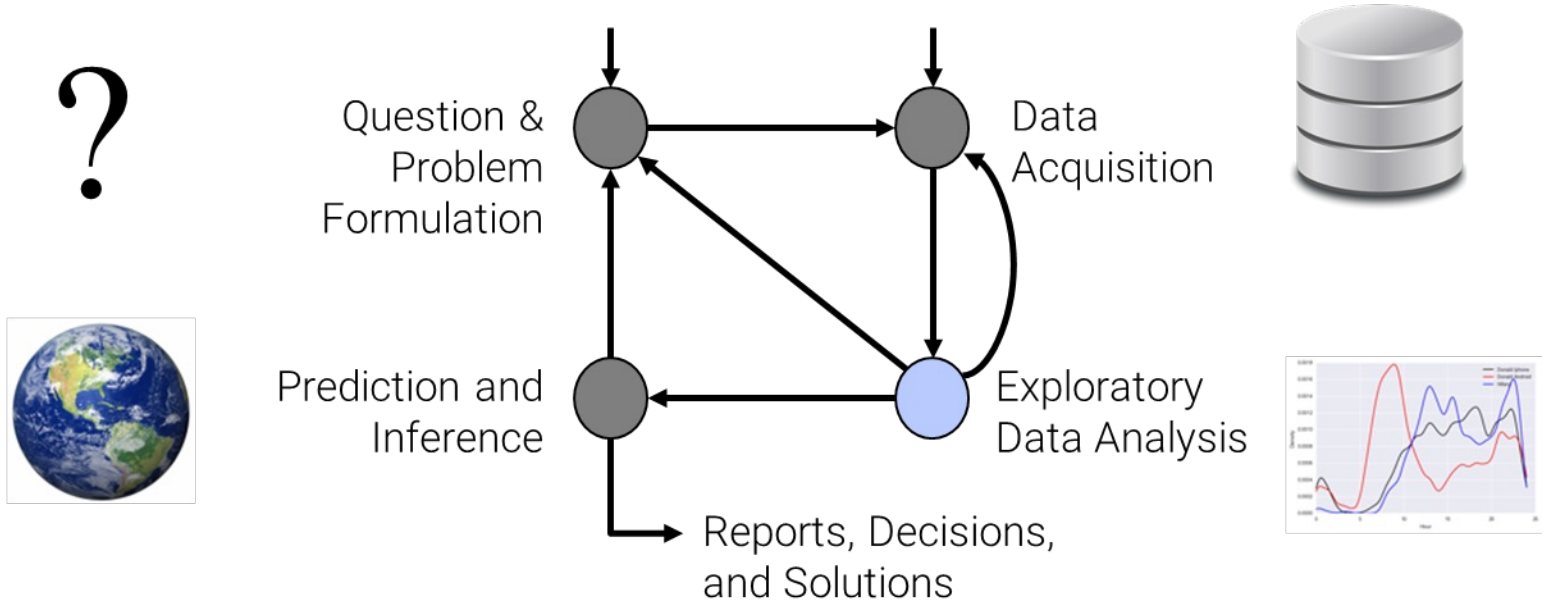
Introduction to Data Science

Pandas

Usman Nazir

Agenda

- What is Pandas?



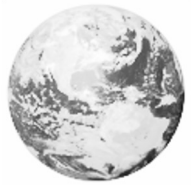
The Python Data Analysis Library

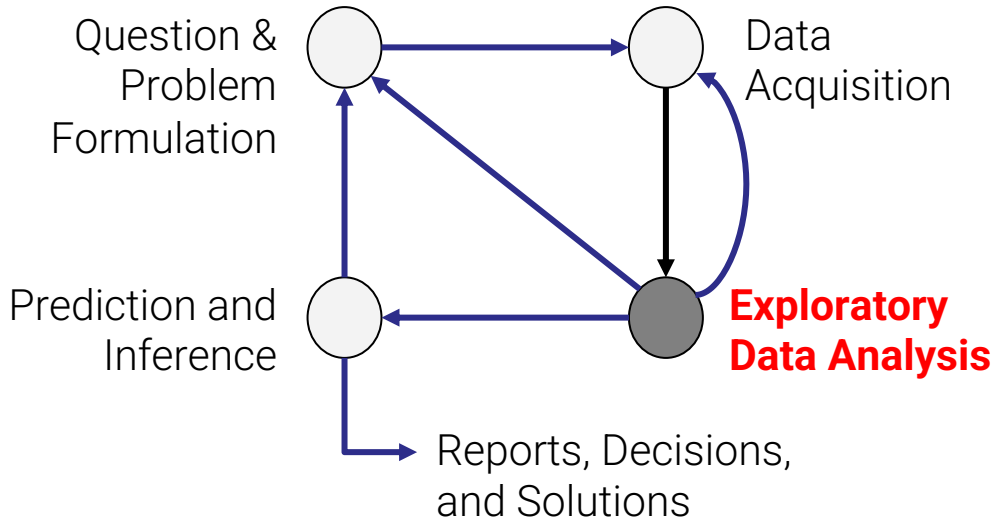
Week 4: Pandas I

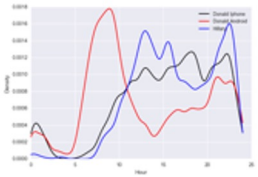
Usman Nazir

From data science to pandas

?







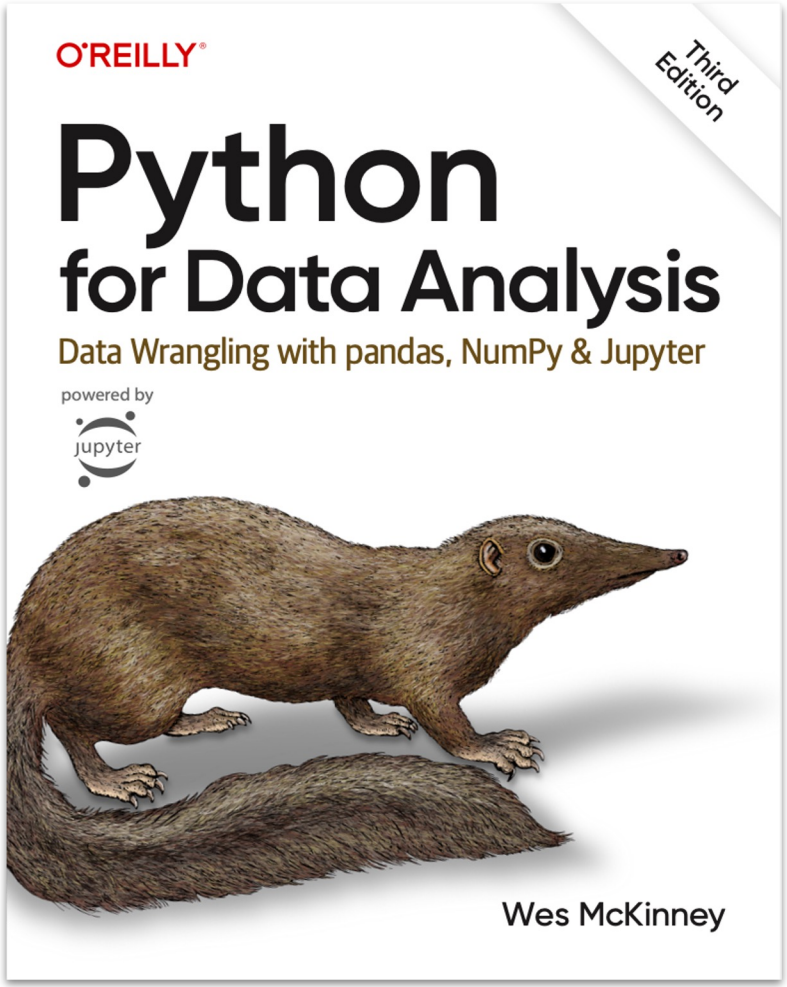
Exploring and Cleaning Tabular Data

Pandas

- **Pandas** (derived from Panel Data) is a **Data Analysis library** to make data cleaning and analysis fast and convenient in Python.
- Pandas adopts many coding idioms from **NumPy**, the biggest difference is that pandas is designed for working with tabular or **heterogeneous data**.
  - **Numpy** by contrast is best suited for working with **homogenous** numerical data.

- Tabular data is one of the most common data formats.
- Will be our primary focus in this course (though not 100%!)

You have free access to a fantastic book by the creator of Pandas!



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

Pandas Data Structures

There are three fundamental data structures in pandas:

- **Series**: 1D labeled array data. I usually think of it as columnar data.
- **Data Frame**: 2D tabular data with both row and column labels
- **Index**: A sequence of row/column labels.

Data Frame

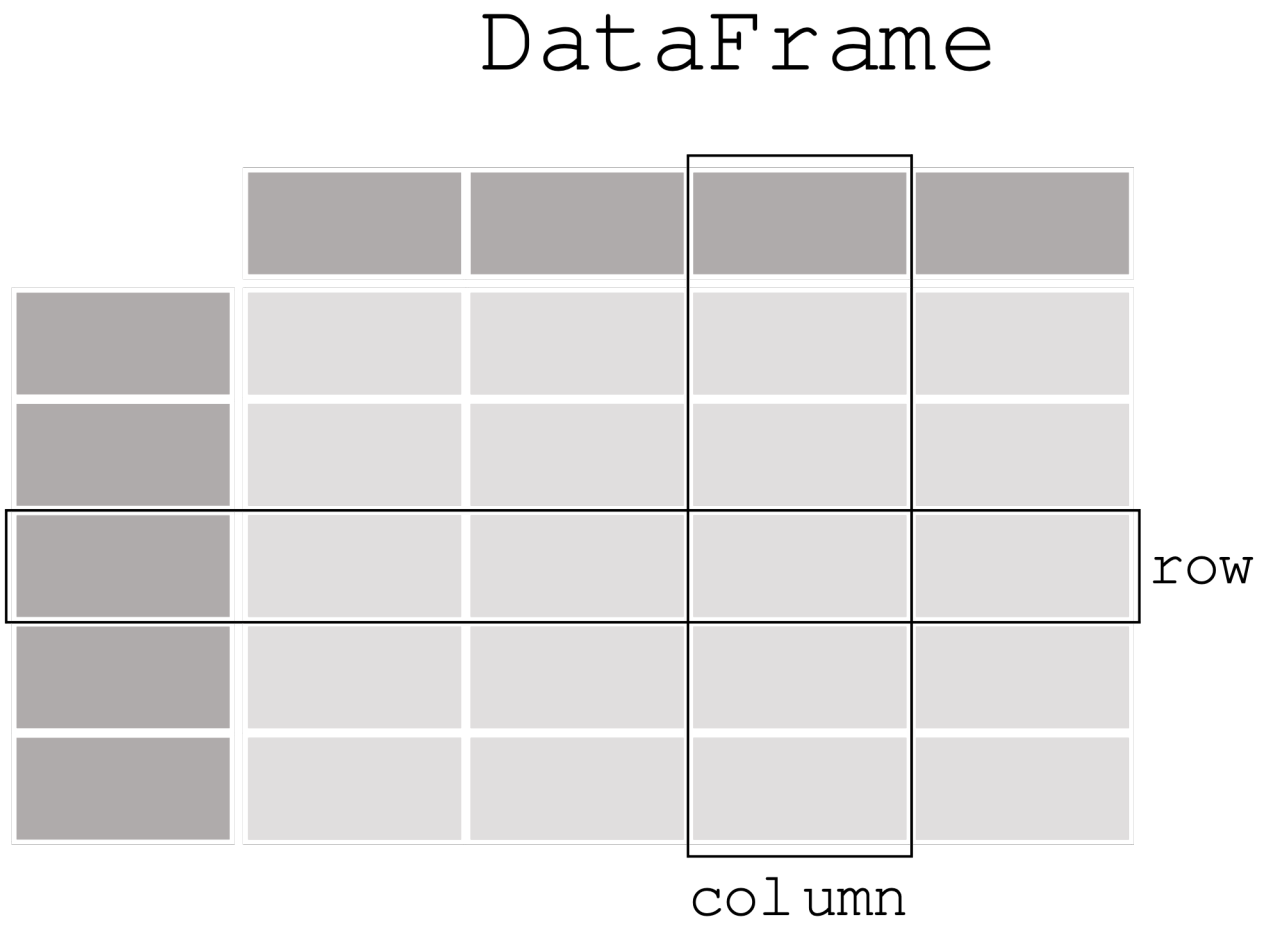
	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

