Disclosure Review Examples & Exercises

This notebook provides you with information on how to prepare research output for disclosure control. It outlines how to prepare different kind of outputs before submitting an export request and gives you an overview of the information needed for disclosure review. *Please read through the entire notebook because it will separately discuss all outputs that will be flagged in the disclosure review process.*

In [ ]:
```
# Load packages
%pylab inline
import os
import pandas as pd
import numpy as np
import psycopg2
import seaborn as sns

import matplotlib.pyplot as plt
%matplotlib inline
matplotlib.style.use('ggplot')
```

In [ ]:
```
# database interaction imports
from pyathenajdbc import connect
```

In [ ]:
```
conn = connect(s3_staging_dir = 's3://usda-iri-2019-queryresults/',
               region_name = 'us-gov-west-1',
               LogLevel = '0',
               workgroup = 'workgroup-iri_usda')
```

General Remarks on Disclosure Review

Files you can export

In general, you can export any kind of file format. However, most researchers typically export tables, graphs, regression outputs and aggregated data. Thus, we ask you to export one of these types, which implies that every result you would like to export needs to be saved in either .csv, .txt or graph format.

Jupyter notebooks are only exported to retrieve code

Unfortunately, you can't export results in a Jupyter notebook. Doing disclosure reviews on output in Jupyter notebooks is too burdensome for us. Jupyter notebooks will only be exported when the output is deleted for the purpose of exporting code. **This does not mean that you won't need your Jupyter notebooks during the export process.**

Documentation of code is important

During the export process, we ask you to provide the code for every output you would like to export. It is important for the ADRF staff to have the code to better understand what you exactly did. Understanding how research results are created is important in understanding your research output. Thus, it is important to document every step of your analysis in your Jupyter notebook.

General rules to keep in mind

A more detailed description of the rules for exporting results can be found on the class website. This is just a quick overview. You should go to the class website and read the entire guidelines (link below) before preparing your files for export.

- The disclosure review is based on the underlying observations of your study. **Every statistic** you want to export should be based on at least three individual data points, and you must show the disclosure review team that every statistic you wish to export is based on at least three individual data points by providing counts in your input file.
- Document your code so the reviewer can follow your data work. Assessing re-identification risks highly depends on the context. Therefore, it is important that you provide context info with your analysis for the reviewer.
- Save the requested output with the corresponding code in your input and output folder. Make sure the code is executable. The code should exactly produce the output you requested.
- If you are exporting powerpoint slides that show project results, you have to provide the code which produces the output in the slide.
- Please export results only when they are final and you need them for your presentation or final project report.

Documentation link: adrf.readthedocs.io/en/latest/export_of_results/guidelines.html#documentation)

IRI-Specific Requirements

As mentioned in class, IRI has its own requirements to be able to release statistics generated from their datasets. We will cover these policies more extensively later in this notebook, but here is a summary of the aspects that will cause you to fail the disclosure review, listed for your convenience:

- Micro data exports
- Anything that identifies a particular product, brand, manufacturer, store, or retailer, especially sales volume or market share
- De-identified data for a particular product, brand, store, or retailer that could be easily re-identified, such as
  - Georgraphic/channel combinations, e.g. only one mass merchandiser in a particular MSA
  - In a highly-concentrated industry, sales volume by a de-identified manufacturer could still be identifying for those in the industry
  - Anything where the cell results are only drawn from one entity (product, brand, store, or retailer)
- Specific UPC descriptions
- Unweighted demographic makeup of the panel
- Household-level data

Disclosure Review Walkthrough

We will use the provided IRI data to construct our statistics we are interested in and prepare them in a way so we can submit the output for disclosure review. Here, we will use code to find an estimate of 100% whole wheat bread purchases for WIC participants in 2016. This code will be a slight adaptation from the machine learning preparation (04_01_ML_Data_Prep.ipynb) notebook.

To calculate an estimate of 100% whole wheat bread expenditures by all WIC households in 2016, we will create a data table where a row corresponds to a purchase in the `trip_all` table. From there, we can easily aggregate the sums to find the estimate.

