



LECTURE 2

Pandas For EDA

Overview of useful Pandas functions for Exploratory Data Analysis

CSCI 3022 @ CU Boulder

Maribeth Oscamou

Content credit: [Acknowledgments](#)

Course Logistics: Your **First Week** At A Glance

Tues 1/16	Wed 1/17	Thurs 1/18	Fri 1/19	Sat 1/20	
Office Hours Begin (See Schedule on Canvas)	Attend & Participate in Class	(Optional): Attend Notebook Discussion with our TA (9am-10am Zoom) HW Due 11:59pm via Gradescope	In-Class Quiz (beginning of class) Attend & Participate in Class HW 2 Released	HW 1 Due 11:59pm via Gradescope (Includes Intro to CSCI 3022 Video assignment)	
	Previous week HW grades posted		HW 2 Released Discussion NB 2 released		

Getting To Know You:

I'd like to get a chance to be introduced to each of you!

1. **Please sign-up for a 15 min. timeslot ([link on first announcement on Canvas and Piazza](#))** to meet with me on Zoom during the first couple weeks to briefly introduce yourself and meet a few other classmates.

HW 1: Due Saturday

<https://canvas.colorado.edu/courses/101142/assignments/1893674>

If you started HW 1 before Tuesday at noon, there was a typo in Question 4. It has been fixed (it should read what's given below). This is in Piazza post labeled @9

"Write a Python expression in this next cell that's equal to $5 \times \left(3 \frac{10}{11}\right) - 50 \frac{1}{3} + 2^{0.5 \times 22} - \frac{7}{33} + 7$.

piazza CSCI 3022-001 Q & A Resources Statistics Manage Class

LIVE Q&A Drafts hw1 hw2 hw3 hw4 hw5 hw6 hw7 hw8 hw9 hw10 hw11 project exam

Unread Updated Unresolved Following Ban User Console · Note History:

New Post Search or add a post...

Show Actions

YESTERDAY

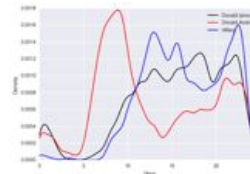
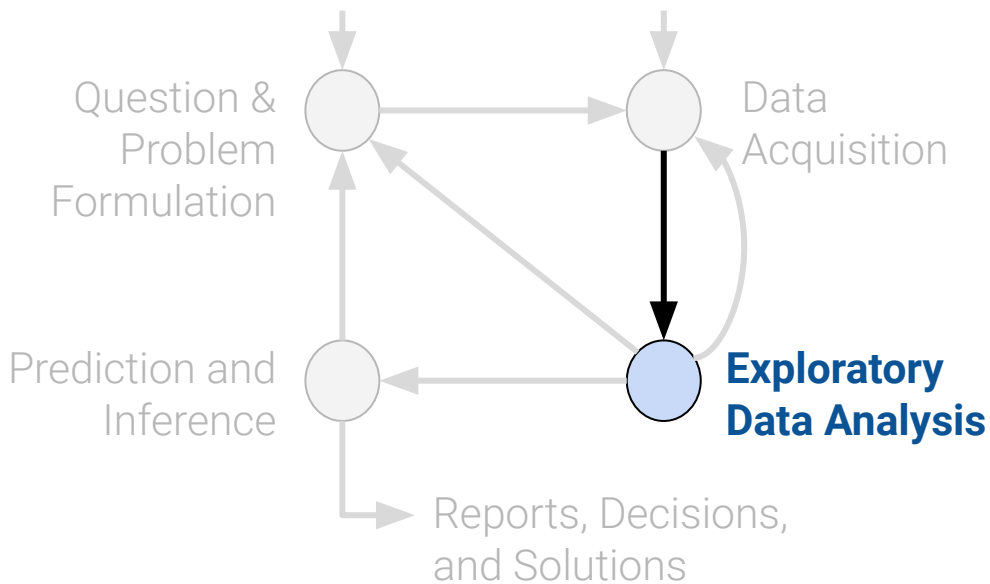
Private Friday Lecture Attendance E... 09:55 PM
Dear Professor Oscamou, I am on the hockey team here at CU, and we are currently on a bus

note @12

Clarification on HW 1 #10

For question 10, you can assume that $\lambda > 0$ when calculating the indefinite integral (

