CMEDS

Civil & Mineral Engineering
UNIVERSITY OF TORONTO

CME538 Introduction to Data Science

Week 2 | Lecture 2 (2.2)

Pandas II.

Pandas II

- String **.str** Methods
- Grouping

Pandas II

- **String .str Methods**
- Grouping

String .str Methods

- The Pandas library provides a suite of tools for string/text manipulation.
- **.str** provides access to vectorized string functions for Series and Index.

String .str Methods

- Let's review some useful **.str** methods using the Baby Names dataset.
- These are baby names from the state of New York from 1910 to 2018.

	State	Sex	Year	Name	Count
942	NY	F	1912	Irene	411
136473	NY	F	2006	Kelis	31
55082	NY	F	1967	Grace	161
258532	NY	M	2000	Markus	17
173616	NY	M	1918	Dewitt	14

Index

State Series

Sex Series

Year Series

Name Series

Count Series

String .str Methods

- So, let's say you want to filter the DataFrame to only include names that start with the letter 'J'.
  - John
  - Janice
  - Josephine
  - Jane
- We could use the [ ] operator and input a list of Booleans.

We can first use Python list comprehension, which was reviewed in Lecture 3 and covered in APS106, to create a Boolean list. The value is True when the name starts with **J** and False when it does not.

```
starts_with_j = [x[0] == 'J' for x in baby_names['Name']]
starts_with_j[0:10]
```

```
[False, False, False, False, False, False, False, False, Fal
se, False]
```

Next, we can use the Boolean list to filter our DataFrame.

```
baby_names[starts_with_j].head()
```

	State	Sex	Year	Name	Count
14	NY	F	1910	Josephine	431
29	NY	F	1910	Jean	250
30	NY	F	1910	Julia	245
44	NY	F	1910	Jennie	178
84	NY	F	1910	Jane	84

String .str Methods

- So, let's say you want to filter the DataFrame to only include names that start with the letter 'J'.
  - John
  - Janice
  - Josephine
  - Jane
- We could also use **.str.startswith('J')**

This method is preferable.
- Idiomatic, easy to understand.

```
baby_names['Name'].str.startswith('J').head()
```

```
0    False
1    False
2    False
3    False
4    False
Name: Name, dtype: bool
```

This produces a Boolean Series which can then be used to filter our DataFrame.

```
baby_names[baby_names['Name'].str.startswith('J')].head()
```

	State	Sex	Year	Name	Count
14	NY	F	1910	Josephine	431
29	NY	F	1910	Jean	250
30	NY	F	1910	Julia	245
44	NY	F	1910	Jennie	178
84	NY	F	1910	Jane	84

Although both approaches are perfectly valid, we would say that **Approach 1** is not idiomatic. Meaning that people from the broader pandas community won't like reading your code. Additionally, **Approach 2** is easiest to understand, which is always important when writing code.

String .str Methods

- .str has many other useful methods.
  - .str.contains()
  - .str.lower()
  - .str.upper()
  - .str.capitalize()
  - .str.count()
  - .str.isdigit()
  - .str.replace()
- More here.

```
baby_names[baby_names['Name'].str.contains('ice')].head()
```

	State	Sex	Year	Name	Count
15	NY	F	1910	Alice	410
23	NY	F	1910	Beatrice	292
76	NY	F	1910	Bernice	92
244	NY	F	1910	Eunice	15
247	NY	F	1910	Millicent	15

```
baby_names['Name'].str.split('a').head()
```

```
0         [M, ry]
1         [Helen]
2          [Rose]
3         [Ann, ]
4    [M, rg, ret]
Name: Name, dtype: object
```

Pandas II

- String **.str** Methods
- **Grouping**

Grouping

- A **.groupby()** operation involves some combination of splitting the object, applying a function, and combining the results.
- This can be used to group large amounts of data and compute operations on these groups.

Grouping

Split → Apply → Combine

- **.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 a particular year.

