

CME538 Introduction to Data Science

Week 2 | Lecture 1 (2.1)

Pandas I.

Pandas I

- What is Pandas?

Pandas I

- What is Pandas?



These are Pandas.

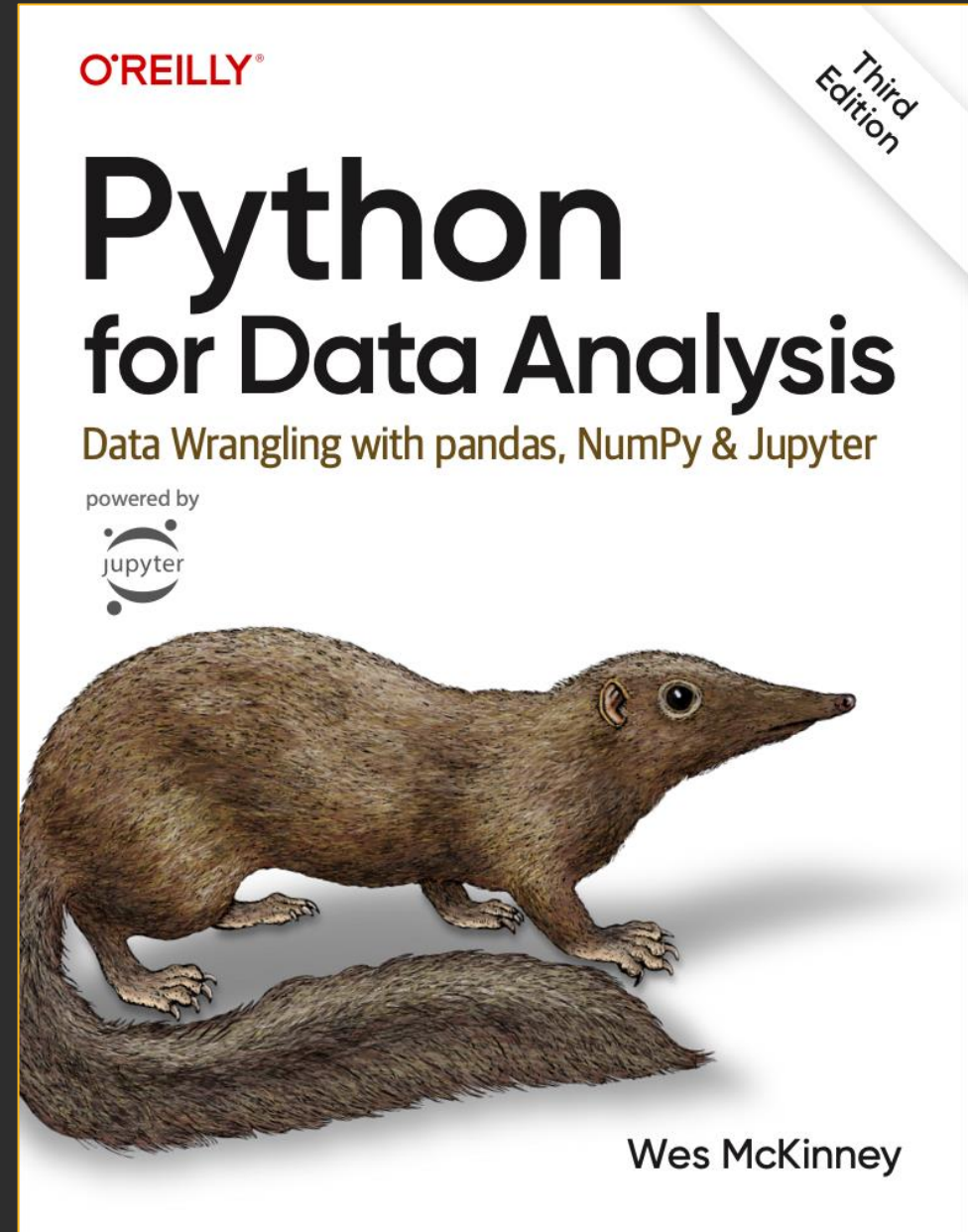
Pandas I

- What else is Pandas?
- A fast, powerful, flexible and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language.
- Pandas stands for "Python Data Analysis Library".



