cleanup

March 6, 2024

Prepares tax-related data from IPUMS for analysis. Includes cleaning, reshaping, and representative parent generation (optional).

[ ]:

```python
import pandas as pd
import numpy as np

# Set the directories for raw and intermediate data
raw_data_root = "/Users/aneeshtekulapally/Documents/College/Thesis/Code/Data/
 ↪IPUMS" # Path to the folder containing downloaded data from IPUMS
clean_data_root = "/Users/aneeshtekulapally/Documents/College/Thesis/Code/Data/
 ↪Clean" # Path to the folder that will contain the cleaned data

# # File paths for input and output
mtr_in_file = f"{intermed_root}/NTR_cleaned.csv"
mtr_out_file = f"{intermed_root}/nominal_tax_rates"

# Year ranges and flags

census_start, census_end = 1940, 2020
cps_start, cps_end = 1970, 2023

# Flags for processing steps, just for compartmentalizing computing
adjust_1940 = True
clean_census = True
clean_cps = True
reshape_census = True
gen_rep_parents = True
reshape_cps = True

# to reshape marginal tax rate
#reshape_mtr = True

# Additional variables for representative parents generation
partype = "richer" # options: richer, older, mother, father
pargender = None # options: male, female, None for no restriction

exclude_gq = "match_cps" #options: all, none, match_cps
```

Adjusts income categories within the 1940 Census data to align with the income categories present in the 1950 Census.

[ ]:

```
if adjust_1940:
    # Load the 1950 Census data
    census1950_df = pd.read_csv(f"{raw_data_root}/Census1950_raw.csv")
    census1950_df.sort_values(by=['SERIAL', 'PERNUM'], inplace=True)

    # Replace specified values with NaN (missing values)
    census1950_df['INCWAGE'].replace(999999, np.nan, inplace=True)
    census1950_df['INCTOT'].replace(9999999, np.nan, inplace=True)
    census1950_df['FTOTINC'].replace([9999999, 9999998], np.nan, inplace=True)

    # Replace FTOTINC missing values with inctot where applicable
    census1950_df.loc[census1950_df['FTOTINC'].isna(), 'FTOTINC'] =␣
↪census1950_df['INCTOT']

    # Keep rows with non-missing FTOTINC and incwage, and FTOTINC >= 0
    census1950_df.dropna(subset=['FTOTINC', 'INCWAGE'], inplace=True)
    census1950_df = census1950_df[census1950_df['FTOTINC'] >= 0]

    # Calculate non-wage income
    census1950_df['NONWAGE'] = census1950_df['FTOTINC'] -␣
↪census1950_df['INCWAGE']

    # Deflate to 1939 Dollars using CPI
    census1950_df['NONWAGE'] *= 0.58

    # Replace specific occupation and race codes with NaN
    census1950_df['OCC1950'].replace([999, 997], np.nan, inplace=True)
    census1950_df['RACE'].replace(list(range(3, max(census1950_df['RACE'])+1)),␣
↪3, inplace=True)

    # Generate incnonwg categories based on nonwage income
    census1950_df['INCNONWG'] = pd.cut(census1950_df['NONWAGE'], bins=[-np.inf,␣
↪0, 50, np.inf], labels=["missing", 1, 2])

    # Collapse the dataframe by mean nonwage for groups
    collapsed_df = census1950_df.groupby(['OCC1950', 'CLASSWKR', 'RACE',␣
↪'INCNONWG'])['NONWAGE'].mean().reset_index()
    collapsed_df['NONWAGE'] = collapsed_df['NONWAGE'].round()

    # Save the adjusted data for 1940
    collapsed_df.to_csv(f"{raw_data_root}/inc_adj_for_1940.csv", index=False)

    # Load the 1940 Census data
    census1940_df = pd.read_csv(f"{raw_data_root}/Census1940_raw.csv")
    # Replace specific occupation and race codes with NaN
    census1940_df['OCC1950'].replace([999, 997], np.nan, inplace=True)
    census1940_df['RACE'].replace(list(range(3, max(census1940_df['RACE'])+1)),␣
↪3, inplace=True)

    # Merge the 1940 dataset with the collapsed dataset
    merged_df = pd.merge(census1940_df, collapsed_df, on=['OCC1950',␣
↪'CLASSWKR', 'RACE', 'INCNONWG'], how='left')

    # Adjust nonwage based on incnonwg and merge status
    # Assuming 'missing' is a category for missing INCNONWG values and actual␣
↪NaNs might exist
    merged_df.loc[(merged_df['INCNONWG'].isnull()) | (merged_df['OCC1950'].
↪isna()), 'NONWAGE'] = np.nan
    # For rows where INCNONWG == 1 and NONWAGE is not NaN, set NONWAGE to 50
    merged_df.loc[(merged_df['INCNONWG'] == '1') & (merged_df['NONWAGE'].
↪notna()), 'NONWAGE'] = 50

    # Save the merged and adjusted 1940 Census data
    merged_df.to_csv(f"{clean_data_root}/Census1940_raw_adjusted.csv",␣
↪index=False)
```