Series

```
0       Obama
1      McCain
2       Obama
3      Romney
4     Clinton
5       Trump
Name: Candidate, dtype: object
```

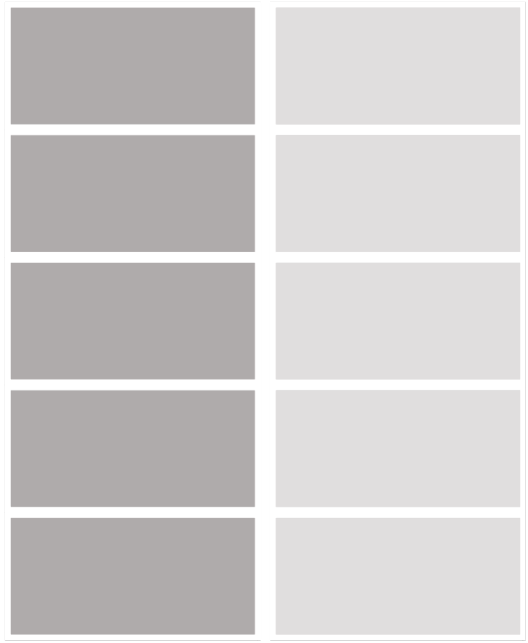
Index

pandas data table representation



Each column in a DataFrame is a Series

Series



Summary

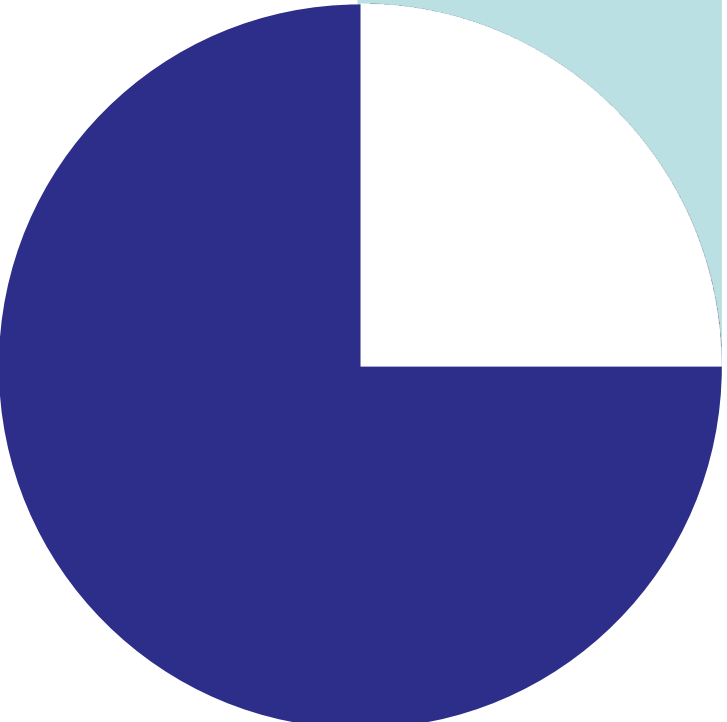
- What is Pandas?

Introduction to Data Science

Series, DataFrames, and Indices

Usman Nazir

Series, DataFrames, and Indices



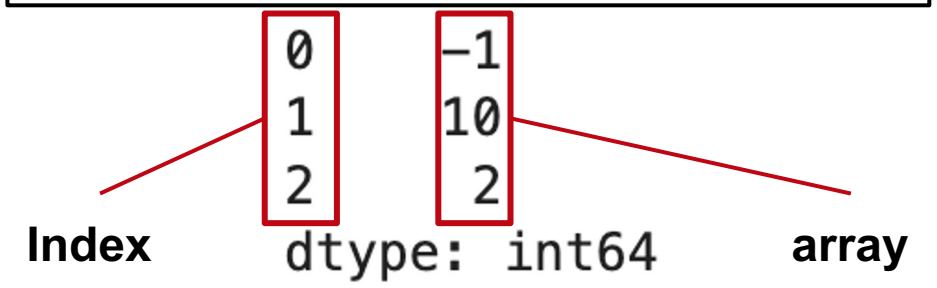
Series, DataFrames, and Indices

- Slicing with loc, iloc, and []
- Demo

Series

- A Series is a 1-dimensional **array-like object** containing a sequence of values of the same type and an associated array of data labels, called its **index**.

```
s = pd.Series([-1, 10, 2])
```




```
s.array
```

```
<PandasArray>
[-1, 10, 2]
Length: 3, dtype: int64
```

```
s.index
```

```
RangeIndex(start=0, stop=3, step=1)
```

Series – custom index

- We can provide index labels for items in a Series by passing an index list.

```
s = pd.Series([-1, 10, 2], index = ["a", "b", "c"])
```

```
a    -1
b    10
c     2
dtype: int64
```

```
s.index
```

```
Index(['a', 'b', 'c'], dtype='object')
```

- A Series index can also be changed.

```
s.index = ["first", "second", "third"])
```

```
first     -1
second    10
third      2
dtype: int64
```

```
s.index
```

```
Index(['first', 'second', 'third'], dtype='object')
```

Selection in series

- We can select a single value or a set of values in a Series using:
  - A single label
  - A list of labels
  - A filtering condition

```
s = pd.Series([4, -2, 0, 6], index = ["a", "b", "c", "d"])
```

```
a    4
b   -2
c    0
d    6
dtype: int64
```

Selection in series

- We can select a single value or a set of values in a Series using:
  - **A single label**
  - A list of labels
  - A filtering condition

```
s = pd.Series([4, -2, 0, 6], index = ["a", "b", "c", "d"])
```

```
a    4
b   -2
c    0
d    6
dtype: int64
```

```
s["a"]
```

```
4
```

Selection in series

- We can select a single value or a set of values in a Series using:
  - A single label
  - **A list of labels**
  - A filtering condition

```
s = pd.Series([4, -2, 0, 6], index = ["a", "b", "c", "d"])
```

```
a    4
b   -2
c    0
d    6
dtype: int64
```

```
s[["a", "c"]]
```

```
a    4
c    0
dtype: int64
```

Selection in series

- We can select a single value or a set of values in a Series using:
  - A single label
  - A list of labels
  - **A filtering condition**

```
s = pd.Series([4, -2, 0, 6], index = ["a", "b", "c", "d"])
```

```
a    4
b   -2
c    0
d    6
dtype: int64
```

- We first must apply a vectorized boolean operation to our Series that encodes the filter condition.

```
s > 0
```

```
a     True
b    False
c    False
d     True
dtype: bool
```

- Upon “indexing” in our Series with this condition, pandas selects only the rows with True values.

```
s[s > 0]
```

```
a    4
d    6
dtype: int64
```

Dataframe

- A DataFrame represents a table of data, containing a named collection of columns.
- The DataFrame can be thought of as a dictionary of Series all sharing the same index.

```
elections = pd.read_csv("elections.csv")
```

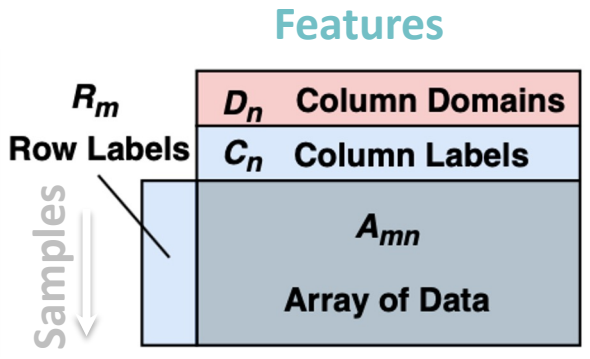
	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832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

182 rows × 6 columns

https://drive.google.com/drive/folders/1rVTrh3T3dtFr_CKXnXiTMwYFJFqVVUBL?usp=sharing

The world

A (statistical) population from which we draw samples.
Each sample has certain features.



	Year	Candidate	Party	Popular vote	Result	%
177	2016	Jill Stein	Green	1457226	loss	1.073699
178	2020	Joseph Biden	Democratic	81268924	win	51.311515
179	2020	Donald Trump	Republican	74216154	loss	46.858542
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
181	2020	Howard Hawkins	Green	405035	loss	0.255731

Here, our population is a census of all major party candidates since 1824.

A generic DataFrame (from https://arxiv.org/abs/2001.00888)

The dataframe api

The API for the DataFrame class is enormous.

- API: “Application Programming Interface”
- The API is the set of abstractions supported by the class.

Full documentation is at https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

- We will only consider a tiny portion of this API.

We want you to get familiar with the real-world programming practice of… Googling!

- Answers to your questions are often found in the panda's documentation, stack overflow, etc.

With that warning, let’s dive in.

