

For our tutorial assignment, my instructor writes: "For this tutorial, your working directory will be set to the location of your Week 7 materials (i.e., the file path for your local copy of the Week 7 GitHub repo). Specifically, we will work in the data folder. Note that the working directory convention differs for Windows vs. Mac and Linux. My machine is a Mac so the code in my notebook looks like this:

```
os.chdir("/Users/obn/Documents/GitHub/UM-  
BGEN632/week7labs/data")
```

Add your desired file path within the parentheses of the `os.chdir()` function in the code cell below and then run it." Then I entered "`os.chdir()` # change the directory
`os.getcwd()` # confirm the change

```
os.chdir("/Users/AlexMcKenzie/Library/CloudStorage/Go  
ogleDrive-alex.c.mckenzie@gmail.com/My  
Drive/Education/UM Graduate School/BGEN632/Week  
7/data")", but I got the following error message: "-----  
-----"
```

```
TypeError                                Traceback (most recent  
call last)
```

```
Cell In[3], line 1
```

```
----> 1 os.chdir() # change the directory  
      2 os.getcwd() # confirm the change  
      3
```

```
os.chdir("/Users/AlexMcKenzie/Library/CloudStorage/Go  
ogleDrive-alex.c.mckenzie@gmail.com/My  
Drive/Education/UM Graduate School/BGEN632/Week  
7/data")
```

```
TypeError: chdir() missing required argument 'path' (pos  
1)". How am I missing the "path" argument?
```

Reasoned about `os.chdir()` error for 13 seconds >

The error arises because you called the `os.chdir()` function without providing any argument in the first call. The `os.chdir()` function requires a file path string as an argument to change the current working directory. In your code, the first line:

```
python
```



```
os.chdir() # change the directory
```

is missing the path argument, which causes the `TypeError: "chdir() missing required argument 'path' (pos 1)".` You only need to call `os.chdir()` once with the correct file path.

To fix this, remove the empty call and only use the version with your desired file path.

For example:

python



```
import os # Change the directory by providing the file path as an
argument
os.chdir("/Users/AlexMcKenzie/Library/CloudStorage/GoogleDrive-
alex.c.mckenzie@gmail.com/My Drive/Education/UM Graduate
School/BGEN632/Week 7/data") # Confirm that the directory change was
successful print(os.getcwd())
```

This code first changes the working directory to the specified path and then prints the current working directory so you can confirm the change.

Make sure that the path you provide is correct and that it exists on your system. This should resolve the error you encountered.



what does "axis = 1" mean in this code?

```
"loblolly_mod_2.drop(loblolly_mod_2.columns[[2, 3]],
axis = 1)"
```

Reasoned for 8 seconds >

In pandas, the parameter `axis` tells the function whether to operate on rows (`axis=0`) or columns (`axis=1`). In your code:

python



```
loblolly_mod_2.drop(loblolly_mod_2.columns[[2, 3]], axis=1)
```

the argument `axis=1` indicates that the drop operation should be applied to columns. Here's what's happening step by step:

1. Selecting Columns:

`loblolly_mod_2.columns[[2, 3]]` selects the column labels at positions 2 and 3 (remember, pandas uses 0-based indexing, so these are the third and fourth columns).

2. Dropping the Columns:

The `drop()` function is then called with these column labels and the argument `axis=1`, which tells pandas to remove those columns from the DataFrame.

If you had used `axis=0`, pandas would have tried to drop rows with labels corresponding to those positions instead of columns.

So in summary, `axis=1` in this code specifies that the operation should target columns rather than rows.



but isn't that redundant to
"`loblolly_mod_2.drop(loblolly_mod_2.columns[[2, 3]]`"
where 'columns' is already identified?

Reasoned about redundancy in pandas code for 12 seconds >

The key point is that while you're selecting column labels from the DataFrame with `loblolly_mod_2.columns[[2, 3]]`, the `drop()` function by default operates on rows (i.e., `axis=0`). Specifying `axis=1` tells pandas that you intend to drop columns, not rows. Even though you're fetching column labels, without `axis=1` pandas would look for those labels in the row index, which could lead to unexpected results or errors. Essentially, the `axis` parameter is not redundant—it's necessary to ensure the operation is performed on the columns.



I am trying to load a CSV file, and used this command:
"`github_teams =`
`pd.read_csv(/Users/AlexMcKenzie/Library/CloudStorage`

```
/GoogleDrive-alex.c.mckenzie@gmail.com/My
Drive/Education/UM Graduate School/BGEN632/Week
7/data/github_teams.csv)". I got this error: "Cell In[1],
line 5
    github_teams =
pd.read_csv(/Users/AlexMcKenzie/Library/CloudStorage
/GoogleDrive-alex.c.mckenzie@gmail.com/My
Drive/Education/UM Graduate School/BGEN632/Week
7/data/github_teams.csv)
    ^
SyntaxError: invalid syntax". Note the file path is correct.
```

Reasoned about CSV file loading error for 10 seconds >

The error arises because the file path isn't being passed as a string literal. In Python, file paths must be enclosed in quotes (either single or double) so that they are interpreted as strings. Without the quotes, Python tries to evaluate the file path as code, which leads to a syntax error.

