hcde410-a2

May 6, 2021

1 Assignment 2: Exploring sources of bias in data

1.1 Bias in data, demystified

All datasets have limitations. Any of these limitations can be a potential source of bias. In the context of data, a "bias" is really just a divergence between what a dataset is supposed to capture about the world–according to the *dataset designer's intention*, or according to the *end-user's expectations*–and what's actually represented in the data.

So biases in data are often highly *contextual*, which can make them subtle and hard to spot. Similarly, it's hard to predict ahead of time what the *consequences* of those biases might be, because it depends on what the data is being used for.

Nevertheless some sources of "bias" in data show up over and over again. Some of these are: - **duplicated data:** the same data shows up multiple times - **incomplete data:** some data is missing from the dataset - **misleading data:** it looks like a piece of data means one thing, but it actually means something different - **unrepresentative data:** the dataset doesn't represent the population it's gathered from OR the population it's intended to model

Any of these sources of bias, unless their properly documented (with a data statement, etc) can have unintended consequences: they cause a researcher who is using the data to reach incorrect conclusions, or cause a machine learning model that is trained on that dataset to make classification errors.

The nature or scale of these consequences can be hard to predict. That's why any time you prepare to use data that you didn't gather yourself, it pays to spend some time exploring the dataset, identifying limitations, and thinking critically about how these limitations might affect your analysis or your machine learning model.

1.2 Introduction to Assignment 2

For this assignment, you'll be working with the Wikipedia Talk Corpus. You have already created a data statement for this dataset in class, so you're an 'expert' on it compared to most people!

In assignment 1, you learned how to process and analyze a dataset created by someone else. In that assignment, we took the dataset at face value: we assumed that there were no major errors, no missing data, nothing that would skew our results. We assumed that the dataset really did accurately capture all the bike and pedestrian traffic on the Burke-Gilman trail over the specified period of time.

In assignment 2, we'll also be analyzing a dataset created by someone else, but this time *we won't assume that the data is complete and correct.* Instead, we will try to identify ways in which the data might be WRONG, and form hypotheses about how the limitations we discover might make it an unsuitable, or at least a potentially problematic, training dataset for a general-purpose hostile speech detector.

There are 16 questions in this assignment, across three sections:

In part 1 of this assignment, you will load one type of Wikipedia Talk Corpus data–the demographics of the crowdworkers who labelled the comments–into your copy of this Jupyter Notebook, and we will walk through a series of data processing steps, with the goal of ending up with a complete and accurate set of data about all of those crowdworkers. In the process, we will discover and discuss several limitations of the dataset.

In part 2 of this assignment, you will generate some basic descriptive statistics about the demographics of the crowdworkers described in that dataset. If you are comfortable in Python, you can perform that analysis in this Notebook. If you aren't as comfortable in Python (yet!), you can perform the analysis in Google Sheets (just be sure to link to that Google Sheet from this notebook, and set the permissions so that your instructors can view it!).

In part 3 of this assignment, you will answer some additional research questions about this dataset. Some of these questions can be answered without writing additional code or analyzing additional data; other questions will require you to combine this dataset another dataset in the corpus. You will have the option to choose whether you want to answer code or no-code questions.

Whether you choose code or no-code question for part 3, you will need to write your responses to the question within this notebook, and submit a link to this notebook for grading.

Part 3 also contains some optional "challenge" questions that you can investigate if you want to (hint: any one of these challenge questions could be the focus of a final course project).

Note: *Since the purpose of this class is to get you comfortable thinking critically (like a researcher), rather than programming perfectly, you won't be graded on your code. You'll be graded on how well you DOCUMENT YOUR PROCESS and REFLECT on the implications of your findings. If you still aren't sure what that means, ask your instructor or TA!*

1.3 Part 1: Cleaning and analyzing the annotator demographic data

For the first part of this assignment, we're going to prepare one set of Wikipedia Talk data–the annotator demographics files–for analysis. In the process we'll perform a few "sanity checks" to make sure we understand what the data means, and know any limitations.

This sort of "data wrangling" is a critical, if sometimes tedious, first step for any quantitative research project.

1.3.1 1.1 Load the data into the notebook

According to the documetation, the worker demographic data for the Wikipedia Talk Corpus is spread across three files: - `toxicity_worker_demographics.tsv` - `aggression_worker_demographics.tsv` - `attack_worker_demographics.tsv`

We will need to combine the data in these three files to come up with our canonical list of workers. **Just like working with data in the real world, you will need to retrieve the files from their source (Figshare) and transfer them into your notebook server**.

- **Toxicity dataset:** data (Note: this is 35Mb!), documentation, questionnaire shown to crowdworkers
- **Aggression dataset:** data (Note: this is 74Mb!), documentation, questionnaire shown to crowdworkers
- **Personal attacks dataset:** data (this is only 92k), documentation, questionnaire shown to crowdworkers

First we'll load each of the annotator datafiles into our Notebook and save them into data structures that's easy to work with. In this case, I'm choosing to save each of these files as a list-of-dictionaries, since that's fairly standard, and it makes it easy to check your work as you go.

By the way: `.tsv` stands for "tab-separated values", and it means that this file is organized into rows and columns, like a spreadsheet, and the data values for each column are separated by "tab" characters.

[1]:
```
#import the csv module, a little code toolkit for working with␣
↪spreadsheet-style data files
import csv
```

The function below will load in a tab-separated (.tsv) file and convert it into a lists-of-dictionaries.

If you don't have much experience with Python, this (and some of the other code in this notebook) might be hard to understand. That's okay! For now, it's most important that you know what it does.

If you have a ***lot*** of experience with Python, the code in this notebook might seem really, really primitive. That's also okay! Remember: in this course we're primarily interested in data, not code. Code is just one of the many tools we use to ask and answer questions about data.

[2]:
```
def prepare_datasets(file_path):
    """
    Accepts: path to a tab-separated plaintext file
    Returns: a list containing a dictionary for every row in the file,
        with the file column headers as keys
    """

    with open(file_path) as infile:
        reader = csv.DictReader(infile, delimiter='\t')
        list_of_dicts = [dict(r) for r in reader]

    return list_of_dicts
```

1.3.2 1.2 Identifying duplicated datafiles

Let's load our three .tsv files into Python and store them as three variables with relevant names, so that we know which is which.

Once we've created these three lists-of-dicts, we will do two things to check our work so far: - we will print the first annotator's demographic data (list index `[0]`) so that we know what the format looks like - we will print the length of each list (the `len` function), to see how many rows is in each file. Each row should correspond to one crowdworker/annotator.

Note: for the cell below to run, your version of these datafiles and folders will need have the same names as the ones below, and your version of this Notebook will need to be stored in the same directory as the three folders that hold the datafiles.

[3]:
```
#load the data from the flat files into three lists-of-dictionaries
toxicity_annotators = prepare_datasets("Toxicity/toxicity_worker_demographics.
 ↪tsv")
print(toxicity_annotators[0])
print(len(toxicity_annotators))

attack_annotators = prepare_datasets("PersonalAttack/attack_worker_demographics.
 ↪tsv")
print(attack_annotators[0])
print(len(attack_annotators))

aggression_annotators = prepare_datasets("Aggression/
 ↪aggression_worker_demographics.tsv")
print(aggression_annotators[0])
print(len(aggression_annotators))
```

```
{'worker_id': '85', 'gender': 'female', 'english_first_language': '0',
'age_group': '18-30', 'education': 'bachelors'}
3591
{'worker_id': '833', 'gender': 'female', 'english_first_language': '0',
'age_group': '45-60', 'education': 'bachelors'}
2190
{'worker_id': '833', 'gender': 'female', 'english_first_language': '0',
'age_group': '45-60', 'education': 'bachelors'}
2190
```

Interesting! This tells us a few things that we didn't know before: 1. it looks like the demographic data matches what's listed in the schema, which is great! 2. it looks like the "toxicity" dataset was annotated by a lot more people (3,591) than the "attack" or "aggression" datasets (2,190) 3. `aggression_worker_demographics.tsv` and `attack_worker_demographics.tsv` seem to contain the same number of workers, and the worker at the beginning of each list has the same ID and demographic data.