Creating a dataframe

Many approaches exist for creating a DataFrame. Here, we will go over the most popular ones.

- Using a list and column name(s)
- From a dictionary
- From a Series

```
pandas.DataFrame(data, index, columns)
```

Creating a dataframe

```
pandas.DataFrame(data, index, columns)
```

Many approaches exist for creating a DataFrame. Here, we will go over the most popular ones.

- **Using a list and column name(s)**
- From a dictionary
- From a Series

```
pd.DataFrame([1, 2, 3],
             columns=["Numbers"])
```

	Numbers
0	1
1	2
2	3

```
pd.DataFrame([[1, "one"], [2, "two"]],
             columns = ["Number", "Description"])
```

	Number	Description
0	1	one
1	2	two

Creating a dataframe

```
pandas.DataFrame(data, index, columns)
```

Many approaches exist for creating a DataFrame. Here, we will go over the most popular ones.

- Using a list and column name(s)
- **From a dictionary**
- From a Series

```
pd.DataFrame({"Fruit":["Strawberry", "Orange"],
              "Price": [5.49, 3.99]})
```

```
pd.DataFrame([{"Fruit":"Strawberry", "Price":5.49},
              {"Fruit":"Orange", "Price":3.99}])
```

	Fruit	Price
0	Strawberry	5.49
1	Orange	3.99

Creating a dataframe

```
pandas.DataFrame(data, index, columns)
```

Many approaches exist for creating a DataFrame. Here, we will go over the most popular ones.

- Using a list and column name(s)
- From a dictionary
- **From a Series**

```
s_a = pd.Series(["a1", "a2", "a3"], index = ["r1", "r2", "r3"])
s_b = pd.Series(["b1", "b2", "b3"], index = ["r1", "r2", "r3"])

pd.DataFrame({"A-column":s_a, "B-column":s_b})
```

	A-column	B-column
r1	a1	b1
r2	a2	b2
r3	a3	b3

```
pd.DataFrame(s_a)
```

```
s_a.to_frame()
```

	0
r1	a1
r2	a2
r3	a3

Summary

- What is Pandas?
  - Series
  - DataFrame
  - Indices

Introduction to Data Science

Series, DataFrames, and Indices

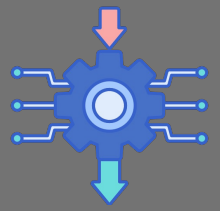
Usman Nazir

What is data science?

Learning about the world from data using computation

Exploration

- Identifying patterns in data
- Uses Visualizations



Inference

- Using data to draw reliable conclusions about the world
- Uses Statistics

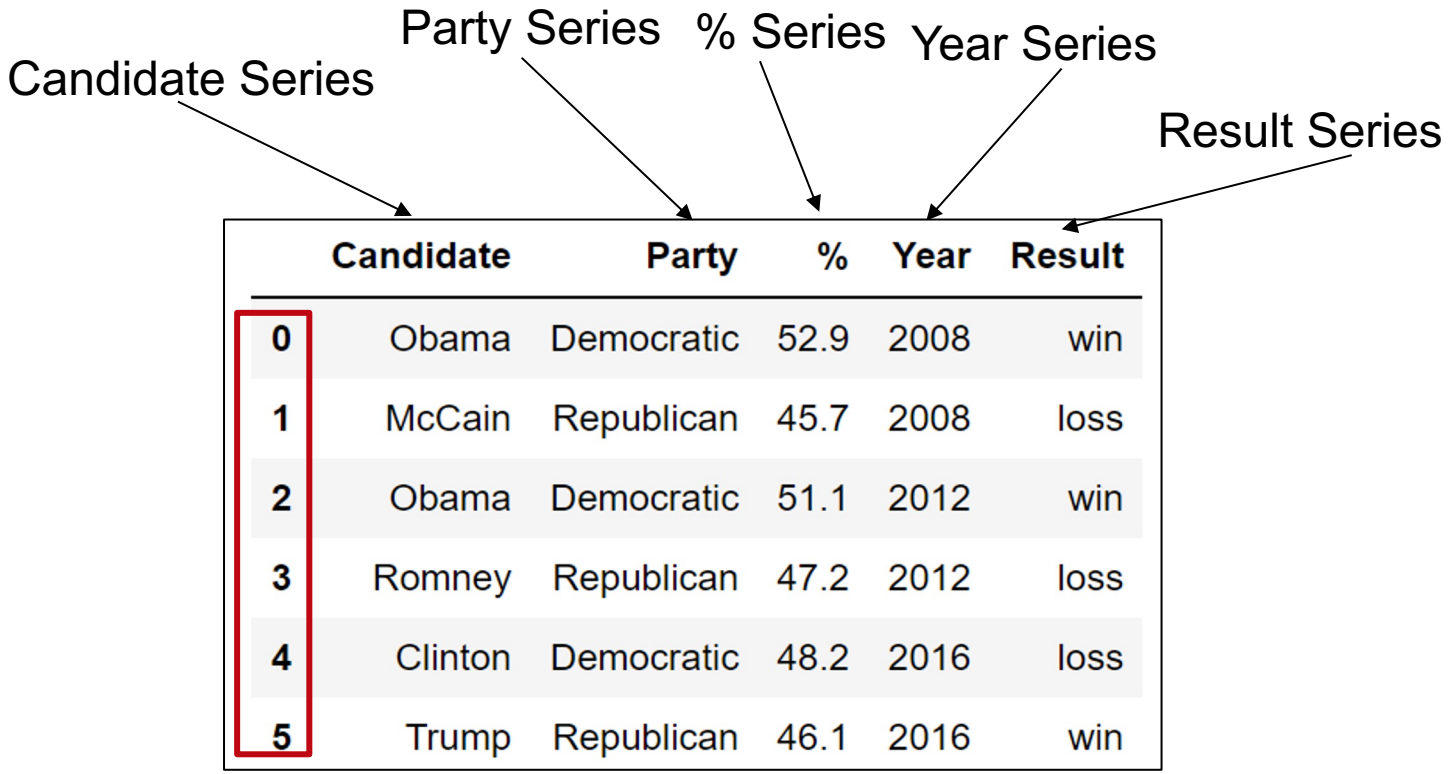
Prediction

- Making informed guesses about the unobserved data
- Uses Machine Learning

The Relationship Between Data Frames, Series, and Indices

We can think of a **Data Frame** as a collection of **Series** that all share the same .

- Candidate, Party, %, Year, and Result **Series** all share an **Index** from 0 to 5.



	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

Indices Are Not Necessarily Row Numbers

An **Index** (a.k.a. row labels) can also:

- Be non-numeric.
- Have a name, e.g. “State”.

```
mottos = pd.read_csv("mottos.csv", index_col = "State")
```

State	Motto	Translation	Language	Date Adopted
Alabama	Audemus jura nostra defendere	We dare defend our rights!	Latin	1923
Alaska	North to the future	—	English	1967
Arizona	Ditat Deus	God enriches	Latin	1863
Arkansas	Regnat populus	The people rule	Latin	1907
California	Eureka (Εὕρηκα)	I have found it	Greek	1849

Indices

The row labels that constitute an index do not have to be unique.

- Left: The **index** values are all unique and numeric, acting as a row number.
- Right: The **index** values are named and non-unique.

	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

	Candidate	Party	%	Result
Year				
2008	Obama	Democratic	52.9	win
2008	McCain	Republican	45.7	loss
2012	Obama	Democratic	51.1	win
2012	Romney	Republican	47.2	loss
2016	Clinton	Democratic	48.2	loss
2016	Trump	Republican	46.1	win

Modifying indices

- We can select a new column and set it as the index of the DataFrame

Example: Setting the index to the Candidate column

```
elections.set_index("Candidate", inplace=True)
```

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

182 rows × 5 columns

Resetting index

- We can select a new column and set it as the index of the DataFrame

Example: Resetting the index to the default list of integers

```
elections.reset_index(inplace=True)
```

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

182 rows × 5 columns

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

182 rows × 6 columns

Column Names Are Usually Unique!

Column names in Pandas are almost always unique!

- Example: Really shouldn't have two columns named "Candidate".
- You can force duplicate columns into existence if you want.

	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

Retrieving Row and Column Labels

Sometimes you'll want to extract the list of row and column labels.

- For row labels, use `DataFrame.index`:

```
mottos.index
```

```
Index(['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California', 'Colorado',
       'Connecticut', 'Delaware', 'Florida', 'Georgia', 'Hawaii', 'Idaho',
       'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana',
       'Maine', 'Maryland', 'Massachusetts', 'Michigan', 'Minnesota',
       'Mississippi', 'Missouri', 'Montana', 'Nebraska', 'Nevada',
       'New Hampshire', 'New Jersey', 'New Mexico', 'New York',
       'North Carolina', 'North Dakota', 'Ohio', 'Oklahoma', 'Oregon',
       'Pennsylvania', 'Rhode Island', 'South Carolina', 'South Dakota',
       'Tennessee', 'Texas', 'Utah', 'Vermont', 'Virginia', 'Washington',
       'West Virginia', 'Wisconsin', 'Wyoming'],
      dtype='object', name='State')
```

- For column labels, use DataFrame.columns: `mottos.columns`

```
Index(['Motto', 'Translation', 'Language', 'Date Adopted'], dtype='object')
```

Summary

- What is Pandas?
  - Series
  - DataFrame
  - Indices

Introduction to Data Science

Slicing with loc, iloc, and []