Plan for first 2 weeks



(Weeks 1 and 2)

EDA, Wrangling, and Data Visualization

Lesson Learning Objectives:

- Identify 5 key data properties to consider when doing Exploratory Data Analysis
- Use common utility functions in Pandas
- Understand methods for extracting data: `.loc`, `.iloc`, and `[]`.

Roadmap

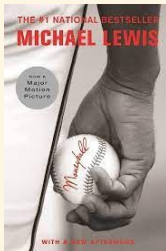
Lecture 02, CSCI 3022

- Exploratory Data Analysis - Key properties
- Pandas Bootcamp:
 - DataFrame Overview
 - Common Utility Functions
 - Extracting Data

Join our iClicker class: <https://join.iclicker.com/NZIG>

**Discuss with your group, then
answer in the poll**

**POLL: What is one question you
could try to answer with this
data?**



```
1 baseball = pd.read_csv("data/baseball.csv")
2 baseball
```

	Team	League	Year	RS	RA	W	OBP	SLG	BA	Playoffs	RankSeason	RankPlayoffs	G	OOBP	OSLG
0	ARI	NL	2012	734	688	81	0.328	0.418	0.259	0	NaN	NaN	162	0.317	0.415
1	ATL	NL	2012	700	600	94	0.320	0.389	0.247	1	4.0	5.0	162	0.306	0.378
2	BAL	AL	2012	712	705	93	0.311	0.417	0.247	1	5.0	4.0	162	0.315	0.403
3	BOS	AL	2012	734	806	69	0.315	0.415	0.260	0	NaN	NaN	162	0.331	0.428
4	CHC	NL	2012	613	759	61	0.302	0.378	0.240	0	NaN	NaN	162	0.335	0.424
...
1227	PHI	NL	1962	705	759	81	0.330	0.390	0.260	0	NaN	NaN	161	NaN	NaN
1228	PIT	NL	1962	706	626	93	0.321	0.394	0.268	0	NaN	NaN	161	NaN	NaN
1229	SFG	NL	1962	878	690	103	0.341	0.441	0.278	1	1.0	2.0	165	NaN	NaN
1230	STL	NL	1962	774	664	84	0.335	0.394	0.271	0	NaN	NaN	163	NaN	NaN
1231	WSA	AL	1962	599	716	60	0.308	0.373	0.250	0	NaN	NaN	162	NaN	NaN

1232 rows x 15 columns

The columns are:

- Team
- League
- Year
- Runs Scored (RS)
- Runs Allowed (RA)
- Wins (W)
- On-Base Percentage (OBP)
- Slugging Percentage (SLG)
- Batting Average (BA)
- Playoffs (binary)
- RankSeason
- RankPlayoffs
- Games Played (G)
- Opponent On-Base Percentage (OOBP)
- Opponent Slugging Percentage (OSLG)

Key Data Properties to Consider in EDA

Structure -- the “shape” of a data file

Granularity -- what does each record represent?

Scope -- how (in)complete is the data

Temporality -- how is the data situated in time

Faithfulness -- how well does the data capture “reality”

REVIEW FROM LAST TIME:

Are the data in a standard format or encoding?

- Tabular data: CSV, TSV, Excel, SQL
- Nested data: JSON or XML

Are the data organized in **records** or nested?

- Can we define records by parsing the data?
- Can we reasonably un-nest the data?

Does the data reference other data?

- Can we join/merge the data?
- Do we need to?

What are the **fields** in each record?

- How are they encoded? (e.g., strings, numbers, binary, dates ...)
- What is the type of the data?



