THE FOUNDATIONS OF DATA SCIENCE

WEEK 1: INTRODUCTION

Usman Nazir

WHAT IS DATA SCIENCE?

WHAT IS DATA SCIENCE?

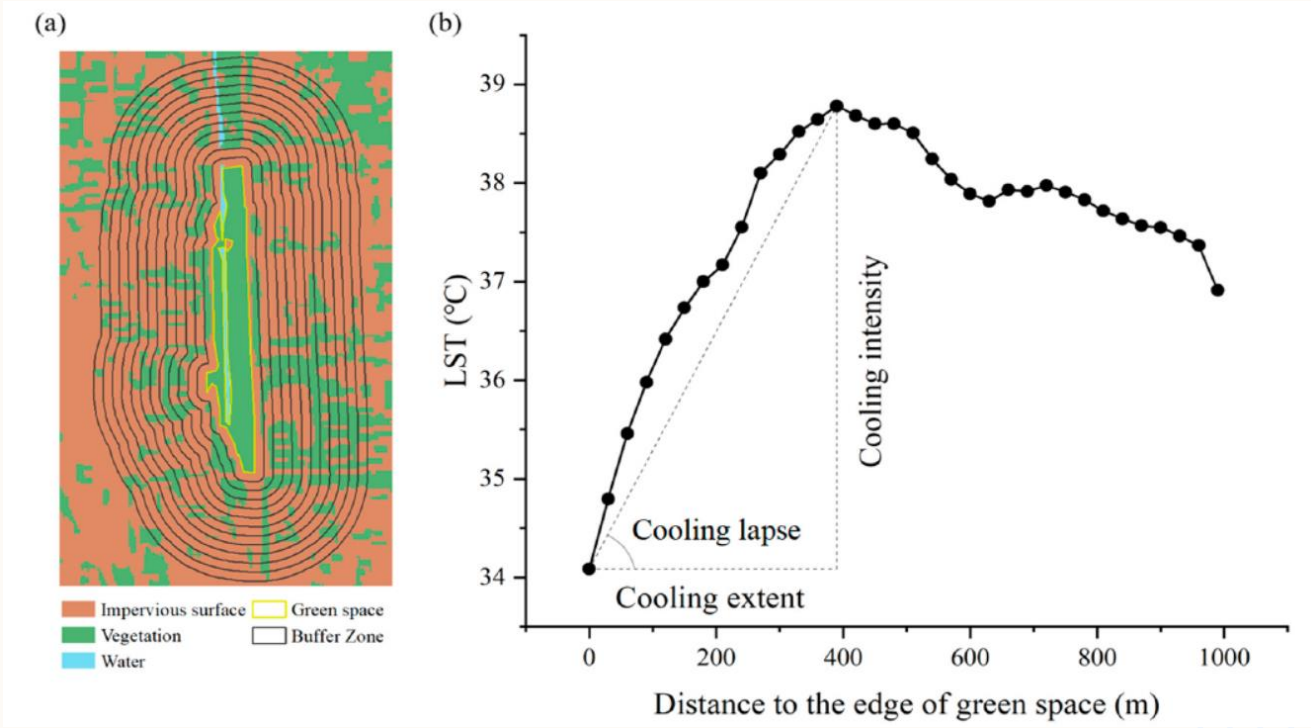
Exploration

- Identifying patterns in data
- Uses Visualizations

WHAT IS DATA SCIENCE?

Exploration

- Identifying patterns in data
- Uses Visualizations



WHAT IS DATA SCIENCE?

Exploration

- Identifying patterns in data
- Uses Visualizations

Inference

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

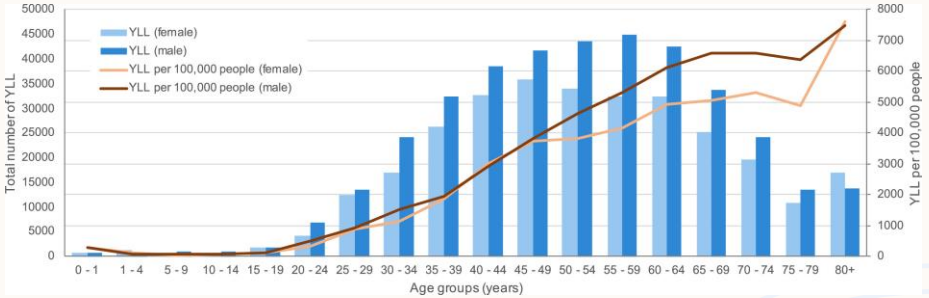
WHAT IS DATA SCIENCE?

Exploration

- Identifying patterns in data
- Uses Visualizations

Inference

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



WHAT IS DATA SCIENCE?

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 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