To fix this, simply enclose your file path in quotes:

```
python                                                                    Copy

github_teams =
pd.read_csv("/Users/AlexMcKenzie/Library/CloudStorage/GoogleDrive-
alex.c.mckenzie@gmail.com/My Drive/Education/UM Graduate
School/BGEN632/Week 7/data/github_teams.csv")
```

This will let the `read_csv` function correctly interpret the argument as a string representing the file path.



remind me how i add in-line commentary to my python code? what is the stylization?

Reasoned about inline comments in Python for 8 seconds >

In Python, you can add an inline comment by using the hash symbol (`#`). When you place a `#` in your code, everything following it on the same line is treated as a comment and is ignored by the Python interpreter. For example:

```
python                                                                    Copy

x = 5 # This is an inline comment explaining that x is set to 5
```

This style of commentary is useful for adding brief explanations or notes within your code without affecting its execution. For multi-line commentary, you can use multiple

single-line comments or triple-quoted strings (though triple quotes are typically used for docstrings).



next i tried to get the names of the rows and columns from our data set using "github_teams.rows, github_teams.columns", but i got the following error message: "-----

```
-----
AttributeError                                Traceback (most recent
call last)
/var/folders/xt/1ps45zhn76962p4cclc2wk0h0000gn/T/ip
ykernel_33882/1581881266.py in ?()
----> 1 github_teams.rows, github_teams.columns
```

```
/opt/anaconda3/lib/python3.12/site-
packages/pandas/core/generic.py in ?(self, name)
    6295         and name not in self._accessors
    6296         and
self._info_axis._can_hold_identifiers_and_holds_name(
name)
    6297     ):
    6298         return self[name]
-> 6299     return object.__getattr__(self, name)
```

AttributeError: 'DataFrame' object has no attribute 'rows'.

Reasoned about pandas DataFrame attributes for 4 seconds

In pandas, DataFrames do not have a "rows" attribute. Instead, the row names or labels are stored in the DataFrame's index. To get the names of the rows and columns, you should use:

python



```
print(github_teams.index) # This prints the row labels (index)
print(github_teams.columns) # This prints the column labels
```

Using `github_teams.index` gives you the row names (or numerical index by default), while `github_teams.columns` gives you the column names. This avoids the `AttributeError` you encountered.



Ok, I got some results but don't like how it's organized visually: "RangeIndex(start=0, stop=608, step=1)
Index(['name_h', 'Team_type', 'Team_size_class',
'human_members_count',
'bot_members_count', 'human_work',

```
'work_per_human', 'human_gini',
    'human_Push', 'human_IssueComments',
    'human_PRRReviewComment',
    'human_MergedPR', 'bot_work', 'bot_Push',
    'bot_IssueComments',
    'bot_PRRReviewComment', 'bot_MergedPR',
    'eval_survival_day_median',
    'issues_count'],
    dtype='object')". Can I make it return this in clean
rows format?
```

Reasoned for 11 seconds >

You can convert the index and columns into Python lists or loop over them to get a cleaner, row-by-row display. Here are a couple of approaches:

1. Converting to a List

For a compact view, you can convert the index and columns to lists:

python



```
# Convert index and columns to lists
rows = github_teams.index.tolist()
columns = github_teams.columns.tolist()
print("Rows:", rows)
print("Columns:", columns)
```

This prints the rows and columns as Python lists, which can be easier to read than the default Index object representation.

2. Looping to Print Each Label on a New Line

If you'd like each row or column printed on a separate line, you can use a loop:

python



```
# Print each row label on a new line
print("Rows:")
for row in github_teams.index:
    print(row)
# Print each column label on a new line
print("\nColumns:")
for col in github_teams.columns:
    print(col)
```

This approach prints every row and column label individually, resulting in a more “clean rows” format for the output.

Either method will help you visualize the output in a more organized manner compared to the default Index and RangeIndex representations.



the results for #1 approach are unwieldy! "Rows: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,

66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95,
96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107,
108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119,
120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131,
132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153,
154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165,
166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177,
178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189,
190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200,
201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211,
212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222,
223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233,
234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244,
245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255,
256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266,
267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277,
278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288,
289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299,
300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310,
311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321,
322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332,
333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343,
344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354,
355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365,
366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376,
377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387,
388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398,
399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409,
410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420,
421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431,
432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442,
443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453,
454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464,
465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475,
476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486,
487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497,
498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508,
509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519,
520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530,
531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541,
542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552,
553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563,
564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574,
575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585,
586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596,
597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607]

Columns: ['name_h', 'Team_type', 'Team_size_class',
'human_members_count', 'bot_members_count',
'human_work', 'work_per_human', 'human_gini',

```
'human_Push', 'human_IssueComments',  
'human_PRRReviewComment', 'human_MergedPR',  
'bot_work', 'bot_Push', 'bot_IssueComments',  
'bot_PRRReviewComment', 'bot_MergedPR',  
'eval_survival_day_median', 'issues_count']" I like the  
"range" return for rows from the original output, while  
the column names are still without a line break after  
each columns. Revise.
```

Reasoned for 10 seconds >

You can print the rows as a RangeIndex and then loop through the columns to print each on its own line. For example:

python



```
# Rows are printed as a RangeIndex (which you like) print("Rows:",  
github_teams.index) # Loop over the columns and print each on a new  
line print("Columns:") for col in github_teams.columns: print(col)
```

When you run this code, you'll get output like:

vbnet



```
Rows: RangeIndex(start=0, stop=608, step=1)  
Columns:  
name_h  
Team_type  
Team_size_class  
human_members_count  
bot_members_count  
human_work  
work_per_human  
human_gini  
human_Push  
human_IssueComments  
human_PRRReviewComment  
human_MergedPR  
bot_work  
bot_Push  
bot_IssueComments  
bot_PRRReviewComment  
bot_MergedPR  
eval_survival_day_median  
issues_count
```

This keeps the nice RangeIndex output for the rows and prints each column name on its own line for a cleaner visual presentation.



is there a way to revise further so it just tells me the number of columns?