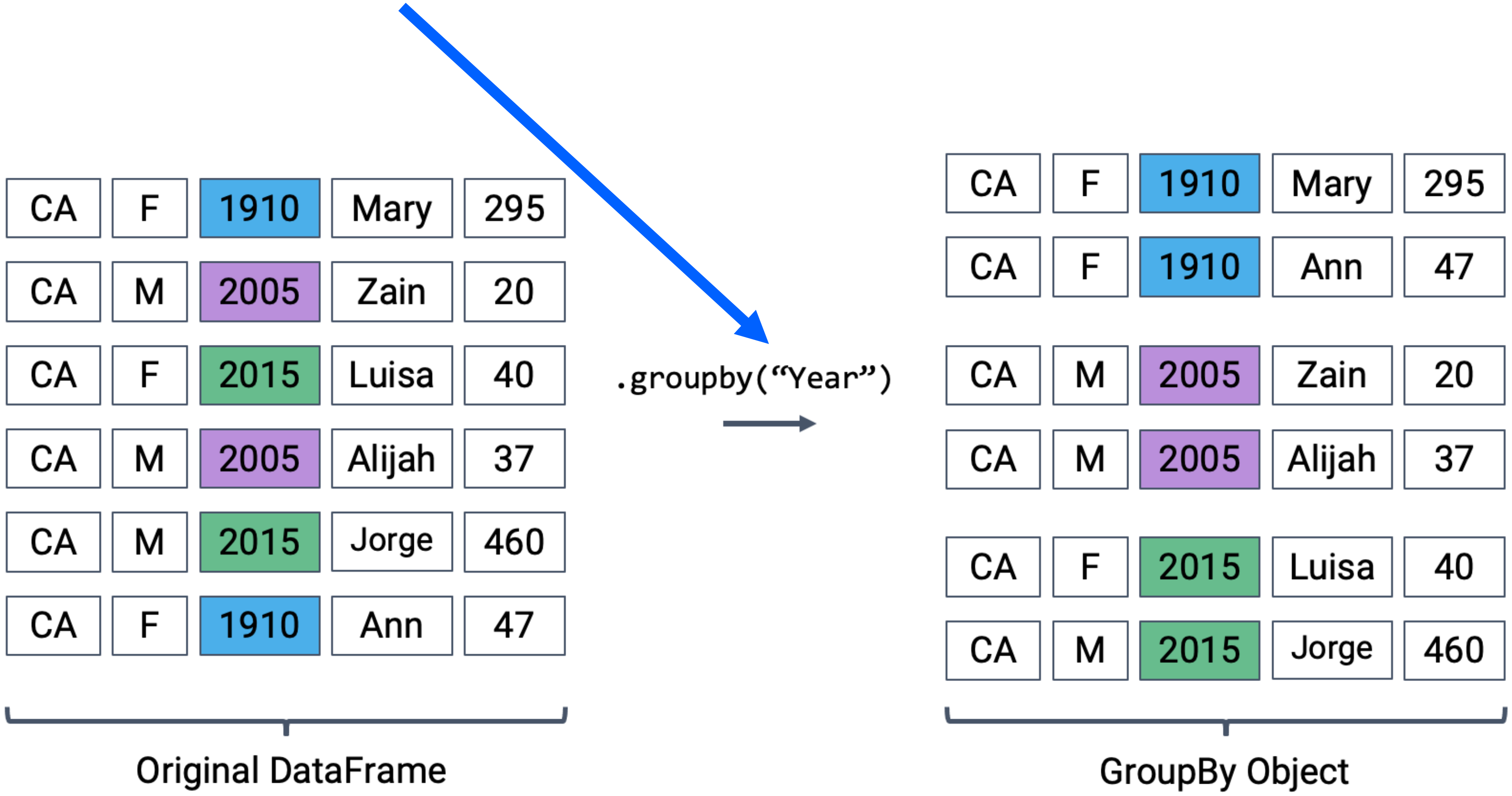


Image by Narges Norouzi, Lisa Yan, Josh Hug

Grouping

