The Python Data Analysis Library

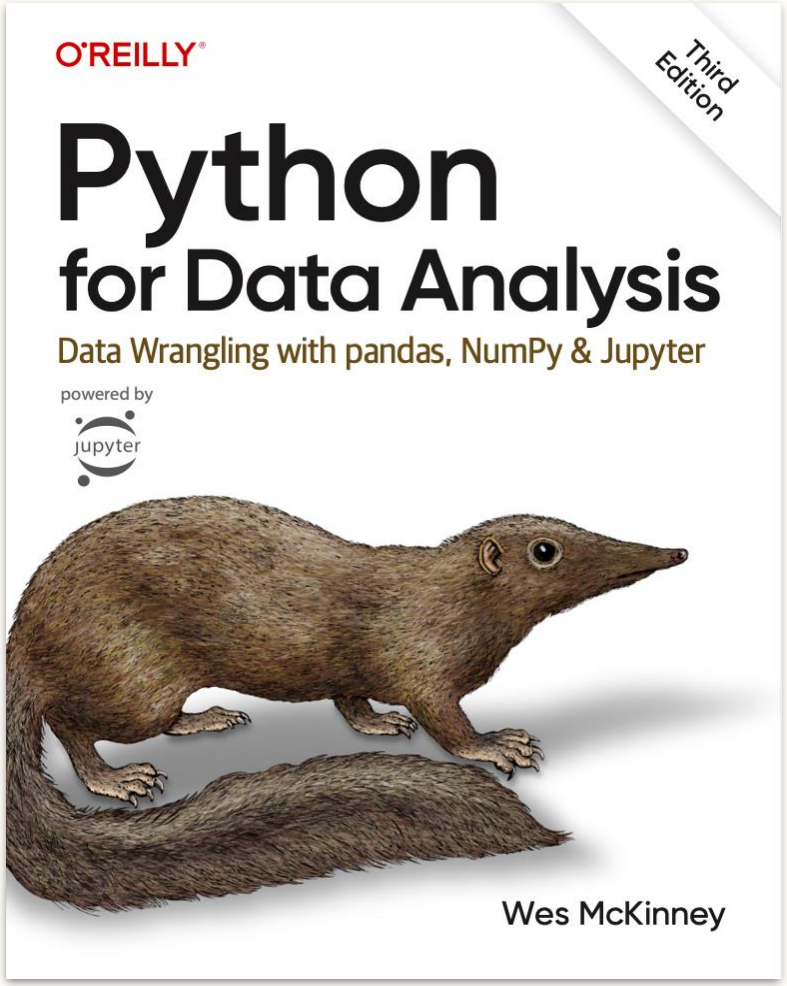
THE FOUNDATIONS OF DATA SCIENCE

WEEK 1: Pandas II

UTILITY FUNCTIONS, GROUPING, AGGREGATION

Usman Nazir

YOU HAVE FREE ACCESS TO A FANTASTIC BOOK BY THE CREATOR OF PANDAS!



The "Open Edition" is freely available at https://wesmckinney.com/book

SELECTION OPERATORS

- **loc** selects items by **label**. First argument is rows, second argument is columns.
- **iloc** selects items by **number**. First argument is rows, second argument is columns.
- [ ] only takes one argument, which may be:
  - A slice of **row numbers**.
  - A list of column labels.
  - A single column label.

MORE ON CONDITIONAL SELECTION

- **Conditional Selection**
- Handy Utility Functions
- Custom Sorts
- Adding, Modifying, and Removing Columns
- Groupby.agg
- Some groupby.agg Puzzles

BOOLEAN ARRAY INPUT PRINT EVEN ROWS

```
babynames_first_10_rows[[True, False, True, False, True, False, True, False, True, False]]
```

```
babynames_first_10_rows = babynames.loc[:9, :]
```

	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
5	CA	F	1910	Ruth	128
6	CA	F	1910	Evelyn	126
7	CA	F	1910	Alice	118
8	CA	F	1910	Virginia	101
9	CA	F	1910	Elizabeth	93

	State	Sex	Year	Name	Count
0	CA	F	1910	Mary	295
2	CA	F	1910	Dorothy	220
4	CA	F	1910	Frances	134
6	CA	F	1910	Evelyn	126
8	CA	F	1910	Virginia	101

```
babynames_first_10_rows.iloc[0:9:2]
```

BOOLEAN ARRAY INPUT

- We can perform the same operation using `loc`.

```
babynames_first_10_rows = babynames.loc[:9, :]
```

	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
5	CA	F	1910	Ruth	128
6	CA	F	1910	Evelyn	126
7	CA	F	1910	Alice	118
8	CA	F	1910	Virginia	101
9	CA	F	1910	Elizabeth	93

```
babynames_first_10_rows.loc[[True, False, True,
False, True, False, True, False, True, False], :]
```

	State	Sex	Year	Name	Count
0	CA	F	1910	Mary	295
2	CA	F	1910	Dorothy	220
4	CA	F	1910	Frances	134
6	CA	F	1910	Evelyn	126
8	CA	F	1910	Virginia	101

BOOLEAN ARRAY INPUT

- Useful because boolean arrays can be generated by using logical operators on Series.

Length 400761 Series where every entry is either “True” or “False”, where “True” occurs for every babyname with "Sex" = "F".

```
logical_operator = (babynames["Sex"] == "F")
0          True
1          True
2          True
3          True
4          True
          ...
400757    False
400758    False
400759    False
400760    False
400761    False
Name: Sex, Length: 400762, dtype: bool
```

