

Intro to pandas, Part 1

UC Berkeley Data 100 Summer 2019
Sam Lau

Learning goals:

- Introduce the DataFrame and the Series
- Learn how to slice a DF using indexing
- Learn commonly used pandas methods

(Slides adapted from Josh Hug and John DeNero)

Announcements

There is a live lecture Piazza thread: Leo will post soon.

Starting Wed, lecture is moved to North Gate Hall room 105!

- Everyone also gets attendance today; GForm tomorrow

Exam Conflict form link changed: <http://bit.ly/su19-alt-final>

- Due Friday at 11:59pm
- DSP exams will have a separate form, will send later.

Announcements

Office hours scheduled today for HW1!

- 11am-12pm, 2-4pm in 355 Evans
- Room will change after this week

Small group tutoring is starting next week; more info soon

I will try to do a better job of asking for names today. Also please add your preferred pronouns.

Pandas Data Structures: Data Frames, Series, and Indices (Reading: Chapter 3)

Will move fast today; use lab time to let material sink in.

Pandas Data Structures

There are three fundamental data structures in pandas:

- Data Frame: 2D data tabular data.
- Series: 1D data. I usually think of it as columnar data.
- Index: A sequence of row 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

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 Series Party Series % Series Year Series Result Series



	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

Non-native English speaker note: The plural of “series” is “series”. Sorry.

Indices Are Not Necessarily Row Numbers

Indices (a.k.a. row labels) can also:

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

	Motto	Translation	Language	Date Adopted
State				
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

Year	Candidate	Party	%	Result
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

Column Names Must Be Unique!

Column names in Pandas are always unique!

- Example: Can't have two columns named "Candidate".

	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

Hands On Exercise

Let's experiment with reading csv files and playing around with indices.

See `lec02-live.ipynb`. (Link on course website)

(demo)

Indexing with The `[]` Operator

Indexing by Column Names Using [] Operator

Given a dataframe, it is common to extract a Series or a collection of Series. This process is also known as “Column Selection” or sometimes “indexing by column”.

- Column name argument to [] yields Series.
- List argument to [] yields a Data Frame.

Indexing by Column Names Using [] Operator

Column name argument to [] yields Series.

```
elections["Candidate"].head(6)
```

```
0      Reagan
1      Carter
2    Anderson
3      Reagan
4    Mondale
5       Bush
Name: Candidate, dtype: object
```

Indexing by Column Names Using [] Operator

Column name argument to [] yields Series.

List argument to [] yields a Data Frame.

```
elections[["Candidate", "Party"]].head(6)
```

```
elections["Candidate"].head(6)
```

```
0    Reagan
1    Carter
2  Anderson
3    Reagan
4  Mondale
5     Bush
Name: Candidate, dtype: object
```

	Candidate	Party
0	Reagan	Republican
1	Carter	Democratic
2	Anderson	Independent
3	Reagan	Republican
4	Mondale	Democratic
5	Bush	Republican

Indexing by Row Slices Using [] Operator


We can also index by row numbers using the [] operator.

- Numeric slice argument to [] yields rows.
- Example: [0:3] yields rows 0 to 2.

```
elections[0:3]
```

	Candidate	Party	%	Year	Result
0	Reagan	Republican	50.7	1980	win
1	Carter	Democratic	41.0	1980	loss
2	Anderson	Independent	6.6	1980	loss

[] Summary


Name →  → Series

Single Column Selection

```
elections["Candidate"].head(6)
```

0	Reagan
1	Carter
2	Anderson
3	Reagan
4	Mondale
5	Bush


Name: Candidate, dtype: object

List →  → DataFrame

Multiple Column Selection

```
elections[["Candidate"]].head(6)
```

	Candidate
0	Reagan
1	Carter
2	Anderson
3	Reagan
4	Mondale
5	Bush

Numeric Slice →  → DataFrame

(Multiple) Row Selection

```
elections[0:3]
```

	Candidate	Party	%	Year	Result
0	Reagan	Republican	50.7	1980	win
1	Carter	Democratic	41.0	1980	loss
2	Anderson	Independent	6.6	1980	loss

Note: Row Selection Requires Slicing!!

`elections[0]` will not work unless the elections data frame has a column whose name is the numeric zero.

- Note: It is actually possible for columns to have names that are non-String types, e.g. numeric, datetime etc.