DATA IS CHANGING THE WORLD

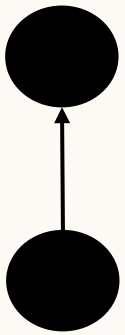
1980s

Hardware Industry

- Sold computers

DATA IS CHANGING THE WORLD

1990s



Software Industry

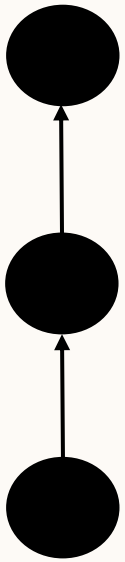
- Sold computer software

1980s

Hardware Industry

- Sold computers

DATA IS CHANGING THE WORLD



2000s

Internet Industry

- Online retailers and services

1990s

Software Industry

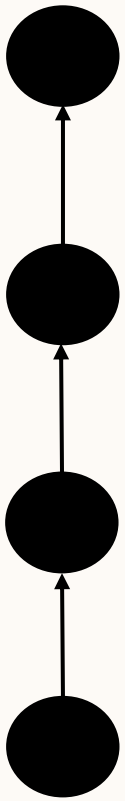
- Sold computer software

1980s

Hardware Industry

- Sold computers

DATA IS CHANGING THE WORLD



2012s

Data Industry

- Collect and sell information

2000s

Internet Industry

- Online retailers and services

1990s

Software Industry

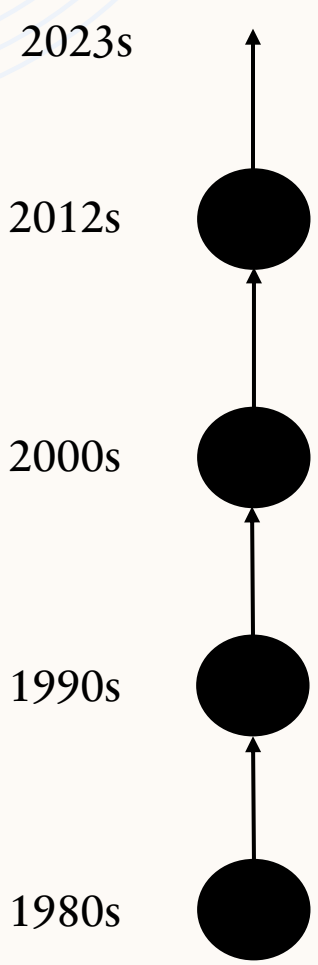
- Sold computer software

1980s

Hardware Industry

- Sold computers

DATA IS CHANGING THE WORLD



?

Data Industry
- Collect and sell information

Internet Industry
- Online retailers and services

Software Industry
- Sold computer software

Hardware Industry
- Sold computers

From Joey Gonzalez.

TENTATIVE LIST OF TOPICS TO BE COVERED

- Pandas and NumPy
- Relational Databases & SQL
- Exploratory Data Analysis
- Regular Expressions
- Visualization
  - matplotlib
  - Seaborn
  - plotly
- Sampling
- Probability and random variables
- Model design and loss formulation
- Linear Regression
- Feature Engineering
- Regularization, Bias-Variance Tradeoff, Cross-Validation
- Gradient Descent
- Data science in the physical world
- Causality
- Logistic Regression
- Clustering
- PCA

RESOURCES

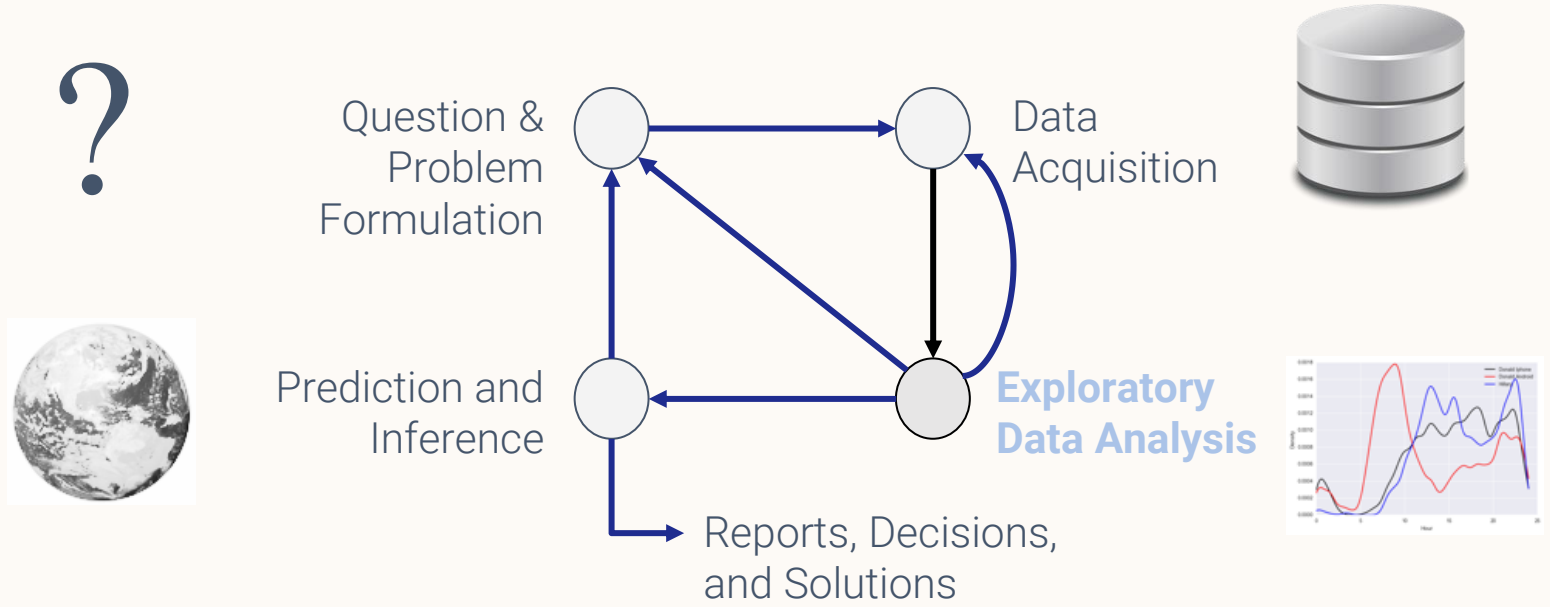
- Pandas API Reference
- The Pandas Cookbook: This provides a nice overview of some of the basic Pandas functions. However, it is slightly out of date.
- Learn Pandas A set of lessons providing an overview of the Pandas library.
- Python for Data Science Another set of notebook demonstrating Pandas functionality.
- Textbook: www.textbook.ds100.org
- Reference book: https://inferentialthinking.com/