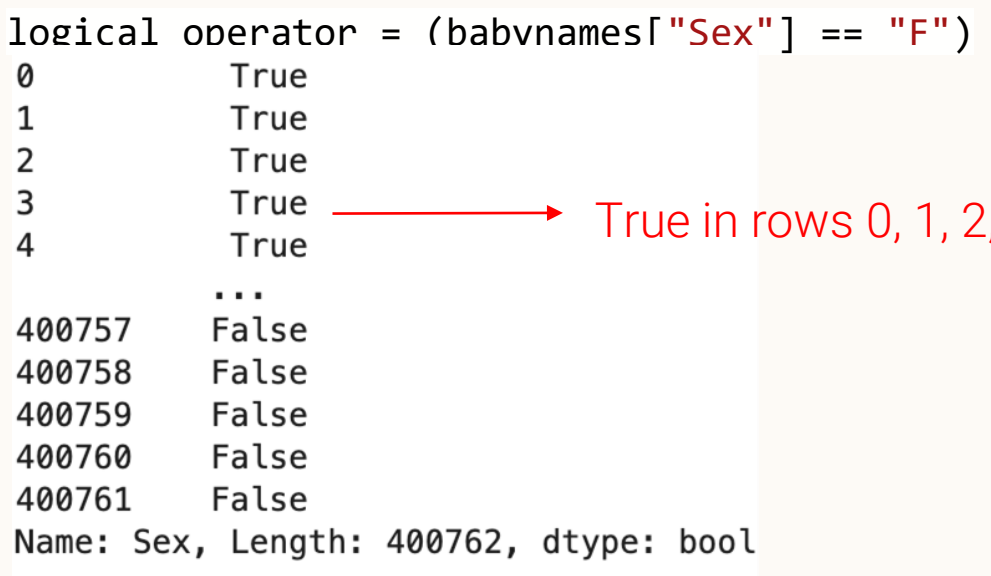
True in rows 0, 1, 2, …

BOOLEAN ARRAY INPUT

- Useful because boolean arrays can be generated by using logical operators on Series.

Length 235791 Series where every entry belongs to a babyname with "Sex" = "F"

Length 400761 Series where every entry is either “True” or “False”, where “True” occurs for every babyname with "Sex" = "F".



True in rows 0, 1, 2, …

BOOLEAN ARRAY INPUT

- Can also use **.loc**.

```
babynames.loc[babynames["Sex"] == "F"]
```

```
0          True
1          True
2          True
3          True
4          True
          ...
400757    False
400758    False
400759    False
400760    False
400761    False
Name: Sex, Length: 400762, dtype: bool
```

	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
...	...	...	...	...	...
235786	CA	F	2021	Zarahi	5
235787	CA	F	2021	Zelia	5
235788	CA	F	2021	Zenobia	5
235789	CA	F	2021	Zeppelin	5
235790	CA	F	2021	Zoraya	5

235791 rows × 5 columns

BOOLEAN ARRAY INPUT

Boolean `Series` can be combined using various operators, allowing filtering of results by multiple criteria.

- Example: The & operator.
- Lab covers more such operators.

```
babynames[(babynames["Sex"] == "F") & (babynames["Year"] < 2000)]
```

	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
...	...	...	...	...	...
149044	CA	F	1999	Zareen	5
149045	CA	F	1999	Zeinab	5
149046	CA	F	1999	Zhane	5
149047	CA	F	1999	Zoha	5
149048	CA	F	1999	Zoila	5

149049 rows × 5 columns

QUESTION

- Which of the following pandas statements returns a `DataFrame` of the first 3 baby names with `Count > 250`.

	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
5	CA	F	1910	Ruth	128
6	CA	F	1910	Evelyn	126

	Name	Count
0	Mary	295
233	Mary	390
484	Mary	534

ANSWER

- Which of the following pandas statements returns a `DataFrame` of the first 3 baby names with `Count > 250`.

i. **`babynames.iloc[[0, 233, 484], [3, 4]]`**

ii. ~~`babynames.loc[[0, 233, 484]]`~~

`iii.babynames.loc[babynames["Count"] > 250, ["Name", "Count"]].head(3)`

~~`iv.babynames.loc[babynames["Count"] > 250, ["Name", "Count"]].iloc[0:2, :]`~~

	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
5	CA	F	1910	Ruth	128
6	CA	F	1910	Evelyn	126

	Name	Count
0	Mary	295
233	Mary	390
484	Mary	534

EXAM PROBLEMS

```
babynames.loc[babynames["Count"] > 250, ["Name", "Count"]].iloc[0:2, :]
```

	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
5	CA	F	1910	Ruth	128
6	CA	F	1910	Evelyn	126

	Name	Count
0	Mary	295
233	Mary	390

ALTERNATIVES TO BOOLEAN ARRAY SELECTION

Boolean array selection is a useful tool, but can lead to overly verbose code for complex conditions.

```
babynames[(babynames["Name"] == "Bella") |
          (babynames["Name"] == "Alex") |
          (babynames["Name"] == "Ani") |
          (babynames["Name"] == "Lisa")]
```

Pandas provides **many** alternatives, for example:

- `.isin`
- `.str.startswith`
- `.groupby.filter`

	State	Sex	Year	Name	Count
6289	CA	F	1923	Bella	5
7512	CA	F	1925	Bella	8
12368	CA	F	1932	Lisa	5
14741	CA	F	1936	Lisa	8
17084	CA	F	1939	Lisa	5
...	...	...	...	...	...
386576	CA	M	2017	Alex	482
389498	CA	M	2018	Alex	494
392360	CA	M	2019	Alex	436
395230	CA	M	2020	Alex	378
398031	CA	M	2021	Alex	331

359 rows × 5 columns

ALTERNATIVES TO BOOLEAN ARRAY SELECTION

Pandas provides **many** alternatives, for example:

- `.isin`
- `.str.startswith`
- `.groupby.filter`

```
names = ["Bella", "Alex", "Ani", "Lisa"]
babynames[babynames["Name"].isin(names)]
```

	State	Sex	Year	Name	Count
6289	CA	F	1923	Bella	5
7512	CA	F	1925	Bella	8
12368	CA	F	1932	Lisa	5
14741	CA	F	1936	Lisa	8
17084	CA	F	1939	Lisa	5
...	...	...	...	...	...
386576	CA	M	2017	Alex	482
389498	CA	M	2018	Alex	494
392360	CA	M	2019	Alex	436
395230	CA	M	2020	Alex	378
398031	CA	M	2021	Alex	331

359 rows × 5 columns

ALTERNATIVES TO BOOLEAN ARRAY SELECTION

Pandas provides **many** alternatives, for example:

- `.isin`
- `.str.startswith`
- `.groupby.filter`

```
babynames[babynames["Name"].str.startswith("N")]
```

