```
import delimited using "Thesis/AnalysisOneDataset.csv", clear

* Basic variables
gen zero_sales = (per_capita_real_gross_sales == 0)
gen log_population = log(population)

*****************************************************
* CREATE DISTANCE BINS
*****************************************************
gen dist_bin = .
replace dist_bin = 0 if dist_mi_to_tn < 20
replace dist_bin = 1 if dist_mi_to_tn >= 20 & dist_mi_to_tn <= 40
replace dist_bin = 2 if dist_mi_to_tn > 40

* Label bins (VERY important for table readability)
label define distbin_lbl 0 "<20" 1 "20–40" 2 "40+"
label values dist_bin distbin_lbl

*****************************************************
* RUN BIN REGRESSION (50+ AS BASELINE)
*****************************************************
eststo clear

reg per_capita_bottles_regular_sold ib2.dist_bin ///
    pct_male_0_17 pct_male_18_24 pct_male_25_44 pct_male_45_64 pct_male_65_plus ///
    zero_sales ///
    pct_male pct_bachelor_plus military_per_100k ///
    pct_white pct_american_indian_alaska_nativ pct_black pct_asian ///
    pct_native_pacific pct_some_other_race pct_two_or_more_races ///
    pct_hispanic_or_latino unemployment_rate ///
    per_capita_real_expenditures_mil ///
    log_population i.year ///
    [aweight=population], cluster(county_name)

eststo mbin

*****************************************************
* EXPORT (MAIN TABLE – ONLY BINS)
*****************************************************
esttab mbin using "bin_table_tn_bottles_main.rtf", replace ///
```

```
    title("Effect of Distance Bins to SC on Alcohol Sales") ///
    keep(0.dist_bin 1.dist_bin) ///
    se star(* 0.10 ** 0.05 *** 0.01) ///
    stats(N r2, labels("Observations" "R-squared")) ///
    label

*****************************************************
* EXPORT (FULL TABLE – ALL CONTROLS)
*****************************************************
esttab mbin using "bin_table_tn_full_bottles.rtf", replace ///
    title("Full Regression Results (Distance Bins)") ///
    se star(* 0.10 ** 0.05 *** 0.01) ///
    stats(N r2, labels("Observations" "R-squared")) ///
    label

import delimited using "Thesis/AnalysisOneDataset.csv", clear

* Basic variables
gen zero_sales = (per_capita_real_gross_sales == 0)
gen log_population = log(population)

*****************************************************
* CREATE DISTANCE BINS
*****************************************************
gen dist_bin = .
replace dist_bin = 0 if dist_mi_to_tn < 20
replace dist_bin = 1 if dist_mi_to_tn >= 20 & dist_mi_to_tn <= 40
replace dist_bin = 2 if dist_mi_to_tn > 40

* Label bins (VERY important for table readability)
label define distbin_lbl 0 "<20" 1 "20–40" 2 "40+"
label values dist_bin distbin_lbl

*****************************************************
* RUN BIN REGRESSION (50+ AS BASELINE)
*****************************************************
eststo clear

reg per_capita_real_gross_sales ib2.dist_bin ///
    pct_male_0_17 pct_male_18_24 pct_male_25_44 pct_male_45_64 pct_male_65_plus ///
```

```
    zero_sales ///
    pct_male pct_bachelor_plus military_per_100k ///
    pct_white pct_american_indian_alaska_nativ pct_black pct_asian ///
    pct_native_pacific pct_some_other_race pct_two_or_more_races ///
    pct_hispanic_or_latino unemployment_rate ///
    per_capita_real_expenditures_mil ///
    log_population i.year ///
    [aweight=population], cluster(county_name)

eststo mbin

**************************************************
* EXPORT (MAIN TABLE – ONLY BINS)
**************************************************
esttab mbin using "bin_table_tn_bottles_main.rtf", replace ///
    title("Effect of Distance Bins to SC on Alcohol Sales") ///
    keep(0.dist_bin 1.dist_bin) ///
    se star(* 0.10 ** 0.05 *** 0.01) ///
    stats(N r2, labels("Observations" "R-squared")) ///
    label

**************************************************
* EXPORT (FULL TABLE – ALL CONTROLS)
**************************************************
esttab mbin using "bin_table_tn_full_bottles.rtf", replace ///
    title("Full Regression Results (Distance Bins)") ///
    se star(* 0.10 ** 0.05 *** 0.01) ///
    stats(N r2, labels("Observations" "R-squared")) ///
    label

import delimited using "Thesis/AnalysisOneDataset.csv", clear

* Basic variables
gen zero_sales = (per_capita_real_gross_sales == 0)
gen log_population = log(population)

**************************************************
* COMMON LABEL
**************************************************
label define distbin_lbl 0 "<20" 1 "20–40" 2 "40+"
```

```
**************************************************
* ========== TENNESSEE ==========
**************************************************
gen dist_bin = .
replace dist_bin = 0 if dist_mi_to_tn < 20
replace dist_bin = 1 if dist_mi_to_tn >= 20 & dist_mi_to_tn <= 40
replace dist_bin = 2 if dist_mi_to_tn > 40
label values dist_bin distbin_lbl

eststo clear

* --- SALES ---
reg per_capita_real_gross_sales ib2.dist_bin ///
    pct_male_0_17 pct_male_18_24 pct_male_25_44 pct_male_45_64 pct_male_65_plus ///
    zero_sales ///
    pct_male pct_bachelor_plus military_per_100k ///
    pct_white pct_american_indian_alaska_nativ pct_black pct_asian ///
    pct_native_pacific pct_some_other_race pct_two_or_more_races ///
    pct_hispanic_or_latino unemployment_rate ///
    per_capita_real_expenditures_mil ///
    log_population i.year ///
    [aweight=population], cluster(county_name)

eststo tn_sales

* --- BOTTLES ---
reg per_capita_bottles_regular_sold ib2.dist_bin ///
    pct_male_0_17 pct_male_18_24 pct_male_25_44 pct_male_45_64 pct_male_65_plus ///
    zero_sales ///
    pct_male pct_bachelor_plus military_per_100k ///
    pct_white pct_american_indian_alaska_nativ pct_black pct_asian ///
    pct_native_pacific pct_some_other_race pct_two_or_more_races ///
    pct_hispanic_or_latino unemployment_rate ///
    per_capita_real_expenditures_mil ///
    log_population i.year ///
    [aweight=population], cluster(county_name)

eststo tn_bottles
```

```
* EXPORT
esttab tn_sales using "bin_table_tn_sales_main.rtf", replace ///
    title("Effect of Distance Bins to TN on Alcohol Sales") ///
    keep(0.dist_bin 1.dist_bin) se star(* 0.10 ** 0.05 *** 0.01) ///
    stats(N r2, labels("Observations" "R-squared")) label

esttab tn_bottles using "bin_table_tn_bottles_main.rtf", replace ///
    title("Effect of Distance Bins to TN on Bottles Sold") ///
    keep(0.dist_bin 1.dist_bin) se star(* 0.10 ** 0.05 *** 0.01) ///
    stats(N r2, labels("Observations" "R-squared")) label


**************************************************
* ========== VIRGINIA ==========
**************************************************
drop dist_bin
gen dist_bin = .
replace dist_bin = 0 if dist_mi_to_va < 20
replace dist_bin = 1 if dist_mi_to_va >= 20 & dist_mi_to_va <= 40
replace dist_bin = 2 if dist_mi_to_va > 40
label values dist_bin distbin_lbl

eststo clear

* --- SALES ---
reg per_capita_real_gross_sales ib2.dist_bin ///
    pct_male_0_17 pct_male_18_24 pct_male_25_44 pct_male_45_64 pct_male_65_plus ///
    zero_sales ///
    pct_male pct_bachelor_plus military_per_100k ///
    pct_white pct_american_indian_alaska_nativ pct_black pct_asian ///
    pct_native_pacific pct_some_other_race pct_two_or_more_races ///
    pct_hispanic_or_latino unemployment_rate ///
    per_capita_real_expenditures_mil ///
    log_population i.year ///
    [aweight=population], cluster(county_name)

eststo va_sales

* --- BOTTLES ---
reg per_capita_bottles_regular_sold ib2.dist_bin ///
```

```
    pct_male_0_17 pct_male_18_24 pct_male_25_44 pct_male_45_64 pct_male_65_plus ///
    zero_sales ///
    pct_male pct_bachelor_plus military_per_100k ///
    pct_white pct_american_indian_alaska_nativ pct_black pct_asian ///
    pct_native_pacific pct_some_other_race pct_two_or_more_races ///
    pct_hispanic_or_latino unemployment_rate ///
    per_capita_real_expenditures_mil ///
    log_population i.year ///
    [aweight=population], cluster(county_name)

eststo va_bottles

* EXPORT
esttab va_sales using "bin_table_va_sales_main.rtf", replace ///
    title("Effect of Distance Bins to VA on Alcohol Sales") ///
    keep(0.dist_bin 1.dist_bin) se star(* 0.10 ** 0.05 *** 0.01) ///
    stats(N r2, labels("Observations" "R-squared")) label

esttab va_bottles using "bin_table_va_bottles_main.rtf", replace ///
    title("Effect of Distance Bins to VA on Bottles Sold") ///
    keep(0.dist_bin 1.dist_bin) se star(* 0.10 ** 0.05 *** 0.01) ///
    stats(N r2, labels("Observations" "R-squared")) label



**************************************************
* ========== SOUTH CAROLINA ==========
**************************************************
drop dist_bin
gen dist_bin = .
replace dist_bin = 0 if dist_mi_to_sc < 20
replace dist_bin = 1 if dist_mi_to_sc >= 20 & dist_mi_to_sc <= 40
replace dist_bin = 2 if dist_mi_to_sc > 40
label values dist_bin distbin_lbl

eststo clear

* --- SALES ---
reg per_capita_real_gross_sales ib2.dist_bin ///
    pct_male_0_17 pct_male_18_24 pct_male_25_44 pct_male_45_64 pct_male_65_plus ///
    zero_sales ///
```

```
    pct_male pct_bachelor_plus military_per_100k ///
    pct_white pct_american_indian_alaska_nativ pct_black pct_asian ///
    pct_native_pacific pct_some_other_race pct_two_or_more_races ///
    pct_hispanic_or_latino unemployment_rate ///
    per_capita_real_expenditures_mil ///
    log_population i.year ///
    [aweight=population], cluster(county_name)

eststo sc_sales

* --- BOTTLES ---
reg per_capita_bottles_regular_sold ib2.dist_bin ///
    pct_male_0_17 pct_male_18_24 pct_male_25_44 pct_male_45_64 pct_male_65_plus ///
    zero_sales ///
    pct_male pct_bachelor_plus military_per_100k ///
    pct_white pct_american_indian_alaska_nativ pct_black pct_asian ///
    pct_native_pacific pct_some_other_race pct_two_or_more_races ///
    pct_hispanic_or_latino unemployment_rate ///
    per_capita_real_expenditures_mil ///
    log_population i.year ///
    [aweight=population], cluster(county_name)

eststo sc_bottles

* EXPORT
esttab sc_sales using "bin_table_sc_sales_main.rtf", replace ///
    title("Effect of Distance Bins to SC on Alcohol Sales") ///
    keep(0.dist_bin 1.dist_bin) se star(* 0.10 ** 0.05 *** 0.01) ///
    stats(N r2, labels("Observations" "R-squared")) label

esttab sc_bottles using "bin_table_sc_bottles_main.rtf", replace ///
    title("Effect of Distance Bins to SC on Bottles Sold") ///
    keep(0.dist_bin 1.dist_bin) se star(* 0.10 ** 0.05 *** 0.01) ///
    stats(N r2, labels("Observations" "R-squared")) label

import delimited using "Thesis/AnalysisOneDataset.csv", clear

gen log_population = log(population)
gen zero_sales = (per_capita_real_gross_sales == 0)
```

```
**************************************************
* ========== TENNESSEE ==========
**************************************************
eststo clear

foreach d in 10 20 30 40 50 {

   reg per_capita_bottles_regular_sold ///
      dist_mi_to_tn ///
      pct_male_0_17 pct_male_18_24 pct_male_25_44 pct_male_45_64 pct_male_65_plus ///
      pct_male pct_bachelor_plus military_per_100k ///
      pct_white pct_american_indian_alaska_nativ pct_black pct_asian ///
      pct_native_pacific pct_some_other_race pct_two_or_more_races ///
      stores_per_100_sqmi ///
      pct_hispanic_or_latino unemployment_rate ///
      per_capita_real_expenditures_mil ///
      log_population i.year ///
      [aweight=population] if dist_mi_to_tn <= `d', cluster(county_name)

   eststo model_tn_`d'
}

esttab model_tn_10 model_tn_20 model_tn_30 model_tn_40 model_tn_50 ///
   using "distance_continuous_bottle_tn.rtf", replace ///
   se star(* 0.10 ** 0.05 *** 0.01) ///
   mtitles("≤10" "≤20" "≤30" "≤40" "≤50") ///
   stats(N r2, labels("Observations" "R-squared")) ///
   label


**************************************************
* ========== VIRGINIA ==========
**************************************************
eststo clear

foreach d in 10 20 30 40 50 {

   reg per_capita_bottles_regular_sold ///
      dist_mi_to_va ///
      pct_male_0_17 pct_male_18_24 pct_male_25_44 pct_male_45_64 pct_male_65_plus ///
```

```
        pct_male pct_bachelor_plus military_per_100k ///
        pct_white pct_american_indian_alaska_nativ pct_black pct_asian ///
        pct_native_pacific pct_some_other_race pct_two_or_more_races ///
        stores_per_100_sqmi ///
        pct_hispanic_or_latino unemployment_rate ///
        per_capita_real_expenditures_mil ///
        log_population i.year ///
        [aweight=population] if dist_mi_to_va <= `d', cluster(county_name)

    eststo model_va_`d'
}

esttab model_va_10 model_va_20 model_va_30 model_va_40 model_va_50 ///
    using "distance_continuous_bottle_va.rtf", replace ///
    se star(* 0.10 ** 0.05 *** 0.01) ///
    mtitles("≤10" "≤20" "≤30" "≤40" "≤50") ///
    stats(N r2, labels("Observations" "R-squared")) ///
    label


**************************************************
* ========== SOUTH CAROLINA ==========
**************************************************
eststo clear

foreach d in 10 20 30 40 50 {

    reg per_capita_bottles_regular_sold ///
        dist_mi_to_sc ///
        pct_male_0_17 pct_male_18_24 pct_male_25_44 pct_male_45_64 pct_male_65_plus ///
        pct_male pct_bachelor_plus military_per_100k ///
        pct_white pct_american_indian_alaska_nativ pct_black pct_asian ///
        pct_native_pacific pct_some_other_race pct_two_or_more_races ///
        stores_per_100_sqmi ///
        pct_hispanic_or_latino unemployment_rate ///
        per_capita_real_expenditures_mil ///
        log_population i.year ///
        [aweight=population] if dist_mi_to_sc <= `d', cluster(county_name)

    eststo model_sc_`d'
```

}

```
esttab model_sc_10 model_sc_20 model_sc_30 model_sc_40 model_sc_50 ///
   using "distance_continuous_bottle_sc.rtf", replace ///
   se star(* 0.10 ** 0.05 *** 0.01) ///
   mtitles("≤10" "≤20" "≤30" "≤40" "≤50") ///
   stats(N r2, labels("Observations" "R-squared")) ///
   label
```

```
import requests
import pandas as pd
import time
from functools import reduce

API_KEY = "87094356620532d418a4f8ced0e770f070b2e383"
NC_FIPS = "37"

#Years to fetch (ACS 5-year endpoints)
YEARS = range(2024, 2010, -1)

#Variables to request (keys are ACS variable codes)
VARIABLES = {
   "NAME": "County Name",
   "B01003_001E": "Total Population",
   **{f"B01001_{str(i).zfill(3)}E": f"B01001_{str(i).zfill(3)}E" for i in
range(2, 50)},
   "B01001_002E": "Total Male",
   "B19013_001E": "Median Household Income",
   "B23025_005E": "Unemployed",
   "B23025_003E": "Civilian Labor Force",
   "B23025_006E": "Armed Forces",
   "B15003_022E": "Bachelor's degree",
   "B15003_023E": "Master's degree",
   "B15003_024E": "Professional degree",
   "B15003_025E": "Doctorate degree",
   "B15003_001E": "Population 25+",
   "B25004_001E": "Total Housing Units",
   "B02001_001E": "Race Total",
   "B02001_002E": "White alone",
   "B02001_003E": "Black or African American alone",
   "B02001_004E": "American Indian and Alaska Native alone",
   "B02001_005E": "Asian alone",
   "B02001_006E": "Native Hawaiian and Other Pacific Islander alone",
   "B02001_007E": "Some other race alone",
   "B02001_008E": "Two or more races",
   "B03003_003E": "Hispanic or Latino"
}
```

```
#Conservative chunk size so I can avoid overly long pulls
CHUNK_SIZE = 40

#Getting data pieces
def chunk_list(lst, n):
    """Yield successive n-sized chunks from list lst."""
    for i in range(0, len(lst), n):
        yield lst[i:i + n]

#Defining key functions
def fetch_acs_data_chunked(year, variables, api_key, state_fips=NC_FIPS,
chunk_size=CHUNK_SIZE):
    """
    Fetch ACS 5-year data for a state by splitting variables into chunks and
merging results.
