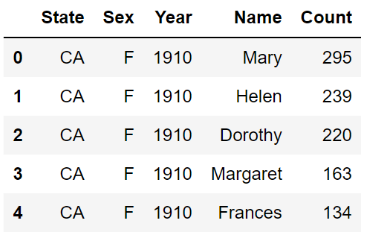
Data 100, Spring 2022 **January 4, 2022**

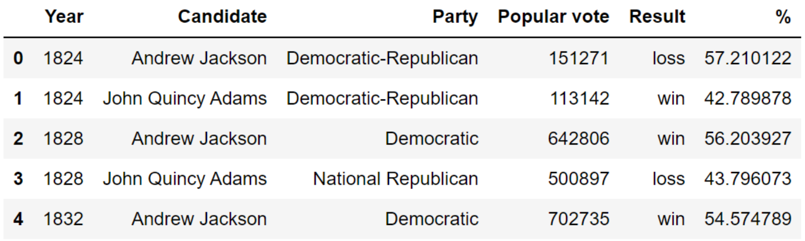
Discussion #3

Pandas Bootcamp

Throughout this section you'll be working with the babynames (left) and elections (right) datasets as shown below (only the first five rows are shown):



	State	Sex	Year	Name	Count
0	CA	F	1910	Mary	295
1	CA	F	1910	Helen	239
2	CA	F	1910	Dorothy	220
3	CA	F	1910	Margaret	163
4	CA	F	1910	Frances	134

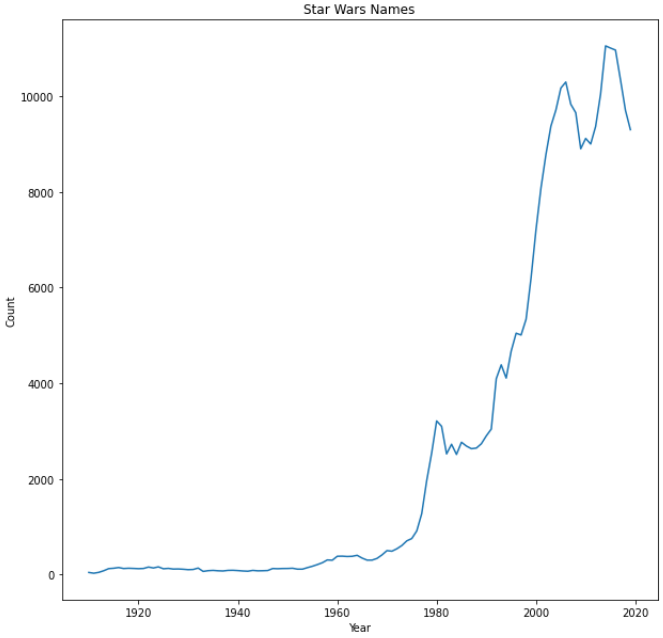


	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789

1. (a) We perform some basic EDA on this data, and we decide to visualize the popularity of the names Luke, Leia, and Han from Star Wars to see if there is a relationship with the release of the major films with the popularity of these names.

   Fill in the blanks to output a Series that contains the year as the index and the number of total Star Wars names as the value, so we can make the plot below!

   Hint: `babynames['Name'].isin(['Helen', 'Jon'])` returns `[False, True, ...]`.



```
sw_names = ['Luke', 'Leia', 'Han']
babynames[_______________________________________] \
         .groupby(_______)_______________________ \
         .plot(ylabel = 'Count', title = 'Star Wars Names',
               figsize = (8, 8))
```

(b) Define the fluctuation of a baby name as the mathematical range of its count per year throughout its history (i.e. maximum count subtracted by minimum count). Write a line of Pandas code to determine **per-state** fluctuations for all baby names, sorted from greatest to least.

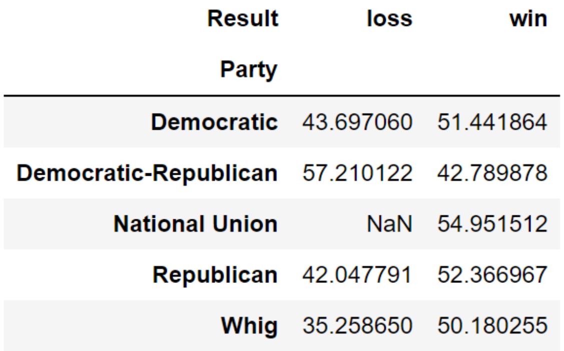
(c) Define an upset as an election result for a party that is an outlier vote share attained in that party's history. Fill in the blanks below to find all the rows in `elections` corresponding to election upsets in American history per this definition.