Let's dig a little deeper into finding #3. Are the ***last*** entries in both of these lists also identical?

[4]:
```
print(attack_annotators[-1]) # "-1" tells Python to find the last item in any␣
 ↪list
print(aggression_annotators[-1])
```

```
{'worker_id': '3876', 'gender': 'female', 'english_first_language': '1',
'age_group': '30-45', 'education': 'bachelors'}
{'worker_id': '3876', 'gender': 'female', 'english_first_language': '1',
'age_group': '30-45', 'education': 'bachelors'}
```

Yes, the last rows in these two lists are also identical!

And in fact if you had opened the two lists in a text editor or spreadsheet program, you would find that the *aggression and attack .tsv files contain exactly the same data.* By the way, it doesn't say anywhere in the dataset documentation that these two files are identical!

That brings us to our first lesson about bias: watch out for duplicate data!

Consider: *What would have happened if we had just combined these three files and then analyzed the worker demographics? What mistaken conclusions might we have drawn from that?*

Fortunately, now that we know that there is duplicate data we can work around it. Since two files are identical, we only need to use one of them. So from now on, we will ignore `aggression_annotators` entirely.

Since want to remember that `attack_annotators` really refers to both "attack" and "aggression" annotators, we can just rename the variable we're using to store that dataset.

[5]:
```
attack_aggro_annotators = attack_annotators
```

Okay, that looks good. Now we can continue getting our data ready for analysis–while keeping an eye out for additional duplicate data and other "gotchas"!

1.3.3 1.3 Understanding what the properties of your data really mean

Whenever you are working with data that you didn't create, it's very useful to perform some basic sanity-checking to make sure the data actually means what you think it means.

For example, take the `worker_id` field in our datasets.

The schema says that the `worker_id` field contains an "anonymized crowd-worker id" and that this ID is meant to join the worker demographics datafiles with the annotator comments datafiles, so that if we wanted find all of the comments that worker "85" (from above) labelled, we could find her by looking for that ID in each row of `toxicity_annotated_comments.tsv`.

So far, so good. But since we want to combine the toxicity and attack + aggression annotator demographics data into a single dataset, we probably want to know…

1. did any of the values for `worker_id` appear in both datasets?
2. if so, do they correspond to the same person (or at least, a person with matching gender, age group, etc.)?

1.3.4 1.4 Checking for duplicate worker IDs

If we want to see if worker_id means the same thing across datasets, the first step is to see if any of the same worker_ids exist across the two datasets.

To check this, let's first pull all the worker IDs out of each dataset and combine them into a single list. Then we can check that list to see if it contains any duplicate values. If it does, we know that there is at least 1 value for `worker_id` that appears in both datasets.

Note: *We're assuming that there are no duplicate values for `worker_id` within each dataset. (There aren't, I checked). It's a pretty safe assumption though, because worker_id is intended to be a unique key that links the `worker_demographics.tsv` files and the `annotated_comments.tsv` datasets. Can you explain why it would be an issue if there were duplicate values for `worker_id` within an individual dataset?*

```
[6]: #pull the worker ids out of the individual files
     tox_w_ids = [item['worker_id'] for item in toxicity_annotators]
     aa_w_ids = [item['worker_id'] for item in attack_aggro_annotators]


     #create a new list to hold all the ids
     all_w_ids = tox_w_ids + aa_w_ids

     # #combine the two worker_id lists into the new list
     all_w_ids.extend(tox_w_ids)
     all_w_ids.extend(aa_w_ids)
```

```
[7]: #how many worker ids do we have, total?
     #this number should match the total count of toxicity_annotators and␣
      ↪attack_aggro_annotators (3591 + 2190 = 5781)
     print(len(all_w_ids))
```

```
11562
```

```
[8]: #what does our new list look like? Let's print the first five values
     print(all_w_ids[0:5])
```

```
['85', '1617', '1394', '311', '1980']
```

Below is our duplicate-checker function. You pass it a list of values, and it will return "True" if it finds at least 1 duplicate value in that list. Can you figure out how it works?

```
[9]: def determine_dupes(w_ids):
         set_of_ids = set()

         found_a_dupe = False

         for w in w_ids:
             if w in set_of_ids:
                 found_a_dupe = True
             else:
                 set_of_ids.add(w)
         return found_a_dupe
```

```
[10]: #we'll call determine_dupes to check for duplicates in the combined worker id␣
       ↪list
      has_dupes = determine_dupes(all_w_ids)

      #if this prints 'True', that means we found at least one duplicate ID
      print(has_dupes)
```

```
True
```

Hmmm… So it looks like there's at least 1 duplicate! Okay, we'll need to do some additional verification before we can decide what to do with that information.

The next thing we'll do is check how many duplicates there are. We'll write a short script that reads through `all_w_ids` and every time it finds a value that appears more than once, it adds that value to a new list.

Can you figure out how the script below works?

[11]:
```
#create an empty list to hold any duplicate worker_id values we find
dupes = []

#look through the data, if you encounter any value more than once, add it to␣
 ↪our 'dupes' list
for w in all_w_ids:
    if all_w_ids.count(w) > 1:
        dupes.append(w)

#how many values were added to 'dupes'?
print(len(dupes))

#how many worker_ids are present twice in the dataset?
#can you explain why we are dividing the length of "dupes" by 2 in order to␣
 ↪answer that question?
print(len(dupes)/2)
```

```
11562
5781.0
```

Huh, so it looks like 1,850 of the `worker_ids` in our merged dataset are duplicates! That's a lot of duplication, since our list was only 5,781 rows in the first place, including these dupes!

We will definitely need to account for these duplicates before we start analyzing worker demographics. - If the duplicate `worker_ids` have different demographic metadata, we will assume they are different people with the same ID. - If the metadata is identical, then we know we can safely remove the duplicates - If some metadata is identical and some is different… well, let's hope that's not the case!

1.3.5 1.5 Check for duplicate worker demographic metadata

Let's see if it's just the value for `worker_id` that is duplicated across the two datasets (meaning that the these duplicate ids correspond to different workers with different demographics), or if `worker_id` really corresponds to the same people across `toxicity_annotators` and `attack_annotators`.

To do this we'll perform some spot checks, meaning we'll visually compare the demographic data of workers with duplicate IDs, to see if they look like the same worker, or not. Running random 'spot checks' is a really common and useful way of finding systematic issues with your data without having to check every entry.

First, we'll extract 10 random ids from our dupe set. Using a random sample (rather than just looking at the first 10 rows, for instance) helps us be more confident that any patterns we see are really there.

[12]:
```python
#handy Python library that lets you select things randomly from a list
import random
```

[13]:
```python
#convert our 'dupes' list into a set
#bonus: can you explain why we created "dupeset" rather than just grabbing 10␣
 ↪random values from "dupes"?
#hint: in Python, a 'set' is like a list that can only contain a single␣
 ↪instance of any value
dupeset = set(dupes)

#store our random sample of dupes in its own list
dupe_sample = random.sample(dupeset, 10)

#print to confirm everything looks how we expect it to...
print(dupe_sample)
```

```
['140', '785', '1919', '3071', '3942', '2250', '3345', '4025', '519', '1693']
```

Now, we'll use this `dupe_sample` list that we created to pull the corresponding worker demographics from each of our two datasets, using the function below. See if you can figure out how the function works!

