STAT 29000 Projects

## Topics: python, R, SQL, and associated tools

**Motivation:** Practice, practice, practice. Continuing to learn about and use the tools we’ve addressed this semester will make you faster and more efficient. You will be able to break problems into logical parts, and will discover you get “stuck” less often.

**Context:** We’ve explored a wealth of tools in using python, R, & SQL. We are now going to use what you’ve learned to solve a variety of different problems.

**Scope:** The scope of this project encompasses all topics covered this semester, namely, python, R, SQL, and associated tools.

**Important note:**

Please remember that we assigned 12 projects this semester, and students are able to *drop* their lowest two scores, so that only 10 projects are included in the overall grading. This is the same scheme as we had during the fall semester.

Also (same as during the fall semester), we will not have a final exam. Instead, we’ve provided 4 “optional” projects (below), which can be use as substitutes for any of the 10 required projects. You do not need to do any of these optional make-up projects, if you are happy with your grades from 10 out of the regular 12 projects, but you are welcome to do as many of them as you like. (A make-up project is simply used for a grade replacement.)

**Important note:** For any of the following projects that require Python, please use the project11 kernel on <https://notebook.scholar.rcac.purdue.edu>. We cannot guarantee that any other setup will work.

# ——— Optional Project 1 ———

#python #scraping

## Question 1: goodreads

**1a.** In [project 8](https://datamine.purdue.edu/seminars/spring2020/stat29000project08.html), we had an R function called guess\_id that, given a space separated query, returned a *guess* at what the imdb id is. We could then use that id to pull up a webpage with information about the respective movie or tv show. The same company (Amazon) owns both imdb.com and goodreads.com, so, unsurprisingly you can do the same sort of thing. Write a function called guess\_id which accepts a space-separated query and returns a “guess” of the goodreads id as an int (use the function int to convert to an int).

**Hints:**

1. <https://www.goodreads.com/search?q=SEARCH+QUERY> is the link to get search results on goodreads.com. If, for example, you wanted to search for “The Eye of the World”, you’d need to GET the page using <https://www.goodreads.com/search?q=the+eye+of+the+world>.
2. Similarly to what was done before, assume the search is accurate and the first result is what you are looking for.
3. The number directly following the /show/ part of the url is the id. Any non-numeric text following that number should be discarded. For example, this is the result for “The Eye of the World”: <https://goodreads.com/book/show/228665.The_Eye_of_the_World?from_search=true&qid=iAY1shIWAs&rank=1>. In this case, “228665” is the id. In fact, the following link will work just as well: <https://goodreads.com/book/show/228665>.

guess\_id("the eye of the world") # 228665  
guess\_id("the book thief") # 19063  
guess\_id("the book theif") # 19063

**Item(s) to submit:**

* A cell in a Jupyter notebook with your function guess\_id. Include the import commands for the libraries/modules/functions that you used.

**1b.** It’s fun to take advantage of systems that goodreads already has in place. For example, when you visit a books webpage <https://www.goodreads.com/book/show/228665> it has a list to the right that shows books that “readers also enjoyed”. If you click the link “See similar books…”, you will see a list of similar books. Write a function called suggest that, given a search query, utilizes our guess\_id function from (1a), and returns a pandas dataframe of the similar books with the following columns: goodreads\_id (same type of id from (1a)), title, author, average\_rating, number\_of\_ratings, and summary. The first row should be the book that our function “guessed”.

**Hints:**

1. Similar to how we find a page for an individual book, there appears to be a “similar” id, that finds books that are similar to a certain book. For example, the similar id for “The Eye of the World” is “2008238”. To use that id, place it in the url: <https://goodreads.com/book/similar/SIMILAR_ID> which ends up as <https://goodreads.com/book/similar/2008238>.
2. You can use the following snippet to extract the id from a url:

import re  
id = re.search('\/([0-9]+)(?=[^\/]\*$)', MY\_URL).group(1)

