QSS20: Modern Statistical Computing

Session 02: Pandas data wrangling

Goal for today's session

▶ **Housekeeping: how to submit pset; misc**
▶ Mapping each question 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

How to submit the problem set

▶ Upload two files on Canvas. See this google doc and Jack's video on Slack for how to work
  ▶ 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
▶ Due Friday 01.14 at 11:59 pm
▶ Four late days available for use across psets (just slack your TA that you're using a late day)

Other housekeeping

▶ DM your TA if reapportioning DataCamp to psets (done at end of course)

▶ Any questions on office hours? summary link here for 3A; here for 6A (have ones Tues-Fri)

▶ **Plotting/graphs:**
  ▶ **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:** assigned optional DataCamp module on ggplot2; x-hours session tomorrow if interest; example code

▶ Replace your avatar in Slack with a picture or something unique so we no longer have a sea of these

Goal for today's session

▶ Housekeeping: how to submit pset; misc

▶ **Mapping each question 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

f ⊙ �🐦 ✉ ↗ 🔖

> *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 question 1

Problem set question	Concepts
1.1: unit of analysis in the data	Aggregating using groupby and agg
1.2.1 difference between original offense and updated offense	Creating new columns using np.where or checking equality
1.2.2 simplifying the charges	Pandas str.contains or list iteration
1.3: cleaning additional variables	Pandas str.contains (for race); np.where (for gender); Pandas quantile (for age); Pandas pd.to_datetime() (for converting a string column to a datetime column)
1.4: subsetting rows to analytic dataset	Row filtering

Concepts in question 2

Problem set question	Concepts
2.1 Over time variation in what % of cases are against Black defendants versus white defendants	Aggregating using groupby and agg
2.2 Over time variation in what % of Black versus white defendants face incarceration versus probation	Aggregating using groupby and agg
2.3.1 Common offenses	value_counts and sort values; set command
2.3.2 Between-group differences in incarceration for those common offenses	Aggregating using groupby and agg
2.3.3 Examining disparities before and after a policy change	Row filtering; loop

Goal for today's session

▶ Housekeeping: how to submit pset; misc
▶ Mapping each question 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 and summarizing one column

```
1  grouping_result = df.groupby('grouping_varname').agg(
2                    {'varname_imsummarizing': 'functiontosummarize'
3                    }).reset_index()
```

► **Why is there a dictionary inside of agg?** Helps us tell it what to summarize by and which functions to use; keys are the variables to summarize by; values are what function to use

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

Review of aggregation syntax: one grouping variable and summarizing multiple columns

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

▶ **When might this be useful?** For the pset questions on disparities, can do one summary of is_black_derived and another summary of is_white_derived

11

Review of aggregation syntax: two grouping variables

```
1  grouping_result = df.groupby(['grouping_varname1',
2                                'grouping_varname2']).agg(
3                                {'varname_imsummarizing': 'functiontosummarize'
4                                }).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 panda 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.mean]})`

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)

```
1  mean_threecols =        df [["colA", "colB",
2                           "colC"]].apply("mean",
3                           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

Pause for practice

Aggregation section (section 1) of `00_pandas_datacleaning`

Goal for today's session

▶ Housekeeping: how to submit pset; misc
▶ Mapping each question 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 (e.g., pset question asking you to code any offense with `Arson` in the string into a single arson category)

Different types of np.where

```
1
2 ## indicator for after 2020 christmas or not (make sure to
3 ## format date in same way)
4 df['is_after_christmas'] = np.where(df.nameofdatecol > "2020−12−25"
      ,
                                      True, False)
5
6 ## indicator for whether month is in spring quarter (april, may,
      june)
7 df['is_spring_q'] = np.where(df.monthname.isin(["April",
8                     "May", "June"]),
9                     True, False)
10
11 ## indicator for whether someone's name contains johnson
12 df['is_johnson'] = np.where(df.fullname.str.contains("Johnson"),
13                     True, False)
14
15 ## strip string of all instances of johnson
16 df['no_johnson'] = df.fullname.str.replace("Johnson", "")
```

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

```
1
2  ## subset to after christmas
3  df_afterchristmas = df[df.is_after_christmas].copy()
4
5  ## subset to after christmas AND spring quarter
6  ## note parantheses around each
7  df_postc_spring = df[(df.is_after_christmas) &
8                       (df.is_spring_q)].copy()
9
10 ## subset to after christmas BUT NOT spring quarter
11 ## note tilde ~ for negation
12 df_postc_notspring = df[(df.is_after_christmas) &
13                       (~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")))
```

20

One approach: np.select

```
1
2  ## step one: create a list of conditions/categories
3  ## i can omit last category if i want or specify it
4  quarter_criteria = [df.monthname.isin(["Sept", "Oct",
5                      "Nov", "Dec"]),
6                      df.monthname.isin(["Jan", "Feb", "March"]),
7                      df.monthname.isin(["April", "May", "June"])]
8
9  ## step two: create a list of what to code each category to
10 quarter_codeto = ["fall_q", "winter_q", "spring_q"]
11
12 ## step three: apply and add as a col
13 ## note i can use default to set to the residual category
14 ## and here that's a fixed value; could also retain
15 ## original value in data by setting default to:
16 ## df["monthname"] in this case
17 df["quarter_type"] = np.select(quarter_criteria,
18                                quarter_codeto,
19                                default = "summer_q")
```

A second approach: map and dictionary

```
1
2  ## step one: create a dictionary where each key is a value
3  ## I want to recode and each value is what I should recode to
4  quarter_dict = {"Jan": "winter_q",
5                  "Feb": "winter_q",
6                  "March": "winter_q",
7                  "April": "spring_q",
8                  "May": "spring_q",
9                  "June": "spring_q",
10                 "Sept": "fall_q",
11                 "Oct": "fall_q",
12                 "Nov": "fall_q",
13                 "Dec": "fall_q"}
14
15 ## step two: map the original col to the new values
16 ## using that dictionary
17 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

```
1
2 ## pool of courses
3 all_courses = df.coursename.unique()
4
5 ## subset to those that contain QSS anywhere
6 only_qss = [course for course in all_courses
7              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...

```
1 import re
2 only_qss_nonum = [re.sub("[0−9][0−9]", "", course)
3                    for course in all_courses
4                    if "QSS" in course]
```

24

Pause for practice

Recoding section (section 2) of 00_pandas_datacleaning

Goal for today's session

► Housekeeping: how to submit pset; misc
► Mapping each question 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

```
1 ## select all rows where course code contains 30 and
2 ## prefix is qss
3 qss_30_courses = df[(df.coursename.startswith("QSS")) &
4                     (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 parantheses (weird for R users)

Column filtering: combining with list comprehension

```
1  ## subset to columns: "studentid" and columns with "2021" in the
       name (wide-format data)
2  grades_2021 = df[["studentid"] +
3                  [col for col in
4                  df.columns
5                  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