    IMPORTANT: Does not include 'state' or 'county' in the 'get' list. They are
returned automatically
    because I use the 'for' and 'in' parameters.
    Returns a pandas DataFrame for the year, or None if all chunks fail.
    """
    base_url = f"https://api.census.gov/data/{year}/acs/acs5"
    var_keys = list(variables.keys())

    #Ensuring NAME is present in every chunk so I can merge by
NAME+state+county+year
    always_include = ['NAME']

    #Removing any accidental 'state'/'county' keys
    var_keys = [v for v in var_keys if v not in ('state', 'county')]

    payload_vars = [v for v in var_keys if v not in always_include]
    #Creating chunks sized so that always_include fits in each chunk
    chunks = list(chunk_list(payload_vars, max(1, chunk_size -
len(always_include))))
    if not chunks:
        chunks = [payload_vars]

    dfs = []
    for i, chunk_vars in enumerate(chunks, start=1):
        vars_for_chunk = always_include + chunk_vars
        params = {
            "get": ",".join(vars_for_chunk),
            "for": "county:*",
            "in": f"state:{state_fips}",
            "key": api_key
        }

        try:
```

```
            resp = requests.get(base_url, params=params, timeout=30)
        except Exception as e:
            print(f"Chunk {i}/{len(chunks)} request error: {e}")
            continue

        if resp.status_code == 404:
            print(f"Chunk {i}/{len(chunks)}: 404 not found for year {year} -
skipping chunk.")
            continue

        if resp.status_code != 200:
            print(f"Chunk {i}/{len(chunks)} failed with status
{resp.status_code}: {resp.text}")
            continue

        payload = resp.json()
        chunk_df = pd.DataFrame(payload[1:], columns=payload[0])
        chunk_df['year'] = year

        #The API returns 'state' and 'county' columns because we used for/in
parameters.
        #Converting numeric-like columns to numeric, keep NAME/state/county/year
as non-numeric
        non_numeric = ['NAME', 'state', 'county', 'year']
        numeric_cols = [c for c in chunk_df.columns if c not in non_numeric]
        chunk_df[numeric_cols] = chunk_df[numeric_cols].apply(pd.to_numeric,
errors='coerce')

        dfs.append(chunk_df)
        time.sleep(0.25)

    if not dfs:
        return None

    #Merging dataframes on state + county + year
    def merge_two(left, right):
        join_keys = ['state', 'county', 'year']
        return pd.merge(left, right, on=join_keys, how='outer')

    merged = reduce(merge_two, dfs)
    return merged

#Gathering together some helpful age brackets
def calculate_age_brackets(df):
    """
    Create age brackets keeping male and female shares separate.
    Uses B01001 indices: male 003-025, female 027-049.
    Produces:
      - pop_male_<bracket>, pop_female_<bracket>
```

```
    - pct_male_<bracket>, pct_female_<bracket>
    - pct_age_<bracket>_bracket  (string "[male_pct, female_pct]")
  Brackets: 0-17, 18-24, 25-44, 45-64, 65+
  """
  male_ranges = {
      '0_17': list(range(3, 7)),    # B01001_003E..B01001_006E
      '18_24': list(range(7, 11)),  # 007..010
      '25_44': list(range(11, 15)), # 011..014
      '45_64': list(range(15, 20)), # 015..019
      '65_plus': list(range(20, 26)) # 020..025
  }

  for bracket, male_idxs in male_ranges.items():
      male_cols = [f"B01001_{str(i).zfill(3)}E" for i in male_idxs if
f"B01001_{str(i).zfill(3)}E" in df.columns]
      female_cols = [f"B01001_{str(i+24).zfill(3)}E" for i in male_idxs if
f"B01001_{str(i+24).zfill(3)}E" in df.columns]

      df[f'pop_male_{bracket}'] = df[male_cols].sum(axis=1) if male_cols else
0
      df[f'pop_female_{bracket}'] = df[female_cols].sum(axis=1) if female_cols
else 0

      if 'B01003_001E' in df.columns:
          df[f'pct_male_{bracket}'] = df[f'pop_male_{bracket}'] /
df['B01003_001E'] * 100
          df[f'pct_female_{bracket}'] = df[f'pop_female_{bracket}'] /
df['B01003_001E'] * 100
          df[f'pct_age_{bracket}_bracket'] = df.apply(
              lambda r: f"[{round(r.get(f'pct_male_{bracket}', 0), 2)},
{round(r.get(f'pct_female_{bracket}', 0), 2)}]",
              axis=1
          )
      else:
          df[f'pct_male_{bracket}'] = pd.NA
          df[f'pct_female_{bracket}'] = pd.NA
          df[f'pct_age_{bracket}_bracket'] = None

  return df

#Calculating several variables
def calculate_derived_variables(df):
  """
  Compute derived variables: age brackets, population, percents, race pct,
unemployment, education shares, etc.
  """
  df = calculate_age_brackets(df)

  #Population fields
```

```
    if 'B01003_001E' in df.columns:
        #Creating a nullable integer population column and a numeric copy
        df['population'] = df['B01003_001E'].fillna(0).round().astype('Int64')
        df['population_numeric'] = df['B01003_001E'].fillna(0).astype(float)

    #pct_male (total male share)
    if 'B01001_002E' in df.columns and 'B01003_001E' in df.columns:
        df['pct_male'] = df['B01001_002E'] / df['B01003_001E'] * 100

    #unemployment rate
    if 'B23025_005E' in df.columns and 'B23025_003E' in df.columns:
        df['unemployment_rate'] = df['B23025_005E'] / df['B23025_003E'] * 100

    #education: percent bachelor+
    deg_cols = [c for c in ['B15003_022E', 'B15003_023E', 'B15003_024E',
'B15003_025E'] if c in df.columns]
    if deg_cols and 'B15003_001E' in df.columns:
        df['pct_bachelor_plus'] = df[deg_cols].sum(axis=1) / df['B15003_001E'] *
100

    #military per 100k
    if 'B23025_006E' in df.columns and 'B01003_001E' in df.columns:
        df['military_per_100k'] = df['B23025_006E'] / df['B01003_001E'] * 100000

    #Race percentages (single-race) + hispanic
    race_map = {
        'white': 'B02001_002E',
        'black': 'B02001_003E',
        'american_indian_alaska_native': 'B02001_004E',
        'asian': 'B02001_005E',
        'native_pacific': 'B02001_006E',
        'some_other_race': 'B02001_007E',
        'two_or_more_races': 'B02001_008E',
        'hispanic_or_latino': 'B03003_003E'
    }

    for name, code in race_map.items():
        if code in df.columns and 'B01003_001E' in df.columns:
            df[f'pct_{name}'] = df[code] / df['B01003_001E'] * 100

    return df


def main(save_per_year=True, save_combined=True):
    all_years = []
    for year in YEARS:
        print(f"Fetching ACS 5-year data for {year}...")
        df = fetch_acs_data_chunked(year, VARIABLES, API_KEY)
        if df is None:
```

```
            print(f"No data returned for {year}, skipping.")
            continue

        df = calculate_derived_variables(df)

        #Picking ick columns to export (only those that exist in df)
        final_cols = [
            'year', 'NAME', 'state', 'county', 'population',
'population_numeric',
            #Age shares and bracket strings
            'pct_male_0_17', 'pct_female_0_17', 'pct_age_0_17_bracket',
            'pct_male_18_24', 'pct_female_18_24', 'pct_age_18_24_bracket',
            'pct_male_25_44', 'pct_female_25_44', 'pct_age_25_44_bracket',
            'pct_male_45_64', 'pct_female_45_64', 'pct_age_45_64_bracket',
            'pct_male_65_plus', 'pct_female_65_plus', 'pct_age_65_plus_bracket',
            'pct_male', 'pct_bachelor_plus', 'unemployment_rate',
'military_per_100k',
            #Race pct fields
            'pct_white', 'pct_black', 'pct_american_indian_alaska_native',
'pct_asian',
            'pct_native_pacific', 'pct_some_other_race',
'pct_two_or_more_races', 'pct_hispanic_or_latino'
        ]

        #Only keeping columns that actually exist
        export_cols = [c for c in final_cols if c in df.columns]
        out_df = df[export_cols].copy()
        #Renaming some columns for readability
        rename_map = {'NAME': 'county_name', 'state': 'state_fips', 'county':
'county_fips'}
        out_df = out_df.rename(columns=rename_map)

        #Ensuring 'population' is nullable int if present
        if 'population' in out_df.columns:
            out_df['population'] = out_df['population'].astype('Int64')

        #Writing per-year file
        if save_per_year:
            filename = f'nc_counties_acs_age_brackets_and_race_{year}.csv'
            out_df.to_csv(filename, index=False)
            print(f"Saved {filename}")

        all_years.append(out_df)

        # polite pacing between years
        time.sleep(0.5)

    if save_combined and all_years:
        combined = pd.concat(all_years, ignore_index=True, sort=False)
```

```
        combined_filename = '../ConsolidatingData/nc_counties_acs_all_years.csv'
        combined.to_csv(combined_filename, index=False)
        print(f"Saved combined file {combined_filename}")

if __name__ == "__main__":
    main(save_per_year=True, save_combined=True)
```

```
#Importing the necessary Python packages to process my data
import camelot
import pandas as pd
import re

#Creating a list of every single pdf to go through
file_names = ["FY-2006.pdf", "FY-2007.pdf", "FY-2008.pdf", "FY-2009.pdf",
"FY-2011.pdf", "FY-2012.pdf",
              "FY-2013.pdf"]

#Creating a list of years for labelling purposes
years = ["2006", "2007", "2008", "2009", "2011", "2012", "2013"]

#Reading in the initial pdf
#tables = camelot.read_pdf("FY-June 2023 Annual Rev by Board_1.pdf",
pages="all", flavor="stream")
#print(f"Found {tables.n} tables")

#List of counties for data use
all_counties = [
    "Alamance", "Alexander", "Alleghany", "Anson", "Ashe", "Avery", "Beaufort",
"Bertie", "Bladen",
    "Brunswick", "Buncombe", "Burke", "Cabarrus", "Caldwell", "Camden",
"Carteret", "Caswell",
    "Catawba", "Chatham", "Cherokee", "Chowan", "Clay", "Cleveland", "Columbus",
"Craven",
    "Cumberland", "Currituck", "Dare", "Davidson", "Davie", "Duplin", "Durham",
"Edgecombe",
    "Forsyth", "Franklin", "Gaston", "Gates", "Graham", "Granville", "Greene",
"Guilford",
    "Halifax", "Harnett", "Haywood", "Henderson", "Hertford", "Hoke", "Hyde",
"Iredell",
    "Jackson", "Johnston", "Jones", "Lee", "Lenoir", "Lincoln", "Macon",
"Madison", "Martin",
    "McDowell", "Mecklenburg", "Mitchell", "Montgomery", "Moore", "Nash", "New
Hanover",
    "Northampton", "Onslow", "Orange", "Pamlico", "Pasquotank", "Pender",
"Perquimans",
    "Person", "Pitt", "Polk", "Randolph", "Richmond", "Robeson", "Rockingham",
"Rowan",
    "Rutherford", "Sampson", "Scotland", "Stanly", "Stokes", "Surry", "Swain",
"Transylvania",
```

```
   "Tyrrell", "Union", "Vance", "Wake", "Warren", "Washington", "Watauga",
"Wayne",
   "Wilkes", "Wilson", "Yadkin", "Yancey"
]

#Using column derived from physical examination of code to rename coluns
columns = [
   "Board Name", "Retail Sales", "Beverage Sales", "Wine Sales", "Gross Sales",
   "Percent Change Over FY", "Total Income", "Cost of Tax", "Operating Sales",
   "Other Expense", "Profit From Operations", "Percent Income & Sales",
   "Profit Before Expense", "Distrib. Law Enf.", "Alcohol Edu. Distrib.",
   "Municipal Income", "Net Income", "Bottles Regular Sold", "Bottles Mix Bev
Sold",
   "Bottles Mini Sold", "# of Stores"
]

#Creating known entities list to filter out empty rows/rows that are not
counties
entities = [
   "Alamance Municipal", "Albemarle", "Andrews", "Angier", "Asheboro",
"Asheville",
   "Beaufort County", "Belville", "Bertie County", "Bessemer City", "Black
Mountain",
   "Blowing Rock", "Boiling Spring Lakes", "Boone", "Brevard", "Brunswick",
   "Brunswick County", "Bryson City", "Bunn", "Burnsville", "Calabash",
   "Camden County", "Canton", "Carteret County", "Caswell County", "Catawba
County",
   "Chatham County", "Cherryville", "Chowan County", "Clay County", "Clinton",
   "Columbus", "Concord", "Cooleemee", "Cramerton", "Craven County",
"Cumberland County",
   "Currituck County", "Dare County", "Dobson", "Dunn", "Durham County",
"Eden",
   "Edgecombe County", "Elizabethtown", "Elkin", "Fairmont", "Fletcher",
"Forest City",
   "Franklin", "Franklinton", "Garland", "Gastonia", "Gates County",
"Gibsonville",
   "Granite Falls", "Granville County", "Greene County", "Greensboro", "Halifax
County",
   "Hamlet", "Hendersonville", "Hertford", "Hertford County", "High Country",
   "High Point", "Highlands", "Hoke County", "Hyde County", "Indian Trail",
   "Johnston County", "Jones County", "Kenansville", "Kings Mountain", "Lake
Lure",
   "Lake Waccamaw", "Laurel Park", "Lenoir City", "Lenoir County", "Lexington",
   "Liberty", "Lillington", "Lincoln County", "Lincolnton", "Locust",
"Louisburg",
   "Lumberton", "Madison", "Maggie Valley", "Marion", "Martin County",
"Maxton",
   "Mecklenburg County", "Monroe", "Montgomery", "Moore County", "Mooresville",
```

```
   "Morganton", "Mount Airy", "Mount Holly", "Mount Pleasant", "Murphy", "Nash
County",
   "New Hanover County", "Newton Grove", "North Wilkesboro", "Northampton
County",
   "Norwood", "Oak Island", "Ocean Isle Beach", "Onslow County", "Orange
County",
   "Pamlico County", "Pasquotank County", "Pender County", "Person County",
   "Pilot Mountain", "Pitt County", "Pittsboro", "Randleman", "Red Springs",
   "Reidsville", "Rockingham", "Roseboro", "Rowan/Kannapolis", "Rowland",
   "Rutherfordton", "Saint Pauls", "Sanford", "Scotland County", "Shallotte",
   "Shelby", "Siler City", "Southport", "Sparta", "Spruce Pine", "Statesville",
   "Sunset Beach", "Sylva", "Tabor City", "Thomasville", "Triad Municipal",
"Tryon",
   "Tyrrell County", "Valdese", "Vance County", "Wadesboro", "Wake County",
"Wallace",
   "Walnut Cove", "Warren County", "Warsaw", "Washington County", "Waxhaw",
   "Wayne County", "Waynesville", "Weaverville", "West Columbus", "West
Jefferson",
   "Whiteville", "Wilkesboro", "Wilson County", "Wingate", "Woodfin",
"Youngsville"
]

#Using known entities dictionary to map individual ABC Boards to counties,
enabling better summing up of data
municipality_to_county = {
   "Davidson County" : "Davidson County", "Marshville" : "Union County",
"Mocksville" : "Davie County", "Mocksville Cooleemee" : "Davie County", "" :
"", "Belmont" : "Gaston County", "Troutman" : "Iredell County", "Alamance
Municipal": "Alamance County", "Albemarle": "Stanly County",
   "Andrews": "Cherokee County", "Angier": "Harnett County", "Asheboro":
"Randolph County",
   "Asheville": "Buncombe County", "Beaufort County": "Beaufort County",
   "Belville": "Brunswick County", "Bertie County": "Bertie County",
   "Bessemer City": "Gaston County", "Black Mountain": "Buncombe County",
   "Blowing Rock": "Watauga County", "Boiling Spring Lakes": "Brunswick
County",
   "Boone": "Watauga County", "Brevard": "Transylvania County", "Brunswick":
"Brunswick County",
   "Brunswick County": "Brunswick County", "Bryson City": "Swain County",
"Bunn": "Franklin County",
   "Burnsville": "Yancey County", "Calabash": "Brunswick County", "Camden
County": "Camden County",
   "Canton": "Haywood County", "Carteret County": "Carteret County", "Caswell
County": "Caswell County",
   "Catawba County": "Catawba County", "Chatham County": "Chatham County",
"Cherryville": "Gaston County",
   "Chowan County": "Chowan County", "Clay County": "Clay County", "Clinton":
"Sampson County",
```

```
   "Columbus": "Polk County", "Concord": "Cabarrus County", "Cooleemee": "Davie
County",
   "Cramerton": "Gaston County", "Craven County": "Craven County", "Cumberland
County": "Cumberland County",
   "Currituck County": "Currituck County", "Dare County": "Dare County",
"Dobson": "Surry County",
   "Dunn": "Harnett County", "Durham County": "Durham County", "Eden":
"Rockingham County",
   "Edgecombe County": "Edgecombe County", "Ramseur" : "Randolph County",
"Elizabethtown": "Bladen County", "Elkin": "Surry County",
   "Fairmont": "Robeson County", "Fletcher": "Henderson County", "Forest City":
"Rutherford County",
   "Franklin": "Macon County", "Franklinton": "Franklin County", "Garland":
"Sampson County",
   "Gastonia": "Gaston County", "Gates County": "Gates County", "Gibsonville":
"Guilford County",
   "Granite Falls": "Caldwell County", "Granville County": "Granville County",
"Greene County": "Greene County",