Split → Apply → Combine

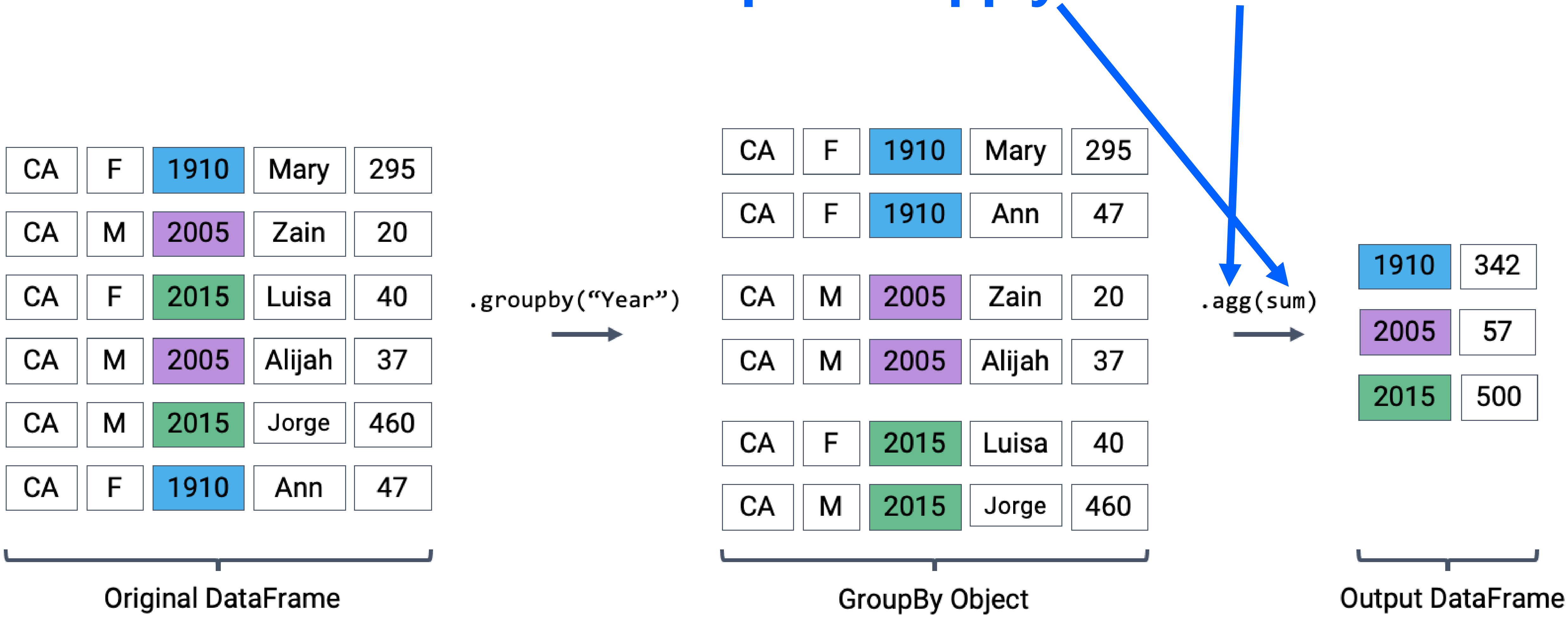


Image by Narges Norouzi, Lisa Yan, Josh Hug

Grouping

