Problem Set 4: Merging and Regular Expressions, Part 2

In [44]:
```
## helpful packages
import pandas as pd
import numpy as np
import random
import re
import os

## repeated printouts
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Merging and regex (17 points total)

1.1 Load data on job postings

The previous dataset contains a small subset of employers who faced temporary bans due to violations of H-2A program regulations

Since most of the bans have expired, we"re going to see which of those employers posted new H-2A jobs in the first quarter of 2021

Loading the `jobs.csv` data stored in `pset3_inputdata`.

Load the `debar_clean` dataset you created in Problem Set 3 (Merging and Regular Expressions, Part 1). This is not the original `debar` dataset, but the cleaned version you created in Problem Set 3.

In [45]:
```
# your code here to load the data


jobs = pd.read_csv("pset4_inputdata/jobs.csv")
debar_clean = pd.read_csv("pset4_inputdata/debar_clean.csv")
```

1.2 Try inner join on employer name (2 points)

- Use the `EMPLOYER_NAME` field of the `jobs` dataset
- Use the `Name` field of the `debar_clean` dataset

A. Use pd.merge with an inner join on those fields to see whether there are any exact matches.

B. If there are exact matches, print the row(s) with exact matches

In [46]:
```
## your code here

# Using pd.merge to find exact matches between 2 datasets.
exact_matches = jobs.merge(debar_clean, left_on="EMPLOYER_NAME", right_on="Name", how="inner")

# Output
if exact_matches.empty:
    print("No exact matches found.")
else:
    print(f"Exact matches found: {len(exact_matches)} rows")
    print(exact_matches[["EMPLOYER_NAME", "Name"]])
```

```
Exact matches found: 1 rows
    EMPLOYER_NAME            Name
0  Rafael Barajas  Rafael Barajas
```

1.3 Targeted regex (10 points total)

You want to see if you can increase the exact match rate with some basic cleaning of each of the employer name fields in each dataset

1.3.1 Converting to upper (2 points)

A. Convert the `EMPLOYER_NAME` and `Name` fields to uppercase using list comprehension rather than df.varname.str.upper() (it"s fine to do a separate list comprehension line for each of the two columns)

B. Print a random sample of 15 values of each result

C. Assign the full vector of uppercase names back to the original data, writing over the original `EMPLOYER_NAME` and `Name` columns

In [47]:
```
# Convert to upper
employer_upper = [name.upper() if pd.notna(name) else name for name in jobs["EMPLOYER_NAME"]]
debar_upper = [name.upper() if pd.notna(name) else name for name in debar_clean["Name"]]
```

In [48]:
```
## insert your code for the random sample
print("Random sample from EMPLOYER_NAME (jobs):")
print(random.sample(employer_upper, 15))

print("Random sample from Name (debar_clean):")
print(random.sample(debar_upper, 15))
```

```
Random sample from EMPLOYER_NAME (jobs):
['ELMORE TRUCK REPAIR', 'JACKSON CITRUS, INC.', 'TRIPLE B TRUCKING LLC', 'DIONISIO PRODUCE AND FARMS, LLC', 'WESTERN
RANGE ASSOCIATION', 'GALLOPS FARM', 'COUSER CATTLE COMPANY ', 'PECC, INC.', 'S. DERRINGER HARVESTING, INC.', 'ERIC TH
OMAS LLC', 'COASTAL FARM LABOR SERVICES, INC.', "THE NORTH CAROLINA GROWER'S ASSOCIATION, INC.", 'J. GUERRA, LLC', 'W
ESTERN RANGE ASSOCIATION', 'LUCKY AG, INC.']
Random sample from Name (debar_clean):
['YOLANDA CHAVEZ', 'STAHLMAN APIARIES, INC', 'YOLANDA CHAVEZ FARMING', 'JOHN & NETA LEOPKY FARMS', 'OMEGA LAMB, LLC',
'J&J HARVESTING', 'OLSON FARMS', 'FIRST AMERICAN HOLDING', 'XAVIER HORNE', 'AVOYELLES HONEY CO., LLC', 'LEONARD SMITH
FARMS', 'CISCO PRODUCE INC.', 'VERN STRATTON FARMS', 'MONICA SAAVEDRA (AGENT)', 'SRT FARMS']
```

In [49]:
```
## insert your code for assigning the uppercase names back to the data
jobs["EMPLOYER_NAME"] = employer_upper
debar_clean["Name"] = debar_upper