   "Greensboro": "Guilford County", "Halifax County": "Halifax County",
"Hamlet": "Richmond County",
   "Hendersonville": "Henderson County", "Hertford": "Perquimans County",
"Hertford County": "Hertford County",
   "High Country": "Avery County", "High Point": "Guilford County",
"Highlands": "Macon County",
   "Hoke County": "Hoke County", "Hyde County": "Hyde County", "Indian Trail":
"Union County",
   "Johnston County": "Johnston County", "Jones County": "Jones County",
"Kenansville": "Duplin County",
   "Kings Mountain": "Cleveland County", "Lake Lure": "Rutherford County",
"Lake Waccamaw": "Columbus County",
   "Laurel Park": "Henderson County", "Lenoir City": "Caldwell County", "Lenoir
County": "Lenoir County",
   "Lexington": "Davidson County", "Liberty": "Randolph County", "Lillington":
"Harnett County",
   "Lincoln County": "Lincoln County", "Lincolnton": "Lincoln County",
"Locust": "Stanly County",
   "Louisburg": "Franklin County", "Lumberton": "Robeson County", "Madison":
"Madison County",
   "Maggie Valley": "Haywood County", "Marion": "McDowell County", "Martin
County": "Martin County",
   "Maxton": "Robeson County", "Mecklenburg County": "Mecklenburg County",
"Monroe": "Union County",
   "Montgomery Municipal": "Montgomery County", "Montgomery": "Montgomery
County", "Moore County": "Moore County", "Mooresville": "Iredell County",
   "Morganton": "Burke County", "Mount Airy": "Surry County", "Mount Holly":
"Gaston County",
   "Mount Pleasant": "Cabarrus County", "Murphy": "Cherokee County", "Nash
County": "Nash County",
```

```
   "New Hanover": "New Hanover County", "New Hanover County": "New Hanover
County", "Newton Grove": "Sampson County",
   "North Wilkesboro": "Wilkes County", "Northampton County": "Northampton
County", "Norwood": "Stanly County",
   "Oak Island": "Brunswick County", "Ocean Isle": "Brunswick County", "Ocean
Isle Beach": "Brunswick County", "Onslow County": "Onslow County",
   "Orange County": "Orange County", "Pamlico County": "Pamlico County",
"Pasquotank County": "Pasquotank County",
   "Pender County": "Pender County", "Person County": "Person County", "Pilot
Mountain": "Surry County",
   "Pitt County": "Pitt County", "Pittsboro": "Chatham County", "Randleman":
"Randolph County",
   "Red Springs": "Robeson County", "Reidsville": "Rockingham County",
"Rockingham": "Richmond County",  # FIXED
   "Roseboro": "Sampson County", "Rowan/Kannapolis": "Rowan County", "Rowland":
"Robeson County",
   "Rutherfordton": "Rutherford County", "Saint Pauls": "Robeson County",
"Sanford": "Lee County",
   "Scotland County": "Scotland County", "Shallotte": "Brunswick County",
"Shelby": "Cleveland County",
   "Siler City": "Chatham County", "Southport": "Brunswick County", "Sparta":
"Alleghany County",
   "Spruce Pine": "Mitchell County", "Statesville": "Iredell County", "Sunset
Beach": "Brunswick County",
   "Sylva": "Jackson County", "Tabor City": "Columbus County", "Thomasville":
"Davidson County",
   "Triad Municipal": "Forsyth County", "Tryon": "Polk County", "Tyrrell
County": "Tyrrell County",
   "Valdese": "Burke County", "Vance County": "Vance County", "Wadesboro":
"Anson County",
   "Wake County": "Wake County", "Wallace": "Duplin County", "Walnut Cove":
"Stokes County",
   "Warren County": "Warren County", "Warsaw": "Duplin County", "Washington
County": "Washington County",
   "Waxhaw": "Union County", "Wayne County": "Wayne County", "Waynesville":
"Haywood County",
   "Weaverville": "Buncombe County", "West Columbus": "Columbus County", "West
Jefferson": "Ashe County",  # FIXED
   "Whiteville": "Columbus County", "Wilkesboro": "Wilkes County", "Wilson
County": "Wilson County",
   "Wingate": "Union County", "Woodfin": "Buncombe County", "Youngsville":
"Franklin County", "Pembroke": "Robeson County",
   "Elkin/Yadkin Valley": "Yadkin County", "Jackson County": "Jackson County",
"Yadkin Valley": "Yadkin County", "Taylorsville" : "Alexander County",
   "Alexander County" : "Alexander County"
}

#Creating a function that cleans out names to aid in processing
def clean_board_name(name):
```

```
    """
    Clean board names by:
    - Removing 'cid'
    - Removing parentheses and colons
    - Removing trailing numbers
    - Keeping only letters, spaces, and forward slashes
    - Consolidating multiple spaces
    """
    name = str(name)
    name = re.sub(r'cid', ' ', name)                 #Removing 'cid'
    name = re.sub(r'[\(\):]', ' ', name)             #Removing parentheses and
colons
    name = re.sub(r'\d+$', ' ', name)                #Removing trailing numbers
    name = re.sub(r'[^a-zA-Z\s/]', ' ', name)        #Keeping only letters,
spaces, and '/'
    name = ' '.join(name.split())                   #Collapsing multiple spaces
    return name.strip()

def consolidating_tables(name):
#Combining all tables first, WITHOUT filtering, to get a single table (it
breaks it up by page)
    df_list = []
    for t in name:
        df = t.df
        #print(df)
        #Removing completely empty rows
        df = df[df[0].str.strip() != ""]
        df_list.append(df)

    super_table = pd.concat(df_list, ignore_index=True)
    return super_table

def cleaning_tables (name):
    #Cleaning the board names (remove extra whitespace, numbers at end) using my
function
    name['Original_Name'] = name[0]  #Keeping original for debugging
    name['Cleaned_Name'] = name[0].apply(clean_board_name)

    #Filtering out header/footer rows and very short entries
    header_footer_keywords = ['ABC Board', 'Board Name', 'Audit', 'Total',
'Percent', 'Retail', 'Mixed', 'Totals', 'BoardName']
    name = name[
        ~name['Cleaned_Name'].isin(header_footer_keywords) &
        (name['Cleaned_Name'].str.len() > 3)
    ]

    #Mapping with the cleaned names to enable aggreagation
    name['County'] = name['Cleaned_Name'].map(municipality_to_county)
    return name
```

```
#CODE THAT WAS USED FOR DEBUGGING THE ORIGINAL PROGRAM
#Checking unmatched entries
#unmapped = super_table[super_table['County'].isna()]
#print(f"\nTotal rows after cleaning: {len(super_table)}")
#print(f"Successfully mapped: {super_table['County'].notna().sum()}")
#print(f"Unmapped boards: {len(unmapped)}")

#if len(unmapped) > 0:
#    print("\nBoards that failed to map:")
#    for orig, cleaned in zip(unmapped['Original_Name'].unique(),
unmapped['Cleaned_Name'].unique()):
#        print(f"  Original: '{orig}' -> Cleaned: '{cleaned}'")

#pd.set_option('display.max_rows', None)       #Showing all rows
#pd.set_option('display.max_columns', None)    #Showing all columns
#pd.set_option('display.width', None)          #No line wrapping
#pd.set_option('display.max_colwidth', None)   #Showing full column content


#Filtering to only successfully mapped rows
#super_table = super_table[super_table['County'].notna()]

#print(super_table)
#print(super_table['County'].nunique())

def numeric_conversion(name):
   #Converting numeric columns
   numeric_cols = list(range(1, len(name.columns) - 3))  # Exclude
Original_Name, Cleaned_Name, County

   for col in numeric_cols:
       name[col] = name[col].astype(str)
       name[col] = name[col].str.replace(",", "")
       name[col] = name[col].str.extract(r"([-+]?\d*\.?\d+)")[0]
       name[col] = pd.to_numeric(name[col], errors='coerce').fillna(0)

   #Aggregating by County
   name = name.groupby('County')[numeric_cols].sum().reset_index()
   return name

#print(f"\nFinal county count: {len(county_table)}")
#print(county_table['County'].unique())


def final_step(name):
   numeric_cols = [
       c for c in name.columns
       if c not in ['Original_Name', 'Cleaned_Name', 'County', 0]
```

```
    ]

    rename_map = {
        col: new_name
        for col, new_name in zip(numeric_cols, columns[1:])
    }

    name = name.rename(columns=rename_map)
    return name

def output(name, i):
    csv_file = "nc_counties_" + years[i] + ".csv"
    name.to_csv(csv_file, index=False)  # index=False avoids adding the row
numbers


#USED TO IDENTIFY MISSING ROWS
#county_table = county_table.rename(columns=rename_map)
#names = county_table['County'].str.replace(' County', '',
regex=False).tolist()
#names = set(names)
#other_names = set(all_counties)
#print(other_names - names)

#csv_file = "nc_counties_2023.csv"  #Specifying file for output
#county_table.to_csv(csv_file, index=False)  # index=False avoids adding the
row numbers

def enforce_all_counties(df, all_counties, year_label):
    full_list = [c + " County" for c in all_counties]

    present = set(df["County"])
    missing = set(full_list) - present

    if missing:
        print(f"\nWARNING ({year_label}): Missing counties being filled with
zeros:")
        print(sorted(missing))

    df = df.set_index("County")
    df = df.reindex(full_list, fill_value=0)
    df = df.reset_index()

    assert len(df) == 100, f"County count mismatch in {year_label}"

    return df

def main():
    for i in range(0, len(file_names)):
```

```
        file_path = "AlcoholSalesPDFs/" + file_names[i]

        tables = camelot.read_pdf(file_path, pages="all", flavor="stream")

        name = consolidating_tables(tables)
        name = cleaning_tables(name)
        name = numeric_conversion(name)

        #ENFORCE ALL 100 COUNTIES HERE
        name = enforce_all_counties(name, all_counties, years[i])

        name = final_step(name)
        output(name, i)

if __name__ == "__main__":
    main()
```

```
#Importing the necessary Python packages to process my data
import camelot
import pandas as pd
import re

#Creating a list of every single pdf to go through
file_names = ["FY-2014.pdf", "FY-2015.pdf", "FY-2016.pdf", "FY-2017.pdf",
"FY-2018.pdf", "FY-2019.pdf",
              "FY-2020.pdf", "FY-2021.pdf", "=FY-2022 Annual Rev by Board=.pdf"]

#Creating a list of years for labelling purposes
years = ["2014", "2015", "2016", "2017", "2018", "2019",
         "2020", "2021", "2022"]

#List of counties for data use
all_counties = [
    "Alamance", "Alexander", "Alleghany", "Anson", "Ashe", "Avery", "Beaufort",
"Bertie", "Bladen",
    "Brunswick", "Buncombe", "Burke", "Cabarrus", "Caldwell", "Camden",
"Carteret", "Caswell",
    "Catawba", "Chatham", "Cherokee", "Chowan", "Clay", "Cleveland", "Columbus",
"Craven",
    "Cumberland", "Currituck", "Dare", "Davidson", "Davie", "Duplin", "Durham",
"Edgecombe",
    "Forsyth", "Franklin", "Gaston", "Gates", "Graham", "Granville", "Greene",
"Guilford",
    "Halifax", "Harnett", "Haywood", "Henderson", "Hertford", "Hoke", "Hyde",
"Iredell",
    "Jackson", "Johnston", "Jones", "Lee", "Lenoir", "Lincoln", "Macon",
"Madison", "Martin",
    "McDowell", "Mecklenburg", "Mitchell", "Montgomery", "Moore", "Nash", "New
Hanover",
```

```
   "Northampton", "Onslow", "Orange", "Pamlico", "Pasquotank", "Pender",
"Perquimans",
   "Person", "Pitt", "Polk", "Randolph", "Richmond", "Robeson", "Rockingham",
"Rowan",
   "Rutherford", "Sampson", "Scotland", "Stanly", "Stokes", "Surry", "Swain",
"Transylvania",
   "Tyrrell", "Union", "Vance", "Wake", "Warren", "Washington", "Watauga",
"Wayne",
   "Wilkes", "Wilson", "Yadkin", "Yancey"
]

#Using column derived from physical examination of code to rename coluns
columns_1 = [
   "Board Name", "Retail Sales", "Beverage Sales", "Wine Sales", "Gross Sales",
   "Percent Change Over FY", "Total Income", "Operating Sales",
   "Other Expense", "Profit From Operations", "Percent Income & Sales",
   "Profit Before Expense", "Distrib. Law Enf.", "Alcohol Edu. Distrib.",
   "Municipal Income", "Net Income", "Bottles Regular Sold", "Bottles Mix Bev
Sold",
   "Bottles Mini Sold", "# of Stores"
]

columns_2 = [
   "Board Name", "# of Stores", "Retail Sales", "Beverage Sales", "Wine Sales",
"Gross Sales",
   "Percent Change Over FY", "Total Income", "Operating Sales",
   "Other Expense", "Profit From Operations", "Percent Income & Sales",
   "Profit Before Expense", "Distrib. Law Enf.", "Alcohol Edu. Distrib.",
   "Municipal Income", "Net Income", "Bottles Regular Sold", "Bottles Mix Bev
Sold",
   "Bottles Mini Sold"
]

#Creating known entities list to filter out empty rows/rows that are not
counties
entities = [
   "Alamance Municipal", "Albemarle", "Andrews", "Angier", "Asheboro",
"Asheville",
   "Beaufort County", "Belville", "Bertie County", "Bessemer City", "Black
Mountain",
   "Blowing Rock", "Boiling Spring Lakes", "Boone", "Brevard", "Brunswick",
   "Brunswick County", "Bryson City", "Bunn", "Burnsville", "Calabash",
   "Camden County", "Canton", "Carteret County", "Caswell County", "Catawba
County",
   "Chatham County", "Cherryville", "Chowan County", "Clay County", "Clinton",
   "Columbus", "Concord", "Cooleemee", "Cramerton", "Craven County",
"Cumberland County",
   "Currituck County", "Dare County", "Dobson", "Dunn", "Durham County",
"Eden",
```

```
   "Edgecombe County", "Elizabethtown", "Elkin", "Fairmont", "Fletcher",
"Forest City",
   "Franklin", "Franklinton", "Garland", "Gastonia", "Gates County",
"Gibsonville",
   "Granite Falls", "Granville County", "Greene County", "Greensboro", "Halifax
County",
   "Hamlet", "Hendersonville", "Hertford", "Hertford County", "High Country",
   "High Point", "Highlands", "Hoke County", "Hyde County", "Indian Trail",
   "Johnston County", "Jones County", "Kenansville", "Kings Mountain", "Lake
Lure",
   "Lake Waccamaw", "Laurel Park", "Lenoir City", "Lenoir County", "Lexington",
   "Liberty", "Lillington", "Lincoln County", "Lincolnton", "Locust",
"Louisburg",
   "Lumberton", "Madison", "Maggie Valley", "Marion", "Martin County",
"Maxton",
   "Mecklenburg County", "Monroe", "Montgomery", "Moore County", "Mooresville",
   "Morganton", "Mount Airy", "Mount Holly", "Mount Pleasant", "Murphy", "Nash
County",
   "New Hanover County", "Newton Grove", "North Wilkesboro", "Northampton
County",
   "Norwood", "Oak Island", "Ocean Isle Beach", "Onslow County", "Orange
County",
   "Pamlico County", "Pasquotank County", "Pender County", "Person County",
   "Pilot Mountain", "Pitt County", "Pittsboro", "Randleman", "Red Springs",
   "Reidsville", "Rockingham", "Roseboro", "Rowan/Kannapolis", "Rowland",
   "Rutherfordton", "Saint Pauls", "Sanford", "Scotland County", "Shallotte",
   "Shelby", "Siler City", "Southport", "Sparta", "Spruce Pine", "Statesville",
   "Sunset Beach", "Sylva", "Tabor City", "Thomasville", "Triad Municipal",
"Tryon",
   "Tyrrell County", "Valdese", "Vance County", "Wadesboro", "Wake County",
"Wallace",
   "Walnut Cove", "Warren County", "Warsaw", "Washington County", "Waxhaw",
   "Wayne County", "Waynesville", "Weaverville", "West Columbus", "West
Jefferson",
   "Whiteville", "Wilkesboro", "Wilson County", "Wingate", "Woodfin",
"Youngsville"
]

#Using known entities dictionary to map individual ABC Boards to counties,
enabling better summing up of data
municipality_to_county = {
   "Davidson County" : "Davidson County", "Marshville" : "Union County",
"Mocksville" : "Davie County", "Mocksville Cooleemee" : "Davie County", "" :
"", "Belmont" : "Gaston County", "Troutman" : "Iredell County", "Alamance
Municipal": "Alamance County", "Albemarle": "Stanly County",
   "Andrews": "Cherokee County", "Angier": "Harnett County", "Asheboro":
"Randolph County",
   "Asheville": "Buncombe County", "Beaufort County": "Beaufort County",
   "Belville": "Brunswick County", "Bertie County": "Bertie County",
```

```
   "Bessemer City": "Gaston County", "Black Mountain": "Buncombe County",
   "Blowing Rock": "Watauga County", "Boiling Spring Lakes": "Brunswick
County",
   "Boone": "Watauga County", "Brevard": "Transylvania County", "Brunswick":
"Brunswick County",
   "Brunswick County": "Brunswick County", "Bryson City": "Swain County",
"Bunn": "Franklin County",
   "Burnsville": "Yancey County", "Calabash": "Brunswick County", "Camden
County": "Camden County",
   "Canton": "Haywood County", "Carteret County": "Carteret County", "Caswell
County": "Caswell County",
   "Catawba County": "Catawba County", "Chatham County": "Chatham County",
"Cherryville": "Gaston County",