Split → Apply → Combine

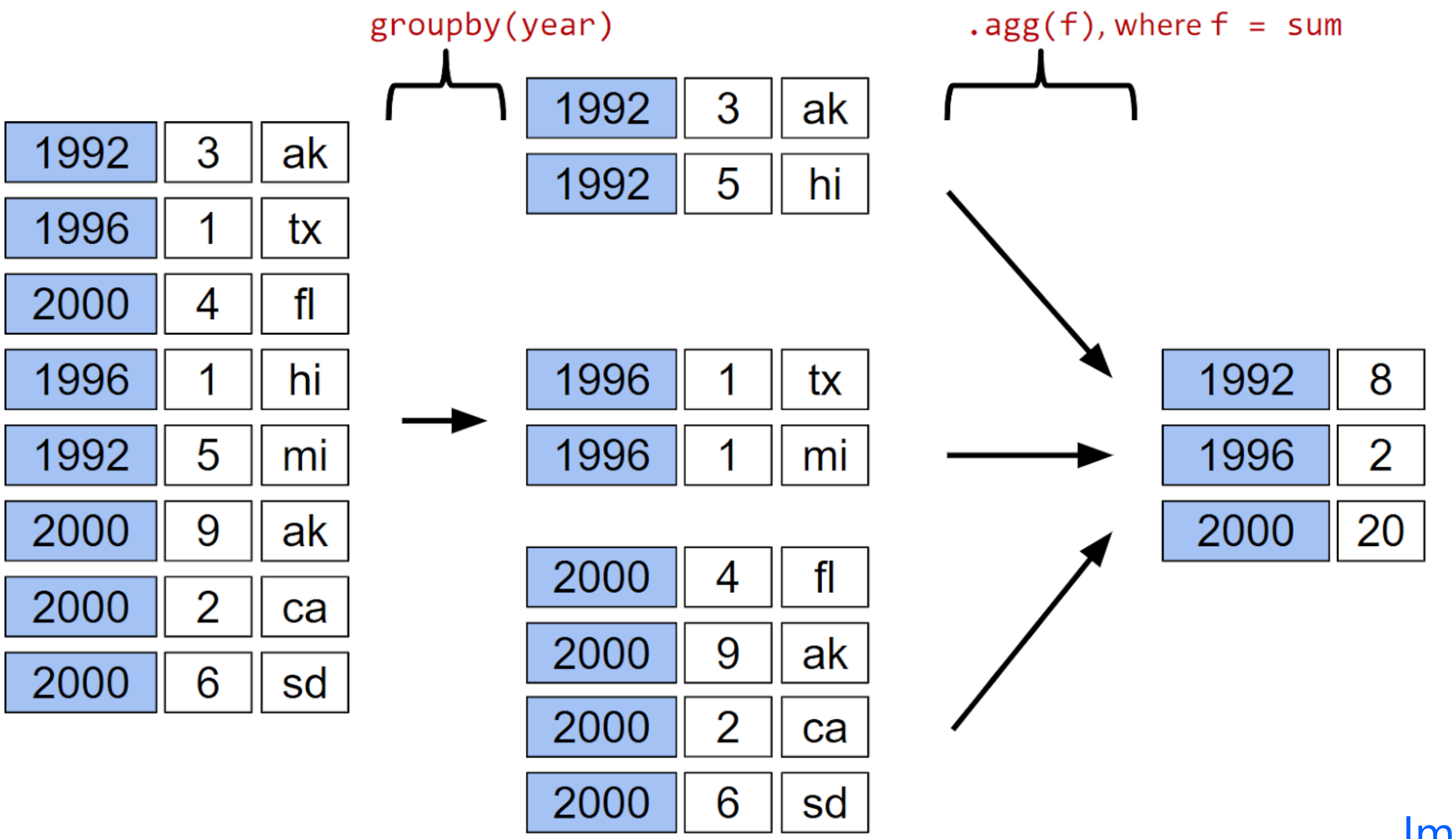
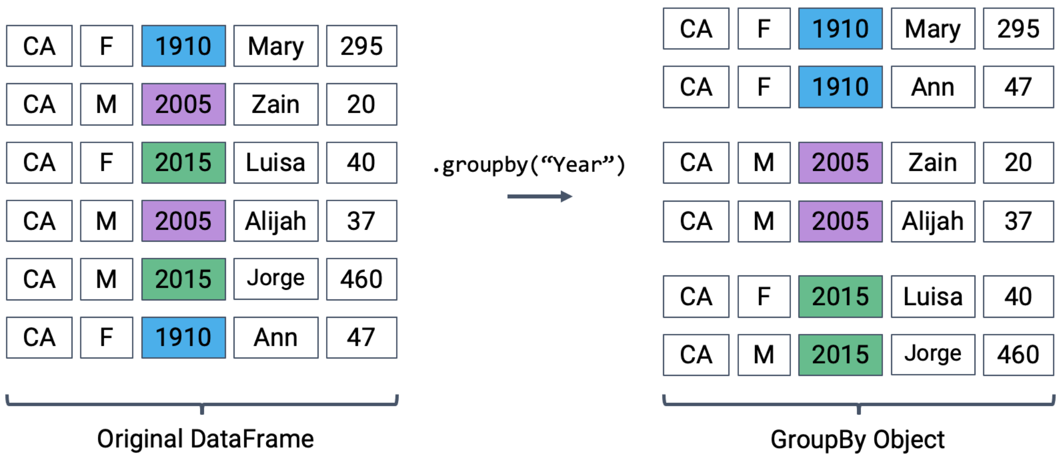