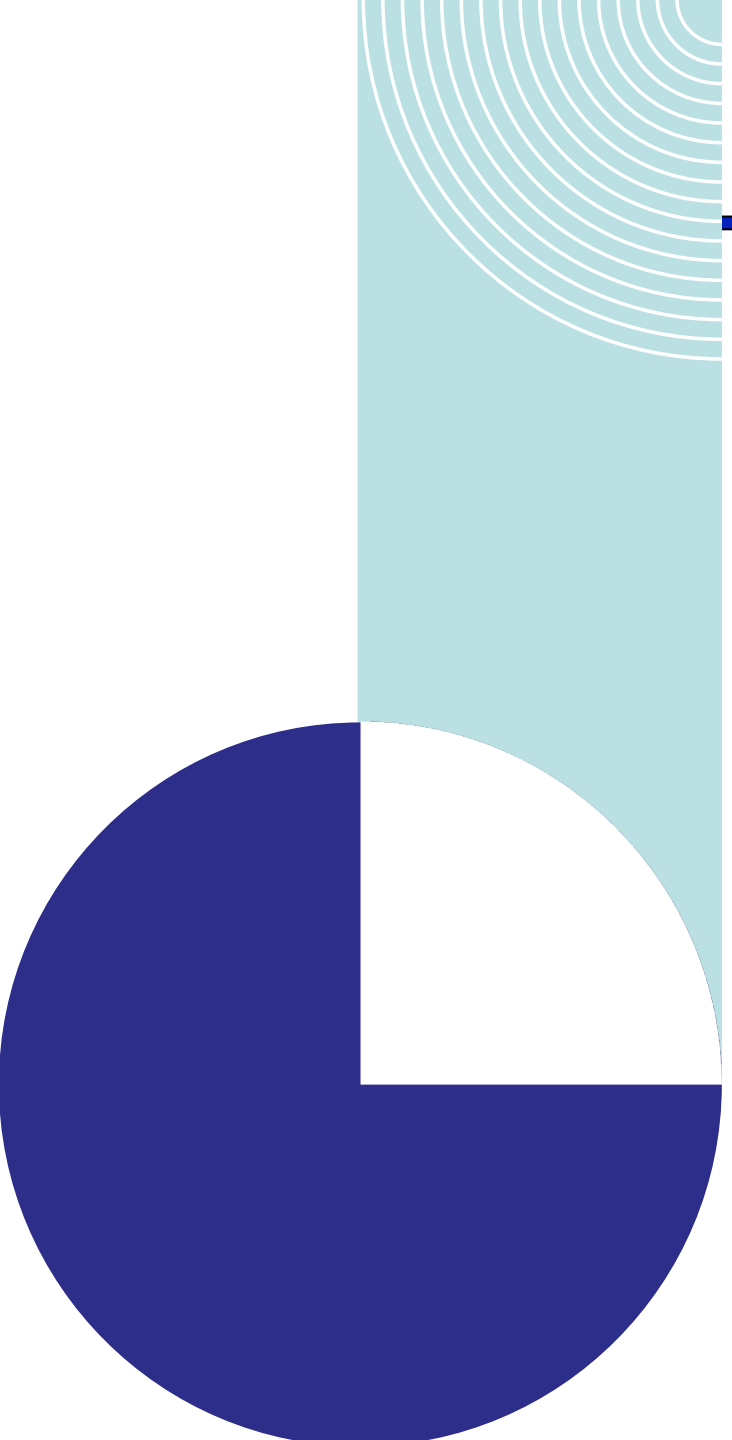
Usman Nazir

The Python Data Analysis Library

Week 5: Pandas I

Slicing with loc, iloc, and []

Usman Nazir



Series, DataFrames, and Indices

Series, DataFrames, and Indices

- **Slicing with loc, iloc, and []**
- Demo

Very Basic Slicing Example

One of the most basic tasks for manipulating a DataFrame is to extract rows and columns of interest. As we'll see, the large pandas API means there are many ways to do things.

For example, consider the loc operator, which we'll learn about shortly.

```
elections.loc[0:4]
```

	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

loc is not a function! It's an operator.

Very Basic Slicing Example

Example methods in API:

The Pandas library has a lot of “syntactic sugar”: Methods that are useful and lead to concise code, but not absolutely necessary for the library to function.

- Examples: **.head** and **.tail**.

```
elections.head(5)
```

	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

Equivalent to `elections.loc[0:4]`

```
elections.tail(5)
```

	Year	Candidate	Party	Popular vote	Result	%
177	2016	Jill Stein	Green	1457226	loss	1.073699
178	2020	Joseph Biden	Democratic	81268924	win	51.311515
179	2020	Donald Trump	Republican	74216154	loss	46.858542
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
181	2020	Howard Hawkins	Green	405035	loss	0.255731

Equivalent to `elections.loc[177:]` or `elections[-5:]`

Very Basic Slicing Example

- `loc` also lets us specify the columns that we want as a second argument.

```
elections.loc[0:4, "Year":"Party"]
```

	Year	Candidate	Party
0	1824	Andrew Jackson	Democratic-Republican
1	1824	John Quincy Adams	Democratic-Republican
2	1828	Andrew Jackson	Democratic
3	1828	John Quincy Adams	National Republican
4	1832	Andrew Jackson	Democratic

Very Basic Slicing Example

Fundamentally `loc` selects items by **label**.

- The labels are the **bolded** text to the top and left of our dataframe.
- Row labels shown: **177**, **178**, **179**, **180**, **181**
- Column labels: **Year**, **Candidate**, **Party**, **Popular vote**, **Result**, **%**

	Year	Candidate	Party	Popular vote	Result	%
177	2016	Jill Stein	Green	1457226	loss	1.073699
178	2020	Joseph Biden	Democratic	81268924	win	51.311515
179	2020	Donald Trump	Republican	74216154	loss	46.858542
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
181	2020	Howard Hawkins	Green	405035	loss	0.255731

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 .
  - A single .

loc

Arguments to loc can be:

- A list.
- A slice (syntax is inclusive of the right hand side of the slice).
- A single value.

	Year	Candidate	Party	Popular vote	Result	%
177	2016	Jill Stein	Green	1457226	loss	1.073699
178	2020	Joseph Biden	Democratic	81268924	win	51.311515
179	2020	Donald Trump	Republican	74216154	loss	46.858542
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
181	2020	Howard Hawkins	Green	405035	loss	0.255731

loc

Arguments to loc can be:

- **A list.**
- A slice (syntax is inclusive of the right hand side of the slice).
- A single value.

```
elections.loc[[87, 25, 179], ["Year", "Candidate", "Result"]]
```

	Year	Candidate	Result
87	1932	Herbert Hoover	loss
25	1860	John C. Breckinridge	loss
179	2020	Donald Trump	loss

loc

Arguments to loc can be:

- A list.
- **A slice (syntax is inclusive of the right hand side of the slice).**
- A single value.

```
elections.loc[[87, 25, 179], "Popular vote":"%"]
```

	Popular vote	Result	%
87	15761254	loss	39.830594
25	848019	loss	18.138998
179	74216154	loss	46.858542

loc

Arguments to loc can be:

- A list.
- A slice (syntax is inclusive of the right hand side of the slice).
- **A single value.**

```
elections.loc[[87, 25, 179], "Popular vote"]
```

```
87     15761254
25       848019
179    74216154
Name: Popular vote, dtype: int64
```

Wait, what? Why did everything get so ugly?

Series?

```
elections.loc[0, "Candidate"]
```

```
'Andrew Jackson'
```

The type is ‘str’.

loc

As we saw earlier, you can omit the second argument if you want all columns.

- If you want all rows, but only some columns, you can use `:` for the left argument.

```
elections.loc[:, ["Year", "Candidate", "Result"]]
```

	Year	Candidate	Result
0	1824	Andrew Jackson	loss
1	1824	John Quincy Adams	win
2	1828	Andrew Jackson	win
3	1828	John Quincy Adams	loss
4	1832	Andrew Jackson	win
...	...	...	...
177	2016	Jill Stein	loss
178	2020	Joseph Biden	win
179	2020	Donald Trump	loss
180	2020	Jo Jorgensen	loss
181	2020	Howard Hawkins	loss

Summary

- **loc**, iloc, []

Introduction to Data Science

Slicing with loc, **iloc**, and []

Usman Nazir

iloc

Pandas also supports another operator called `iloc`.

Fundamentally **`iloc`** selects items by **number**.

- Row numbers are 0 through 181 (in this example, same as labels!).
- Column numbers are 0 through 5.

iloc

Arguments to iloc can be:

- A list.
- A slice (syntax is **exclusive** of the right hand side of the slice).
- A single value.

iloc

Arguments to iloc can be:

- **A list.**
- A slice (syntax is **exclusive** of the right hand side of the slice).
- A single value.

```
elections.iloc[[1, 2, 3], [0, 1, 2]]
```

	Year	Candidate	Party
1	1824	John Quincy Adams	Democratic-Republican
2	1828	Andrew Jackson	Democratic
3	1828	John Quincy Adams	National Republican

iloc

Arguments to iloc can be:

- A list.
- **A slice (syntax is exclusive of the right hand side of the slice).**
- A single value.

```
elections.iloc[[1, 2, 3], 0:3]
```

	Year	Candidate	Party
1	1824	John Quincy Adams	Democratic-Republican
2	1828	Andrew Jackson	Democratic
3	1828	John Quincy Adams	National Republican

iloc

Arguments to iloc can be:

- A list.
- A slice (syntax is **exclusive** of the right hand side of the slice).
- **A single value.**

```
elections.iloc[[1, 2, 3], 1]
```

```
1    John Quincy Adams
2       Andrew Jackson
3    John Quincy Adams
Name: Candidate, dtype: object
```

As before, the result for a single value argument is a Series.

```
elections.loc[0, 1]
```

```
'Andrew Jackson'
```

The type is ‘str’.

iloc

- Just like loc:

```
elections.iloc[:, 0:3]
```

	Year	Candidate	Party
0	1824	Andrew Jackson	Democratic-Republican
1	1824	John Quincy Adams	Democratic-Republican
2	1828	Andrew Jackson	Democratic
3	1828	John Quincy Adams	National Republican
4	1832	Andrew Jackson	Democratic
...	...	...	...
177	2016	Jill Stein	Green
178	2020	Joseph Biden	Democratic
179	2020	Donald Trump	Republican
180	2020	Jo Jorgensen	Libertarian
181	2020	Howard Hawkins	Green

loc vs iloc

When choosing between **loc** and **iloc**, you'll usually choose .

- Safer: If the order of columns gets shuffled in a public database, your code still works.
- Legible: Easier to understand what `elections.loc[:, ["Year", "Candidate", "Result"]]` means than `elections.iloc[:, [0, 1, 4]]`

iloc can still be useful.

- Example: If you have a DataFrame of movie earnings sorted by earnings, can use **iloc** to get the median earnings for a given year (index into the middle).

Summary

- loc, **iloc**, []

Introduction to Data Science

Slicing with loc, iloc, and []

Usman Nazir

[]

[ ] only takes one argument, which may be:

- **A slice of row numbers**.
- A list of .
- A single .

```
elections[3:7]
```

	Year	Candidate	Party	Popular vote	Result	%
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
5	1832	Henry Clay	National Republican	484205	loss	37.603628
6	1832	William Wirt	Anti-Masonic	100715	loss	7.821583

[]

[ ] only takes one argument, which may be:

- A slice of row numbers.
- **A list of .**
- A single

```
elections[["Year", "Candidate", "Result"]].tail(5)
```

	Year	Candidate	Result
177	2016	Jill Stein	loss
178	2020	Joseph Biden	win
179	2020	Donald Trump	loss
180	2020	Jo Jorgensen	loss
181	2020	Howard Hawkins	loss

[]

[ ] only takes one argument, which may be:

- A slice of row numbers.
- A list of .
- **A single .**

```
elections["Candidate"].tail(5)
```

```
177         Jill Stein
178      Joseph Biden
179      Donald Trump
180      Jo Jorgensen
181    Howard Hawkins
Name: Candidate, dtype: object
```

Same as before, the output type is a Series.

