LECTURE 4

Pandas Bootcamp: Part 2

Advanced Pandas (Aggregation, Merging)

CSCI 3022 @ CU Boulder

Maribeth Oscamou

Content credit: [Acknowledgments]

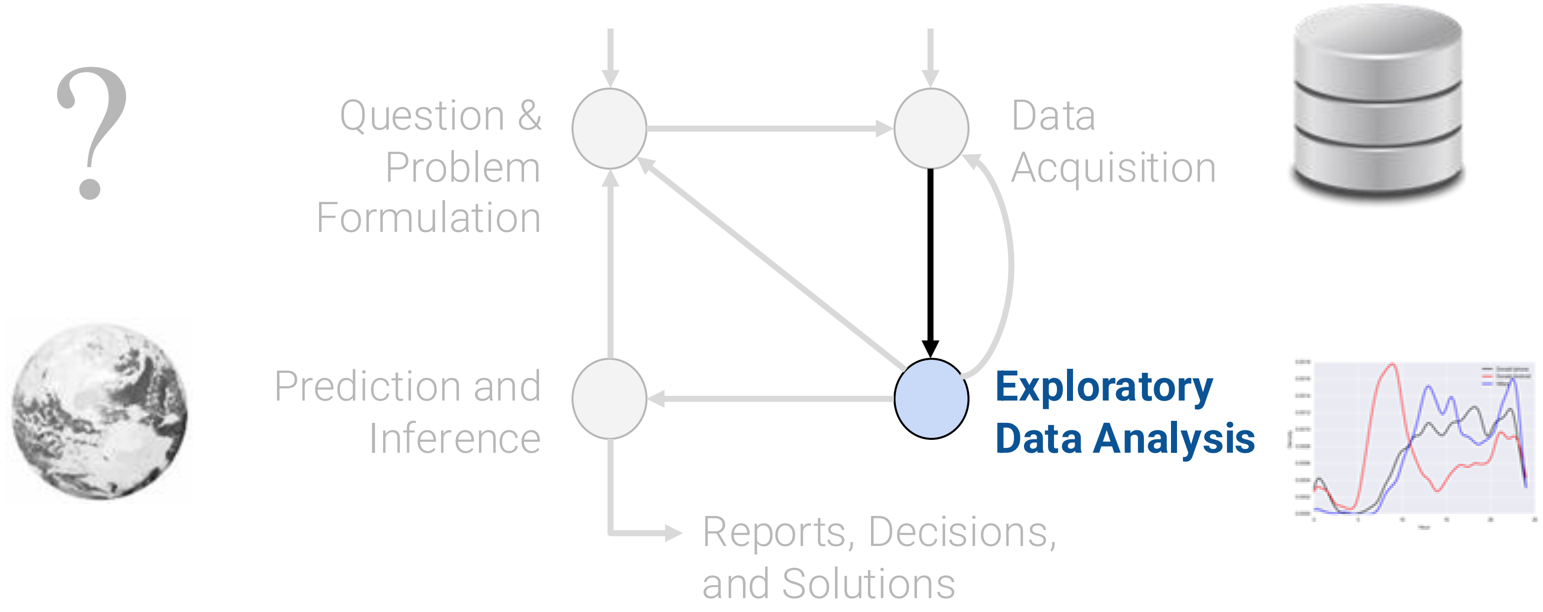
Lesson Learning Objectives:

- Recognize situations where aggregation is useful and identify the correct technique for performing an aggregation
- Define primary keys vs foreign keys; perform merges on DataFrames

Pandas Bootcamp Part 2

- Pandas Bootcamp Part 2:
  - Grouping
  - Joining

Plan for first 2 weeks



(Weeks 1 and 2)

EDA, Wrangling, and Data Visualization

Lesson Learning Objectives:

- Recognize situations where aggregation is useful and identify the correct technique for performing an aggregation

Grouping Data

- Finish Pandas Bootcamp:
  - **Grouping**
  - Joining

Learning Objectives:

- Use groupby to aggregate data

Groupby.agg

- **Groupby.agg**
- Some groupby.agg Puzzles

Why Group?

Our goal:

- Group together rows that fall under the same category.
  - For example, group together all rows from the same year.
- Perform an operation that *aggregates* across all rows in the category.
  - For example, sum up the total number of babies born in that year.

Grouping is a powerful tool to

1) perform large operations, all at once

and 2) summarize trends in a dataset.

`.groupby()`

A `.groupby()` operation involves some combination of **splitting the object, applying a function**, and **combining the results**.

- Calling `.groupby()` generates `DataFrameGroupBy` objects → "mini" sub-DataFrames
- Each subframe contains all rows that correspond to the same group (here, a particular year)

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
1	CO	F	2015	Emma	355
2	CO	F	1910	Frances	56
3	CO	F	2008	Galilea	6
4	CO	F	1910	Marie	32
5	CO	F	2015	Olivia	348

Original DataFrame

`.groupby("Year")` →

	State	Sex	Year	Name	Count
2	CO	F	1910	Frances	56
4	CO	F	1910	Marie	32

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
3	CO	F	2008	Galilea	6

	State	Sex	Year	Name	Count
1	CO	F	2015	Emma	355
5	CO	F	2015	Olivia	348

GroupBy Object

`.groupby().agg()`

- We cannot work directly with **`DataFrameGroupBy`** objects! The diagram below is to help understand what goes on conceptually – in reality, we can't "see" the result of calling `.groupby`.

- Instead, we transform a **`DataFrameGroupBy`** object back into a DataFrame using `.agg`
  - `.agg` is how we apply an aggregation operation to the data.

`babynames_temp.groupby("Year")`

`.agg({"Count":"sum"})`

or `[["Count"]].agg(sum)`

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
1	CO	F	2015	Emma	355
2	CO	F	1910	Frances	56
3	CO	F	2008	Galilea	6
4	CO	F	1910	Marie	32
5	CO	F	2015	Olivia	348

Original DataFrame

	State	Sex	Year	Name	Count
2	CO	F	1910	Frances	56
4	CO	F	1910	Marie	32

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
3	CO	F	2008	Galilea	6

	State	Sex	Year	Name	Count
1	CO	F	2015	Emma	355
5	CO	F	2015	Olivia	348

GroupBy Object

	Count
Year	
1910	88
2008	11
2015	703

Output DataFrame

Index of output is the col you grouped on

A Note on Nuisance Columns

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
1	CO	F	2015	Emma	355
2	CO	F	1910	Frances	56
3	CO	F	2008	Galilea	6
4	CO	F	1910	Marie	32
5	CO	F	2015	Olivia	348

```
babynames_temp.groupby("Year").agg({"Count":"sum"})
babynames_temp.groupby("Year")[["Count"]].agg(sum)
```

Year	Count
1910	88
2008	11
2015	703

If you don't specify the column to aggregate, the aggregation function will be applied to all columns:

```
babynames_temp.groupby("Year").agg(sum)
```

Year	State	Sex	Name	Count
1910	COCO	FF	FrancesMarie	88
2008	COCO	FF	BrittanyGalilea	11
2015	COCO	FF	EmmaOlivia	703

If the aggregation function can't be applied to all columns it results in a `TypeError`.

```
babynames_temp.groupby("Year").agg(mean)
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call
File /opt/conda/lib/python3.10/site-packages/pandas/core/groupby/grou
lues)
   1489 try:
-> 1490     result = self.grouper._cython_operation(
   1491         "aggregate",
   1492         values,
   1493         how,
```

Aggregation Functions

What goes inside of `.agg( )`?