Reasoned about counting DataFrame columns for 6 seconds >

Yes, you can simply count the number of columns using either the length of the columns attribute or the shape of the DataFrame. For example:

python



```
# Using the length of the columns attribute print("Number of columns:",  
len(github_teams.columns)) # Alternatively, using the shape attribute  
(shape[1] gives number of columns) print("Number of columns:",  
github_teams.shape[1])
```

Either method will display just the number of columns in your DataFrame.



what about if i want number of rows and number of columns?

Reasoned for 6 seconds >

You can obtain both numbers using the DataFrame's shape attribute, which returns a tuple where the first element is the number of rows and the second element is the number of columns. For example:

python



```
# Get the shape tuple: (number of rows, number of columns) num_rows,  
num_columns = github_teams.shape print("Number of rows:", num_rows)  
print("Number of columns:", num_columns)
```

Another option is to use the length of the index for rows and the length of the columns for columns:

python



```
print("Number of rows:", len(github_teams.index)) print("Number of  
columns:", len(github_teams.columns))
```

Either method will give you the count of rows and columns in your DataFrame.



I'm supposed to answer: "How many unique values does Team_type have?

* How many unique values does Team_size_class have?". But my code only returned results for one:

```
"pd.unique(github_teams.Team_type)
pd.unique(github_teams.Team_size_class)"
```

Reasoned for 11 seconds >

To count the number of unique values instead of just listing them, you can use either Python's built-in `len()` function with `pd.unique()`, or use the pandas Series method `nunique()`. Here are both options for your DataFrame:

Option 1: Using `len()` and `pd.unique()`

python



```
unique_team_type_count = len(pd.unique(github_teams.Team_type))
unique_team_size_class_count =
len(pd.unique(github_teams.Team_size_class)) print("Number of unique
values in 'Team_type':", unique_team_type_count) print("Number of
unique values in 'Team_size_class':", unique_team_size_class_count)
```

Option 2: Using `nunique()`

python



```
unique_team_type_count = github_teams.Team_type.nunique()
unique_team_size_class_count = github_teams.Team_size_class.nunique()
print("Number of unique values in 'Team_type':",
unique_team_type_count) print("Number of unique values in
'Team_size_class':", unique_team_size_class_count)
```

Both approaches will print the count of unique values for `Team_type` and `Team_size_class`. This should help you answer the question correctly.



remind me what len does?

Reasoned about the `len()` function for 6 seconds >

The `len()` function in Python returns the number of items in an object. For instance:

- If you pass a list to `len()`, it returns the number of elements in the list.
- If you use it on a string, it returns the number of characters in that string.
- If you apply it to a dictionary, it returns the number of keys in the dictionary.

Here's a quick example:

python



```
# For a list: my_list = [10, 20, 30, 40] print(len(my_list)) # Output: 4
# For a string: my_string = "hello" print(len(my_string)) # Output: 5
# For a dictionary: my_dict = {"a": 1, "b": 2, "c": 3}
print(len(my_dict)) # Output: 3
```

So, when you use `len(pd.unique(github_teams.Team_type))`, you're counting how many unique values there are in the `Team_type` column of your DataFrame.



for rows, I know my formulas need to operate on a "-1" syntax, wherein data from row 45 should be referenced as 44, due to data storage beginning with row 0. Is the same true for columns?

Reasoned about row and column indexing for 9 seconds >

Yes, the same rule applies to columns. Both rows and columns in pandas (and Python in general) are zero-indexed, which means:

- The first row or column is referenced as index 0.
- Therefore, when you want to refer to what might be called "row 45" in a human-readable format, you would use index 44.
- Similarly, for columns, if you want to reference the 5th column, you'll use index 4.

This zero-based indexing is consistent across all Python data structures, which is why you subtract one when converting from human-friendly numbers to Python indices.



so what should this return? `github_teams.iloc[45], [2,6]`

Reasoned about indexing in pandas for 25 seconds >

When you write:

```
python
```



```
github_teams.iloc[45], [2,6]
```

you're not performing a combined row-and-column selection. Instead, Python interprets this as two separate expressions separated by a comma. This results in a tuple whose first element is the result of `github_teams.iloc[45]` (which returns the entire row at index 45 as a Series) and whose second element is simply the list `[2,6]`.

If you intended to select row index 45 and only retrieve the columns at indices 2 and 6 from that row, you should combine them within the same indexing brackets, like so:

python



```
github_teams.iloc[45, [2,6]]
```

This revised code will return a Series containing only the values from columns with indices 2 and 6 in the specified row (remembering that indices are zero-based).



why isn't this working? "github_teams.iloc[45, 'human_members_count', 'human_gini']" Note that 'human_members_count' and 'human_gini' are column names.

Reasoned about index and column selection for 11 seconds >

The issue is that `.iloc` is for positional (integer index) selection only, and it doesn't accept column labels like `'human_members_count'` or `'human_gini'`. When you specify labels with `.iloc`, Python raises an error because it expects numeric indices.