[ ]:

```
import os
if clean_census:
    for year in range(census_start, census_end + 1, 10):
        os.chdir(raw_data_root)
        df = pd.read_csv(f"Census{year}_raw.csv")

        # Dropping observations based on missing values
        df.dropna(subset=['YEAR', 'PERNUM', 'SERIAL', 'SEX', 'AGE'],␣
↪inplace=True)

        # Dropping observations based on 'gq' values
        if exclude_gq == "all":
            df = df[df['GQ'] <= 1]
        elif exclude_gq == "match_cps":
            df = df[~df['GQTYPE'].isin([1, 2, 3, 4, 6])]

        # Dropping variables if they exist
        for drop_var in ['DATADNUM', 'CLUSTER', 'STRATA', 'RELATED', 'PERWT',␣
↪'GQ', 'GQTYPE']:
            if drop_var in df.columns:
                df.drop(columns=[drop_var], inplace=True)

        # Creating variables if they don't exist
        for required_var in ['INCTOT', 'FTOTINC', 'INCWELFR', 'INCSUPP']:
            if required_var not in df.columns:
                df[required_var] = np.nan

        # Replacing certain values with NaN based on conditions
        df['INCWAGE'].replace({999999: np.nan, 999998: np.nan}, inplace=True)
        if year > 1940:
            df['INCTOT'].replace({9999999: np.nan}, inplace=True)
            df['FTOTINC'].replace({9999999: np.nan, 9999998: np.nan},␣
 ↪inplace=True)
        if year > 1960:
            df['INCWELFR'].replace({99999: np.nan}, inplace=True)
        if year > 1990:
            df['INCSUPP'].replace({99999: np.nan}, inplace=True)
        if year >= 2000:
            df['INCWELFR'] = np.where(df['INCSUPP'].notna(), df['INCWELFR'] +␣
 ↪df['INCSUPP'], df['INCWELFR'])

        # Generating a unique ID and sorting
        df['id'] = 100 * df['SERIAL'] + df['PERNUM']
        df.sort_values(by=['YEAR', 'id'], inplace=True)

        # Changing directory and saving the cleaned file
        os.chdir(clean_data_root)
        df.to_csv(f"census{year}.csv", index=False)
```