THE FOUNDATIONS OF DATA SCIENCE

WEEK 1: CAUSE AND EFFECT

Usman Nazir

DATA SCIENCE LIFECYCLE



Exploring and Cleaning Tabular Data

A LINK

Coffee

Three coffees a day linked to a range of health benefits

Research based on 200 previous studies worldwide says frequent drinkers less likely to get diabetes, heart disease, dementia and some cancers

34 6

Staff and agencies

Wednesday 22 November 2017 19.54 EST

The findings supported other studies showing the health benefits of drinking coffee. Photograph: Wu Hong/EPA

From Guardian UK

A STRONGER LINK

eating and health

Chocolate, Chocolate, It's Good For Your Heart, Study Finds

JUNE 19, 2015 5:03 AM ET

ALLISON AUBREY

From npr.org (report on a study in heart.bmj.com)

OBSERVATION

- **individuals**, study subjects, participants, units
  - *European adults*
- **treatment**
  - *chocolate consumption*
- **outcome**
  - *heart disease*

FIRST QUESTION

Is there **any relation** between chocolate consumption and heart disease?

- **Association**
  - any relation
  - link

SOME DATA

"Among those in the top tier of chocolate consumption, 12 percent developed or died of cardiovascular disease during the study, compared to 17.4 percent of those who didn't eat chocolate."

From Howard LeWine of Harvard Health Blog, reported by npr.org

SOME DATA

"Among those in the top tier of chocolate consumption, 12 percent developed or died of cardiovascular disease during the study, compared to 17.4 percent of those who didn't eat chocolate."

From Howard LeWine of Harvard Health Blog, reported by npr.org

Does this point to an association?

NEXT QUESTION

Does chocolate consumption lead to a reduction in heart disease?

- **Causality**

NEXT QUESTION

Does chocolate consumption lead to a reduction in heart disease?

- **Causality**

This question is often harder to answer.

"[The study] doesn't prove a **cause-and-effect** relationship between chocolate and reduced risk of heart disease and stroke."

From JoAnn Manson, chief of Preventive Medicine at Brigham and Women's Hospital, Boston

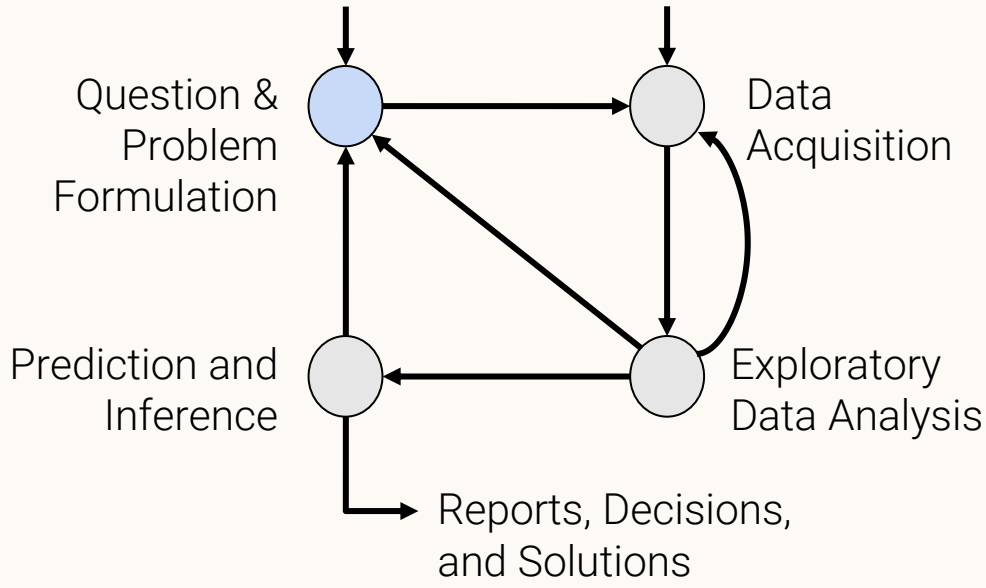
THE FOUNDATIONS OF DATA SCIENCE

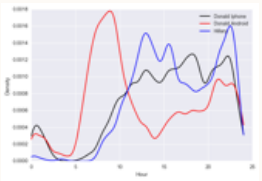
WEEK 1: Data Science Life Cycle

Usman Nazir

DATA SCIENCE LIFE CYCLE

?