For your viewing pleasure, the following cells display the code we used to create two tables in the `iri_usda_2019_db` database, `disclosure_purchase` and `disclosure_final`. `disclosure_purchase` contains all 100% whole wheat bread purchases in 2016 with additional product details (reasoning will be provided for each variable's inclusion), and `disclosure_final` contains a subset of `disclosure_purchase` to just include the purchases for 2016 WIC participants with sufficient purchasing data and their corresponding sample weights.

```
create table iri_usda_2019_db.disclosure_purchase
    with(
    format = 'Parquet',
    parquet_compression = 'SNAPPY'
    )
    as
    select t.panid, t.dollarspaid, t.coupon, t.upc, t.storename, p.manufactu
rer, p.brand
    from iri_usda.pd_pos_all p, iri_usda.trip_all t
    where p.upc = t.upc and t.year = '2016' and p.upcdesc like '%100% WHOLE
 WHEAT%'
```

```
create table iri_usda_2019_db.disclosure_final
    with(
    format = 'Parquet',
    parquet_compression = 'SNAPPY'
    )
    as
    select p.*, d.projection61k
    from iri_usda.demo_all d
    left join iri_usda_2019_db.disclosure_purchase p
    on d.panid = p.panid
    where d.wic_june = 1 and d.year = '2016' and d.projection61k > 0 and p.p
anid is not null
```

Pull data

Let's see what `disclosure_final` looks like. Keep in mind that you cannot include any micro data outputs (i.e. `.head`). However, we will use the command, along with `df.info()` just to check our dataframe to give you a sense of the contents of `df`. We will also use `df.describe()`, which doesn't directly display micro data. However, **you cannot include the outputs of a regular .describe() command since the minimum, maximum and quartiles may represent individual rows.**

In [ ]:
```python
# Get data
query = """
select *
from iri_usda_2019_db.disclosure_final
"""

df = pd.read_sql(query, conn)
```

In [ ]:
```
# Check dataframe
df.head()
```

In [ ]:
```
# another way to check dataframe
df.info()
```

In [ ]:
```
# check basic stats of df
df.describe()
```

Data Exploration For Estimation

Now, let's find a dollar amount estimate of 100% whole wheat bread expenditures for WIC households in 2016. Recall that we can calculate the total cost of a product by subtracting `coupon` from `dollarspaid`. Thus, we need to start by creating a column that finds this difference for each product purchase.

In [ ]:
```
# initialize total_cost column
df['total_cost'] = df['dollarspaid'] - df['coupon']

#confirm total_cost is calculated properly
df.head()
```

If you wanted to include the distribution of the `total_cost` category by in a numerical summary, you could not used the outputs from `.describe()`, as mentioned above. Instead, you would have to create *weighted* fuzzy quartiles to represent the 25th, 50th and 75th quartiles (and any others you'd want to include). Let's walk through code to create these fuzzy quartiles. We will use the `.quantile()` function to find the true values for some quantiles. First, though, we need to find the amount of money each household spent on 100% whole wheat bread in the year and include their weights.

In [ ]:
```
# find total costs by household
temp = pd.DataFrame(df.groupby(['panid'])['total_cost'].sum())

#find projection61k corresponding to each household
# need to drop duplicates for df so it doesn't keep grabbing projection6
1k for each row in df
weights_df = temp.merge(df[['panid', 'projection61k']].drop_duplicates
(), 'right', on = 'panid')
weights_df.head()
```

In [ ]:
```
# simulate weighted dataframe by repeating the total_cost the amount of
 the weight for each household
weighted_cost = np.repeat(weights_df['total_cost'], weights_df['projecti
on61k'])
#see first few
weighted_cost[0:10]
```

In [ ]:
```
# assign true quantiles around 25, 50 and 75 to true
true = weighted_cost.quantile([.20, .30, .45, .55, .70, .80])
# create list of all the fuzzy quantiles you want to calculate
var = ['fuzzy_25', 'fuzzy_50', 'fuzzy_75']
```

In [ ]:
```
# find values for the fuzzy quantiles
fq_25 = str((true[.20] + true[.30])/2)
fq_50 = str((true[.45] + true[.55])/2)
fq_75 = str((true[.70] + true[.80])/2)

#save values in a second list of corresponding values
val = [fq_25, fq_50, fq_75]
```

In [ ]:
```
# save in pandas dataframe
fuzzy = pd.DataFrame(val, var)
fuzzy[0]
```

To export these fuzzy quartiles as a csv, you can use the `to_csv` function and designate the file path and the name, which needs to end in .csv. Here, we will call the csv `fuzzy_statistic1` , since it is the first statistic we wish to export.

> You just need to switch `benjaminfeder` in the file path to your username in order to save the csv in your home folder.

In [ ]:
```
fuzzy.to_csv('/nfshome/benjaminfeder/fuzzy_statistic1.csv')
```

As proof that the underlying counts for the number of households, products, stores, brands and manufacturers were at least three for these fuzzy statistics, we can save the following cell as a csv `counts_statistic1` to designate that these counts correspond to our `fuzzy_statistic1` csv file. We can eventually include this csv file in our input folder for our export review.