	State	Sex	Year	Name	Count
76	CA	F	1910	Norma	23
83	CA	F	1910	Nellie	20
127	CA	F	1910	Nina	11
198	CA	F	1910	Nora	6
310	CA	F	1911	Nellie	23
...	...	...	...	...	...
400648	CA	M	2021	Nirvan	5
400649	CA	M	2021	Nivin	5
400650	CA	M	2021	Nolen	5
400651	CA	M	2021	Nomar	5
400652	CA	M	2021	Nyles	5

11994 rows × 5 columns

ALTERNATIVES TO BOOLEAN ARRAY SELECTION

Pandas provides **many** alternatives, for example:

- `.isin`
- `.str.startswith`
- **`.groupby.filter`**

```
elections.groupby("Year")
         .filter(lambda sf: sf["%"].max() < 45)
         .set_index("Year")
         .sort_index()
```

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

HANDY UTILITY FUNCTIONS

- Conditional Selection
- Handy Utility Functions
- Custom Sorts
- Adding, Modifying, and Removing Columns
- Groupby.agg
- Some groupby.agg Puzzles

Numpy

Pandas `Series` and `DataFrames` support a large number of operations, including mathematical operations, so long as the data is numerical.

```
bella_count = babynames[babynames["Name"] == "Bella"]["Count"]
```

```
np.mean(bella_counts)
270.1860465116279
```

```
np.max(bella_counts)
902
```

```
6289        5
7512        8
35477       5
54487       7
58451       6
68845       6
73387       5
93601       5
96397       5
108054      7
111276      8
114677     10
117991     14
121524     17
125545     13
128946     18
132163     31
136362     15
139366     28
142917     27
146251     39
149607     65
153241     97
156955    122
160707    191
164586    213
168557    310
172646    334
176836    384
181090    439
185287    699
189455    902
193562    777
197554    761
201650    807
205629    704
209653    645
213592    643
217451    719
221207    747
224905    720
228576    566
232200    494
Name: Count, dtype: int64
```

Pandas

In addition to its rich syntax for indexing and support for other libraries (numpy, built-in functions), Pandas provides an enormous number of useful utility functions. Today, we'll discuss:

- `size/shape`
- `describe`
- `sample`
- `value_counts`
- `unique`
- `sort_values`

Shape/Size

	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
...	...	...	...	...	...
400757	CA	M	2021	Zyan	5
400758	CA	M	2021	Zyion	5
400759	CA	M	2021	Zyire	5
400760	CA	M	2021	Zylo	5
400761	CA	M	2021	Zyrus	5

400762 rows × 5 columns

```
babynames.shape
(400762, 5)
```

```
babynames.size
2003810
```

describe()

	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
...	...	...	...	...	...
400757	CA	M	2021	Zyan	5
400758	CA	M	2021	Zyion	5
400759	CA	M	2021	Zyire	5
400760	CA	M	2021	Zylo	5
400761	CA	M	2021	Zyrus	5

400762 rows × 5 columns

```
babynames.describe()
```

	Year	Count
count	400762.000000	400762.000000
mean	1985.131287	79.953781
std	26.821004	295.414618
min	1910.000000	5.000000
25%	1968.000000	7.000000
50%	1991.000000	13.000000
75%	2007.000000	38.000000
max	2021.000000	8262.000000

describe()

- A different set of statistics will be reported if `.describe()` is called on a `Series`.

```
babynames["Sex"].describe()
```

```
count     400762
unique         2
top            F
freq      235791
Name: Sex, dtype: object
```

sample()

If you want a `DataFrame` with a random selection of rows, you can use the `sample()` method.

- By default, ***it is without replacement***. Use **`replace=True`** for **replacement**.
- Naturally, can be chained with other methods and operators (`iloc`, etc).

```
babynames.sample()
```

	State	Sex	Year	Name	Count
108418	CA	F	1988	Janielle	6

```
babynames.sample(5).iloc[:, 2:]
```

	Year	Name	Count
169346	2005	Greta	36
231690	2020	Yui	6
203404	2013	Libby	14
385359	2016	Cael	9
386609	2017	Everett	353

```
babynames[babynames["Year"] == 2000]
         .sample(4, replace=True)
         .iloc[:, 2:]
```

	Year	Name	Count
340297	2000	Emmet	8
339662	2000	Fred	17
150463	2000	Rosalia	18
152732	2000	Shanae	5

value_counts()

The `Series.value_counts` method counts the number of occurrences of a each unique value in a `Series`.

- Return value is also a `Series`.

```
babyname["Name"].value_counts()
```

```
Jean         221
Francis      219
Guadalupe    216
Jessie       215
Marion       213
             ...
Janin          1
Jilliann       1
Jomayra        1
Karess         1
Zyrus          1
Name: Name, Length: 20239, dtype: int64
```

unique()

- The `Series.unique` method returns an array of every unique value in a `Series`.

```
babynames["Name"].unique()
```

```
array(['Mary', 'Helen', 'Dorothy', ..., 'Zyire', 'Zylo', 'Zyrus'],
      dtype=object)
```

sort_values()

- The `DataFrame.sort_values` and `Series.sort_values` methods sort a `DataFrame` (or `Series`).
  - The `DataFrame` version requires an argument specifying the column on which to sort.

```
babynames["Name"].sort_values()
```

```
380256      Aadan
362255      Aadan
365374      Aadan
394460    Aadarsh
366561      Aaden
           ...
232144      Zyrah
217415      Zyrah
197519      Zyrah
220674      Zyrah
400761      Zyrus
Name: Name, Length: 400762, dtype: object
```

```
babynames.sort_values(by = "Count", ascending=False)
```

	State	Sex	Year	Name	Count
263272	CA	M	1956	Michael	8262
264297	CA	M	1957	Michael	8250
313644	CA	M	1990	Michael	8247
278109	CA	M	1969	Michael	8244
279405	CA	M	1970	Michael	8197
...	...	...	...	...	...
159967	CA	F	2002	Arista	5
159966	CA	F	2002	Arisbeth	5
159965	CA	F	2002	Arisa	5
159964	CA	F	2002	Arionna	5
400761	CA	M	2021	Zyrus	5

400762 rows × 5 columns

CUSTOM SORTS

- Conditional Selection
- Handy Utility Functions
- Custom Sorts
- Adding, Modifying, and Removing Columns
- Groupby.agg
- Some groupby.agg Puzzles

MANIPULATING STRING DATA

- How we could find, for example, the top 5 most popular names in California in the year 2021?

```
babynames[babynames["Year"] == 2021]
         .sort_values("Count", ascending=False)
```

	State	Sex	Year	Name	Count
397909	CA	M	2021	Noah	2591
397910	CA	M	2021	Liam	2469
232145	CA	F	2021	Olivia	2395
232146	CA	F	2021	Emma	2171
397911	CA	M	2021	Mateo	2108

MANIPULATING STRING DATA

What if we wanted to find the longest names in California?

