Assignment 1

Data Preparation by Gordon Wall

- Analytics in Practice: MIS 64038

Summary

In summary, I learned to import several kinds of raw data, both JSON and csv. Further, I learned to filter, subset, and manipulate dataframes to output what information I'm looking for. I spent time formatting and learning to clean/organize datasets in the pre-processing phase so future analysis is easier, and I learned to filter the datasets with multiple conditional statements. Finally, I used several libraries to facilitate this learning endeavor including pandas, native json, ijson, and the pycharm IDE for python coding.

Exercise 1

1. Column exploration (#1-4)
   Through an analysis of shape, there appear to be **11 columns** in the Agency dataset and **22 columns** in the Company dataset. **No columns are missing** from either, per an explanation of the datasets on opendata500.com verifying that these variables are the one intended to be present in the preceding two datasets. After further analysis of the data columns, **no variable names are missing** and the names are all unique and correctly assigned with **no errors.**

```
Index(['agency_name', 'agency_abbrev', 'agency_type', 'subagency_name', 'subagency_abbrev', 'url', 'used_by', 'used_by_category', 'used_by_fte',
       'dataset_name', 'dataset_url'],
      dtype='object')
(1123, 11)
Index(['company_name_id', 'company_name', 'url', 'year_founded', 'city', 'state', 'country', 'zip_code', 'full_time_employees', 'company_type',
       'company_category', 'revenue_source', 'business_model', 'social_impact', 'description', 'description_short', 'source_count', 'data_types',
       'example_uses', 'data_impacts', 'financial_info', 'last_updated'],
      dtype='object')
(529, 22)
```

Next is an output of percentage missing values by column for both datasets. This is a helpful statistic in illustrating which columns have missing data AND what percentage of the total is missing all in one line of code. (agency pic 1; company pic 2-3) For example, 'social_impact' column of the companies dataset may be worth dropping since ~97% of its data is missing.

agency_name	0.00
agency_abbrev	27.87
agency_type	0.00
subagency_name	0.00
subagency_abbrev	63.13
url	26.00
used_by	0.00
used_by_category	0.36
used_by_fte	3.92
dataset_name	48.80
dataset_url	71.95

company_name_id	0.00
company_name	0.00
url	0.00
year_founded	0.19
city	6.24
state	0.00
country	0.00
zip_code	6.99
full_time_employees	5.48
company_type	3.02
company_category	0.57

revenue_source	1.89
business_model	14.37
social_impact	96.98
description	0.00
description_short	0.00
source_count	57.28
data_types	73.16
example_uses	98.49
data_impacts	0.00
financial_info	73.16
last_updated	0.00

1. (cont.) The data in each column seems to be organized appropriately, meaning data relevant to the title of each variable column is organized into the respective column.

2. Dataset "Fixing" (#5-7)
   Are the datasets in the proper shape for analysis? **No.** This is because certain variables needed to be addressed first. For example, the column "country" is not unique - every value is "USA" – and, thus, the variable is unnecessary for analysis. Dropping unneeded variables will change the shape of the data by reducing the column dimensionality by n number of columns dropped. After dropping unwanted variable columns, another attempt to fix this dataset would be to impute missing values for the remaining columns that still have values missing. We can do this by using the mean of the column, ffill() for using the previous value to fill the next missing value, etc. Or, we could revisit the data-gathering phase and try to find the real data for each of the missing values from other sources on the internet. Once these missing values have been addressed, we would have to **index each dataset**. It doesn't appear that there is a **common primary key** between the two datasets as they stand now. However, by indexing the two datasets based on shared values in a similar variable column, we could easily create on by resetting the index of each and then assigning a common index and joining the two datasets with an outer or inner join (depending on whether we want to keep non-inclusive rows). A good candidate for indexing this might be "company_name_id" in the company dataset and "agency_name" in the agency dataset. By indexing the sets with these columns, we accomplish both organizing the individual tables by an index for easy loc accessing, while also creating the ability to join the sets and work with them as one dataframe.

Exercise 2

1. After importing the JSON file into python and reading it as a dictionary object, I then performed an operation to extract the column names – aka "variables" – nested in the JSON file. Finally, a count of this list shows that **23 variables** in total, taken from the metadata. HOWEVER, not all of these variables are relevant to data analysis (i.e. ":created_meta"), so filtering this dataset to only important columns, we can see **9 relevant variables** present in the traffic dataset.

```
import json
import ijson
filename = "C:/Users/Owner/Documents/Datasets/ChicagoTraffic.json"
with open(filename, 'r') as x:
    objects = ijson.items(x, 'meta.view.columns.item')
    columns = list(objects)

column_names = [col["fieldName"] for col in columns]
print(len(column_names))
```

2. Next, we can make a list for these "good" columns and print their names.

```
data = []
with open(filename, 'r') as x:
    objects = ijson.items(x, 'data.item')
    for row in objects:
        selected_row = []
        for item in good_columns:
            selected_row.append(row[column_names.index(item)])
            data.append(selected_row)
print(*good_columns, sep="\n")
```

```
id
traffic_volume_count_location_address
street
date_of_count
total_passing_vehicle_volume
vehicle_volume_by_each_direction_of_traffic
latitude
longitude
location
```

3. With the dataset imported and readable as a new pandas DataFrame, we can now run this code to see both the **total traffic volume by street** within the range 100th St to 115th St AND the **total volume for the whole range,** which comes in at **4,385,700 cars**.

```
traffic['total_passing_vehicle_volume'] = pd.to_numeric(traffic['total_passing_vehicle_volume'])
unique_streets = traffic.groupby(traffic['street']).total_passing_vehicle_volume.sum()
print(unique_streets[:'115th St'])
print(unique_streets[:'115th St'].sum())
```

```
100th St         214200
101st St          61200
103rd St        1366200
103rd Street     335700
106th St         148500
107th St         396900
109th St           7200
111th St        1205100
112th St         126000
115th St         524700
Name: total_passing_vehicle_volume, dtype: int64
4385700
```

4. Finally, we can apply similar subsetting to determine the total traffic of vehicles based on geolocation:

```
geolocation1 = traffic[(traffic['latitude'] == '41.651861') & (traffic['longitude'] == '-87.54501')].total_passing_vehicle_volume.sum()
geolocation2 = traffic[(traffic['latitude'] == '41.66836') & (traffic['longitude'] == '-87.620176')].total_passing_vehicle_volume.sum()
print(geolocation1)
print(geolocation2)
```

The total volume at each location is **42,300** for (41.651861, -87.54501) and **80,100** for (41.66836, -87.620176). In summation, this is a grand total of **122,400 cars** across both geolocations.

```
42300
80100

Process finished with exit code 0
```