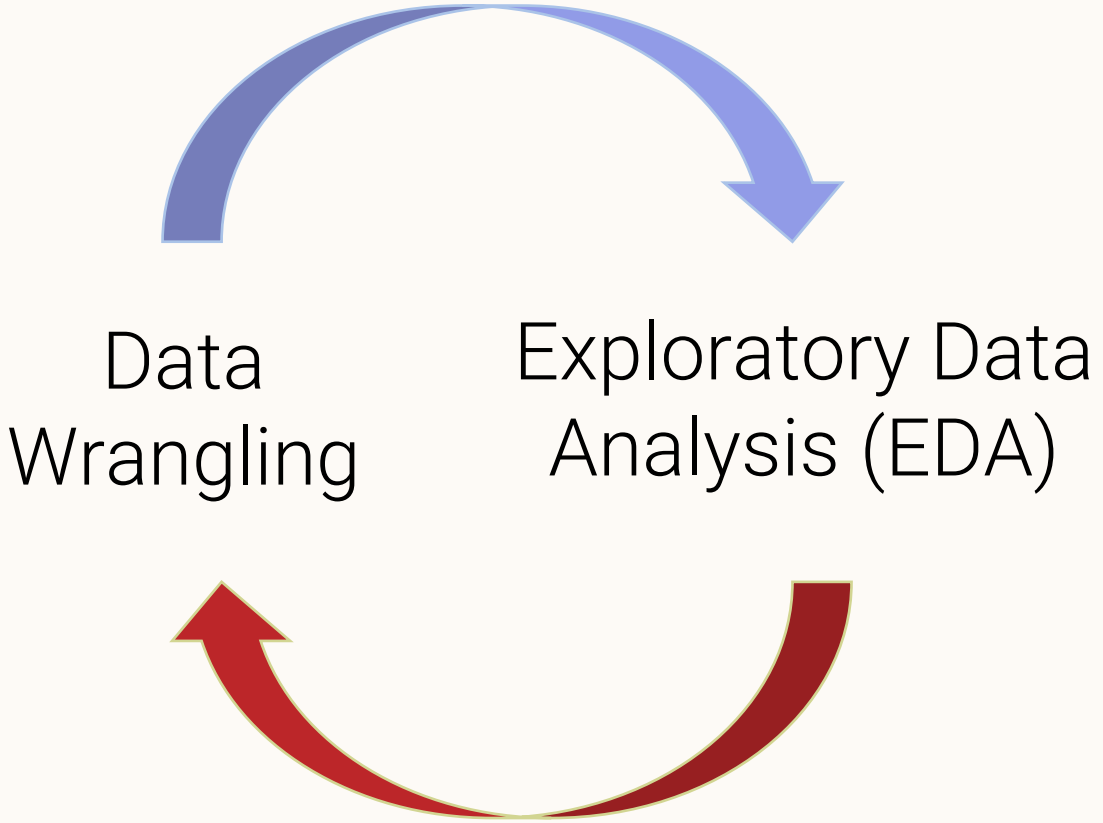




CONGRATULATIONS!

You have collected or have been given a box of data. What will you do next?

