

EDA

COGS 108 Spring 2025
Jason Chen
Week 5

xic007@ucsd.edu

OH: Thu 3-5 pm

Discussion slides and materials adapted from Ruby Ying & previous quarter

Due dates

- Project Proposal: Wednesday Apr 30 @ 11:59 PM
- D4: Friday May 02 @ 11:59 PM

Project Proposal

- Due: Wednesday Apr 30
- Just make sure you've pushed your completed Project Proposal to your github group repo by 11:59 pm
 - Nothing else to submit

Project Proposal

- Work with your group to make a strong proposal
- Practice your git/github commands and strategies
- Use ReviewNB to look at changes between jupyter notebooks in Git
- Follow the instructions fully!

D4: Descriptive AND Exploratory Data Analysis

Web scraping tools

packages helpful for web scraping
import requests

The requests library is the de facto standard for making HTTP requests in Python. It abstracts the complexities of making requests behind a beautiful, simple API so that you can focus on interacting with services and consuming data in your application.

import bs4
from bs4 import BeautifulSoup

```
# packages helpful for webscraping  
import requests  
import bs4  
from bs4 import BeautifulSoup
```

Beautiful Soup is a Python library for pulling data out of HTML and XML files.

PART II: DESCRIPTIVE ANALYSIS

- Determine the shape of the data: `shape`
- Get descriptive statistics for quantitative variables: `describe()`
- Take a look at how party breaks down : `politics.party.value_counts()`
- Take a look at chamber breakdown: `politics.chamber.value_counts()`
- What about party broken down by chamber?
 - `DataFrame.groupby('_1_')['_2_'].value_counts()`
 - 1: what column are you “breaking down” by?
 - 2: what column are you interested in looking at?

But that first table included many years that we don't have Congressional data for...so what if we just got 1950 to now. **Get the subset of the `age` dataset where the years overlap with what we have in the `politics` dataset (data from 1950 to 2018).**

Store this in the variable `age_sub`.

```
[ ]: # YOUR CODE HERE
      raise NotImplementedError()
```

Filter to only include **Democrats** and Republicans

Filter your `politics` dataset to only include those members of Congress that belong to either the Democratic Party ('D') or Republican Party ('R'). Store these data in the variable `dem_rep`

(Hint: `dem_rep` should be a dataframe contains data only from the Democratic Party and the Replublican Party)

```
: # YOUR CODE HERE
  raise NotImplementedError()
```


PART II: DESCRIPTIVE ANALYSIS

- How do we get the age data after 1950?
 - Remember "iloc"
 - `age.iloc[start_row_index:end_row_index,]`
 - Look at `age['boolean condition']`
 - *Boolean condition* \Rightarrow `age['year'] == 1950`
- How do we get just democrats and republicans?
 - Remember the boolean condition
 - Note: In Pandas, we use "|" as an "or"
 - Ex: `DataFrame[(DataFrame['column1'] == value) | (DataFrame['column2'] == value2)]`
 - Ex: `DataFrame[(DataFrame['party'] == 'Democrat')]`

The plot you generated should make it clear that average age in Congress has clearly increased in recent years. But, is this driven by one party over another? Let's break this down by party to see. Additionally, we'd rather the years be on the x-axis, rather than the congress, as we have a better understanding of years. To do this, take a look at the `to_datetime()` function from `pandas` and consider how the `'termstart'` Series in the `politics` dataset can be used to extract the year. Assign the year to a new column `'year'` in the `politics` dataset.

(Hint: The `'termstart'` Series in the `politics` are strings, we will need to use `to_datetime()` function([document](#)) to convert the strings to `pandas.datetime` object which then can be used to extract the year.)

```
[ ]: ## get year column in there for x-axis  
      # YOUR CODE HERE  
      raise NotImplementedError()
```

EDA - date and time

`pd.to_datetime(politics['termstart']).dt.year`

pandas.to_datetime

```
pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False,
                    utc=None, format=None, exact=True, infer_datetime_format=False,
                    origin='unix', cache=True) [source]
```

Convert argument to datetime.

This function converts a scalar, array-like, `Series` or `DataFrame` /dict-like to a pandas datetime object.

Parameters: `arg` : *int, float, str, datetime, list, tuple, 1-d array, Series, DataFrame/dict-like*

The object to convert to a datetime. If a `DataFrame` is provided, the method expects minimally the following columns: `"year"`, `"month"`, `"day"`.

errors : `{'ignore', 'raise', 'coerce'}`, default `'raise'`

- If `'raise'`, then invalid parsing will raise an exception.
- If `'coerce'`, then invalid parsing will be set as `NaT`.
- If `'ignore'`, then invalid parsing will return the input.

dayfirst : *bool*, default `False`

Specify a date parse order if `arg` is str or is list-like. If `True`, parses dates with the day first, e.g. `"10/11/12"` is parsed as `2012-11-10`.

pandas.Series.dt.year

Series.dt.year

[source]

The year of the datetime.

Examples

```
>>> datetime_series = pd.Series(
...     pd.date_range("2000-01-01", periods=3, freq="Y")
... )
>>> datetime_series
0    2000-12-31
1    2001-12-31
2    2002-12-31
dtype: datetime64[ns]
>>> datetime_series.dt.year
0    2000
1    2001
2    2002
dtype: int64
```

EDA - Plot the ages in Congress broken down by party.

seaborn.lineplot

```
seaborn.lineplot(data=None, *, x=None, y=None, hue=None, size=None,
style=None, units=None, palette=None, hue_order=None, hue_norm=None,
sizes=None, size_order=None, size_norm=None, dashes=True, markers=None,
style_order=None, estimator='mean', errorbar=('ci', 95), n_boot=1000,
seed=None, orient='x', sort=True, err_style='band', err_kws=None,
legend='auto', ci='deprecated', ax=None, **kwargs)
```

```
sns.lineplot(x=..., y=..., hue=x, data=...);
```

x: the column name you are breaking the data down by.

EDA - trend differ by chamber

Does this trend differ by chamber?

Generate a plot to see if this trend looks the same in both chambers of Congress.

Relational Plots

```
sns.relplot(x = ...,  
            y = ...,  
            hue = ...,  
            col = 'chamber',  
            kind = "line",  
            data = ...);
```

seaborn.relplot

```
seaborn.relplot(data=None, *, x=None, y=None, hue=None, size=None,  
                style=None, units=None, row=None, col=None, col_wrap=None, row_order=None,  
                col_order=None, palette=None, hue_order=None, hue_norm=None, sizes=None,  
                size_order=None, size_norm=None, markers=None, dashes=None, style_order=None,  
                legend='auto', kind='scatter', height=5, aspect=1, facet_kws=None,  
                **kwargs)
```

Figure-level interface for drawing relational plots onto a FacetGrid.

This function provides access to several different axes-level functions that show the relationship between two variables with semantic mappings of subsets. The `kind` parameter selects the underlying axes-level function to use:

- `scatterplot()` (with `kind="scatter"`; the default)
- `lineplot()` (with `kind="line"`)

Extra keyword arguments are passed to the underlying function, so you should refer to the documentation for each to see kind-specific options.

Ignore this warning if you see it

```
/tmp/ipykernel_417/827630053.py:1: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`  
    assert party_counts[0] == 10290  
/tmp/ipykernel_417/827630053.py:2: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`  
    assert party_counts[-1] == 1
```

Section Materials

https://github.com/JasonC1217/COGS_108_B03-B04_Sp25/tree/master

or:

<https://tinyurl.com/4d8wx3ne>



THANKS!

Questions on EdStem or office hours

Office hours: Thu, 3-5 PM