Pandas |

- You have free access to a fantastic book by the creator of Pandas!
- Click [here](#).



Pandas I

- DataFrames
- Pandas Data Structures
- Importing Data
- Indexing
- Utility Methods



Pandas |

- **DataFrames**
- Pandas Data Structures
- Importing Data
- Indexing
- Utility Methods



DataFrames

- A DataFrame is a 2-dimensional labeled data structure with columns of potentially different types.
 - Matrix?
- You can think of it like a spreadsheet or SQL table.
- DataFrames were first introduced in the R Programming Language and are generally the most used Pandas object.
- Pandas is the most popular Python package for working with DataFrames.
 - Dask, Polar, Vaex.

index labels

column names

	Mountain	Height (m)	Range	Coordinates	Parent mountain	First ascent	Ascents bef. 2004	Failed attempts bef. 2004
0	Mount Everest / Sagarmatha / Chomolungma	8848	Mahalangur Himalaya	27°59'17"N 86°55'31"E	NaN	1953	>>145	121.0
1	K2 / Qogir / Godwin Austen	8611	Baltoro Karakoram	35°52'53"N 76°30'48"E	Mount Everest	1954	45	44.0
2	Kangchenjunga	8586	Kangchenjunga Himalaya	27°42'12"N 88°08'51"E	Mount Everest	1955	38	24.0
3	Lhotse	8516	Mahalangur Himalaya	27°57'42"N 86°55'59"E	Mount Everest	1956	26	26.0
4	Makalu	8485	Mahalangur Himalaya	27°53'23"N 87°05'20"E	Mount Everest	1955	45	52.0
5	Cho Oyu	8188	Mahalangur Himalaya	28°05'39"N 86°39'39"E	Mount Everest	1954	79	28.0
6	Dhaulagiri I	8167	Dhaulagiri Himalaya	28°41'48"N 83°29'35"E	K2	1960	51	39.0
7	Manaslu	8163	Manaslu Himalaya	28°33'00"N 84°33'35"E	Cho Oyu	1956	49	45.0
8	Nanga Parbat	8126	Nanga Parbat Himalaya	35°14'14"N 74°35'21"E	Dhaulagiri	1953	52	67.0
9	Annapurna I	8091	Annapurna Himalaya	28°35'44"N 83°49'13"E	Cho Oyu	1950	36	47.0

data

Pandas I

- DataFrames
- **Pandas Data Structures**
- Importing Data
- Indexing
- Utility Methods



Data Structures

- There are three fundamental data structures in pandas:
 - Data Frame: 2D data tabular data.
 - Series: 1D data. We can think of Series as columnar data.
 - Index: A sequence of row labels.

Dataset

U.S. Election Results.

Axis: 1 →

Columns

Index

Axis: 0 ↓

	Candidate	Party	Percent	Year	Outcome
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

Series

Data Structures

- There are three fundamental data structures in pandas:
 - **Data Frame: 2D data tabular data.**
 - Series: 1D data. We can think of Series as columnar data.
 - Index: A sequence of row labels.

DataFrames

Axis: 1 →

Columns

Index

Axis: 0 ↓

	Candidate	Party	Percent	Year	Outcome
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

Series

Data Structures

- There are three fundamental data structures in pandas:
 - **Data Frame: 2D data tabular data.**
 - Series: 1D data. We can think of Series as columnar data.
 - Index: A sequence of row labels.

```
elections.max(axis=0)
```

```
Candidate      Trump  
Party      Republican  
%           58.8  
Year         2016  
Result         win  
dtype: object
```

Axis 0
Column-wise

`.max()` along the 1 axis will return a value for each row.

```
elections.max(axis=1)
```

```
0      1980.0  
1      1980.0  
2      1980.0  
3      1984.0  
4      1984.0  
5      1988.0  
6      1988.0  
7      1992.0  
8      1992.0  
9      1992.0  
10     1996.0  
11     1996.0  
12     1996.0  
13     2000.0  
14     2000.0  
15     2004.0  
16     2004.0  
17     2008.0  
18     2008.0  
19     2012.0  
20     2012.0  
21     2016.0  
22     2016.0  
dtype: float64
```

Axis 1
Row-wise

Data Structures

- There are three fundamental data structures in pandas:
 - Data Frame: 2D data tabular data.
 - **Series: 1D data. We can think of Series as columnar data.**
 - Index: A sequence of row labels.