# Show head to confirm changes
print(jobs["EMPLOYER_NAME"].head())
print(debar_clean["Name"].head())
```

```
0    FAZIO FARMS OPERATING COMPANY, LLC
1                    CHARLIE SUNDERLAND
2                     MICHAEL RUDEBUSCH
3                          LODAHL FARMS
4                DUNSON HARVESTING, INC.
Name: EMPLOYER_NAME, dtype: object
0     AUTUMN HILL ORCHARD
1        DOVE CREEK FARMS
2               F&W FARMS
3    MACKY AND BRAD FARMS
4             MARK DUNCAN
Name: Name, dtype: object
```

1.3.2 Cleaning up punctuation (4 points)

You notice that INC, CO, and LLC are sometimes followed by a period (.) but sometimes not

A. For each dataset, write a regex pattern using `re.sub` to remove the . but only if it"s preceded by INC, LLC, or CO

Make sure LLC, INC, CO remain part of the string but just without the dot

B. Test the pattern on the positive and negative example we provide below and print the result. See the Github issue for examples of what to return

Hint: https://stackoverflow.com/questions/7191209/python-re-sub-replace-with-matched-content

In [50]:

```python
pos_example_1 = "CISCO PRODUCE INC."
pos_example_2 = "AVOYELLES HONEY CO., LLC"
neg_example = "E.V. RANCH LLP"
```

In [51]:

```python
## insert your code here with the regex pattern for part A
pattern = r"\b(INC|LLC|CO)\."

def clean_names(name):
    return re.sub(pattern, r"\1", name) if pd.notna(name) else name

example_series = pd.Series([pos_example_1, pos_example_2, neg_example])
cleaned = example_series.apply(clean_names)
print(cleaned)
```

```
0          CISCO PRODUCE INC
1    AVOYELLES HONEY CO, LLC
2             E.V. RANCH LLP
dtype: object
```

1.3.3 (4 points)

Use that pattern in conjunction with `re.sub` and list comprehension to clean the employer name columns in each dataset. Save the new columns as `name_clean` in each. Then, use row subsetting to (1) subset to rows that changed names and (2) for:

- `debar_clean` print the `Name` and `name_clean` columns
- `jobs` print the `EMPLOYER_NAME` and `name_clean` columns

Make sure to use the uppercase versions of the variables

In [52]:
```
## your code here to clean the columns
debar_clean["name_clean"] = debar_clean["Name"].apply(clean_names)
jobs["name_clean"] = jobs["EMPLOYER_NAME"].apply(clean_names)

# Subset to rows where the cleaned name is different from the original
debar_changed = debar_clean[debar_clean["Name"] != debar_clean["name_clean"]]
jobs_changed = jobs[jobs["EMPLOYER_NAME"] != jobs["name_clean"]]
```

In [53]:
```
## your code here to print the head
print("debar_clean: rows with changed names")
print(debar_changed[["Name", "name_clean"]].head())

print("jobs: rows with changed names")
print(jobs_changed[["EMPLOYER_NAME", "name_clean"]].head())
```

```
debar_clean: rows with changed names
                        Name               name_clean
14  ALTENDORF TRANSPORT INC.  ALTENDORF TRANSPORT INC
18     ANTON FERTILIZER INC.     ANTON FERTILIZER INC
19  AVOYELLES HONEY CO., LLC  AVOYELLES HONEY CO, LLC
26        CISCO PRODUCE INC.        CISCO PRODUCE INC
27        CISCO PRODUCE INC.        CISCO PRODUCE INC
jobs: rows with changed names
                               EMPLOYER_NAME  \
4                     DUNSON HARVESTING, INC.
7   FARM LABOR ASSOCIATION FOR GROWERS, INC.
14                         MCLAIN FARMS, INC.
17                        BONNIE PLANTS, INC.
18                B & W QUALITY GROWERS, INC.

                                 name_clean
4                     DUNSON HARVESTING, INC
7   FARM LABOR ASSOCIATION FOR GROWERS, INC
14                         MCLAIN FARMS, INC
17                        BONNIE PLANTS, INC
18                B & W QUALITY GROWERS, INC
```

1.4 More joins and more cleaning (5 points)

A. Conduct another inner join between `jobs` and `debar_clean` now using the `name_clean` column; print the result. Did the cleaning result in any more employers matched between the two datasets?

B. Create a new column in `debar_clean` called `name_clean_2` that uses regex to take the following name in that dataset:

- `SLASH E.V. RANCH LLP` in the `debar_clean` dataset

And cleans it up so that it matches with this employer in `jobs`

- `SLASH EV RANCH` in the `jobs` dataset

Eg a pattern to remove the dots in the EV and the space+LLP-- you can apply the pattern to all employer names in debar_clean (so don"t need to worry about only applying it to that one employer)

C. Conduct a left join using `name_clean_2` as the join column where the left hand dataframe is `jobs`; right hand dataframe is `debar_clean`, store the result as a dataframe, and print the rows where the merge indicator indicates the row was found in both dataframe

Note: this manual cleaning process is inefficient and helps motivate why talked about fuzzy matching. Fuzzy matching could recognize that Slash EV ranch is a highly similar string to slash ev ranch llp and match them without us needing to use regex to make the strings identical.

In [54]:
```
## your code here