Question

```
weird = pd.DataFrame({1:["topdog","botdog"], "1":["topcat","botcat"]})  
weird
```

	1	1
0	topdog	topcat
1	botdog	botcat

Name → **[]** → Series
Single Column Selection

List → **[]** → DataFrame
Multiple Column Selection

Numeric Slice → **[]** → DataFrame
(Multiple) Row Selection

Try to predict the output of the following:

- `weird[1]`
- `weird["1"]`
- `weird[1:]`

(demo)

Boolean Array Selection

Boolean Array Input

Yet another input type supported by [] is the boolean array.

```
elections[[False, False, False, False, False,  
           False, False, True, False, False,  
           True, False, False, False, True,  
           False, False, False, False, False,  
           False, False, True]]
```

	Candidate	Party	%	Year	Result
7	Clinton	Democratic	43.0	1992	win
10	Clinton	Democratic	49.2	1996	win
14	Bush	Republican	47.9	2000	win
22	Trump	Republican	46.1	2016	win



Boolean Array Input

Yet another input type supported by [] is the boolean array. Useful because boolean arrays can be generated by using logical operators on Series.

Length 23 Series where every entry is “Republican”, “Democrat” or “Independent.”

Length 23 Series where every entry is either “True” or “False”, where “True” occurs for every independent candidate.

```
elections[elections['Party'] == 'Independent']
```

	Candidate	Party	%	Year	Result
2	Anderson	Independent	6.6	1980	loss
9	Perot	Independent	18.9	1992	loss
12	Perot	Independent	8.4	1996	loss

Boolean Array Input

Boolean Series can be combined using the & operator, allowing filtering of results by multiple criteria.

```
elections[(elections['Result'] == 'win')  
          & (elections['%'] < 50)]
```

	Candidate	Party	%	Year	Result
7	Clinton	Democratic	43.0	1992	win
10	Clinton	Democratic	49.2	1996	win
14	Bush	Republican	47.9	2000	win
22	Trump	Republican	46.1	2016	win

(demo)

Indexing with loc and iloc

.loc and .iloc

.loc and .iloc are alternate ways to index into a DataFrame.

- They take a lot of getting used to! Documentation and ideas behind them are quite complex.
- I'll go over common usages (see docs for weirder ones).

Documentation:

- loc: <https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.loc.html>
- iloc: <https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.iloc.html>
- More general docs on indexing and selecting: [Link](#)

.loc

.loc does two things:

- Access values by labels.
- Access values using a boolean array (a la Boolean Array Selection).

.loc with Lists

The most basic use of loc is to provide a list of row and column labels, which returns a DataFrame.

```
elections.loc[[0, 1, 2, 3, 4], ['Candidate', 'Party', 'Year']]
```

	Candidate	Party	Year
0	Reagan	Republican	1980
1	Carter	Democratic	1980
2	Anderson	Independent	1980
3	Reagan	Republican	1984
4	Mondale	Democratic	1984

.loc with Slices

.loc is also commonly used with slices.

- Slicing works with all label types, not just numeric labels.
- Slices with loc are **inclusive**, not **exclusive**.

```
elections.loc[0:4, 'Candidate':'Year']
```

	Candidate	Party	Year
0	Reagan	Republican	1980
1	Carter	Democratic	1980
2	Anderson	Independent	1980
3	Reagan	Republican	1984
4	Mondale	Democratic	1984

.loc with Single Values for Column Label

If we provide only a single label as column argument, we get a Series.

```
elections.loc[0:4, 'Candidate']
```

```
0      Reagan
```

```
1      Carter
```

```
2    Anderson
```

```
3      Reagan
```

```
4    Mondale
```

```
Name: Candidate, dtype: object
```

.loc with Single Values for Column Label

As before with the `[]` operator, if we provide a list of only one label as an argument, we get back a dataframe.

```
elections.loc[0:4, 'Candidate']
```

```
0      Reagan
1      Carter
2    Anderson
3      Reagan
4    Mondale
Name: Candidate, dtype: object
```

```
elections.loc[0:4, ['Candidate']]
```

	Candidate
0	Reagan
1	Carter
2	Anderson
3	Reagan
4	Mondale

.loc with Single Values for Row Label

If we provide only a single row label, we get a Series.