- Just sorting by name won't work!

```
babynames.sort_values("Name", ascending=False)
```

	State	Sex	Year	Name	Count
400761	CA	M	2021	Zyrus	5
197519	CA	F	2011	Zyrah	5
232144	CA	F	2020	Zyrah	5
217415	CA	F	2016	Zyrah	5
220674	CA	F	2017	Zyrah	6
...	...	...	...	...	...
360532	CA	M	2008	Aaden	135
394460	CA	M	2019	Aadarsh	6
380256	CA	M	2014	Aadan	5
362255	CA	M	2008	Aadan	7
365374	CA	M	2009	Aadan	6

400762 rows × 5 columns

MANIPULATING STRING DATA

What if we wanted to find the longest names in California?

- Just sorting by name won't work!

```
babynames.sort_values("Name", key=lambda x: x.str.len(), ascending=False)
         .head()
```

	State	Sex	Year	Name	Count
313143	CA	M	1989	Franciscojavier	6
333732	CA	M	1997	Ryanchristopher	5
330421	CA	M	1996	Franciscojavier	8
323615	CA	M	1993	Johnchristopher	5
310235	CA	M	1988	Franciscojavier	10

ADDING, MODIFYING, AND REMOVING COLUMNS

- Conditional Selection
- Handy Utility Functions
- Custom Sorts
- Adding, Modifying, and Removing Columns
- Groupby.agg
- Some groupby.agg Puzzles

SORTING BY LENGTH

Let's try to solve the sorting problem with different approaches:

- We will create a temporary column, then sort on it.
- Approach 1: Adding a column is easy

```
# Create a Series of the length of each name
babyname_lengths = babynames["Name"].str.len()

# Add a column named "name_lengths" that includes the length of each name
babynames["name_lengths"] = babyname_lengths
```

	State	Sex	Year	Name	Count	name_lengths
0	CA	F	1910	Mary	295	4
1	CA	F	1910	Helen	239	5
2	CA	F	1910	Dorothy	220	7
3	CA	F	1910	Margaret	163	8
4	CA	F	1910	Frances	134	7

SORTING BY LENGTH

Let's try to solve the sorting problem with different approaches:

- We will create a temporary column, then sort on it.
- Approach 1: Adding a column is easy
  - Can also do both steps on one line of code

```
babynames = babynames.sort_values(by = "name_lengths", ascending=False)
```

	State	Sex	Year	Name	Count	name_lengths
0	CA	F	1910	Mary	295	4
1	CA	F	1910	Helen	239	5
2	CA	F	1910	Dorothy	220	7
3	CA	F	1910	Margaret	163	8
4	CA	F	1910	Frances	134	7

SYNTAX FOR DROPPING A COLUMN (OR ROW)

After sorting, we can drop the temporary column.

- The `drop()` method assumes you're dropping a row by default. Use `axis = "columns"` to drop a column instead.

```
babynames = babynames.drop("name_lengths", axis = "columns")
```

	State	Sex	Year	Name	Count	name_lengths
313143	CA	M	1989	Franciscojavier	6	15
340695	CA	M	2000	Franciscojavier	6	15
333732	CA	M	1997	Ryanchristopher	5	15
318049	CA	M	1991	Ryanchristopher	7	15
333556	CA	M	1997	Franciscojavier	5	15

	State	Sex	Year	Name	Count
313143	CA	M	1989	Franciscojavier	6
340695	CA	M	2000	Franciscojavier	6
333732	CA	M	1997	Ryanchristopher	5
318049	CA	M	1991	Ryanchristopher	7
333556	CA	M	1997	Franciscojavier	5

SORTING BY ARBITRARY FUNCTIONS

Suppose we want to sort by the number of occurrences of "dr" + number of occurrences of "ea".

- Use the `Series.map` method.