1. The summary data will be messy. Use the [contents](https://www.crummy.com/software/BeautifulSoup/bs4/doc/#contents-and-children) instead of [string](https://www.crummy.com/software/BeautifulSoup/bs4/doc/#string) to access the contents of the summary. Assume that the variable contents is the result of accessing the contents attribute:

# to "flatten" a list: https://stackoverflow.com/questions/2158395/flatten-an-irregular-list-of-lists  
import collections  
def flatten(l):  
 for el in l:  
 if isinstance(el, collections.Iterable) and not isinstance(el, (str, bytes)):  
 yield from flatten(el)  
 else:  
 yield el  
   
contents = soup.find(something).contents  
summary = ' '.join(flatten(contents))  
  
# the result, "summary", will be sufficient. No need for further cleaning.

suggest("harry potter and the sorcerer's stone")  
# first 6 goodreads\_id's: 3, 2767052, 5907, 41865, 17157681, 28187

**Item(s) to submit:**

* A cell in a Jupyter notebook with your function suggest. Include the import commands for the libraries/modules/functions that you used.

## Question 2: create a question

**2a.** The entire internet is yours for the taking. Find a website with data you are interested in scraping. You have two options:

1. Write a question that asks the user to write a function to scrape info from the website given an id of some sort. Based on the id, the function will fetch a particular set of data.
2. Write a question that asks the user to scrape a set of data. Unlike option (1), this question should require the scraped data to be of a significant size – not merely a few factoids.

For each option, provide the answer to your question, including any code you used. You only need to do (1) or (2) not both.

**Item(s) to submit:**

* A markdown cell in a Jupyter notebook with your newly created question.
* A code cell in the same Jupyter notebook with the solution to your question.

**2b.** Create an interesting plot or graphic using the data you scraped in (2a). If option (1) was chosen, use your function to scrape info for at least 5 “ids”, and use the resulting data to create your graphic.

**Item(s) to submit:**

* A jpeg or png image with your newly created graphic.
* As you are not limited on *how* you create your graphic, you are not required to submit any code for this question.

# ——— Optional Project 2 ———

#python #numpy #pandas

## Question 1: [bears, beets, battlestar galactica](https://www.youtube.com/watch?v=WaaANll8h18)

We’ve provided you with a set of data [here](https://datamine.purdue.edu/seminars/spring2020/the_office_dialogue.csv) and on scholar:

/class/datamine/data/spring2020/the\_office\_dialogue.csv.

Load this data into a pandas dataframe called office\_data.Take a look at the first 5 rows of the data. Each row in the dataframe corresponds to a character’s line in an episode (and season). Can we predict if the an episode will be good based on how many lines loved characters have?

First, we need to define “good”. Consider the episode good if it’s imdb\_rating is greater or equal to 8.5. Next, we must organize our data. In this question you will be guided through the steps.

**1a.** *(1 pt)* We want to create a dataframe where every row corresponds to a single, unique episode, and the columns contain information on that unique episode. Information should include imdb\_rating and the number of lines certain characters have. To do so we will create two dataframes, office\_ratings and office\_dialogue, and will merge them.

First, aggregate office\_data to a dataframe named office\_ratings. This dataframe should contain the imdb\_rating for every episode and season. Then, run the code below to reset the index of office\_rating. The columns should be: season, episode, and imdb\_rating.

import pandas as pd  
import numpy as np  
  
# your code here to obtain office\_rating  
  
office\_rating = office\_rating.reset\_index()

You can test your dataframe by running the command below:

office\_rating.shape # (should be (186, 3))

**Keywords:** *pandas: groupby, agg, mean*

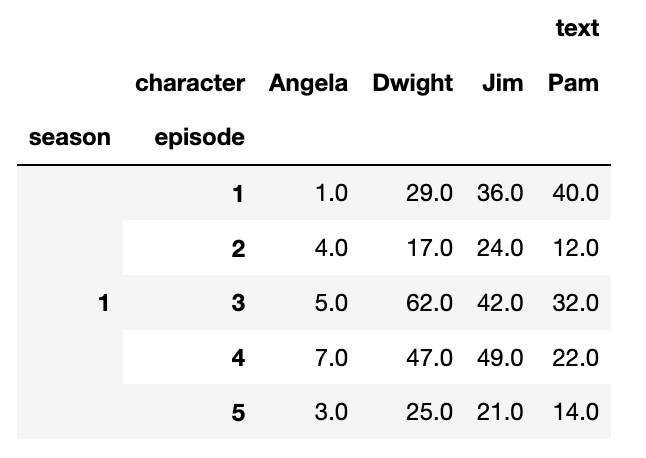
**Item(s) to submit:**

* A cell in the Jupyter notebook containing the python code used to create the office\_rating dataframe.