How to Fix It

1. Using Label-Based Indexing:

If you want to select the row with index 45 and columns with labels `'human_members_count'` and `'human_gini'`, you should use `.loc` instead:

python



```
github_teams.loc[45, ['human_members_count', 'human_gini']]
```

This returns the values for the specified row and columns using the labels.

2. Using Positional Indexing:

Alternatively, if you want to use `.iloc` you need to provide the integer positions of the columns. For instance, if `'human_members_count'` is at position 3 and `'human_gini'` is at position 7 (remembering zero-based indexing), you would write:

python



```
github_teams.iloc[45, [3, 7]]
```

Make sure you confirm the correct column positions from your DataFrame.

Summary

- Use `.loc` for label-based indexing.
- Use `.iloc` for positional indexing only.

Choose the method that best fits your needs.



next i'm supposed to create a new data frame. I'm a bit lost, because in our tutorial, our instructor entered some new data, but here the challenge is worded slightly differently: "Create a new DataFrame for the column `bot_work` using two different methods." This sounds a bit different than what she demonstrated in her tutorial: "### Adding Rows

An important task we often engage in as we work with data in Python is appending new rows or adding new columns to a DataFrame. pandas provides a simple

approach.

First, create some new rows of data to populate the loblolly data. We are creating these rows as a new DataFrame so that we can easily combine it with the original loblolly DataFrame by using the `concat()` function. # some new old trees

```
new_rows = pd.DataFrame({'height': [71.22, 85.05, 68.34],  
                          'age': [30, 30, 30],  
                          'Seed': [400, 401, 402]},  
                          index = [84, 85, 86])
```

Second, we can concatenate our new DataFrame to the existing loblolly DataFrame. Save this as a new DataFrame object named `loblolly_mod_1`. The reason we do this is to create a historical trail. If we need access to the original DataFrame, we cannot overwrite it. # merge new trees with existing trees data
`loblolly_mod_1 = pd.concat([loblolly, new_rows], ignore_index = True)`". Help me understand the concept a bit better, because I'm a little lost on DataFrames.

****Reasoned about creating a new DataFrame**** for 23 seconds >

A DataFrame in pandas is essentially a table of data with rows and columns, each with associated labels (indexes for rows and names for columns). When you're asked to create a new DataFrame for a particular column (like `bot_work`), what you're doing is taking that single column from your existing DataFrame (a pandas Series) and turning it into its own DataFrame. This is useful if you want to work specifically with that column without affecting or using the entire original DataFrame.

There are a couple of common methods to do this:

Method 1: Using Double Bracket Notation

Pandas lets you select a subset of a DataFrame by using double brackets. This always returns another DataFrame, even if you're selecting just one column.

python



```
# This returns a new DataFrame with just the 'bot_work' column.  
bot_work_df = github_teams[['bot_work']]
```

Here, the double brackets tell pandas, "I want a DataFrame with the column 'bot_work'." Even though `github_teams['bot_work']` by itself would return a Series, wrapping it in an extra set of brackets preserves the DataFrame structure.

Method 2: Using the `.to_frame()` Method

When you select a single column from a DataFrame using single brackets, you get a Series. The Series object has a method `.to_frame()` that converts it into a DataFrame.

python



```
# This gets the 'bot_work' column as a Series and then converts it to a DataFrame. bot_work_df = github_teams['bot_work'].to_frame()
```

This method is functionally equivalent to the first one; it just goes through an intermediate step of creating a Series before converting it.

Why Create a New DataFrame for a Column?

Creating a new DataFrame for just one column can help in various scenarios:

- **Focused Analysis:** You might want to run analyses or visualizations only on that column.
- **Data Integrity:** Keeping a separate copy preserves the original DataFrame, ensuring that any modifications to the new DataFrame don't affect your original data.
- **Modularity:** It allows you to perform operations like merging or concatenating subsets of data in a controlled way, similar to creating a "historical trail" where you keep track of intermediate steps.

Both methods achieve the goal of isolating the `bot_work` column in a new DataFrame. Choose whichever method you find more intuitive—double bracket selection is very common for its simplicity, whereas `.to_frame()` makes it explicitly clear that you're converting a Series to a DataFrame.

I hope this clarifies the concept and gives you confidence in handling DataFrames in your assignment!



Select human–bot teams that have a `bot_members_count` value greater than and equal to 2.

Reasoned for 9 seconds >

You can use boolean indexing to filter your DataFrame for rows that meet both conditions. Here's an example code snippet:

python



```
human_bot_teams = github_teams[(github_teams['Team_type'] == 'human-bot') & (github_teams['bot_members_count'] >= 2)]
```

Explanation

- **Boolean Indexing:**

The expression inside the square brackets creates a boolean mask by evaluating both conditions:

- `(github_teams['Team_type'] == 'human-bot')` checks for rows where the `Team_type` column equals `"human-bot"`.
- `(github_teams['bot_members_count'] >= 2)` checks for rows where the `bot_members_count` column has a value of 2 or more.

- **Combining Conditions:**

The `&` operator is used to combine these two conditions. It's important to wrap each condition in parentheses so that Python correctly evaluates them.

- **Result:**

The filtered DataFrame, stored in `human_bot_teams`, will only contain rows where both conditions are met.

