QSS20: Modern Statistical Computing

Unit 04: User-defined functions and LaTeX

Goals for today's session

- Review of upcoming deadlines
- Recap of data manipulation with pandas
- User-defined functions
  - Lecture slides + example
  - Group activity
- LaTeX/Overleaf

Goals for today's session

- **Review of upcoming deadlines**
- Recap of data manipulation with pandas
- User-defined functions
  - Lecture slides + example
  - Group activity
- LaTeX/Overleaf

Where we're headed

Upcoming deadlines:

- **Problem set one:** Returned by Monday 01/23
- **Problem set two:** due Sunday 01/22 at 11:59 PM
- **Final project review questions:** you bring Monday 01/23
- **Final project survey: due Friday, 01/27, 11:59 PM** (then we'll put you in groups)
  - If you've formed a team already, feel free to get started on final project milestones (first one *not due until 02/12*)

Content:

- Today 01/18: Functions
- Monday 01/23: Workflow basics
- Wednesday 01/25: Reshaping & merging data

Problem set grading:

- Pset 1 grades back by coming Monday, 01/22 (if submitted on time)
- Pset 2 grades back by Monday after, 01/29

DataCamp deadlines

- **Today:** Writing your own functions
- **Monday 01/23:** Intro to GitHub; Intro to Shell
- **Wednesday 01/25:** Data merging basics; Merging tables with different join types

TITLE	ASSIGNEES	STATUS	DUE BY	C	A	CR	DETAILS
Python Data Science Toolbox (Part 1) **Writing your own functions** Chapter	Organization	DUE SOON	Jan 18, 15:30 EST	14	29	48%	View
GitHub Concepts **Introduction to GitHub** Chapter	Organization	Active	Jan 23, 15:30 EST	6	29	20%	View
Introduction to Shell **Manipulating files and directories** Chapter	Organization	Active	Jan 23, 15:30 EST	9	29	31%	View
Joining Data with pandas **Data Merging Basics** Chapter	Organization	Active	Jan 25, 15:30 EST	3	29	10%	View
Joining Data with pandas **Merging Tables With Different Join Types** Chapter	Organization	Active	Jan 25, 15:30 EST	3	29	10%	View

Note on difficulty of activities/psets vs. DataCamp

1. **DataCamp**: meant as gentle intro before pset challenges; not realistic for entry-level data science jobs; provides a lot of handholding in terms of noting (1) exactly which commands to use; (2) helper code; (3) very simplified/cleaned data
2. **Real-world data science:** more difficult than the problem set; would be asked "hey, did this policy reduce or widen disparities" and start with a blank notebook and be 100% reliant on google/stackoverflow
   - **Translating question into concrete approach:** define disparities (charges, incarceration or not, sentencing conditional on incarceration); find which variables measure that; deal with duplicates
   - **Data cleaning without scaffolding:** recognizing the errors in the datetime and that `errors_coerce = True` would set a lot of valuable data to missing; further deduplication of judge names; investigating `PROMIS CONVERSION` (eg coding that to missing)
   - A lot of these things won't throw errors if you run an analysis without fixing but will lead to flawed results/incorrect policy conclusions

Office hours: Some reminders

- **Required** that you attend office hours or group tutoring at least once
- **Prof. office hours:**
  - Mondays & Wednesdays 2:15-3:15 pm
    - Location: Silsby 103
- **TA office hours:**
  - Ramsey: Sunday & Friday 2-3pm
    - Location: Haldeman 046 and Zoom
  - Eunice: Tuesday 1-2pm
    - Location: Zoom
- **Peer tutoring with Ellie:**
  - Sundays 8-9 PM in Dart 002
  - Mondays 7-8 PM in Reed 101
  - Thursday 9-10 PM in Baker 370

Intro to collaborative function guide

See "Wiki" page on `QSS20_public` GitHub repo:
https://github.com/jhaber-zz/QSS20_public/wiki

Where we are

- Review of upcoming deadlines
- **Recap of data manipulation with pandas**
- User-defined functions
  - Lecture slides + example
  - Group activity
- LaTeX/Overleaf

Recap of pandas column creation & filtering

What do you remember from last class?

