HW 2: Pandas Overview

Due Date: Fri 4/1, 11:59 PM

Collaboration Policy: You may talk with others about the homework, but we ask that you **write your solutions individually**. If you do discuss the assignments with others, please **include their names** in the following line.

Collaborators: *list collaborators here (if applicable)*

This Assignment ¶

Pandas (https://pandas.pydata.org/) is one of the most widely used Python libraries in data science. In this homework, you will learn commonly used data wrangling operations/tools in Pandas. We aim to give you familiarity with:

- Creating dataframes
- Slicing data frames (ie. selecting rows and columns)
- Filtering data (using boolean arrays)
- Data Aggregation/Grouping dataframes
- Merging dataframes

In this homework, you are going to use several pandas methods like `drop()`, `loc[]`, `groupby()`.

You may press `shift+tab` on the method parameters to see the documentation for that method.

Score Breakdown

Question	Points
1a	2
1b	2
2	2
3	2
4	2
5	3
6a	2
6b	2
7a	2
7b	2
7c	3
7d	3
8	3
Total	30

Just as a side note: Pandas operations can be confusing at times and the documentation is not great, but it is OK to be stumped when figuring out why a piece of code is not doing what it's supposed to. We don't expect you to memorize all the different Pandas functions, just know the basic ones like `iloc[]`, `loc[]`, slicing, and other general dataframe operations.

Throughout the semester, you will have to search through Pandas documentation and experiment, but remember it is part of the learning experience and will help shape you as a data scientist!

Setup

In [107]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline


class bcolor:
   BLACK = '\033[40m'
   YELLOW = '\033[93m'
   RED = '\033[91m'
   BOLD = '\033[1m'
   END = '\033[0m'

def print_passed(str_in):
  print(bcolor.BLACK + bcolor.YELLOW + bcolor.BOLD + str_in + bcolor.END)
```

Creating DataFrames & Basic Manipulations

A dataframe (http://pandas.pydata.org/pandas-docs/stable/dsintro.html#dataframe) is a two-dimensional labeled data structure with columns of potentially different types.

The pandas `DataFrame` function (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html) provides at least two syntaxes to create a data frame.

Syntax 1: You can create a data frame by specifying the columns and values using a dictionary as shown below.

The keys of the dictionary are the column names, and the values of the dictionary are lists containing the row entries.

In [108]:

```
fruit_info = pd.DataFrame(
    data={'fruit': ['apple', 'orange', 'banana', 'raspberry'],
          'color': ['red', 'orange', 'yellow', 'pink']
         })
fruit_info
```

Out[108]:

	fruit	color
0	apple	red
1	orange	orange
2	banana	yellow
3	raspberry	pink

Syntax 2: You can also define a dataframe by specifying the rows like below.

Each row corresponds to a distinct tuple, and the columns are specified separately.

In [109]:

```
fruit_info2 = pd.DataFrame(
    [("red", "apple"), ("orange", "orange"), ("yellow", "banana"),
     ("pink", "raspberry")],
    columns = ["color", "fruit"])
fruit_info2
```

Out[109]:

	color	fruit
0	red	apple
1	orange	orange
2	yellow	banana
3	pink	raspberry

You can obtain the dimensions of a matrix by using the shape attribute dataframe.shape

In [110]:

```
(num_rows, num_columns) = fruit_info.shape
num_rows, num_columns
```

Out[110]:

(4, 2)

Question 1(a)

You can add a column by `dataframe['new column name'] = [data]`. Please add a column called `rank1` to the `fruit_info` table which contains a 1,2,3, or 4 based on your personal preference ordering for each fruit.

In [111]:

```
# BEGIN YOUR CODE
# ------------------------
fruit_info['rank1'] = [1,3,2,4]
# ------------------------
# END YOUR CODE
```

In [112]:

```
fruit_info
```

Out[112]:

	fruit	color	rank1
0	apple	red	1
1	orange	orange	3
2	banana	yellow	2
3	raspberry	pink	4

In [113]:

```
assert fruit_info["rank1"].dtype == np.dtype('int64')
assert len(fruit_info["rank1"].dropna()) == 4
print_passed('Q1a: Passed all unit tests!')
```