This code will select only those rows that represent human-bot teams with 2 or more bots.



```
what's wrong with this one? "teams =
github_teams[(github_teams['Team_type'] == 'human')
& (github_teams['human'] == 'Large')] &
(github_teams['human_gini'] >= '0.75'"

```

Reasoned for 19 seconds >

There are a couple of issues with your expression:

1. **Parentheses Grouping:**

You closed the indexing brackets before adding the condition on `human_gini`. All the conditions should be combined within a single pair of square brackets. Right now, your code splits the filtering into two parts.

2. **Data Type Comparison:**

If the `human_gini` column is numeric, compare it to a numeric value (i.e., `0.75` without quotes) rather than a string (`'0.75'`).

Corrected Code

Combine all three conditions inside the same indexing brackets and use a numeric literal for `human_gini`:

python



```
teams = github_teams[ (github_teams['Team_type'] == 'human') &
(github_teams['human'] == 'Large') & (github_teams['human_gini'] >=
0.75) ]

```


Explanation

- **Combining Conditions:**

All three conditions are now grouped together with the & operator inside one set of square brackets. This tells pandas to filter rows that satisfy **all** conditions.

- **Numeric Comparison:**

Comparing `human_gini` against `0.75` (a float) ensures that the numeric comparison is valid.

This revised code should work without syntax errors and give you the desired filtered DataFrame.



that gave me this: "-----

KeyError Traceback (most recent call
last)

File /opt/anaconda3/lib/python3.12/site-
packages/pandas/core/indexes/base.py:3805, in
Index.get_loc(self, key)

3804 try:
-> 3805 return self._engine.get_loc(casted_key)
3806 except KeyError as err:

File index.pyx:167, in
pandas._libs.index.IndexEngine.get_loc()

File index.pyx:196, in
pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:7081, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:7089, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'human'

The above exception was the direct cause of the
following exception:

KeyError Traceback (most recent call
last)

Cell In[51], line 5

1 # It returned an error, so I went to the oracle and
came back with:

3 teams = github_teams[
4 (github_teams['Team_type'] == 'human') &
----> 5 (github_teams['human'] == 'Large') &

```
6 (github_teams['human_gini'] >= 0.75)
7 ]
```

```
File /opt/anaconda3/lib/python3.12/site-
packages/pandas/core/frame.py:4102, in
DataFrame.__getitem__(self, key)
4100 if self.columns.nlevels > 1:
4101     return self._getitem_multilevel(key)
-> 4102 indexer = self.columns.get_loc(key)
4103 if is_integer(indexer):
4104     indexer = [indexer]
```

```
File /opt/anaconda3/lib/python3.12/site-
packages/pandas/core/indexes/base.py:3812, in
Index.get_loc(self, key)
3807 if isinstance(casted_key, slice) or (
3808     isinstance(casted_key, abc.Iterable)
3809     and any(isinstance(x, slice) for x in
casted_key)
3810 ):
3811     raise InvalidIndexError(key)
-> 3812 raise KeyError(key) from err
3813 except TypeError:
3814     # If we have a listlike key,
_check_indexing_error will raise
3815     # InvalidIndexError. Otherwise we fall through
and re-raise
3816     # the TypeError.
3817     self._check_indexing_error(key)
```

KeyError: 'human'"

Reasoned for 13 seconds >

The error is caused by referencing a non-existent column name: you're using 'human' instead of the correct column name. If you intend to filter for rows where the team size is labeled as "Large", the correct column appears to be 'Team_size_class' (based on your earlier output).

Try updating your code to:

python



```
teams = github_teams[ (github_teams['Team_type'] == 'human') &
(github_teams['Team_size_class'] == 'Large') &
(github_teams['human_gini'] >= 0.75) ]
```

This should resolve the KeyError by using the proper column name.



the question is "* Find the human teams that are Large and have a human_gini value greater than and equal to 0.75."

Reasoned about filtering human teams for 12 seconds >

You can filter the DataFrame to include only rows that match all three conditions using boolean indexing. Here's how you can do it:

python



```
teams = github_teams[ (github_teams['Team_type'] == 'human') &
(github_teams['Team_size_class'] == 'Large') &
(github_teams['human_gini'] >= 0.75) ]
```

Explanation

- **Boolean Conditions:**

Each condition inside the square brackets checks for a specific requirement:

- `(github_teams['Team_type'] == 'human')` selects rows where `Team_type` is `'human'`.
- `(github_teams['Team_size_class'] == 'Large')` selects rows where `Team_size_class` is `'Large'`.
- `(github_teams['human_gini'] >= 0.75)` selects rows where `human_gini` is greater than or equal to 0.75.

- **Combining Conditions:**

The `&` operator is used to ensure that all these conditions must be true for any given row. Parentheses around each condition are required due to operator precedence.

- **Result:**

The variable `teams` will now be a new DataFrame that contains only those rows that meet all three criteria.