THE INFINITE LOOP OF DATA SCIENCE



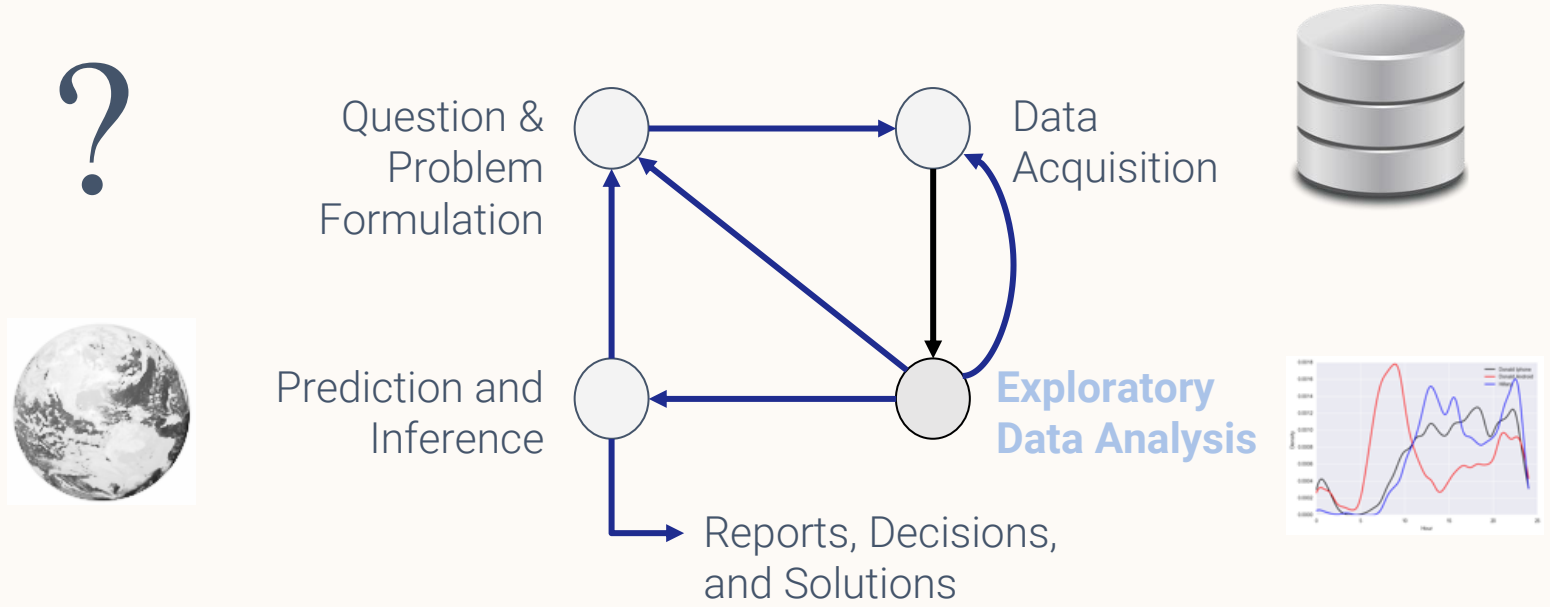
The Python Data Analysis Library

THE FOUNDATIONS OF DATA SCIENCE

WEEK 1: Pandas I

Usman Nazir

FROM DATA SCIENCE TO PANDAS



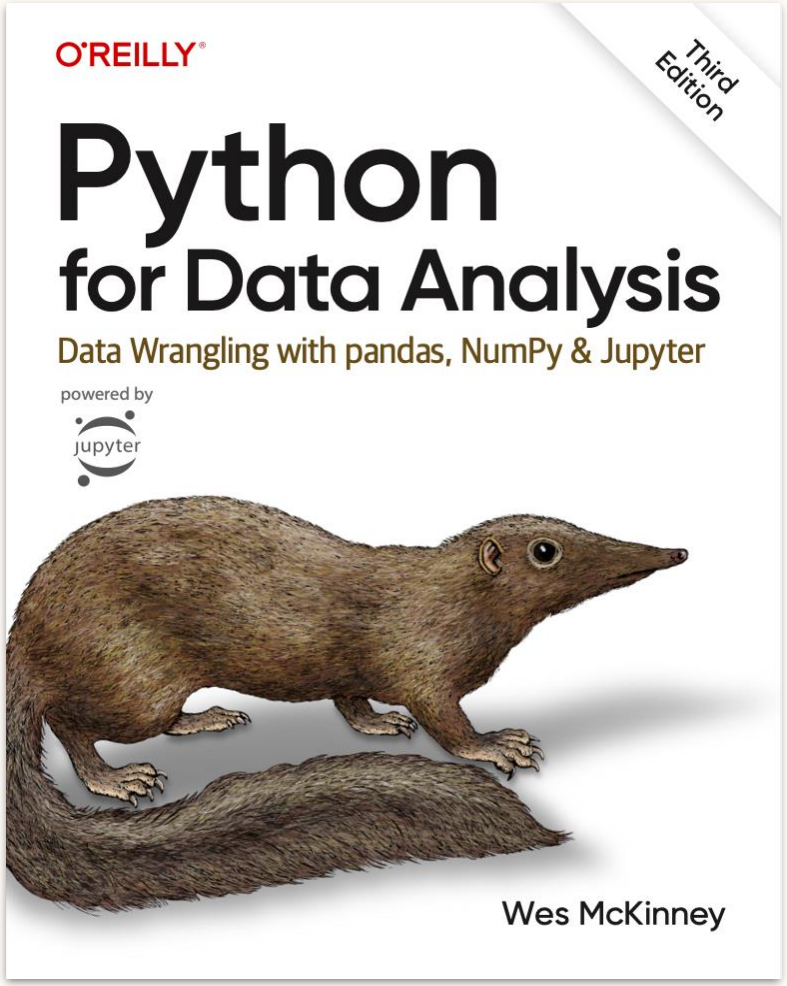
Exploring and Cleaning Tabular Data

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

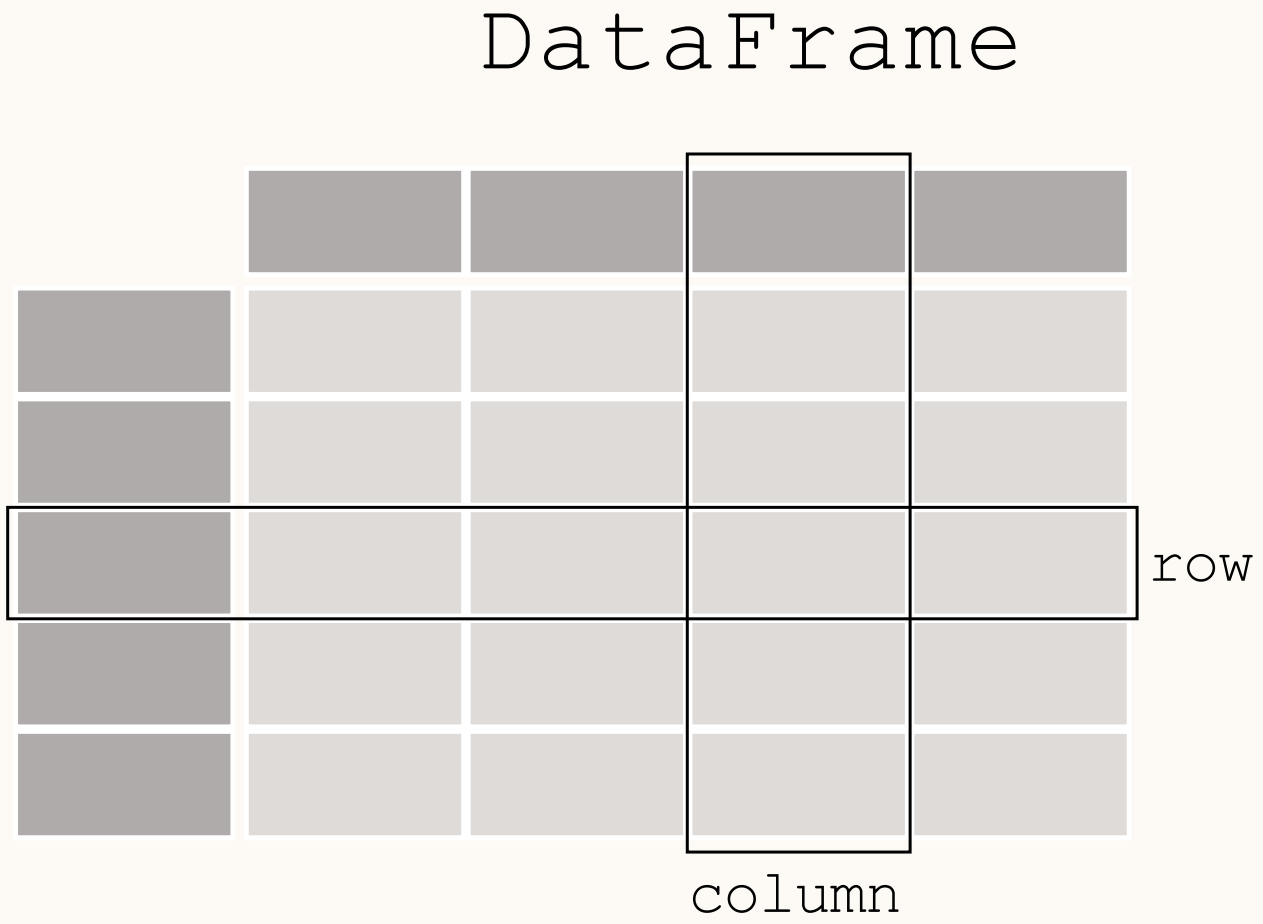
	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