Recap of Pandas column creation & filtering

Tips:

- Use `np.where` to make binary indicator with a single condition
  - Aimen corollary: If your outcome conditions are just True/False, this is same as using the condition alone (see code below)
- Use `np.select` if you have multiple conditions and categories
- String columns: access handy functions like `str.replace`
- List comp. to get columns with condition for strings, e.g. `if 'DATE' in col`, vs. for series, e.g. `df.col.str.contains('DATE')`

Useful code snippets:

```
np.where(df.name.str.contains("Johnson"), True, False) # same as...
df.fullname.str.contains("Johnson") # also True/False
np.where(df.fullname.str.contains("Johnson"),
    "is_johnson", "not_johnson") # more typical use of np.where
np.select(criteria, codeto, default="summer") # multiple conditions
df[(df.after_christmas)&(~df.is_spring)] # ()&(); '~' for negation
df[["OFFENSE"]+[col for col in df.columns if "DATE" in col]]
```

Where we are

- Review of upcoming deadlines
- Recap of data manipulation with pandas
- **User-defined functions**
  - **Lecture slides + example**
  - Group activity
- LaTeX/Overleaf

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

```
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 "above" 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)

```
## 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)

```
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

```
1
2 ## repetitive execution
3 compare_wards(focal_ward = 3, df = dc_crim_2020)
4 compare_wards(focal_ward = 6, df = dc_crim_2020)
5
6 ## using list comprehension
7 [compare_wards(focal_ward = i, df = dc_crim_2020)
8     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
- Recap of data manipulation with pandas
- **User-defined functions**
  - Lecture slides + example
  - **Group activity**
- LaTeX/Overleaf

Before we code, let's group!

Find your pset 2 partner (same as last time)!

While you're up, take a stretch/bio break

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

Goals for today's session

- Review of upcoming deadlines
- Recap of data manipulation with pandas
- User-defined functions
  - Lecture slides + example
  - Group activity
- **LaTeX/Overleaf**

Overview before activity

- LaTeX: typesetting language
- Can work with locally using things like TexMaker, etc.
- Here, we'll be interacting with it via Overleaf, which is similar to Google docs but for LaTeX and facilitates collaboration/easier troubleshooting of compile errors

Non-exhaustive list of things that can cause compilation errors

1. Underscores or certain special characteristics without an "escape" before them, e.g.:

```
## Ex. 1: this causes error due to underscore without escape
The file is called: file_here.R
## works
The file is called: file\_here.R

## Ex. 2: comments out rest of code after percent symbol
This increased by 5%
## this works
This increased by 5\%
```

2. Start entering math mode but fail to exit it, e.g.:

```
## Ex. 3: this causes errors
We calculate fraction as $\dfrac{5}{10} and then do...
## this works
We calculate fraction as $\dfrac{5}{10}$ and then do
```

“Environments”, or ways to go beyond standard text

- Itemized list

```
\begin{itemize}
\item First item...
\item
\end{itemize}
```

- Numbered list

```
\begin{enumerate}
\item First item...
\item
\end{enumerate}
```

