QSS20: Modern Statistical Computing

Unit 03: Pandas for data manipulation

Agenda

- Housekeeping: logistics and psets
- Recap of Python basics
- Intro to static visualization
- Data manipulation using pandas
    1. Intro to dataframes
    2. Aggregation
    3. Creating new columns/transforming their type
    4. Row and column filtering

Meet Ellie, our fantastic peer tutor!

Eleanor (Ellie) Sullivan took QSS20 in Winter 2022 and was in a final project team with Ramsey, one of our TAs!

Please make use of peer tutoring sessions as a collaborative, supportive workspace for completing psets led by an experienced fellow student. Ellie will tell us more...

Peer tutoring schedule: Sun 8-9pm; Mon 7-8pm; Thurs 9-10pm

Agenda

- **Housekeeping: logistics and psets**
- Recap of Python basics
- Intro to static visualization
- **Data manipulation using `pandas`**
  1. Intro to dataframes
  2. Aggregation
  3. Creating new columns/transforming their type
  4. Row and column filtering

Class schedule and special guests

- Class moved from Monday, 01/16 at 3A time period to Tuesday, 01/17 at 3B time period
  - This means class meets later than usual this coming Tuesday: 4:30-6:20 PM (OK to bring coffee/tea to that class!)
- Special guest on Tuesday, 01/17 (at 3:30 PM): Ashley Doolittle, Associate Director of Dartmouth's Center for Social Impact and head of the Social Impact Practicum (SIP)
- Special guests on Wednesday, 01/25 (at 5:00 PM): Andrea Caoili and Ann Klein of the National Center for START Services (NCSS), our project partner
  - *Note:* "START" stands for "Systemic, Therapeutic, Assessment, Resources, and Treatment" and is a service model for treatment of individuals with intellectual and developmental disabilities (IDD). Here's a 1-hr documentary about their work.

Logistics of problem set 1

- To submit, upload two files using the Problem Set 1 Assignment on Canvas
  - 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
- Save each with your netid—e.g.: `pset1_f004bt8.ipynb` and `pset1_f004bt8.html`
- Pset 1 Due Sunday 01.15 at 11:59 PM
- *Note:* The wording for 3.2.C and 3.3 assume two TAs, but the data in parts 1-2 has only one TA. Adjust the numbering to reflect this as I indicated on Piazza

Logistics of problem sets in general

- I will release problem set 2 to Canvas and the course GitHub tomorrow; due date Sunday, 01.22 at 11:59 PM
- Pset 1 must be entirely your own work; we will put you in pairs for pset 2 (you will submit a single notebook between you)
- You will submit psets 3-5 using GitHub, which we will go over in two weeks
- Four late days available for use across psets; let teaching team know via Piazza if you're using a late day

Where we are

- Housekeeping: logistics and psets
- **Recap of Python basics**
- Intro to static visualization
- Data manipulation using `pandas`
    1. Intro to dataframes
    2. Aggregation
    3. Creating new columns/transforming their type
    4. Row and column filtering

Recap of Python basics

What do you remember from last class?

Recap of Python basics

Tips:

- Python data types: string, int, float, boolean, list, datetime, objects
- Another basic data type for course: `numpy` array
  - Like lists except only store a single data type and can be visualized as matrices/tables. Great for math (ML, NLP, etc)
- To transform each element of a list, use a list comprehension
- Can use `if` or `if/else` with list comprehension; these have slightly different syntax (see exs below)

Useful code snippets:

```
type(var) # get data type of var
int(var), float(var), str(var) # coerce type
len(somelist) # get length of a list
somearray.shape # get num rows/columns of arr
somelist[1:3] # get 2nd & 3rd elems of list
[el for el in somelist if len(el)>0] # filter list
[el if len(el)>0 else 0 for el in somelist] # filter/except
```

Where we are

- Housekeeping: logistics and psets
- Recap of Python basics
- **Intro to static visualization**
- Data manipulation using `pandas`
  1. Intro to dataframes
  2. Aggregation
  3. Creating new columns/transforming their type
  4. Row and column filtering

Walk through notebook with plotting example code

Link to static viz notebook using plotnine

Can use any plotting syntax for problem sets — popular ones are matplotlib (covered by DataCamp last chapter of introduction to pandas); seaborn; plotnine

Where we are

- Housekeeping: logistics and psets
- Recap of Python basics
- Intro to static visualization
- **Data manipulation using** `pandas`
    1. Intro to dataframes
    2. Aggregation
    3. Creating new columns/transforming their type
    4. Row and column filtering

Policy background for pset 2 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 set 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

Now let's practice these concepts using crime reports from DC!

Where we are

- Housekeeping: logistics and psets
- Recap of Python basics
- Intro to static visualization
- **Data manipulation using `pandas`**
  1. **Intro to dataframes**
  2. Aggregation
  3. Creating new columns/transforming their type
  4. Row and column filtering

Intro to data wrangling: how do dataframes differ from lists and arrays?