- Any function that aggregates several values into one summary value. Common examples:

Built-in Python Functions	NumPy Functions	Built-In `pandas` functions	
`.agg(sum)`	`.agg(np.sum)`	`.agg("sum")`	Returns sum of each col in each group
`.agg(max)`	`.agg(np.max)`	`.agg("max")`	Returns max of each col in each group
`.agg(min)`	`.agg(np.min)`	`.agg("min")`	Returns min of each col in each group
	`.agg(np.mean)`	`.agg("mean")`	Returns mean of each col in each group
		`.agg("first")` `.agg("last")`	Returns first/last non-null entry in each group for each column
		`.agg("count")`	Returns counts of non-null values in each col of each group
		`.agg("size")`	Returns series counting # of rows in each group, including missing values

- You can also define your own function!

The built-in Pandas functions listed above can also be called directly, without the explicit use of `.agg( )`

```
babynames.groupby("Year").mean()
```

Practice

	alp	num	st
0	A	3	ak
1	B	1	tx
2	C	4	fl
3	A	1	hi
4	B	5	mi
5	C	9	ak
6	A	2	ca
7	C	5	sd
8	B	6	nc

groupby(“alp”)

	num	st
A	3	ak
A	1	hi
A	2	ca
B	1	tx
B	5	mi
B	6	nc
C	4	fl
C	9	ak
C	5	sd

.agg(“max”)

	num	st
A	??	??
B	??	??
C	??	??

What will go in the ??

Practice

	alp	num	st
0	A	3	ak
1	B	1	tx
2	C	4	fl
3	A	1	hi
4	B	5	mi
5	C	9	ak
6	A	2	ca
7	C	5	sd
8	B	6	nc

`groupby("alp")`

	num	st
A	3	ak
A	1	hi
A	2	ca

B	1	tx
B	5	mi
B	6	nc

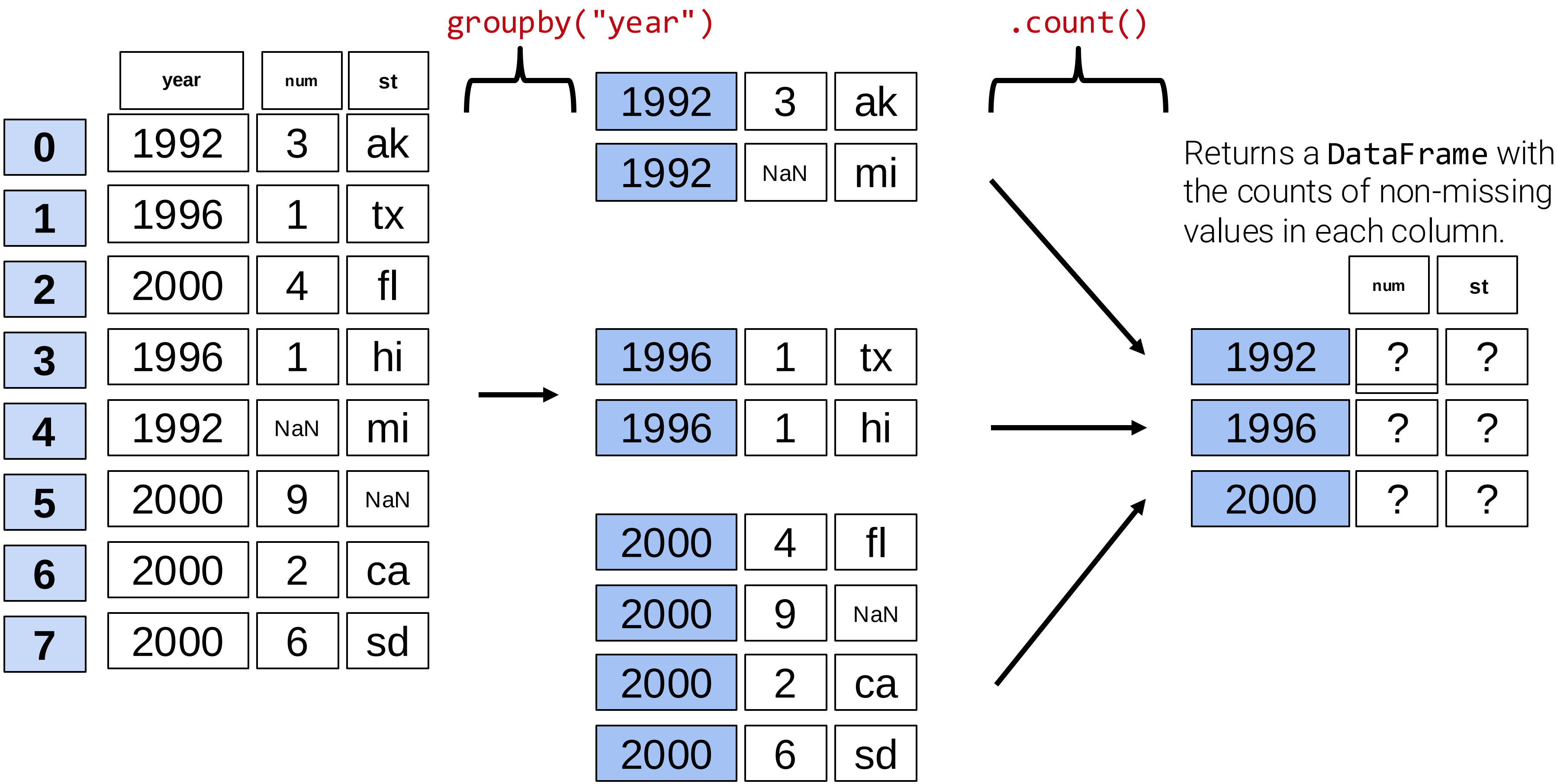
C	4	fl
C	9	ak
C	5	sd

`.agg("max")`

	num	st
A	3	hi
B	6	tx
C	9	sd

What will go in the ??

`groupby.count()`



Returns a `DataFrame` with the counts of non-missing values in each column.

groupby.count()

	year	num	st
0	1992	3	ak
1	1996	1	tx
2	2000	4	fl
3	1996	1	hi
4	1992	NaN	mi
5	2000	9	NaN
6	2000	2	ca
7	2000	6	sd

groupby("year")

1992	3	ak
1992	NaN	mi

1996	1	tx
1996	1	hi

2000	4	fl
2000	9	NaN
2000	2	ca
2000	6	sd

.count()

Returns a `DataFrame` with the counts of non-missing values in each column.

	num	st
1992	1	2
1996	2	2
2000	4	3

`groupby.size()`

	year	num	st
0	1992	3	ak
1	1996	1	tx
2	2000	4	fl
3	1996	1	hi
4	1992	NaN	mi
5	2000	9	NaN
6	2000	2	ca
7	2000	6	sd

`groupby("year")`

	num	st
1992	3	ak
1992	NaN	mi
1996	1	tx
1996	1	hi
2000	4	fl
2000	9	NaN
2000	2	ca
2000	6	sd

`.size()`

1992	2
1996	2
2000	4

Returns a `Series` object counting the number of rows in each group.