Hint: the `quantile` function can return the quartiles of the data; for example, `elections['%'].quantile(0.25)` returns the first quartile (Q_1). Recall that a point is an outlier if it is outside the interval $[Q_1 - 1.5\text{IQR}, Q_3 + 1.5\text{IQR}]$.

```
def outlier(subdf):
    q1, q3 = ________________________, ________________________
    iqr = _____________________________________________________
    return subdf[______________________________________________
                 ______________________________________________]
elections.groupby(_________).apply(_________)
```

(d) Write a line of code to output a DataFrame showing the average winning and losing vote share for every party that has won an election (a sample of 5 rows are shown below).

Hint: The arguments to `pivot_table` are `index`, `columns`, `values`, and `aggfunc`.



Result	loss	win
Party		
Democratic	43.697060	51.441864
Democratic-Republican	57.210122	42.789878
National Union	NaN	54.951512
Republican	42.047791	52.366967
Whig	35.258650	50.180255

(e) Fill in the blanks below to create a new column `Middle Name` containing every candidate's middle name (or middle initial). If a candidate has no middle name, that entry should be NaN.

Hint: The default entry of any element in a DataFrame if unspecified is NaN!

```
mid = ______________________________________
elections.loc[_____________________, 'Middle Name'] = ________________
```

(f) Define election twins as two candidates that share the same middle name (or middle initial). Fill in the code below to determine the number of election twins.

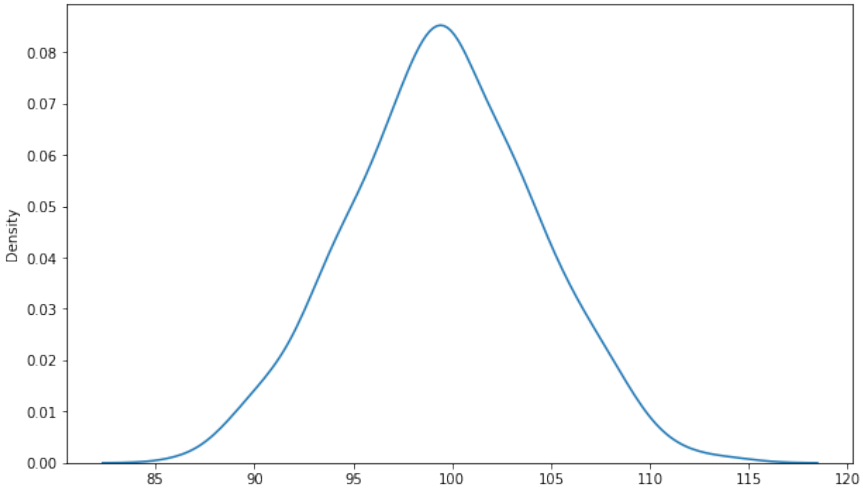
Hint: Try to use a merge, and recall that for merges with non-unique column names, Pandas will rename the non-unique column name with an `_x` suffix for the left table and `_y` suffix for the right table (i.e. for a column `col`, the resulting names would be `col_x` and `col_y`).

```
def election_twins(elections):
    elections = _________________________________________
    twins = _____________________________________________
    twins = twins[_______________________________________]
    return len(twins)
```

Dealing with Missing Data

2. While working with a movie dataset from IMDB, Anirudhan realizes that nearly 20% of the votes field is missing with NaN values (however, none of the other fields have null values)! He wants to use the dataset for modeling, so he must impute or drop the missing values. Help him make the correct decision to solve the missing data problem in these subparts given the distribution of the variable.

   (a) Suppose that the distribution of this variable (i.e. `df['votes'].plot.kde()`) is given by the following figure. Which of the following techniques would be **most** effective in solving this issue of missing data?



