PPOL564: Data Science I

Unit 03: User-defined functions (part one)

Goals for today's session

- Review of upcoming deadlines
- Part three of previous lecture (row and column subsetting)
- User-defined functions
    - Lecture slides + example
    - Group activity
- Walk through notebook with plotting example code

Upcoming deadlines

- **Problem set one:** returned by Monday 09/06
- **Final project intro:** will release guideline doc soon; review questions in class Wednesday 09/14
- **Problem set two:** due Friday 09/16

Where we are

- Review of upcoming deadlines
- **Part three of previous lecture (row and column subsetting)**
- User-defined functions
    - Lecture slides + example
    - Group activity
- Walk through notebook with plotting example code

Where we are

- Review of upcoming deadlines
- Part three of previous lecture (row and column subsetting)
- **User-defined functions**
    - **Lecture slides + example**
    - Group activity
- Walk through notebook with plotting example code

Example from material on aggregating data

Used a one-line function (lambda function) to sort offenses from most to least frequent and pull the most-frequent offense:

```
dc_crim_2020.groupby(['WARD',
           'SHIFT']).agg({'OFFENSE':
           lambda x: x.value_counts(sort = True,
           ascending = False).index[0]})
```

Lambda functions versus “normal” python functions

- **Lambda functions**: think of as *single-use, throwaway* functions — code works there but if we wanted to perform similar operation (eg find most frequent weapon used), would need to copy/paste that lambda function into different aggregation calls
- **“Normal” python functions covered in DataCamp**: defined using the `def` command — helps us save time/make code more readable by avoiding repetitive code

Same example putting the code inside a function

```python
def most_common(one_col: pd.Series):
    '''
    Function to return name of most common category
    Parameters:
        one_col (pd.Series): pandas series

    Returns:
        top (str): string with name of most frequent category
    '''

    ## sort values
    sorted_series = one_col.value_counts(sort = True, ascending =
    False)
    ## get top
    top = sorted_series.index[0]
    ## return
    return(top)

## execute
dc_crim_2020.groupby(['WARD',
                'SHIFT']).agg({'OFFENSE':
                 lambda x: most_common(x)})
```

Three ingredients in a user-defined function

1. **Name of function and inputs**: name is arbitrary; multiple inputs are separated by commas (later, we'll cover setting inputs to default values)

   ```
   def most_common(one_col: pd.Series):
   ```

2. **Meat of function:** what the function does inside with the inputs

   ```
   ## sort values
   sorted_series = one_col.value_counts(sort = True,
   ascending = False)
   ## get top
   top = sorted_series.index[0]
   ```

3. **Return statement (if any):** returning one or more outputs; note that non-returned objects (eg in this example, the `sorted_series`) are discarded

   ```
   ## return
   return(top)
   ```

Building a function together

See first part of this notebook to follow along with the code:

02_functions_part1_blank.ipynb

Task

Write a function that takes in two arguments–a dataframe and an integer of a Ward number

- The function should subset to:
  - That ward
  - The ward immediately "below"that ward (if focal ward is Ward 2, Ward 1)
  - The ward immediately äbove"that ward (if focal ward is Ward 2, Ward 3)
- Find the number of unique crime reports (unique CCN) in each ward
- Should print the name + number of crimes in the ward with the most unique crime reports of that comparison set (returns nothing)

Breaking down into steps

1. Get the **meat** of the function working outside the function with one example
2. Figure out what parts of that meat you want to generalize
3. Get that generalization working outside the function
4. Construct the function
5. Execute it on the one example and make sure it produces same output as step 1
6. Execute it on multiple examples

Meat of function with one example (ward 3)

```python
## get list of wards + neighbors
neighbor_wards = [3 - 1, 3 + 1]
wards_touse = [3] + neighbor_wards

## then, use isin command to subset the data
## to those wards
df_focal = dc_crim_2020[dc_crim_2020.WARD.isin(wards_touse)].copy()

## then, use groupby to find unique
ward_ccn = df_focal.groupby('WARD')['CCN'].nunique().reset_index
        ()

## finally, get the top one (multiple ways)
top_ward = ward_ccn.sort_values(by = 'CCN',
                ascending = False).head(1)

## print
print("Ward with most reports of neighbors is WARD " + str(top_ward
    ['WARD'].values[0]) +
     " with N reports: " + str(top_ward.CCN.values[0]))
```