[ ]:
```
if clean_cps:
    # Change directory to raw data root
    os.chdir(raw_data_root)

    # Load the data
    cps_data = pd.read_csv("CPS_1962to2023.csv")

    # Merge with cps_swapvalues data
    cps_swapvalues = pd.read_csv("cps_swapvalues.csv")
    cps_data = cps_data.merge(cps_swapvalues, on=["YEAR", "SERIAL", "PERNUM"],␣
 ↪how="left", indicator=False)

    # Variables to be replaced with NaN where value is 99999
    vlist = ['INCALIM', 'INCALOTH', 'INCASIST', 'INCCHILD','INCDRT',
           ␣
 ↪'INCEDUC','INCGOV','INCIDR','INCINT','INCOTHER','INCRENT','INCSS','INCSSI','INCUNEMP','INCV
    for var in vlist:
        cps_data[var] = cps_data[var].replace(99999, np.nan)

    # Variables to be replaced with NaN where value is 999999
    vrbls = ["INCDISAB", "INCDIVID", "INCLONGJ", "INCRETIR", "INCSURV",␣
 ↪"OINCBUS", "OINCFARM", "FTOTVAL"]
```

```
    for var in vrbls:
        cps_data[var] = cps_data[var].replace(999999, np.nan)

    # Variables to be replaced with NaN where value is 9999999 or 9999998
    vrs = ["INCBUS", "INCFARM", "INCWAGE", "OINCWAGE"]
    for var in vrs:
        cps_data[var] = cps_data[var].replace([9999999, 9999998], np.nan)

    # Special cases
    cps_data["INCTOT"] = cps_data["INCTOT"].replace([99999999, 99999998], np.
↪nan)
    cps_data["SPLOC"] = cps_data["SPLOC"].fillna(0).astype(int)
    cps_data.loc[cps_data["SPLOC"] < 0, "sploc"] = 0

    # Swap values where applicable
    vars_to_swap = ["INCWAGE", "INCBUS", "INCFARM", "INCSS", "INCWELFR",␣
↪"INCGOV", "INCALOTH", "INCRETIR", "INCSSI", "INCDRT", "INCINT", "INCUNEMP",␣
↪"INCWKCOM", "INCVET", "INCSURV", "INCDISAB", "INCDIVID", "INCRENT",␣
↪"INCEDUC", "INCCHILD", "INCALIM", "INCASIST", "INCOTHER", "INCLONGJ",␣
↪"OINCBUS", "OINCFARM", "OINCWAGE"]
    for var in vars_to_swap:
        swap_var = f"{var}_SWAP"
        condition = (~pd.isna(cps_data[swap_var])) & (cps_data[swap_var] != 0)
        cps_data.loc[condition, var] = cps_data.loc[condition, swap_var]

    # Define lists for row total calculations
    list75 = ['INCWAGE', 'INCBUS', 'INCFARM', 'INCSS', 'INCWELFR', 'INCIDR',␣
↪'INCALOTH', 'INCGOV']
    list87 = ['INCWAGE', 'INCBUS', 'INCFARM', 'INCSS', 'INCWELFR', 'INCRETIR',␣
↪'INCSSI', 'INCINT', 'INCDRT', 'INCALOTH', 'INCGOV']
    list88 = ['OINCWAGE', 'OINCBUS', 'OINCFARM', 'INCSS', 'INCWELFR',␣
↪'INCRETIR', 'INCSSI', 'INCINT', 'INCLONGJ', 'INCUNEMP', 'INCWKCOM',␣
↪'INCVET', 'INCSURV', 'INCDISAB', 'INCDIVID', 'INCRENT', 'INCEDUC',␣
↪'INCCHILD', 'INCALIM', 'INCASIST', 'INCOTHER']

    # Calculate row totals for specified years
    cps_data['inctotal75'] = cps_data[list75].sum(axis=1).
↪where(cps_data['YEAR'].between(1968, 1975), 0)
    cps_data['inctotal87'] = cps_data[list87].sum(axis=1).
↪where(cps_data['YEAR'].between(1976, 1987), 0)
    cps_data['inctotal88'] = cps_data[list88].sum(axis=1).
↪where(cps_data['YEAR'] >= 1988, 0)
    cps_data['INCTOT_NO_TOP'] = cps_data[['inctotal75', 'inctotal87',␣
↪'inctotal88']].sum(axis=1)
```

```
    cps_data.rename(columns={'INCTOT': 'INCTOT_TOPCODED', 'INCTOT_NO_TOP':␣
↪'INCTOT'}, inplace=True)
    cps_data["INCWELFR"] = np.where((cps_data["YEAR"] >= 1976) &␣
↪(~cps_data["INCSSI"].isna()), cps_data["INCWELFR"] + cps_data["INCSSI"],␣
↪cps_data["INCWELFR"])

    # Generating ids
    cps_data["ID"] = 100 * cps_data["SERIAL"] + cps_data["PERNUM"]
    cps_data["MOM_ID"] = np.where((cps_data["MOMLOC"].notna()) &␣
↪(cps_data["MOMLOC"] > 0), 100 * cps_data["SERIAL"] + cps_data["MOMLOC"], np.
↪nan)
    cps_data["POP_ID"] = np.where((cps_data["POPLOC"].notna()) &␣
↪(cps_data["POPLOC"] > 0), 100 * cps_data["SERIAL"] + cps_data["POPLOC"], np.
↪nan)

    # Renaming weights
    cps_data.rename(columns={"ASECWT": "WTSUPP", "ASECWTH": "HWTSUPP"},␣
↪inplace=True)

    # Keeping specified columns
    columns_to_keep = ['YEAR', 'ID', 'MARST', 'SERIAL', 'MOM_ID', 'POP_ID',␣
↪'SPLOC', 'WTSUPP', 'BPL', 'AGE', 'SEX', 'INCTOT', 'INCTOT_TOPCODED',␣
↪'FTOTVAL', 'INCWAGE', 'INCWELFR', 'INCSSI', 'NCHILD', 'FAMSIZE']
    cps_data = cps_data[columns_to_keep]

    # Compressing (in pandas, this usually means converting data types for␣
↪efficiency)
    cps_data = cps_data.convert_dtypes()

    # Change directory to intermediate data root and save
    os.chdir(clean_data_root)
    cps_data.to_csv("CPS_intermed.csv", index=False)
```