Structure -- the “shape” of a data file

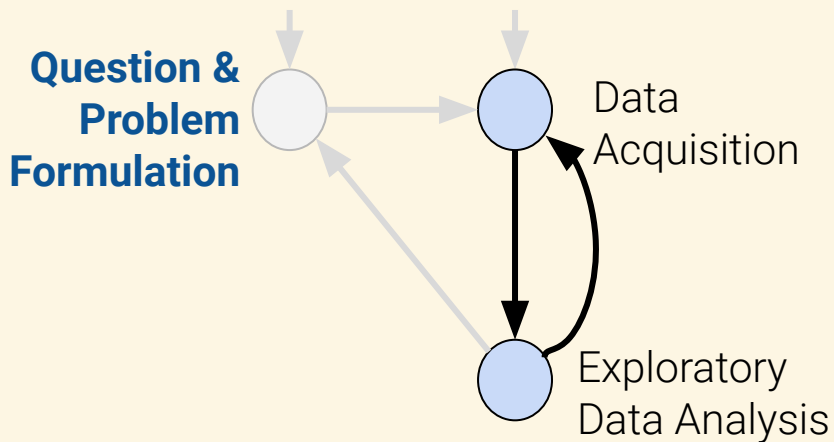
Granularity -- what does each record represent?

Scope -- how (in)complete is the data

Summary

You will do the most data wrangling when analyzing the structure of your data.

We need more Pandas knowledge to be able to answer these questions! So today, we'll [skip to the Pandas Bootcamp Slides](#), and revisit this after



Structure -- the “shape” of a data file

Granularity -- what does each record represent?

Scope -- how (in)complete is the data?

Temporality -- how is the data situated in time?

Faithfulness -- how well does the data capture “reality”?

Granularity

What does each **record** represent?

- Examples: a purchase, a person, a group of users, a house, a team

Do all records capture granularity at the same level?

- Some data will include summaries (aka **rollups**) as records

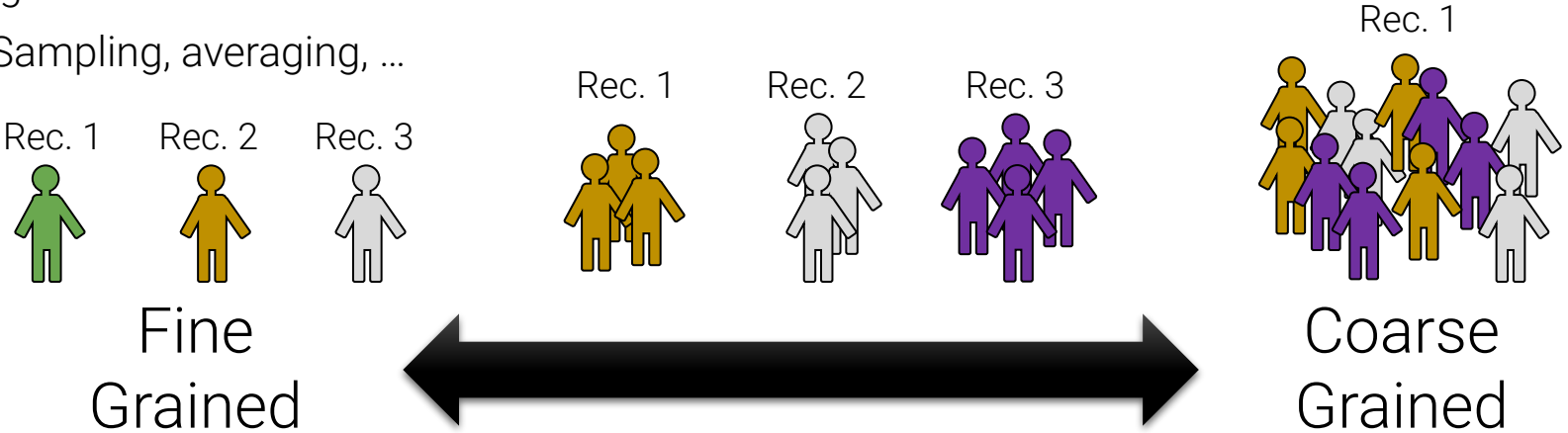
If the data are **coarse**, how were the records aggregated?

