wid

June 30, 2022

1 World Inequality Database

The World Inequality Database (WID.world) aims to provide open and convenient access to the most extensive available database on the historical evolution of the world distribution of income and wealth, both within countries and between countries. The dataset addresses some of the main limitations household surveys produce in national statistics of this kind: under-coverage at the top of the distribution due to non-response (the richest tend to not answer this kind of surveys or omit their income) or measurement error (the richest underreport their income for convenience or not actually knowing an exact figure if all their activities are added). The problem is handled with the combination of fiscal and national accounts data along household surveys based on the work of the leading researchers in the area: Anthony B. Atkinson, Thomas Piketty, Emmanuel Saez, Facundo Alvaredo, Gabriel Zucman, and hundreds of others. The initiative is based in the Paris School of Economics (as the World Inequality Lab) and compiles the World Inequality Report, a yearly publication about how inequality has evolved until the last year.

Besides income and wealth distribution data, the WID has recently added carbon emissions to generate carbon inequality indices. It also offers decomposed stats on national income. The data can be obtained from the website and by R and Stata commands.

1.1 Distributions considered in this analysis

Three income distributions are considered, coming in three different csv files: - **wid_pretax_992j_dist.csv** is the pretax income distribution `ptinc`, which includes social insurance benefits (and remove corresponding contributions), but exclude other forms of redistribution (income tax, social assistance benefits, etc.). - **wid_posttax_nat_992j_dist.csv** is the post-tax national income distribution `diinc`, which includes both in-kind and in-cash redistribution. - **wid_posttax_dis_992j_dist.csv** is the post-tax disposable income distribution `cainc`, which excludes in-kind transfers (because the distribution of in-kind transfers requires a lot of assumptions).

These distributions are the main DINA (distributional national accounts) income variables available at WID. DINA income concepts are distributed income concepts that are consistent with national accounts aggregates. The precise definitions are outlined in the DINA guidelines and country-specific papers.

All of these distributions are generated using equal-split adults (j) as the population unit, meaning that the unit is the individual, but that income or wealth is distributed equally among all household members. The age group is individuals over age 20 (992, adult population), which excludes children (with 0 income in most of the cases). Extrapolations and interpolations are excluded from these files, as WID discourages its use at the level of individual countries (see the `exclude` description at help wid in Stata). More information about the variables and definitions can be found on WID's codes dictionary.

The distributions analysed in this notebook come from commands given in the `wid` function in Stata. These commands are located in the `wid_distribution.do` file from this same folder. Opening the file and pressing the *Execute (do)* button will generate the most recent data from WID. Both `.csv` and `.dta` files are available for analysis.

1.2 Main variables

[1]:
```python
import pandas as pd
import numpy as np
from pathlib import Path
import time
import seaborn as sns

#keep_default_na and na_values are included because there is a country labeled␣
↪NA, Namibia, which becomes null without the parameters

file = Path('wid_pretax_992j_dist.csv')
wid_pretax = pd.read_csv(file, keep_default_na=False,
                         na_values=['-1.#IND', '1.#QNAN', '1.#IND', '-1.#QNAN',␣
↪'#N/A N/A', '#N/A', 'N/A', 'n/a', '', '#NA',
                                    'NULL', 'null', 'NaN', '-NaN', 'nan',␣
↪'-nan', ''])

file = Path('wid_posttax_nat_992j_dist.csv')
wid_posttax_nat = pd.read_csv(file, keep_default_na=False,
                              na_values=['-1.#IND', '1.#QNAN', '1.#IND', '-1.
↪#QNAN', '#N/A N/A', '#N/A', 'N/A', 'n/a', '', '#NA',
                                         'NULL', 'null', 'NaN', '-NaN', 'nan',␣
↪'-nan', ''])

file = Path('wid_posttax_dis_992j_dist.csv')
wid_posttax_dis = pd.read_csv(file, keep_default_na=False,
                              na_values=['-1.#IND', '1.#QNAN', '1.#IND', '-1.
↪#QNAN', '#N/A N/A', '#N/A', 'N/A', 'n/a', '', '#NA',
                                         'NULL', 'null', 'NaN', '-NaN', 'nan',␣
↪'-nan', ''])

#The variable 'country_year' is created, to identify unique distributions:
wid_pretax['country_year'] = wid_pretax['country'] + wid_pretax['year'].
↪astype(str)
wid_posttax_nat['country_year'] = wid_posttax_nat['country'] +␣
↪wid_posttax_nat['year'].astype(str)
wid_posttax_dis['country_year'] = wid_posttax_dis['country'] +␣
↪wid_posttax_dis['year'].astype(str)
```