Similar to `value_counts()` except that `size()` does not sort the index based on the frequency of entries.

groupby.first()

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
1	CO	F	2015	Emma	355
2	CO	F	1910	Frances	56
3	CO	F	2008	Galilea	6
4	CO	F	1910	Marie	32
5	CO	F	2015	Olivia	348

Original DataFrame

.groupby("Year")

2	CO	F	1910	Frances	56
4	CO	F	1910	Marie	32

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
3	CO	F	2008	Galilea	6

	State	Sex	Year	Name	Count
1	CO	F	2015	Emma	355
5	CO	F	2015	Olivia	348

GroupBy Object

.first()

Year	State	Sex	Name	Count
1910	CO	F	Frances	56.0
2008	CO	F	Brittany	5.0
2015	CO	F	Brittany	355.0

The "first" row in each sub-DataFrame depends on how the original DataFrame is sorted:

```python
babynames_temp.sort_values(by="Count").groupby("Year").first()
```

Year	State	Sex	Name	Count
1910	CO	F	Marie	32.0
2008	CO	F	Brittany	5.0
2015	CO	F	Olivia	348.0

Aggregating the Same Column Using Multiple Aggregation Functions

```
babynames_temp.groupby("Year").agg({"Count":[max, min, sum]})
```

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
1	CO	F	2015	Emma	355
2	CO	F	1910	Frances	56
3	CO	F	2008	Galilea	6
4	CO	F	1910	Marie	32
5	CO	F	2015	Olivia	348

Original DataFrame

	Count		
	max	**min**	**sum**
Year			
1910	56	32	88
2008	6	5	11
2015	355	348	703

Aggregating Different Columns Using Different Functions

```
babynames_temp.groupby("Year").agg({"Count":max, "Name":min})
```

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
1	CO	F	2015	Emma	355
2	CO	F	1910	Frances	56
3	CO	F	2008	Galilea	6
4	CO	F	1910	Marie	32
5	CO	F	2015	Olivia	348

Original DataFrame

	Count	Name
Year		
1910	56	Frances
2008	6	Brittany
2015	355	Emma

Notice, the column names don't indicate how they've been aggregated.

```
.rename(columns={"Count":"MaxCount", "Name":"MinName"})
```

	MaxCount	MinName
Year		
1910	56	Frances
2008	6	Brittany
2015	355	Emma

Grouping by Multiple Columns

Suppose we want to build a table showing the total number of babies born of each sex in each year. One way is to `groupby` *using both columns* of interest:

```
babynames.groupby(["Year", "Sex"])[["Count"]].agg(sum).head(6)
```

Year	Sex	Count
1910	F	2471
	M	1138
1911	F	2343
	M	1307
1912	F	3251
	M	2633

Note: Resulting `DataFrame` is multi-indexed. That is, its index has multiple dimensions.

Can reset using

`.reset_index()`

	Year	Sex	Count
0	1910	F	2471
1	1910	M	1138
2	1911	F	2343
3	1911	M	1307
4	1912	F	3251
5	1912	M	2633

Learning Objectives:

- Use groupby to aggregate data

Groupby.agg

- Groupby.agg
- **Some groupby.agg Puzzles**

Groupby Puzzle 1

Goal: Find the baby name with sex "F" that has fallen in popularity the most.

```
f_babynames = babynames[babynames["Sex"] == "F"]
f_babynames = f_babynames.sort_values(["Year"])
jenn_counts_series = f_babynames[f_babynames["Name"] == "Jennifer"]["Count"]
```

Number of Jennifers Born in Colorado Per Year

What Is "Popularity"?

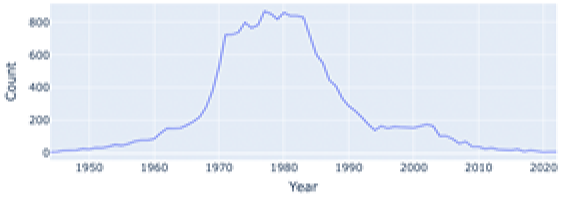
Goal: Find the baby name with sex "F" that has fallen in popularity the most.

How do we define "fallen in popularity?"

- Let's create a metric: "ratio to peak" (RTP).
- The RTP is the ratio of babies born with a given name in the year of the most recent data we have for that name to the *maximum* number of babies born with that name in *any* year.

Example for "Jennifer":

- In 1977, we hit peak Jennifer: 866 Jennifers were born.
- In 2022, there were only 8 Jennifers.
- RTP is 8 /866 = 0.00923…



Calculating RTP

```
max_jenn = max(f_babynames[f_babynames["Name"] == "Jennifer"]["Count"])
866
```

```
curr_jenn = f_babynames[f_babynames["Name"] == "Jennifer"]["Count"].iloc[-1]
8
```

Remember: `f_babynames` is sorted by year.
`.iloc[-1]` means "grab the latest year"

```
rtp = curr_jenn / max_jenn
0.009237875288683603
```

```
def ratio_to_peak(series):
    return series.iloc[-1] / max(series)
```

```
jenn_counts_ser = f_babynames[f_babynames["Name"] == "Jennifer"]["Count"]
ratio_to_peak(jenn_counts_ser)
0.009237875288683603
```

Calculating RTP Using `.groupby()`

`.groupby()` makes it easy to compute the RTP for all names at once!

```
rtp_table = f_babynames.groupby("Name")[["Year","Count"]].agg(ratio_to_peak)
```

	Count	Year
Name		
Aadhya	1.000000	1.0
Aaliyah	0.523256	1.0
Aanya	1.000000	1.0
Aaralyn	0.714286	1.0
Aarna	1.000000	1.0
...	...	...
Zora	1.000000	1.0
Zoya	1.000000	1.0
Zuleyka	1.000000	1.0
Zuri	1.000000	1.0
Zyla	1.000000	1.0

3568 rows × 2 columns

Poll

In the 10 rows shown, note the Year is 1.0 for every value.

Are there any rows for which Year is **not** 1.0?

A. Yes, names that appeared for the first time in 2022.
B. Yes, names that did not appear in 2022.
C. Yes, names whose peak Count was in 2022.
D. No, every row has a Year value of 1.0.

```python
rtp_table = (
    f_babynames
    .groupby("Name")[["Year","Count"]]
    .agg(ratio_to_peak)
)
```

Name	Count	Year
Aadhya	1.000000	1.0
Aaliyah	0.523256	1.0
Aanya	1.000000	1.0
Aaralyn	0.714286	1.0
Aarna	1.000000	1.0
...	...	...
Zora	1.000000	1.0
Zoya	1.000000	1.0
Zuleyka	1.000000	1.0
Zuri	1.000000	1.0
Zyla	1.000000	1.0

3568 rows × 2 columns

Answer

In the five rows shown, note the Year is 1.0 for every value.

Are there any rows for which Year is **not** 1.0?

A. Yes, names that appeared for the first time in 2022.
B. Yes, names that did not appear in 2022.
C. Yes, names whose peak Count was in 2022.
D. No, every row has a Year value of 1.0.

```
rtp_table = (
    f_babynames
    .groupby("Name")[["Year","Count"]]
    .agg(ratio_to_peak)
    )
```

Name	Count	Year
Aadhya	1.000000	1.0
Aaliyah	0.523256	1.0
Aanya	1.000000	1.0
Aaralyn	0.714286	1.0
Aarna	1.000000	1.0
...	...	...
Zora	1.000000	1.0
Zoya	1.000000	1.0
Zuleyka	1.000000	1.0
Zuri	1.000000	1.0
Zyla	1.000000	1.0

3568 rows × 2 columns

A Note on Nuisance Columns

At least as of the time of this slide creation, executing our agg call results in a `TypeError`.

```python
f_babynames.groupby("Name").agg(ratio_to_peak)
```

```
Cell In[110], line 5, in ratio_to_peak(series)
      1 def ratio_to_peak(series):
      2     """
      3     Compute the RTP for a Series containing the counts per year for a single name
      4     """
----> 5     return series.iloc[-1] / np.max(series)

TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

A Note on Nuisance Columns

Below, we explicitly select the column(s) we want to apply our aggregation function to **BEFORE** calling `agg`. This avoids the warning (and can prevent unintentional loss of data).