Series

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

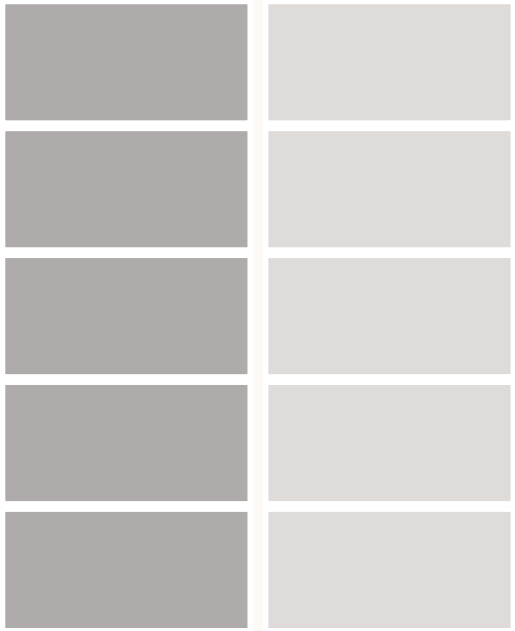
Index

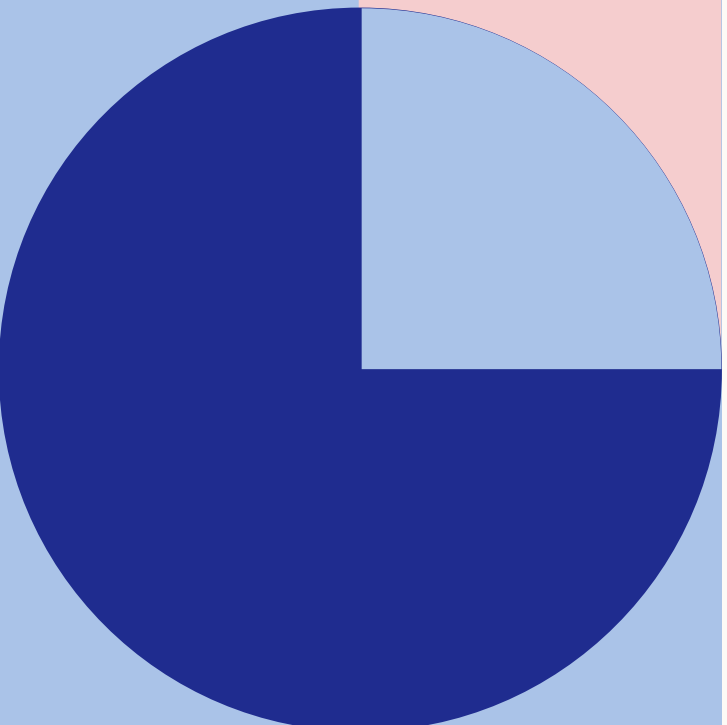
PANDAS DATA TABLE REPRESENTATION



EACH COLUMN IN A DATAFRAME IS A SERIES

Series





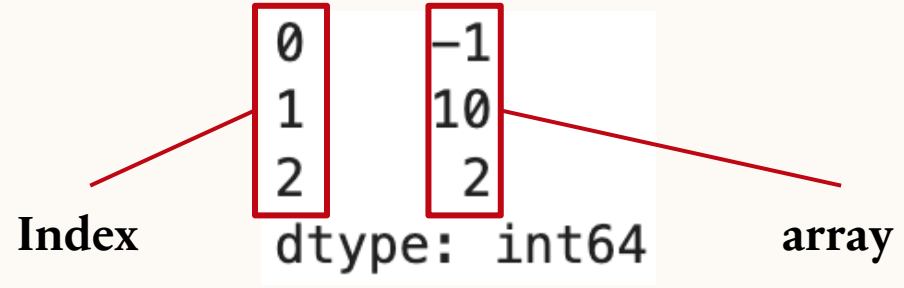
SERIES, DATAFRAMES, AND INDICES

- **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")
```

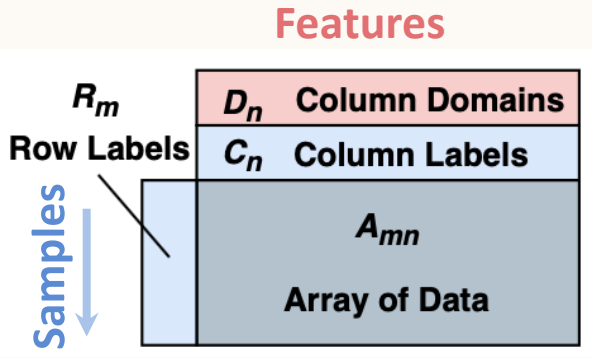
	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832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

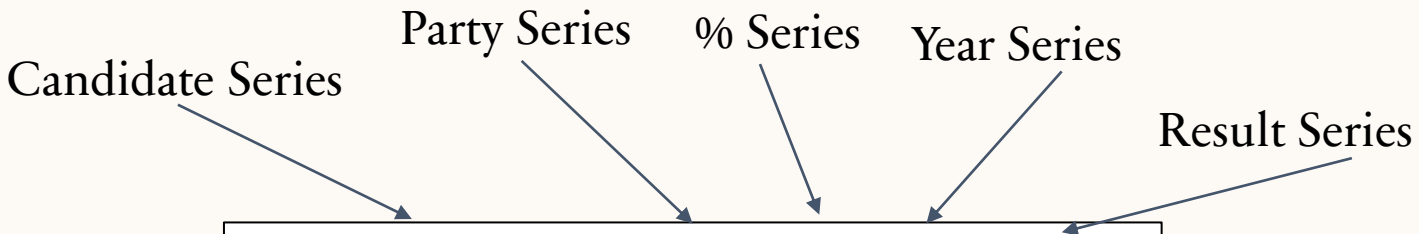
```
s_a.to_frame()
```

	0
r1	a1
r2	a2
r3	a3

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 Index.

- 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

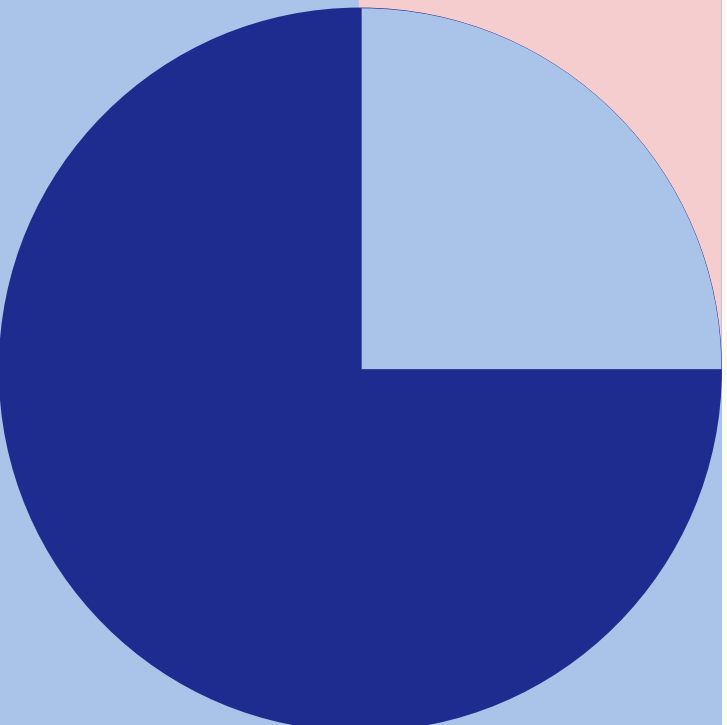
182 rows × 6 columns

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



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 column labels.