   "Chowan County": "Chowan County", "Clay County": "Clay County", "Clinton":
"Sampson County",
   "Columbus": "Polk County", "Concord": "Cabarrus County", "Cooleemee": "Davie
County",
   "Cramerton": "Gaston County", "Craven County": "Craven County", "Cumberland
County": "Cumberland County",
   "Currituck County": "Currituck County", "Dare County": "Dare County",
"Dobson": "Surry County",
   "Dunn": "Harnett County", "Durham County": "Durham County", "Eden":
"Rockingham County",
   "Edgecombe County": "Edgecombe County", "Ramseur" : "Randolph County",
"Elizabethtown": "Bladen County", "Elkin": "Surry County",
   "Fairmont": "Robeson County", "Fletcher": "Henderson County", "Forest City":
"Rutherford County",
   "Franklin": "Macon County", "Franklinton": "Franklin County", "Garland":
"Sampson County",  # FIXED
   "Gastonia": "Gaston County", "Gates County": "Gates County", "Gibsonville":
"Guilford County",
   "Granite Falls": "Caldwell County", "Granville County": "Granville County",
"Greene County": "Greene County",
   "Greensboro": "Guilford County", "Halifax County": "Halifax County",
"Hamlet": "Richmond County",
   "Hendersonville": "Henderson County", "Hertford": "Perquimans County",
"Hertford County": "Hertford County",  # FIXED
   "High Country": "Avery County", "High Point": "Guilford County",
"Highlands": "Macon County",  # FIXED
   "Hoke County": "Hoke County", "Hyde County": "Hyde County", "Indian Trail":
"Union County",
   "Johnston County": "Johnston County", "Jones County": "Jones County",
"Kenansville": "Duplin County",
   "Kings Mountain": "Cleveland County", "Lake Lure": "Rutherford County",
"Lake Waccamaw": "Columbus County",
   "Laurel Park": "Henderson County", "Lenoir City": "Caldwell County", "Lenoir
County": "Lenoir County",
   "Lexington": "Davidson County", "Liberty": "Randolph County", "Lillington":
"Harnett County",
```

```
   "Lincoln County": "Lincoln County", "Lincolnton": "Lincoln County",
"Locust": "Stanly County",
   "Louisburg": "Franklin County", "Lumberton": "Robeson County", "Madison":
"Madison County",
   "Maggie Valley": "Haywood County", "Marion": "McDowell County", "Martin
County": "Martin County",
   "Maxton": "Robeson County", "Mecklenburg County": "Mecklenburg County",
"Monroe": "Union County",
   "Montgomery Municipal": "Montgomery County", "Montgomery": "Montgomery
County", "Moore County": "Moore County", "Mooresville": "Iredell County",
   "Morganton": "Burke County", "Mount Airy": "Surry County", "Mount Holly":
"Gaston County",
   "Mount Pleasant": "Cabarrus County", "Murphy": "Cherokee County", "Nash
County": "Nash County",
   "New Hanover": "New Hanover County", "New Hanover County": "New Hanover
County", "Newton Grove": "Sampson County",
   "North Wilkesboro": "Wilkes County", "Northampton County": "Northampton
County", "Norwood": "Stanly County",
   "Oak Island": "Brunswick County", "Ocean Isle": "Brunswick County", "Ocean
Isle Beach": "Brunswick County", "Onslow County": "Onslow County",
   "Orange County": "Orange County", "Pamlico County": "Pamlico County",
"Pasquotank County": "Pasquotank County",
   "Pender County": "Pender County", "Person County": "Person County", "Pilot
Mountain": "Surry County",
   "Pitt County": "Pitt County", "Pittsboro": "Chatham County", "Randleman":
"Randolph County",
   "Red Springs": "Robeson County", "Reidsville": "Rockingham County",
"Rockingham": "Richmond County",  # FIXED
   "Roseboro": "Sampson County", "Rowan/Kannapolis": "Rowan County", "Rowland":
"Robeson County",
   "Rutherfordton": "Rutherford County", "Saint Pauls": "Robeson County",
"Sanford": "Lee County",
   "Scotland County": "Scotland County", "Shallotte": "Brunswick County",
"Shelby": "Cleveland County",
   "Siler City": "Chatham County", "Southport": "Brunswick County", "Sparta":
"Alleghany County",
   "Spruce Pine": "Mitchell County", "Statesville": "Iredell County", "Sunset
Beach": "Brunswick County",
   "Sylva": "Jackson County", "Tabor City": "Columbus County", "Thomasville":
"Davidson County",
   "Triad Municipal": "Forsyth County", "Tryon": "Polk County", "Tyrrell
County": "Tyrrell County",
   "Valdese": "Burke County", "Vance County": "Vance County", "Wadesboro":
"Anson County",
   "Wake County": "Wake County", "Wallace": "Duplin County", "Walnut Cove":
"Stokes County",
   "Warren County": "Warren County", "Warsaw": "Duplin County", "Washington
County": "Washington County",
```

```
   "Waxhaw": "Union County", "Wayne County": "Wayne County", "Waynesville":
"Haywood County",
   "Weaverville": "Buncombe County", "West Columbus": "Columbus County", "West
Jefferson": "Ashe County",  # FIXED
   "Whiteville": "Columbus County", "Wilkesboro": "Wilkes County", "Wilson
County": "Wilson County",
   "Wingate": "Union County", "Woodfin": "Buncombe County", "Youngsville":
"Franklin County", "Pembroke": "Robeson County",
   "Elkin/Yadkin Valley": "Yadkin County", "Jackson County": "Jackson County",
"Yadkin Valley": "Yadkin County", "Taylorsville" : "Alexander County",
   "Alexander County" : "Alexander County"
}

#Creating a function that cleans out names to aid in processing
def clean_board_name(name):
   """
   Clean board names by:
   - Removing 'cid'
   - Removing parentheses and colons
   - Removing trailing numbers
   - Keeping only letters, spaces, and forward slashes
   - Consolidating multiple spaces
   """
   name = str(name)
   name = re.sub(r'cid', ' ', name)                #Removing 'cid'
   name = re.sub(r'[\(\):]', ' ', name)            #Removing parentheses and
colons
   name = re.sub(r'\d+$', ' ', name)               #Removing trailing numbers
   name = re.sub(r'[^a-zA-Z\s/]', ' ', name)       #Keeping only letters,
spaces, and '/'
   name = ' '.join(name.split())                  #Collapsing multiple spaces
   return name.strip()

def consolidating_tables(name):
#Combining all tables first, WITHOUT filtering, to get a single table (it
breaks it up by page)
   df_list = []
   for t in name:
       df = t.df
       #print(df)
       #Removing completely empty rows
       df = df[df[0].str.strip() != ""]
       df_list.append(df)

   super_table = pd.concat(df_list, ignore_index=True)
   return super_table

def cleaning_tables (name):
```

```
    #Cleaning the board names (remove extra whitespace, numbers at end) using my
function
    name['Original_Name'] = name[0]  #Keeping original for debugging
    name['Cleaned_Name'] = name[0].apply(clean_board_name)

    #Filtering out header/footer rows and very short entries
    header_footer_keywords = ['ABC Board', 'Board Name', 'Audit', 'Total',
'Percent', 'Retail', 'Mixed', 'Totals', 'BoardName']
    name = name[
        ~name['Cleaned_Name'].isin(header_footer_keywords) &
        (name['Cleaned_Name'].str.len() > 3)
    ]

    #Mapping with the cleaned names to enable aggreagation
    name['County'] = name['Cleaned_Name'].map(municipality_to_county)
    return name

#CODE THAT WAS USED FOR DEBUGGING THE ORIGINAL PROGRAM
#Checking unmatched entries
#unmapped = super_table[super_table['County'].isna()]
#print(f"\nTotal rows after cleaning: {len(super_table)}")
#print(f"Successfully mapped: {super_table['County'].notna().sum()}")
#print(f"Unmapped boards: {len(unmapped)}")

#if len(unmapped) > 0:
#    print("\nBoards that failed to map:")
#    for orig, cleaned in zip(unmapped['Original_Name'].unique(),
unmapped['Cleaned_Name'].unique()):
#        print(f"  Original: '{orig}' -> Cleaned: '{cleaned}'")

#pd.set_option('display.max_rows', None)       #Showing all rows
#pd.set_option('display.max_columns', None)    #Showing all columns
#pd.set_option('display.width', None)          #No line wrapping
#pd.set_option('display.max_colwidth', None)   #Showing full column content


#Filtering to only successfully mapped rows
#super_table = super_table[super_table['County'].notna()]

#print(super_table)
#print(super_table['County'].nunique())

def numeric_conversion(name):
    #Converting numeric columns
    numeric_cols = list(range(1, len(name.columns) - 3))  # Exclude
Original_Name, Cleaned_Name, County

    for col in numeric_cols:
        name[col] = name[col].astype(str)
```

```
        name[col] = name[col].str.replace(",", "")
        name[col] = name[col].str.extract(r"([-+]?\d*\.?\d+)")[0]
        name[col] = pd.to_numeric(name[col], errors='coerce').fillna(0)

    #Aggregating by County
    name = name.groupby('County')[numeric_cols].sum().reset_index()
    return name

#print(f"\nFinal county count: {len(county_table)}")
#print(county_table['County'].unique())


def final_step(name):
    numeric_cols = [
        c for c in name.columns
        if c not in ['Original_Name', 'Cleaned_Name', 'County', 0]
    ]

    rename_map = {
        col: new_name
        for col, new_name in zip(numeric_cols, columns_1[1:])
    }

    name = name.rename(columns=rename_map)
    return name

def output(name, i):
    csv_file = "nc_counties_" + years[i] + ".csv"
    name.to_csv(csv_file, index=False)  # index=False avoids adding the row
numbers


#USED TO IDENTIFY MISSING ROWS
#county_table = county_table.rename(columns=rename_map)
#names = county_table['County'].str.replace(' County', '',
regex=False).tolist()
#names = set(names)
#other_names = set(all_counties)
#print(other_names - names)

#csv_file = "nc_counties_2023.csv"  #Specifying file for output
#county_table.to_csv(csv_file, index=False)  # index=False avoids adding the
row numbers

def enforce_all_counties(df, all_counties, year_label):
    full_list = [c + " County" for c in all_counties]

    present = set(df["County"])
    missing = set(full_list) - present
```

```
    if missing:
        print(f"\nWARNING ({year_label}): Missing counties being filled with
zeros:")
        print(sorted(missing))

    df = df.set_index("County")
    df = df.reindex(full_list, fill_value=0)
    df = df.reset_index()

    assert len(df) == 100, f"County count mismatch in {year_label}"

    return df

def main():
    for i in range(0, len(file_names)):
        file_path = "AlcoholSalesPDFs/" + file_names[i]

        tables = camelot.read_pdf(file_path, pages="all", flavor="stream")

        name = consolidating_tables(tables)
        name = cleaning_tables(name)
        name = numeric_conversion(name)

        #ENFORCE ALL 100 COUNTIES HERE
        name = enforce_all_counties(name, all_counties, years[i])

        name = final_step(name)
        output(name, i)

if __name__ == "__main__":
    main()
```

```
#Importing the necessary Python packages to process my data
import camelot
import pandas as pd
import re

#Creating a list of every single pdf to go through
file_names = ["FY-June 2023 Annual Rev by Board_1.pdf",
              "ANNUAL REV @ FY-June-2024-2.pdf"]

#Creating a list of years for labelling purposes
years = ["2023", "2024"]

#Reading in the initial pdf
#tables = camelot.read_pdf("FY-June 2023 Annual Rev by Board_1.pdf",
pages="all", flavor="stream")
#print(f"Found {tables.n} tables")
```

```
#List of counties for data use
all_counties = [
   "Alamance", "Alexander", "Alleghany", "Anson", "Ashe", "Avery", "Beaufort",
"Bertie", "Bladen",
   "Brunswick", "Buncombe", "Burke", "Cabarrus", "Caldwell", "Camden",
"Carteret", "Caswell",
   "Catawba", "Chatham", "Cherokee", "Chowan", "Clay", "Cleveland", "Columbus",
"Craven",
   "Cumberland", "Currituck", "Dare", "Davidson", "Davie", "Duplin", "Durham",
"Edgecombe",
   "Forsyth", "Franklin", "Gaston", "Gates", "Graham", "Granville", "Greene",
"Guilford",
   "Halifax", "Harnett", "Haywood", "Henderson", "Hertford", "Hoke", "Hyde",
"Iredell",
   "Jackson", "Johnston", "Jones", "Lee", "Lenoir", "Lincoln", "Macon",
"Madison", "Martin",
   "McDowell", "Mecklenburg", "Mitchell", "Montgomery", "Moore", "Nash", "New
Hanover",
   "Northampton", "Onslow", "Orange", "Pamlico", "Pasquotank", "Pender",
"Perquimans",
   "Person", "Pitt", "Polk", "Randolph", "Richmond", "Robeson", "Rockingham",
"Rowan",
   "Rutherford", "Sampson", "Scotland", "Stanly", "Stokes", "Surry", "Swain",
"Transylvania",
   "Tyrrell", "Union", "Vance", "Wake", "Warren", "Washington", "Watauga",
"Wayne",
   "Wilkes", "Wilson", "Yadkin", "Yancey"
]

#Using column derived from physical examination of code to rename coluns
columns = [
   "Board Name", "# of Stores", "Retail Sales", "Beverage Sales", "Wine Sales",
"Gross Sales",
   "Percent Change Over FY", "Total Income", "Operating Sales",
   "Other Expense", "Profit From Operations", "Percent Income & Sales",
   "Profit Before Expense", "Distrib. Law Enf.", "Alcohol Edu. Distrib.",
   "Municipal Income", "Net Income", "Bottles Regular Sold", "Bottles Mix Bev
Sold",
   "Bottles Mini Sold"
]

#Creating known entities list to filter out empty rows/rows that are not
counties
entities = [
   "Alamance Municipal", "Albemarle", "Andrews", "Angier", "Asheboro",
"Asheville",
   "Beaufort County", "Belville", "Bertie County", "Bessemer City", "Black
Mountain",
```

```
   "Blowing Rock", "Boiling Spring Lakes", "Boone", "Brevard", "Brunswick",
   "Brunswick County", "Bryson City", "Bunn", "Burnsville", "Calabash",
   "Camden County", "Canton", "Carteret County", "Caswell County", "Catawba
County",
   "Chatham County", "Cherryville", "Chowan County", "Clay County", "Clinton",
   "Columbus", "Concord", "Cooleemee", "Cramerton", "Craven County",
"Cumberland County",
   "Currituck County", "Dare County", "Dobson", "Dunn", "Durham County",
"Eden",
   "Edgecombe County", "Elizabethtown", "Elkin", "Fairmont", "Fletcher",
"Forest City",
   "Franklin", "Franklinton", "Garland", "Gastonia", "Gates County",
"Gibsonville",
   "Granite Falls", "Granville County", "Greene County", "Greensboro", "Halifax
County",
   "Hamlet", "Hendersonville", "Hertford", "Hertford County", "High Country",
   "High Point", "Highlands", "Hoke County", "Hyde County", "Indian Trail",
   "Johnston County", "Jones County", "Kenansville", "Kings Mountain", "Lake
Lure",
   "Lake Waccamaw", "Laurel Park", "Lenoir City", "Lenoir County", "Lexington",
   "Liberty", "Lillington", "Lincoln County", "Lincolnton", "Locust",
"Louisburg",
   "Lumberton", "Madison", "Maggie Valley", "Marion", "Martin County",
"Maxton",
   "Mecklenburg County", "Monroe", "Montgomery", "Moore County", "Mooresville",
   "Morganton", "Mount Airy", "Mount Holly", "Mount Pleasant", "Murphy", "Nash
County",
   "New Hanover County", "Newton Grove", "North Wilkesboro", "Northampton
County",
   "Norwood", "Oak Island", "Ocean Isle Beach", "Onslow County", "Orange
County",
   "Pamlico County", "Pasquotank County", "Pender County", "Person County",
   "Pilot Mountain", "Pitt County", "Pittsboro", "Randleman", "Red Springs",
   "Reidsville", "Rockingham", "Roseboro", "Rowan/Kannapolis", "Rowland",
   "Rutherfordton", "Saint Pauls", "Sanford", "Scotland County", "Shallotte",
   "Shelby", "Siler City", "Southport", "Sparta", "Spruce Pine", "Statesville",
   "Sunset Beach", "Sylva", "Tabor City", "Thomasville", "Triad Municipal",
"Tryon",
   "Tyrrell County", "Valdese", "Vance County", "Wadesboro", "Wake County",
"Wallace",
   "Walnut Cove", "Warren County", "Warsaw", "Washington County", "Waxhaw",
   "Wayne County", "Waynesville", "Weaverville", "West Columbus", "West
Jefferson",
   "Whiteville", "Wilkesboro", "Wilson County", "Wingate", "Woodfin",
"Youngsville"
]

#Using known entities dictionary to map individual ABC Boards to counties,
enabling better summing up of data
```

```
municipality_to_county = {
   "Davidson County" : "Davidson County", "Marshville" : "Union County",
"Mocksville" : "Davie County", "Mocksville Cooleemee" : "Davie County", "" :
"", "Belmont" : "Gaston County", "Troutman" : "Iredell County", "Alamance
Municipal": "Alamance County", "Albemarle": "Stanly County",
   "Andrews": "Cherokee County", "Angier": "Harnett County", "Asheboro":
"Randolph County",
   "Asheville": "Buncombe County", "Beaufort County": "Beaufort County",
   "Belville": "Brunswick County", "Bertie County": "Bertie County",
   "Bessemer City": "Gaston County", "Black Mountain": "Buncombe County",
   "Blowing Rock": "Watauga County", "Boiling Spring Lakes": "Brunswick