Series

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

Axis: 1 →

Columns

Index

Axis: 0 ↓

	Candidate	Party	Percent	Year	Outcome
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

Series

Candidate Series Party Series Percent Series Outcome Series Year Series

Data Structures

- There are three fundamental data structures in pandas:
 - Data Frame: 2D data tabular data.
 - **Series: 1D data. We can think of Series as columnar data.**
 - Index: A sequence of row labels.

Series

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

Axis: 1 →

Columns

Index ↓ Axis: 0

	Candidate	Party	Percent	Year	Outcome
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

Series

Candidate Series

Data Structures

- There are three fundamental data structures in pandas:
 - Data Frame: 2D data tabular data.
 - **Series: 1D data. We can think of Series as columnar data.**
 - Index: A sequence of row labels.

```
candidate = elections['Candidate']  
candidate
```

```
0      Reagan  
1      Carter  
2    Anderson  
3      Reagan  
4    Mondale  
5       Bush  
6    Dukakis  
7    Clinton  
8       Bush  
9      Perot  
10    Clinton  
11      Dole  
12    Perot  
13      Gore  
14      Bush  
15     Kerry  
16      Bush  
17     Obama  
18    McCain  
19     Obama  
20    Romney  
21    Clinton  
22     Trump  
Name: Candidate, dtype: object
```

Check the data type of `candidate` .

```
type(candidate)
```

```
pandas.core.series.Series
```

Data Structures

- There are three fundamental data structures in pandas:
 - Data Frame: 2D data tabular data.
 - Series: 1D data. We can think of Series as columnar data.
 - **Index: A sequence of row labels.**

Index

Axis: 1 →

Columns

Index

Axis: 0 ↓

	Candidate	Party	Percent	Year	Outcome
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

Series

Index

Data Structures

- There are three fundamental data structures in pandas:
 - Data Frame: 2D data tabular data.
 - Series: 1D data. We can think of Series as columnar data.
 - **Index: A sequence of row labels.**

```
elections.index
```

```
RangeIndex(start=0, stop=23, step=1)
```

`.index` returns a `RangeIndex()` object, which shows the start, end and step size of the row indices. `RangeIndex` is a memory-saving special case of `Int64Index` limited to representing monotonic ranges. If we want to simply get an array of index values, we can use `.to_numpy()`.

```
elections.index.to_numpy()
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,
        13, 14, 15, 16,
        17, 18, 19, 20, 21, 22], dtype=int64)
```


Data Structures

- There are three fundamental data structures in pandas:
 - Data Frame: 2D data tabular data.
 - Series: 1D data. We can think of Series as columnar data.
 - **Index: A sequence of row labels.**

```
elections_sample
```

	Candidate	Party	%	Year	Result
11	Dole	Republican	40.7	1996	loss
10	Clinton	Democratic	49.2	1996	win
21	Clinton	Democratic	48.2	2016	loss
14	Bush	Republican	47.9	2000	win
20	Romney	Republican	47.2	2012	loss
1	Carter	Democratic	41.0	1980	loss
13	Gore	Democratic	48.4	2000	loss
22	Trump	Republican	46.1	2016	win
16	Bush	Republican	50.7	2004	win
8	Bush	Republican	37.4	1992	loss

```
elections_sample.index
```

```
Int64Index([11, 10, 21, 14, 20, 1, 13, 22, 16, 8], dtype='int64')
```

Notice that the index is different and can no longer be expressed as `RangeIndex`. It maintained the index of the rows in the original table. This is very useful if we wanted to go back and relate derived tables with their original values.