```
Q1a: Passed all unit tests!
```

Question 1(b)

You can ALSO add a column by `dataframe.loc[:, 'new column name'] = [data]`. This way to modify an existing dataframe is preferred over the assignment above. In other words, it is best that you use `loc[]`. Although using `loc[]` is more verbose, it is faster. (However, this tradeoff is more likely to be valuable in production than during interactive use.) We will explain in more detail what `loc[]` does, but essentially, the first parameter is for the rows and second is for columns. The `:` means keep all rows and the `new column name` indicates the column you are modifying or in this case adding.

Please add a column called `rank2` to the `fruit_info` table which contains a 1,2,3, or 4 based on your personal preference ordering for each fruit.

In [114]:

```
# BEGIN YOUR CODE
# ------------------------
fruit_info.loc[:, 'rank2'] = [1,3,2,4]
# ------------------------
# END YOUR CODE
```

In [115]:

```
fruit_info
```

Out[115]:

	fruit	color	rank1	rank2
0	apple	red	1	1
1	orange	orange	3	3
2	banana	yellow	2	2
3	raspberry	pink	4	4

In [116]:

```
assert fruit_info["rank2"].dtype == np.dtype('int64')
assert len(fruit_info["rank2"].dropna()) == 4

print_passed('Q1b: Passed all unit tests!')
```

```
Q1b: Passed all unit tests!
```

Question 2

Use the `.drop()` method to drop (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.drop.html) the both the `rank1` and `rank2` columns you created. (Make sure to use the `axis` parameter correctly)

Hint: Look through the documentation to see how you can drop multiple columns of a Pandas dataframe at once, it may involve a list.

In [117]:

```
# BEGIN YOUR CODE
# ------------------------
fruit_info_original = fruit_info.drop(['rank1', 'rank2'], axis = 1)
# ------------------------
# END YOUR CODE
```

In [118]:

```
fruit_info_original
```

Out[118]:

	fruit	color
0	apple	red
1	orange	orange
2	banana	yellow
3	raspberry	pink

In [119]:

```
assert fruit_info_original.shape[1] == 2

print_passed('Q2: Passed all unit tests!')
```

```
Q2: Passed all unit tests!
```

Question 3

Use the `.rename()` method to rename (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.rename.html) the columns of `fruit_info_original` so they begin with a capital letter. Set the `inplace` parameter correctly to change the `fruit_info_original` dataframe. (**Hint:** in Question 2, `drop` creates and returns a new dataframe instead of changing `fruit_info` because `inplace` by default is `False`)

In [120]:

```
# BEGIN YOUR CODE
# ------------------------
fruit_info_original.rename(columns={"fruit": "Fruit" , "color" : "Color"}, inplace = True) ## t
he original dataframe gets modified when inplace= True.. but when inplace = False, then it just
 creates a copy of the original dataframe with the modification but no modification in the origi
nal dataframe

# ------------------------
# END YOUR CODE
```

In [121]:

```
fruit_info_original
```

Out[121]:

	Fruit	Color
0	apple	red
1	orange	orange
2	banana	yellow
3	raspberry	pink

In [122]:

```
assert fruit_info_original.columns[0] == 'Fruit'
assert fruit_info_original.columns[1] == 'Color'

print_passed('Q3: Passed all unit tests!')
```

```
Q3: Passed all unit tests!
```

Mount your Google Drive

When you run a code cell, Colab executes it on a temporary cloud instance. Every time you open the notebook, you will be assigned a different machine. All compute state and files saved on the previous machine will be lost. Therefore, you may need to re-download datasets or rerun code after a reset. Here, you can mount your Google drive to the temporary cloud instance's local filesystem using the following code snippet and save files under the specified directory (note that you will have to provide permission every time you run this).

In [123]:

```
# mount Google drive
from google.colab import drive
drive.mount('/content/drive')

# now you can see files
!echo -e "\nNumber of Google drive files in /content/drive/My Drive/:"
!ls -l "/content/drive/My Drive/" | wc -l
# by the way, you can run any linux command by putting a ! at the start of the line

# by default everything gets executed and saved in /content/
!echo -e "\nCurrent directory:"
!pwd

workspace_path = '/content/drive/MyDrive/DataScience/'  # Change this path!
print(f'Your workspace: {workspace_path}')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call driv
e.mount("/content/drive", force_remount=True).