**1b.** *(2 pts)* Create a new dataframe called office\_dialogue with 6 columns: season, episode, Jim, Pam, Dwight, and Angela. For each episode and season, we want the corresponding number of lines for each character listed in the characters\_to\_evaluate list provided below:

characters\_to\_evaluate = ['Dwight', 'Jim', 'Pam', 'Angela']

Below is a snapshot of what your office\_dialogue should look like:



**Keywords:** *pandas: groupby, agg, count, pivot\_table, isin*

**Hints:**

1. Begin by selecting only rows where the character is in the characters\_to\_evaluate list.
2. Group by: season, episode, and character.
3. Aggregate the text by count (see the next **Hint**) to get the number of lines each character has.
4. [Pivot](https://pandas.pydata.org/docs/user_guide/reshaping.html#pivot-tables) the aggregated data using season and episode as indexes, character as column, and text (result of (3)) as values.\*

**Hint:** To know how many lines a character has in a certain episode and season, count the number of rows with that character. For example, “Jim” has in total 6,268 lines during the 9 seasons:

office\_data[office\_data['character'].eq('Jim')]['text'].count()

**Item(s) to submit:**

* A cell in the Jupyter notebook containing the python code used to create office\_dialogue dataframe.

**1c.** *(2 pts)* Merge the office\_dialogue and the office\_rating dataframes by season and episode. Call this new dataframe combined\_office\_df. Then, use the command below to rename some of the columns to be easier to understand AND creates a column called is\_good that has value True if imdb\_rating is equal or greater than 8.5, and False if not.

combined\_office\_df = combined\_office\_df.rename(columns={('text', 'Dwight'): "Dwight", ('text', 'Jim'): "Jim", ('text', 'Pam'): "Pam", ('text', 'Angela'): "Angela"})  
combined\_office\_df['is\_good'] = combined\_office\_df['imdb\_rating'] >= 8.5

Separate combined\_office\_df into a train set called train\_df and a test set called test\_df. Place the data for seasons 1-7 into train\_df, and leave the data from seasons 8-9 for the test\_df.

We are going to build a classifier that classifies an episode to tell us if it is “good” or not. We will train our classifier using the data in train\_df and test our classifier on the data in test\_df.

Test your train\_df and test\_df by running:

train\_df.shape # (Should be (115, 8))   
test\_df.shape # (Should be (71, 8))

**Keywords:** *pandas: merge, isin, range*

**Item(s) to submit:**

* A cell in the Jupyter notebook containing the python code used to create: combined\_office\_df, train\_df and test\_df.

**1d.** *(1 pt)* Separate train\_df into 2 numpy arrays named train\_good and train\_notGood. train\_good is for the “good” episodes and train\_notGood is for episodes that are not “good”. We based whether an episode is “good” or not on the column is\_good. The numpy arrays should contain only the following columns: Angela, Jim, Pam, and Dwight (characters in characters\_to\_evaluate).

Run the code below to calculate the mean and standard deviation for each column in train\_good and train\_notGood:

mean\_good = np.nanmean(train\_good, axis=0)  
std\_good = np.nanstd(train\_good, axis=0)  
mean\_notGood = np.nanmean(train\_notGood, axis=0)  
std\_notGood = np.nanstd(train\_notGood, axis=0)

**Hint:** *You* ***may*** *want to first separate the train\_df dataframe, and convert it to a numpy array.*

**Keywords:** *pandas: to\_numpy, eq*

**Item(s) to submit:**

* A cell in the Jupyter notebook containing the python code used to create the two numpy arrays,
* The code provided to calculate mean and standard deviation for each column in train\_good and train\_notGood.