Data Structures

- There are three fundamental data structures in pandas:
 - Data Frame: 2D data tabular data.
 - Series: 1D data. We can think of Series as columnar data.
 - **Index: A sequence of row labels.**

You can use the `.set_index()` operation to set the index of a DataFrame to one of the columns.

```
elections_sample.set_index('Year')
```

	Candidate	Party	%	Result
Year				
1996	Dole	Republican	40.7	loss
1996	Clinton	Democratic	49.2	win
2016	Clinton	Democratic	48.2	loss
2000	Bush	Republican	47.9	win
2012	Romney	Republican	47.2	loss
1980	Carter	Democratic	41.0	loss
2000	Gore	Democratic	48.4	loss
2016	Trump	Republican	46.1	win
2004	Bush	Republican	50.7	win
1992	Bush	Republican	37.4	loss

Indices (row labels) can also:
(1) be non-numeric
(2) have a name (e.g. "Year").

Data Structures

- There are three fundamental data structures in pandas:
 - Data Frame: 2D data tabular data.
 - Series: 1D data. We can think of Series as columnar data.
 - **Index: A sequence of row labels.**

You can use the `.set_index()` operation to set the index of a DataFrame to one of the columns.

```
elections_sample.set_index('Year')
```

	Candidate	Party	%	Result
Year				
1996	Dole	Republican	40.7	loss
1996	Clinton	Democratic	49.2	win
2016	Clinton	Democratic	48.2	loss
2000	Bush	Republican	47.9	win
2012	Romney	Republican	47.2	loss
1980	Carter	Democratic	41.0	loss
2000	Gore	Democratic	48.4	loss
2016	Trump	Republican	46.1	win
2004	Bush	Republican	50.7	win
1992	Bush	Republican	37.4	loss

Indices (row labels) do not have to be unique.

Data Structures

- There are three fundamental data structures in pandas:
 - Data Frame: 2D data tabular data.
 - Series: 1D data. We can think of Series as columnar data.
 - Index: A sequence of row labels.
- **Columns** (Series names).

```
elections.columns
```

```
Index(['Candidate', 'Party', '%', 'Year', 'Result'], dtype='object')
```

Column names in Pandas are almost always unique!

You really shouldn't have two columns named "Candidate".

	a	a.1	a.2	b	c	d
0	1	2	3	4	5	6
1	7	8	9	10	11	12

Pandas I

- DataFrames
- Pandas Data Structures
- **Importing Data**
- Indexing
- Utility Methods



Importing Data

- **Pandas** has a number of useful functions for importing data from common formats.

Pandas has a number of very useful file reading tools. You can see them enumerated by typing `pd.read` and pressing tab. Some common tools include:

- `pd.read_csv()` - Import a **comma-separated values (.csv)** file.
- `pd.read_excel()` - Import a **Microsoft Excel (.xlsx)** file.
- `pd.read_hdf()` - Import a **Hierarchical Data Format (.hdf)** file.
- `pd.read_html()` - Import a **Hypertext Markup Language (.html)** file.
- `pd.read_json()` - Import a **JavaScript Object Notation (.json)** file.
- `pd.read_pickle()` - Import a **Python Pickle (.pickle)** file.
- `pd.read_sql()` - Import a **Structured Query Language (.sql)** file.

Importing Data

- **Pandas** has a number of useful functions for importing data from common formats.

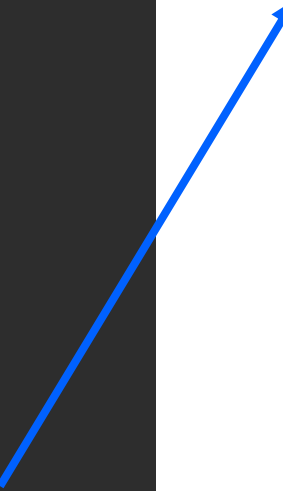
To get a quick look at the first 5 rows of a DataFrame, use the **.head()** method.

CSV Table

Lets import the CSV file **election.csv**.

```
# Import html tables to DataFrame
elections = pd.read_csv('elections.csv')

# View the first few rows
elections.head()
```



	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

Excel Table

Lets import the Excel file **fossil_fuel.xlsx**.

```
# Import html tables to DataFrame
fossil_fuel = pd.read_excel('fossil_fuel.xlsx',
                           sheet_name='Data1')

# View the first few rows
fossil_fuel.head()
```


Pandas I

- DataFrames
- Pandas Data Structures
- Importing Data
- **Indexing**
- Utility Methods