   □ A. Using the mean to impute the missing values

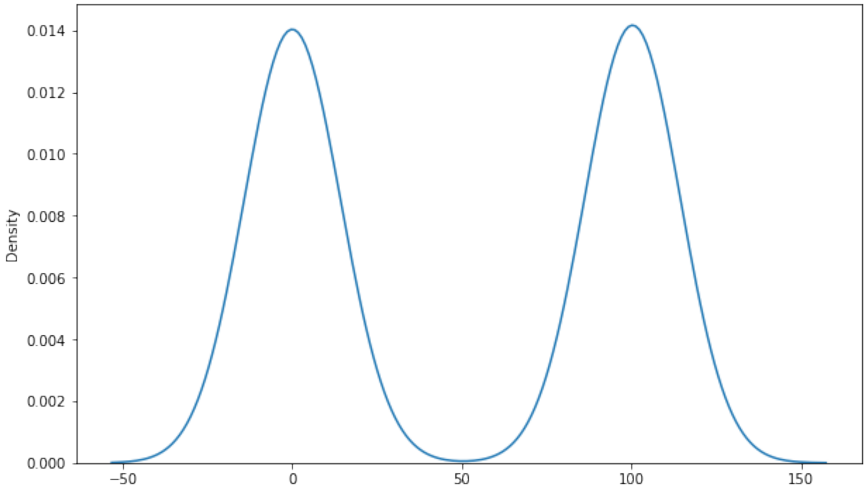
   □ B. Using the mode to impute the missing values

   □ C. Using the median to impute the missing values

   □ D. Dropping any rows with missing values

   □ E. Imputing missing values with zero

   (b) Suppose that the distribution of this variable is given by the following figure. Which of the following techniques would be **most** effective in solving this issue of missing data?



□ A. Using the mean to impute the missing values

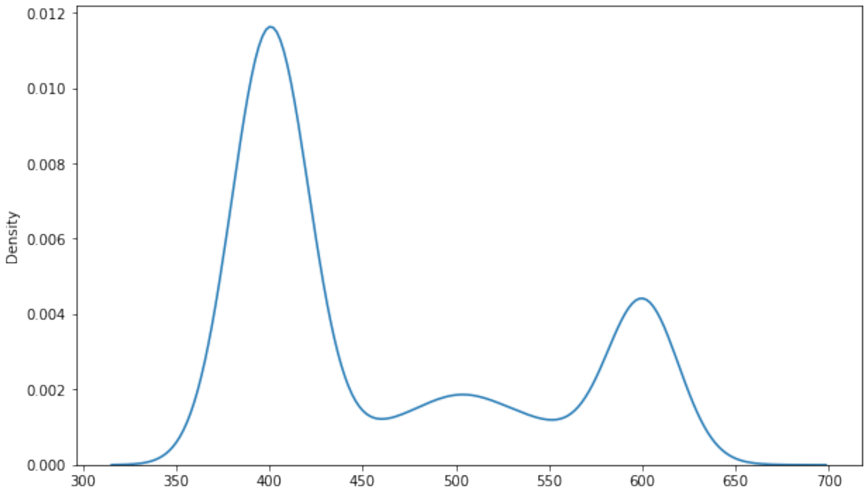
□ B. Using the mode to impute the missing values

□ C. Using the median to impute the missing values

□ D. Dropping any rows with missing values

□ E. Imputing missing values with zero

(c) Suppose that the distribution of this variable is given by the following figure. Which of the following techniques would be **reasonably** effective in solving this issue of missing data?