```
rtp_table = f_babynames.groupby("Name")[["Count"]].agg(ratio_to_peak)
```

	Count
Name	
Aadhya	1.000000
Aaliyah	0.523256
Aanya	1.000000
Aaralyn	0.714286
Aarna	1.000000
...	...
Zora	1.000000
Zoya	1.000000
Zuleyka	1.000000
Zuri	1.000000
Zyla	1.000000

Renaming Columns After Grouping

By default, `.groupby` will not rename any aggregated columns (the column is still named "Count", even though it now represents the RTP.

For better readability, we may wish to rename “Count” to “Count RTP”

```
rtp_table = female_babynames.groupby("Name")[["Count"]].agg(ratio_to_peak)
rtp_table = rtp_table.rename(columns = {"Count": "Count RTP"})
```

Name	Count
Aadhya	1.000000
Aaliyah	0.523256
Aanya	1.000000
Aaralyn	0.714286
Aarna	1.000000
...	...

→

Name	Count RTP
Aadhya	1.000000
Aaliyah	0.523256
Aanya	1.000000
Aaralyn	0.714286
Aarna	1.000000
...	...

Some Data Science Payoff

By sorting `rtp_table` we can see the names whose popularity has decreased the most.

```
rtp_table.sort_values("Count RTP")
```

	Count RTP
Name	
Linda	0.006750
Debra	0.007728
Amanda	0.007974
Jennifer	0.009238
Patricia	0.010309
...	...
Julietta	1.000000
Juliann	1.000000
Jules	1.000000
Kaida	1.000000
Zyla	1.000000

3568 rows × 1 columns

Groupby Puzzle 1: Some Data Science Payoff

By sorting `rtp_table` we can see the names whose popularity has decreased the most.

```
rtp_table.sort_values("Count RTP")
```

	Count RTP
Name	
Linda	0.006750
Debra	0.007728
Amanda	0.007974
Jennifer	0.009238
Patricia	0.010309
...	...
Julietta	1.000000
Juliann	1.000000
Jules	1.000000
Kaida	1.000000
Zyla	1.000000

3568 rows × 1 columns

```
px.line(f_babynames[f_babyname["Name"] == "Linda")],
                        x = "Year", y = "Count")
```

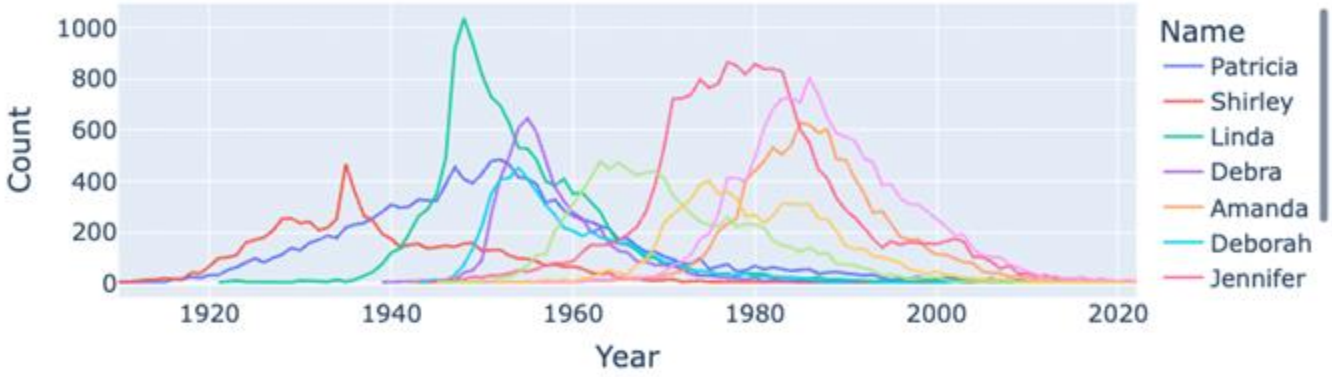
Some Data Science Payoff

We can get the list of the top 10 names and then plot popularity with::

```
top10 = rtp_table.sort_values("Count RTP").head(10).index
```

```
Index(['Linda', 'Debra', 'Amanda', 'Jennifer', 'Patricia', 'Lisa', 'Shirley',
       'Deborah', 'Jessica', 'Heather'],
      dtype='object', name='Name')
```

```
px.line(f_babynames[f_babyname["Name"].isin(top10)],
                      x = "Year", y = "Count", color = "Name")
```



Practice! GroupBy Puzzle 2

a). Write code to compute the total number of babies with each name.

b). Write code to compute the total number of babies born each year.

Answer: Part A

Before, we saw that the code below generates the Count RTP for all female names.

```
babynames.groupby("Name")[["Count"]].agg(ratio_to_peak)
```

We use similar logic to compute the summed counts of all baby names.

```
babynames.groupby("Name")[["Count"]].agg(sum)
```

or

```
babynames.groupby("Name")[["Count"]].sum()
```

	Count
Name	
Aaden	65
Aadhya	10
Aaliyah	1469
Aanya	5
Aaralyn	12
...	...
Zuleyka	5
Zuri	112
Zyaire	35
Zyla	11
Zyon	10

Answer: Part B

Now, we create groups for each *year*.