Indexing

- There are many ways to access rows and columns of a Pandas DataFrame.
- We will spend some time reviewing the most used options.
- []
- .loc[]
- Boolean Array Selection
- .iloc[]

Indexing

- There are many ways to access rows and columns of a Pandas DataFrame.
- We will spend some time reviewing the most used options.
- `[]`
- `.loc[]`
- Boolean Array Selection
- `.iloc[]`

Indexing - `[]` Operator

- You can access columns using the square `[]` brackets.
- You can pass a list of column names to select only those columns.

```
elections_sample[['Candidate', 'Year', 'Result']]
```

	Candidate	Year	Result
11	Dole	1996	loss
10	Clinton	1996	win
21	Clinton	2016	loss
14	Bush	2000	win
20	Romney	2012	loss
1	Carter	1980	loss
13	Gore	2000	loss
22	Trump	2016	win
16	Bush	2004	win
8	Bush	1992	loss

Indexing - `[]` Operator

- You can access columns using the square `[]` brackets.
- If you pass a list with a single element you get back a DataFrame.

```
elections_sample[['Candidate']]
```

Candidate	
11	Dole
10	Clinton
21	Clinton
14	Bush
20	Romney
1	Carter
13	Gore
22	Trump
16	Bush
8	Bush

Indexing - `[]` Operator

- You can access columns using the square `[]` brackets.
- If you pass single column name string, you get back a Series.

```
elections_sample['Candidate']
```

```
11      Dole
10     Clinton
21     Clinton
14      Bush
20     Romney
1      Carter
13      Gore
22     Trump
16      Bush
8       Bush
Name: Candidate, dtype: object
```

Indexing - `[]` Operator

- You can access columns using the square `[]` brackets.
- You can modify and even add columns using the square brackets `[]`.

```
temp = elections_sample.copy()
temp['Year'] = temp['Year'] * -1 + 25.
temp
```

	Candidate	Party	%	Year	Result
11	Dole	Republican	40.7	-1971.0	loss
10	Clinton	Democratic	49.2	-1971.0	win
21	Clinton	Democratic	48.2	-1991.0	loss
14	Bush	Republican	47.9	-1975.0	win
20	Romney	Republican	47.2	-1987.0	loss
1	Carter	Democratic	41.0	-1955.0	loss
13	Gore	Democratic	48.4	-1975.0	loss
22	Trump	Republican	46.1	-1991.0	win
16	Bush	Republican	50.7	-1979.0	win
8	Bush	Republican	37.4	-1967.0	loss

Indexing - `[]` Operator

- You can access columns using the square `[]` brackets.
- We can add a new column by assignment.

```
temp['Corrected Year'] = temp['Year'] * -1 + 25.  
temp
```

	Candidate	Party	%	Year	Result	Corrected Year
11	Dole	Republican	40.7	-1971.0	loss	1996.0
10	Clinton	Democratic	49.2	-1971.0	win	1996.0
21	Clinton	Democratic	48.2	-1991.0	loss	2016.0
14	Bush	Republican	47.9	-1975.0	win	2000.0
20	Romney	Republican	47.2	-1987.0	loss	2012.0
1	Carter	Democratic	41.0	-1955.0	loss	1980.0
13	Gore	Democratic	48.4	-1975.0	loss	2000.0
22	Trump	Republican	46.1	-1991.0	win	2016.0
16	Bush	Republican	50.7	-1979.0	win	2004.0
8	Bush	Republican	37.4	-1967.0	loss	1992.0

Indexing - `[]` Operator

- You can access columns using the square `[]` brackets.
- We can also index by row numbers using the `[]` operator.
- This is called a numeric slice.

```
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

Indexing - `[]` Operator

- You can access columns using the square `[]` brackets.
- Note:
 - `elections[0]` will not work unless the elections DataFrame has a column whose name is the numeric zero.
 - It is possible for columns to have names that are non-String types, e.g. numeric, datetime etc.