□ A. Using the mean to impute the missing values

□ B. Using the mode to impute the missing values

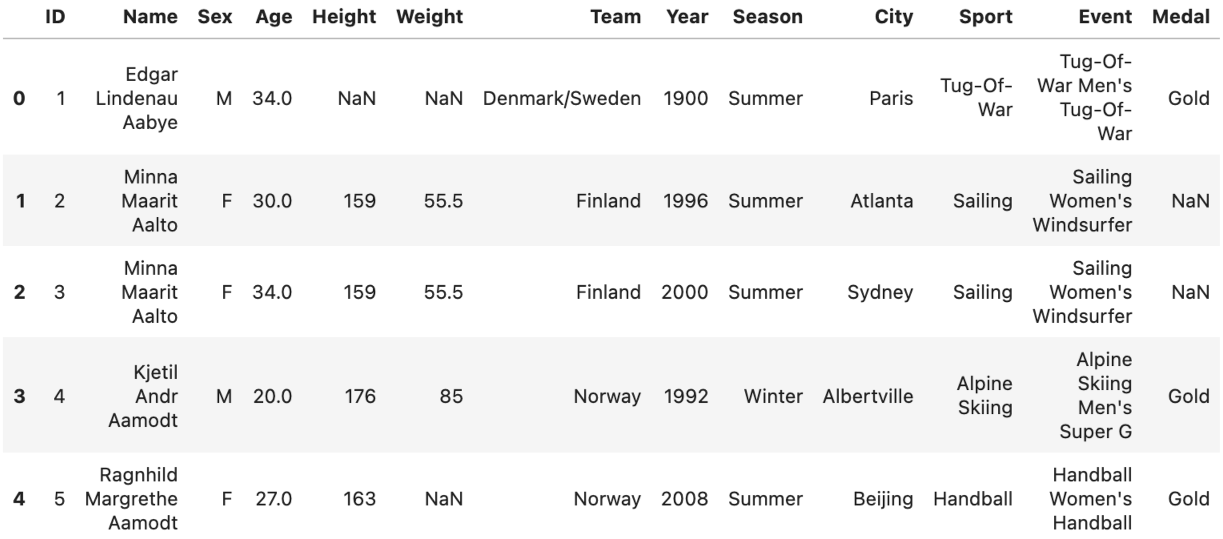
□ C. Using the median to impute the missing values

□ D. Dropping any rows with missing values

□ E. Imputing missing values with zero

Pandas: Olympics (Bonus)

3. We will work with an Olympics dataset containing the names of all athletes who participated in the Olympic Games, including all the Games from Athens 1896 to Tokyo 2020. The first 5 lines of the table are shown below. You may assume that the ID column is the primary key of the table and that the only column with null values are height, weight, and medal.



	ID	Name	Sex	Age	Height	Weight	Team	Year	Season	City	Sport	Event	Medal
0	1	Edgar Lindenau Aabye	M	34.0	NaN	NaN	Denmark/Sweden	1900	Summer	Paris	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	Gold
1	2	Minna Maarit Aalto	F	30.0	159	55.5	Finland	1996	Summer	Atlanta	Sailing	Sailing Women's Windsurfer	NaN
2	3	Minna Maarit Aalto	F	34.0	159	55.5	Finland	2000	Summer	Sydney	Sailing	Sailing Women's Windsurfer	NaN
3	4	Kjetil Andr Aamodt	M	20.0	176	85	Norway	1992	Winter	Albertville	Alpine Skiing	Alpine Skiing Men's Super G	Gold
4	5	Ragnhild Margrethe Aamodt	F	27.0	163	NaN	Norway	2008	Summer	Beijing	Handball	Handball Women's Handball	Gold

(a) Write a line of Pandas code to determine the 10 most common middle names among gold medal winners. You may assume that all athletes in the table have a middle name.

(b) What are the oldest athletes to participate in each sport along with the corresponding year in which they participated? Fill in the blanks below to answer the question.

```
ath.groupby(______________) \
   .apply(_______________________________________________) \
   [['Name', 'Year']]
```

(c) Fill in the blanks below to return the names of all the athletes who won a medal after a gap of 10 years or more of not winning any Olympics medals. You may assume that each individual's name is unique to them.

```
def filter_func(subframe):
    return ____________________________________________________

ath.sort_values(_____________________) \
   [_____________________] \
   .groupby(_________) \
   .filter(filter_func)['Name'] \
   .unique()
```

(d) For all athletes that performed in more than 2 Olympics, which athletes had a **better** Olympics debut year than any of their subsequent performances in terms of total medal count? Sort by least difference in medal count from their debut year from any of their performances from subsequent years, from greatest to least.

Hint: The instructor solution uses `groupby` three times (you may not need to though)!

```
def f1(s):
    return _________________________________________
def f2(s):
    ____________________________________________________
    return _________________________________________
________________________________________________________
________________________________________________________
________________________________________________________
________________________________________________________
________________________________________________________
names = ____________________________________________
```