In [ ]:
```
# create csv of all counts we need
data = {'n_unique_households': [df['panid'].nunique()], 'n_unique_prod':
[df['upc'].nunique()],
        'n_unique_store': [df['storename'].nunique()], 'n_unique_brand':
[df['brand'].nunique()],
        'n_unique_manufacturer': [df['manufacturer'].nunique()]}
counts = pd.DataFrame(data)
counts.to_csv('/nfshome/benjaminfeder/counts_statistic1.csv')
```

We can also plot a histogram of the distribution of amount of money spent on 100% whole wheat bread in our weighted sample. Luckily, we can use our `weights_df` dataframe to do this. The plot call will have three outputs, which will we assign accordingly: the counts of the bin sizes, the edges of the bins, and the actual graphical image.

```
In [ ]: #plot histogram
        counts, edges, graph = plt.hist(weights_df['total_cost'], weights = weig
        hts_df['projection61k'])
```

Here, given the default bin size of 10, we are not sure if each of these bins has at least three entries. To makes sure, we can check the `counts` variable.

```
In [ ]: counts
```

For our weighted histogram, all the counts are at least three. Now let's check to see if the same holds for the unweighted histogram, because we need to make sure that each bin contains at least three unweighted households.

```
In [ ]: # plot unweighted histogram
        counts, edges, graph = plt.hist(weights_df['total_cost'])
```

```
In [ ]: # check counts
        counts
```

Here, we see that not all of our bin sizes are at least 3. Thus, we can manipulate our bins so that each bin contains at least three entries. To do so, we will combine the last four bins with help from our `edges` variable.

> Note: There are other ways to adjust the bins. You can play around with the number of bins, drop outliers, or peform other manipulations. This is just one example.

```
In [ ]: # look at edges so we can combine the last four
        edges
```

```
In [ ]: # update edges to combine last four bins
        counts, edges, graph = plt.hist(weights_df['total_cost'], bins = [XX, XX
        , XX, XX, XX, XX, XX, XX])
```

```
In [ ]: # confirm counts are okay
        counts
```

Now, lets save the unweighted counts and the resulting histogram so we can place these as evidence in our input folder before adding these bin changes to the weighted histogram.

```
In [ ]: # Save the barchart
        plt.hist(weights_df['total_cost'], bins = [XX, XX, XX, XX, XX, XX, XX, X
        X])
        plt.savefig('/nfshome/benjaminfeder/unweighted_hist.pdf')
```

```
In [ ]: # Save the counts of the unweighted hist
        pd.DataFrame(counts).to_csv('/nfshome/benjaminfeder/unweighted_hist_coun
        ts.csv')
```

Finally, for the histogram, we can update our weighted histogram with the edges from the unweighted histogram and save it so we can add it to our output folder later.

```
In [ ]: # weighted histogram with updated bin edges
        plt.hist(weights_df['total_cost'], weights = weights_df['projection61k'
        ], bins = [XX, XX, XX, XX, XX, XX, XX, XX])
        plt.savefig('/nfshome/benjaminfeder/weighted_hist.pdf')
```

As another option, instead of creating a histogram, you can also release a density plot using the `distplot()` function from `seaborn`. The advantage of releasing a density plot here is that you do not need to reveal counts, but it also may not be as descriptive as a histogram. To create a weighted density plot, we can use the variable `weighted_cost` we created before.

> You cannot use the `hist = True` argument.

```
In [ ]: sns.distplot(weighted_cost, hist=False)
        plt.savefig('/nfshome/benjaminfeder/densityplot.pdf')
```

Calculating The Estimate

Now, we can find our estimate for each household when weighted by multiplying `total_cost` by `projection61k` for each purchase in our original dataframe, `df`.

```
In [ ]: df['weighted'] = df['total_cost'] * df['projection61k']
```

```
In [ ]: # take sum of weighted
        print('WIC households spent approximately ${:,.2f} in 2016 on 100% whole
        wheat products.'.format(sum(df['weighted'])))
```

You should be able to safely export this statistic since it is generated using the weights. **You cannot release any statistics, other than counts, for unweighted data.** However, we still need to generate a few more confirmations before this statistic is okay to release. Namely, we need to confirm that there are at least three and no dominance (constitutes a share of at least 80%) of the following:

- Product
- Store
- Brand
- Manufacturer

Let's check to make sure our estimate follows the disclosure review guidelines.

In [ ]:
```
# check amount and dominance of products (upc)
print(df['upc'].nunique())

#check for dominance of upcs by selecting top five most represented upcs
print(df['upc'].value_counts(normalize = True)[0:4])
```

In [ ]:
```
# check amount and dominance of stores (storename)
print(df['storename'].nunique())

#check for dominance of stores by selecting top five most represented st
ores
print(df['storename'].value_counts(normalize = True)[0:4])
```

In [ ]:
```
# check amount and dominance of brand (brand)
print(df['brand'].nunique())

#check for dominance of brands by selecting top five most represented br
ands
print(df['brand'].value_counts(normalize = True)[0:4])
```

In [ ]:
```
# check amount and dominance of manufacturer (manufacturer)
print(df['manufacturer'].nunique())

#check for dominance of manufacturers by selecting top five most represe
nted manufacturers
print(df['manufacturer'].value_counts(normalize = True)[0:4])
```

Now that you've shown that this statistic should pass disclosure review, where should you put your proof? As you will read in the documentation, there is an input file to include these statistics and other relevant counts. To load these in, we can save them all to two .csv files: (1) number of unique stores/brands/manufacturers/products and (2) proof of no dominance. The following code cells provide code to do this.