**1e.** *(2 pts)* The function calculate\_ind\_probability provided below calculates the probability of a vector named lines\_vector, based on a mean\_vector and a std\_vector. The lines\_vector contains information on the number of lines for each character in characters\_to\_evaluate in one episode.

import math  
def calculate\_ind\_probability(lines\_vector, mean\_vector, std\_vector):  
 exponent\_step = np.exp(-((lines\_vector-mean\_vector)\*\*2 / (2 \* std\_vector\*\*2 )))  
 prob\_array = (1 / (np.sqrt(2 \* math.pi) \* std\_vector)) \* exponent\_step  
 return np.prod(prob\_array)

Test the function by running the command to calculate the probability for one “Good” episode in train\_good under the distribution of “Good” episodes:

calculate\_ind\_probability(train\_good[0,:], mean\_good, std\_good)

*Note that we are making a big (and false) assumption to consider the character’s number of lines as independent of each other.*

Create a function called classify\_episode that takes as an input lines\_vector. The function should calculate two probabilities: 1. Given that an episode is “good”, the probability we will see the lines from lines\_vector. 2. Given that an episode is not “good”, the probability we will see the lines from lines\_vector.

Then, it should compare both probabilities, and based on the higher probability, classify the episode. To be consistent with our dataset, if the episode is classified as “good”, your function should return True, and if it is classified as not “good”, your function should return False.

To calculate (1) the probability of seeing the lines from lines\_vector given the episode is “good”, use the provided calculate\_ind\_probability function with the mean\_good and std\_good from (1d). Similarly, use mean\_notGood, std\_notGood, and calculate\_ind\_probability function to get the (2) second probability.

**Keywords:** *def*

**Item(s) to submit:**

* A cell in the Jupyter notebook containing your classify\_episode function.

**1f.** *(2 pts)* Create a numpy array named test for the test\_df. test should contain only the number of lines for each character in characters\_to\_evaluate. Create an array called predicted that uses your classify\_episode function from (1e) to classify if the epsiode in test will be good or not. Be sure to use np.apply\_along\_axis function. Then, run the code below to compare the predicted class vs. the observed class, and get a percentage of how accurate our naive bayes classifier is.

**keywords:** *pandas: to\_numpy, numpy:apply\_along\_axis*

test = # your code here to create numpy array test  
predicted = # your code here to get predicted classes for the episodes in seasons 8 and 9  
round(100\*np.mean(predicted == test\_df['is\_good']),2)

**Item(s) to submit:**

* A cell in the Jupyter notebook containing the update Python code to obtain the predictions for episodes in seasons 8 and 9.

# ——— Optional Project 3 ———

#R #shiny

Recently you became familar Shiny, an R package for building interactive web apps straight from R. Shiny can be very useful to communicate the results of your analysis in an interactive manner. This is a valuable skill to anyone dealing with data.

We’ve provided you with the start to an app.

Your friends are planning their next vacation, you decided to create a Shiny app to provide some interesting information that could help them better plan their vacation. We’ve provided you with the start to the app [here](https://datamine.purdue.edu/seminars/spring2020/stat29000optionalproject03template.R) or on scholar:

/class/datamine/data/spring2020/stat29000optionalproject03template.R.

**1.** Complete the “Lead time” tab by including:

* A histogram of the lead\_time from the hotelSubset data.
* A summary() of the lead\_time from the hotelSubset data.

**2.** Complete the “Vacation length” tab by including:

* A side-by-side boxplot with the totalstay variable for the selected month (months), as well as the totalstay variable for the prior month and next month.
* Use renderPrint to display a small dataframe that shows names of the three months in one column, and the calculated average stay time in another.