```
babynames.groupby("Year")[["Count"]].agg(sum)
```

or

```
babynames.groupby("Year")[["Count"]].sum()
```

or

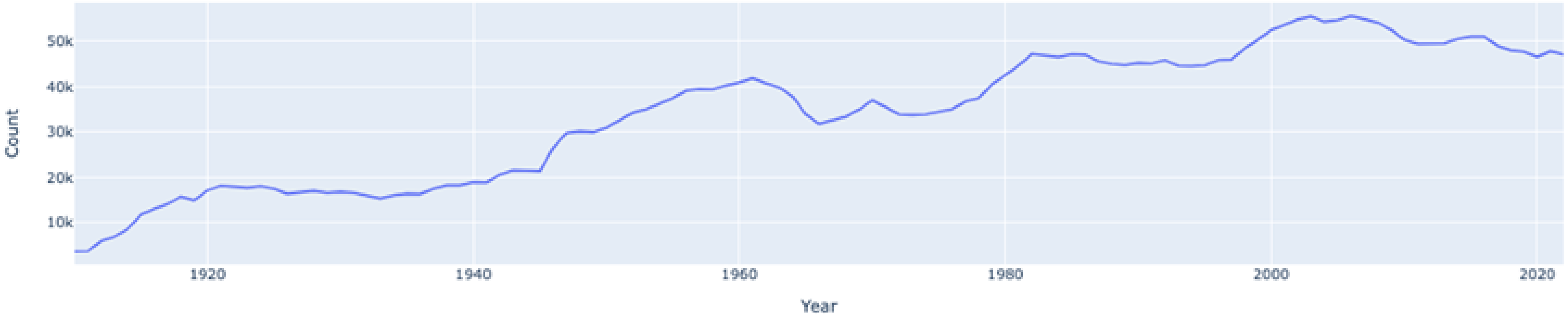
```
babynames.groupby("Year").sum(numeric_only=True)
```

	Count
Year	
1910	3609
1911	3650
1912	5884
1913	6831
1914	8528
...	...
2018	48008
2019	47736
2020	46548
2021	47815
2022	47048

Plotting Birth Counts

Plotting the `DataFrame` we just generated tells an interesting story.

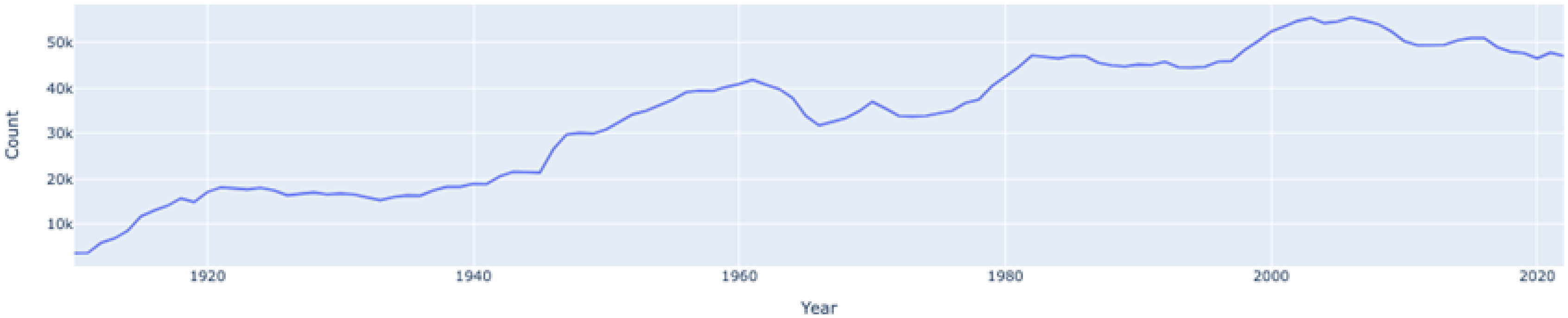
```
puzzle2 = babynames.groupby("Year")[["Count"]].agg(sum)
px.line(puzzle2, y = "Count")
```



A Word of Warning!

We made an enormous assumption when we decided to use this dataset to estimate the birth rate.

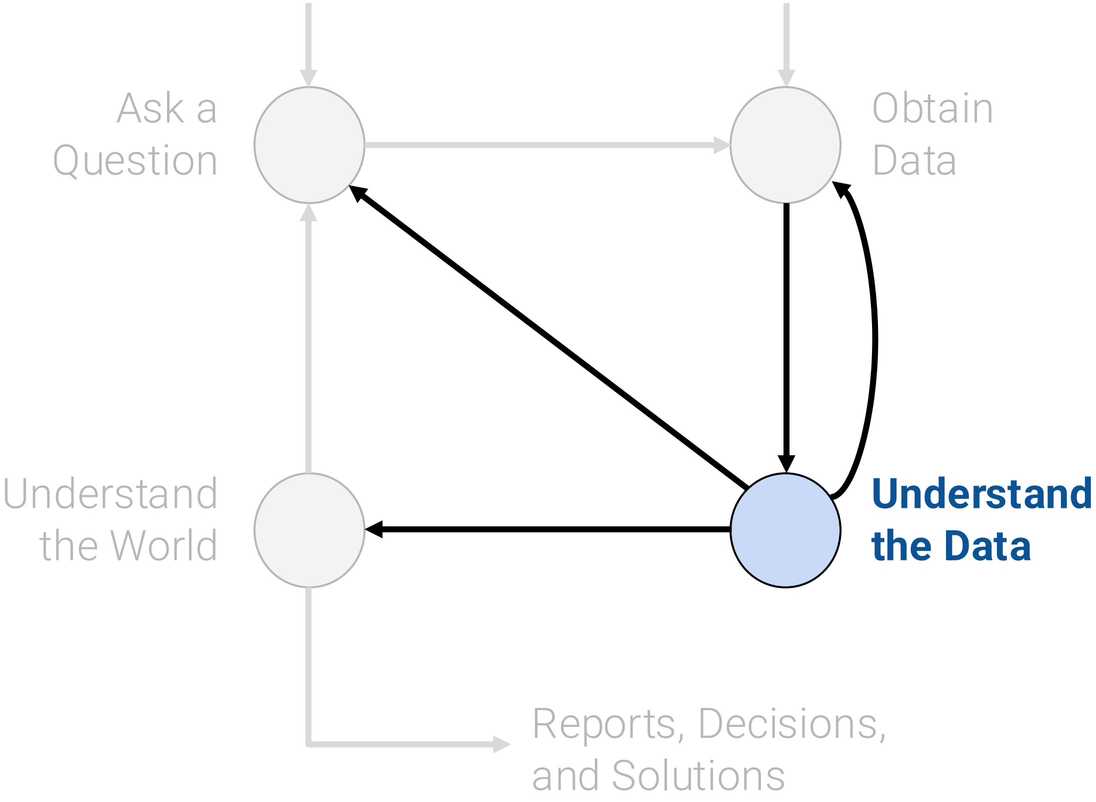
- According to https://cohealthviz.dphe.state.co.us/t/HealthInformaticsPublic/views/COHIDLiveBirthsDashboard/LiveBirthStatistics the true number of babies born in Colorado in 2020 was 61,496 but our plot shows 46,548 babies.
- What happened?



Recall: Exploratory Data Analysis and Visualization

- How is our data organized and what does it contain?
- Do we already have relevant data?
- What are the biases, anomalies, or other issues with the data?
- How do we transform the data to enable effective analysis?

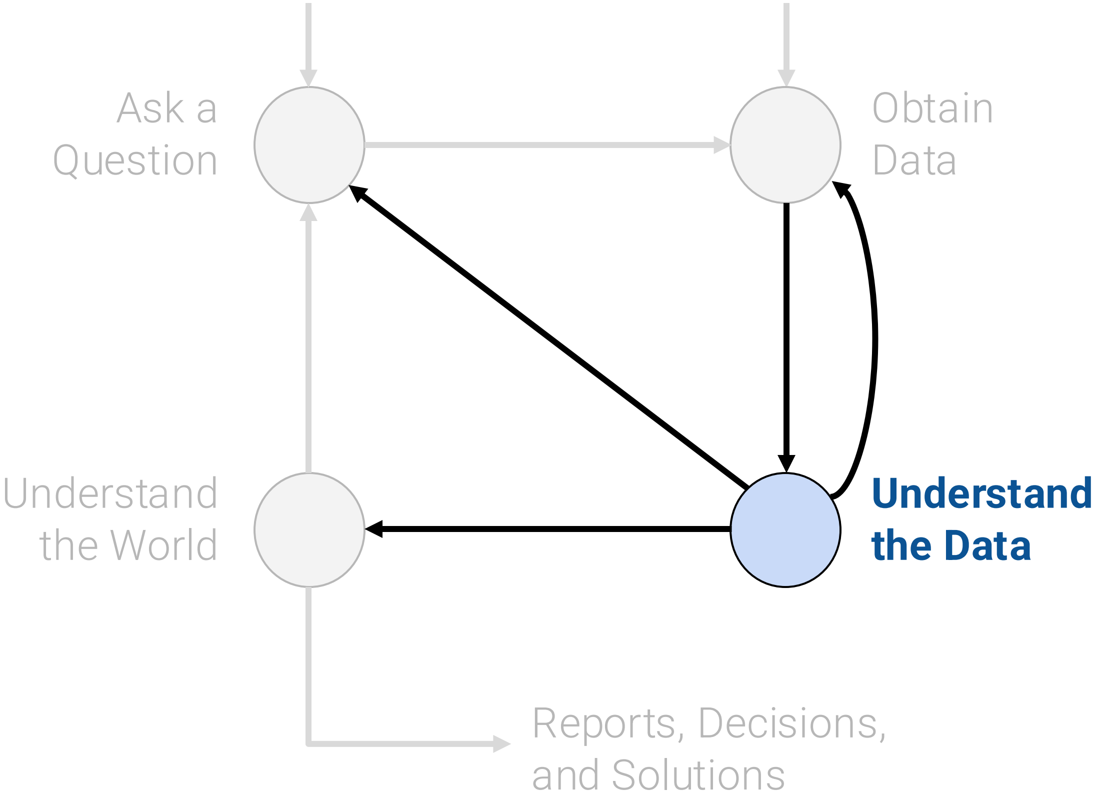
Bottom line: Blindly using tools is dangerous!



Recall: Exploratory Data Analysis and Visualization

What are the biases, anomalies, or other issues with the data?

- The database does not include names of popularity less than 5 per year
- Not all babies register for social security.



Lesson Learning Objectives:

- Recognize situations where aggregation is useful and identify the correct technique for performing an aggregation
- Define primary keys vs foreign keys; perform merges on DataFrames

Merging DataFrames

- Finish Pandas Bootcamp:
  - Grouping
  - **Joining**

Joining DataFrames

Suppose want to know how many babies born in Colorado in 2023 share a name with a former US president.

To solve this problem, we'll have to join DataFrames

Structure: Primary Keys and Foreign Keys

Sometimes your data comes in multiple files:

- Often data will reference other pieces of data.
- Alternatively, you will collect multiple pieces of related data.

Use `.merge()` to **join** data on **keys**.

Customers.csv

CustID	Addr
171345	Harmon..
281139	Main ..

Orders.csv

OrderNum	CustID	Date
1	171345	8/21/2017
2	281139	8/30/2017

Products.csv

ProdID	Cost
42	3.14
999	2.72

Purchases.csv

OrderNum	ProdID	Quantity
1	42	3
1	999	2
2	42	1

Structure: Primary Keys and Foreign Keys

Sometimes your data comes in multiple files:

- Often data will reference other pieces of data.
- Alternatively, you will collect multiple pieces of related data.

Use `.merge()` to join data on **keys**.

Primary key: the column or set of columns in a table that *uniquely* determine the values in the remaining columns

- Primary keys are unique, but could be tuples.
- Examples: SSN, ProductIDs, …

Primary Key

Customers.csv

CustID	Addr
171345	Harmon..
281139	Main ..

Primary Key

Orders.csv

OrderNum	CustID	Date
1	171345	8/21/2017
2	281139	8/30/2017

Products.csv

ProdID	Cost
42	3.14
999	2.72

Primary Key

Purchases.csv

OrderNum	ProdID	Quantity
1	42	3
1	999	2
2	42	1

Structure: Primary Keys and Foreign Keys

Sometimes your data comes in multiple files:

- Often data will reference other pieces of data.
- Alternatively, you will collect multiple pieces of related data.

Use `.merge()` to join data on **keys**.

Primary key: the column or set of columns in a table that determine the values of the remaining columns

- Primary keys are unique, but could be tuples.
- Examples: SSN, ProductIDs, …

Foreign keys: the column or sets of columns that reference primary keys in other tables.

Primary Key

Customers.csv

CustID	Addr
171345	Harmon..
281139	Main ..

Foreign Key

Orders.csv

OrderNum	CustID	Date
1	171345	8/21/2017
2	281139	8/30/2017

Products.csv

ProdID	Cost
42	3.14
999	2.72

Purchases.csv

OrderNum	ProdID	Quantity
1	42	3
1	999	2
2	42	1

Merging on columns

- Basic syntax for joining two dataframes **df** and **df2**

OPTION 1