This code meets the requirement to "find the human teams that are Large and have a human_gini value greater than and equal to 0.75."



ok, but it's not actually returning anything. Presumably it's identifying the results, but it's not telling me anything

Reasoned for 23 seconds >

When you assign the filtered DataFrame to `teams`, Python stores the result but doesn't automatically print it to your screen unless you explicitly request it. Depending on the environment in which you're working, you may need to take additional steps to see any output. Here are a few ways to display the filtered results:

1. Using a Print Statement:

python



```
teams = github_teams[ (github_teams['Team_type'] == 'human') &
(github_teams['Team_size_class'] == 'Large') &
(github_teams['human_gini'] >= 0.75) ] print(teams)
```

2. Directly Displaying in Interactive Environments:

If you're using Jupyter Notebook or another interactive Python environment, you can simply put `teams` as the last line in the cell:

python



```
teams = github_teams[ (github_teams['Team_type'] == 'human') &
(github_teams['Team_size_class'] == 'Large') &
(github_teams['human_gini'] >= 0.75) ] teams # This will
automatically display the DataFrame in a notebook cell.
```

3. Using the `display()` Function from IPython:

python



```
from IPython.display import display teams = github_teams[
(github_teams['Team_type'] == 'human') &
(github_teams['Team_size_class'] == 'Large') &
(github_teams['human_gini'] >= 0.75) ] display(teams)
```

If you're still not seeing any output, double-check the following:

- **Empty Result:**

It's possible that none of the rows in your DataFrame satisfy all three conditions.

You can verify this by checking the shape of your result:

python



```
print("Number of rows in teams:", teams.shape[0])
```

If the number of rows is 0, then no data met your criteria.

- **Data Type and Values:**

Verify that the columns contain the expected values. For example, ensure that `Team_type` and `Team_size_class` have the values you're checking against (i.e., `'human'` and `'Large'`) and that `human_gini` contains numeric data.

By explicitly printing or displaying the result, you should be able to see the filtered DataFrame.



```
i'm stuck on this one: "* How many teams are in the
Small or Large category?" "unique_team_size_count1
= 'Small'
unique_team_size_count = 'Large'

print("Number of Small Teams:",
unique_team_size_count1)
print("Number of Large Teams:",
unique_team_size_count2)"
```

Reasoned about team size counting for 11 seconds >

In your current code you're simply assigning the strings 'Small' and 'Large' to variables rather than counting rows in your DataFrame that belong to those categories. To count the teams in each category, you need to use the values in the `Team_size_class` column of your DataFrame.

Here are two common methods to achieve this:

Method 1: Using Boolean Indexing and `len()`

Filter the DataFrame to include only rows where `Team_size_class` is either 'Small' or 'Large' and then count them:

python



```
# Count teams where Team_size_class is 'Small' small_count =
len(github_teams[github_teams['Team_size_class'] == 'Small']) # Count
teams where Team_size_class is 'Large' large_count =
len(github_teams[github_teams['Team_size_class'] == 'Large'])
print("Number of Small Teams:", small_count) print("Number of Large
Teams:", large_count)
```

Method 2: Using `value_counts()`

The `value_counts()` method counts the occurrences of each unique value in a Series. You can then extract the counts for 'Small' and 'Large':

python



```
# Get counts of each team size category size_counts =
github_teams['Team_size_class'].value_counts() # Use the .get() method
to safely retrieve counts (default to 0 if the category doesn't exist)
```

```
small_count = size_counts.get('Small', 0) large_count =  
size_counts.get('Large', 0) print("Number of Small Teams:",  
small_count) print("Number of Large Teams:", large_count)
```

Explanation

- **Boolean Indexing and len() :**

In the first method, the

expression `github_teams[github_teams['Team_size_class'] == 'Small']` filters the DataFrame to only include rows where the team size is 'Small', and then `len()` gives the number of these rows. The same process is repeated for 'Large'.

- **value_counts() :**

In the second method, `value_counts()` creates a Series where the index consists of the unique values from `Team_size_class` and the values are their frequencies.

This method is concise and convenient for getting counts for multiple categories.

Either method will give you the number of teams in the 'Small' or 'Large' category correctly.



```
For the question "How many teams are in the Small or  
Large category with a human_gini value less than and  
equal to 0.20?", I wrote "teams = github_teams[  
    (github_teams['Team_size'] == 'Small', 'Large') &  
    (github_teams['human_gini'] <= 0.20)  
]  
display(teams) " returned "-----  
-----"
```

```
KeyError                                Traceback (most recent call  
last)
```

```
File /opt/anaconda3/lib/python3.12/site-  
packages/pandas/core/indexes/base.py:3805, in  
Index.get_loc(self, key)
```

```
    3804 try:
```

```
-> 3805     return self._engine.get_loc(casted_key)
```

```
    3806 except KeyError as err:
```

```
File index.pyx:167, in
```

```
pandas._libs.index.IndexEngine.get_loc()
```

```
File index.pyx:196, in
```

```
pandas._libs.index.IndexEngine.get_loc()
```

```
File pandas/_libs/hashtable_class_helper.pxi:7081, in
```

```
pandas._libs.hashtable.PyObjectHashTable.get_item()
```

File pandas/_libs/hashtable_class_helper.pxi:7089, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

```
KeyError: 'Team_size'
```

The above exception was the direct cause of the
following exception:

```
KeyError                                Traceback (most recent call  
last)
```

```
Cell In[61], line 2
```

```
1 teams = github_teams[  
----> 2     (github_teams['Team_size'] == 'Small',  
'Large') &  
3     (github_teams['human_gini'] <= 0.20)  
4 ]  
5 display(teams)
```

```
File /opt/anaconda3/lib/python3.12/site-  
packages/pandas/core/frame.py:4102, in  
DataFrame.__getitem__(self, key)  
4100 if self.columns.nlevels > 1:  
4101     return self._getitem_multilevel(key)  
-> 4102 indexer = self.columns.get_loc(key)  
4103 if is_integer(indexer):  
4104     indexer = [indexer]
```

```
File /opt/anaconda3/lib/python3.12/site-  
packages/pandas/core/indexes/base.py:3812, in  
Index.get_loc(self, key)  
3807 if isinstance(casted_key, slice) or (  
3808     isinstance(casted_key, abc.Iterable)  
3809     and any(isinstance(x, slice) for x in  
casted_key)  
3810 ):  
3811     raise InvalidIndexError(key)  
-> 3812 raise KeyError(key) from err  
3813 except TypeError:  
3814     # If we have a listlike key,  
_check_indexing_error will raise  
3815     # InvalidIndexError. Otherwise we fall through  
and re-raise  
3816     # the TypeError.  
3817     self._check_indexing_error(key)
```

```
KeyError: 'Team_size'"
```