Image by Josh Hug

Grouping

- **.groupby()**
- **Split**
- A DataFrame is split (grouped) into smaller DataFrames according to a column name or multiple column names.




```python
for name, group in baby_names.groupby('Name'):

    # Print the name and the group DataFrame
    print(name)
    print(group)
    print('')
```

DataFrame

```
Aaban
        State Sex  Year   Name  Count
285414     NY   M  2013  Aaban      6
287497     NY   M  2014  Aaban      6

Aaden
        State Sex  Year   Name  Count
272587     NY   M  2007  Aaden      8
273666     NY   M  2008  Aaden     63
275781     NY   M  2009  Aaden     59
278272     NY   M  2010  Aaden     20
280505     NY   M  2011  Aaden     16
282671     NY   M  2012  Aaden     14
285042     NY   M  2013  Aaden      9
286814     NY   M  2014  Aaden     14
288771     NY   M  2015  Aaden     18
291086     NY   M  2016  Aaden     12
293071     NY   M  2017  Aaden     13
295596     NY   M  2018  Aaden      7

Aadhya
        State Sex  Year    Name  Count
160742     NY   F  2015  Aadhya      8
163399     NY   F  2016  Aadhya      7
166090     NY   F  2017  Aadhya      6
168374     NY   F  2018  Aadhya      7
```

Grouping

- **.groupby()**
- **Apply**
- We can apply a number of functions, both built-in and custom, to these smaller grouped DataFrames.
  - **Aggregation**
  - Transformation
  - Filtering
  - Applying our own function

```
baby_names.head()
```

	State	Sex	Year	Name	Count
0	NY	F	1910	Mary	1923
1	NY	F	1910	Helen	1290
2	NY	F	1910	Rose	990
3	NY	F	1910	Anna	951
4	NY	F	1910	Margaret	926

```
baby_names.groupby('Name').sum().head()
```

	Year	Count
Name		
Aaban	4027	12
Aaden	24150	253
Aadhya	8066	28
Aadi	10058	31
Aadil	2016	5

Civil & Mineral Engineering
UNIVERSITY OF TORONTO

Grouping

- **.groupby()**
- **Combine**
- Lastly, we combine the output into a new DataFrame where the index is set to the **.groupby()** key ('Name').

```
baby_names.head()
```

	State	Sex	Year	Name	Count
0	NY	F	1910	Mary	1923
1	NY	F	1910	Helen	1290
2	NY	F	1910	Rose	990
3	NY	F	1910	Anna	951
4	NY	F	1910	Margaret	926

```
baby_names.groupby('Name').sum().head()
```

	Year	Count
Name		
Aaban	4027	12
Aaden	24150	253
Aadhya	8066	28
Aadi	10058	31
Aadil	2016	5

Grouping

- **.groupby()**
- You cannot take the sum of a Series of strings.
- This can be confusing and lead to problems if you don't understand what's happening under the hood.

```
baby_names.head()
```

	State	Sex	Year	Name	Count
0	NY	F	1910	Mary	1923
1	NY	F	1910	Helen	1290
2	NY	F	1910	Rose	990
3	NY	F	1910	Anna	951
4	NY	F	1910	Margaret	926

```
baby_names.groupby('Name').sum().head()
```

Name	Year	Count
Aaban	4027	12
Aaden	24150	253
Aadhya	8066	28
Aadi	10058	31
Aadil	2016	5

Where did these columns go?

Grouping

- .groupby()
- Let's import the elections dataset.

```
elections = pd.read_csv('elections.csv')
elections.head()
```

	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

Grouping

- .groupby()
- Let's apply max function.