County",
   "Boone": "Watauga County", "Brevard": "Transylvania County", "Brunswick":
"Brunswick County",
   "Brunswick County": "Brunswick County", "Bryson City": "Swain County",
"Bunn": "Franklin County",
   "Burnsville": "Yancey County", "Calabash": "Brunswick County", "Camden
County": "Camden County",
   "Canton": "Haywood County", "Carteret County": "Carteret County", "Caswell
County": "Caswell County",
   "Catawba County": "Catawba County", "Chatham County": "Chatham County",
"Cherryville": "Gaston County",
   "Chowan County": "Chowan County", "Clay County": "Clay County", "Clinton":
"Sampson County",
   "Columbus": "Polk County", "Concord": "Cabarrus County", "Cooleemee": "Davie
County",
   "Cramerton": "Gaston County", "Craven County": "Craven County", "Cumberland
County": "Cumberland County",
   "Currituck County": "Currituck County", "Dare County": "Dare County",
"Dobson": "Surry County",
   "Dunn": "Harnett County", "Durham County": "Durham County", "Eden":
"Rockingham County",
   "Edgecombe County": "Edgecombe County", "Ramseur" : "Randolph County",
"Elizabethtown": "Bladen County", "Elkin": "Surry County",
   "Fairmont": "Robeson County", "Fletcher": "Henderson County", "Forest City":
"Rutherford County",
   "Franklin": "Macon County", "Franklinton": "Franklin County", "Garland":
"Sampson County",  # FIXED
   "Gastonia": "Gaston County", "Gates County": "Gates County", "Gibsonville":
"Guilford County",
   "Granite Falls": "Caldwell County", "Granville County": "Granville County",
"Greene County": "Greene County",
   "Greensboro": "Guilford County", "Halifax County": "Halifax County",
"Hamlet": "Richmond County",
   "Hendersonville": "Henderson County", "Hertford": "Perquimans County",
"Hertford County": "Hertford County",  # FIXED
   "High Country": "Avery County", "High Point": "Guilford County",
"Highlands": "Macon County",  # FIXED
```

```
   "Hoke County": "Hoke County", "Hyde County": "Hyde County", "Indian Trail":
"Union County",
   "Johnston County": "Johnston County", "Jones County": "Jones County",
"Kenansville": "Duplin County",
   "Kings Mountain": "Cleveland County", "Lake Lure": "Rutherford County",
"Lake Waccamaw": "Columbus County",
   "Laurel Park": "Henderson County", "Lenoir City": "Caldwell County", "Lenoir
County": "Lenoir County",
   "Lexington": "Davidson County", "Liberty": "Randolph County", "Lillington":
"Harnett County",
   "Lincoln County": "Lincoln County", "Lincolnton": "Lincoln County",
"Locust": "Stanly County",
   "Louisburg": "Franklin County", "Lumberton": "Robeson County", "Madison":
"Madison County",
   "Maggie Valley": "Haywood County", "Marion": "McDowell County", "Martin
County": "Martin County",
   "Maxton": "Robeson County", "Mecklenburg County": "Mecklenburg County",
"Monroe": "Union County",
   "Montgomery Municipal": "Montgomery County", "Montgomery": "Montgomery
County", "Moore County": "Moore County", "Mooresville": "Iredell County",
   "Morganton": "Burke County", "Mount Airy": "Surry County", "Mount Holly":
"Gaston County",
   "Mount Pleasant": "Cabarrus County", "Murphy": "Cherokee County", "Nash
County": "Nash County",
   "New Hanover": "New Hanover County", "New Hanover County": "New Hanover
County", "Newton Grove": "Sampson County",
   "North Wilkesboro": "Wilkes County", "Northampton County": "Northampton
County", "Norwood": "Stanly County",
   "Oak Island": "Brunswick County", "Ocean Isle": "Brunswick County", "Ocean
Isle Beach": "Brunswick County", "Onslow County": "Onslow County",
   "Orange County": "Orange County", "Pamlico County": "Pamlico County",
"Pasquotank County": "Pasquotank County",
   "Pender County": "Pender County", "Person County": "Person County", "Pilot
Mountain": "Surry County",
   "Pitt County": "Pitt County", "Pittsboro": "Chatham County", "Randleman":
"Randolph County",
   "Red Springs": "Robeson County", "Reidsville": "Rockingham County",
"Rockingham": "Richmond County",  # FIXED
   "Roseboro": "Sampson County", "Rowan/Kannapolis": "Rowan County", "Rowland":
"Robeson County",
   "Rutherfordton": "Rutherford County", "Saint Pauls": "Robeson County",
"Sanford": "Lee County",
   "Scotland County": "Scotland County", "Shallotte": "Brunswick County",
"Shelby": "Cleveland County",
   "Siler City": "Chatham County", "Southport": "Brunswick County", "Sparta":
"Alleghany County",
   "Spruce Pine": "Mitchell County", "Statesville": "Iredell County", "Sunset
Beach": "Brunswick County",
```

```
    "Sylva": "Jackson County", "Tabor City": "Columbus County", "Thomasville":
"Davidson County",
    "Triad Municipal": "Forsyth County", "Tryon": "Polk County", "Tyrrell
County": "Tyrrell County",
    "Valdese": "Burke County", "Vance County": "Vance County", "Wadesboro":
"Anson County",
    "Wake County": "Wake County", "Wallace": "Duplin County", "Walnut Cove":
"Stokes County",
    "Warren County": "Warren County", "Warsaw": "Duplin County", "Washington
County": "Washington County",
    "Waxhaw": "Union County", "Wayne County": "Wayne County", "Waynesville":
"Haywood County",
    "Weaverville": "Buncombe County", "West Columbus": "Columbus County", "West
Jefferson": "Ashe County",  # FIXED
    "Whiteville": "Columbus County", "Wilkesboro": "Wilkes County", "Wilson
County": "Wilson County",
    "Wingate": "Union County", "Woodfin": "Buncombe County", "Youngsville":
"Franklin County", "Pembroke": "Robeson County",
    "Elkin/Yadkin Valley": "Yadkin County", "Jackson County": "Jackson County",
"Yadkin Valley": "Yadkin County", "Taylorsville" : "Alexander County",
    "Alexander County" : "Alexander County"
}

#Creating a function that cleans out names to aid in processing
def clean_board_name(name):
    """
    Clean board names by:
    - Removing 'cid'
    - Removing parentheses and colons
    - Removing trailing numbers
    - Keeping only letters, spaces, and forward slashes
    - Consolidating multiple spaces
    """
    name = str(name)
    name = re.sub(r'cid', ' ', name)                 #Removing 'cid'
    name = re.sub(r'[\(\):]', ' ', name)             #Removing parentheses and
colons
    name = re.sub(r'\d+$', ' ', name)                #Removing trailing numbers
    name = re.sub(r'[^a-zA-Z\s/]', ' ', name)        #Keeping only letters,
spaces, and '/'
    name = ' '.join(name.split())                   #Collapsing multiple spaces
    return name.strip()

def consolidating_tables(name):
#Combining all tables first, WITHOUT filtering, to get a single table (it
breaks it up by page)
    df_list = []
    for t in name:
        df = t.df
```

```
        #print(df)
        #Removing completely empty rows
        df = df[df[0].str.strip() != ""]
        df_list.append(df)

    super_table = pd.concat(df_list, ignore_index=True)
    return super_table

def cleaning_tables (name):
    #Cleaning the board names (remove extra whitespace, numbers at end) using my
function
    name['Original_Name'] = name[0]  #Keeping original for debugging
    name['Cleaned_Name'] = name[0].apply(clean_board_name)

    #Filtering out header/footer rows and very short entries
    header_footer_keywords = ['ABC Board', 'Board Name', 'Audit', 'Total',
'Percent', 'Retail', 'Mixed', 'Totals', 'BoardName']
    name = name[
        ~name['Cleaned_Name'].isin(header_footer_keywords) &
        (name['Cleaned_Name'].str.len() > 3)
    ]

    #Mapping with the cleaned names to enable aggreagation
    name['County'] = name['Cleaned_Name'].map(municipality_to_county)
    return name

#CODE THAT WAS USED FOR DEBUGGING THE ORIGINAL PROGRAM
#Checking unmatched entries
#unmapped = super_table[super_table['County'].isna()]
#print(f"\nTotal rows after cleaning: {len(super_table)}")
#print(f"Successfully mapped: {super_table['County'].notna().sum()}")
#print(f"Unmapped boards: {len(unmapped)}")

#if len(unmapped) > 0:
#    print("\nBoards that failed to map:")
#    for orig, cleaned in zip(unmapped['Original_Name'].unique(),
unmapped['Cleaned_Name'].unique()):
#        print(f"  Original: '{orig}' -> Cleaned: '{cleaned}'")

#pd.set_option('display.max_rows', None)       #Showing all rows
#pd.set_option('display.max_columns', None)    #Showing all columns
#pd.set_option('display.width', None)          #No line wrapping
#pd.set_option('display.max_colwidth', None)   #Showing full column content


#Filtering to only successfully mapped rows
#super_table = super_table[super_table['County'].notna()]

#print(super_table)
```

```
#print(super_table['County'].nunique())

def numeric_conversion(name):
   #Converting numeric columns
   numeric_cols = list(range(1, len(name.columns) - 3))  # Exclude
Original_Name, Cleaned_Name, County

   for col in numeric_cols:
       name[col] = name[col].astype(str)
       name[col] = name[col].str.replace(",", "")
       name[col] = name[col].str.extract(r"([-+]?\d*\.?\d+)")[0]
       name[col] = pd.to_numeric(name[col], errors='coerce').fillna(0)

   #Aggregating by County
   name = name.groupby('County')[numeric_cols].sum().reset_index()
   return name

#print(f"\nFinal county count: {len(county_table)}")
#print(county_table['County'].unique())


def final_step(name):
   numeric_cols = [
       c for c in name.columns
       if c not in ['Original_Name', 'Cleaned_Name', 'County', 0]
   ]

   rename_map = {
       col: new_name
       for col, new_name in zip(numeric_cols, columns[1:])
   }

   name = name.rename(columns=rename_map)
   return name

def output(name, i):
   csv_file = "nc_counties_" + years[i] + ".csv"
   name.to_csv(csv_file, index=False)  # index=False avoids adding the row
numbers


#USED TO IDENTIFY MISSING ROWS
#county_table = county_table.rename(columns=rename_map)
#names = county_table['County'].str.replace(' County', '',
regex=False).tolist()
#names = set(names)
#other_names = set(all_counties)
#print(other_names - names)
```

```
#csv_file = "nc_counties_2023.csv"  #Specifying file for output
#county_table.to_csv(csv_file, index=False)  # index=False avoids adding the
row numbers

def enforce_all_counties(df, all_counties, year_label):
   full_list = [c + " County" for c in all_counties]

   present = set(df["County"])
   missing = set(full_list) - present

   if missing:
       print(f"\nWARNING ({year_label}): Missing counties being filled with
zeros:")
       print(sorted(missing))

   df = df.set_index("County")
   df = df.reindex(full_list, fill_value=0)
   df = df.reset_index()

   assert len(df) == 100, f"County count mismatch in {year_label}"

   return df

def main():
   for i in range(0, len(file_names)):
       file_path = "AlcoholSalesPDFs/" + file_names[i]

       tables = camelot.read_pdf(file_path, pages="all", flavor="stream")

       name = consolidating_tables(tables)
       name = cleaning_tables(name)
       name = numeric_conversion(name)

       #ENFORCE 100 COUNTIES BEFORE RENAMING
       name = enforce_all_counties(name, all_counties, years[i])

       name = final_step(name)
       output(name, i)

if __name__ == "__main__":
   main()
```

```
"""
Created on 2/25/26
@author: zevvanzanten
"""

import pandas as pd
```

```
# ---------------------------------------------------------------------
# Canonical NC FIPS Dictionary (EXACT copy you provided earlier)
# ---------------------------------------------------------------------

NC_FIPS_TO_COUNTY = {
   "37001": "Alamance County",
   "37003": "Alexander County",
   "37005": "Alleghany County",
   "37007": "Anson County",
   "37009": "Ashe County",
   "37011": "Avery County",
   "37013": "Beaufort County",
   "37015": "Bertie County",
   "37017": "Bladen County",
   "37019": "Brunswick County",
   "37021": "Buncombe County",
   "37023": "Burke County",
   "37025": "Cabarrus County",
   "37027": "Caldwell County",
   "37029": "Camden County",
   "37031": "Carteret County",
   "37033": "Caswell County",
   "37035": "Catawba County",
   "37037": "Chatham County",
   "37039": "Cherokee County",
   "37041": "Chowan County",
   "37043": "Clay County",
   "37045": "Cleveland County",
   "37047": "Columbus County",
   "37049": "Craven County",
   "37051": "Cumberland County",
   "37053": "Currituck County",
   "37055": "Dare County",
   "37057": "Davidson County",
   "37059": "Davie County",
   "37061": "Duplin County",
   "37063": "Durham County",
   "37065": "Edgecombe County",
   "37067": "Forsyth County",
   "37069": "Franklin County",
   "37071": "Gaston County",
   "37073": "Gates County",
   "37075": "Graham County",
   "37077": "Granville County",
   "37079": "Greene County",
   "37081": "Guilford County",
   "37083": "Halifax County",
   "37085": "Harnett County",
   "37087": "Haywood County",
```

```
  "37089": "Henderson County",
  "37091": "Hertford County",
  "37093": "Hoke County",
  "37095": "Hyde County",
  "37097": "Iredell County",
  "37099": "Jackson County",
  "37101": "Johnston County",
  "37103": "Jones County",
  "37105": "Lee County",
  "37107": "Lenoir County",
  "37109": "Lincoln County",
  "37111": "McDowell County",
  "37113": "Macon County",
  "37115": "Madison County",
  "37117": "Martin County",
  "37119": "Mecklenburg County",
  "37121": "Mitchell County",
  "37123": "Montgomery County",
  "37125": "Moore County",
  "37127": "Nash County",
  "37129": "New Hanover County",
  "37131": "Northampton County",
  "37133": "Onslow County",
  "37135": "Orange County",
  "37137": "Pamlico County",
  "37139": "Pasquotank County",
  "37141": "Pender County",
  "37143": "Perquimans County",
  "37145": "Person County",
  "37147": "Pitt County",
  "37149": "Polk County",
  "37151": "Randolph County",
  "37153": "Richmond County",
  "37155": "Robeson County",
  "37157": "Rockingham County",
  "37159": "Rowan County",
  "37161": "Rutherford County",
  "37163": "Sampson County",
  "37165": "Scotland County",
  "37167": "Stanly County",
  "37169": "Stokes County",
  "37171": "Surry County",
  "37173": "Swain County",
  "37175": "Transylvania County",
  "37177": "Tyrrell County",
  "37179": "Union County",
  "37181": "Vance County",
  "37183": "Wake County",
  "37185": "Warren County",
```

```
    "37187": "Washington County",
    "37189": "Watauga County",
    "37191": "Wayne County",
    "37193": "Wilkes County",
    "37195": "Wilson County",
    "37197": "Yadkin County",
    "37199": "Yancey County",
}

# ---------------------------------------------------------------------
# Panel Construction
# ---------------------------------------------------------------------

years = range(2012, 2025)
acs_base = "nc_counties_acs_age_brackets_and_race_{}.csv"

NC_FIPS = list(NC_FIPS_TO_COUNTY.keys())


def load_clean_acs(year):

    df = pd.read_csv(acs_base.format(year))

    # Standardize FIPS components
    df["state_fips"] = df["state_fips"].astype(str).str.zfill(2)
    df["county_fips"] = df["county_fips"].astype(str).str.zfill(3)
    df["fips"] = df["state_fips"] + df["county_fips"]

    # Keep only NC counties
    df = df[df["fips"].isin(NC_FIPS)].copy()

    # Identify missing counties
    present = set(df["fips"])
    missing = sorted(set(NC_FIPS) - present)

    if missing:
        print(f"{year}: Adding {len(missing)} missing counties")

        zero_rows = pd.DataFrame({"fips": missing})
        zero_rows["state_fips"] = zero_rows["fips"].str[:2]
        zero_rows["county_fips"] = zero_rows["fips"].str[2:]
        zero_rows["county"] = zero_rows["fips"].map(NC_FIPS_TO_COUNTY)

        for col in df.columns:
            if col not in ["fips", "state_fips", "county_fips", "county"]:
                zero_rows[col] = 0

        df = pd.concat([df, zero_rows], ignore_index=True)
```

```
    # Ensure consistent county naming
    df["county"] = df["fips"].map(NC_FIPS_TO_COUNTY)

    df = df.sort_values("fips").reset_index(drop=True)
    df["year"] = year

    return df


acs_all = []

for year in years:
    print(f"Processing ACS {year}")
    acs_all.append(load_clean_acs(year))

acs_panel = pd.concat(acs_all, ignore_index=True)

acs_panel.to_csv("nc_acs_panel_2012_2024.csv", index=False)

print("ACS panel written.")