[ ]:

```
if reshape_census:
    for year in range(census_start, census_end + 1, 10):
        # Reshape data into couples
        os.chdir(clean_data_root)
        df = pd.read_csv(f"census{year}.csv")

        # Processing singles
        df_singles = df[df['SPLOC'] == 0]
        df_singles.sort_values(by=['YEAR', 'id'], inplace=True)
        census_singles = f"census_singles_{year}.csv"
        df_singles.to_csv(census_singles)

        # Processing spouses
        df = pd.read_csv(f"census{year}.csv")
        df_spouses = df[(df['SPLOC'].notnull()) & (df['SPLOC'] > 0)]
        df_spouses['ID_SPOUSE'] = 100 * df_spouses['SERIAL'] +␣
↪df_spouses['SPLOC']
        df_spouses.rename(columns={'id': 'OWNID', 'INCWAGE': 'INCWAGE_SPOUSE',␣
↪'AGE': 'AGE_SPOUSE', 'SEX': 'SEX_SPOUSE', 'BPL': 'BPL_SPOUSE', 'INCWELFR':␣
↪'INCWELFR_SPOUSE', 'INCSUPP': 'INCSUPP_SPOUSE', 'INCTOT': 'INCTOT_SPOUSE'},␣
↪inplace=True)
        df_spouses = df_spouses[['YEAR', 'ID_SPOUSE', 'AGE_SPOUSE', 'OWNID',␣
↪'SEX_SPOUSE', 'BPL_SPOUSE', 'INCWAGE_SPOUSE', 'INCSUPP_SPOUSE',␣
↪'INCTOT_SPOUSE', 'INCWELFR_SPOUSE']]
        census_spouses = f"census_spouses_{year}.csv"
        df_spouses.to_csv(census_spouses)

        # Merging spouses
        df = pd.read_csv(f"census{year}.csv")
        df.rename(columns={'id': 'ID_SPOUSE'}, inplace=True)
        df.sort_values(by=['YEAR', 'SERIAL', 'PERNUM'], inplace=True)
        df_merged = pd.merge(df, df_spouses, on=['YEAR', 'ID_SPOUSE'],␣
↪how='inner').drop_duplicates()
        df_merged['dup_tag'] = df_merged.groupby('SERIAL').cumcount() + 1
        df_merged = df_merged[(df_merged['dup_tag'] % 2) == 1]
        df_merged.rename(columns={'ID_SPOUSE': 'id', 'OWNID': 'ID_SPOUSE'},␣
↪inplace=True)

        # Appending singles and saving
        df_final = pd.concat([df_merged, df_singles]).sort_values(by=['YEAR',␣
↪'id'])
        if 'NONWAGE' not in df_final.columns:
            df_final['NONWAGE'] = np.nan

        df_final.to_csv(f"census{year}_couples.csv")

        # Generating identifiers for parents based on paternal and maternal␣
↪locations
        df = pd.read_csv(f"census{year}_couples.csv")
        df['ID_FATHER'] = np.where(df['POPLOC'].notnull() & (df['POPLOC'] > 0),␣
↪100 * df['SERIAL'] + df['POPLOC'], np.nan)
        df['ID_MOTHER'] = np.where(df['MOMLOC'].notnull() & (df['MOMLOC'] > 0),␣
↪100 * df['SERIAL'] + df['MOMLOC'], np.nan)
        df['ID_PARENT'] = df['ID_MOTHER']
        df['ID_PARENT'].fillna(df['ID_FATHER'], inplace=True)

        df.rename(columns={'id': 'ID_CHILD',
                           'AGE': 'AGE_CHILD',
                           'SEX': 'SEX_CHILD',
                           'BPL': 'BPL_CHILD'}, inplace=True)
```

```python
        df_children = df[['YEAR', 'ID_PARENT', 'ID_CHILD', 'AGE_CHILD', 'SEX_CHILD', 'BPL_CHILD']].dropna(subset=['ID_PARENT'])

        # Save children data for later merge
        census_kids = f"census_kids_{year}.csv"
        df_children.to_csv(census_kids)

        # Load couples data
        df_couples = pd.read_csv(f"census{year}_couples.csv")
        df_couples = df_couples[df_couples['NCHILD'] > 0]

        # Determine parent ID
        df_couples['ID_PARENT'] = np.where(df_couples['SEX_SPOUSE'] == 2, df_couples['ID_SPOUSE'], df_couples['id'])

        # Drop duplicate parents
        df_parents = df_couples[['ID_PARENT']].drop_duplicates()

        # Save parents data for later merge
        census_parents = f"census_parents_{year}.csv"
        df_parents.to_csv(census_parents)

        # Merge children with their parents
        df_families = pd.merge(df_children, df_parents, on='ID_PARENT', how='inner')

        # Final compress and save
        df_families.to_csv(f"census{year}_families.csv")
```

```
/usr/local/lib/python3.7/site-packages/pandas/util/_decorators.py:311:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return func(*args, **kwargs)
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:16:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  app.launch_new_instance()
/usr/local/lib/python3.7/site-packages/pandas/core/frame.py:5047:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

[ ]:

```
# os.chdir(clean_data_root)
# if gen_rep_parents:
#     for census_year in range(census_start, census_end + 1, 10):
#         families_df = pd.read_csv(f"census{year}_families.csv")
```

[ ]:

```
os.chdir(clean_data_root)
cps_intermed = pd.read_csv('CPS_intermed.csv')
print(cps_intermed.columns)
```

```
Index(['YEAR', 'ID', 'MARST', 'SERIAL', 'MOM_ID', 'POP_ID', 'SPLOC', 'WTSUPP',
       'BPL', 'AGE', 'SEX', 'INCTOT', 'INCTOT_TOPCODED', 'FTOTVAL', 'INCWAGE',
       'INCWELFR', 'INCSSI', 'NCHILD', 'FAMSIZE'],
      dtype='object')
```

[ ]:

```
os.chdir(clean_data_root)
if reshape_cps:
    # Load the data into a DataFrame
    cps_intermed = pd.read_csv('CPS_intermed.csv')

    # Generate `hasSpouse` column
    cps_intermed['hasSpouse'] = (cps_intermed['MARST'] == 1) &␣
 ↪(cps_intermed['SPLOC'] != 0)

    # Extract single people and put them aside for now
    cps_singles = cps_intermed[cps_intermed['hasSpouse'] == 0].
 ↪sort_values(by=['YEAR', 'ID'])
    # Save the singles DataFrame to a temporary file
    cps_singles_path = 'cps_singles.csv'
    cps_singles.to_csv(cps_singles_path, index=False)

    # Extract married individuals and reshape them
    cps_intermed = pd.read_csv('CPS_intermed.csv')
    cps_intermed['hasSpouse'] = (cps_intermed['MARST'] == 1) &␣
 ↪(cps_intermed['SPLOC'] != 0)
    cps_married = cps_intermed[cps_intermed['hasSpouse'] > 0]
    cps_married['SPOUSE_ID'] = 100 * cps_married['SERIAL'] +␣
 ↪cps_married['SPLOC']

    # Renaming columns
    rename_dict = {
        'ID': 'OWNID',
        'INCTOT': 'INCTOT_SPOUSE',
        'INCTOT_TOPCODED': 'INCTOT_TOPCODED_SPOUSE',
        'INCWAGE': 'INCWAGE_SPOUSE',
        'INCWELFR': 'INCWELFR_SPOUSE',
        'INCSSI': 'INCSSI_SPOUSE',
        'AGE': 'AGE_SPOUSE',
        'SEX': 'SEX_SPOUSE',
        'BPL': 'BPL_SPOUSE'
    }
    cps_married.rename(columns=rename_dict, inplace=True)