[14]:
```python
def worker_id_lookup(annotator_list, dupe_id):
    """
    Accepts: a list of dictionaries
        & a list of known duplicate values for
        the key 'worker_id' in those dictionaries

    If a duplicate value for worker_id is found,
        print the complete dictionary of demographic data for that worker

    """

    for a in annotator_list:
        if a['worker_id'] == dupe_id:
            print(a)

#loop through the duplicate sample list and call our worker_id_lookup function
#to check each dataset for corresponding worker demographic data
for d in dupe_sample:
    worker_id_lookup(toxicity_annotators, d)
    worker_id_lookup(attack_aggro_annotators, d)
```

```
{'worker_id': '140', 'gender': 'female', 'english_first_language': '0',
'age_group': '18-30', 'education': 'hs'}
{'worker_id': '140', 'gender': 'male', 'english_first_language': '0',
'age_group': '18-30', 'education': 'bachelors'}
```

```
{'worker_id': '785', 'gender': 'male', 'english_first_language': '0',
'age_group': '30-45', 'education': 'professional'}
{'worker_id': '1919', 'gender': 'female', 'english_first_language': '1',
'age_group': '45-60', 'education': 'hs'}
{'worker_id': '3071', 'gender': 'male', 'english_first_language': '0',
'age_group': '18-30', 'education': 'professional'}
{'worker_id': '3942', 'gender': 'male', 'english_first_language': '0',
'age_group': '18-30', 'education': 'hs'}
{'worker_id': '2250', 'gender': 'female', 'english_first_language': '0',
'age_group': '30-45', 'education': 'bachelors'}
{'worker_id': '2250', 'gender': 'male', 'english_first_language': '0',
'age_group': '18-30', 'education': 'hs'}
{'worker_id': '3345', 'gender': 'female', 'english_first_language': '0',
'age_group': '45-60', 'education': 'masters'}
{'worker_id': '4025', 'gender': 'male', 'english_first_language': '0',
'age_group': '18-30', 'education': 'hs'}
{'worker_id': '519', 'gender': 'male', 'english_first_language': '0',
'age_group': '30-45', 'education': 'professional'}
{'worker_id': '519', 'gender': 'male', 'english_first_language': '0',
'age_group': '30-45', 'education': 'professional'}
{'worker_id': '1693', 'gender': 'male', 'english_first_language': '0',
'age_group': '30-45', 'education': 'bachelors'}
{'worker_id': '1693', 'gender': 'female', 'english_first_language': '0',
'age_group': '30-45', 'education': 'bachelors'}
```

Aha! It looks like few if any of these `worker_id` values correspond to the same demographic data info across the two datasets.

It's possible that some of the same people labelled both datasets and were assigned different IDs each time. But there's no way for us to determine that with the data we have.

So for now we will assume that these datasets were labelled by two entirely different sets of crowd-workers. Which means we can combine these two datasets into one, with no duplication to mess up our analysis.

1.3.6 1.6 Final preparation of the dataset

Before we start our analysis of worker demographics, let's do two more things:

1. Since we know now that worker ID isn't unique, let's give each worker in our new, combined dataset a **truly** unique ID.
2. While we're at it, let's also add a new field to each worker's demographic dictionary that lists which dataset the worker worked on (toxicity or attack/aggressive).

[15]:
```
def add_dataset_id(list_of_dicts, dataset_ref):
    """
    Accepts: a list of dictionaries & a strig value for "dataset"
    Returns: that list of dictionaries, with the new "dataset" key and the␣
↪specified value
    """
    for w in list_of_dicts:
        w.update({"dataset" : dataset_ref})

    return list_of_dicts
```

[16]:
```
#update our worker demographic datasets with the new key and value
toxicity_annotators = add_dataset_id(toxicity_annotators, "toxicity")
attack_aggro_annotators = add_dataset_id(attack_aggro_annotators, "attack and␣
 ↪aggression")
```

[17]:
```
#did it work?
print(toxicity_annotators[0])
print(attack_aggro_annotators[0])
```

```
{'worker_id': '85', 'gender': 'female', 'english_first_language': '0',
'age_group': '18-30', 'education': 'bachelors', 'dataset': 'toxicity'}
{'worker_id': '833', 'gender': 'female', 'english_first_language': '0',
'age_group': '45-60', 'education': 'bachelors', 'dataset': 'attack and
aggression'}
```

Great. Now we'll use the function below to…

1. combine the two datasets into one
2. assign a truly unique id to each worker

It doesn't matter what this ID is, as long as its unique within the dataset. We'll use sequential IDs, starting at "1" and going up from there.

[18]:
```
def finalize_dataset(list1, list2):

    #combine the two lists into a new one
    combined_list = list1 + list2

    #initialize our counter variable
    new_id = 1

    for w in combined_list:
        #add the new sequential id field, and populate with the current value␣
 ↪of 'counter'
        w.update({"unique_id" : str(new_id)})

        #increment the counter variable by 1, so that the next ID will be one␣
 ↪number higher
        new_id = new_id + 1

    return combined_list
```

[19]:
```
#create our new combined list with sequential IDs
all_annotators = finalize_dataset(toxicity_annotators, attack_aggro_annotators)

#print the list length and a sample, for sanity-checking purposes
print(len(all_annotators))
print(all_annotators[0])
```

```
5781
{'worker_id': '85', 'gender': 'female', 'english_first_language': '0',
'age_group': '18-30', 'education': 'bachelors', 'dataset': 'toxicity',
'unique_id': '1'}
```

It worked! Now we can FINALLY start to analyze this dataset, and find out more about the workers who labelled the Wikipedia Talk Corpus.

1.4 Part 2: Analyzing worker demographics

Now that we have our worker demographic data de-duplicated, combined and uniquely identified, we can start using it to ask and answer research questions!

You are welcome to do this next steps in Python, here in this notebook. But if you aren't super comfortable with Python, you can run the cell below to export this dataset to a .csv file called `a2_all_annotator_demographics.csv` which you can open in Google Sheets and do the analysis there.

Note: if you choose Google Sheets, please title your Google Sheet "A2 worker demographics", share it with your instructor and TA ("can view" or "can edit") and paste a link to it in a Markdown cell below.

[20]:
```
with open('a2_all_annotator_demographics.csv', 'w', encoding='utf-8') as f:
    writer = csv.writer(f)
    #write a header row
    writer.writerow(('unique_id',
                     'worker_id',
                     'gender',
                     'english_first_language',
                     'age_group',
                     'education',
                     'dataset',))
    #loop through our dataset and write it to the file, row by row
    for a in all_annotators:
        writer.writerow((a['unique_id'], a['worker_id'], a['gender'],␣
 ↪a['english_first_language'], a['age_group'], a['education'], a['dataset']))
```

1.4.1 2.1 Questions 1-4: compute descriptive statistics

Please create tables or bar charts that answer the following questions:

- **Q1.** What is the gender distribution of these workers?
- **Q2.** What is the distribution of the workers by first language?

- **Q3.** What is the distribution of these workers by age group?
- **Q4.** what is the distribution of these workers by education level?

If you *do not know* how to create tables in Python, you can use Excel or another tool and upload it to your notebook server. However I would encourage you to try out the Pandas library, which makes it very easy to convert your data into tidy tabular representations. If you do not know how to create charts in Python, you can use an easy library such as Plotly to create charts.

POST YOUR ANSWERS TO QUESTIONS 1-4 HERE.

Use both Markdown and code-formatted cells.

