Pandas

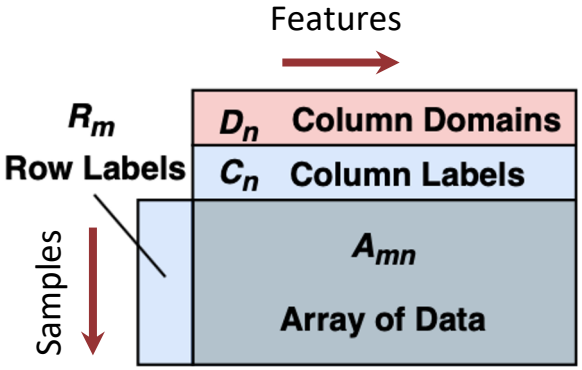
Introduction to Pandas syntax and operators

Recap

- Introduce Pandas, with emphasis on:
  - Create a dataframe
    - From scratch
    - Import from file
  - How to index into a dataframe
  - How to read files to create these structures
  - Basic operations on these structures.
    - Indexing
    - Filtering
    - Insert/ Delete
    - Sorting
    - Pivot table
    - Group by
    - Join

Dataframe

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

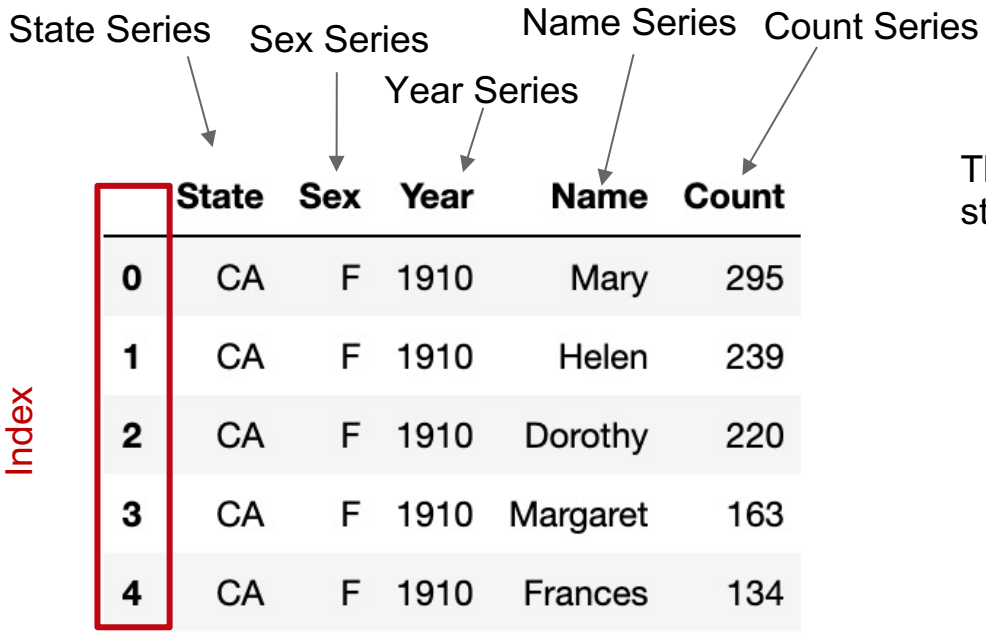


A generic DataFrame
(from https://arxiv.org/abs/2001.00888)

	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

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.



	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

There are three fundamental data structures in pandas:

- Data Frame: 2D data tabular data.
- Series: 1D data. Usually treated as column vector.
- Index: A sequence of row labels

Indices Are Not Necessarily Row Numbers

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

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

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 Are Usually Unique!

Column names in Pandas are almost always unique!

- Example: Really shouldn'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

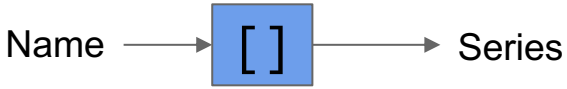
Indexing

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

	1	1
0	topdog	topcat
1	botdog	botcat

Try to predict the output of the following:

- `weird[1]`
- `weird[“1”]`
- `weird[1:]`



Single Column Selection

Multiple Column Selection

(Multiple) Row Selection

Note: Row Selection Requires Slicing!!

weird[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.

Indexing with Loc and iloc

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

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:
  - https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html

Loc

Loc does two things:

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

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

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

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

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

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

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

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.