- In last lecture, we covered two structures for storing information in python:
  - Lists: structure built into python; 1-dimensional storage of information that can deal with information of different types in the same list
  - Arrays: requires the `numpy` package; n-dimensional (can be > 2) storage of information - usually use to store numeric information for efficient math calculations/model estimation
- DataFrames: 2-dimensional with rows (first dimension) and columns (second dimension) — sometimes called tabular data structure
  - `pandas` package (usually aliased as `pd`)
  - Each column can contain a different type of information
  - Each row references some unit of analysis (person; nation; city; 911 call; etc)

Two ways of interacting with dataframes

1. Creating our own
   - Less common
   - **Main use**: when we're creating data from some non-tabular source (e.g., a text string of a short politician bio; extract their name, party, and religious affiliation)
2. Reading in data stored in different formats
   - Focus for now: csv
   - Others we'll get to: excel; json; pkl or other serialized format; txt; spatial data stored as shapefiles

Creating our own dataframe: dictionary approach

Keys: column names; **values:** lists or arrays containing information

```
## create own df
### approach 1: dictionary where keys
## are names of columns
### and items are lists with information
name_list = ['Rebecca', 'Yifan', 'Sonali']
role_list = ['Instructor', 'TA', 'TA']
ht_list = [63, 70, 63.5]
my_df = pd.DataFrame({'names': name_list,
                     'role': role_list,
                    'fictional_height': ht_list})
my_df
my_df.dtypes
```

	names	role	fictional_height
0	Rebecca	Instructor	63.0
1	Yifan	TA	70.0
2	Sonali	TA	63.5

```
names                 object
role                  object
fictional_height     float64
dtype: object
```

Creating our own dataframe: nested lists

```python
## approach 2: list of lists
### each person's information is one list
rj_info = ['Rebecca', 'Instructor', 63]
yl_info = ['Yifan', 'TA', 70]
ss_info = ['Sonali', 'TA', 63.5]

### together they're a nested list
nested_info = [rj_info, yl_info, ss_info]
nested_info

### we can then make that into a dataframe
### need to specify column names
my_df_2 = pd.DataFrame(nested_info,
                      columns = ['names',
                                 'role',
                                 'fictional_height'])
my_df_2
```

```
[['Rebecca', 'Instructor', 63], ['Yifan', 'TA', 7
0], ['Sonali', 'TA', 63.5]]
```

	names	role	fictional_height
0	Rebecca	Instructor	63.0
1	Yifan	TA	70.0
2	Sonali	TA	63.5

More common way of interacting with dataframes: reading in data

- `os` package is important for finding the path of the file
  - `os.getcwd()` tells you the working directory you're in
- Two ways to structure path names (will return to these when we cover command line + GitHub in a couple weeks)
- **Way one (avoid if possible) absolute paths:**

  `'/Users/rebeccajohnson/Dropbox/ppol564_prepwork/prep_activities/f22_materials'`
- **Better way: relative paths to .py or .ipynb:** my data is stored two levels up from where my notebook is; can provide abbreviated pathname:

  `'../../data/example_data.csv'`
- Structure of read command pd.read_csv('path to file')

Once we have the data in Python, can summarize using built-in attributes or functions

- `dfname.dtypes:` returns a pandas series where the index is the name of the column; the value is the type of data it contains (eg str; int; float)
- `dfname.shape`: returns a length-2 tuple with dimensions of dataframe (number of rows, number of columns)
- `dfname.head()`: prints first n rows (defaults to 5)
- `dfname.tail()`: prints last n rows (defaults to 5)

Manipulations of data

Examples: finding mean height across the TAs; recoding heights into different categories; subsetting to a dataframe only containing TAs

names	**role**	**fictional_height**
Jaren	Instructor	68.0
Ramsey	TA	70.0
Eunice	TA	63.5
Ellie	Peer Tutor	64.5

Where we are

- Housekeeping: logistics and psets
- Recap of Python basics
- Intro to static visualization
- **Data manipulation using `pandas`**
  1. Intro to dataframes
  2. **Aggregation**
  3. Creating new columns/transforming their type
  4. Row and column filtering

Basic 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)

```
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!

Count off to 8...

Then find your number to form a group!

Pause for practice

Aggregation section (section 1) of `01_pandas_blank.ipynb`

Where we are

- Housekeeping: logistics and psets
- Recap of Python basics
- Intro to static visualization
- **Data manipulation using `pandas`**
    1. Intro to dataframes
    2. Aggregation
    3. **Creating new columns/transforming their type**
    4. 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")))
```

Better for recoding with multiple categories: `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")
```

Pause for practice

Recoding section (section 2) of `01_pandas_blank.ipynb`

Where we are

- Housekeeping: logistics and psets
- Recap of Python basics
- Intro to static visualization
- **Data manipulation using `pandas`**
  1. Intro to dataframes
  2. Aggregation
  3. Creating new columns/transforming their type
  4. **Row and column filtering**

Row filtering: combining multiple conditions

Say we have a column in our data that contains Dartmouth courses (e.g., QSS17, QSS20, ECON20, GOV10, COSC1) and we want to select all rows where the course code is "30" and the 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?
- Use parentheses around each when combining multiple conditions (weird for R users)

A useful tool for row filtering before we learn regex: pandas built-in functions

- `contains` and `startswith` are functions built into `pandas` that help us work with string/character variables
- General syntax: nameofdata.nameofstringcol.str.nameoffunction(argument if relevent)
- Examples:
  - df.stringvar.str.upper() and lower()
  - df.stringvar.str.replace()
  - df.stringvar.str.split() — defaults to splitting on spaces; can feed it other delimiters like ;

Column filtering: combining with list comprehension

Say we have a huge DataFrame listing all Dartmouth courses ever, and we want to find all the QSS courses plus COSC1. We could filter to "COSC1" plus any columns that have "QSS" in the name.

```
QSS_any2 = df[["COSC1"] +
              [col for col in
               df.columns
               if "QSS" in col]].copy()
```

Notes:

- Use .copy() to tell python that we're assigning a copy of the original dataframe (`df`) to the new object `QSS_any2`; otherwise, gives us `SettingwithCopy` warning because unsure whether we want our changes to propagate back to original DF (we almost never want this)
- Put "COSC1" in brackets to combine two lists

Pause for practice

Filtering section (section 3) of `01_pandas_blank.ipynb`

We covered

- Intro to static visualization
- Data manipulation using `pandas`
    1. Intro to dataframes
    2. Aggregation
    3. Creating new columns/transforming their type
    4. Row and column filtering