The key variables that come following transformations in Stata are: - **country** mostly follows the ISO 3166-1 alpha-2 standard, but also includes world regions, country subregions (rural and urban China, for example), former countries and countries not officially included in the standard. All the countries available are extracted. See https://wid.world/codes-dictionary/#country-code - **year** is the year of the distribution. All available years are extracted. - **percentile** is the percentile (or, more broadly, quantile) of the distribution. They are in the format $pXpY$, where X and Y are both numbers between 0 and 100. X correspond to the percentile for the lower bound of the group, and Y to the percentile for the upper bound (hence X < Y). 130 different quantiles are extracted, from p0p1 to p99p100, tenths of a percentile in the top 1% (p99p99.1, p99.1p99.2, p99.2p99.3, …, p99.8p99.9, p99.9p100), hundreds of a percentile in the top 0.1% (p99.9p99.91, p99.92p99.93, …, p99.98p99.99, p99.99p100), and thousands of a percentile in the top 0.01% (p99.99p99.991, p99.992p99.993, … , p99.998p99.999, p99.999p100). See https://wid.world/codes-dictionary/#percentile-code - **p** represent the same variable *percentile*, but presented in a more simple way to sort the dataset: the lower bound X is extracted from $pXpY$ and divided by 100 to get only numbers from 0 to 1. - **threshold** is the minimum level of income that gets you into a group. For example, the income threshold of the group p90p100 is the income of the poorest individuals in the top 10%. By definition, it is equal to the income threshold of the groups p90p99 or p90p91. - **average** is the average income of the people in the group. For example, the wealth average of the group p90p99 is the average income of the top 10% excluding the top 1%. - **share** is the income of the group, divided by the total for the whole population. For example, the income of the group p99p100 is the top 1% income share.

Threshold and average data is converted to 2017 USD PPP with the `xlcusp` command in Stata (see https://wid.world/codes-dictionary/#exchange-rate). The variables **age** and **pop** (age group and population unit, respectively) are also in the dataset, but mainly for internal reference as it is the same value for each observation (992 and j). Although there are more age groups and population units available to query, most of them do not return results as massive as with the 992 and j combination or they just do not return data (see the options here and here).

Basic descriptive statistics are presented for the three distributions:

[2]:
```python
wid_pretax.describe(include='all')
```

[2]:

	country	year	percentile	p	threshold
count	731157	731157.000000	731157	731157.000000	5.301080e+05
unique	225	NaN	130	NaN	NaN
top	US	NaN	p99.9p100	NaN	NaN
freq	13780	NaN	6358	NaN	NaN
mean	NaN	1997.424206	NaN	0.612528	3.845721e+05
std	NaN	17.986790	NaN	0.330383	2.101428e+06
min	NaN	1870.000000	NaN	0.000000	-1.396364e+05
25%	NaN	1989.000000	NaN	0.320000	4.664600e+03
50%	NaN	2001.000000	NaN	0.650000	2.008982e+04
75%	NaN	2010.000000	NaN	0.970000	7.836561e+04
max	NaN	2021.000000	NaN	0.999990	3.425009e+08