```
pd.merge(left = df, right = df2,
                   how = "inner",
         left_on = "col_label_in_df",
          right_on = "col_label_in_df2")
```

```
df.merge(df2,
    how = "inner",
    left_on = "column_label_in_df",
    right_on = "column_label_in_df2")
```

	id	name
0	1	Tom
1	2	Jenny
2	3	James
3	4	Dan

df_customer

	customer_id	age	sex
0	2	31	F
1	3	20	M
2	4	40	M
3	5	70	F

df_info_2

```
df_customer.merge(df_info_2, left_on='id',right_on='customer_id')
```

	id	name	customer_id	age	sex
0	2	Jenny	2	31	F
1	3	James	3	20	M
2	4	Dan	4	40	M

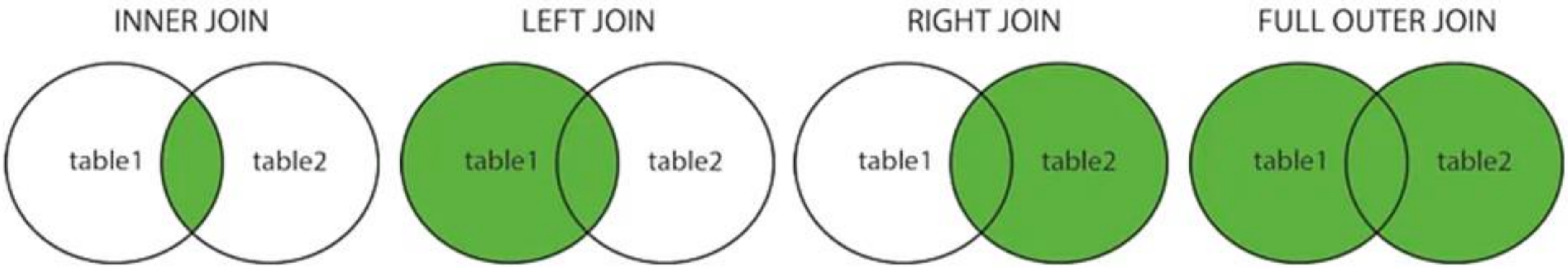
The default setting is Inner Join (so it will only keep the rows that have matching keys in both dataframes).

Merging - More options

