QSS20: Modern Statistical Computing

Session 02: Pandas data wrangling

Goal for today's session

- **Housekeeping: how to submit pset; misc**
- Mapping problem sets to pandas concepts and practicing each one using crime reports from DC
    1. Aggregation
    2. Creating new columns/transforming their type
    3. Row and column filtering

Logistics of problem set 1

- To submit, upload two files on Canvas (later on, via GitHub):
  - Raw `.ipynb` file that contains your answers in response to questions; please put these answers in `pset1_blank` rather than starting a new .py or .ipynb file
  - Compiled html or PDF
- Save each with your netid- eg: `pset1_f004bt8.ipynb` and `pset1_f004bt8.html`
- I will release problem set 1 to Canvas and the course GitHub no later than Sunday 09.18: `https://github.com/jhaber-zz/QSS20_public`
- Due Sunday 09.25 at 11:59 pm
- Four late days available for use across psets (let TA know if you're using a late day)

Other housekeeping

- DM your TA if reapportioning DataCamp to psets
- Any questions on office hours?
- **Plotting/graphs:**
  - You need to know the basics for the problem sets
  - **Those who know ggplot, seaborn, or matplotlib:** fine to use any for class work; for ggplot syntax, use `plotnine` wrapper
  - **Those who don't know any R or python-based viz:** do the optional DataCamp modules on Matplotlib and/or ggplot2
- Upload a photo or something unique to your Piazza profile picture

Goal for today's session

- Housekeeping: how to submit pset; misc
- **Mapping problem sets to pandas concepts and practicing each one using crime reports from DC**
  1. Aggregation
  2. Creating new columns/transforming their type
  3. Row and column filtering

Policy background for sentencing data

- **Data**: deidentified felony sentencing data from Cook County State's Attorney's Office (SAO)
- Released as part of push towards transparency with election of a new prosecutor in 2016

Opinion

EDITORIAL

Unequal Sentences for Blacks and Whites

By The Editorial Board

Dec. 17, 2016

Earlier this month Kim Foxx, the state's attorney for Cook County, Illinois, which covers Chicago, released six years' worth of raw data regarding felony prosecutions in her office. It was a simple yet profound act of good governance, and one that is all too rare among the nation's elected prosecutors. Foxx asserted that "for too long, the work of the criminal justice system has been largely a mystery. That lack of openness undermines the legitimacy of the criminal justice system." Source

Concepts in problem sets 1-2

Creating new columns
Aggregating using groupby and agg
Row filtering
Pandas operations like quantile, pd.to_datetime()
value_counts() and sort_values()
Loops and functions
Visualizing results

Goal for today's session

- Housekeeping: how to submit pset; misc
- Mapping problem sets to pandas concepts and practicing each one using crime reports from DC
    1. **Aggregation**
    2. Creating new columns/transforming their type
    3. Row and column filtering

Review of aggregation syntax: one grouping variable, summarizing one column)

```
grouping_result = df.groupby('grouping_varname')
                  ['varname_imsummarizing'].agg(
                  {functiontosummarize
                  }).reset_index()
```

- **Why might we use reset_index()?** This helps us treat the output as a DataFrame with clear, one-level columns

More aggregation syntax: one grouping variable, custom function to summarize one column

```
grouping_result = df.groupby('grouping_varname').agg(
                  {'varname_imsummarizing': functiontosummarize
                  }).reset_index(drop = True)
```

- **Why is there a dictionary inside of agg?** Lets us name the specific columns to summarize by and what functions to use; keys in the dictionary are the variables to summarize by, values are what function to use
- **Why might we use reset_index(drop = True)?** If we don't want to keep the old index (e.g., to avoid cluttering our DataFrame)

More aggregation syntax: one grouping variable, custom function to summarize multiple columns

```
grouping_result = df.groupby('grouping_varname').agg(
                  {'varname_imsummarizing': functiontosummarize,
                  'othervarname_imsummarizing': functiontosummarize
                  }).reset_index()
```

- **When might this be useful?** Can summarize different columns in different ways

More aggregation syntax: two grouping variables

```
grouping_result = df.groupby(['grouping_varname1',
                        'grouping_varname2']).agg(
                        {'varname_imsummarizing': 'functiontosummarize'
                        }).reset_index()
```

- **When might this be useful?** things like “how does this vary by time and category x?”