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

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	2016	Woodrow Wilson	69498516	win	61.344703
Democratic-Republican	1824	John Quincy Adams	151271	win	57.210122

Grouping

- **.groupby()**
- Let's apply max function.
- We have to be careful when using aggregation functions.
- For example, the results might be misinterpreted to say that Woodrow Wilson ran for election in 2016. Why is this happening?
- Every column is calculated independently! Among Democrats:
  - Last year they ran: 2016
  - Alphabetically latest candidate name: Woodrow Wilson
  - Highest number of votes: 69498516
  - Alphabetically latest Result ['loss', 'win']: win
  - Highest % of vote: 61.34

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

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	2016	Woodrow Wilson	69498516	win	61.344703
Democratic-Republican	1824	John Quincy Adams	151271	win	57.210122

Grouping

- **.groupby()**
- Here, we are using the aggregation .agg() method to apply the .max() function.
- Note that .agg(max) and .max() result in the same output.
- The aggregation .agg() method can apply built-in function (min, max, mean, etc.) and custom functions as well.

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

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	2016	Woodrow Wilson	69498516	win	61.344703
Democratic-Republican	1824	John Quincy Adams	151271	win	57.210122

Grouping

- Let's switch back to the baby names dataset and apply a custom function.
- Let's say we want to find out which name has seen the greatest change in popularity.
- To do this, we'll use the absolute max/min difference.
  - AMMD = max(count) - min(count)

	State	Sex	Year	Name	Count
942	NY	F	1912	Irene	411
136473	NY	F	2006	Kelis	31
55082	NY	F	1967	Grace	161
258532	NY	M	2000	Markus	17
173616	NY	M	1918	Dewitt	14

```
def ammd(series):
    return max(series) - min(series)
```

Grouping

- Let's switch back to the baby names dataset and apply a custom function.
- Let's say we want to find out which name has seen the greatest change in popularity.
- To do this, we'll use the absolute max/min difference.
    - AMMD = max(count) - min(count)

```
jennifer_counts = baby_names[baby_names['Name']=='Jennifer']
jennifer_counts.head()
```

	State	Sex	Year	Name	Count
16256	NY	F	1932	Jennifer	6
17091	NY	F	1933	Jennifer	5
17813	NY	F	1934	Jennifer	5
19291	NY	F	1936	Jennifer	5
19873	NY	F	1937	Jennifer	9

Let's calculate the AMMD for Jennifer.

```
def ammd(series):
    return max(series) - min(series)
```

```
ammd(jennifer_counts['Count'])
```

```
5519
```

Grouping

- So, we will want to do the following:
  1. Group the data by name:
     - DataFrame for Jennifer
     - DataFrame for Matt
     - DataFrame for Karl
     - etc.
  2. Calculate the **ammd** for each name.
  3. Create a new DataFrame including names and **ammd**.
- We can do this using **.groupby()** and **.agg()**.

```
baby_names.
```

	Count
Name	
Aaban	0
Aaden	56
Aadhya	2
Aadi	2
Aadil	0

Grouping

- So, we will want to do the following:
  1. Group the data by name:
     - DataFrame for Jennifer
     - DataFrame for Matt
     - DataFrame for Karl
     - etc.
  2. Calculate the **ammd** for each name.
  3. Create a new DataFrame including names and **ammd**.
- We can do this using **.groupby()** and **.agg()**.

```
baby_names.groupby('Name')
```

Name	Count
Aaban	0
Aaden	56
Aadhya	2
Aadi	2
Aadil	0

Grouping

- So, we will want to do the following:
  1. Group the data by name:
     - DataFrame for Jennifer
     - DataFrame for Matt
     - DataFrame for Karl
     - etc.
  2. Calculate the **ammd** for each name.
  3. Create a new DataFrame including names and **ammd**.
- We can do this using **.groupby()** and **.agg()**.

```
baby_names.groupby('Name')[['Count']]
```