```
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

Indexing

- There are many ways to access rows and columns of a Pandas DataFrame.
- We will spend some time reviewing the most used options.
- []
- **.loc[]**
- Boolean Array Selection
- **.iloc[]**

Indexing - **.loc[]**

- You can access rows and columns of a DataFrame by name using the **.loc[]** syntax.
- The syntax for **.loc[]** is:
 - **df.loc[rows_list, column_list]**

elections_sample

	Candidate	Party	%	Year	Result
11	Dole	Republican	40.7	1996	loss
10	Clinton	Democratic	49.2	1996	win
21	Clinton	Democratic	48.2	2016	loss
14	Bush	Republican	47.9	2000	win
20	Romney	Republican	47.2	2012	loss
1	Carter	Democratic	41.0	1980	loss
13	Gore	Democratic	48.4	2000	loss
22	Trump	Republican	46.1	2016	win
16	Bush	Republican	50.7	2004	win
8	Bush	Republican	37.4	1992	loss

elections_sample.index

Int64Index([11, 10, 21, 14, 20, 1, 13, 22, 16, 8], dtype='int64')

elections_sample.columns

Index(['Candidate', 'Party', '%', 'Year', 'Result'], dtype='object')

Indexing - `.loc[]`

- You can access rows and columns of a DataFrame by name using the `.loc[]` syntax.
- The syntax for `.loc[]` is:
 - `df.loc[rows_list, column_list]`

```
elections_sample.loc[[11, 8], ['Party', 'Year']]
```

	Party	Year
11	Republican	1996
8	Republican	1992

```
elections_sample.loc[:, ['Party', 'Year']]
```

	Party	Year
11	Republican	1996
10	Democratic	1996
21	Democratic	2016
14	Republican	2000
20	Republican	2012
1	Democratic	1980
13	Democratic	2000
22	Republican	2016
16	Republican	2004
8	Republican	1992

```
elections_sample.loc[[11, 8]]
```

	Candidate	Party	%	Year	Result
11	Dole	Republican	40.7	1996	loss
8	Bush	Republican	37.4	1992	loss

Indexing - **.loc[]**

- You can access rows and columns of a DataFrame by name using the **.loc[]** syntax.
- The syntax for **.loc[]** is:
 - **df.loc[rows_list, column_list]**
- **Range Slicing [0:10]** works with **.loc[]** but only when the index is monotonic increasing or decreasing.

elections_sample

	Candidate	Party	%	Year	Result
11	Dole	Republican	40.7	1996	loss
10	Clinton	Democratic	49.2	1996	win
21	Clinton	Democratic	48.2	2016	loss
14	Bush	Republican	47.9	2000	win
20	Romney	Republican	47.2	2012	loss
1	Carter	Democratic	41.0	1980	loss
13	Gore	Democratic	48.4	2000	loss
22	Trump	Republican	46.1	2016	win
16	Bush	Republican	50.7	2004	win
8	Bush	Republican	37.4	1992	loss

Will it work for elections_sample?

Indexing - **.loc[]**

- You can access rows and columns of a DataFrame by name using the **.loc[]** syntax.
- The syntax for **.loc[]** is:
 - **df.loc[rows_list, column_list]**
- **Range Slicing [0:10]** works with **.loc[]** but only when the index is monotonic increasing or decreasing.

elections_sample

	Candidate	Party	%	Year	Result
11	Dole	Republican	40.7	1996	loss
10	Clinton	Democratic	49.2	1996	win
21	Clinton	Democratic	48.2	2016	loss
14	Bush	Republican	47.9	2000	win
20	Romney	Republican	47.2	2012	loss
1	Carter	Democratic	41.0	1980	loss
13	Gore	Democratic	48.4	2000	loss
22	Trump	Republican	46.1	2016	win
16	Bush	Republican	50.7	2004	win
8	Bush	Republican	37.4	1992	loss

Will it work for elections_sample?

```
elections_sample.loc[0:10, 'Candidate':'Year']
```

Returns:

```
ValueError: index must be monotonic increasing or decreasing
```

Indexing - **.loc[]**

- You can access rows and columns of a DataFrame by name using the **.loc[]** syntax.
- The syntax for **.loc[]** is:
 - **df.loc[rows_list, column_list]**
- If you give **.loc[]** single scalar arguments for the requested rows and columns, you get back just a single value.

```
elections.loc[19, 'Candidate']
```

```
'Obama'
```