print("Final shape:", acs_panel.shape)
```

```
"""
Created on 2/25/26
@author: zevvanzanten
"""
import pandas as pd

nc_counties = [
    "Alamance County",
    "Alexander County",
    "Alleghany County",
    "Anson County",
    "Ashe County",
    "Avery County",
    "Beaufort County",
    "Bertie County",
    "Bladen County",
    "Brunswick County",
    "Buncombe County",
    "Burke County",
    "Cabarrus County",
    "Caldwell County",
    "Camden County",
    "Carteret County",
    "Caswell County",
    "Catawba County",
    "Chatham County",
    "Cherokee County",
```

```
  "Chowan County",
  "Clay County",
  "Cleveland County",
  "Columbus County",
  "Craven County",
  "Cumberland County",
  "Currituck County",
  "Dare County",
  "Davidson County",
  "Davie County",
  "Duplin County",
  "Durham County",
  "Edgecombe County",
  "Forsyth County",
  "Franklin County",
  "Gaston County",
  "Gates County",
  "Graham County",
  "Granville County",
  "Greene County",
  "Guilford County",
  "Halifax County",
  "Harnett County",
  "Haywood County",
  "Henderson County",
  "Hertford County",
  "Hoke County",
  "Hyde County",
  "Iredell County",
  "Jackson County",
  "Johnston County",
  "Jones County",
  "Lee County",
  "Lenoir County",
  "Lincoln County",
  "McDowell County",
  "Macon County",
  "Madison County",
  "Martin County",
  "Mecklenburg County",
  "Mitchell County",
  "Montgomery County",
  "Moore County",
  "Nash County",
  "New Hanover County",
  "Northampton County",
  "Onslow County",
  "Orange County",
  "Pamlico County",
```

```
    "Pasquotank County",
    "Pender County",
    "Perquimans County",
    "Person County",
    "Pitt County",
    "Polk County",
    "Randolph County",
    "Richmond County",
    "Robeson County",
    "Rockingham County",
    "Rowan County",
    "Rutherford County",
    "Sampson County",
    "Scotland County",
    "Stanly County",
    "Stokes County",
    "Surry County",
    "Swain County",
    "Transylvania County",
    "Tyrrell County",
    "Union County",
    "Vance County",
    "Wake County",
    "Warren County",
    "Washington County",
    "Watauga County",
    "Wayne County",
    "Wilkes County",
    "Wilson County",
    "Yadkin County",
    "Yancey County"
]

retained_columns = ["County", "Retail Sales", "Beverage Sales", "Wine Sales",
"Gross Sales", "Bottles Regular Sold", "Bottles Mix Bev Sold", "Bottles Mini
Sold", "# of Stores"]

def load_clean_alcohol(year):

    df = pd.read_csv(f"nc_counties_{year}.csv")

    # Keep only relevant columns if they exist
    df = df[[c for c in retained_columns if c in df.columns]].copy()

    # Ensure county names match canonical list exactly
    df["County"] = df["County"].astype(str).str.strip()

    # Drop any rows not in the official 100-county list
    df = df[df["County"].isin(nc_counties)].copy()
```

```
    # Convert numeric columns safely
    numeric_cols = [c for c in df.columns if c != "County"]

    for col in numeric_cols:
        df[col] = (
            df[col]
            .astype(str)
            .str.replace(",", "", regex=False)
            .str.replace("%", "", regex=False)
        )
        df[col] = pd.to_numeric(df[col], errors="coerce")

    df["year"] = year

    # --- Identify missing counties ---
    present = set(df["County"])
    expected = set(nc_counties)

    missing = sorted(expected - present)

    if missing:
        print(f"{year}: Adding {len(missing)} missing counties")

        # Create zero rows for missing counties
        zero_rows = pd.DataFrame({
            "County": missing
        })

        # Add all other columns with value 0
        for col in df.columns:
            if col != "County":
                zero_rows[col] = 0

        # Append and reset
        df = pd.concat([df, zero_rows], ignore_index=True)

    # Ensure consistent ordering
    df = df.sort_values("County").reset_index(drop=True)

    return df

years = range(2006, 2025)

alcohol_all = []

for year in years:
    if year == 2010:
        continue
```

```
    alcohol_all.append(load_clean_alcohol(year))

alcohol_panel = pd.concat(alcohol_all, ignore_index=True)

alcohol_panel.to_csv("nc_alcohol_panel_2012_2024.csv", index=False)
print("Alcohol panel written.")
```

```
import pandas as pd
import glob
import re

def load_core_tourism_panel(path_pattern="nc_counties_tourism_*.csv",
                            start_year=2012):

    files = glob.glob(path_pattern)
    dfs = []

    for file in files:

        # Extract year from filename
        year = int(re.search(r"(\d{4})", file).group(1))

        # ---- Skip earlier years ----
        if year < start_year:
            continue

        df = pd.read_csv(file)

        # ---- Identify Expenditures Column ----
        expend_cols = [col for col in df.columns if "Expend" in col]

        if len(expend_cols) == 0:
            raise ValueError(f"No expenditure column found in {file}")

        expend_col = expend_cols[0]  # take first match

        # ---- Keep Only Needed Columns ----
        df = df[["County", expend_col]].copy()

        # ---- Standardize Column Name ----
        df = df.rename(columns={expend_col: "Expenditures_Millions"})

        # ---- Clean Numeric ----
        df["Expenditures_Millions"] = (
            df["Expenditures_Millions"]
            .astype(str)
            .str.replace(",", "", regex=False)
            .str.replace("$", "", regex=False)
            .str.strip()
```

```
        )

        df["Expenditures_Millions"] = pd.to_numeric(
            df["Expenditures_Millions"],
            errors="coerce"
        )

        df["Year"] = year

        dfs.append(df)

    panel = pd.concat(dfs, ignore_index=True)

    panel = panel.sort_values(["County", "Year"]).reset_index(drop=True)

    return panel


if __name__ == "__main__":

    panel = load_core_tourism_panel(start_year=2012)

    print(panel.groupby("Year")["County"].nunique())

    panel.to_csv("nc_tourism_expenditures_panel_2012_onward.csv", index=False)
```

```
"""
Created on 2/25/26
Unemployment Panel Builder (LAUS, FIPS-based)
@author: zevvanzanten
"""

import pandas as pd

years = range(2012, 2025)
unemployment_base = "LaborData/laucnty{:02d}.xlsx"

NC_FIPS_TO_COUNTY = {
    "37001": "Alamance County",
    "37003": "Alexander County",
    "37005": "Alleghany County",
    "37007": "Anson County",
    "37009": "Ashe County",
    "37011": "Avery County",
    "37013": "Beaufort County",
    "37015": "Bertie County",
    "37017": "Bladen County",
    "37019": "Brunswick County",
    "37021": "Buncombe County",
```

```
  "37023": "Burke County",
  "37025": "Cabarrus County",
  "37027": "Caldwell County",
  "37029": "Camden County",
  "37031": "Carteret County",
  "37033": "Caswell County",
  "37035": "Catawba County",
  "37037": "Chatham County",
  "37039": "Cherokee County",
  "37041": "Chowan County",
  "37043": "Clay County",
  "37045": "Cleveland County",
  "37047": "Columbus County",
  "37049": "Craven County",
  "37051": "Cumberland County",
  "37053": "Currituck County",
  "37055": "Dare County",
  "37057": "Davidson County",
  "37059": "Davie County",
  "37061": "Duplin County",
  "37063": "Durham County",
  "37065": "Edgecombe County",
  "37067": "Forsyth County",
  "37069": "Franklin County",
  "37071": "Gaston County",
  "37073": "Gates County",
  "37075": "Graham County",
  "37077": "Granville County",
  "37079": "Greene County",
  "37081": "Guilford County",
  "37083": "Halifax County",
  "37085": "Harnett County",
  "37087": "Haywood County",
  "37089": "Henderson County",
  "37091": "Hertford County",
  "37093": "Hoke County",
  "37095": "Hyde County",
  "37097": "Iredell County",
  "37099": "Jackson County",
  "37101": "Johnston County",
  "37103": "Jones County",
  "37105": "Lee County",
  "37107": "Lenoir County",
  "37109": "Lincoln County",
  "37111": "McDowell County",
  "37113": "Macon County",
  "37115": "Madison County",
  "37117": "Martin County",
  "37119": "Mecklenburg County",
```

```
    "37121": "Mitchell County",
    "37123": "Montgomery County",
    "37125": "Moore County",
    "37127": "Nash County",
    "37129": "New Hanover County",
    "37131": "Northampton County",
    "37133": "Onslow County",
    "37135": "Orange County",
    "37137": "Pamlico County",
    "37139": "Pasquotank County",
    "37141": "Pender County",
    "37143": "Perquimans County",
    "37145": "Person County",
    "37147": "Pitt County",
    "37149": "Polk County",
    "37151": "Randolph County",
    "37153": "Richmond County",
    "37155": "Robeson County",
    "37157": "Rockingham County",
    "37159": "Rowan County",
    "37161": "Rutherford County",
    "37163": "Sampson County",
    "37165": "Scotland County",
    "37167": "Stanly County",
    "37169": "Stokes County",
    "37171": "Surry County",
    "37173": "Swain County",
    "37175": "Transylvania County",
    "37177": "Tyrrell County",
    "37179": "Union County",
    "37181": "Vance County",
    "37183": "Wake County",
    "37185": "Warren County",
    "37187": "Washington County",
    "37189": "Watauga County",
    "37191": "Wayne County",
    "37193": "Wilkes County",
    "37195": "Wilson County",
    "37197": "Yadkin County",
    "37199": "Yancey County",
}

def load_clean_unemployment(year):

    # Load exactly like your original script
    unemp = pd.read_excel(unemployment_base.format(year % 100), header=1)

    # OG filter: North Carolina only
    unemp = unemp[unemp["State FIPS Code"] == 37].copy()
```

```
    # Keep only the two variables you want
    unemp = unemp[
        ["County Name/State Abbreviation", "Unemployment Rate (%)"]
    ].copy()

    # Split county name at comma
    unemp["county"] = (
        unemp["County Name/State Abbreviation"]
        .str.split(",", n=1)
        .str[0]
        .str.replace(" County", "", regex=False)
        .str.strip()
    )

    # Convert unemployment rate to numeric
    unemp["unemployment_rate"] = pd.to_numeric(
        unemp["Unemployment Rate (%)"],
        errors="coerce"
    )

    # Keep clean columns only
    unemp = unemp[["county", "unemployment_rate"]]

    unemp["year"] = year

    return unemp


# Stack all years
unemp_all = []

for year in years:
    print(f"Processing unemployment {year}")
    unemp_all.append(load_clean_unemployment(year))

unemp_panel = pd.concat(unemp_all, ignore_index=True)

unemp_panel.to_csv("nc_unemployment_panel_2012_2024.csv", index=False)

print("Unemployment panel written.")