Summary

- loc, iloc, []

Introduction to Data Science

Slicing with loc, iloc, and []

Usman Nazir

The Python Data Analysis Library

Week 6: Pandas II

Utility Functions, Grouping, Aggregation

Usman Nazir

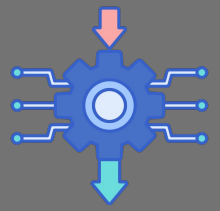
What is data science?

What is data science?

Learning about the world from data using computation

Exploration

- Identifying patterns in data
- Uses Visualizations



Inference

- Using data to draw reliable conclusions about the world
- Uses Statistics

Prediction

- Making informed guesses about the unobserved data
- Uses Machine Learning

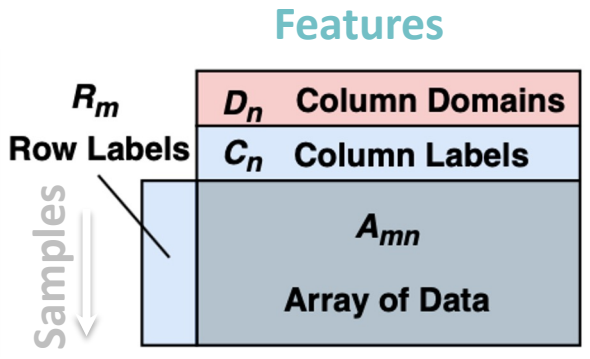
What is AI?

The science of making machines that:

What is a data-frame?

The world

A (statistical) population from which we draw samples.
Each sample has certain features.



	Year	Candidate	Party	Popular vote	Result	%
177	2016	Jill Stein	Green	1457226	loss	1.073699
178	2020	Joseph Biden	Democratic	81268924	win	51.311515
179	2020	Donald Trump	Republican	74216154	loss	46.858542
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
181	2020	Howard Hawkins	Green	405035	loss	0.255731

Here, our population is a census of all major party candidates since 1824.

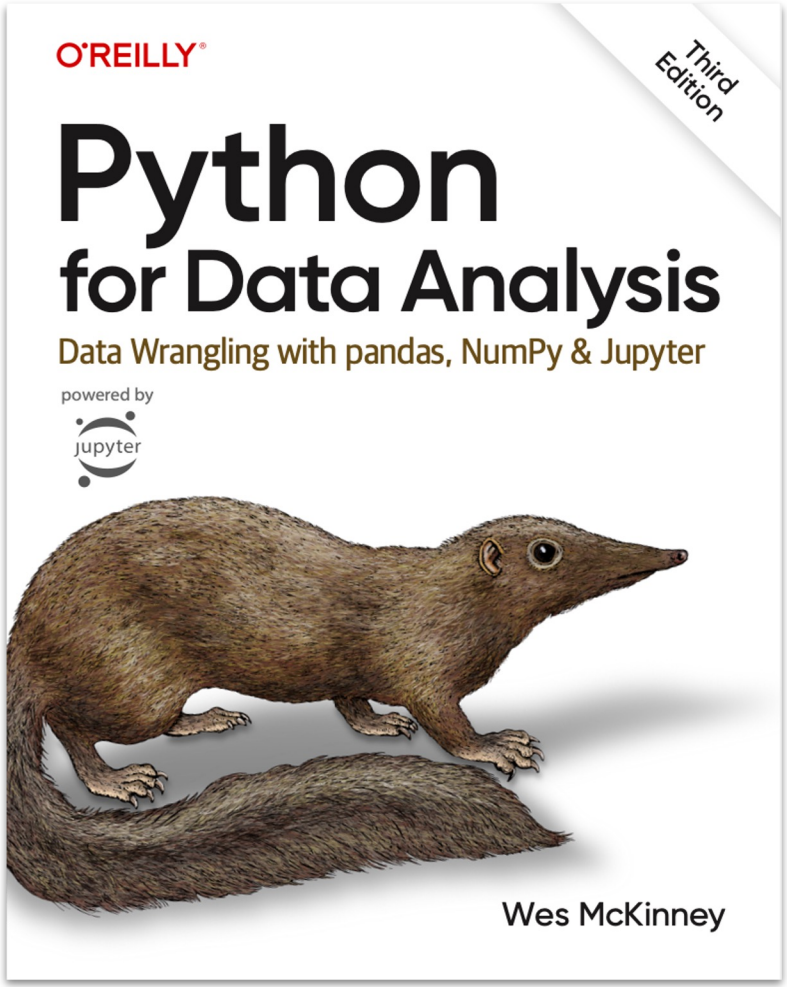
A generic DataFrame (from https://arxiv.org/abs/2001.00888)

Tentative List of Topics to be Covered

- Pandas and NumPy
- Exploratory Data Analysis
- Regular Expressions
- Visualization
  - matplotlib
  - Seaborn
  - Plotly
- Sampling
- Probability and random variables
- Model design and loss formulation
- Data science in the physical world

- Linear Regression
- Feature Engineering
- Regularization, Bias-Variance Tradeoff, Cross-Validation
- Gradient Descent
- Causality
- Logistic Regression
- Clustering
- PCA
- **Advanced topics in ML (CNN, LSTM, Transformers)**

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 .
  - **A single** .

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

```
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

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.   [[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".

```
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

- 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`
- `uniques`
- `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

Introduction to Data Science

Slicing with loc, iloc, and []

Usman Nazir

The Python Data Analysis Library

The Foundations of Data Science
Week 7: Pandas II

Utility Functions, Custom sorts, Grouping, Aggregation

Usman Nazir

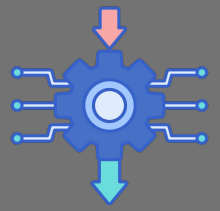
What is data science?

What is data science?

Learning about the world from data using computation

Exploration

- Identifying patterns in data
- Uses Visualizations



Inference

- Using data to draw reliable conclusions about the world
- Uses Statistics

Prediction

- Making informed guesses about the unobserved data
- Uses Machine Learning

What is AI?

The science of making machines that:

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?

	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

- 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?

	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", 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

The Python Data Analysis Library

The Foundations of Data Science
week 7: Pandas III
Utility Functions, Grouping, Aggregation