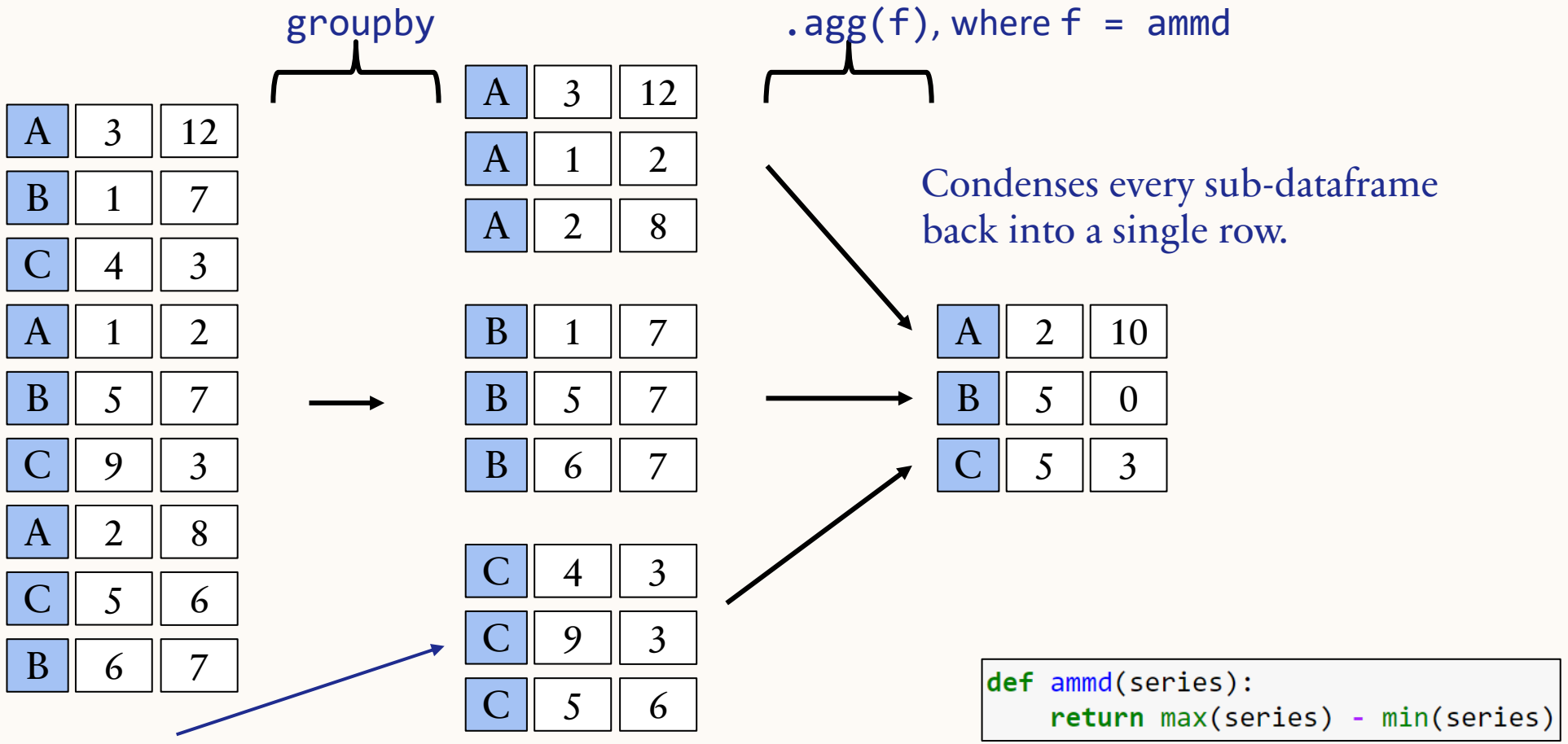
```
def dr_ea_count(string):
    return string.count('dr') + string.count('ea')


# Use `map` to apply `dr_ea_count` to each name in the "Name" column
babynames["dr_ea_count"] = babynames["Name"].map(dr_ea_count)
babynames = babynames.sort_values(by = "dr_ea_count", ascending=False)
```

	State	Sex	Year	Name	Count	dr_ea_count
304390	CA	M	1985	Deandrea	6	3
131022	CA	F	1994	Leandrea	5	3
101969	CA	F	1986	Deandrea	6	3
108723	CA	F	1988	Deandrea	5	3
115950	CA	F	1990	Deandrea	5	3

GROUPBY.AGG

- Conditional Selection
- Handy Utility Functions
- Custom Sorts
- Adding, Modifying, and Removing Columns
- Groupby.agg
- Some groupby.agg Puzzles

GROUPING AND COLLECTION

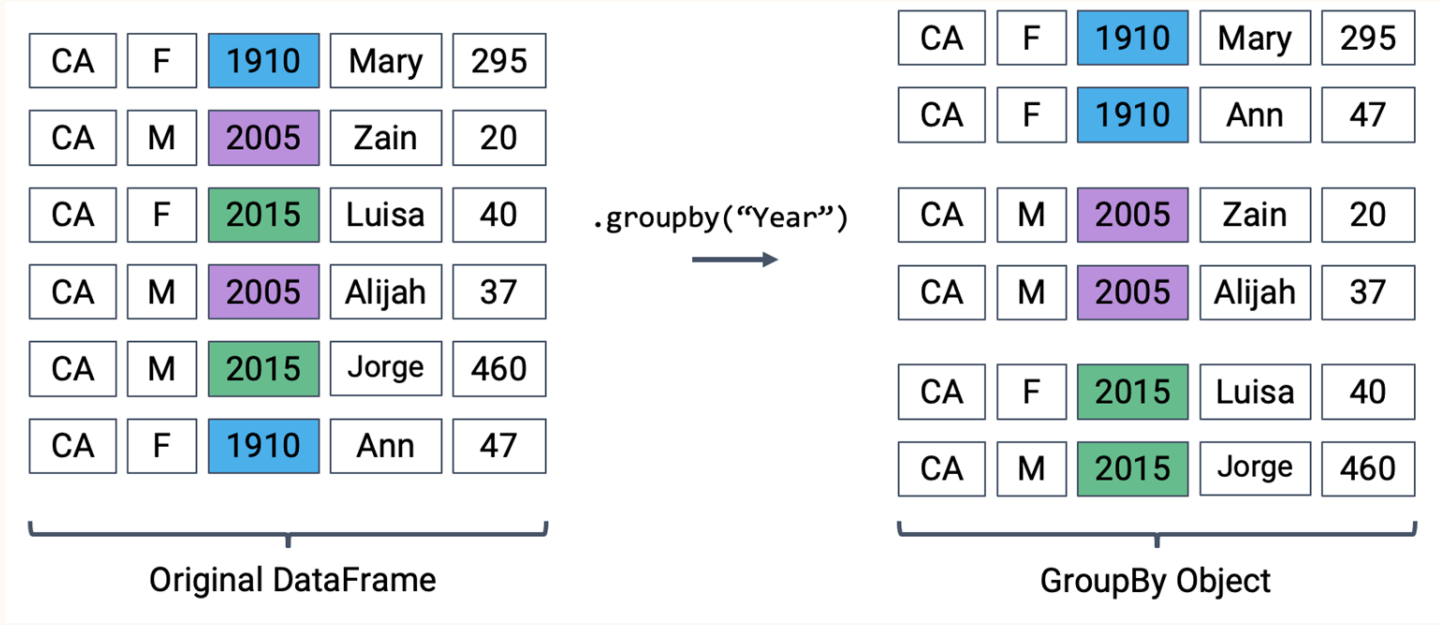



```
def ammd(series):
    return max(series) - min(series)
```

groupby()