    # Keeping only the first instance in each group sorted by year and serial
    cps_married = cps_married.sort_values(by=['YEAR', 'SERIAL']).drop_duplicates(subset=['YEAR', 'SERIAL'], keep='first')
    # Keeping specified columns
    keep_columns = ['YEAR', 'SPOUSE_ID', 'OWNID', 'AGE_SPOUSE', 'SEX_SPOUSE', 'INCTOT_SPOUSE', 'INCWAGE_SPOUSE', 'INCWELFR_SPOUSE', 'INCSSI_SPOUSE', 'BPL_SPOUSE']
    cps_spouses = cps_married[keep_columns]

    # Save the spouses DataFrame to a temporary file
    cps_spouses_path = 'cps_spouses.csv'
    cps_spouses.to_csv(cps_spouses_path, index=False)

    # Merging back with the main dataset
    cps_intermed = pd.read_csv('CPS_intermed.csv')
    cps_intermed['hasSpouse'] = (cps_intermed['MARST'] == 1) & (cps_intermed['SPLOC'] != 0)
    cps_married_second = cps_intermed[cps_intermed['hasSpouse'] > 0]
    cps_married_second.rename(columns={'ID': 'SPOUSE_ID'}, inplace=True)
    cps_married_second = cps_married_second.sort_values(by=['YEAR', 'SERIAL']).drop_duplicates(subset=['YEAR', 'SERIAL'], keep='last')

    # Merging
    merged_data = pd.merge(cps_married_second, cps_spouses, on=['YEAR', 'SPOUSE_ID'], how='left', validate='1:m', indicator=True)
    merged_data = merged_data[merged_data['_merge'] == 'both'].drop(columns=['_merge'])

    # Renaming columns back
    merged_data.rename(columns={'SPOUSE_ID': 'ID', 'OWNID': 'SPOUSE_ID'}, inplace=True)

    # Adding singles back to reshaped couple data
    final_data = pd.concat([merged_data, cps_singles], ignore_index=True, sort=False)
    # Recoding `bpl` and generating `foreigner` and `foreigner_spouse`
    final_data['FOREIGNER'] = np.where(final_data['BPL'] == 9900, 0, np.
↪where(final_data['BPL'].isnull(), np.nan, 1))
    final_data['FOREIGNER_SPOUSE'] = np.where(final_data['BPL_SPOUSE'] == 9900,␣
↪0, np.where(final_data['BPL_SPOUSE'].isnull(), np.nan, 1))

    # Updating famsize`
    final_data['FAMSIZE'] = 1  # self
    final_data.loc[final_data['hasSpouse'] > 0, 'FAMSIZE'] += 1  # add one for␣
↪spouse
    final_data['FAMSIZE'] += final_data['NCHILD'].fillna(0)  # add kids,␣
↪assuming missing nchild implies 0

    # Dropping rows with specific conditions
    final_data = final_data.drop(final_data[(final_data['FTOTVAL'].isnull()) &␣
↪(final_data['INCTOT'].isnull()) & (final_data['INCTOT_SPOUSE'].isnull())].
↪index)

    # Keeping specified columns and sorting
    final_columns = ['YEAR', 'ID', 'MARST', 'NCHILD', 'SPOUSE_ID', 'MOM_ID',␣
↪'POP_ID', 'SEX', 'SEX_SPOUSE', 'AGE', 'AGE_SPOUSE', 'FOREIGNER',␣
↪'FOREIGNER_SPOUSE', 'FAMSIZE', 'WTSUPP', 'FTOTVAL', 'INCTOT',␣
↪'INCTOT_SPOUSE', 'INCWAGE', 'INCWAGE_SPOUSE', 'INCWELFR', 'INCWELFR_SPOUSE',␣
↪'INCSSI', 'INCSSI_SPOUSE']

    final_data = final_data[final_columns].sort_values(by=['YEAR', 'ID'])

    # Compressing and saving the final DataFrame
    final_data_path = 'CPS_clean.csv'
    final_data.to_csv(final_data_path, index=False)
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:19:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[ ]:

```
os.chdir(clean_data_root)
df = pd.read_csv("CPS_clean.csv")
num_rows = len(df)
```

[ ]:

```
print(num_rows)
```

```
7689156
```