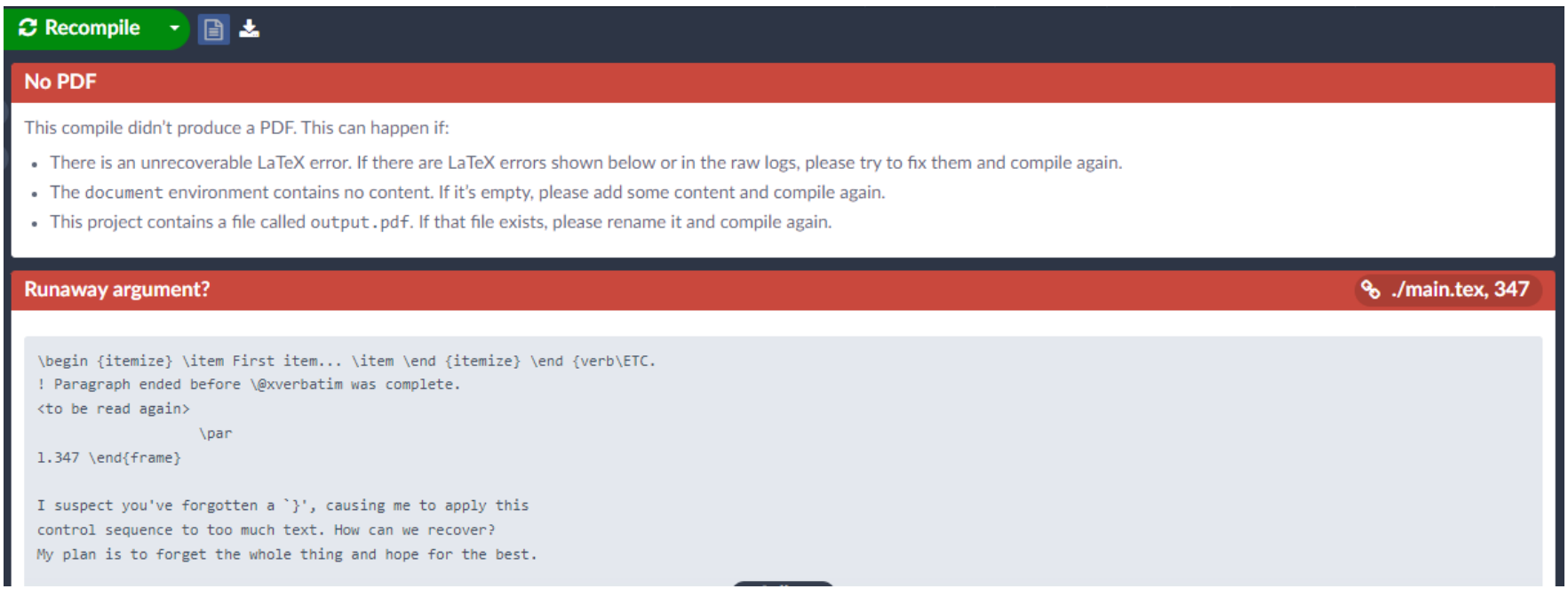
- Figure

```
\begin{figure}
\caption{my caption}
\label{fig:myfig}
\includegraphics[scale = 0.5]{example_graphic.png}
\end{figure}
```

Leads to another set of compilation errors

- Runaway argument or forgotten end group
- Usually means you began an environment but forgot to end it; can happen with long tables, deeply nested lists, etc. where easy to lose track

Example:



Compilation errors

- Common w/ complicated docs
- Ways to address:
  1. **Recompile frequently!**
  2. Try to interpret and google the error–not always easy since error messages may not be clear/informative w.r.t. line numbers (esp. on Overleaf)

Other useful commands

```
## create a numbered section and label it to cross-ref
\section{This is my section outlining disparities}
\label{sec:disparities}

## reference a section in text
In Section \ref{sec:disparities} I discuss...

## reference a table or fig in text
Table \ref{tab:tabname} and Figure \ref{fig:myfig} show...

## stop a figure or table from going into the next section
[! h] (inside \figure{} env; stay where it is in code)
\FloatBarrier (before \& after figure/table; don't float off)
(in addition to stuff at the start of the \begin{table})
```

Break for LaTeX tables and figures activity

- Link to template to copy over (click 'Menu' in top-left then Actions/'Copy Project')
- Link to Python activity:

`02_latex_output_examples_blank.ipynb`

We covered

- User-defined functions
  - Lecture slides + example
  - Group activity
- LaTeX/Overleaf

Lastly: Notecards

- One thing you learned today
- One challenge or lingering question you encountered