- Sampling, averaging, ...

	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

A **row** represents one record (i.e. an observation)

A **column** represents some characteristic, or feature, of that observation (here, the political party of that person).



Scope

Scope includes considering the **target population** we want to study, how to **access information** about that population, and what we are **actually measuring**

Target Population: The group that you want to learn something about.

Sampling Frame: The source from which the sample is drawn.

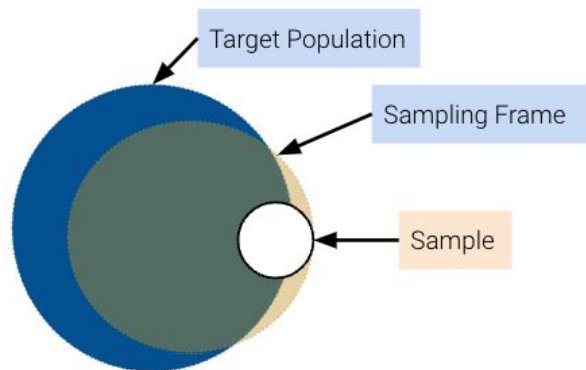
- If you're sampling people, the sampling frame is the set of all people that could possibly end up in your sample.

Sample: Who you actually end up sampling.

- A subset of your sampling frame.

The elements in a target population are not always people! Could be:

- **Bacteria** in your gut (sampled using DNA sequencing)
- **Houses** in a particular county
- **Small businesses** receiving a microloan
- **Published results** in a journal / field
- **Major League Baseball Teams**



Scope includes considering the **target population** we want to study, how to **access information** about that population, and what we are **actually measuring**

Does my data cover my population of interest?

- **Example:** I am interested in studying crime in Colorado but I only have Boulder crime data.

Is my data too expansive?

- **Example:** I am interested in student grades for CSCI 3022 but have student grades for all statistics classes.
- **Solution: Filtering** ⇒ Implications on sample?
 - If the data is a sample I may have poor coverage after filtering ...

Does my data cover the right time frame?

- More on this in Temporality...

Does my data cover my area of interest?

- **Example:** I am interested in studying crime in Colorado but I only have Boulder crime data.

Are my data too expansive?

- **Example:** I am interested in student grades for CSCI 3022 but have student grades for all statistics classes.
- **Solution: Filtering** ⇒ Implications on sample?
 - If the data is a sample I may have poor coverage after filtering ...

Does my data cover the right time frame?

- More on this in Temporality...

The **sampling frame** is the population from which the data were sampled. Note that this may not be the same as the actual target population

How complete/incomplete is the frame (and its data)?

- How is the frame/data situated in place?
- How well does the frame/data capture reality?
- How is the frame/data situated in time?

Data changes – when was the data collected/last updated?

Periodicity – Is there periodicity? Diurnal (24-hr) patterns?

What is the meaning of the time and date fields? A few options:

- When the “event” happened?
- When the data was collected or was entered into the system?
- Date the data was copied into a database? (look for many matching timestamps)

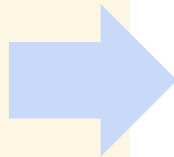
Time depends on where! (**time zones** & daylight savings)

- Learn to use **datetime** python library and Pandas **dt** accessors
- Regions have different datestring representations: 07/08/09?

Are there strange null values?

- E.g., **January 1st 1970**, January 1st 1900...?

Key Data Properties to Consider in EDA



Structure -- the “shape” of a data file

Granularity -- how fine/coarse is each datum

Scope -- how (in)complete is the data

Temporality -- how is the data situated in time

Faithfulness -- how well does the data capture “reality”

Faithfulness: Do I trust this data?