print("Final shape:", unemp_panel.shape)
```

```
"""
Created on 2/25/26
Unemployment Panel Builder (LAUS, FIPS-based)
@author: zevvanzanten
"""
```

```python
import pandas as pd

years = range(2006, 2025)
unemployment_base = "laucnty{:02d}.xlsx"

NC_FIPS_TO_COUNTY = {
    "37001": "Alamance County",
    "37003": "Alexander County",
    "37005": "Alleghany County",
    "37007": "Anson County",
    "37009": "Ashe County",
    "37011": "Avery County",
    "37013": "Beaufort County",
    "37015": "Bertie County",
    "37017": "Bladen County",
    "37019": "Brunswick County",
    "37021": "Buncombe County",
    "37023": "Burke County",
    "37025": "Cabarrus County",
    "37027": "Caldwell County",
    "37029": "Camden County",
    "37031": "Carteret County",
    "37033": "Caswell County",
    "37035": "Catawba County",
    "37037": "Chatham County",
    "37039": "Cherokee County",
    "37041": "Chowan County",
    "37043": "Clay County",
    "37045": "Cleveland County",
    "37047": "Columbus County",
    "37049": "Craven County",
    "37051": "Cumberland County",
    "37053": "Currituck County",
    "37055": "Dare County",
    "37057": "Davidson County",
    "37059": "Davie County",
    "37061": "Duplin County",
    "37063": "Durham County",
    "37065": "Edgecombe County",
    "37067": "Forsyth County",
    "37069": "Franklin County",
    "37071": "Gaston County",
    "37073": "Gates County",
    "37075": "Graham County",
    "37077": "Granville County",
    "37079": "Greene County",
    "37081": "Guilford County",
    "37083": "Halifax County",
    "37085": "Harnett County",
```

```
  "37087": "Haywood County",
  "37089": "Henderson County",
  "37091": "Hertford County",
  "37093": "Hoke County",
  "37095": "Hyde County",
  "37097": "Iredell County",
  "37099": "Jackson County",
  "37101": "Johnston County",
  "37103": "Jones County",
  "37105": "Lee County",
  "37107": "Lenoir County",
  "37109": "Lincoln County",
  "37111": "McDowell County",
  "37113": "Macon County",
  "37115": "Madison County",
  "37117": "Martin County",
  "37119": "Mecklenburg County",
  "37121": "Mitchell County",
  "37123": "Montgomery County",
  "37125": "Moore County",
  "37127": "Nash County",
  "37129": "New Hanover County",
  "37131": "Northampton County",
  "37133": "Onslow County",
  "37135": "Orange County",
  "37137": "Pamlico County",
  "37139": "Pasquotank County",
  "37141": "Pender County",
  "37143": "Perquimans County",
  "37145": "Person County",
  "37147": "Pitt County",
  "37149": "Polk County",
  "37151": "Randolph County",
  "37153": "Richmond County",
  "37155": "Robeson County",
  "37157": "Rockingham County",
  "37159": "Rowan County",
  "37161": "Rutherford County",
  "37163": "Sampson County",
  "37165": "Scotland County",
  "37167": "Stanly County",
  "37169": "Stokes County",
  "37171": "Surry County",
  "37173": "Swain County",
  "37175": "Transylvania County",
  "37177": "Tyrrell County",
  "37179": "Union County",
  "37181": "Vance County",
  "37183": "Wake County",
```

```
    "37185": "Warren County",
    "37187": "Washington County",
    "37189": "Watauga County",
    "37191": "Wayne County",
    "37193": "Wilkes County",
    "37195": "Wilson County",
    "37197": "Yadkin County",
    "37199": "Yancey County",
}

def load_clean_unemployment(year):

    # Load exactly like your original script
    unemp = pd.read_excel(unemployment_base.format(year % 100), header=1)

    # OG filter: North Carolina only
    unemp = unemp[unemp["State FIPS Code"] == 37].copy()

    # Keep only the two variables you want
    unemp = unemp[
        ["County Name/State Abbreviation", "Unemployment Rate (%)"]
    ].copy()

    # Split county name at comma
    unemp["county"] = (
        unemp["County Name/State Abbreviation"]
        .str.split(",", n=1)
        .str[0]
        .str.replace(" County", "", regex=False)
        .str.strip()
    )

    # Convert unemployment rate to numeric
    unemp["unemployment_rate"] = pd.to_numeric(
        unemp["Unemployment Rate (%)"],
        errors="coerce"
    )

    # Keep clean columns only
    unemp = unemp[["county", "unemployment_rate"]]

    unemp["year"] = year

    return unemp


# Stack all years
unemp_all = []
```

```
for year in years:
    print(f"Processing unemployment {year}")
    unemp_all.append(load_clean_unemployment(year))

unemp_panel = pd.concat(unemp_all, ignore_index=True)

unemp_panel.to_csv("nc_unemployment_panel_2006_2024.csv", index=False)

print("Unemployment panel written.")
print("Final shape:", unemp_panel.shape)
```

```
import pandas as pd

years = range(2006, 2025)
pop_base = "{}Pop.csv"

def load_clean_population(file_path):

    df = pd.read_csv(file_path, sep=";")

    # Fix BOM issue
    df.columns = df.columns.str.replace("\ufeff", "")

    # Keep only counties
    df = df[df["Area Type"] == "County"].copy()

    # Extract year from date column
    df["year"] = pd.to_datetime(df["Year"]).dt.year

    # Clean variables
    df["county"] = df["Area"].str.strip()
    df["population"] = pd.to_numeric(df["value"], errors="coerce")

    return df[["county", "population", "year"]]


# Build panel
pop_all = []

for year in years:
    file_path = pop_base.format(year)
    print(f"Processing {file_path}")
    pop_all.append(load_clean_population(file_path))

pop_panel = pd.concat(pop_all, ignore_index=True)

# Optional: sort
pop_panel = pop_panel.sort_values(["county", "year"])
```

```
# Save
pop_panel.to_csv("nc_population_panel_2006_2024.csv", index=False)

print("Population panel written.")
print("Final shape:", pop_panel.shape)
```

```
import camelot
import pandas as pd

file_names = [
   "RawTourismData/2006-County-Level-Visitor-Expenditures_0.pdf",
   "RawTourismData/2007-County-Level-Visitor-Expenditures_0.pdf",
   "RawTourismData/2008-County-Level-Visitor-Expenditures_0.pdf",
   "RawTourismData/2009-County-Level-Visitor-Expenditures_0.pdf"
]

file_names_one = [
   "RawTourismData/2010-County-Level-Visitor-Expenditures_0.pdf",
   "RawTourismData/2011-County-Level-Visitor-Expenditures_0.pdf",
   "RawTourismData/2012-County-Level-Visitor-Expenditures_0.pdf",
   "RawTourismData/2013-County-Level-Visitor-Expenditures_0.pdf",
   "RawTourismData/2014-County-Level-Visitor-Expenditures_0.pdf"
]

file_names_two = [
   "RawTourismData/2015-County-Level-Visitor-Expenditures_0.pdf",
   "RawTourismData/2016-Revised-County-Level-Visitor-Expenditures_0.pdf",
   "RawTourismData/2017-County-Level-Visitor-Expenditures_0.pdf",
   "RawTourismData/2018+NC+County+PRELIMINARY+Estimates+ALPHA.pdf",
   "RawTourismData/2019-NC-County-PRELIMINARY-Estimates-ALPHA-REVISED.pdf"
]

file_names_three = [
   "RawTourismData/2020-County.pdf",
   "RawTourismData/2021-County-Level-Visitor-Expenditures.pdf",
   "RawTourismData/2022+County+Level+Visitor+Expenditures.pdf"
]

file_names_four = [
   "RawTourismData/2023+County+Level+Visitor+Expenditures.pdf",
   "RawTourismData/7UBJY7-2024 County Level Visitor Expenditures.pdf"
]

years = ["2006", "2007", "2008", "2009"]
years_one = ["2010", "2011", "2012", "2013", "2014"]
years_two = ["2015", "2016", "2017", "2018", "2019"]
years_three = ["2020", "2021", "2022"]
years_four = ["2023", "2024"]
```

```
columns = [
   "County",
   "Expenditures_(Millions)",
   "Expenditures_Percent_Change",
   "Payroll_(Millions)",
   "Employment_(Thousands)",
   "State_Tax_Receipts_(Millions)",
   "Local_Tax_Receipts_(Millions)"
]


columns_one = [
   "County",
   "Expenditures_(Millions)",
   "Expenditures_Percent_Change",
   "Payroll_(Millions)",
   "Employment_(Thousands)",
   "State_Tax_Receipts_(Millions)",
   "Local_Tax_Receipts_(Millions)"
]

columns_two = [
   "County",
   "Expenditures_(Millions)",
   "Expenditures_Percent_Change",
   "Payroll_(Millions)",
   "Payroll_Percent_Change",
   "Employment_(Thousands)",
   "Employment_Percent_Change",
   "State_Tax_Receipts_(Millions)",
   "State_Tax_Receipts_Percent_Change",
   "Local_Tax_Receipts_(Millions)",
   "Local_Tax_Receipts_Percent_Change",
]

columns_three = [
   "County",
   "Lodging",
   "F&B",
   "Recreation",
   "Retail",
   "Transport",
   "Expenditures_(Millions)",
   "Share of State",
   "Spending_Growth_Rate",
   "Payroll",
   "Share of State",
   "Labor Income",
   "State Taxes",
```

```
    "Local Taxes",
    "State/Local Tax Savings per Resident"
]

columns_four = [
    "County",
    "Lodging",
    "F&B",
    "Recreation",
    "Retail",
    "Transport",
    "Expenditures_(Millions)",
    "Share of State",
    "Spending_Growth_Rate",
    "Payroll",
    "Share of State",
    "Labor Income",
    "State Taxes",
    "Local Taxes",
    "State/Local Tax Savings per Resident",
    "NaN"
]

county_mapping = {
    "ALAMANCE": "Alamance",
    "ALEXANDER": "Alexander",
    "ALLEGHANY": "Alleghany",
    "ANSON": "Anson",
    "ASHE": "Ashe",
    "AVERY": "Avery",
    "BEAUFORT": "Beaufort",
    "BERTIE": "Bertie",
    "BLADEN": "Bladen",
    "BRUNSWICK": "Brunswick",
    "BUNCOMBE": "Buncombe",
    "BURKE": "Burke",
    "CABARRUS": "Cabarrus",
    "CALDWELL": "Caldwell",
    "CAMDEN": "Camden",
    "CARTERET": "Carteret",
    "CASWELL": "Caswell",
    "CATAWBA": "Catawba",
    "CHATHAM": "Chatham",
    "CHEROKEE": "Cherokee",
    "CHOWAN": "Chowan",
    "CLAY": "Clay",
    "CLEVELAND": "Cleveland",
    "COLUMBUS": "Columbus",
    "CRAVEN": "Craven",
```

```
  "CUMBERLAND": "Cumberland",
  "CURRITUCK": "Currituck",
  "DARE": "Dare",
  "DAVIDSON": "Davidson",
  "DAVIE": "Davie",
  "DUPLIN": "Duplin",
  "DURHAM": "Durham",
  "EDGECOMBE": "Edgecombe",
  "FORSYTH": "Forsyth",
  "FRANKLIN": "Franklin",
  "GASTON": "Gaston",
  "GATES": "Gates",
  "GRAHAM": "Graham",
  "GRANVILLE": "Granville",
  "GREENE": "Greene",
  "GUILFORD": "Guilford",
  "HALIFAX": "Halifax",
  "HARNETT": "Harnett",
  "HAYWOOD": "Haywood",
  "HENDERSON": "Henderson",
  "HERTFORD": "Hertford",
  "HOKE": "Hoke",
  "HYDE": "Hyde",
  "IREDELL": "Iredell",
  "JACKSON": "Jackson",
  "JOHNSTON": "Johnston",
  "JONES": "Jones",
  "LEE": "Lee",
  "LENOIR": "Lenoir",
  "LINCOLN": "Lincoln",
  "MACON": "Macon",
  "MADISON": "Madison",
  "MARTIN": "Martin",
  "MCDOWELL": "McDowell",
  "MECKLENBURG": "Mecklenburg",
  "MITCHELL": "Mitchell",
  "MONTGOMERY": "Montgomery",
  "MOORE": "Moore",
  "NASH": "Nash",
  "NEW HANOVER": "New Hanover",
  "NORTHAMPTON": "Northampton",
  "ONSLOW": "Onslow",
  "ORANGE": "Orange",
  "PAMLICO": "Pamlico",
  "PASQUOTANK": "Pasquotank",
  "PENDER": "Pender",
  "PERQUIMANS": "Perquimans",
  "PERSON": "Person",
  "PITT": "Pitt",
```

```
    "POLK": "Polk",
    "RANDOLPH": "Randolph",
    "RICHMOND": "Richmond",
    "ROBESON": "Robeson",
    "ROCKINGHAM": "Rockingham",
    "ROWAN": "Rowan",
    "RUTHERFORD": "Rutherford",
    "SAMPSON": "Sampson",
    "SCOTLAND": "Scotland",
    "STANLY": "Stanly",
    "STOKES": "Stokes",
    "SURRY": "Surry",
    "SWAIN": "Swain",
    "TRANSYLVANIA": "Transylvania",
    "TYRRELL": "Tyrrell",
    "UNION": "Union",
    "VANCE": "Vance",
    "WAKE": "Wake",
    "WARREN": "Warren",
    "WASHINGTON": "Washington",
    "WATAUGA": "Watauga",
    "WAYNE": "Wayne",
    "WILKES": "Wilkes",
    "WILSON": "Wilson",
    "YADKIN": "Yadkin",
    "YANCEY": "Yancey",
}

valid_county_names = [
    "Alamance", "Alexander", "Alleghany", "Anson", "Ashe", "Avery",
    "Beaufort", "Bertie", "Bladen", "Brunswick", "Buncombe", "Burke",
    "Cabarrus", "Caldwell", "Camden", "Carteret", "Caswell", "Catawba",
    "Chatham", "Cherokee", "Chowan", "Clay", "Cleveland", "Columbus",
    "Craven", "Cumberland", "Currituck", "Dare", "Davidson", "Davie",
    "Duplin", "Durham", "Edgecombe", "Forsyth", "Franklin", "Gaston",
    "Gates", "Graham", "Granville", "Greene", "Guilford", "Halifax",
    "Harnett", "Haywood", "Henderson", "Hertford", "Hoke", "Hyde",
    "Iredell", "Jackson", "Johnston", "Jones", "Lee", "Lenoir",
    "Lincoln", "Macon", "Madison", "Martin", "McDowell", "Mecklenburg",
    "Mitchell", "Montgomery", "Moore", "Nash", "New Hanover",
    "Northampton", "Onslow", "Orange", "Pamlico", "Pasquotank",
    "Pender", "Perquimans", "Person", "Pitt", "Polk", "Randolph",
    "Richmond", "Robeson", "Rockingham", "Rowan", "Rutherford",
    "Sampson", "Scotland", "Stanly", "Stokes", "Surry", "Swain",
    "Transylvania", "Tyrrell", "Union", "Vance", "Wake", "Warren",
    "Washington", "Watauga", "Wayne", "Wilkes", "Wilson",
    "Yadkin", "Yancey"
]
```

```python
def process_and_save_csv(file_list, year_list, column_list):

   for i in range(len(file_list)):
       filename = file_list[i]
       year = year_list[i]

       print(f"\nProcessing {year}...")