1.4.2 Importing Tools & Getting Data

[21]:
```
#imports
import numpy as np
import pandas as pd
import plotly.express as px


#selected colors
colors = ['purple', 'turquoise']

# data
df = pd.read_csv("a2_all_annotator_demographics.csv")
```

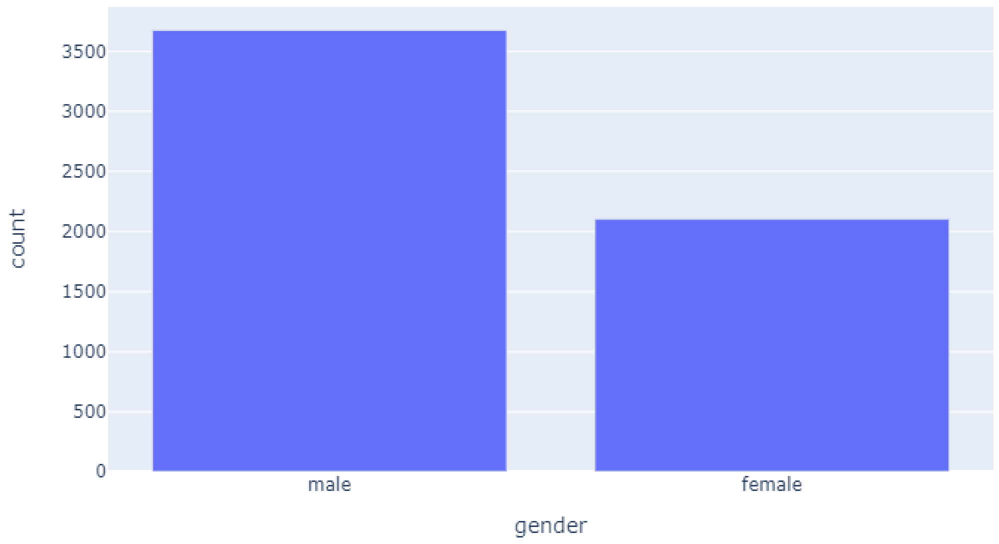
1.5 Answer #1

Q: What is the gender distribution of these workers?

The following code creates a bar chart pertaining to gender through Plotly.

[22]:
```
# data prep for bar graph
male = 0
female = 0 # counters
for item in df['gender']:
    if item == 'female':
        female += 1
    else:
        male += 1
df2 = pd.DataFrame({'gender': ['male', 'female'], 'count': [male, female]})

# bar graph
fig = px.bar(df2, x='gender', y='count') # create figure
print(df2)
fig.show()
```

```
   gender  count
0    male   3678
1  female   2103
```



1.6 Answer #2

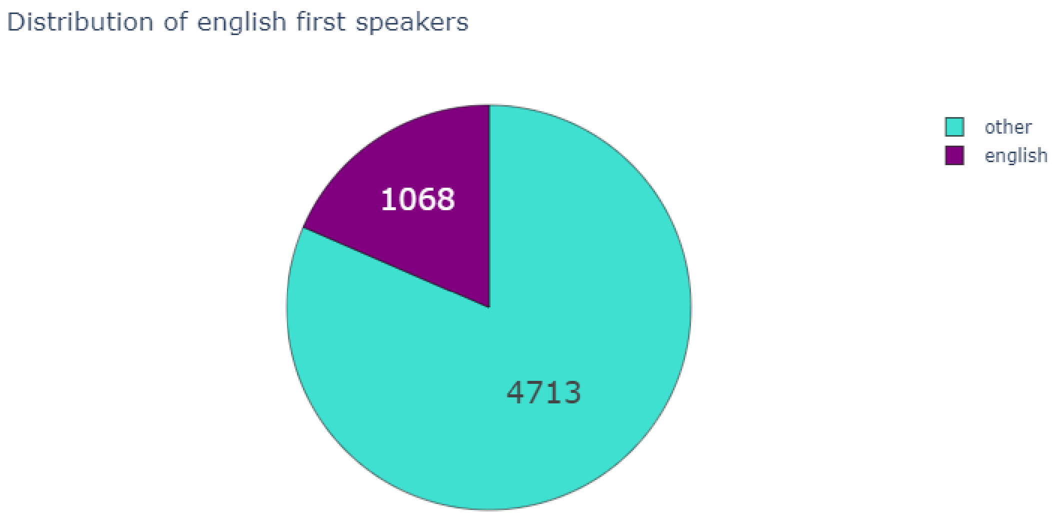
Q: What is the distribution of the workers by first language?

Same as #1, but for a pie chart representing whether english was the annotator's first language.

[23]:
```python
# data prep for pie chart
english = 0
other = 0
for item in df['english_first_language']:
    if item == 1:
        english += 1 # english speaker counter
    else:
        other += 1 # other counter
df3 = pd.DataFrame({"count": [english, other], "type": ['english', 'other']}) #␣
 ↪created a data frame with 'count' and 'type'



# pie chart
fig = px.pie(df3, values='count', names='type', title='Distribution of english␣
 ↪first speakers') #create figure
fig.update_traces(hoverinfo='label+percent', textinfo='value', textfont_size=20,
                  marker=dict(colors=colors, line=dict(color='#000000', width=0.
 ↪5)))
print(df3)
```

```
fig.show()
```

```
   count     type
0   1068  english
1   4713    other
```

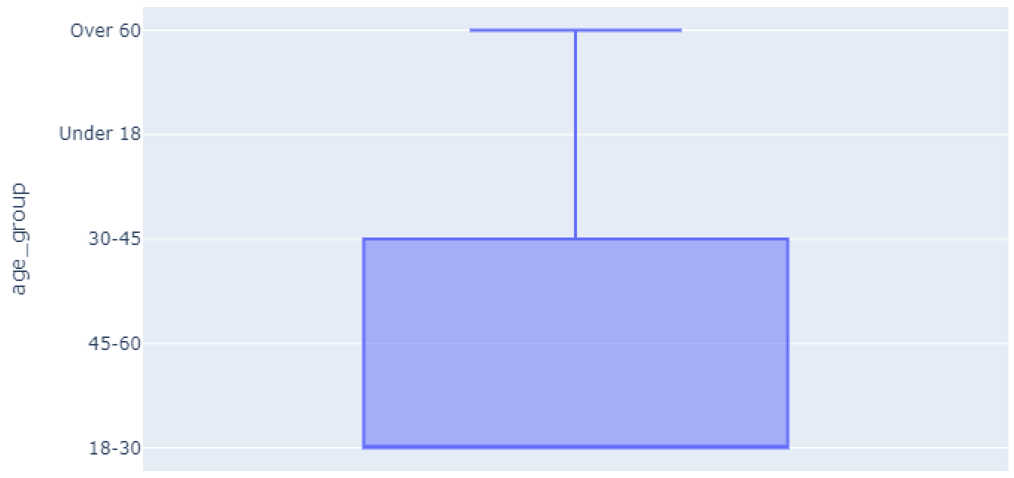


1.7 Answer #3

Q: What is the distribution of these workers by age group?