In [ ]:
```
# create csv of all counts we need
data = {'n_unique_prod': [df['upc'].nunique()], 'n_unique_store': [df['s
torename'].nunique()],
        'n_unique_brand': [df['brand'].nunique()], 'n_unique_manufacture
r': [df['manufacturer'].nunique()]}
counts = pd.DataFrame(data)
counts.to_csv('/nfshome/benjaminfeder/counts_estimate_stat.csv')
```

In [ ]:
```
#create csv of dominance proof by taking max of value_counts
data = {'max_prod': [df['upc'].value_counts(normalize = True).max()],
        'max_store': [df['storename'].value_counts(normalize = True).max
()],
        'max_brand': [df['brand'].value_counts(normalize = True).max()],
        'max_manufacturer': [df['manufacturer'].value_counts(normalize =
True).max()],
        }
dominance = pd.DataFrame(data)
dominance.to_csv('/nfshome/benjaminfeder/dominance_estimate_stat.csv')
```

Now, you can easily add `dominance.csv` and `counts.csv` to your input folder for disclosure review. Let's also save our estimate in a separate csv.

In [ ]:
```
sum(df['weighted'])
```

In [ ]:
```
data = pd.DataFrame({'estimate': [sum(df['weighted'])]})
data.to_csv('/nfshome/benjaminfeder/estimate.csv')
```

Grouped Example from Machine Learning

As another example, let's say you wanted to explore `ml_model_train`, which contains every WIC and WIC-eligible household in 2014 and whether or not they purchased 100% whole wheat bread at least once in 2015, among other variables.

> `ml_model_train` was created in the [Data Preparation (04_01_ML_Data_Prep.ipynb)](04_01_ML_Data_Prep.ipynb) notebook.

In [ ]:
```
qry = '''
select *
from iri_usda_2019_db.ml_model_train
'''

df_train = pd.read_sql(qry, conn)
```

Let's say we wanted to visualize the proportion of households by their `wic_june` categorization for their `label` using a barchart. Recall that we can only show the proportion of households after they are weighted. To do so, we can add up the `projection61k` values after grouping by `wic_june` and `label`.

> Since `projection61k` simply accounts for the amount of households in the general population one in our sample is representing, you can add `projection61k` to find weighted counts.

In [ ]:
```
# counts of weights by wic_june and label
grouped = df_train.groupby(['wic_june', 'label'])['projection61k'].sum()
print(grouped.unstack())
```

In [ ]:
```
# Now we can generate the graph
mygraph = grouped.unstack().plot(kind='bar')
```

However, because this barplot was generated based on weighted totals, we need to provide counts for both the weighted and unweighted populations. Clearly, we can see that the totals are all greater than three for each of the six groups, but just in case, we can add a `table=True` arguement to the plot call to display the underlying counts of the weighted table.

In [ ]:
```
# Graphical representation including underlying values: the option table
=True displays the underlying counts
mygraph = grouped.unstack().plot(kind='bar', table=True, figsize=(7,5),
fontsize=7)
```

Finally, to show the counts of the unweighted table, recall that we can use a crosstab, as we did in the [machine learning (04_2_Machine_Learning.ipynb)](04_2_Machine_Learning.ipynb) notebook.

In [ ]:
```
# compute crosstab
pd.crosstab(index=df_train['label'], columns=df_train['wic_june'])

#save as csv
pd.crosstab(index=df_train['label'], columns=df_train['wic_june']).to_cs
v('/nfshome/benjaminfeder/barchart_counts.csv')
```

Since we have confirmed that the underlying unweighted counts are all at least three, we can export this graph as a pdf.

In [ ]:
```
# Now we can export the graph as pdf
# Save plot to file
export = mygraph.get_figure()
export.set_size_inches(15,10, forward=True)
export.savefig('/nfshome/benjaminfeder/sample_barchart.pdf', bbox_inches
='tight', dpi=300)
```

Reminder

Every single item you wish to export, regardless of whether it is a .csv, .pdf, .png, or something else, must have corresponding proof in your input file to show that every group used to create this statistic followed our disclosure review rules.

> We will add more examples in the coming weeks if they can be helpful.