\mathbb{X}^T\mathbb{Y}$		
LASSO Regression L1 Regularization	$\frac{1}{n}\|\|\mathbb{Y} - \mathbb{X}\theta\|\|_2^2 + \alpha\sum_{i=1}^d \|\theta_i\|$	LASSO Regression (constrained form)	$\frac{1}{n}\|\|\mathbb{Y} - \mathbb{X}\theta\|\|_2^2$ such that $\sum_{i=1}^d \|\theta_i\| \leq Q$

Scikit-Learn

Suppose `sklearn.model_selection` and `sklearn.linear_model` are both imported packages.

Package	Function(s)	Description
`sklearn.linear_model`	`LinearRegression(fit_intercept=True)`	Returns an ordinary least squares Linear Regression model.
	`LassoCV(fit_intercept=True)`, `RidgeCV(fit_intercept=True)`	Returns a Lasso (L1 Regularization) or Ridge (L2 regularization) linear model, respectively, and picks the best model by cross validation.
	`model.fit(X, y)`	Fits the scikit-learn `model` to the provided `X` and `y`.
	`model.predict(X)`	Returns predictions for the X passed in according to the fitted `model`.
	`model.coef_`	Estimated coefficients for the linear model, not including the intercept term.
	`model.intercept_`	Bias/intercept term of the linear model. Set to 0.0 if `fit_intercept=False`.
`sklearn.model_selection`	`train_test_split(*arrays, test_size=0.2)`	Returns two random subsets of each array passed in, with 0.8 of the array in the first subset and 0.2 in the second subset.