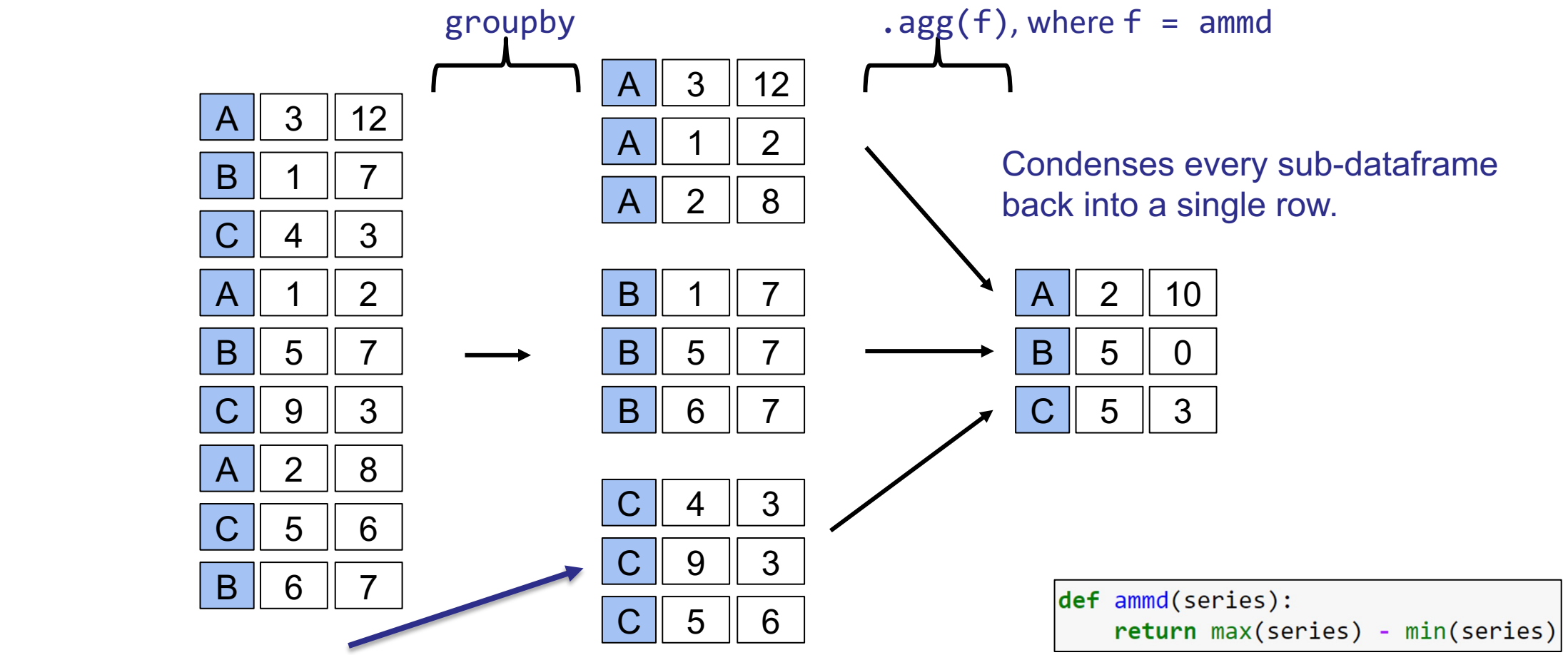
Usman Nazir

Groupby.agg

Conditional Selection

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

Grouping and Collection

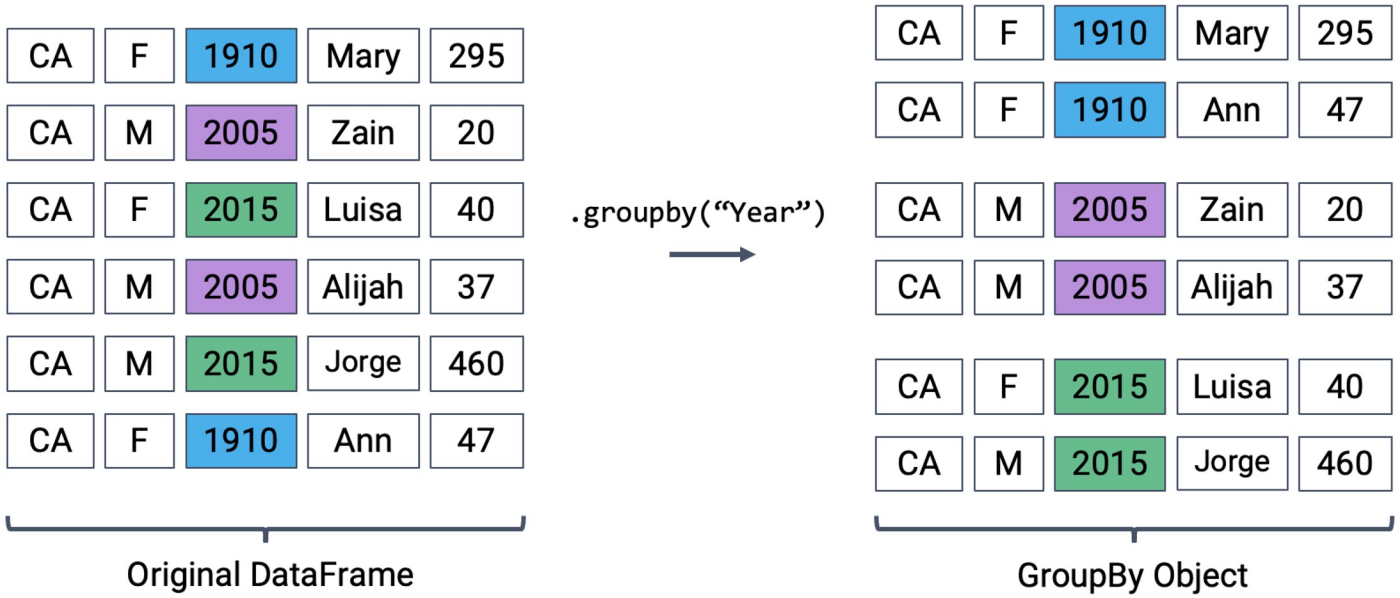



```
def ammd(series):
    return max(series) - min(series)
```

groupby()

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

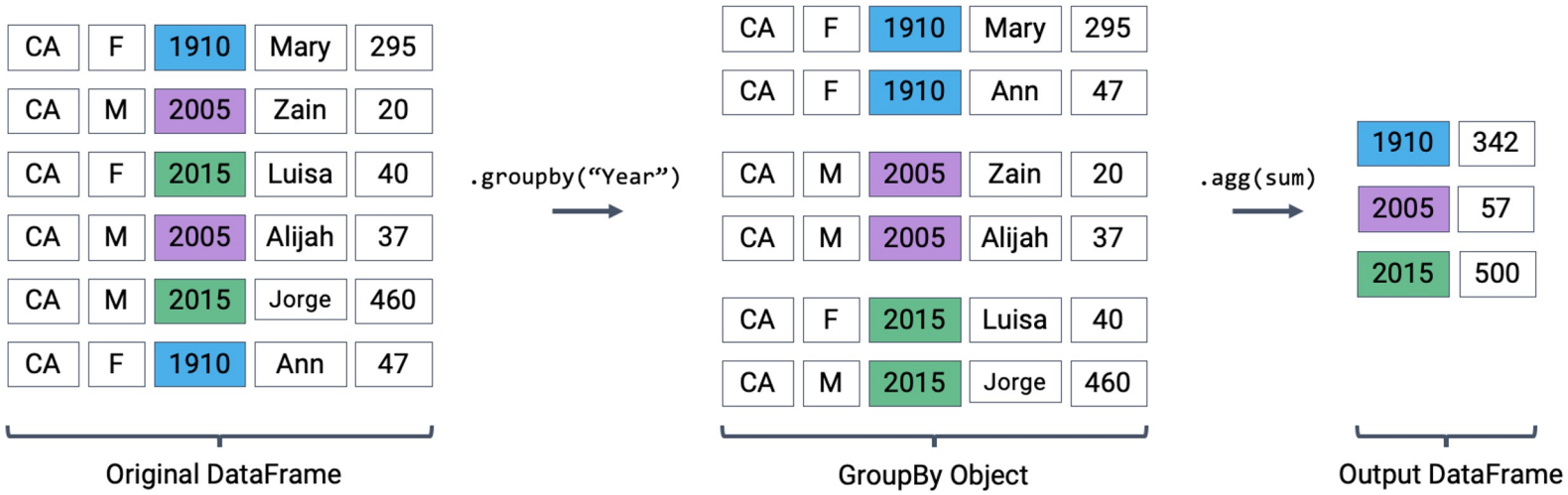
- Calling `.groupby()` generates `DataFrameGroupBy` objects → "mini" sub-DataFrames
- Each subframe contains all rows that correspond to a particular year



groupby.agg

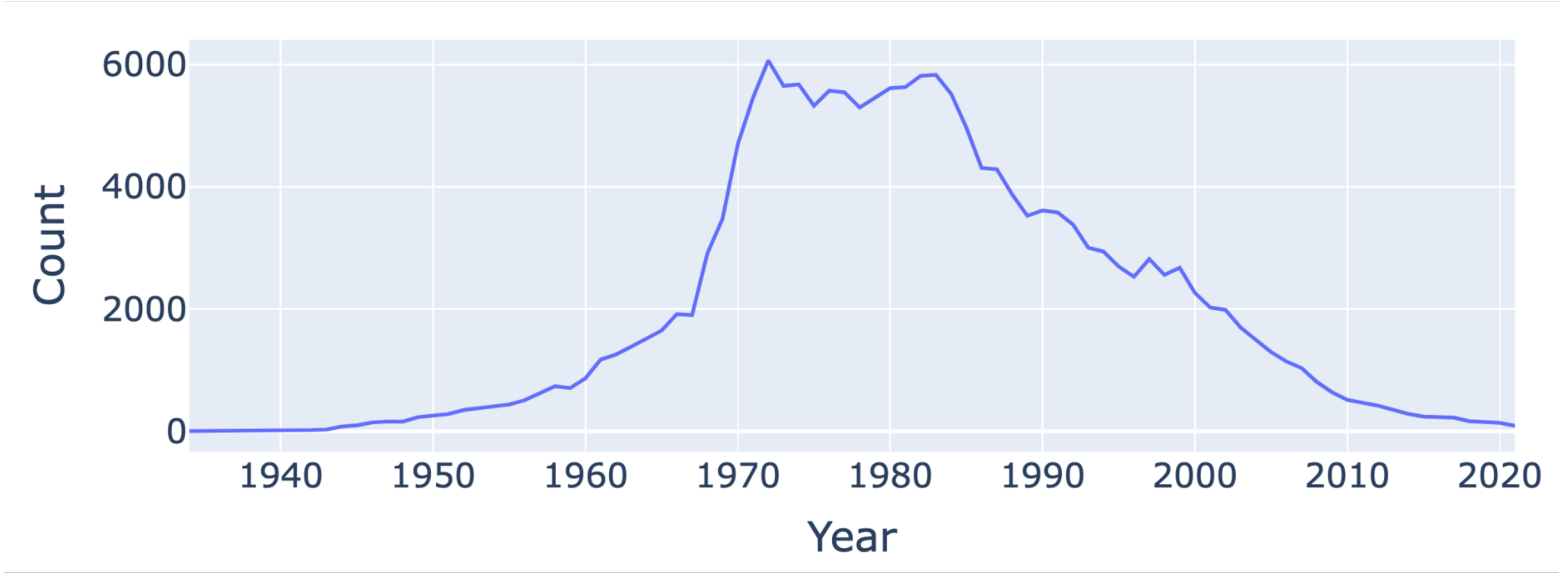
A `groupby` operation involves some combination of **splitting the object, applying a function**, and **combining the results**.