Does my data contain **unrealistic or “incorrect” values**?

- Dates in the future for events in the past
- Locations that don't exist
- Negative counts
- Misspellings of names
- Large outliers

Does my data violate **obvious dependencies**?

- E.g., age and birthday don't match

Was the data **entered by hand**?

- Spelling errors, fields shifted ...
- Did the form require all fields or provide default values?

Are there obvious signs of **data falsification**?

- Repeated names, fake looking email addresses, repeated use of uncommon names or fields.

Signs that your data may not be faithful (and proposed solutions)

Truncated data

Early Microsoft Excel limits: 65536 Rows, 255 Columns

Spelling Errors

Apply corrections or drop records not in a dictionary

Time Zone Inconsistencies

Convert to a common timezone (e.g., UTC)

Duplicated Records or Fields

Identify and eliminate (use primary key).

Units not specified or consistent

Infer units, check values are in reasonable ranges for data

- Be aware of consequences in analysis when using data with inconsistencies.
- Understand the potential implications for how data were collected.

Missing Data???

Examples

" "	1970, 1900
0, -1	NaN
999, 12345	Null

NaN: "Not a Number"

A. Drop records with missing values

- Probably most common
- **Caution:** check for biases induced by dropped values
 - Missing or corrupt records might be related to something of interest

B. Keep as NaN

C. Imputation/Interpolation: Inferring missing values

- **Average Imputation:** replace with an average value
 - Which average? Often use closest related subgroup mean.
- **Hot deck imputation:** replace with a random value
- **Regression imputation:** replace with a predicted value, using some model
- **Multiple imputation:** replace with multiple random values.

Examples

" "

0, -1

999, 12345

1970, 1900

NaN

Null

Missing Data/Default Values: Solutions

A. Drop records with missing values

- Probably most common
- **Caution:** check for biases induced by dropped values
 - Missing or corrupt records might be related to something of interest

B. Keep as NaN

C. Imputation/Interpolation: Inferring missing values

- **Average Imputation:** replace with an average value
 - Which average? Often use closest related subgroup mean.
- **Hot deck imputation:** replace with a random value
- **Regression imputation:** replace with a predicted value, using some model
- **Multiple imputation:** replace with multiple random values.

} (beyond this course)

Choice affects bias and uncertainty quantification (large statistics literature)

Essential question: why are the records missing?

Lesson Learning Objectives:

- Identify 5 key data properties to consider when doing Exploratory Data Analysis

- Exploratory Data Analysis - Key properties
- Pandas Bootcamp:
 - **DataFrame Overview**
 - Common Utility Functions
 - Extracting Data
 - Conditional Selection

Pandas Bootcamp

Lecture 02, CSCI 3022

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 **pandas** documentation, Stack Overflow, etc.

Review: Pandas DataFrames

We can think of a **DataFrame** as a collection of **Series** that all share the same **Index**.

0	1824
1	1824
2	1828
3	1828
4	1832
...	
177	2016
178	2020
179	2020
180	2020
181	2020
Name: Year,	

The Series "Year"

0	Andrew Jackson
1	John Quincy Adams
2	Andrew Jackson
3	John Quincy Adams
4	Andrew Jackson
...	
177	Jill Stein
178	Joseph Biden
179	Donald Trump
180	Jo Jorgensen
181	Howard Hawkins
Name: Candidate,	

The Series "Candidate"



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

DataFrame

Pandas DataFrames: Labels

- We describe "labels" as the bolded text at the top and left of a **DataFrame**.
- The **row labels** have a special name. We call them the *index* of a dataframe, and pandas stores the row labels in a special `pd.Index` object
 - By default, pandas assigns row labels as incrementing integers starting from 0.