  - A single column label.

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

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 loc.

- 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).

[ ]

[ ] only takes one argument, which may be:

- **A slice of row numbers.**
- A list of column labels.
- A single column label.

```
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 column labels.**
- A single column label.

```
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 column labels.
- **A single column label.**

```
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.

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')
```

THE FOUNDATIONS OF DATA SCIENCE

WEEK 1: INTRODUCTION

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 Python Data Analysis Library

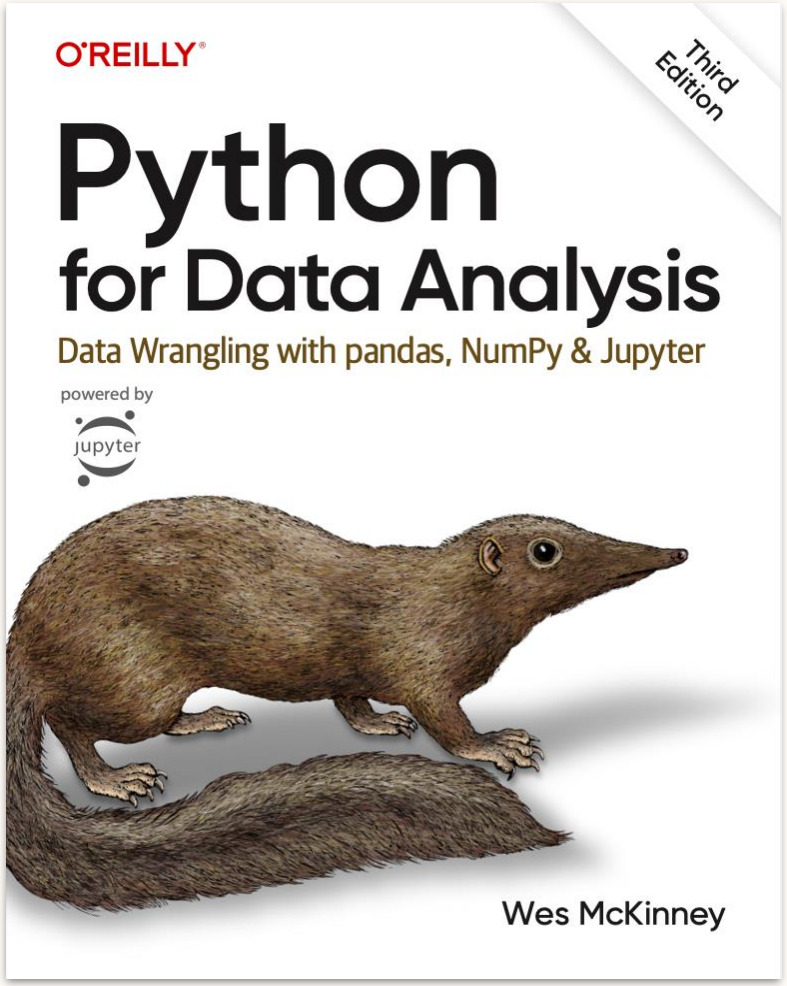
THE FOUNDATIONS OF DATA SCIENCE

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