	average	share	inv_paretolorenz	age	pop
count	5.320890e+05	731157.000000	2.202000e+04	731157.0	731157

```
unique            NaN          NaN              NaN      NaN       1
top               NaN          NaN              NaN      NaN       j
freq              NaN          NaN              NaN      NaN  731157
mean     5.253262e+05     0.009878     1.125843e+05    992.0     NaN
std      3.507536e+06     0.020805     1.784960e+06      0.0     NaN
min     -6.981829e+04    -0.037000     9.999999e-01    992.0     NaN
25%      5.764000e+03     0.002300     1.681378e+00    992.0     NaN
50%      2.223688e+04     0.005600     2.208449e+00    992.0     NaN
75%      9.356016e+04     0.010700     2.583091e+00    992.0     NaN
max      8.711651e+08     1.314900     5.443422e+07    992.0     NaN

       country_year
count        731157
unique         6454
top          AE1998
freq            130
mean            NaN
std             NaN
min             NaN
25%             NaN
50%             NaN
75%             NaN
max             NaN
```

With 731,157 observations, the pretax income distribution file is with difference the largest out of the three. It also contains 224 different countries/regions, almost 5 times the number of the post-tax files. This makes up for a total of 6451 different distributions (different country-years available). Although there is data starting from the year 1870, the data is concentrated mostly in the last three decades (the median of the *year* variable is 2001).

[3]:
```
wid_posttax_nat.describe(include='all')
```

[3]:
```
       country           year percentile               p     threshold  \
count   191034  191034.000000     191034  191034.000000  1.696500e+05
unique      48            NaN        130            NaN           NaN
top         US            NaN    p99p100            NaN           NaN
freq     13780            NaN       1533            NaN           NaN
mean       NaN    1998.023551        NaN       0.611196  2.935455e+05
std        NaN      16.279737        NaN       0.330177  1.223517e+06
min        NaN    1900.000000        NaN       0.000000 -2.331180e+07
25%        NaN    1989.000000        NaN       0.320000  1.905012e+04
50%        NaN    2001.000000        NaN       0.650000  3.773512e+04
75%        NaN    2010.000000        NaN       0.970000  9.102725e+04
max        NaN    2020.000000        NaN       0.999990  6.422674e+07

             average          share  inv_paretolorenz       age     pop  \
count   1.696500e+05  191034.000000       5876.000000  191034.0  191034
```

```
unique            NaN            NaN                NaN       NaN        1
top               NaN            NaN                NaN       NaN        j
freq              NaN            NaN                NaN       NaN   191034
mean     3.912583e+05       0.008559           1.972184     992.0      NaN
std      2.213586e+06       0.009196           0.528526       0.0      NaN
min     -5.420606e+05      -0.108200           1.000000     992.0      NaN
25%      1.932018e+04       0.003700           1.622764     992.0      NaN
50%      3.810682e+04       0.007100           1.864115     992.0      NaN
75%      9.493897e+04       0.010900           2.194164     992.0      NaN
max      1.706076e+08       0.220200           6.889863     992.0      NaN

       country_year
count        191034
unique         1533
top          AL1996
freq            130
mean            NaN
std             NaN
min             NaN
25%             NaN
50%             NaN
75%             NaN
max             NaN
```

The post-tax national income distribution file contains 191,034 observations for only 48 countries, which make 1533 different distributions. The minimum year is 1900, although the distributions are again concentrated more recently (median 2001).