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

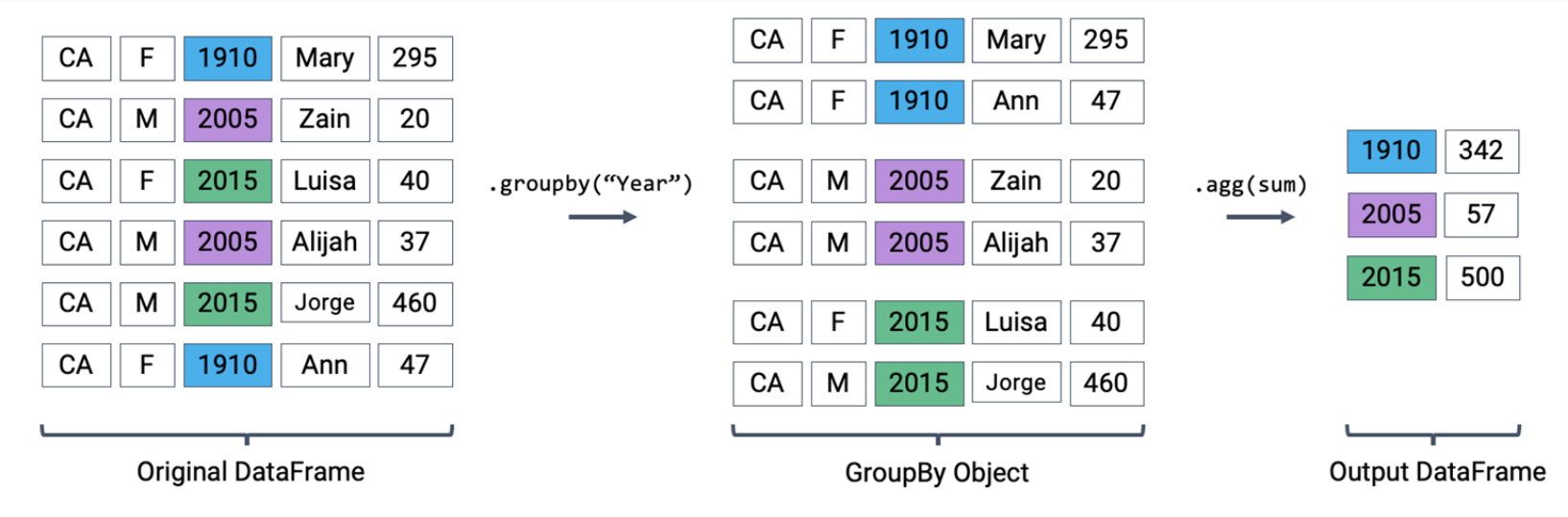
- Calling `.groupby()` generates `DataFrameGroupBy` objects → "mini" sub-DataFrames
- Each subframe contains all rows that correspond to a particular year



groupby.agg

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
- Since we can't work directly with `DataFrameGroupBy` objects, we will use aggregation methods to summarize each `DataFrameGroupBy` object into one aggregated row per subframe.



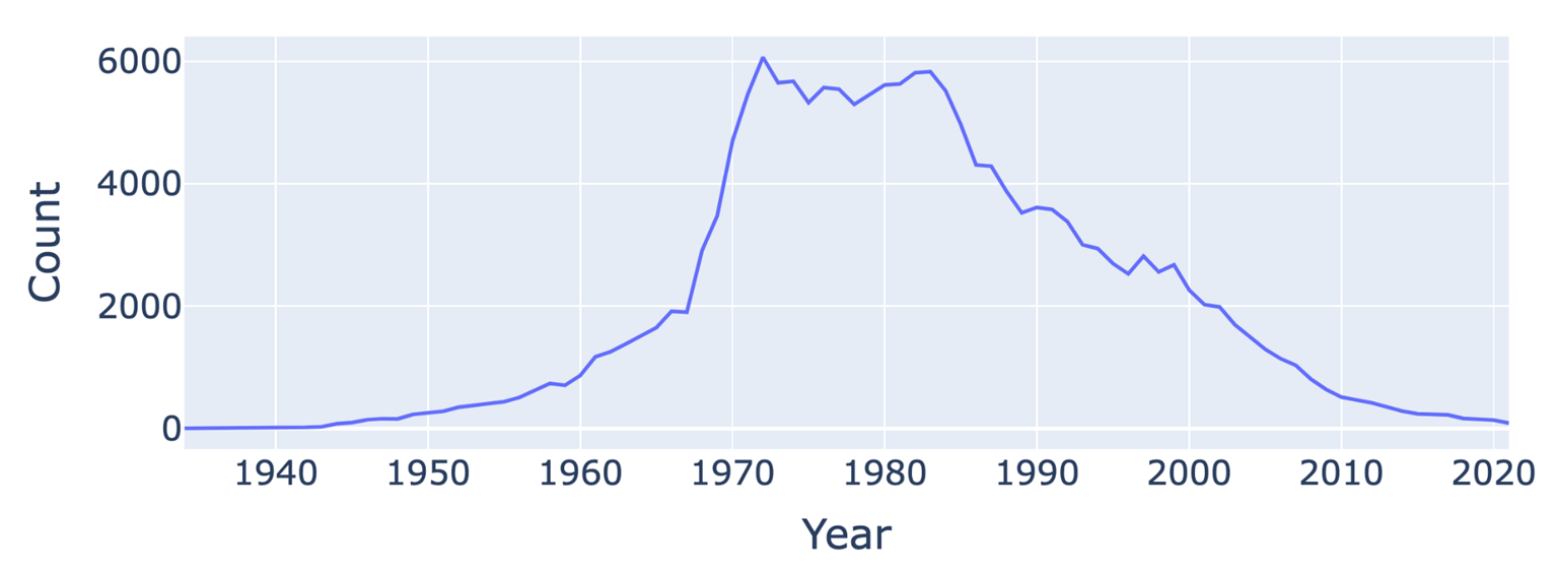
QUESTION

- Find the female baby name whose popularity has fallen the most.

ANSWER

```
female_babynames = babynames[babynames["Sex"] == "F"]
female_babynames = female_babynames.sort_values(["Year", "Count"])
jenn_counts_ser = female_babynames[female_babynames["Name"] == "Jennifer"]["Count"]
```

Number of Jennifers Born in California Per Year



ANSWER

Let's start by defining what we mean by changed popularity.