	Count
Name	
Aaban	0
Aaden	56
Aadhya	2
Aadi	2
Aadil	0

Grouping

- So, we will want to do the following:
  1. Group the data by name:
     - DataFrame for Jennifer
     - DataFrame for Matt
     - DataFrame for Karl
     - etc.
  2. Calculate the **ammd** for each name.
  3. Create a new DataFrame including names and **ammd**.
- We can do this using **.groupby()** and **.agg()**.

```
baby_names.groupby('Name')[['Count']].agg(     ).head()
```

	Count
Name	
Aaban	0
Aaden	56
Aadhya	2
Aadi	2
Aadil	0

Grouping

- So, we will want to do the following:
  1. Group the data by name:
     - DataFrame for Jennifer
     - DataFrame for Matt
     - DataFrame for Karl
     - etc.
  2. Calculate the **ammd** for each name.
  3. Create a new DataFrame including names and **ammd**.
- We can do this using **.groupby()** and **.agg()**.

```
baby_names.groupby('Name')[['Count']].agg(ammd).head()
```

Name	Count
Aaban	0
Aaden	56
Aadhya	2
Aadi	2
Aadil	0

Grouping

- So, we will want to do the following:
  1. Group the data by name:
     - DataFrame for Jennifer
     - DataFrame for Matt
     - DataFrame for Karl
     - etc.
  2. Calculate the **ammd** for each name.
  3. Create a new DataFrame including names and **ammd**.
- We can do this using **.groupby()** and **.agg()**.

```
baby_names.groupby('Name')[['Count']].agg(ammd).head()
```

Name	Count
Aaban	0
Aaden	56
Aadhya	2
Aadi	2
Aadil	0

```
baby_names.groupby('Name')[['Count']]
          .agg(ammd)
          .rename(columns={'Count': 'ammd'}).head()
```

Name	ammd
Aaban	0
Aaden	56
Aadhya	2
Aadi	2
Aadil	0

Grouping

- **.groupby()**
- **Apply**
- We can apply a number of functions, both built-in and custom, to these smaller grouped DataFrames.
    - Aggregation
    - Transformation
    - **Filtering**
    - Applying our own function

Grouping

Only include months with more than 10 million in revenue.

- **.filter()**
- Filter gives a copy of the original DataFrame where row r is included if its group obeys the given condition.
- Note: Filtering is done per GROUP, not per ROW.