```
pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None,
         left_index=False, right_index=False, sort=True,
         suffixes=('_x', '_y'), copy=True, indicator=False)
```

- `left`: A DataFrame object
- `right`: Another DataFrame object
- `on`: Columns (names) to join on. Must be found in both the left and right DataFrame objects. If not passed and `left_index` and `right_index` are `False`, the intersection of the columns in the DataFrames will be inferred to be the join keys
- `left_on`: Columns from the left DataFrame to use as keys. Can either be column names or arrays with length equal to the length of the DataFrame
- `right_on`: Columns from the right DataFrame to use as keys. Can either be column names or arrays with length equal to the length of the DataFrame
- `left_index`: If `True`, use the index (row labels) from the left DataFrame as its join key(s). In the case of a DataFrame with a MultiIndex (hierarchical), the number of levels must match the number of join keys from the right DataFrame
- `right_index`: Same usage as `left_index` for the right DataFrame
- `how`: One of `'left'`, `'right'`, `'outer'`, `'inner'`. Defaults to `inner`. See below for more detailed description of each method
- `sort`: Sort the result DataFrame by the join keys in lexicographical order. Defaults to `True`, setting to `False` will improve performance substantially in many cases
- `suffixes`: A tuple of string suffixes to apply to overlapping columns. Defaults to `('_x', '_y')`.
- `copy`: Always copy data (default `True`) from the passed DataFrame objects, even when reindexing is not necessary. Cannot be avoided in many cases but may improve performance / memory usage. The cases where copying can be avoided are somewhat pathological but this option is provided nonetheless.
- `indicator`: Add a column to the output DataFrame called `_merge` with information on the source of each row. `_merge` is Categorical-type and takes on a value of `left_only` for observations whose merge key only appears in `'left'` DataFrame, `right_only` for observations whose merge key only appears in `'right'` DataFrame, and `both` if the observation's merge key is found in both.

Joining Tables: Types of Joins



- `inner` : the default join type in Pandas `merge()` function and it produces records that have matching values in both DataFrames
- `left` : produces all records from the left DataFrame and the matched records from the right DataFrame
- `right` : produces all records from the right DataFrame and the matched records from the left DataFrame
- `outer` : produces all records when there is a match in either left or right DataFrame

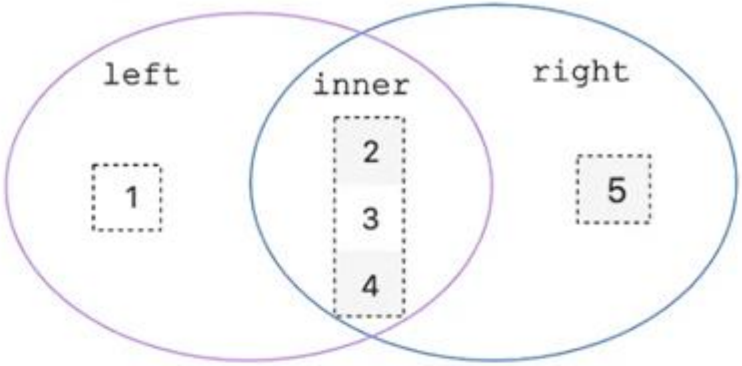
Joining Tables

	id	name
0	1	Tom
1	2	Jenny
2	3	James
3	4	Dan

df_customer

	id	age	sex
0	2	31	F
1	3	20	M
2	4	40	M
3	5	70	F

df_info



Creating Table 1: Babynames in 2023

babynames_temp.groupby("Year"):

```
babynames_2023 = babynames[babynames["Year"] == 2023]
babynames_2023
```

	State	Sex	Year	Name	Count
63786	CO	F	2023	Charlotte	288
63787	CO	F	2023	Olivia	265
63788	CO	F	2023	Sophia	212
63789	CO	F	2023	Emma	211
63790	CO	F	2023	Amelia	200
63791	CO	F	2023	Mia	200
63792	CO	F	2023	Evelyn	187
63793	CO	F	2023	Isabella	174
63794	CO	F	2023	Harper	169
63795	CO	F	2023	Ava	153

Creating Table 2: Presidents with First Names

To join our table, we'll also need to set aside the first names of each candidate.

```
elections["First Name"] = elections["Candidate"].str.split().str[0]
```

	Year	Candidate	Party	Popular vote	Result	%	First Name
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122	Andrew
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878	John
2	1828	Andrew Jackson	Democratic	642806	win	56.203927	Andrew
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073	John
4	1832	Andrew Jackson	Democratic	702735	win	54.574789	Andrew
...	...	...	...	...	...	...	...
177	2016	Jill Stein	Green	1457226	loss	1.073699	Jill
178	2020	Joseph Biden	Democratic	81268924	win	51.311515	Joseph
179	2020	Donald Trump	Republican	74216154	loss	46.858542	Donald
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979	Jo
181	2020	Howard Hawkins	Green	405035	loss	0.255731	Howard

182 rows × 7 columns

Joining Our Tables: Two Options

```
merged = pd.merge(left = elections, right = babynames_2023,
                  left_on = "First Name", right_on = "Name")
```

```
OR:
merged = elections.merge(right = babynames_2023,
                  left_on = "First Name", right_on = "Name")
```

	Year_x	Candidate	Party	Popular vote	Result	%	First Name	First_Name	State	Sex	Year_y	Name	Count
0	2020	Joseph Biden	Democratic	81268924	win	51.311515	Joseph	Joseph	CO	M	2023	Joseph	87
1	2020	Donald Trump	Republican	74216154	loss	46.858542	Donald	Donald	CO	M	2023	Donald	8
2	2016	Donald Trump	Republican	62984828	win	46.407862	Donald	Donald	CO	M	2023	Donald	8
3	2020	Howard Hawkins	Green	405035	loss	0.255731	Howard	Howard	CO	M	2023	Howard	7
4	1996	Howard Phillips	Taxpayers	184656	loss	0.192045	Howard	Howard	CO	M	2023	Howard	7

Supporting Materials

Supporting Materials:

GroupBy Practice

Back to the Elections Dataset

- For the next practice problems, we'll be working with the elections dataset that we practiced with when first introducing Pandas:

	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

More on `DataFrameGroupby` Object

We can look into **`DataFrameGroupby`** objects in following ways:

```
grouped_by_party = elections.groupby("Party")
grouped_by_party.groups
```

```
{'American': [22, 126], 'American Independent': [115, 119, 124], 'Anti-Masonic': [6], 'Anti-Monopoly': [38], 'Citiz
ens': [127], 'Communist': [89], 'Constitution': [160, 164, 172], 'Constitutional Union': [24], 'Democratic': [2, 4,
8, 10, 13, 14, 17, 20, 28, 29, 34, 37, 39, 45, 47, 52, 55, 57, 64, 70, 74, 77, 81, 83, 86, 91, 94, 97, 100, 105, 10
8, 111, 114, 116, 118, 123, 129, 134, 137, 140, 144, 151, 158, 162, 168, 176, 178], 'Democratic-Republican': [0,
1], 'Dixiecrat': [103], 'Farmer-Labor': [78], 'Free Soil': [15, 18], 'Green': [149, 155, 156, 165, 170, 177, 181],
'Greenback': [35], 'Independent': [121, 130, 143, 161, 167, 174], 'Liberal Republican': [31], 'Libertarian': [125,
128, 132, 138, 139, 146, 153, 159, 163, 169, 175, 180], 'National Democratic': [50], 'National Republican': [3, 5],
'National Union': [27], 'Natural Law': [148], 'New Alliance': [136], 'Northern Democratic': [26], 'Populist': [48,
61, 141], 'Progressive': [68, 82, 101, 107], 'Prohibition': [41, 44, 49, 51, 54, 59, 63, 67, 73, 75, 99], 'Reform':
[150, 154], 'Republican': [21, 23, 30, 32, 33, 36, 40, 43, 46, 53, 56, 60, 65, 69, 72, 79, 80, 84, 87, 90, 96, 98,
104, 106, 109, 112, 113, 117, 120, 122, 131, 133, 135, 142, 145, 152, 157, 166, 171, 173, 179], 'Socialist': [58, 6
2, 66, 71, 76, 85, 88, 92, 95, 102], 'Southern Democratic': [25], 'States' Rights': [110], 'Taxpayers': [147], 'Uni
on': [93], 'Union Labor': [42], 'Whig': [7, 9, 11, 12, 16, 19]}
```

```
grouped_by_party.get_group("Socialist")
```

	Year	Candidate	Party	Popular vote	Result	%
58	1904	Eugene V. Debs	Socialist	402810	loss	2.985897
62	1908	Eugene V. Debs	Socialist	420852	loss	2.850866
66	1912	Eugene V. Debs	Socialist	901551	loss	6.004354
71	1916	Allan L. Benson	Socialist	590524	loss	3.194193

Groupby Puzzle #3

Why does the table seem to claim that Woodrow Wilson won the presidency in 2020?