- let's define the "ratio to peak" or RTP as the ratio of babies born with a given name today to the maximum number of the name born in a single year.

Example for "Jennifer":

- In 1972, we hit peak Jennifer. 6,065 Jennifers were born.
- In 2021, there were only 91 Jennifers.
- RTP is 91 / 6065 = 0.015004.

SOME GROUPBY.AGG PUZZLES

- Conditional Selection
- Handy Utility Functions
- Custom Sorts
- Adding, Modifying, and Removing Columns
- Groupby.agg
- Some groupby.agg Puzzles

The Python Data Analysis Library

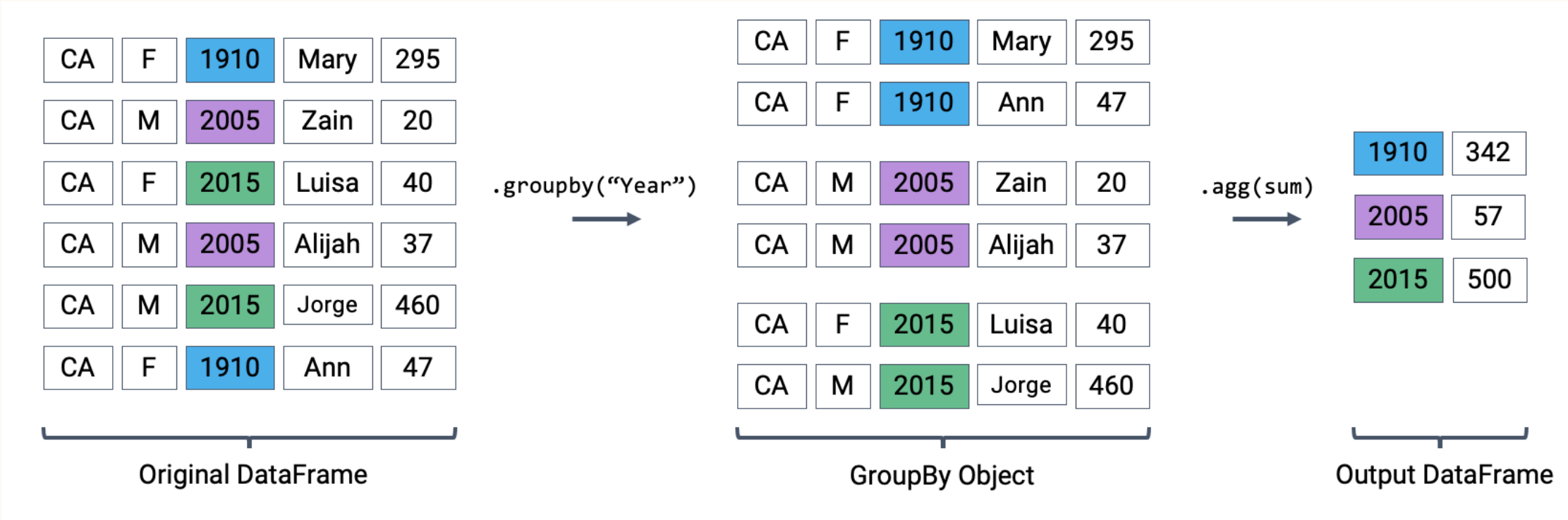
THE FOUNDATIONS OF DATA SCIENCE

WEEK 1: Pandas III

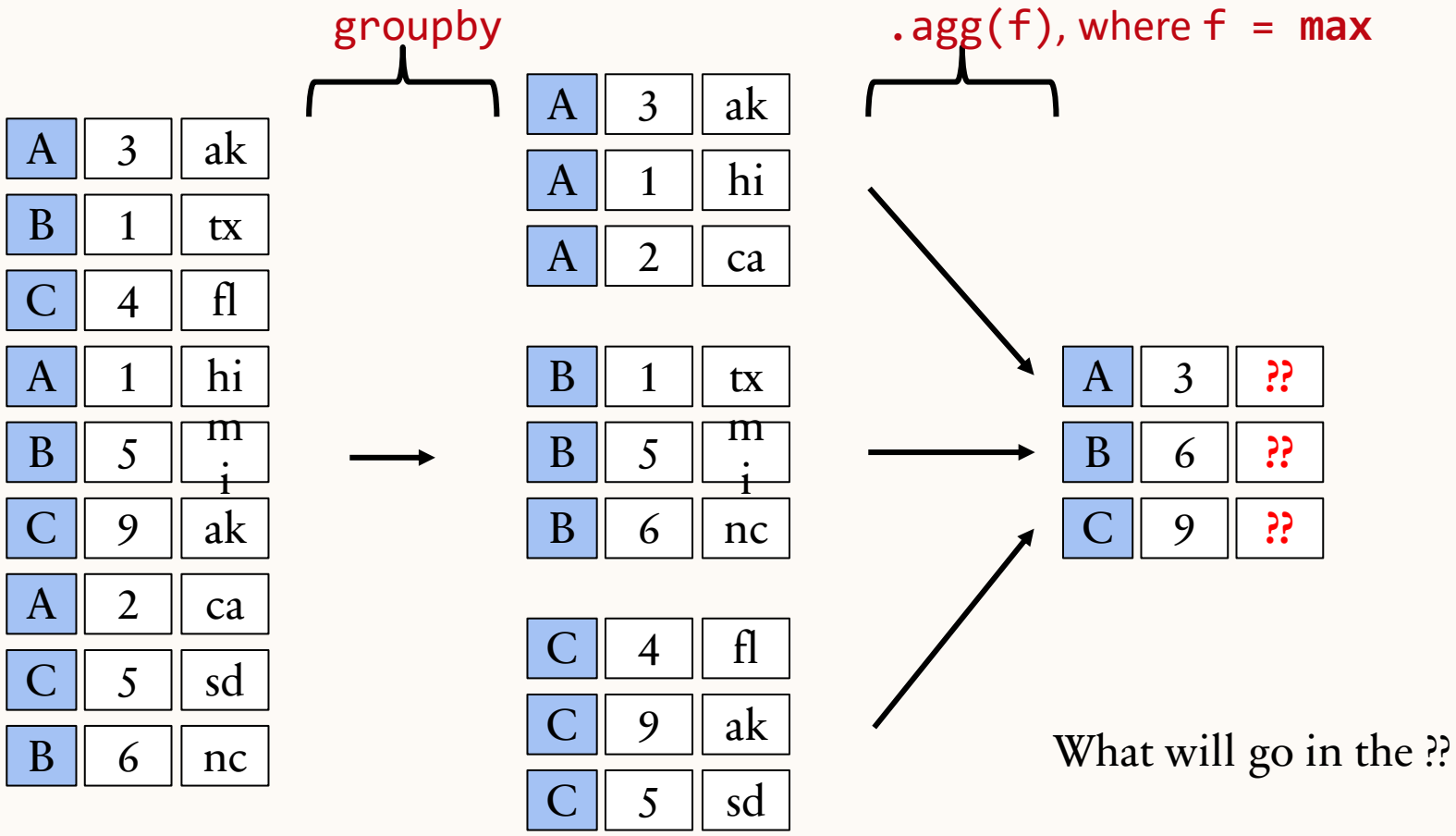
UTILITY FUNCTIONS, GROUPING, AGGREGATION