Many things we could generalize

Focusing on bolded two (ward and dataframe name) but large list; depends on what we want to use function to do:

- **Ward we're focusing on (hard coded to 3)**
- **Name of data frame (hard coded to** `dc_crim_2020`
- Name of ward column (hard coded to WARD)
- Number of neighbors to look at (hard coded to 1 above and 1 below)
- Name of crime identifier column (hard coded to CCN)

Highlighting parts where ward and dataframe name are hard coded

```
## get list of wards + neighbors
neighbor_wards = [3 - 1, 3 + 1]
wards_touse = [3] + neighbor_wards

## then, use isin command to subset the data
## to those wards
df_focal = dc_crim_2020[dc_crim_2020.WARD.isin(wards_touse)].copy()

## then, use groupby to find unique
ward_ccn = df_focal.groupby('WARD')['CCN'].nunique().reset_index()

## finally, get the top one (multiple ways)
top_ward = ward_ccn.sort_values(by = 'CCN',
            ascending = False).head(1)
```

Replace hard-coded parts with placeholder

```
## get list of wards + neighbors
neighbor_wards = [focal_ward - 1, focal_ward + 1]
wards_touse = [focal_ward] + neighbor_wards

## then, use isin command to subset the data
## to those wards
df_focal = df[df.WARD.isin(wards_touse)].copy()

## then, use groupby to find unique
ward_ccn = df_focal.groupby('WARD')['CCN'].nunique().reset_index()

## finally, get the top one (multiple ways)
top_ward = ward_ccn.sort_values(by = 'CCN',
            ascending = False).head(1)
```

Can still test outside the function

```
## testing obj
focal_ward = 3
df = dc_crim_2020.copy()

## get list of wards + neighbors
neighbor_wards = [focal_ward - 1, focal_ward + 1]
wards_touse = [focal_ward] + neighbor_wards

## then, use isin command to subset the data
## to those wards
df_focal = df[df.WARD.isin(wards_touse)].copy()

## then, use groupby to find unique
ward_ccn = df_focal.groupby('WARD')['CCN'].nunique().reset_index()

## finally, get the top one (multiple ways)
top_ward = ward_ccn.sort_values(by = 'CCN',
                ascending = False).head(1)
```

Then, putting it all together for the function

(see notebook for documentation; omitted here on slide for space reasons)

```python
def compare_wards(focal_ward: int, df: pd.DataFrame):

    ## get list of wards to use
    neighbor_wards = [focal_ward - 1, focal_ward + 1]
    wards_touse = [focal_ward] + neighbor_wards

    ## subset to those
    df_focal = df[df.WARD.isin(wards_touse)].copy()

    ## find crimes per ward
    ward_ccn = df_focal.groupby('WARD')['CCN'].nunique().
    reset_index()

    ## finally, get the top one
    top_ward = ward_ccn.sort_values(by = 'CCN', ascending = False).
    head(1)

    ## print
    print("Ward with most reports of neighbors is WARD " + \
          str(top_ward['WARD'].values[0]) +
     " with N reports: " + str(top_ward.CCN.values[0]))
```

Executing repeatedly: can combine with list comprehension

```

## repetitive execution
compare_wards(focal_ward = 3, df = dc_crim_2020)
compare_wards(focal_ward = 6, df = dc_crim_2020)

## using list comprehension
[compare_wards(focal_ward = i, df = dc_crim_2020)
    for i in [3, 6]]
```

Latter may be especially useful if the function returns something that we later want to combine

Where we are

- Review of upcoming deadlines
- Part three of previous lecture (row and column subsetting)
- **User-defined functions**
  - Lecture slides + example
  - **Group activity**
- Walk through notebook with plotting example code

Break for group activity

We provide the “outside of function” code; you work to generalize this into a function and execute

Section 2 of this notebook: 02_functions_part1_blank.ipynb

Where we are

- Review of upcoming deadlines
- Part three of previous lecture (row and column subsetting)
- User-defined functions
  - Lecture slides + example
  - Group activity
- **Walk through notebook with plotting example code**
  - Can use any plotting syntax for problem set — popular ones are matplotlib (covered by DataCamp last chapter of introduction to pandas); seaborn; plotnine
  - Notebook gives plotnine syntax; more practice next week