Indexing

- There are many ways to access rows and columns of a Pandas DataFrame.
- We will spend some time reviewing the most used options.
- []
- .loc[]
- **Boolean Array Selection**
- .iloc[]

Indexing – Boolean Array Selection

- `[]` and `.loc[]` support arrays of Booleans as an input.
- In this case, the array must be exactly as long as the number of rows or columns.
- The result is a filtered version of the DataFrame, where only rows corresponding to True appear.
- This functionality is similar to **WHERE** in SQL.

elections_sample

	Candidate	Party	%	Year	Result
11	Dole	Republican	40.7	1996	loss
10	Clinton	Democratic	49.2	1996	win
21	Clinton	Democratic	48.2	2016	loss
14	Bush	Republican	47.9	2000	win
20	Romney	Republican	47.2	2012	loss
1	Carter	Democratic	41.0	1980	loss
13	Gore	Democratic	48.4	2000	loss
22	Trump	Republican	46.1	2016	win
16	Bush	Republican	50.7	2004	win
8	Bush	Republican	37.4	1992	loss

```
boolean_list = [False, False, False, False, True,  
                False, False, True, True, False]
```

```
elections_sample.loc[boolean_list]
```

	Candidate	Party	%	Year	Result
20	Romney	Republican	47.2	2012	loss
22	Trump	Republican	46.1	2016	win
16	Bush	Republican	50.7	2004	win

Indexing – Boolean Array Selection

- `[]` and `.loc[]` support arrays of Booleans as an input.
- In this case, the array must be exactly as long as the number of rows or columns.
- The result is a filtered version of the DataFrame, where only rows corresponding to True appear.
- This functionality is similar to **WHERE** in SQL.

elections_sample

	Candidate	Party	%	Year	Result
11	Dole	Republican	40.7	1996	loss
10	Clinton	Democratic	49.2	1996	win
21	Clinton	Democratic	48.2	2016	loss
14	Bush	Republican	47.9	2000	win
20	Romney	Republican	47.2	2012	loss
1	Carter	Democratic	41.0	1980	loss
13	Gore	Democratic	48.4	2000	loss
22	Trump	Republican	46.1	2016	win
16	Bush	Republican	50.7	2004	win
8	Bush	Republican	37.4	1992	loss

```
boolean_list = [False, False, False, False, True,  
               False, False, True, True, False]
```

```
elections_sample[boolean_list]
```

	Candidate	Party	%	Year	Result
20	Romney	Republican	47.2	2012	loss
22	Trump	Republican	46.1	2016	win
16	Bush	Republican	50.7	2004	win

Indexing – Boolean Array Selection

- One very common task in Data Science is filtering.
- Boolean Array Selection is one way to achieve this in Pandas.
- We start by observing logical operators like the equality operator `==` can be applied to Pandas Series data to generate a Boolean array.
- For example, we can compare the Result column to the String 'win'.

```
elections.head()
```

	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

```
iswin = elections['Result'] == 'win'  
iswin
```

```
0      True  
1     False  
2     False  
3      True  
4     False  
5      True  
6     False  
7      True  
8     False  
9     False  
10     True  
11     False  
12     False  
13     False  
14     True  
15     False  
16     True  
17     True  
18     False  
19     True  
20     False  
21     False  
22     True  
Name: Result, dtype: bool
```

Indexing – Boolean Array Selection

- The output of the logical operator applied to the Series is another Series with the same name and index, but of datatype Boolean.
- The entry at row *i* represents the result of the application of that operator to the entry of the original Series at row *i*.
- Such a Boolean Series can be used as an argument to the `[]` operator.

```
elections[elections['Result'] == 'win']
```

	Candidate	Party	%	Year	Result
0	Reagan	Republican	50.7	1980	win
3	Reagan	Republican	58.8	1984	win
5	Bush	Republican	53.4	1988	win
7	Clinton	Democratic	43.0	1992	win
10	Clinton	Democratic	49.2	1996	win
14	Bush	Republican	47.9	2000	win
16	Bush	Republican	50.7	2004	win
17	Obama	Democratic	52.9	2008	win
19	Obama	Democratic	51.1	2012	win
22	Trump	Republican	46.1	2016	win