       tables = camelot.read_pdf(filename, pages="all", flavor="stream")
       df_list = []

       for table in tables:
           df = table.df
           df = df[df[0].str.strip() != ""]
           df_list.append(df)

       final_df = pd.concat(df_list, ignore_index=True)

       # Handle column mismatch
       if final_df.shape[1] == len(column_list):
           final_df.columns = column_list
       elif final_df.shape[1] == len(column_list) - 1:
           final_df.columns = column_list[:-1]
       else:
           print(f"⚠ Skipping {year} due to unexpected column count:
{final_df.shape[1]}")
           continue

       final_df = final_df.iloc[1:].reset_index(drop=True)

       # Clean + map counties
       final_df["County"] = (
           final_df["County"]
           .str.upper()
           .str.strip()
           .map(county_mapping)
       )

       final_df = final_df[
           final_df["County"].isin(valid_county_names)
       ].reset_index(drop=True)

       # -------- SAVE CSV --------
       output_path = f"nc_counties_tourism_{year}.csv"
       final_df.to_csv(output_path, index=False)

       print(f"Saved: {output_path}")
       print(f"Rows: {len(final_df)} | Columns: {len(final_df.columns)}")
```

```
if __name__ == "__main__":
    process_and_save_csv(file_names, years, columns)
    #process_and_save_csv(file_names_one, years_one, columns_one)
    #process_and_save_csv(file_names_two, years_two, columns_two)
    #process_and_save_csv(file_names_three, years_three, columns_three)
    #process_and_save_csv(file_names_four, years_four, columns_four)
```

```
import pandas as pd

years = [y for y in range(2006, 2025) if y != 2010]
dfs = []

for year in years:
    file = f"ProcessedTourismData/nc_counties_tourism_{year}.csv"

    df = pd.read_csv(file)

    # find expenditures column (assume exactly one match)
    exp_col = [col for col in df.columns if "Expenditure" in col][0]

    df = df[["County", exp_col]].rename(
        columns={exp_col: "Expenditures"}
    )

    df["year"] = year

    dfs.append(df)

final_df = pd.concat(dfs, ignore_index=True)

final_df.to_csv("nc_tourism_panel_2006_2024.csv", index=False)
```

```
"""
Tourism Processing 2021-2024
Your original structure + built-in diagnostics
"""

import camelot
import pandas as pd
import re


# ------------------------------------------------
# FILES (2021-2024)
# ------------------------------------------------

file_names = [
    "2021-County-Level-Visitor-Expenditures.pdf",
    "2022+County+Level+Visitor+Expenditures.pdf",
```

```
    "2023+County+Level+Visitor+Expenditures.pdf",
    "7UBJY7-2024 County Level Visitor Expenditures.pdf"
]

years = ["2021", "2022", "2023", "2024"]


columns_four = [
    "County",
    "Lodging",
    "F&B",
    "Recreation",
    "Retail",
    "Transport",
    "Expenditures_(Millions)",
    "Share of State",
    "Spending_Growth_Rate",
    "Payroll",
    "Share of State_2",
    "Labor Income",
    "State Taxes",
    "Local Taxes",
    "State/Local Tax Savings per Resident",
    "NaN"
]


# ---------------------------------------------------
# COUNTY MAPPING (FULL)
# ---------------------------------------------------

county_mapping = {
    "ALAMANCE": "Alamance",
    "ALEXANDER": "Alexander",
    "ALLEGHANY": "Alleghany",
    "ANSON": "Anson",
    "ASHE": "Ashe",
    "AVERY": "Avery",
    "BEAUFORT": "Beaufort",
    "BERTIE": "Bertie",
    "BLADEN": "Bladen",
    "BRUNSWICK": "Brunswick",
    "BUNCOMBE": "Buncombe",
    "BURKE": "Burke",
    "CABARRUS": "Cabarrus",
    "CALDWELL": "Caldwell",
    "CAMDEN": "Camden",
    "CARTERET": "Carteret",
    "CASWELL": "Caswell",
```

```
  "CATAWBA": "Catawba",
  "CHATHAM": "Chatham",
  "CHEROKEE": "Cherokee",
  "CHOWAN": "Chowan",
  "CLAY": "Clay",
  "CLEVELAND": "Cleveland",
  "COLUMBUS": "Columbus",
  "CRAVEN": "Craven",
  "CUMBERLAND": "Cumberland",
  "CURRITUCK": "Currituck",
  "DARE": "Dare",
  "DAVIDSON": "Davidson",
  "DAVIE": "Davie",
  "DUPLIN": "Duplin",
  "DURHAM": "Durham",
  "EDGECOMBE": "Edgecombe",
  "FORSYTH": "Forsyth",
  "FRANKLIN": "Franklin",
  "GASTON": "Gaston",
  "GATES": "Gates",
  "GRAHAM": "Graham",
  "GRANVILLE": "Granville",
  "GREENE": "Greene",
  "GUILFORD": "Guilford",
  "HALIFAX": "Halifax",
  "HARNETT": "Harnett",
  "HAYWOOD": "Haywood",
  "HENDERSON": "Henderson",
  "HERTFORD": "Hertford",
  "HOKE": "Hoke",
  "HYDE": "Hyde",
  "IREDELL": "Iredell",
  "JACKSON": "Jackson",
  "JOHNSTON": "Johnston",
  "JONES": "Jones",
  "LEE": "Lee",
  "LENOIR": "Lenoir",
  "LINCOLN": "Lincoln",
  "MACON": "Macon",
  "MADISON": "Madison",
  "MARTIN": "Martin",
  "MCDOWELL": "McDowell",
  "MECKLENBURG": "Mecklenburg",
  "MITCHELL": "Mitchell",
  "MONTGOMERY": "Montgomery",
  "MOORE": "Moore",
  "NASH": "Nash",
  "NEW HANOVER": "New Hanover",
  "NORTHAMPTON": "Northampton",
```

```
    "ONSLOW": "Onslow",
    "ORANGE": "Orange",
    "PAMLICO": "Pamlico",
    "PASQUOTANK": "Pasquotank",
    "PENDER": "Pender",
    "PERQUIMANS": "Perquimans",
    "PERSON": "Person",
    "PITT": "Pitt",
    "POLK": "Polk",
    "RANDOLPH": "Randolph",
    "RICHMOND": "Richmond",
    "ROBESON": "Robeson",
    "ROCKINGHAM": "Rockingham",
    "ROWAN": "Rowan",
    "RUTHERFORD": "Rutherford",
    "SAMPSON": "Sampson",
    "SCOTLAND": "Scotland",
    "STANLY": "Stanly",
    "STOKES": "Stokes",
    "SURRY": "Surry",
    "SWAIN": "Swain",
    "TRANSYLVANIA": "Transylvania",
    "TYRRELL": "Tyrrell",
    "UNION": "Union",
    "VANCE": "Vance",
    "WAKE": "Wake",
    "WARREN": "Warren",
    "WASHINGTON": "Washington",
    "WATAUGA": "Watauga",
    "WAYNE": "Wayne",
    "WILKES": "Wilkes",
    "WILSON": "Wilson",
    "YADKIN": "Yadkin",
    "YANCEY": "Yancey",
}

valid_county_names = list(county_mapping.values())


# ------------------------------------------------
# MAIN
# ------------------------------------------------

def main():

    for file, year in zip(file_names, years):

        print(f"\nProcessing {year}...")
```

```
        tables = camelot.read_pdf(file, pages="all", flavor="stream")

        if len(tables) == 0:
            raise ValueError(f"{year}: No tables detected.")

        df_list = []

        for table in tables:
            df = table.df
            df = df[df[0].str.strip() != ""]
            df_list.append(df)

        final_df = pd.concat(df_list, ignore_index=True)

        print("Raw shape:", final_df.shape)

        # ---- YOUR ORIGINAL FLEXIBLE COLUMN LOGIC ----
        if final_df.shape[1] == len(columns_four):
            final_df.columns = columns_four
        elif final_df.shape[1] == len(columns_four) - 1:
            final_df.columns = columns_four[:-1]
        else:
            raise ValueError(f"{year}: Unexpected column count
{final_df.shape[1]}")

        final_df = final_df.iloc[1:].reset_index(drop=True)

        print("Rows after header drop:", len(final_df))

        # ---- County Normalization ----
        final_df["County"] = (
            final_df["County"]
            .str.upper()
            .str.replace(" COUNTY", "", regex=False)
            .str.strip()
        )

        mapped = final_df["County"].map(county_mapping)

        print("Mapped counties:", mapped.notna().sum())

        if mapped.notna().sum() < 95:
            print("Unmapped counties:")
            print(final_df.loc[mapped.isna(), "County"].unique())

        final_df["County"] = mapped
        final_df =
final_df[final_df["County"].isin(valid_county_names)].reset_index(drop=True)
```

```
        print("Final row count:", len(final_df))

        if len(final_df) != 100:
            raise ValueError(f"{year}: Expected 100 counties, got
{len(final_df)}")

        if final_df["County"].duplicated().any():
            raise ValueError(f"{year}: Duplicate counties detected.")

        for col in final_df.columns:
            if re.search(r"expenditure", col, re.IGNORECASE):
                final_df.rename(columns={col: "Expenditures_(Millions)"},
inplace=True)
                break

        csv_file = f"nc_counties_tourism_{year}.csv"
        final_df.to_csv(csv_file, index=False)

        print(f"{year} PASSED and saved.")


if __name__ == "__main__":
    main()
```

```
"""
Created on 3/17/26
@author: zevvanzanten
"""
import pandas as pd

#This file is mistakenly named and contains 2006 onwards, thus I filter it to
get the right dimensions
sales = pd.read_csv("nc_alcohol_panel_2012_2024.csv")
sales = sales[sales['year'] > 2011]

#County name here is written without the "County" appended onto the end.
#I fix this to enable a future merge.
labor = pd.read_csv("nc_unemployment_panel_2012_2024.csv")
labor['county'] = labor['county'] + " County"

#County name here is written without the "County" appended onto the end.
#I fix this to enable a future merge.
tourism = pd.read_csv("nc_tourism_expenditures_panel_2012_onward.csv")
tourism['County'] = tourism['County'] + " County"

controls = pd.read_csv("nc_acs_panel_2012_2024.csv")

#First merge - Sales + ACS Controls
```

```
merged = pd.merge(sales, controls, right_on = ["county", "year"], left_on =
["County", "year"], how="left")

#Second merge - existing stuff + unemployment data
merged = pd.merge(merged, labor, right_on = ["county", "year"], left_on =
["county", "year"], how="left")

#Third merge - existing stuff + tourism data
merged = pd.merge(merged, tourism, right_on = ["County", "Year"], left_on =
["county", "year"], how="left")

print(merged.columns)

#Creating a list of desired columns to make cleaning merged data easier
columns = [
   "County_x",
   "Retail Sales",
   "Beverage Sales",
   "Wine Sales",
   "Gross Sales",
   "Bottles Regular Sold",
   "Bottles Mix Bev Sold",
   "Bottles Mini Sold",
   "# of Stores",
   "population",
   "pct_male_0_17",
   "pct_female_0_17",
   "pct_male_18_24",
   "pct_female_18_24",
   "pct_male_25_44",
   "pct_female_25_44",
   "pct_male_45_64",
   "pct_female_45_64",
   "pct_male_65_plus",
   "pct_female_65_plus",
   "pct_male",
   "pct_bachelor_plus",
   "military_per_100k",
   "pct_white",
   "pct_black",
   "pct_american_indian_alaska_native",
   "pct_asian",
   "pct_native_pacific",
   "pct_some_other_race",
   "pct_two_or_more_races",
   "pct_hispanic_or_latino",
   "county",
   "unemployment_rate_y",
   "Expenditures_Millions",
```

```
    "Year"
]

#Filtering and renaming
merged = merged[columns]
merged = merged.rename(columns = {
    "County_x": "County",
    "unemployment_rate_y" : "unemployment_rate"
})

merged.columns = (
    merged.columns
    .str.lower()
    .str.strip()
    .str.replace(" ", "_")
)

#Inflation adjustment
cpi = pd.read_csv("CPI.csv")

#Converting data to datetime
cpi["observation_date"] = pd.to_datetime(cpi["observation_date"])

#Extracting year
cpi["year"] = cpi["observation_date"].dt.year

#Annualizing (i.e., taking average within each year)
cpi_annual = (
    cpi.groupby("year")["CPIAUCSL_20260311"]
    .mean()
    .reset_index()
)

cpi_annual = cpi_annual[
    (cpi_annual["year"] >= 2012) &
    (cpi_annual["year"] <= 2024)
]

cpi_annual = cpi_annual.rename(columns = {
    "CPIAUCSL_20260311" : "CPI"
})

#Merging on CPI to account for inflation, then adjusting for inflation
merged = pd.merge(merged, cpi_annual, on = "year")

base_cpi = cpi_annual.loc[
    cpi_annual["year"] == 2024, "CPI"
].values[0]
```

```
inflation_adjustable = [
   "retail_sales",
   "beverage_sales",
   "wine_sales",
   "gross_sales",
   "expenditures_millions"
   ]

for col in inflation_adjustable:
   merged[f"real_{col}"] = merged[col] * (base_cpi / merged["CPI"])

#Getting things per capita
population_adjustable_columns = [
   "bottles_regular_sold",
   "bottles_mix_bev_sold",
   "bottles_mini_sold",
   "real_retail_sales",
   "real_beverage_sales",
   "real_wine_sales",
   "real_gross_sales",
   "real_expenditures_millions"
]

for col in population_adjustable_columns:
   merged[f"per_capita_{col}"] = merged[col] / merged["population"]

#Dropping duplicated county
merged = merged.loc[:, ~merged.columns.duplicated()]

#Pulling in distances
distances = pd.read_csv("nc_county_pwcentroid_distances.csv")
distances["county_name"] = distances["county_name"] + " County"

merged = pd.merge(merged, distances, right_on = "county_name", left_on =
"county", how="left")

#Pulling in geographic stuff
area = pd.read_csv("CountySizes.txt", sep="|")

#Keeping only NC counties
area = area[area["USPS"].str.startswith("NC")]

area = area[["GEOID", "ALAND_SQMI", "NAME"]]

area = area.rename(columns={
   "GEOID": "fips",
   "ALAND_SQMI": "land_area_sqmi",
   "NAME": "county",
})
```

```
merged = pd.merge(merged, area, on = "county", how = "left")

merged["stores_per_100_sqmi"] = (
   merged["#_of_stores"] / merged["land_area_sqmi"]
) * 100

merged["stores_per_100k"] = (
   merged["#_of_stores"] / merged["population"]
) * 100000

merged.to_csv("AnalysisOneDataset.csv", index=False)
```

```
import pandas as pd


def check_merge(left, right, merged, left_keys, right_keys=None, name=""):
   if right_keys is None:
       right_keys = left_keys

   print(f"\n--- {name} ---")
   print(f"Left rows: {left.shape[0]}")
   print(f"Right rows: {right.shape[0]}")
   print(f"Merged rows: {merged.shape[0]}")

   if merged.shape[0] == left.shape[0]:
       print("Row count: OK")
   elif merged.shape[0] > left.shape[0]:
       print("Row count: INCREASED (possible duplicates)")
   else:
       print("Row count: DECREASED (lost rows)")

   test = left.merge(
       right,
       left_on=left_keys,
       right_on=right_keys,
       how="left",
       indicator=True
   )
   unmatched = (test["_merge"] == "left_only").sum()
   print(f"Unmatched left rows: {unmatched}")


cpi = {
   2006: 201.56, 2007: 207.34, 2008: 215.25, 2009: 214.56,
   2010: 218.08, 2011: 224.92, 2012: 229.59, 2013: 232.95,
   2014: 236.72, 2015: 237.00, 2016: 240.01, 2017: 245.12,
   2018: 251.10, 2019: 255.65, 2020: 258.86, 2021: 270.97,
   2022: 292.63, 2023: 304.70, 2024: 313.70
```

```
}

population_count = pd.read_csv("nc_population_panel_2006_2024.csv")
unemployment_count = pd.read_csv("nc_unemployment_panel_2006_2024.csv")
alcohol_sales = pd.read_csv("nc_alcohol_panel_2006_2024.csv")
tourism = pd.read_csv("nc_tourism_panel_2006_2024.csv")

# --- Merge 1 ---
m1 = pd.merge(population_count, unemployment_count, on=["county", "year"])
check_merge(population_count, unemployment_count, m1, ["county", "year"],
name="pop + unemp")

# --- Merge 2 ---
alcohol_sales["County"] = alcohol_sales["County"].str.replace(" County", "",
regex=False)
m2 = pd.merge(m1, alcohol_sales, left_on=["county", "year"],
right_on=["County", "year"])
check_merge(m1, alcohol_sales, m2, ["county", "year"], ["County", "year"],
name="+ alcohol")

# --- Merge 3 ---
m3 = pd.merge(m2, tourism, left_on=["county", "year"], right_on=["County",
"year"])
check_merge(m2, tourism, m3, ["county", "year"], ["County", "year"], name="+
tourism")

# --- distances ---
distances = pd.read_csv("nc_county_pwcentroid_distances.csv")
distances["county_name"] = distances["county_name"]

m4 = pd.merge(m3, distances, right_on="county_name", left_on="county",
how="left")
check_merge(m3, distances, m4, ["county"], ["county_name"], name="+ distances")

# --- land area ---
area = pd.read_csv("CountySizes.txt", sep="|")
area = area[area["USPS"].str.startswith("NC")]
area = area[["GEOID", "ALAND_SQMI", "NAME"]]
area = area.rename(columns={
   "GEOID": "fips",
   "ALAND_SQMI": "land_area_sqmi",
   "NAME": "county",
})
area["county"] = area["county"].str.replace(" County", "", regex=False)

m5 = pd.merge(m4, area, on="county", how="left")
check_merge(m4, area, m5, ["county"], name="+ area")

merged = m5
```

```
columns = [
   "county",
   "population",
   "unemployment_rate",
   "Retail Sales",
   "Beverage Sales",
   "Wine Sales",
   "Gross Sales",
   "Bottles Regular Sold",
   "Bottles Mix Bev Sold",
   "Bottles Mini Sold",
   "# of Stores",
   "Expenditures",
   "land_area_sqmi",
   "year",
   "dist_mi_to_SC",
   "dist_mi_to_TN",
   "dist_mi_to_VA"
]

df = merged[columns]

df.columns = (
   df.columns
   .str.lower()
   .str.replace("#", "num", regex=False)
   .str.replace(r"[^\w]+", "_", regex=True)
   .str.strip("_")
)

numeric_cols = [
   "retail_sales",
   "beverage_sales",
   "wine_sales",
   "gross_sales",
   "expenditures",
   "bottles_regular_sold",
   "bottles_mix_bev_sold",
   "bottles_mini_sold",
   "num_of_stores",
]

for col in numeric_cols:
   df[col] = pd.to_numeric(
       df[col]
       .astype(str)
       .str.replace(",", "", regex=False)
       .str.replace("$", "", regex=False)
```

```
        .str.strip(),
        errors="coerce"
    )

real = [
    "retail_sales",
    "beverage_sales",
    "wine_sales",
    "gross_sales",
    "expenditures",
]

per_capita = [
    "retail_sales_real",
    "beverage_sales_real",
    "wine_sales_real",
    "gross_sales_real",
    "bottles_regular_sold",
    "bottles_mix_bev_sold",
    "bottles_mini_sold",
    "expenditures_real",
]

base_cpi = cpi[2024]

df['cpi'] = df['year'].map(cpi)
df['inflator'] = base_cpi / df['cpi']

for col in real:
    df[f"{col}_real"] = df[col] * df['inflator']

for col in per_capita:
    source = f"{col}_real" if col in real else col
    df[f"{col}_pc"] = df[source] / df['population']

df = df.drop(columns=['cpi', 'inflator'])

df["stores_per_100_sqmi"] = (
                                    df["num_of_stores"] / df["land_area_sqmi"]
                            ) * 100

df["stores_per_100k"] = (
                                df["num_of_stores"] / df["population"]
                        ) * 100000

df.to_csv("AnalysisTwoDataset.csv", index=False)
```