Same as #1, but a box chart representing the age groups

[24]:
```
# box graph
box = px.box(df, y='age_group') #create figure
box.show()
```



1.8 Answer #4

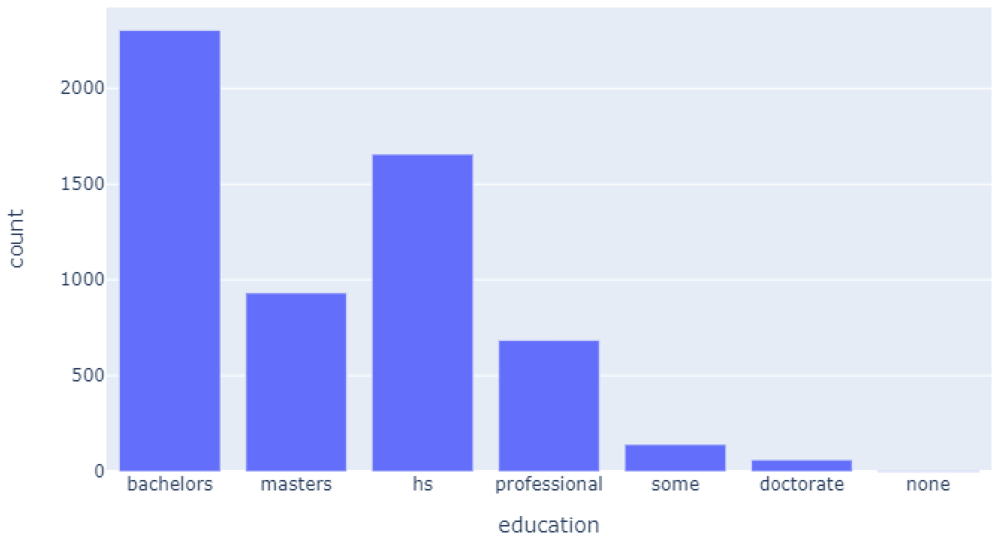
Q: What is the distribution of these workers by education level?

Same as #1, but a bar chart for the education levels

[25]:
```python
# data prep for bar chart
counter = {}
for item in df['education']:
    if item in counter:
        # count!
        counter[item] += 1
    else:
        # add to the dictionary
        counter[item] = 1
df4 = pd.DataFrame({'count': counter.values(), 'education': counter.keys()})
print(df4)
fig = px.bar(df4, x='education', y='count') #create figure
fig.show()
```

```
   count     education
0   2303     bachelors
1    931       masters
2   1657            hs
3    684  professional
4    141          some
```

```
5     61     doctorate
6      4          none
```



1.8.1 2.2 Questions 5-8

Each of the next four questions requires a different kinds of answer.

Questions 5 and 6 have a “correct” answer, but we’ll still give you credit for the “incorrect” answer as long as you document your process well (in Markdown, in this notebook). These questions are intended to get you thinking about how even small errors in data processing can create bias in your data.

Question 6 requires you to re-run some (but not all) of the data processing steps you performed above. Do this by copying the code into NEW cells below this line, and make sure to document each step using Markdown cells or inline (#) comments. You will probably want to change the names of the variables too.

- **Q5.** Analyze all_annotators in Sheets or Jupyter: Do any of the fields in this dataset have missing data? If so, which fields, and what % of the rows contain missing values?
- **Q6.** Build a version of the all_annotators dataset *with removed duplicates*, then re-run the summary statistics from questions 1-4. How have these summary statistics changed?

Questions 7 and 8 don’t have one “correct” answer. These questions are intended to get you thinking about how using this dataset to train a machine learning model might lead to bias in the way that model performs when used as intended. You will need to provide a written answer to each of these questions in 3-4 full sentences. Consider what you know about the workers themselves, the task they were given, and about the purpose of the Perspective API.

Question 7 requires you to do some research of your own (hint: use Google). Question 8 doesn't require any new analysis or research, just critical thinking.

- **Q7.** In general, how do the demographics of these workers compare to those of English-speaking internet users overall? Why would it matter if the worker demographics don't match the demographics of the intended end users of the Perspective API?
- **Q8.** Given what you now know about the Wikipedia Talk Corpus, what issues might arise if it was used to train a machine learning-driven "hostile speech detector" that could be used on any website or social media platform?

POST YOUR ANSWERS TO QUESTIONS 5-8 HERE.

Use both Markdown and code-formatted cells.

[26]:
```
count = len(np.where(pd.isnull(df))[0])
total = len(df)
```

1.9 Answer #5

Q: Analyze all_annotators in Sheets or Jupyter: Do any of the fields in this dataset have missing data? If so, which fields, and what % of the rows contain missing values?

- Pandas.isnull() <= returns where NaaN occurs in the dataframe.

[27]:
```
print("The total percent of missing data from the annotator dataset is␣
 ↪{percent}%".format(count = count, percent = round(count / total * 100, 2)),␣
 ↪'\n')

# what rows is it null in?
# pd.isnull indicates where there are missing values
print(np.where(pd.isnull(df))[1], '\n', 'As shown above, much of the data is␣
 ↪missing on index 4, which is the age_group column.', '\n')
```

```
The total percent of missing data from the annotator dataset is 1.94%

[4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4]
 As shown above, much of the data is missing on index 4, which is the age_group
column.
```

1.10 Answer #6

Q: Build a version of the all_annotators dataset with removed duplicates, then re-run the summary statistics from questions 1-4. How have these summary statistics changed?

- Removed duplicates by only appending the first "worker_id" and rejecting any after. Using mostly hashes and sets, the new dataset has duplicates removed!

1.10.1 Building the new dataset

[28]:
```python
dups = {}
dup_list = [] # now
dup_set = set() # all dups
for data in all_annotators:
    worker_id = data['worker_id'] #find worker_id
    if worker_id not in dups: #first time around
        dups[worker_id] = 1
        dup_list.append(data)
    else: #all other times
        dup_set.add(worker_id)

print("all_annotator's list size is {original_size} which had {duplicates}␣
 ↪number of duplicates, so after removal, the modified list size is now␣
 ↪{current_size}".format(current_size=len(dup_list),␣
 ↪original_size=len(all_annotators), duplicates=len(dup_set)))



# create modified file
with open('a2_modified_annotators_demographics.csv', 'w', encoding='utf-8') as␣
 ↪f:
    writer = csv.writer(f)
    #write a header row
    writer.writerow(('unique_id',
                     'worker_id',
                     'gender',
                     'english_first_language',
                     'age_group',
                     'education',
                     'dataset',))
    #loop through our dataset and write it to the file, row by row
    for a in dup_list:
        writer.writerow((a['unique_id'], a['worker_id'], a['gender'],␣
 ↪a['english_first_language'], a['age_group'], a['education'], a['dataset']))

print("According to the determine_dupes method created Part 1.4, the modified␣
 ↪list is clear of any duplicates \n")
#determine_dupes(worker_ids) # You can uncomment to see that there are no␣
 ↪duplicates! The first value is always kept, while any duplicates were␣
 ↪removed through a hash function.
```

```
all_annotator's list size is 5781 which had 1858 number of duplicates, so after
removal, the modified list size is now 3923
According to the determine_dupes method created Part 1.4, the modified list is
clear of any duplicates
```