Reasoned for 12 seconds >

There are two main issues with your code:

1. Incorrect Column Name:

The error indicates that there is no column called `'Team_size'`. Based on previous examples, the correct column name is likely `'Team_size_class'`.

2. Incorrect Syntax for Checking Multiple Values:

You wrote `(github_teams['Team_size'] == 'Small', 'Large')`, which creates a tuple rather than checking whether the value is either `'Small'` or `'Large'`. To test for membership in multiple values, use the `.isin()` method.

Correct Code

Replace the tuple condition with `.isin(['Small', 'Large'])` and use the proper column name:

python



```
teams = github_teams[ github_teams['Team_size_class'].isin(['Small',  
'Large']) & (github_teams['human_gini'] <= 0.20) ] display(teams)
```

Explanation

- `.isin(['Small', 'Large'])`:
This checks whether each value in the `'Team_size_class'` column is either `'Small'` or `'Large'`.
- **Boolean Mask:**
The code combines the condition for team size with the condition for `human_gini` using the `&` operator, ensuring both are satisfied for each row in the DataFrame.
- **Display Output:**
Using `display(teams)` (or simply `teams` in a Jupyter Notebook cell) will show the filtered rows.

This revised code should filter the DataFrame correctly and display only the teams in the `Small` or `Large` categories with a `human_gini` value less than or equal to 0.20.



but it's supposed to return a count of teams that meet the condition, not display them

Title: Reasoned about counting filtered teams for 5 seconds

To get a count rather than displaying the filtered teams, you'll need to count the number of rows that meet your condition. You can do this by chaining the `.shape` attribute

(which returns a tuple with the number of rows and columns) and selecting the first element of that tuple. For example:

python



```
count_teams = github_teams[
github_teams['Team_size_class'].isin(['Small', 'Large']) &
(github_teams['human_gini'] <= 0.20) ].shape[0] print("Number of teams
meeting the condition:", count_teams)
```

Explanation

- **Filtering with `.isin(['Small', 'Large'])`:**
This checks whether the values in the `'Team_size_class'` column are either `'Small'` or `'Large'`.
- **Applying the Condition on `human_gini`:**
`(github_teams['human_gini'] <= 0.20)` filters for rows where the `human_gini` value is less than or equal to 0.20.
- **Counting Rows with `.shape[0]`:**
After filtering, `.shape[0]` returns the number of rows in the resulting DataFrame, which represents the count of teams meeting the condition.

This code will display the total number of teams that are either in the `Small` or `Large` category and have a `human_gini` value of 0.20 or less.



* Create a subsample of 50% of your data.

Reasoned about creating a data subsample for 5 seconds >

You can create a subsample of 50% of your data using the `sample()` method with the `frac` parameter set to 0.5. This method randomly selects a fraction of the rows from your DataFrame. For example:

python



```
subsample = github_teams.sample(frac=0.5, random_state=42)
print(subsample)
```

Explanation

- **`frac=0.5`:**
This parameter tells pandas to take 50% of the DataFrame's rows.
- **`random_state=42`:**
This parameter ensures reproducibility by seeding the random number generator. You can choose any integer; if you omit this parameter, you'll get a different random sample each time the code is run.

- **Result:**

The variable `subsample` now contains 50% of your original data, selected at random.

This is a straightforward way to create a random subsample of your dataset for further analysis or testing.



* Create samples for a 8-fold cross validation test.

***Reasoned about 8-fold cross-validation sample generation** for 7 seconds* >

An efficient way to create samples for an 8-fold cross validation test is by using scikit-learn's `KFold` class. This method splits the data into 8 (roughly) equal parts and then allows you to iterate through each fold as a test set while the remaining folds serve as the training set.

Here's an example code snippet:

python



```
from sklearn.model_selection import KFold # Initialize the KFold object
with 8 splits kf = KFold(n_splits=8, shuffle=True, random_state=42) #
Create a list to store train-test splits (optional) splits = [] #
Iterate over the KFold splits for train_index, test_index in
kf.split(github_teams): # Create training and test samples based on
current split train_fold = github_teams.iloc[train_index] test_fold =
github_teams.iloc[test_index] splits.append((train_fold, test_fold))
print("Train set size:", train_fold.shape[0], "Test set size:",
test_fold.shape[0])
```

Explanation

- `n_splits=8` : This tells `KFold` to divide your data into 8 folds.
- `shuffle=True` : Randomizes the order of your data before splitting, which is good practice for ensuring that each fold is representative of the overall dataset.
- `random_state=42` : Seeds the random number generator to ensure reproducibility.
- `.split(github_teams)` : Generates indices for the 8 folds. In each iteration, it provides indices for the training set and the test set.
- `.iloc[train_index]` **and** `.iloc[test_index]` : These use the indices to retrieve the corresponding rows from your DataFrame.