How do we structure the function inside the aggregation?

Three common ways of calling the function:

1. Functions that operate on pandas series, e.g.:
   ```
   df.groupby('month').agg({'offense': ['nunique', 'first']})
   ```
2. Functions from numpy (aliased here as np), e.g.:
   ```
   df.groupby('month').agg({'offense': [np.mean, np.median]})
   ```
3. "Lambda" functions we write ourself that take an argument
   ```
   df.groupby('month').agg({'offense':
                           lambda x: len(x.unique())})
   ```

Summarizing over multiple rows or columns (without aggregation)

```python
mean_threecols =     df[['colA', 'colB',
                     'colC']].apply(np.mean,
                     axis = 0)
```

- Pandas apply function: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html
- `axis` argument tells us whether to apply the function over columns (axis 0) or rows (axis 1)
- Can see in activity what happens

Before we code, let's group!

Groups for today and problem set 1:

Partner A	**Partner B**
Max Konzerowsky	Omario Corral-Williams
Anish Sikhinam	Johnny Reid
Emma Johnson	Nate Haile
Caroline Mahony	Siera Daly
Andrew Cho	Luca D'Ambrosio
Saige Gitlin	Daniel Céspedes
Nick Romans	Kayla Hamann
Daniel Xu	Filippo de Min
Giulio Frey	Justin Sapun

Pause for practice

Aggregation section (section 1) of `00_pandas_datacleaning`

Goal for today's session

- Housekeeping: how to submit pset; misc
- Mapping problem sets to pandas concepts and practicing each one using crime reports from DC
    1. Aggregation
    2. **Creating new columns/transforming their type**
    3. Row and column filtering

First type of column creation: binary indicators

Two general approaches that are "vectorized," or they work across all rows automatically without you needing to do a for loop:

1. `np.where`: similar to `ifelse` in R; useful if there's only 1-2 True/False conditions; can be used in conjunction with things like `df.varname.str.contains(‘‘some pattern’’)` if the column is string/char type
2. `np.select`: similar to `case_when` in R; useful for when there's either (1) several True/False conditions or (2) you're coding one set of categories into a different set of categories (this may come up in the problem sets)

Different types of np.where

```

## indicator for after 2020 christmas or not (make sure to
## format date in same way)
df['is_after_christmas'] = np.where(
                                    df.nameofdatecol > "2020-12-25",
                                    True, False)

## indicator for whether month is in spring quarter (april, may,
   june)
df['is_spring_q'] = np.where(
                             df.monthname.isin(["April", "May", "June"]),
                             True, False)

## indicator for whether someone's name contains johnson
df['is_johnson'] = np.where(
                            df.fullname.str.contains("Johnson"),
                            True, False)

## strip string of all instances of johnson
df['no_johnson'] = df.fullname.str.replace("Johnson", "")
```

Then, if we created binary indicator, can use for subsetting rows

```

## subset to after christmas
df_afterchristmas = df[df.is_after_christmas].copy()

## subset to after christmas AND spring quarter
## note parentheses around each
df_postc_spring = df[(df.is_after_christmas) &
                     (df.is_spring_q)].copy()

## subset to after christmas BUT NOT spring quarter
## note tilde ~ for negation
df_postc_notspring = df[(df.is_after_christmas) &
                     (~df.is_spring_q)].copy()
```

np.where is useful for single conditions, but what about multiple conditions?