```
elections.groupby("Party").agg(max).head(10)
```

	Year	Candidate	Popular vote	Result	%
Party					
American	1976	Thomas J. Anderson	873053	loss	21.554001
American Independent	1976	Lester Maddox	9901118	loss	13.571218
Anti-Masonic	1832	William Wirt	100715	loss	7.821583
Anti-Monopoly	1884	Benjamin Butler	134294	loss	1.335838
Citizens	1980	Barry Commoner	233052	loss	0.270182
Communist	1932	William Z. Foster	103307	loss	0.261069
Constitution	2016	Michael Peroutka	203091	loss	0.152398
Constitutional Union	1860	John Bell	590901	loss	12.639283
Democratic	2020	Woodrow Wilson	81268924	win	61.344703
Democratic-Republican	1824	John Quincy Adams	151271	win	57.210122

Groupby Puzzle #3

Why does the table seem to claim that Woodrow Wilson won the presidency in 2020?

```
elections.groupby("Party").agg(max).head(10)
```

Every column is calculated independently! Among Democrats:

- Last year they ran: 2020
- Alphabetically latest candidate name: Woodrow Wilson
- Highest % of vote: 61.34

Party	Year	Candidate	Popular vote	Result	%
American	1976	Thomas J. Anderson	873053	loss	21.554001
American Independent	1976	Lester Maddox	9901118	loss	13.571218
Anti-Masonic	1832	William Wirt	100715	loss	7.821583
Anti-Monopoly	1884	Benjamin Butler	134294	loss	1.335838
Citizens	1980	Barry Commoner	233052	loss	0.270182
Communist	1932	William Z. Foster	103307	loss	0.261069
Constitution	2016	Michael Peroutka	203091	loss	0.152398
Constitutional Union	1860	John Bell	590901	loss	12.639283
Democratic	2020	Woodrow Wilson	81268924	win	61.344703
Democratic-Republican	1824	John Quincy Adams	151271	win	57.210122

Groupby Puzzle #4

Try to write code that returns the table below.

- Each row shows the best result (in %) by each party.
    - For example: Best Democratic result ever was Johnson's 1964 win.

	Year	Candidate	Popular vote	Result	%
Party					
American	1856	Millard Fillmore	873053	loss	21.554001
American Independent	1968	George Wallace	9901118	loss	13.571218
Anti-Masonic	1832	William Wirt	100715	loss	7.821583
Anti-Monopoly	1884	Benjamin Butler	134294	loss	1.335838
Citizens	1980	Barry Commoner	233052	loss	0.270182
Communist	1932	William Z. Foster	103307	loss	0.261069
Constitution	2008	Chuck Baldwin	199750	loss	0.152398
Constitutional Union	1860	John Bell	590901	loss	12.639283
Democratic	1964	Lyndon Johnson	43127041	win	61.344703

Groupby Puzzle #4

Try to write code that returns the table below.

- First sort the DataFrame so that rows are in descending order of %.
- Then group by Party and take the first item of each series.

```
elections_sorted_by_percent = elections.sort_values("%", ascending=False)
elections_sorted_by_percent.groupby("Party").first()
```

	Year	Candidate	Party	Popular vote	Result	%
114	1964	Lyndon Johnson	Democratic	43127041	win	61.344703
91	1936	Franklin Roosevelt	Democratic	27752648	win	60.978107
120	1972	Richard Nixon	Republican	47168710	win	60.907806
79	1920	Warren Harding	Republican	16144093	win	60.574501
133	1984	Ronald Reagan	Republican	54455472	win	59.023326

elections_sorted_by_percent

→

	Year	Candidate	Popular vote	Result	%
Party					
American	1856	Millard Fillmore	873053	loss	21.554001
American Independent	1968	George Wallace	9901118	loss	13.571218
Anti-Masonic	1832	William Wirt	100715	loss	7.821583
Anti-Monopoly	1884	Benjamin Butler	134294	loss	1.335838
Citizens	1980	Barry Commoner	233052	loss	0.270182
Communist	1932	William Z. Foster	103307	loss	0.261069
Constitution	2008	Chuck Baldwin	199750	loss	0.152398
Constitutional Union	1860	John Bell	590901	loss	12.639283
Democratic	1964	Lyndon Johnson	43127041	win	61.344703

Groupby Puzzle #4 - Alternate Approaches

Using a **lambda** function

```
elections_sorted_by_percent = elections.sort_values("%", ascending=False)
elections_sorted_by_percent.groupby("Party").agg(lambda x : x.iloc[0])
```

Using **idxmax** function

```
best_per_party = elections.loc[elections.groupby("Party")["%"].idxmax()]
```

Using **drop_duplicates** function

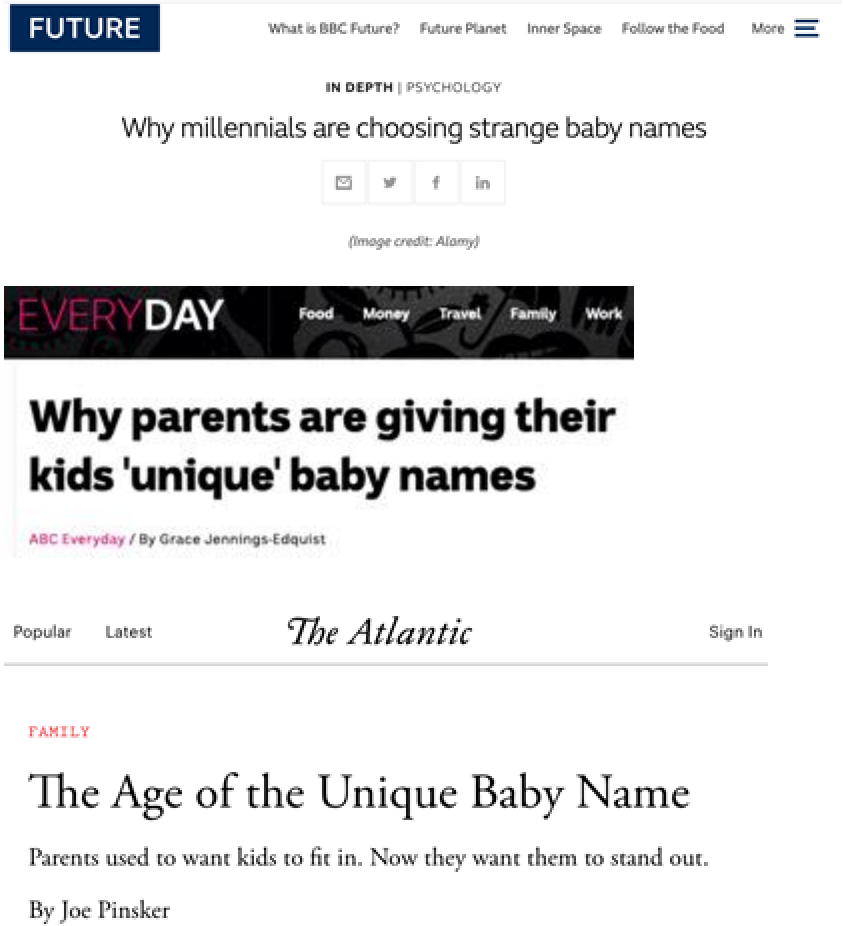
```
best_per_party2 = elections.sort_values("%").drop_duplicates(["Party"], keep="last")
```

There's More Than One Way to Find the Best Result by Party

In `Pandas`, there's more than one way to get to the same answer.

- Each approach has different tradeoffs in terms of readability, performance, memory consumption, complexity, etc.
- Takes a very long time to understand these tradeoffs!
- If you find your current solution to be particularly convoluted or hard to read, maybe try finding another way!

A fun little data science personal project?



- Are there enough unique baby names in recent years to skew these data significantly?
- How would you test that?