Using this approach, you now have 8 distinct sets of train/test splits, which you can use to train and evaluate your model in an 8-fold cross validation procedure.



* Select columns that are numeric and save it as a new DataFrame.

Reasoned for 5 seconds >

You can use pandas' `select_dtypes()` method to filter and create a new DataFrame that contains only the numeric columns. For example:

python



```
# Option 1: Using string type 'number'
numeric_df = github_teams.select_dtypes(include=['number'])
print(numeric_df.head())
# Option 2: Using numpy's number type (requires numpy import)
import numpy as np
numeric_df = github_teams.select_dtypes(include=[np.number])
print(numeric_df.head())
```

Explanation

- `select_dtypes(include=[...])` :
This method filters the DataFrame to include only the columns with the specified data types.
- `'number'` or `np.number` :
These arguments indicate that you only want columns with numeric types (e.g., int, float).

The above code creates a new DataFrame `numeric_df` that contains only the numeric columns from `github_teams`. You then print the first few rows with `head()` to confirm the result.



* Save the columns `Team_size_class` and `human_members_count` as a new DataFrame.

Reasoned for 5 seconds >

The user wants to save just two columns, `Team_size_class` and `human_members_count`, as a new DataFrame. Given the code example, it seems like these columns need to be selected from the `github_teams` DataFrame. I think the code should look something like this:

python



```
subset_df = github_teams[['Team_size_class', 'human_members_count']]
```

That should do it. I'll mention displaying the DataFrame too — we can print it by adding `print(subset_df)` if needed

To create a new DataFrame that contains only the `Team_size_class` and `human_members_count` columns, you can use double bracket indexing. This returns a DataFrame rather than a Series. For example:

python



```
new_df = github_teams[['Team_size_class', 'human_members_count']]
print(new_df.head())
```

Explanation

- **Double Bracket Indexing:**

Using two sets of brackets (i.e. `[['col1', 'col2']]`) tells pandas to return a DataFrame containing only the specified columns.

- **Result:**

The variable `new_df` will now contain just the `Team_size_class` and `human_members_count` columns from your original DataFrame `github_teams`.

This method ensures that your new DataFrame contains only the desired columns.



```
this didn't work: "new_df.rename(columns =
{'Team_Size':'Team_size_class'},
{'Humans_Count':'human_members_count'}, inplace =
True) new_df.columns"
```

Reasoned about DataFrame column renaming for 6 seconds >

The issue is with how the `rename` method is used. The `rename()` function expects a single dictionary for the `columns` parameter that maps the old column names to the new ones, not two separate dictionaries. Instead of:

python



```
new_df.rename(columns = {'Team_Size':'Team_size_class'},
{'Humans_Count':'human_members_count'}, inplace = True) new_df.columns
```

you should combine the mappings in one dictionary:

python



```
new_df.rename(columns={'Team_Size': 'Team_size_class', 'Humans_Count':
'human_members_count'}, inplace=True) print(new_df.columns)
```

Explanation

- **Single Dictionary for columns :**
Pass a single dictionary where each key is the original column name and each value is the new name.
- **inplace=True :**
This tells pandas to update the DataFrame in place rather than returning a new DataFrame.

Once you run the corrected code, calling `new_df.columns` will display the new column names.



```
"/var/folders/xt/1ps45zhn76962p4cclc2wk0h0000gn/T/i
pykernel_33882/3304427153.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a
DataFrame
```

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
`new_df.rename(columns={'Team_Size':
'Team_size_class', 'Humans_Count':
'human_members_count'}, inplace=True)"`

Reasoned for 6 seconds >

The warning is letting you know that you're modifying a slice of the original DataFrame rather than a standalone copy. When you create a new DataFrame by subsetting (for example, using double bracket indexing), sometimes pandas returns a view on the original data, and then modifying it (with operations like `rename()`) can lead to ambiguous behavior. This is known as the SettingWithCopyWarning.

How to Fix It

There are two common ways to handle this:

1. Create an Explicit Copy

Make a new DataFrame that is an explicit copy of your subset. This ensures that any changes you make are on a new, independent DataFrame.

python



```
new_df = github_teams[['Team_size_class',  
'human_members_count']].copy() new_df.rename(columns={'Team_Size':  
'Team_size_class', 'Humans_Count': 'human_members_count'},  
inplace=True) print(new_df.columns)
```

2. Reassign the Renamed DataFrame

Instead of using the `inplace=True` parameter, you can reassign the renamed DataFrame back to `new_df`. This method avoids modifying a view in place.

python



```
new_df = github_teams[['Team_size_class', 'human_members_count']]
new_df = new_df.rename(columns={'Team_Size': 'Team_size_class',
                                'Humans_Count': 'human_members_count'})
print(new_df.columns)
```

Additional Tips

- **Double-check Column Names:**

Ensure that your original DataFrame (`github_teams`) actually has columns named `'Team_Size'` and `'Humans_Count'` if you're trying to rename them. Sometimes the columns might already be correctly named or might use a different capitalization.

- **Understanding the Warning:**

The `SettingWithCopyWarning` is not necessarily an error—it's a caution from pandas that the behavior might be unpredictable if you're working on a view. Creating an explicit copy is a recommended practice for clarity and safety when modifying subsets.

By following one of these approaches, you should no longer see the `SettingWithCopyWarning`, and your modifications will be applied safely to a new DataFrame.