	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

Row labels

Column labels

The DataFrame `elections`

Indices Are Not Necessarily Unique

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

Indices Are Not Necessarily Row Numbers

A **Row Index** (a.k.a. row labels) can:

- Be non-numeric.
- Have a name, e.g. "Candidate".

```
# Creating a DataFrame from a CSV file and specifying the Index column
elections = pd.read_csv("data/elections.csv", index_col = "Candidate")
```

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



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

Example: Setting the index to the "Party" column.

elections

	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

elections.set_index("Party")

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


Resetting the Index

- We can change our mind and reset the **Index** back to the default list of integers.

`elections.reset_index()`

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

	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



Column Labels Are Usually Unique!

Column labels 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

Accessing a DataFrame's columns and row indices: .index and .columns Attributes

```
elections = pd.read_csv("data/elections.csv", index_col="Year")
```

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

To view a DataFrame's row labels, use `df.index`:

```
elections.index
```

```
Index([1824, 1824, 1828, 1828, 1832, 1832, 1832, 1836, 1836, 1836,  
      ...  
      2016, 2016, 2016, 2016, 2016, 2016, 2020, 2020, 2020, 2020],  
      dtype='int64', name='Year', length=182)
```

To view a DataFrame's col labels, use `df.columns`:

```
elections.columns
```

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

The DataFrame `elections` with `"Year"` as Index

Lesson Learning Objectives:

- Identify 5 key data properties to consider when doing Exploratory Data Analysis
- Use common utility functions in Pandas

Pandas: Utility Functions

Lecture 02, CSCI 3022

- Exploratory Data Analysis - Key properties
- Pandas Bootcamp:
 - DataFrame Overview
 - **Common Utility Functions**
 - Extracting Data
 - Conditional Selection

Pandas provides an enormous number of useful utility functions.

Here are a few we will use frequently in this class:

- `head/tail`
- `shape`
- `info`
- `describe`
- `sort_values`
- `value_counts`
- `unique`

.head() and .tail()

Suppose we want to extract the first or last **n** rows from the **DataFrame**.

- `df.head(n)` will return the first **n** rows of the DataFrame **df**.
- `df.tail(n)` will return the last **n** rows.

elections

	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

elections.head(5)

	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

elections.tail(5)

	Candidate	Year	Party	Popular vote	Result	%
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

.shape

- `.shape` returns the shape of a **DataFrame** or **Series** in the form (number of rows, number of columns)

	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

`elections.shape`
`(182, 6)`

Prints the column integer positions, column labels, data types, memory usage, and the number of non-null cells in each column

elections

	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

elections.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 182 entries, 0 to 181
Data columns (total 6 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Year            182 non-null   int64
1   Candidate       182 non-null   object
2   Party          182 non-null   object
3   Popular vote    182 non-null   int64
4   Result         182 non-null   object
5   %              182 non-null   float64
dtypes: float64(1), int64(2), object(3)
memory usage: 8.7+ KB
```

describe()

- `describe()` returns a "description" of a **DataFrame** or **Series** that lists summary statistics of the data

	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

`elections.describe()`

	Year	Popular vote	%
count	182.000000	1.820000e+02	182.000000
mean	1934.087912	1.235364e+07	27.470350
std	57.048908	1.907715e+07	22.968034
min	1824.000000	1.007150e+05	0.098088
25%	1889.000000	3.876395e+05	1.219996
50%	1936.000000	1.709375e+06	37.677893
75%	1988.000000	1.897775e+07	48.354977
max	2020.000000	8.126892e+07	61.344703

.sort_values()

DataFrame.**sort_values**(column_name) must specify the name of the column to be used for sorting.