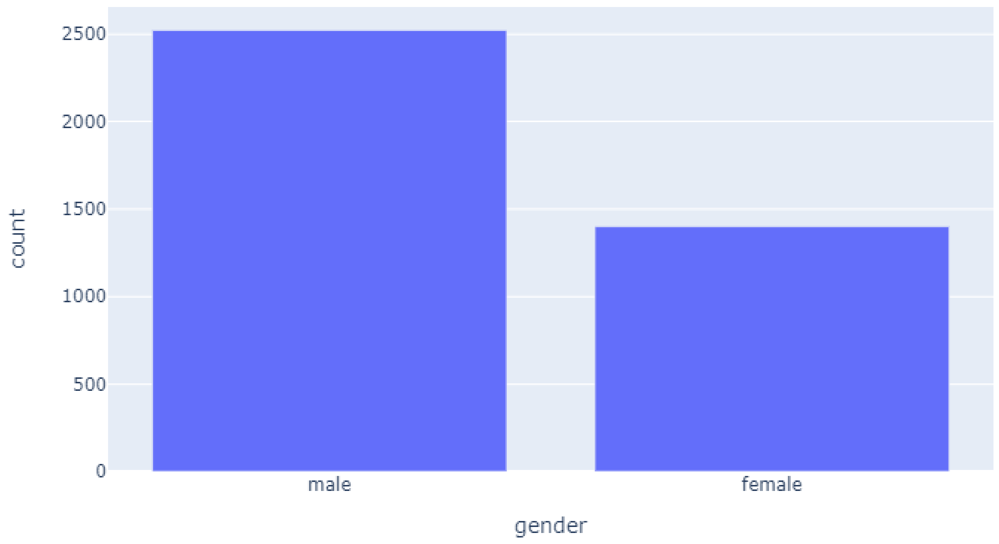
1.10.2 Regraphing with the new list

[29]:
```
# data
df = pd.read_csv("a2_modified_annotators_demographics.csv")
```

1.10.3 Gender Distribution

[30]:
```
# data prep for bar graph
male = 0
female = 0
for item in df['gender']:
    if item == 'female':
        female += 1
    else:
        male += 1
df2 = pd.DataFrame({'gender': ['male', 'female'], 'count': [male, female]})


# bar graph
fig = px.bar(df2, x='gender', y='count') # create figure
print("Gender distribution among the workers", '\n')
print(df2)
fig.show()
```

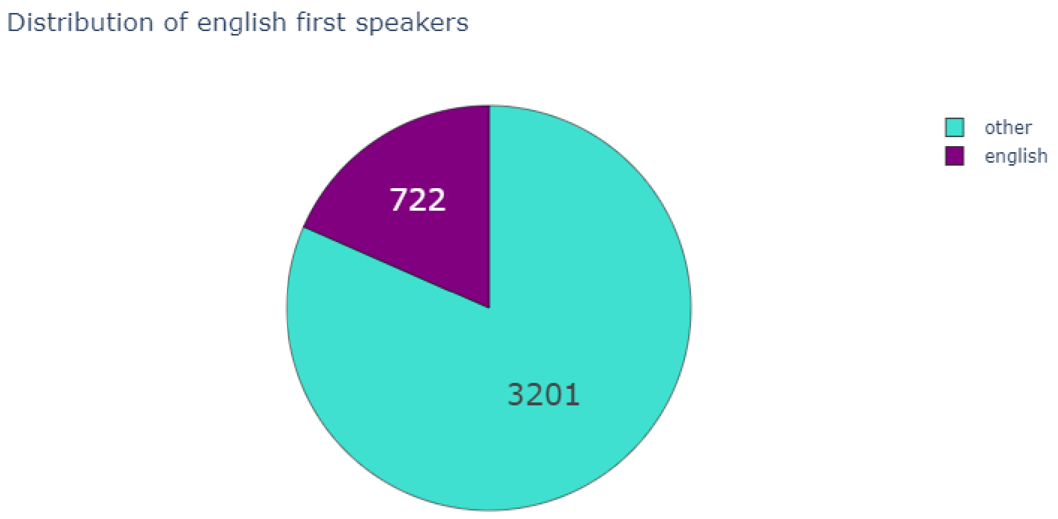
```
Gender distribution among the workers

   gender  count
0    male   2523
1  female   1400
```



1.10.4 Language Distribution

[31]:
```
# data prep for pie chart
english = 0
other = 0
for item in df['english_first_language']:
    if item == 1:
        english += 1 # english speaker counter
    else:
        other += 1 # other counter
df3 = pd.DataFrame({"count": [english, other], "type": ['english', 'other']}) #␣
 ↪created a data frame with 'count' and 'type'



# pie chart
fig = px.pie(df3, values='count', names='type', title='Distribution of english␣
 ↪first speakers') # create figure
fig.update_traces(hoverinfo='label+percent', textinfo='value', textfont_size=20,
                  marker=dict(colors=colors, line=dict(color='#000000', width=0.
 ↪5)))
print('Pie chart to represent the distribution of english first speakers', '\n')
print(df3)
fig.show()
```

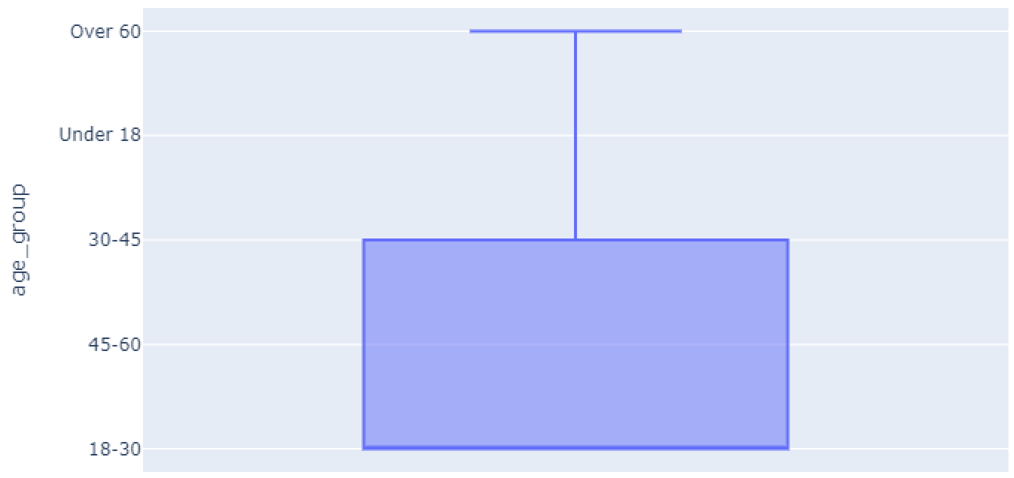
```
Pie chart to represent the distribution of english first speakers

   count     type
0    722  english
1   3201    other
```



1.10.5 Age Distribution

[32]:
```python
# box graph
box = px.box(df, y='age_group') # create figure
print("Age group distribution", '\n')
box.show()
```

```
Age group distribution
```

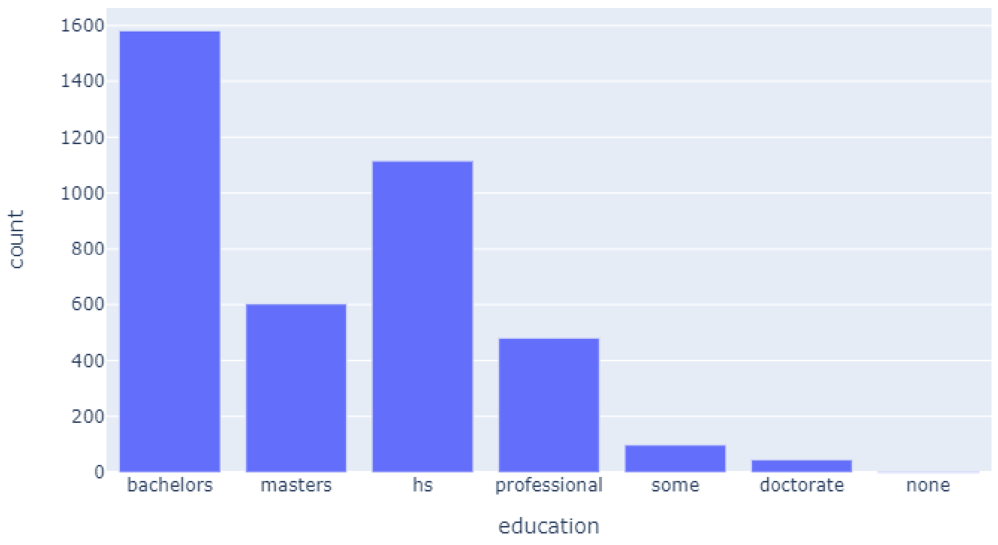


1.10.6 Education Distribution

[33]:
```python
# data prep
counter = {}
for item in df['education']:
    if item in counter:
        # do something
        counter[item] += 1
    else:
        # add to the dictionary
        counter[item] = 1
df4 = pd.DataFrame({'count': counter.values(), 'education': counter.keys()})
print('Distribution of the workers dependent on education', '\n')
print(df4)
fig = px.bar(df4, x='education', y='count') # create figure
fig.show()
```

```
Distribution of the workers dependent on education

   count    education
0   1580    bachelors
1    602      masters
2   1115           hs
3    480  professional
4     98         some
5     45    doctorate
```

```
6       3             none
```



1.10.7 How does it compare to the graphs earlier?

After rerunning the graphing operations, there doesn't seem to be much change in the distribution of the field values. Gender, education, first language, and the age group seemed to maintain a similar percentage to the previous data.