- **Example**: code to fall q if September, October, November, or December; code to winter q if January, February, or March; code to spring q if April, May, or June; code to summer q if otherwise
- Gets pretty ugly if nested np.where

```

## quarter ind
df["quarter_type"] = np.where(df.monthname.isin(["Sept",
                        "Oct", "Nov",  "Dec"]), "fall_q",
                        np.where(df.monthname.isin(["Jan",
                        "Feb", "March"]), "winter_q",
                        np.where(df.monthname.isin(["April",
                        "May", "June"]), "spring_q", "summer_q")))
```

One approach: np.select

```

## step one: create a list of conditions/categories
## i can omit last category if i want or specify it
quarter_criteria = [df.monthname.isin(["Sept", "Oct",
                    "Nov", "Dec"]),
                    df.monthname.isin(["Jan", "Feb", "March"]),
                    df.monthname.isin(["April", "May", "June"])]

## step two: create a list of what to code each category to
quarter_codeto = ["fall_q", "winter_q", "spring_q"]

## step three: apply and add as a col
## note i can use default to set to the residual category
## and here that's a fixed value; could also retain
## original value in data by setting default to:
## df["monthname"] in this case
df["quarter_type"] = np.select(quarter_criteria,
                              quarter_codeto,
                              default = "summer_q")
```

A second approach: map and dictionary

```

## step one: create a dictionary where each key is a value
## I want to recode and each value is what I should recode to
quarter_dict = {"Jan": "winter_q",
                "Feb": "winter_q",
                "March": "winter_q",
                "April": "spring_q",
                "May": "spring_q",
                "June": "spring_q",
                "Sept": "fall_q",
                "Oct": "fall_q",
                "Nov": "fall_q",
                "Dec": "fall_q"}

## step two: map the original col to the new values
## using that dictionary
df["quarter_type"] = df.monthname.map(quarter_dict).fillna("
    summer_q")
```

Each was still tedious; are there ways to further simplify?

- Depending on the example, rather than enumerating all the conditions (e.g., `[‘‘April", "May", "June"]`), you can use a loop to subset a larger list to create that list more efficiently
- **Example**:
  - We have a column containing Dartmouth courses (e.g., QSS17, QSS20, ECON20, GOV10)
  - We want to pull out the QSS-prefix courses without using `df.coursename.str.contains`

List comprehension to loop through list

```python

## pool of courses
all_courses = df.coursename.unique()

## subset to those that contain QSS anywhere
only_qss = [course for course in all_courses
            if "QSS" in course]
```

- **for course in all_courses**: iterates over the list of courses
- **if condition**: tells it when to retain
- **course** at beginning: just return as is and don't do anything
- if we wanted to not only select but also strip out the course number, would do something like...

```python
import re
only_qss_nonum = [re.sub("[0-9][0-9]", "", course)
                  for course in all_courses
                  if "QSS" in course]
```

Pause for practice

Recoding section (section 2) of `00_pandas_datacleaning`

Goal for today's session

- Housekeeping: how to submit pset; misc
- Mapping problem sets to pandas concepts and practicing each one using crime reports from DC
    1. Aggregation
    2. Creating new columns/transforming their type
    3. **Row and column filtering**

Row filtering: combining multiple conditions

```
## select all rows where course code contains 30 and
## prefix is qss
qss_30_courses = df[(df.coursename.startswith("QSS")) &
                    (df.coursename.str.contains("30")].copy()
```

Two notes

- Using pandas built in methods (`startswith` and `str` accessor)- what would happen with latter if the variable was not a string?
- Extra parentheses (weird for R users)

Column filtering: combining with list comprehension

```
## subset to columns: "studentid" and columns with "2021" in the
    name (wide-format data)
grades_2021 = df[["studentid"] +
                [col for col in
                df.columns
                if "2021" in col]].copy()
```

Notes:

- Use .copy() to tell python that we're assigning a copy of the original dataframe (`df`) to the new object `grades_2021`; otherwise, gives us `SettingwithCopy` warning; ambiguity about whether we want any further changes to: (1) only apply to the slice or (2) propagate back to original df (in applied contexts, almost always want 1)
- Put "studentid" in brackets because i'm combining two lists

Pause for practice

Filtering section (section 3) of `00_pandas_datacleaning`