[4]:
```python
wid_posttax_dis.describe(include='all')
```

[4]:
```
       country           year percentile              p     threshold  \
count   177572  177572.000000     177572  177572.000000  1.558700e+05
unique      48            NaN        130            NaN           NaN
top         FR            NaN    p99p100            NaN           NaN
freq      5394            NaN       1533            NaN           NaN
mean       NaN    2000.489210        NaN       0.611895  2.245611e+05
std        NaN      11.308023        NaN       0.330284  8.362206e+05
min        NaN    1900.000000        NaN       0.000000 -1.901357e+07
25%        NaN    1991.000000        NaN       0.320000  1.415995e+04
50%        NaN    2002.000000        NaN       0.650000  2.847279e+04
75%        NaN    2010.000000        NaN       0.970000  7.100878e+04
max        NaN    2020.000000        NaN       0.999990  5.140730e+07

             average          share  inv_paretolorenz       age     pop  \
count   1.558700e+05  177572.000000       5452.000000  177572.0  177572
unique           NaN            NaN               NaN       NaN       1
top              NaN            NaN               NaN       NaN       j
```

```
freq              NaN          NaN             NaN      NaN  177572
mean     3.134349e+05     0.008684        2.108109    992.0     NaN
std      1.701817e+06     0.009994        0.540296      0.0     NaN
min     -4.424091e+05    -0.117900        1.000103    992.0     NaN
25%      1.477483e+04     0.003500        1.679212    992.0     NaN
50%      2.976164e+04     0.006900        2.072798    992.0     NaN
75%      7.743828e+04     0.011100        2.461455    992.0     NaN
max      1.477681e+08     0.228100        7.032377    992.0     NaN

       country_year
count        177572
unique         1533
top          AL1996
freq            130
mean            NaN
std             NaN
min             NaN
25%             NaN
50%             NaN
75%             NaN
max             NaN
```

The post-tax disposable income file is the one with less observations (177,552) for 48 countries making 1533 different distributions again. The minimum year is 1900 (median 2002).

1.3 Sanity checks for the income distributions

The distributions are explored more in detail to find and correct (if possible) errors in the original data.

1.3.1 The same quantiles available for each country-year

It is very important that the distribution contains all 130 quantiles requested by the original query in Stata, to be able to estimate inequality statistics properly. One way to see if this holds is by counting the different occurrences of *percentile* for each distribution. The dataframes are grouped by country and year for this purpose.

Pretax income

[5]:

```
pretax_count = wid_pretax.groupby(['country', 'year', 'country_year']).
 ↪nunique() #counts unique values for the variables
pretax_not130 = pretax_count[pretax_count['percentile']!=130].reset_index()␣
 ↪#new dataframe with new indices
pretax_not130
```

[5]:

```
  country  year country_year  percentile  p  threshold  average  share  \
0      AR  1932       AR1932           3  3          0        0      3
1      AR  1933       AR1933           3  3          0        0      3
2      AR  1934       AR1934           3  3          0        0      3
```

```
3        AR  1935        AR1935            3  3           0        0       3
4        AR  1936        AR1936            3  3           0        0       3
..       ...   ...          ...           ... ..           ...        ...     ...
846      ZW  1974        ZW1974            2  2           0        2       2
847      ZW  1975        ZW1975            2  2           0        2       2
848      ZW  1976        ZW1976            2  2           0        2       2
849      ZW  1977        ZW1977            2  2           0        2       2
850      ZW  1978        ZW1978            2  2           0        2       2

     inv_paretolorenz  age  pop
0                   0    1    1
1                   0    1    1
2                   0    1    1
3                   0    1    1
4                   0    1    1
..                ...  ...  ...
846                 0    1    1
847                 0    1    1
848                 0    1    1
849                 0    1    1
850                 0    1    1

[851 rows x 11 columns]
```

In the case of the pretax data there are 851 different distributions that do not have the 130 quantiles. The main stats of this group are in the following table.