Indexing – Boolean Array Selection

- We can select multiple criteria by creating multiple Boolean Series and combining them using the **&** operator.
- Using the logical negation **~** operator, which means Not.
- **~**(elections['%'] < 50)
- You can also use the **|** operator, which mean Or.

```
elections[elections['Result'] == 'win']
```

	Candidate	Party	%	Year	Result
0	Reagan	Republican	50.7	1980	win
3	Reagan	Republican	58.8	1984	win
5	Bush	Republican	53.4	1988	win
7	Clinton	Democratic	43.0	1992	win
10	Clinton	Democratic	49.2	1996	win
14	Bush	Republican	47.9	2000	win
16	Bush	Republican	50.7	2004	win
17	Obama	Democratic	52.9	2008	win
19	Obama	Democratic	51.1	2012	win
22	Trump	Republican	46.1	2016	win

We can select multiple criteria by creating multiple boolean Series and combining them using the & operator.

```
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

Indexing

- There are many ways to access rows and columns of a Pandas DataFrame.
- We will spend some time reviewing the most used options.
- `[]`
- `.loc[]`
- Boolean Array Selection
- `.iloc[]`

Indexing - `.iloc()`

- `.loc`'s cousin `iloc` is very similar, but is used to access based on numerical position instead of label.
- For example, to access to the first 3 rows and first 3 columns of a table, we can use `.iloc[0:3, 0:3]`.
- `iloc` slicing is exclusive, just like standard Python slicing of numerical values.

elections_sample

		0	1	2	3	4
		Candidate	Party	%	Year	Result
0	11	Dole	Republican	40.7	1996	loss
1	10	Clinton	Democratic	49.2	1996	win
2	21	Clinton	Democratic	48.2	2016	loss
3	14	Bush	Republican	47.9	2000	win
4	20	Romney	Republican	47.2	2012	loss
5	1	Carter	Democratic	41.0	1980	loss
6	13	Gore	Democratic	48.4	2000	loss
7	22	Trump	Republican	46.1	2016	win
8	16	Bush	Republican	50.7	2004	win
9	8	Bush	Republican	37.4	1992	loss

elections_sample.iloc[2:, 3:5]

	Year	Result
21	2016	loss
14	2000	win
20	2012	loss
1	1980	loss
13	2000	loss
22	2016	win
16	2004	win
8	1992	loss

Indexing

- We will use both `.loc[]` and `.iloc[]` in the course.
- `.loc[]` is generally preferred for a number of reasons, for example:
 - It is harder to make mistakes since you have to literally write out what you want to get.
 - Code is easier to read, because the reader doesn't have to know e.g. what column #31 represents.
 - It is robust against permutations of the data, e.g. the database admit switches the order of two columns.

Pandas I

- DataFrames
- Pandas Data Structures
- Importing Data
- Indexing
- **Utility Methods**



Utility Methods

- Pandas has number of very useful helper methods.

Utility Methods

- `.shape`
- `.size`

For example, `.shape` returns the number of rows and columns in a DataFrame as a tuple `(rows, cols)`.

```
elections.shape
```

```
(23, 5)
```

`.size` describes the number of "cells" in a DataFrame.

```
elections.size
```

```
115
```

Utility Methods

- `.sort_values()`

```
# Sort by Year in ascending order  
elections.sort_values(['Year'], ascending=True)
```

	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
7	Clinton	Democratic	43.0	1992	win
8	Bush	Republican	37.4	1992	loss
9	Perot	Independent	18.9	1992	loss

Utility Methods

- `.rename()`

```
elections.rename(columns={'%': 'Percent', 'Result': 'Outcome'}).head()
```

	Candidate	Party	Percent	Year	Outcome
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

Utility Methods

- .dtypes

```
elections.dtypes
```

```
Candidate    object  
Party        object  
%            float64  
Year         int64  
Result       object  
dtype: object
```

Utility Methods

- `.describe()`
- `.info()`
- `.unique()`
- `.value_count()`
- `.astype()`
- Many, many, many more.

Practice!

- Launch the Lecture 2.1 notebook from Quercus and review the material from this lecture in more detail.
- [Link](#)



CME538 Introduction to Data Science

Week 2 | Lecture 1 (2.1)

Pandas I.