```
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.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
```

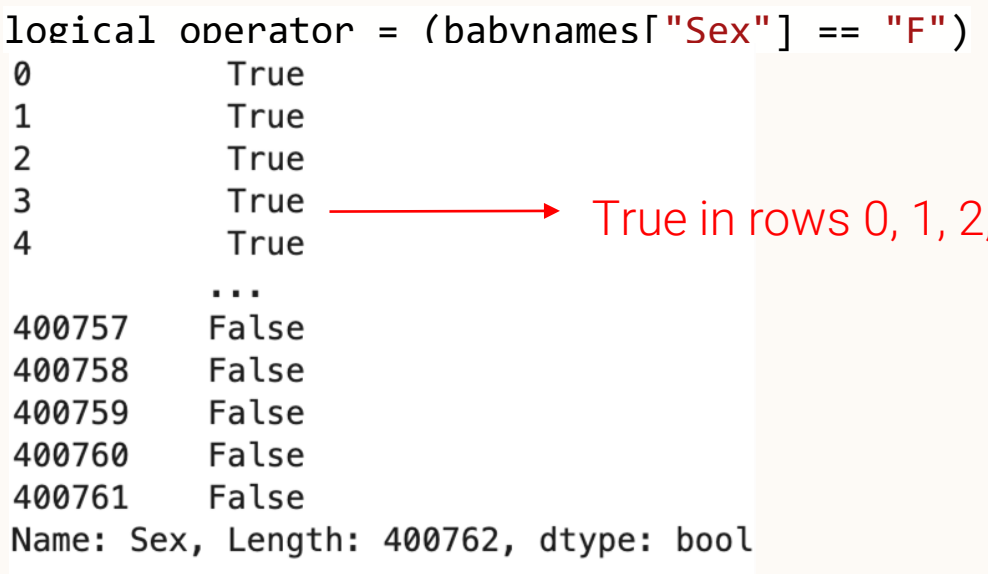
True in rows 0, 1, 2, …

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".



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`.

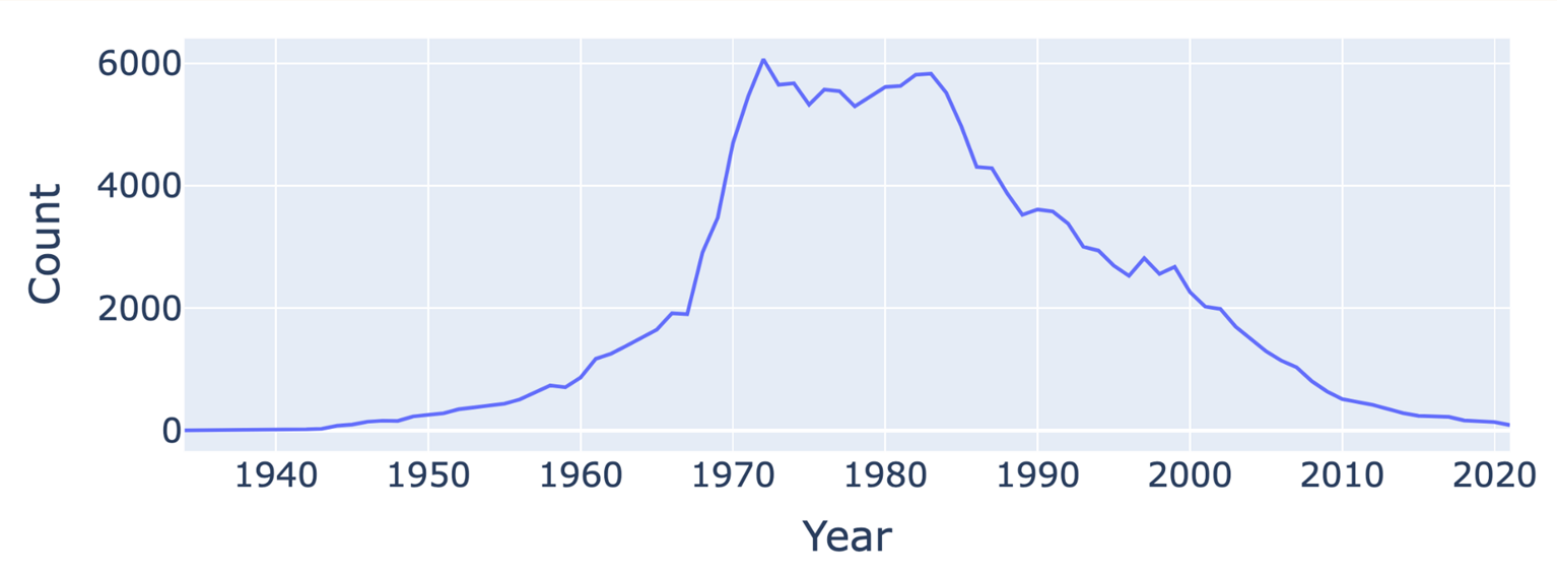
	State	Sex	Year	Name	Count
0	CA	F	1910	Mary	295
1	CA	F	1910	Helen	239
2	CA	F	1910	Dorothy	220
3	CA	F	1910	Margaret	163
4	CA	F	1910	Frances	134
5	CA	F	1910	Ruth	128
6	CA	F	1910	Evelyn	126

	Name	Count
0	Mary	295
233	Mary	390
484	Mary	534

QUESTION # 2

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

Hint



THANK YOU

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