1.11 Answer #7

Q: In general, how do the demographics of these workers compare to those of English-speaking internet users overall? Why would it matter if the worker demographics don't match the demographics of the intended end users of the Perspective API?

It is important to have worker demographics that match the demographics of the intended users to weed out as much biases as possible. Because if they happen to not match, the training of the API may result in less accurate predictions.

To best compare the demographics of the workers to english-speaking intenet users, I used data from the below resource and used numbers pertaining to the broadband avaliability by gender, education, race, and age.

Age (*Heavier on younger age_groups*) - 18-29: 70% - 30-49: 86% - 50-64: 79% - 65+: 64%

Race (*Heavier on white, compare to other races*) - White: 80% - Black: 71% - Hispanic: 65%

Gender (*Even*) - Men: 77% - Women: 77%

Education (*Lower bracket for anything below a college degree, Heavier on people with any college* *experience*) - Less than high school graduate: 59% - High school graduate: 59% - High school or less: 59% - Some college: 80% - College graduate: 94%

1.11.1 How do they compare?

The demographics of the workers had a pretty good representation of the market, except a couple problems.

- Gender distribution should not be as wide, since males and females use the internet an equal amount.
- Though race doesn't specifically mention language, it could definitely influence the first language of the individual. Our dataset contained a lot more people in the "other" category, which may not represent enough english spreaking individuals. 722 vs 3201 is a rather large difference.
- The distribution among education and age represented the general internet user base pretty well!

Resource: https://www.pewresearch.org/internet/fact-sheet/internet-broadband/?menuItem=6d2e5a1d-0fea-4cff-84ef-5999713abe5e

1.12 Answer #8

Q: Given what you now know about the Wikipedia Talk Corpus, what issues might arise if it was used to train a machine learning-driven "hostile speech detector" that could be used on any website or social media platform?

- Something to consider would be the imbalance between male to female annotators. This could lead to gender bias imbedded into the hostile speech detector.
- Another issue would be the underreporting of toxic, aggressive, and attacking comments. Partly due to the fact that the majority of annotators didn't speak English as their first language, so they may assign lower toxicity scores.
- The other distributions (education and age) are rather even, which could discourage inherit biases!

An important statement in the question I think is: "that could be used on any website or social media platform". Since the technology is for general use, having an even distribution is important in managing the inherit biases.

1.13 Part 3: Digging deeper

1.13.1 3.1 Questions 9-16

Below is a list of additional questions that you should be able to answer, based on what you've done today.

You may answer EITHER the "code" questions (Q9-Q12) OR the "no code" questions (Q13-Q16).

If you don't feel very comfortable with Python yet, choose the "no code" questions.

- ***For the "code" questions*** you will need to load `toxicity_annotations.tsv` into Python and join that dataset with `all_annotators` on `worker_id`.

- ***For the "no code" questions*** you will need to load the file `toxicity_labelled_comments_3k_sample.csv` into Google Sheets. This file contains the text of 3000 comments that were annotated for toxicity, as well as the toxicity score that each worker gave to these comments. The file is a random sample taken from the ~1.5 million comments contained in `toxicity_annotated_comments.tsv`

Note: answering question 13-16 below ***will require reading some comments that contain offensive speech!*** If you do not want to be exposed to offensive speech, either answer the "code" questions instead, or reach out to your Instructor or TA and ask for an alternate activity.

Code questions

- **Q9.** What % of *workers* who labelled the toxicity dataset do we have demographic data for?
- **Q10.** What % of the *comments* in the toxicity dataset were labelled by male vs. female-identified workers?
- **Q11.** What % of the *comments* in the toxicity dataset were labelled by people for whom English is NOT their first language?
- **Q12.** Based on your findings from questions 9-11, how might this dataset present a biased view of "toxicity"? How would you expect such biases to impact how the Perspective API performs?

No-code questions Before you answer the questions below, read through at least 20 comments from `toxicity_labelled_comments_3k_sample.csv` that were labelled "toxic" (-1 or -2), 20 that were labelled "non-toxic" (1 or 2), and 20 that are labelled "neutral" (0). Note in your spreadsheet whether you agree or disagree with the labeller's judgement.

- **Q13.** Pick 2-3 examples of comments where you disagreed with the labeller about the toxicity of a comment. Why did you disagree? Why do you think that the labeller might have labelled these comments the way they did?
- **Q14.** Pick 2-3 examples of comments where you don't understand what the commenter was saying, and therefore had a hard time classifying as toxic or non-toxic. What additional context or information would you need in order to be confident in your judgement about the toxicity of this comment?
- **Q15.** Read through the instructions and labelling options given to the crowdworkers. How could the way these instructions were written have made it difficult for crowdworkers to accurately label comments as toxic or non-toxic? If you were going to run a labelling campaign like this one yourself, how would you change these instructions or labelling options to help the crowdworkers make more accurate or consistent judgements about toxicity?
- **Q16.** Based on your findings from questions 13-15, how might this dataset present a biased view of "toxicity"? How would you expect such biases to impact how the Perspective API performs as a general-purpose hostile speech detector?

POST YOUR ANSWERS TO QUESTIONS 9-16 HERE.

Use both Markdown and code-formatted cells.

1.13.2 Merging the toxicity annotation file with all annotators.

- Prepare the data
- Count the total workers of toxicity annotators and all annotators

- Default dictionary to merge the data together on the worker_id
- Output a CSV file for the merged data
- Count the rows