Number of Google drive files in /content/drive/My Drive/:
161

Current directory:
/content
Your workspace: /content/drive/MyDrive/DataScience/
```

Babyname datasets

Now that we have learned the basics, let's move on to the babynames dataset. Let's clean and wrangle the following data frames for the remainder of the lab.

First let's run the following cells to build the dataframe `baby_names`. The cells below download the data from the web and extract the data in a California region. There should be a total of 5933561 records.

The following cell builds the final full `baby_names` DataFrame. Here is documentation for pd.concat (https://pandas.pydata.org/pandas-docs/version/0.22/generated/pandas.concat.html) if you want to know more about its functionality.

In [124]:

```
import zipfile
zf = zipfile.ZipFile(workspace_path+'namesbystate.zip', 'r')

field_names = ['State', 'Sex', 'Year', 'Name', 'Count']

def load_dataframe_from_zip(zf, f):
    with zf.open(f) as fh:
        return pd.read_csv(fh, header=None, names=field_names)

# List comprehension
states = [
    load_dataframe_from_zip(zf, f)
    for f in sorted(zf.filelist, key=lambda x:x.filename)
    if f.filename.endswith('.TXT')
]

baby_names = pd.concat(states).reset_index(drop=True)
```

In [125]:

```
baby_names.head()
```

Out[125]:

	State	Sex	Year	Name	Count
0	AK	F	1910	Mary	14
1	AK	F	1910	Annie	12
2	AK	F	1910	Anna	10
3	AK	F	1910	Margaret	8
4	AK	F	1910	Helen	7

In [126]:

```
len(baby_names)
```

Out[126]:

5933561

Slicing Data Frames - selecting rows and columns

Selection Using Label

Column Selection To select a column of a `DataFrame` by column label, the safest and fastest way is to use the `.loc` method (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.loc.html). General usage looks like `frame.loc[rowname,colname]`. (Reminder that the colon `:` means "everything"). For example, if we want the `color` column of the `ex` data frame, we would use : `ex.loc[:, 'color']`

- You can also slice across columns. For example, `baby_names.loc[:, 'Name':]` would give select the columns `Name` and the columns after.
- *Alternative:* While `.loc` is invaluable when writing production code, it may be a little too verbose for interactive use. One recommended alternative is the `[]` method, which takes on the form `frame['colname']`.

Row Selection Similarly, if we want to select a row by its label, we can use the same `.loc` method. In this case, the "label" of each row refers to the index (ie. primary key) of the dataframe.

In [127]:

```
#Example:
baby_names.loc[2:5, 'Name']
```

Out[127]:

```
2       Anna
3    Margaret
4      Helen
5      Elsie
Name: Name, dtype: object
```

In [128]:

```
#Example:  Notice the difference between these two methods
baby_names.loc[2:5, ['Name']]
```

Out[128]:

	Name
2	Anna
3	Margaret
4	Helen
5	Elsie

The `.loc` actually uses the Pandas row index rather than row id/position of rows in the dataframe to perform the selection. Also, notice that if you write `2:5` with `loc[]`, contrary to normal Python slicing functionality, the end index is included, so you get the row with index 5.

There is another Pandas slicing function called `iloc[]` which lets you slice the dataframe by row id and column id instead of by column name and row index (for `loc[]`). This is really the main difference between the 2 functions and it is important that you remember the difference and why you might want to use one over the other.

In addition, with `iloc[]`, the end index is NOT included, like with normal Python slicing.

Here is an example of how we would get the 2nd, 3rd, and 4th rows with only the `Name` column of the `baby_names` dataframe using both `iloc[]` and `loc[]`. Observe the difference.

In [129]:

```
baby_names.iloc[1:4, 3]
```

Out[129]:

```
1       Annie
2        Anna
3    Margaret
Name: Name, dtype: object
```

In [130]:

```
baby_names.loc[1:3, "Name"]
```

Out[130]:

```
1       Annie
2        Anna
3    Margaret
Name: Name, dtype: object
```

Lastly, we can change the index of a dataframe using the `set_index` method.

In [131]:

```
#Example: We change the index from 0,1,2... to the Name column
df = baby_names[:5].set_index("Name")
df
```

Out[131]:

Name	State	Sex	Year	Count
Mary	AK	F	1910	14
Annie	AK	F	1910	12
Anna	AK	F	1910	10
Margaret	AK	F	1910	8
Helen	AK	F	1910	7

We can now lookup rows by name directly:

In [132]:

```
df.loc[['Mary', 'Anna'], :]
```

Out[132]:

Name	State	Sex	Year	Count
Mary	AK	F	1910	14
Anna	AK	F	1910	10

However, if we still want to access rows by location we will need to use the integer loc (`iloc`) accessor:

In [133]:

```
#Example:
#df.loc[2:5,"Year"] You can't do this
df.iloc[1:4,2:3]
```

Out[133]:

Name	Year
Annie	1910
Anna	1910
Margaret	1910

Question 4

Selecting multiple columns is easy. You just need to supply a list of column names. Select the `Name` and `Year` **in that order** from the `baby_names` table.

In [134]:

```
# BEGIN YOUR CODE
# ------------------------
name_and_year = baby_names.loc[:, ['Name', 'Year']]
# ------------------------
# END YOUR CODE
```

In [135]:

```
name_and_year[:5]
```

Out[135]:

	Name	Year
0	Mary	1910
1	Annie	1910
2	Anna	1910
3	Margaret	1910
4	Helen	1910

In [136]:

```
assert name_and_year.shape == (5933561, 2)
assert name_and_year.loc[0,"Name"] == "Mary"
assert name_and_year.loc[0,"Year"] == 1910

print_passed('Q4: Passed all unit tests!')
```

```
Q4: Passed all unit tests!
```

As you may have noticed above, the .loc() method is a way to re-order the columns within a dataframe.

Filtering Data

Filtering with boolean arrays

Filtering is the process of removing unwanted material. In your quest for cleaner data, you will undoubtedly filter your data at some point: whether it be for clearing up cases with missing values, culling out fishy outliers, or analyzing subgroups of your data set. Note that compound expressions have to be grouped with parentheses. Example usage looks like `df[df[column name] < 5]]` .

For your reference, some commonly used comparison operators are given below.

Symbol	Usage	Meaning
==	a == b	Does a equal b?
<=	a <= b	Is a less than or equal to b?
>=	a >= b	Is a greater than or equal to b?
<	a < b	Is a less than b?
>	a > b	Is a greater than b?
~	~p	Returns negation of p
|	p | q	p OR q
&	p & q	p AND q
^	p ^ q	p XOR q (exclusive or)

In the following we construct the DataFrame containing only names registered in California

In [137]:

```
ca = baby_names[baby_names['State'] == "CA"]
```

Question 5

Select the names in Year 2000 (for all baby_names) that have larger than 3000 counts. What do you notice?

(If you use `p & q` to filter the dataframe, make sure to use `df[df[(p) & (q)]]` or `df.loc[df[(p) & (q)]]`)

Remember that both slicing and using `loc` will achieve the same result, it is just that `loc` is typically faster in production. You are free to use whichever one you would like.

In [138]:

```
# BEGIN YOUR CODE
# ------------------------
result = baby_names.loc[(baby_names['Year'] == 2000) &(baby_names['Count'] > 3000) ]
# ------------------------
# END YOUR CODE
```

In [139]:

```
result
```

Out[139]:

	State	Sex	Year	Name	Count
697386	CA	M	2000	Daniel	4339
697387	CA	M	2000	Anthony	3838
697388	CA	M	2000	Jose	3803
697389	CA	M	2000	Andrew	3600
697390	CA	M	2000	Michael	3571
697391	CA	M	2000	Jacob	3520
697392	CA	M	2000	Joshua	3356
697393	CA	M	2000	Christopher	3335
697394	CA	M	2000	David	3280
697395	CA	M	2000	Matthew	3254
5289638	TX	M	2000	Jose	3097

In [140]:

```
assert len(result) == 11
assert result["Count"].sum() == 38993
assert result["Count"].iloc[0] == 4339

print_passed('Q5: Passed all unit tests!')
```

Q5: Passed all unit tests!

Data Aggregration (Grouping Data Frames)

Question 6a

To count the number of instances of each unique value in a `Series`, we can use the `value_counts()` method (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.value_counts.html) as `df["col_name"].value_counts()`. Count the number of different names for each Year in `CA` (California). (You may use the `ca` DataFrame created above.)

Note: *We are not computing the number of babies but instead the number of names (rows in the table) for each year.*

In [141]:

```
# BEGIN YOUR CODE
# ------------------------
num_of_names_per_year = ca['Year'].value_counts()
# ------------------------
# END YOUR CODE
```

In [142]:

```
num_of_names_per_year[:5]
```

Out[142]:

```
2007    7248
2008    7156
2009    7119
2006    7075
2010    7009
Name: Year, dtype: int64
```

In [143]:

```
assert num_of_names_per_year[2007] == 7248
assert num_of_names_per_year[:5].sum() == 35607
assert num_of_names_per_year[1910] == 363
assert num_of_names_per_year[:15].sum() == 103699

print_passed('Q6a: Passed all unit tests!')
```

```
Q6a: Passed all unit tests!
```

Question 6b

Count the number of different names for each gender in `CA`. Does the result help explaining the findings in Question 5?

In [144]:

```
# BEGIN YOUR CODE
# ------------------------
num_of_names_per_gender = ca['Sex'].value_counts()
# ------------------------
# END YOUR CODE
```

In [145]:

```
num_of_names_per_gender
```

Out[145]:

```
F    221084
M    153550
Name: Sex, dtype: int64
```

In [146]:

```
assert num_of_names_per_gender["F"] > 200000
assert num_of_names_per_gender["F"] == 221084
assert num_of_names_per_gender["M"] == 153550

print_passed('Q6b: Passed all unit tests!')
```

```
Q6b: Passed all unit tests!
```

Question 7: Groupby

Before we jump into using the `groupby` function in Pandas, let's recap how grouping works in general for tabular data through a guided set of questions based on a small toy dataset of movies and genres.

Note: If you want to see a visual of how grouping of data works, here is a link to an animation from last week's slides: Groupby Animation (http://www.ds100.org/sp18/assets/lectures/lec03/03-groupby_and_pivot.pdf)

Problem Setting: This summer 2018, there were a lot of good and bad movies that came out. Below is a dataframe with 5 columns: name of the movie as a `string`, the genre of the movie as a `string`, the first name of the director of the movie as a `string`, the average rating out of 10 on Rotten Tomatoes as an `integer`, and the total gross revenue made by the movie as an `integer`. The point of these guided questions (parts a and b) below is to understand how grouping of data works in general, **not** how grouping works in code. We will worry about how grouping works in Pandas in 7c, which will follow.

Below is the `movies` dataframe we are using, imported from the `movies.csv` file located in the `lab02` directory.

In [147]:

```
movies = pd.read_csv(workspace_path+"movies.csv")
movies
```

Out[147]:

	director	genre	movie	rating	revenue
0	David	Action & Adventure	Deadpool 2	7	318344544
1	Bill	Comedy	Book Club	5	68566296
2	Ron	Science Fiction & Fantasy	Solo: A Star Wars Story	6	213476293
3	Baltasar	Drama	Adrift	6	31445012
4	Bart	Drama	American Animals	6	2847319
5	Gary	Action & Adventure	Oceans 8	6	138803463
6	Drew	Action & Adventure	Hotel Artemis	8	6708147
7	Brad	Animation	Incredibles 2	5	594398019
8	Jeff	Comedy	Tag	6	54336863
9	J.A.	Science Fiction & Fantasy	Jurassic World: Fallen Kingdom	6	411873505
10	Charles	Comedy	Uncle Drew	5	42201656
11	Gerard	Horror	The First Purge	7	68765655
12	Peyton	Action & Adventure	Ant-Man and the Wasp	5	208681866
13	Genndy	Animation	Hotel Transylvania 3: Summer Vacation	5	154418311
14	Rawson	Action & Adventure	Skyscraper	6	66801215
15	Ol	Comedy	Mamma Mia! Here We Go Again	8	111705055
16	Christopher	Action & Adventure	Mission: Impossible-Fallout	6	182080372
17	Marc	Comedy	Christopher Robbin	6	6786317

Question 7a

If we grouped the `movies` dataframe above by `genre`, how many groups would be in the output and what would be the groups? Assign `num_groups` to the number of groups created and fill in `genre_list` with the names of genres as strings that represent the groups.

In [148]:

```
# BEGIN YOUR CODE
# ------------------------
num_groups = 6
genre_list = ['Action & Adventure', 'Comedy', 'Science Fiction & Fantasy', 'Drama', 'Animation',
'Horror']
# ------------------------
# END YOUR CODE
```

In [149]:

```
assert num_groups == 6
assert set(genre_list) == set(['Action & Adventure', 'Comedy', 'Science Fiction & Fantasy', 'Dr
ama', 'Animation', 'Horror'])

print_passed('Q7a: Passed all unit tests!')
```

```
Q7a: Passed all unit tests!
```

Question 7b

Whenever we group tabular data, it is usually the case that we need to aggregate values from the ungrouped column(s). If we were to group the `movies` dataframe above by `genre`, which column(s) in the `movies` dataframe would it make sense to aggregate if we were interested in finding how well each genre did in the eyes of people? Fill in `agg_cols` with the column name(s).

In [150]:

```
# BEGIN YOUR CODE
# ------------------------
agg_cols = ['rating', 'revenue']
# ------------------------
# END YOUR CODE
```

In [151]:

```
assert set(agg_cols) == set(['rating', 'revenue'])

print_passed('Q7b: Passed all unit tests!')
```

```
Q7b: Passed all unit tests!
```

Now, let's see `groupby` in action, instead of keeping everything abstract. To aggregate data in Pandas, we use the `.groupby()` function (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.groupby.html). The code below will group the `movies` dataframe by `genre` and find the average revenue and rating for each genre. You can verify you had the same number of groups as what you answered in 7a.

In [152]:

```
movies.loc[:, ['genre', 'rating', 'revenue']].groupby('genre').mean()
```

Out[152]:

genre	rating	revenue
Action & Adventure	6.333333	153569934.5
Animation	5.000000	374408165.0
Comedy	6.000000	56719237.4
Drama	6.000000	17146165.5
Horror	7.000000	68765655.0
Science Fiction & Fantasy	6.000000	312674899.0

Question 7c

Let's move back to baby names and specifically, the `ca` dataframe. Find the sum of `Count` for each `Name` in the `ca` table. You should use `df.groupby("col_name").sum()` . Your result should be a Pandas Series.

Note: *In this question we are now computing the number of registered babies with a given name.*

In [153]:

```
# BEGIN YOUR CODE
# ------------------------
count_for_names = ca.groupby(['Name'])['Count'].sum()
# ------------------------
# END YOUR CODE
```

In [154]:

```
count_for_names.sort_values(ascending=False)[:5]
```

Out[154]:

```
Name
Michael    429827
David      371646
Robert     351051
John       313615
James      279985
Name: Count, dtype: int64
```

In [155]:

```
assert count_for_names["Michael"] == 429827
assert count_for_names[:100].sum() == 95519
assert count_for_names["David"] == 371646
assert count_for_names[:1000].sum() == 1320144

print_passed('Q7c: Passed all unit tests!')
```

```
Q7c: Passed all unit tests!
```

Question 7d

Find the sum of `Count` for each female name after year 1999 (>1999) in California.

In [156]:

```
# BEGIN YOUR CODE
# ------------------------
female_name_count = ca.loc[(ca['Sex'] == 'F') & (ca['Year'] > 1999)].groupby('Name')['Count'].sum()

# ------------------------
# END YOUR CODE
```

In [157]:

```
female_name_count.sort_values(ascending=False)[:5]
```

Out[157]:

```
Name
Emily       48093
Isabella    45232
Sophia      43934
Mia         35639
Emma        34881
Name: Count, dtype: int64
```

In [158]:

```
assert female_name_count["Emily"] == 48093
assert female_name_count[:100].sum() == 48596
assert female_name_count["Isabella"] == 45232
assert female_name_count[:10000].sum() == 3914766

print_passed('Q7d: Passed all unit tests!')
```

```
Q7d: Passed all unit tests!
```

Question 8: Merging

Time to put everything together! Merge `movies` and `count_for_names` to find the number of registered baby names for each director. Only include names that appear in both `movies` and `count_for_names` .

Hint: You might need to convert the `count_for_names` series to a dataframe. Take a look at the `to_frame` method of a series to do this.

Your first row should look something like this:

Note: It is ok if you have 2 separate columns with names instead of just one column.

	director	genre	movie	rating	revenue	Count
0	David	Action & Adventure	Deadpool 2	7	318344544	371646

In [159]:

```
# BEGIN YOUR CODE
# ------------------------
merged_df = movies.merge(count_for_names.to_frame(), how = 'inner', left_on = 'director', right_
on = 'Name' )
# ------------------------
# END YOUR CODE
```

In [160]:

```
assert merged_df.loc[0, 'Count'] == 371646
assert merged_df.loc[3, 'Count'] == 5
assert merged_df.loc[7, 'Count'] == 7236
assert merged_df['Count'].sum() == 861694
assert len(merged_df) == 14

print_passed('Q8: Passed all unit tests!')
```

```
Q8: Passed all unit tests!
```

Congratulations! You have completed HW2.

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output.,

Please save before submitting!