Usman Nazir

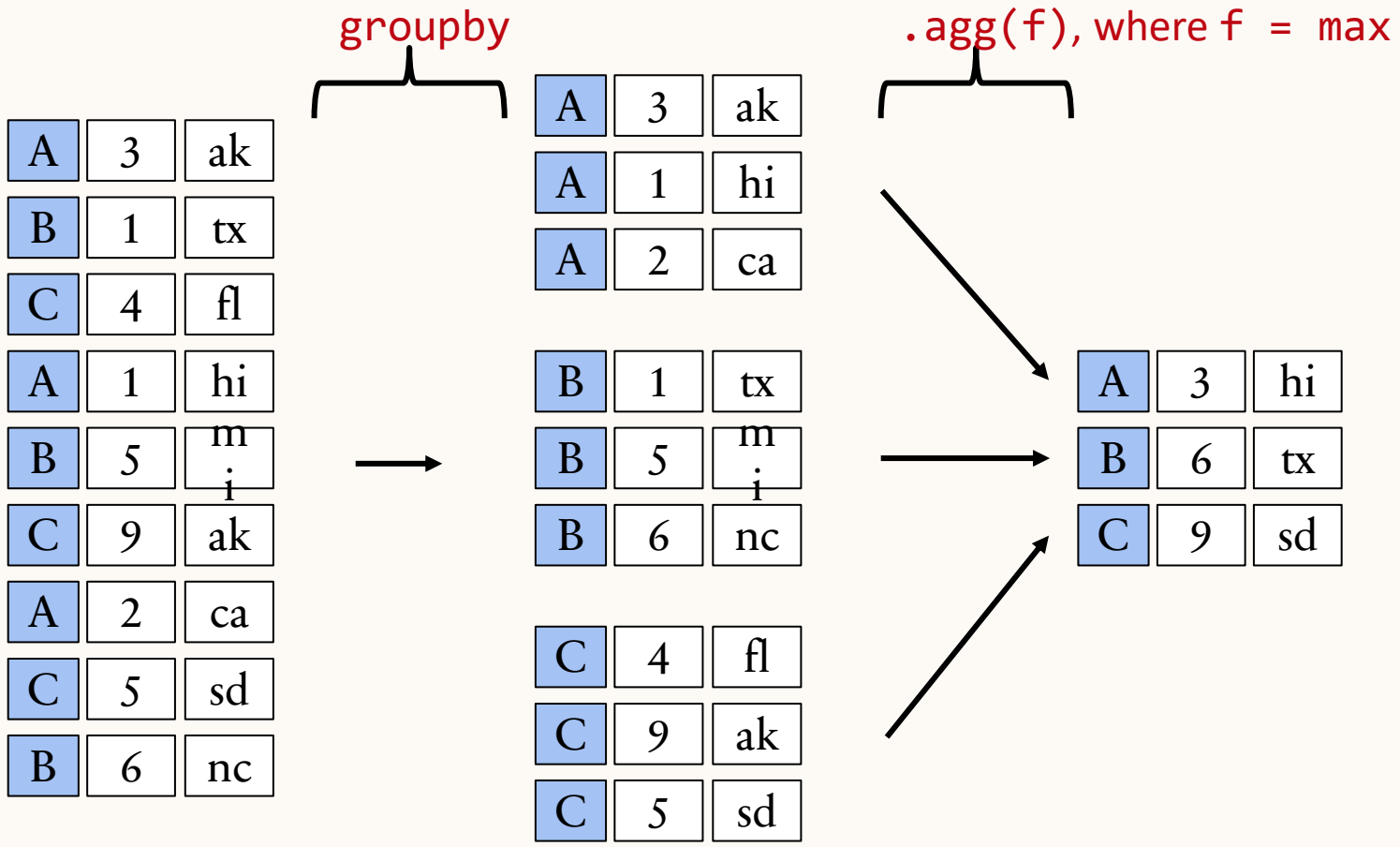
groupby


CA F 1910 Mary 295
CA M 2005 Zain 20
CA F 2015 Luisa 40
CA M 2005 Alijah 37
CA M 2015 Jorge 460
CA F 1910 Ann 47
Original DataFrame
.groupby("Year")
CA F 1910 Mary 295
CA F 1910 Ann 47
CA M 2005 Zain 20
CA M 2005 Alijah 37
CA F 2015 Luisa 40
CA M 2015 Jorge 460
GroupBy Object
.agg(sum)
1910 342
2005 57
2015 500
Output DataFrame

groupby REVIEW QUESTION



ANSWER


groupby
.agg(f), where f = max
A 3 ak
B 1 tx
C 4 fl
A 1 hi
B 5 mi
C 9 ak
A 2 ca
C 5 sd
B 6 nc
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
A 3 hi
B 6 tx
C 9 sd

QUESTION

	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

→

Party	Year	Candidate	Popular vote	Result	%
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

THANK YOU

usman.nazir@lums.edu.pk
usmanweb.github.io