[34]:
```
# imports
from collections import defaultdict

# prepare dataset
toxicity_annotators = prepare_datasets("Toxicity/toxicity_annotations.tsv")


# count the total workers of each dataset
toxicity_w = set()
all_w = set()
for item in toxicity_annotators:
   toxicity_w.add(item['worker_id'])

for item in all_annotators:
   all_w.add(item['worker_id'])


# code from https://stackoverflow.com/questions/5501810/
 ↪join-two-lists-of-dictionaries-on-a-single-key
# adding to a default dictionary by going through the two lists (toxic and␣
 ↪annotators).
# Within there, select each variable and add to a specific worker_id within the␣
 ↪dictionary.
# total length is 4301, which essentially got rid of duplicate worker_ids␣
 ↪through the update function.
d = defaultdict(dict)
for l in (all_annotators, toxicity_annotators):
    for elem in l:
        d[elem['worker_id']].update(elem)
l3 = list(d.values())

print("Outputting a2_all_plus_toxicity_annotators.csv... \n")
print("After using a set to count the total toxicity annotators by their␣
 ↪worker_id, the toxicity annotators contained {keys} unique workers.".
 ↪format(keys=len(toxicity_w)))
print("While the all_annotators list contained a total of {workers}.".
 ↪format(workers=len(all_w)))
print("Since the toxicity annotators contained more unique worker_ids, the␣
 ↪outputted modified annotators should contain the same number of keys as␣
 ↪toxicity annotators.")

# create modified csv
with open('a2_all_plus_toxicity_annotators.csv', 'w', encoding='utf-8') as f:
    writer = csv.writer(f)
    #write a header row
    writer.writerow(('worker_id',
                     'gender',
                     'english_first_language',
                     'age_group',
                     'education',
                     'dataset',
                     'unique_id',
                     'rev_id',
                     'toxicity',
                     'toxicity_score',
                    ))


    #loop through our dataset and write it to the file, row by row
    # have to check if the key exists everytime, if not, None.
    for a in l3:
        worker_id = a['worker_id']
        gender = a['gender'] if 'gender' in a else None
        e_f_l = a['english_first_language'] if 'english_first_language' in a␣
 ↪else None
        a_g = a['age_group'] if 'age_group' in a else None
        education = a['education'] if 'education' in a else None
        dataset = a['dataset'] if 'dataset' in a else None
        u_id = a['unique_id'] if 'unique_id' in a else None
        r_id = a['rev_id'] if 'rev_id' in a else None
        toxicity = a['toxicity'] if 'toxicity' in a else None
        t_s = a['toxicity_score'] if 'toxicity_score' in a else None
        writer.writerow((worker_id, gender, e_f_l, a_g, education, dataset,␣
 ↪u_id, r_id, toxicity, t_s))

# counting the total rows in the csv produced above.
file = open('a2_all_plus_toxicity_annotators.csv')
reader = csv.reader(file)
lines = len(list(reader)) - 1

print("The outputted file contained {lines} lines... Exactly what was expected".
 ↪format(lines=lines), '\n')
```

```
Outputting a2_all_plus_toxicity_annotators.csv…

After using a set to count the total toxicity annotators by their worker_id, the
toxicity annotators contained 4301 unique workers.
While the all_annotators list contained a total of 3923.
Since the toxicity annotators contained more unique worker_ids, the outputted
modified annotators should contain the same number of keys as toxicity
```

```
annotators.
The outputted file contained 4301 lines… Exactly what was expected
```

1.14 Answer #9

Q: What % of workers who labelled the toxicity dataset do we have demographic data for?

- Pandas.DataFrame.isna().sum() <= returns the NaaN count from each column of the dataframe

[35]:
```
# create data frame for the csv produced above
df = pd.read_csv("a2_all_plus_toxicity_annotators.csv")
print("\n", "According to the table below, we are missing data for about 378 of␣
↪the workers in our dataset", '\n', 'Meaning we are missing about␣
↪{percentage}% of demographic data on workers'.format(percentage= round((378.
↪0/lines *100), 2)))
print(df.isna().sum(), '\n')
```

```
 According to the table below, we are missing data for about 378 of the workers
in our dataset
 Meaning we are missing about 8.79% of demographic data on workers
worker_id                  0
gender                   378
english_first_language   378
age_group                447
education                378
dataset                  378
unique_id                378
rev_id                     0
toxicity                   0
toxicity_score             0
dtype: int64
```

1.15 Answer #10

Q: What % of the comments in the toxicity dataset were labelled by male vs. female-identified workers?

Count the males and females where the key exists

[36]:
```
# tickers to count the male, female, and total (just in case)
male = 0
female = 0
total = 0
for item in l3:
    if 'gender' in item: # only add if the item contains a gender key
        gender = item['gender']
        if gender == 'female':
            female += 1
        else:
            male += 1
        total += 1

print("Out of {total} comments which have gender labeled, {male} were labeled␣
 ↪by males, while {female} were labeled by females.".format(total=total,␣
 ↪male=male, female=female))
print("In terms of percentage, males labeled {male}% while females labeled␣
 ↪{female}%.".format(male=round(male/total * 100, 2), female=round(female/
 ↪total * 100, 2)), '\n')
```

```
Out of 3923 comments which have gender labeled, 2454 were labeled by males,
while 1469 were labeled by females.
In terms of percentage, males labeled 62.55% while females labeled 37.45%.
```

1.16 Answer #11

Q: What % of the comments in the toxicity dataset were labelled by people for whom English is NOT their first language?

Same as #10 but for whether english was their first language

[37]:
```
# Tickers to count the people who spoke english as their first language, and␣
 ↪others.
not_fl = 0
is_fl = 0
total = 0
for item in l3:
    if 'english_first_language' in item: # only add if the item contains a␣
 ↪english_first_language key
        fl = item['english_first_language']
        if fl == '1':
            is_fl += 1
        else:
            not_fl += 1
        total += 1

print("{percentage}% of the comments were labeled by people whom English is not␣
 ↪their first language".format(percentage=round(not_fl/total * 100, 2)))
```

```
81.29% of the comments were labeled by people whom English is not their first
language
```

1.17 Answer #12

Q: Based on your findings from questions 9-11, how might this dataset present a biased view of "toxicity"? How would you expect such biases to impact how the Perspective API performs?

There are multiple potential issues regarding the dataset.

- Missing demographics for 8% of the sample. The missing data could misrepresent the data from the selected sample. Analyzing biases will be more difficult due to the missing information. I'm not sure what would be an appropriate amount of missing data, but 8% seems rather steep.
- Though the males to female ratio is not horrible, getting the ratio as close to 1:1 would be beneficial in discouraging bias.
- Lastly, the biggest issue would be the 81% of individuals who stated English was not their first language. This is a problem due to the fact that they are reading English responses, and could be further disconnected from cultural understandings of toxicity, aggression, and attacks.

1.18 Conclusion

Based on the analysis done on the dataset, the annotator's could have an impact on the dataset due to the inbalances in distribution among gender and first language. I think the annotator dataset could benefit from having at least 95% of the demographics populated for the annotators and getting as close to 1:1 in gender would be ideal as well. In addition to the dataset's imbalances, it also contained duplicates. Due to the duplicates in the dataset, the impact of whether I chose to keep the original or duplicates in worker_id could've had an impact on the demographical statistics that were produced, which could influence the accuracy of the analysis and should be kept in consideration. Overall, I believe that my analyzation shows there are high possibilities of inherit biases, and further analysis should be done regarding the imbalances into finer detail.

1.18.1 3.2 Challenge questions (optional)

The questions below are optional; you don't need to answer them to receive full credit for this assignment.

If you'd like to explore additional analyses of this dataset, here are a few questions to get you started!

- ***Challenge #1.*** Are female-identified workers more likely to label a comment as "toxic" than male identified workers?
- ***Challenge #2.*** Are workers with a higher level of education more consistent in their "toxicity" labelling–in other words, do they tend to agree with other labellers more often? (remember that, according to the documentation, every comment was labelled by at least 10 crowdworkers).
- ***Challenge #3.*** What are the words most frequently associated with toxic comments? (perhaps focus on comments where most or all of the workers agree are toxic)
- ***Challenge #4.*** What are the most polarizing comments–comments that some workers consider toxic, and others consider non-toxic? What about these comments made them so polarizing, or hard to classify?
- ***Challenge #5.*** Answer some or all of the homework questions above for one the other datasets ("attack" and "aggressive"). How are the potential sources of bias for these datasets

the same, or different, than for the toxicity dataset? Would you “trust” a machine learning model based on these datasets more or less than one trained on the “toxicity” dataset? Why?

OPTIONAL: POST YOUR ANSWERS TO THE CHALLENGE QUESTIONS HERE.

Use both Markdown and code-formatted cells.