```
elections.sort_values(by = "Candidate", ascending=False)
```

	Year	Candidate	Party	Popular vote	Result	%
16	1848	Zachary Taylor	Whig	1360235	win	47.309296
74	1916	Woodrow Wilson	Democratic	9126868	win	49.367987
70	1912	Woodrow Wilson	Democratic	6296284	win	41.933422
37	1880	Winfield Scott Hancock	Democratic	4444976	loss	48.278422
19	1852	Winfield Scott	Whig	1386942	loss	44.056548
...
108	1956	Adlai Stevenson	Democratic	26028028	loss	42.174464
105	1952	Adlai Stevenson	Democratic	27375090	loss	44.446312
27	1864	Abraham Lincoln	National Union	2211317	win	54.951512
23	1860	Abraham Lincoln	Republican	1855993	win	39.699408
75	1920	Aaron S. Watkins	Prohibition	188787	loss	0.708351



By default, rows are sorted in **ascending** order.

182 rows x 6 columns

DEMO - Jupyter NB



The `Series.value_counts` method counts the number of occurrences of a each unique value in a Series.

- Return value is also a Series.

```
elections["Candidate"].value_counts()
```

```
Norman Thomas      5
Ralph Nader        4
Franklin Roosevelt  4
Eugene V. Debs     4
Andrew Jackson     3
..
Silas C. Swallow   1
Alton B. Parker    1
John G. Woolley    1
Joshua Levering    1
Howard Hawkins     1
Name: Candidate, Length: 132, dtype: int64
```


The `Series.unique` method returns an array of every unique value in a Series.

```
elections["Party"].unique()
```

```
array(['Democratic-Republican', 'Democratic', 'National Republican',  
      'Anti-Masonic', 'Whig', 'Free Soil', 'Republican', 'American',  
      'Constitutional Union', 'Southern Democratic',  
      'Northern Democratic', 'National Union', 'Liberal Republican',  
      'Greenback', 'Anti-Monopoly', 'Prohibition', 'Union Labor',  
      'Populist', 'National Democratic', 'Socialist', 'Progressive',  
      'Farmer-Labor', 'Communist', 'Union', 'Dixiecrat',  
      "States' Rights", 'American Independent', 'Independent',  
      'Libertarian', 'Citizens', 'New Alliance', 'Taxpayers',  
      'Natural Law', 'Green', 'Reform', 'Constitution'], dtype=object)
```

Lesson Learning Objectives:

- Identify 5 key data properties to consider when doing Exploratory Data Analysis
- Use common utility functions in Pandas
- Understand methods for extracting data: `.loc`, `.iloc`, and `[]`.

Pandas: Extracting Data

Lecture 02, CSCI 3022

- Exploratory Data Analysis - Key properties
- Pandas Bootcamp:
 - DataFrame Overview
 - Common Utility Functions
 - **Extracting Data**
 - Conditional Selection

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.

Common ways we may want to extract data:

- Grab the first or last **n** rows in the **DataFrame**.
- Grab data with a certain label.
- Grab data at a certain position.

We'll find that all three of these methods are useful to us in data manipulation tasks.

Label-based Extraction: .loc

Suppose we want to extract data with specific column or index labels.

```
df.loc[row_labels, column_labels]
```

The `.loc` accessor allows us to specify the **labels** of rows and columns to extract.

	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

Row labels

Column labels

The DataFrame `elections`

```
elections.loc[3, "Candidate"]
```

'John Quincy Adams'

```
elections.loc[[1, 4, 5], ["Party", "Candidate", "Result"]]
```

	Party	Candidate	Result
1	Democratic-Republican	John Quincy Adams	win
4	Democratic	Andrew Jackson	win
5	National Republican	Henry Clay	loss

```
elections.loc[[1, 4, 5], "Year": "Party" ]
```

	Year	Candidate	Party
1	1824	John Quincy Adams	Democratic-Republican
4	1832	Andrew Jackson	Democratic
5	1832	Henry Clay	National Republican




Label-based Extraction: .loc

To extract *all* rows or *all* columns, use a colon (:)

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

All rows for the columns with labels "Year", "Candidate", and "Result".

Ellipses (...) indicate more rows not shown. 

	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

```
elections.loc[[87, 25, 179], :]
```

All columns for the rows with labels 87, 25, 179.

	Candidate	Year	Party	Popular vote	Result	%
87	Herbert Hoover	1932	Republican	15761254	loss	39.830594
25	John C. Breckinridge	1860	Southern Democratic	848019	loss	18.138998
179	Donald Trump	2020	Republican	74216154	loss	46.858542

DEMO: NB



Integer-based Extraction: .iloc

Suppose we want to extract data according to its *position*.