- Series made up of the values from the requested row, not column
- Index is the names of the columns from the data frame.
- Putting the single row label in a list yields a dataframe version.

```
elections.loc[0, 'Candidate':'Year']
```

Candidate	Reagan
Party	Republican
%	50.7
Year	1980

Name: 0, dtype: object

```
elections.loc[[0], 'Candidate':'Year']
```

	Candidate	Party	%	Year
0	Reagan	Republican	50.7	1980

.loc Supports Boolean Arrays

.loc supports Boolean Arrays exactly as you'd expect.

```
elections.loc[(elections['Result'] == 'win') & (elections['%'] < 50), 'Candidate':'%']
```

	Candidate	Party	%
7	Clinton	Democratic	43.0
10	Clinton	Democratic	49.2
14	Bush	Republican	47.9
22	Trump	Republican	46.1

.iloc: Selection by Position

In contrast to `loc`, `iloc` doesn't think about labels at all. Instead, it returns the items that appear in the numerical positions specified.

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

	Candidate	Party	%
0	Reagan	Republican	50.7
1	Carter	Democratic	41.0
2	Anderson	Independent	6.6

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

	Motto	Translation	Language
State			
Alabama	Audemus jura nostra defendere	We dare defend our rights!	Latin
Alaska	North to the future	—	English
Arizona	Ditat Deus	God enriches	Latin

Advantages of `loc`:

- Harder to make mistakes.
- Easier to read code.
- Not vulnerable to changes to the ordering of rows/cols in raw data files.

(demo)

Nonetheless, `iloc` can be more convenient. Use `iloc` judiciously.

Slicing Connections

```
elections["Candidate"].head(6)
```

```
0    Reagan
1    Carter
2    Anderson
3    Reagan
4    Mondale
5     Bush
Name: Candidate, dtype: object
```

```
elections.loc[:, 'Candidate'].head(6)
```

```
elections[["Candidate"]].head(6)
```

	Candidate
0	Reagan
1	Carter
2	Anderson
3	Reagan
4	Mondale
5	Bush

```
elections.loc[:, ['Candidate']].head(6)
```

```
elections[0:3]
```

	Candidate	Party	%	Year	Result
0	Reagan	Republican	50.7	1980	win
1	Carter	Democratic	41.0	1980	loss
2	Anderson	Independent	6.6	1980	loss

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

5 min break

Handy Properties and Utility Functions for Series and DataFrames

head, size, shape, and describe

head: Displays only the top few rows.

size: Gives the total number of data points.

shape: Gives the size of the data in rows and columns.

describe: Provides a summary of the data.

index and columns

`index`: Returns the index (a.k.a. row labels).

`columns`: Returns the labels for the columns.

The sort_values Method

One incredibly useful method for DataFrames is `sort_values`, which creates a copy of a DataFrame sorted by a specific column.

```
elections.sort_values('%', ascending=False)
```

	Candidate	Party	%	Year	Result
3	Reagan	Republican	58.8	1984	win
5	Bush	Republican	53.4	1988	win
17	Obama	Democratic	52.9	2008	win
19	Obama	Democratic	51.1	2012	win
0	Reagan	Republican	50.7	1980	win

The sort_values Method

We can also use `sort_values` on a Series, which returns a copy with the values in order.

```
mottos['Language'].sort_values().head(5)
```

State	
Washington	Chinook Jargon
Wyoming	English
New Jersey	English
New Hampshire	English
Nevada	English

Name: Language, dtype: object

The value_counts Method

Series also has the function `value_counts`, which creates a new Series showing the counts of every value.

```
elections['Party'].value_counts()
```

Democratic	10
Republican	10
Independent	3
Name: Party, dtype: int64	

The unique Method

Another handy method for Series is `unique`, which returns all unique values as an array.

```
mottos['Language'].unique()  
array(['Latin', 'English', 'Greek', 'Hawaiian', 'Italian', 'French',  
      'Spanish', 'Chinook Jargon'], dtype=object)
```

Baby Names Case Study Q1

Baby Names

Let's try solving a real world problem using the baby names dataset: **What was the most popular name in California last year (2019)?**

Along the way, we'll see some examples of what it's like to deal with real data, and will also explore some fancy iPython features.

(demo)

Summary

- pandas data structures:
 - DataFrames are 2D tables of data.
 - Series are 1D array-like columns of data.
- Slicing:
 - .loc for slicing by label, .iloc by index
 - Boolean slicing to slice by condition
- Useful methods:
 - .read_csv, .head, .shape, .describe, .sort_values, .value_counts, .unique.