# Part A: Find matches after cleaning
clean_matches = jobs.merge(debar_clean, left_on="name_clean", right_on="name_clean", how="inner")

if clean_matches.empty:
    print("No matches found after cleaning.")
else:
    print(f"Matches found after cleaning: {len(clean_matches)} rows")
    print("No Changes Found" if len(exact_matches) == len(clean_matches) else "Changes found after cleaning.")
    print(clean_matches[["EMPLOYER_NAME", "Name", "name_clean"]])
```

```
Matches found after cleaning: 1 rows
No Changes Found
    EMPLOYER_NAME             Name      name_clean
0  RAFAEL BARAJAS  RAFAEL BARAJAS  RAFAEL BARAJAS
```

In [55]:
```
# Part B: Remove dots in EV and space in EV
def clean_ev_llp(name):
    if pd.isna(name):
        return name
    name = re.sub(r"\b([A-Z])\.", r"\1", name)
    name = re.sub(r"\sLLP\b", "", name)
    return name.strip()


debar_clean['name_clean_2'] = debar_clean['Name'].apply(clean_ev_llp)
jobs['name_clean_2'] = jobs['EMPLOYER_NAME'].apply(clean_ev_llp)

print(debar_clean[debar_clean['Name'].str.contains('SLASH')][['Name', 'name_clean_2']])
print(jobs[jobs['EMPLOYER_NAME'].str.contains('SLASH')][['EMPLOYER_NAME', 'name_clean_2']])
```

```
                    Name    name_clean_2
90  SLASH E.V. RANCH LLP  SLASH EV RANCH
       EMPLOYER_NAME    name_clean_2
1115  SLASH EV RANCH  SLASH EV RANCH
```

In [56]:
```
# Part C: Merge datasets using the new cleaned column
merged = jobs.merge(debar_clean, left_on="name_clean_2", right_on="name_clean_2", how="left", indicator=True)

print(merged[merged['_merge'] == 'both'][["EMPLOYER_NAME", "Name", "name_clean_2", "_merge"]])
```

```
       EMPLOYER_NAME                  Name    name_clean_2 _merge
791   RAFAEL BARAJAS        RAFAEL BARAJAS  RAFAEL BARAJAS   both
1115  SLASH EV RANCH  SLASH E.V. RANCH LLP  SLASH EV RANCH   both
```

2. Optional extra credit 1: regex to separate companies from individuals (1 point)

You notice some employers in `debar_clean` have both the name of the company and the name of individual, e.g.:

COUNTY FAIR FARM (COMPANY) AND ANDREW WILLIAMSON (INDIVIDUAL)*

Use the uppercase/cleaned `name_clean` in `debar_clean`

A. Write a regex pattern that does the following: - Captures the pattern that occurs before COMPANY if (COMPANY) is in string; so in example above, extracts COUNTY FAIR FARM - Captures the pattern that occurs before INDIVIDUAL if (INDIVIDUAL) is also in string -- so in above, extracts ANDREW WILLIAMSON (so omit the "and")

B. Test the pattern on `pos_example` and `neg_example` -- make sure former returns a list (if using find.all) or match object (if using re.search) with the company name and individual name separated out; make sure latter returns empty

Hints and resources: for step A, you can either use re.search, re.match, or re.findall; don"t worry about matching B&R Harvesting and Paul Cruz (Individual)

- Same regex resources as above

In [57]:
```
pos_example = "COUNTY FAIR FARM (COMPANY) AND ANDREW WILLIAMSON (INDIVIDUAL)*"
neg_example = "CISCO PRODUCE INC"
```

```
## your code here to define the pattern
# (?P<co>.+?) = capture group for company name (non-greedy)
# (?P<ind>.+?) = capture group for individual name (non-greedy)
# \s* = optional space (also handles extra spaces)
# \s+AND\s+ = " AND " with optional space around it
# \(COMPANY\) = literal string "(COMPANY)"
# \(INDIVIDUAL\) = literal string "(INDIVIDUAL)"
pattern = r"^(?P<co>.+?)\s*\(COMPANY\)\s+AND\s+(?P<ind>.+?)\s*\(INDIVIDUAL\)"

## your code here to apply it to the pos_example
pos_match = re.match(pattern, pos_example)
print("POS match:", pos_match.groups() if pos_match else "No match")

## your code here to apply it to the negative example
pos_match_neg = re.match(pattern, neg_example)
print("NEG match:", pos_match_neg.groups() if pos_match_neg else "No match")
```