```
df.iloc[row_integers, column_integers]
```

The `.iloc` accessor allows us to specify the **integers** of rows and columns we wish to extract.

- Python convention: The first position has integer index 0.

Ex:

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

'Andrew Jackson'

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

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

	0	1	2	3	4	5	
	Year	Candidate	Party	Popular vote	Result	%	
0	0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	4	1832	Andrew Jackson	Democratic	702735	win	54.574789
	

The DataFrame `elections`

Select the rows at positions 1, 2, and 3.

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

Select the columns at positions 0, 1, and 2.

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

Select *all* columns from integer 0 to integer 2.

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

Remember: integer-based slicing is right-end exclusive!

Selection Operators in Pandas

- `loc` performs **label-based** extraction. 1st argument is rows, 2nd argument is columns:
`df.loc[row_labels, column_labels]`
- `iloc` performs **integer-based** extraction. 1st argument is rows, 2nd argument is columns:
`df.iloc[row_integers, column_integers]`

- **SHORTCUT OPERATOR FOR 3 COMMON TYPES OF SELECTIONS: `[]`**

Only takes one argument, which may be:

- A list of **column labels**. `df[["Year", "Result"]]` (shortcut for `df.loc[:, ["Year", "Result"]]`)
- A single **column label**. `df["Candidate"]` (shortcut for `df.loc[:, "Candidate"]`)
- A slice of **row numbers** `df[3:7]` (shortcut for `df.iloc[3:7,:]`)

That is, `[]` is context sensitive.

Remember:

- `.loc` performs **label-based** extraction
- `.iloc` performs **integer-based** extraction
- `[]` is context sensitive (useful because it's much more concise than `.loc` or `.iloc`) In practice, `[]` is often used over `.iloc` and `.loc` in data science work. Typing time adds up!

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

- Safer: If the order of data gets shuffled in a public database, your code still works.
- Readable: 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).

. The following dataframe contains data about weather for a one week period:

```
import pandas as pd
```

```
df=pd.read_csv("data/weather.csv", index_col=["Day"])
```

	Weather	Temperature	Wind	Humidity
Mon	Sunny	72	13	30
Tue	Sunny	84	28	96
Wed	Sunny	91	16	20
Thu	Cloudy	67	11	22
Fri	Shower	71	26	79
Sat	Shower	65	27	62
Sun	Sunny	88	20	10

(a) (5 pts) Which code will return the output shown here?
Select all that apply.

	Weather	Temperature	Wind	Humidity
Thu	Cloudy	67	11	22
Fri	Shower	71	26	79

- | | | |
|---|---|---|
| <input type="checkbox"/> <code>df[4:5]</code> | <input type="checkbox"/> <code>df.iloc[3:4,:]</code> | <input type="checkbox"/> <code>df.loc["Thu":"Fri", "Weather":"Humidity"]</code> |
| <input type="checkbox"/> <code>df[3:5]</code> | <input type="checkbox"/> <code>df.iloc[3:5,:]</code> | <input type="checkbox"/> <code>df.loc[["Thu","Fri"],:]</code> |
| <input type="checkbox"/> <code>df[3:6]</code> | <input type="checkbox"/> <code>df.iloc[:,3:5]</code> | <input type="checkbox"/> <code>df[["Thu","Fri"]]</code> |
| | <input type="checkbox"/> <code>df.loc["Thu","Fri"]</code> | <input type="checkbox"/> <code>df.iloc[[3, 4], [0, 1, 2, 3]]</code> |