**3.** Use the data provided [here](https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-02-11/hotels.csv) to add an additional tab to the app. Feel free to supplement that data with the airbnb data on scholar:

/class/datamine/data/airbnb

Your tab needs to include the following:

1. A title for your tab
2. An in-app description for the user that explains what your tab does
3. Your tab must accept the two variables (arrival month and country) as inputs, and display at least one new output.
4. The output must be clear and include an appropriate name and description so the user understands how to interpret the results.

These datasets are rich, and can be explored in many different ways. Be creative and have fun! Remember that the end goal is to provide an interface to help your friends plan their vacation.

# ——— Optional Project 4 ———

#python #plotting

## Question 1: plotting with [pygal](http://www.pygal.org/en/stable/documentation/index.html)

*(5 pts)*

First, we must install the pygal package. In scholar, open up a shell and type the following:

python3.6 -m pip install pygal --user

Load the data [here](https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-03-31/brewing_materials.csv) into a pandas dataframe.

Explore the data. As you can see there is information on beer’s brewing material for all months between 2008 and 2017. There are two material types (material\_type): “Grain Products”, and " Non-Grain Products“, which can be of different types (type), like”Hops (dry)“,”Sugar and syrups“, etc. Note that in material\_type the total of grain products and non-grain products, as well as the overall total, have already been calculated per month and year, and can be found in rows that contain the word”Total" (“Total Grain products”, “Total Non-Grain products”, and “Total Used”) under the column type. Additionally, the current number of barrels for corresponding year/month/material can be found in month\_current.

Create a [basic pie chart](http://www.pygal.org/en/stable/documentation/types/pie.html) to plot the brewing material type for a specific month and year – for example ‘August’ of ‘2017’. We want the pieces of the pie chart to show the percentage of barrels produced using the corresponding brewing material (grain products vs non-grain products) on the month of the given year. Make sure to calculate the percentage of barrels produced from the number of barrels (month\_current).

**Important note:** It is not necessary for you to follow the hints in (1a) or (1b), however, they may be helpful as they are designed to walk you through what to do step-by-step.

**Hint(1a):** *Filter the data to include only the month and year you want. Filter the data to include only the material\_type that contains the word “Total” in it. Use the values in month\_current column to calculate the percentage of barrels produced per brewing material type (grain vs non-grain products) per year. You can calculate the percentage by “hand”.*

**Keywords:** *isin, str.contains*

**Hint(1b):** *Use the percentage information from (1a) to create a basic pie chart. Use the command below to show your plot. Make sure you name your pie chart brewing\_material\_pie\_chart.*

from IPython.display import HTML  
html\_pygal = """  
<!DOCTYPE html>  
<html>  
 <head>  
 <script type="text/javascript" src="http://kozea.github.com/pygal.js/latest/pygal-tooltips.min.js"></script>  
 <!-- ... -->  
 </head>  
 <body>  
 <figure>  
 {pygal\_render}  
 </figure>  
 </body>  
</html>  
"""  
  
# your code to create the brewing\_material\_pie\_chart here  
  
HTML(html\_pygal.format(pygal\_render=brewing\_material\_pie\_chart.render(is\_unicode=True)))

**Item(s) to submit:**

* A cell in the Jupyter notebook containing the python code used to create the pie chart using pygal.

## Question 2: race bar chart with [matplotlib](http://matplotlib.org/index.html)

Race bar charts have been gaining attention lately. Examples include [urban population growth](https://app.flourish.studio/@flourish/bar-chart-race) and [Interbrand’s Top Global Brands](https://observablehq.com/@johnburnmurdoch/bar-chart-race). In this question, we will create our own race bar chart for beer production across some US states since 2008.

Load the data [here](https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-03-31/beer_states.csv) into a pandas dataframe. As you can see there is information on the state, year, and the quantity and type of produced beer barrels.

**2a.** *(1 pt)* Organize the dataset. Filter the data to include only rows where state is in the list states\_to\_select provided below. In addition, include only rows where barrels are of type equal to “Bottles and Cans”. Save this dataset to a variable named beer.

states\_to\_select = ['CA', 'OR', 'MI', 'CO', 'VT', 'OH', 'PA', 'MA', 'IL', 'WI', 'IN']

**Keywords:** *isin*

**Item(s) to submit:**

* A cell in the Jupyter notebook containing the python code used to obtain beer dataset.