- 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"] == “Mary"]["Count"]
```

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":

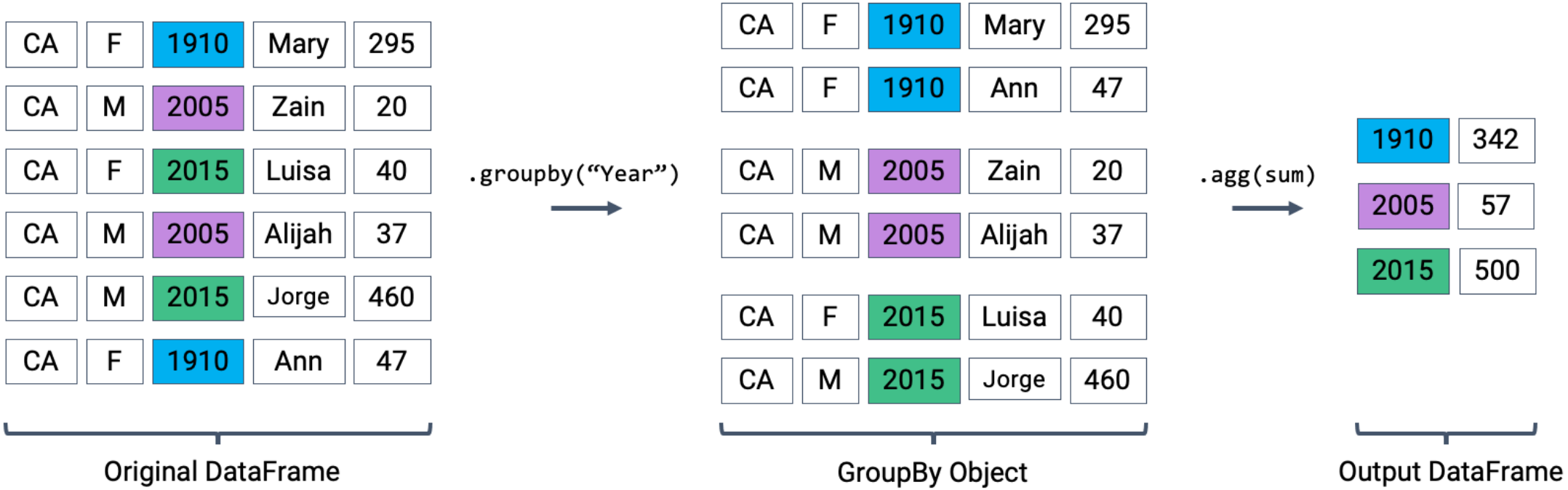
- In 1972, we hit peak Jennifer. 6,065 Jennifers were born.
- In 2021, there were only 91 Jennifers.
- RTP is 91 / 6065 = 0.015004.

Some groupby.agg Puzzles

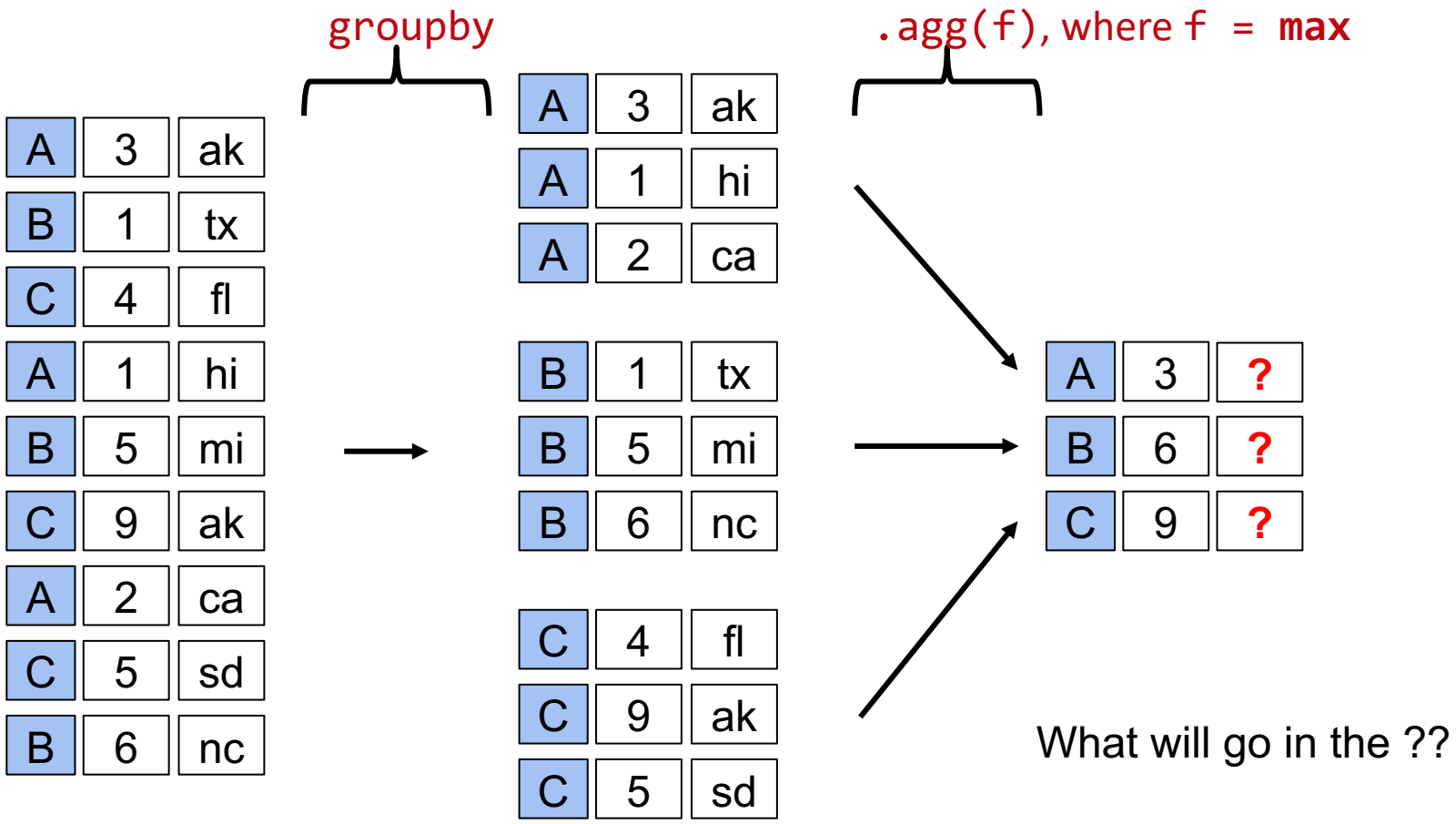
Conditional Selection

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

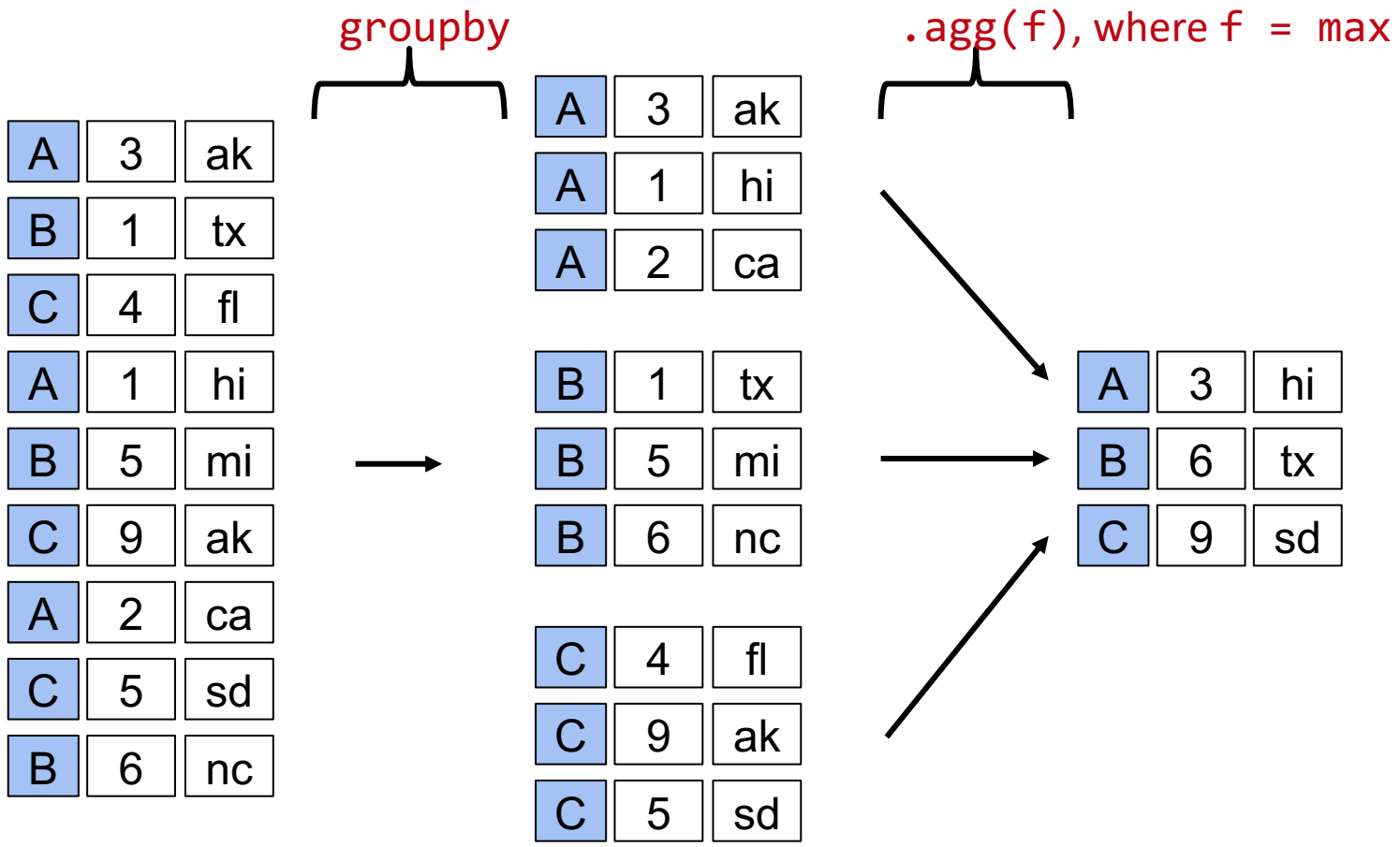
groupby


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

Find the Best Result by Party?

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

The Python Data Analysis Library

The Foundations of Data Science
Week 7: Pandas III

Advanced Pandas
(More on Grouping, Aggregation, Pivot Tables Merging)

Usman Nazir

New Syntax / Concept Summary

- Groupby: Output of .groupby("Name") is a DataFrameGroupBy object. Condense back into a DataFrame or Series with:
  - groupby.size
  - groupby.filter
  - and more...
- Pivot tables: An alternate way to group by exactly two columns.
- Joining tables using pd.merge.
- Exploratory data analysis

Groupby Puzzle #1

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

```
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	2020	Woodrow Wilson	81268924	win	61.344703
Democratic-Republican	1824	John Quincy Adams	151271	win	57.210122

Groupby Puzzle #1

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

Every column is calculated independently! Among Democrats:

- Last year they ran: 2020
- Alphabetically latest candidate name: Woodrow Wilson
- 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	2020	Woodrow Wilson	81268924	win	61.344703
Democratic-Republican	1824	John Quincy Adams	151271	win	57.210122

Groupby Puzzle #1

Very hard puzzle: 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.

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

Groupby Puzzle #1

Very hard puzzle: 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.
- Note: Lab will give you a chance to try this out if you didn't quite follow during lecture.

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

	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`

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

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!