```
POS match: ('COUNTY FAIR FARM', 'ANDREW WILLIAMSON')
NEG match: No match
```

C. Iterate over the `name_clean` column in debar and use regex to create two new columns in `debar_clean`:

- `co_name`: A column for company (full `name_clean` string if no match; pattern before COMPANY if one extracted)
- `ind_name`: A column for individual (full `name_clean` string if no match; pattern before INDIVIDUAL if one extracted)

In [61]:
```
# your code here
co_list = []
ind_list = []

for name in debar_clean["name_clean"]:
    match = re.match(pattern, name)
    if match:
        co = match.group("co").strip()
        ind = match.group("ind").strip()
        co_list.append(co)
        ind_list.append(ind)
    else:
        co_list.append(name)
        ind_list.append(name)

debar_clean["co_name"] = co_list
```

```
debar_clean["ind_name"] = ind_list

print(debar_clean[debar_clean["name_clean"].str.contains("AND")][["name_clean", "co_name", "ind_name"]])
```

```
                                             name_clean  \
3                                 MACKY AND BRAD FARMS   
16                             ANNABELLA LAND & CATTLE   
17                             ANNABELLA LAND & CATTLE   
20        B & R HARVESTING AND PAUL CRUZ (INDIVIDUAL)   
28                       CITY PINESTRAW AND HARVESTING   
29   COUNTY FAIR FARM (COMPANY) AND ANDREW WILLIAMS...   
44                                  GONZALO FERNANDEZ*   
51                                     JEREMY CHANDLER   
53               JIM AND ANN SHIPLEY WILLIAM SHIPLEY*   
59                                LANDMARK LANDSCAPING   
93                            TRAVIS AND TARA LAMBOURN   
103                                     YOLANDA CHAVEZ   
104                             YOLANDA CHAVEZ FARMING   

                                         co_name  \
3                           MACKY AND BRAD FARMS   
16                       ANNABELLA LAND & CATTLE   
17                       ANNABELLA LAND & CATTLE   
20   B & R HARVESTING AND PAUL CRUZ (INDIVIDUAL)   
28                 CITY PINESTRAW AND HARVESTING   
29                              COUNTY FAIR FARM   
44                            GONZALO FERNANDEZ*   
51                               JEREMY CHANDLER   
53         JIM AND ANN SHIPLEY WILLIAM SHIPLEY*   
59                          LANDMARK LANDSCAPING   
93                      TRAVIS AND TARA LAMBOURN   
103                               YOLANDA CHAVEZ   
104                       YOLANDA CHAVEZ FARMING   

                                        ind_name  
3                           MACKY AND BRAD FARMS  
16                       ANNABELLA LAND & CATTLE  
17                       ANNABELLA LAND & CATTLE  
20   B & R HARVESTING AND PAUL CRUZ (INDIVIDUAL)  
28                 CITY PINESTRAW AND HARVESTING  
29                             ANDREW WILLIAMSON  
44                            GONZALO FERNANDEZ*  
51                               JEREMY CHANDLER  
53         JIM AND ANN SHIPLEY WILLIAM SHIPLEY*  
59                          LANDMARK LANDSCAPING  
93                      TRAVIS AND TARA LAMBOURN  
```

```
103                              YOLANDA CHAVEZ
104                      YOLANDA CHAVEZ FARMING
```

D. Print three columns for the rows in `debar_clean` containing the negative example and positive example described above (county fair farm and cisco produce):

- `name_clean`
- `co_name`
- `ind_name`
- `Violation`

Note: as shown in the outcome there may be duplicates of the same company reflecting different violations

In [59]:
```
# your code here
mask_county = debar_clean["name_clean"].str.contains("COUNTY FAIR FARM")
mask_cisco  = debar_clean["name_clean"].str.contains("CISCO PRODUCE")

cols = ["name_clean", "co_name", "ind_name", "Violation"]

print("COUNTY FAIR FARM rows:")
print(debar_clean.loc[mask_county, cols])

print("\nCISCO PRODUCE rows:")
print(debar_clean.loc[mask_cisco, cols])
```

```
COUNTY FAIR FARM rows:
                                          name_clean           co_name  \
29  COUNTY FAIR FARM (COMPANY) AND ANDREW WILLIAMS...  COUNTY FAIR FARM

             ind_name      Violation
29  ANDREW WILLIAMSON  WHD Debarment

CISCO PRODUCE rows:
           name_clean            co_name           ind_name  \
26  CISCO PRODUCE INC  CISCO PRODUCE INC  CISCO PRODUCE INC
27  CISCO PRODUCE INC  CISCO PRODUCE INC  CISCO PRODUCE INC

                                  Violation
26   Failure to respond to audit (no response)
27  Impeding the Audit Process – Non- Response
```