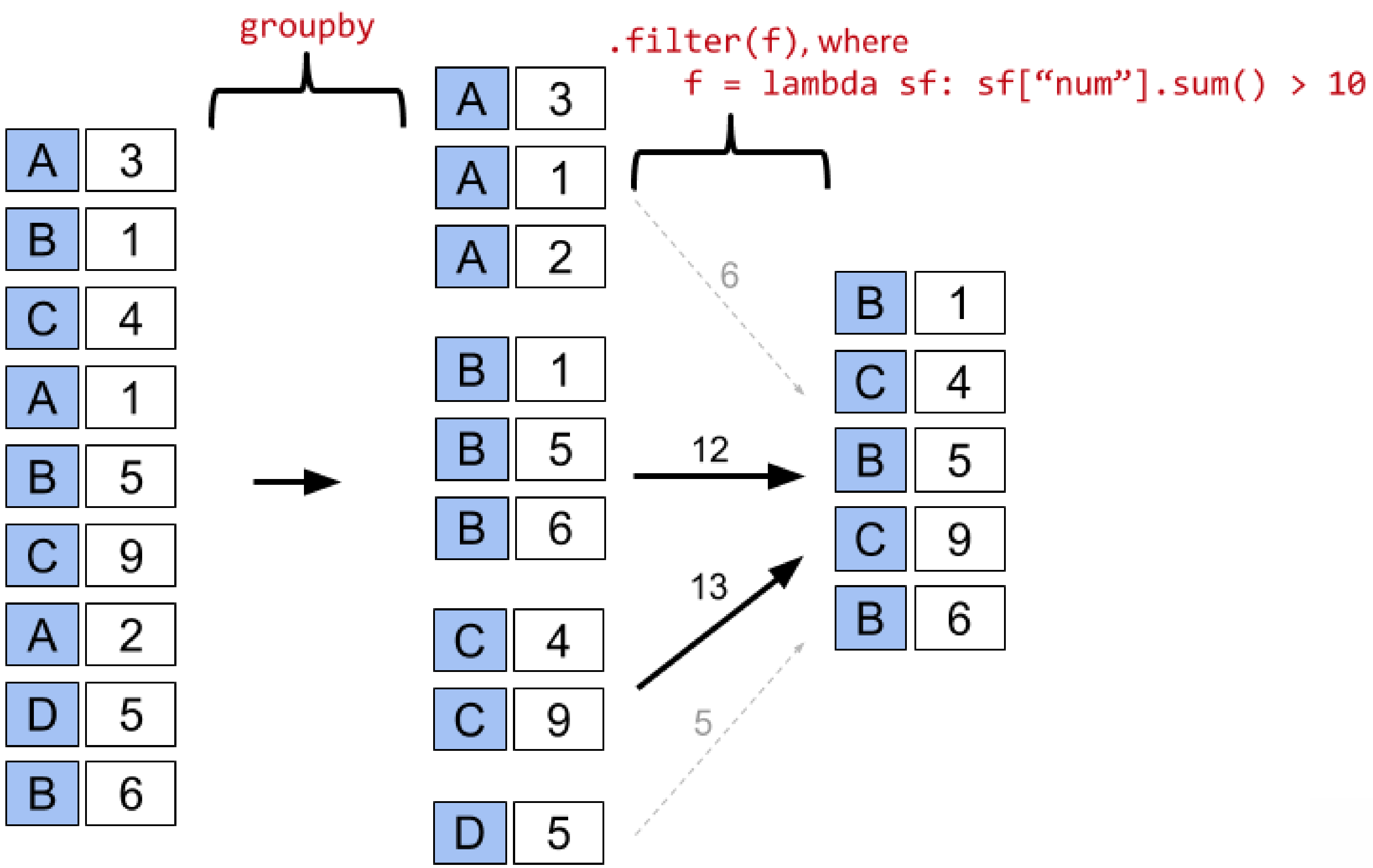


Image by Josh Hug

Civil & Mineral Engineering
UNIVERSITY OF TORONTO

Grouping

- **.filter()**
- Filter gives a copy of the original DataFrame where row r is included if its group obeys the given condition.
- Note: Filtering is done per GROUP, not per ROW.

```
elections.groupby('Year').filter(lambda df: df['%'].max() < 45)
```

	Year	Candidate	Party	Popular vote	Result	%
23	1860	Abraham Lincoln	Republican	1855993	win	39.699408
24	1860	John Bell	Constitutional Union	590901	loss	12.639283
25	1860	John C. Breckinridge	Southern Democratic	848019	loss	18.138998
26	1860	Stephen A. Douglas	Northern Democratic	1380202	loss	29.522311
66	1912	Eugene V. Debs	Socialist	901551	loss	6.004354
67	1912	Eugene W. Chafin	Prohibition	208156	loss	1.386325
68	1912	Theodore Roosevelt	Progressive	4122721	loss	27.457433
69	1912	William Taft	Republican	3486242	loss	23.218466
70	1912	Woodrow Wilson	Democratic	6296284	win	41.933422
115	1968	George Wallace	American Independent	9901118	loss	13.571218
116	1968	Hubert Humphrey	Democratic	31271839	loss	42.863537
117	1968	Richard Nixon	Republican	31783783	win	43.565246
139	1992	Andre Marrou	Libertarian	290087	loss	0.278516
140	1992	Bill Clinton	Democratic	44909806	win	43.118485
141	1992	Bo Gritz	Populist	106152	loss	0.101918
142	1992	George H. W. Bush	Republican	39104550	loss	37.544784
143	1992	Ross Perot	Independent	19743821	loss	18.956298

Practice!

- Launch the Lecture 2.2 notebook from Quercus and review the material from this lecture in more detail.
- Link

CMEDS

Civil & Mineral Engineering
UNIVERSITY OF TORONTO

CME538 Introduction to Data Science

Week 2 | Lecture 2 (2.2)

Pandas II.