Groupby Puzzle #1

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

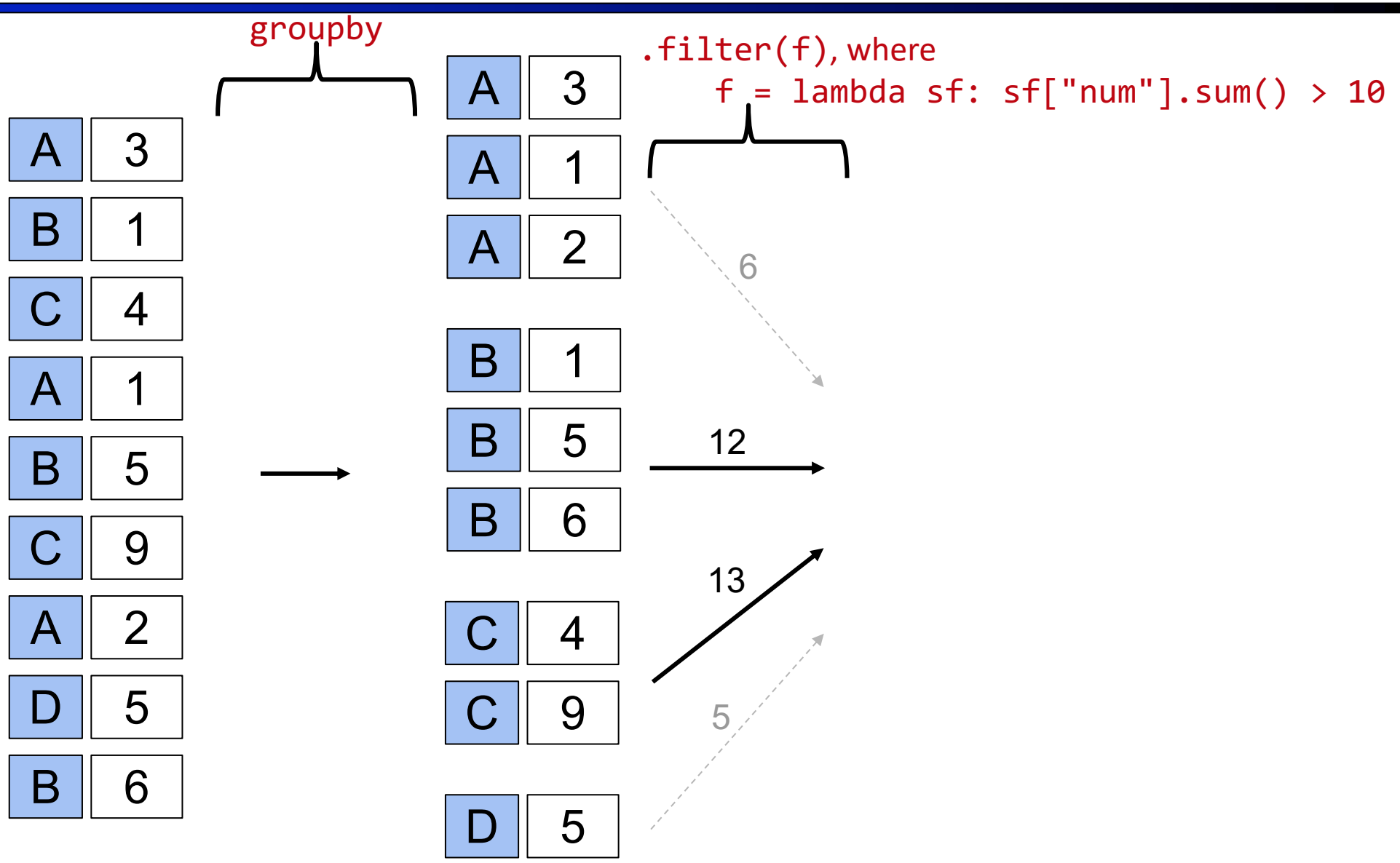
```
best_per_party2 = elections.sort_values("%").drop_duplicates(["Party"], keep="last")
```

Filtering by Group

Another common use for groups is to filter data.

- `groupby.filter` takes an argument `f`.
- `f` is a function that:
  - Takes a DataFrame as input.
  - Returns either True or False.
- For each group g, f is applied to the subframe comprised of the rows from the original dataframe corresponding to that group.

groupby.filter



Filtering Elections Dataset

Let's keep only election year results where the max '%' is less than 45%.

```
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

Groupby and PivotTables

- DataFrameGroupBy Features
- Pivot Tables
- A Quick Look at Joining Tables

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"]).agg(sum).head(6)
```

Example:

		Count
Year	**Sex**	
1880	**F**	90994
	M	110490
1881	**F**	91953
	M	100737
1882	**F**	107847
	M	113686

Note: Resulting DataFrame is multi-indexed. That is, its index has multiple dimensions. Will explore in a later lecture.

Pivot Tables

A more natural approach is to use our brains and create a pivot table.

```
babynames_pivot = babynames.pivot_table(
    index="Year",     # rows (turned into index)
    columns="Sex",    # column values
    values=["Count"], # field(s) to process in each group
    aggfunc=np.sum,   # group operation
)
babynames_pivot.head(6)
```

	Count	
Sex	F	M
Year		
1880	90994	110490
1881	91953	100737
1882	107847	113686
1883	112319	104625
1884	129019	114442
1885	133055	107799

groupby(["Year", "Sex"]) vs. pivot_table

The pivot table more naturally represents our data.

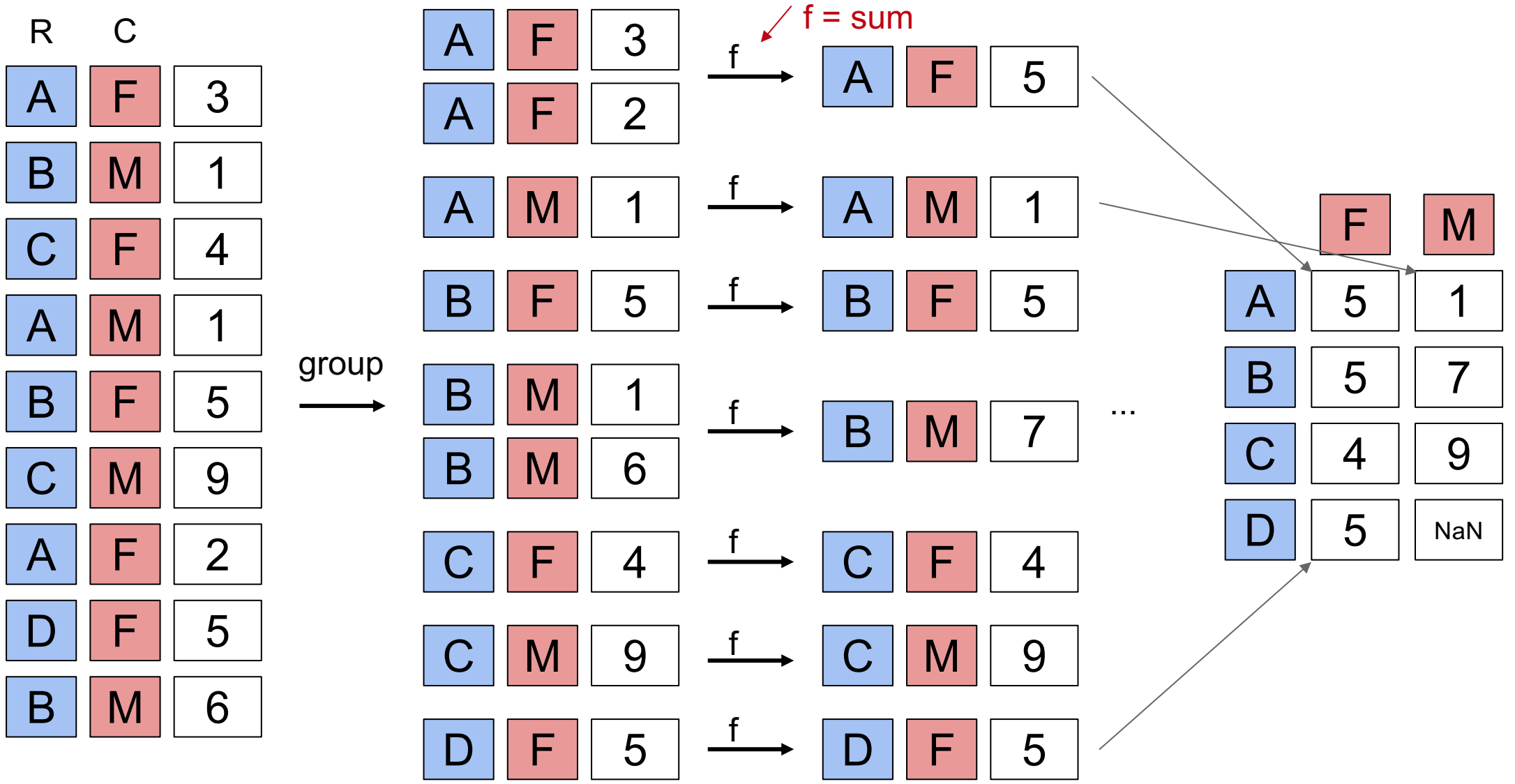
groupby output

		Count
Year	**Sex**	
1880	**F**	90994
	M	110490
1881	**F**	91953
	M	100737
1882	**F**	107847
	M	113686

pivot_table output

	Count	
Sex	**F**	**M**
Year		
1880	90994	110490
1881	91953	100737
1882	107847	113686
1883	112319	104625
1884	129019	114442
1885	133055	107799

Pivot Table Mechanics



Pivot Tables

We can include multiple values in our pivot tables.

```
babynames_pivot = babynames.pivot_table(
    index="Year",     # rows (turned into index)
    columns="Sex",    # column values
    values=["Count", "Name"],
    aggfunc=np.max,   # group operation
)
babynames_pivot.head(6)
```

	Count		Name	
Sex	**F**	**M**	**F**	**M**
Year				
1910	295	237	Yvonne	William
1911	390	214	Zelma	Willis
1912	534	501	Yvonne	Woodrow
1913	584	614	Zelma	Yoshio
1914	773	769	Zelma	Yoshio
1915	998	1033	Zita	Yukio

Introduction to Data Science

Pandas III

Usman Nazir

Joining Tables

Suppose want to know the 2020 popularity of presidential candidate's names.

- Example: Dwight Eisenhower's name Dwight is not popular today, with only 5 babies born with this name in California in 2020.

To solve this problem, we'll have to join tables.

Creating Table 1: Babynames in 2020

Let's set aside names from 2020 first:

```
babynames_2020 = babynames[babynames["Year"] == 2020]
babynames_2020
```

	Name	Sex	Count	Year
0	Olivia	F	17641	2020
1	Emma	F	15656	2020
2	Ava	F	13160	2020
3	Charlotte	F	13065	2020
4	Sophia	F	13036	2020
...	...	...	...	...
31448	Zykell	M	5	2020
31449	Zylus	M	5	2020
31450	Zymari	M	5	2020
31451	Zyn	M	5	2020
31452	Zyran	M	5	2020

31453 rows × 4 columns

Creating Table 2: Presidents with First Names

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

- You'll have a chance to write this code again on lab, so don't worry about the details too much.

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

	Year	Candidate	Party	Popular vote	Result	%	First Name
...	...	...	...	...	...	...	...
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

Joining Our Tables

```
merged = pd.merge(left = elections, right = babynames_2020,
                  left_on = "First Name", right_on = "Name")
```

	Year_x	Candidate	Party	Popular vote	Result	%	First Name	State	Sex	Year_y	Name	Count
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122	Andrew	CA	M	2020	Andrew	867
1	1828	Andrew Jackson	Democratic	642806	win	56.203927	Andrew	CA	M	2020	Andrew	867
2	1832	Andrew Jackson	Democratic	702735	win	54.574789	Andrew	CA	M	2020	Andrew	867
3	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878	John	CA	M	2020	John	617
4	1828	John Quincy Adams	National Republican	500897	loss	43.796073	John	CA	M	2020	John	617

THANK YOU

usman.nazir@lums.edu.pk

usmanweb.github.io

Slides and content credits: Narges Norouzi, Lisa Yan, Josh Hug