- Such a series represents a ROW not a column!
- The index of this Series 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

iloc: Integer-Based Indexing for 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]
```

State	Motto	Translation	Language
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.

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

Question Challenge

Which of the following pandas statements returns a DataFrame of the first 3 Candidate names and the Year only for candidates that won with more than 50% of the vote.

A. elections.iloc[[0, 3, 5], [0, 3]]

B. elections.loc[[0, 3, 5], ["Candidate":"Year"]

C. elections.loc[elections["%"] > 50, ["Candidate", "Year"]].head(3)

D. elections.loc[elections["%"] > 50, ["Candidate", "Year"]].iloc[0:2, :]

	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
3	Reagan	Republican	58.8	1984	win
4	Mondale	Democratic	37.6	1984	loss
5	Bush	Republican	53.4	1988	win
6	Dukakis	Democratic	45.6	1988	loss

	Candidate	Year
0	Reagan	1980
3	Reagan	1984
5	Bush	1988

Question Challenge

Which of the following pandas statements returns a DataFrame of the first 3 Candidate names and the Year only for candidates that won with more than 50% of the vote.

A. elections.iloc[[0, 3, 5], [0, 3]]

B. elections.loc[[0, 3, 5], ["Candidate":"Year"]

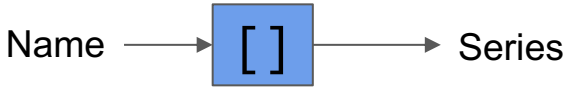
C. elections.loc[elections["%"] > 50, ["Candidate", "Year"]].head(3)

D. elections.loc[elections["%"] > 50, ["Candidate", "Year"]].iloc[0:2, :]

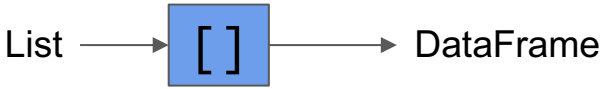
	Candidate	Party	%	Year	Result
0	Reagan	Republican	50.7	1980	win
1	Carter	Democratic	41.0	1980	loss
2	Anderson	Independent	6.6	1980	loss
3	Reagan	Republican	58.8	1984	win
4	Mondale	Democratic	37.6	1984	loss
5	Bush	Republican	53.4	1988	win
6	Dukakis	Democratic	45.6	1988	loss

	Candidate	Year
0	Reagan	1980
3	Reagan	1984
5	Bush	1988

Indexing Summary: [], loc, iloc



Single Column Selection



Multiple Column Selection

(Multiple) Row Selection

loc: Labels

- Strings, integers – row/ column labels
- Lists – returns dataframe
- Slices of labels: end-point inclusive
- Booleans: “mask”

iloc: integer/ positional

- Count from 0
- End-point exclusive
- Use carefully (you might miscount)

Sample

If you want a DataFrame consisting of 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 our selection operators [], loc, iloc.

```
elections.sample(10)
```

	Candidate	Party	%	Year	Result
15	Kerry	Democratic	48.3	2004	loss
16	Bush	Republican	50.7	2004	win
22	Trump	Republican	46.1	2016	win
9	Perot	Independent	18.9	1992	loss
21	Clinton	Democratic	48.2	2016	loss
11	Dole	Republican	40.7	1996	loss
20	Romney	Republican	47.2	2012	loss
14	Bush	Republican	47.9	2000	win
8	Bush	Republican	37.4	1992	loss
1	Carter	Democratic	41.0	1980	loss

```
elections.query("Year < 1992").sample(4, replace=True)
```

	Candidate	Party	%	Year	Result
1	Carter	Democratic	41.0	1980	loss
4	Mondale	Democratic	37.6	1984	loss
6	Dukakis	Democratic	45.6	1988	loss
1	Carter	Democratic	41.0	1980	loss

isin

The `isin` function makes it more convenient to find rows that match one of many possible values.

Example: Suppose we want to find “Republican” or “Democratic” candidates. Could use the | operator (| means or), or we can use `isin`.

- Ugly:
```
df[(df["Party"] == "Democratic") | (df["Party"] == "Republican")]
```
- Better:
```
df[df["Party"].isin(["Republican", "Democratic"])]
```

Handy functions for dataframes

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.

sort_values

value_counts

unique

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 use also `sort_values` on a Series, which returns a copy with 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 Exploration

Goal 1: Find the most popular name in California in 2018
Goal 2: Find all names that start with J.
Goal 3: Sort names by length.
Goal 4: Find the name whose popularity has changed the most.
Goal 5: Count the number of female and male babies born in each year.