**2b.** *(1 pt)* Create a (default options) [horizontal bar chart](https://matplotlib.org/3.1.1/gallery/lines_bars_and_markers/barh.html) using matplotlib and our dataset, beer. Filter rows where the year is “2008”. Specifically, finish the code below to plot the amount of produced beer (x-axis) by state (y-axis) for the year “2008”.

import matplotlib.pyplot as plt  
import matplotlib.animation as animation  
  
fig, ax = plt.subplots()  
# Code to create bar chart

**Hint:** *To get the bars organized by the beer production, make sure to sort the values prior to making the plot. To sort the beer dataframe by barrels, use the code:*

beer.sort\_values(by='barrels', ascending=True)

Note that we want to sort the dataframe that contains data from only the year “2008” (not *all* of the data).

**Item(s) to submit:**

* A cell in the Jupyter notebook containing the python code used to create the horizontal bar chart.

**2c.** *(1 pt)* Let’s give our bars some color. Make our horizontal bar graph colored based on the colors\_dict provided below. Use the argument color from the plotting function, and note that it requires a list of colors.

purdue\_colors = ['#5B6870', '#BAA892', '#6E99B4', '#A3D6D7',  
 '#085C11', '#849E2A', '#B46012', '#29A592', '#AD1F65' ,  
 '#B1810B', '#4D4038']  
   
colors\_dict = dict(zip(states\_to\_select,purdue\_colors))

**Hint:** *We can get a list of colors from colors\_dict for ['IN', 'CO', 'MI'] using the following command:*

[colors\_dict[state] for state in ['IN', 'CO', 'MI']]

**Item(s) to submit:**

* A cell in the Jupyter notebook containing the python code used to create the colored horizontal bar chart.

**2d.** *(1 pt)* Complete the function called make\_beer\_bar\_chart. The function accepts an argument called year, and makes a colored horizontal bar chart. The bar chart shows “Bottles and Cans” production per states\_to\_select for the given year. Use the code from (1b) and (1c). Test your function on the year “2008”. It should equivalent to your bar chart from (1c).

fig, ax = plt.subplots()  
  
def make\_beer\_bar\_chart(year):  
 ax.clear() # so positions can change  
 # YOUR CODE HERE  
 ax.text(1, 0.4, year, color='#5D8AA8', size=15, ha='right', weight=800) # text for year  
  
make\_beer\_bar\_chart(2008)

**Item(s) to submit:**

* A cell in the Jupyter notebook containing the modified python code with the complete make\_beer\_bar\_chart function.

**2e.** *(1 pt)* Now let’s make an animation using the completed function make\_beer\_bar\_chart from (2d). Write 1-2 sentences about the resulting animation. Are the results expected? Is there anything you found interesting?

from IPython.display import HTML  
  
fig, ax = plt.subplots()  
animator = animation.FuncAnimation(fig, make\_beer\_bar\_chart, frames=range(2008, 2019))  
# display on your jupyter notebook  
HTML(animator.to\_jshtml())   
  
# You can save your animation as a gif. However, you must have the ffmpeg library or libav-tools installed.  
# animator.save('animator.gif', fps=0.5)

**Item(s) to submit:**

* A cell in the Jupyter notebook with the code above
* A cell in the Jupyter notebook with 1-2 sentences describing your conclusions based on the resulting animation.

## Project(s) Submission:

Submit solutions for all projects using the instructions found in the GitHub Classroom instructions folder on Blackboard.

You do not need to do any of these make-up projects if you are happy with your grades on your top 10 projects. A make-up project would replace whatever your lowest score is (which could be a project you didn’t do).

Submit make-up projects using the following links:

**Optional Project 1:** <https://classroom.github.com/a/hU_7asC1>

**Optional Project 2:** <https://classroom.github.com/a/r35pVz5g>

**Optional Project 3:** <https://classroom.github.com/a/z5VbWrQR>

**Optional Project 4:** <https://classroom.github.com/a/y-75gfky>