[6]:
```
pretax_not130.describe(include='all')
```

[6]:
```
       country         year country_year  percentile            p  threshold  \
count      851   851.000000          851  851.000000   851.000000      851.0
unique      21          NaN          851         NaN          NaN        NaN
top         JP          NaN       AR1932         NaN          NaN        NaN
freq        93          NaN            1         NaN          NaN        NaN
mean       NaN  1942.028202          NaN    3.251469     3.156287        0.0
std        NaN    23.922555          NaN    5.087954     4.568976        0.0
min        NaN  1870.000000          NaN    1.000000     1.000000        0.0
25%        NaN  1927.000000          NaN    2.000000     2.000000        0.0
50%        NaN  1944.000000          NaN    3.000000     3.000000        0.0
75%        NaN  1961.000000          NaN    3.000000     3.000000        0.0
max        NaN  1979.000000          NaN   31.000000    28.000000        0.0

          average        share  inv_paretolorenz    age    pop
count  851.000000   851.000000             851.0  851.0  851.0
unique        NaN          NaN               NaN    NaN    NaN
top           NaN          NaN               NaN    NaN    NaN
freq          NaN          NaN               NaN    NaN    NaN
```

```
mean     2.443008   3.162162          0.0   1.0   1.0
std      5.328943   4.623421          0.0   0.0   0.0
min      0.000000   1.000000          0.0   1.0   1.0
25%      0.000000   2.000000          0.0   1.0   1.0
50%      2.000000   3.000000          0.0   1.0   1.0
75%      3.000000   3.000000          0.0   1.0   1.0
max     31.000000  31.000000          0.0   1.0   1.0
```

21 different countries are in this situation, with a range of years from 1870 to 1979 (median 1944). The amount of percentiles in this group range from 1 to 31 (median 3). The 21 countries are:

[7]:
```
pretax_not130.country.value_counts(dropna=False) #counts the ocurrences of the issue in each country
```

[7]:
```
JP    93
DE    71
GB    71
DK    65
FI    60
NO    50
SE    48
IE    46
ZW    45
NL    42
ZA    40
HU    30
IN    27
AR    27
CH    24
MW    24
ES    24
SG    21
ID    20
GR    13
KR    10
Name: country, dtype: int64
```

The list of country-years without 130 quantiles can be extracted and filtered to the original dataset to see which are the few quantiles presented.

[8]:
```
pretax_not130_list = list(pretax_not130.country_year.unique()) #Gets the list of country-year which do not have 130 quantiles

#Dataframe with only the country-year in the list:
wid_pretax_not130 = wid_pretax[wid_pretax['country_year'].isin(pretax_not130_list)].reset_index(drop=True)
#"Clean" dataframe, excluding countries-year with less than 130 quantiles
```

```
wid_pretax_clean = wid_pretax[~wid_pretax['country_year'].
 ↪isin(pretax_not130_list)].reset_index(drop=True)

wid_pretax_not130.percentile.value_counts(dropna=False) #counts unique values␣
 ↪of quantiles in the country-years with issues
```

[8]:
```
p99.9p100          755
p99p100            703
p99.99p100         553
p99.95p99.96        27
p99.998p99.999      27
p99.997p99.998      27
p99.996p99.997      27
p99.995p99.996      27
p99.994p99.995      27
p99.993p99.994      27
p99.992p99.993      27
p99.991p99.992      27
p99.99p99.991       27
p99.98p99.99        27
p99.97p99.98        27
p99.96p99.97        27
p99.93p99.94        27
p99.94p99.95        27
p99.92p99.93        27
p99.91p99.92        27
p99.9p99.91         27
p99.8p99.9          27
p99.7p99.8          27
p99.6p99.7          27
p99.5p99.6          27
p99.4p99.5          27
p99.3p99.4          27
p99.2p99.3          27
p99.1p99.2          27
p99p99.1            27
p99.999p100         27
Name: percentile, dtype: int64
```

All of them come from the top 1%, the last percentile or one of its subdivisions.

And filtering out the exceptions, now all the percentiles are represented uniformly in 5603 distributions:

[9]:
```
wid_pretax_clean.percentile.value_counts(dropna=False)
```

[9]:
```
p0p1         5603
p97p98       5603
```

```
p95p96         5603
p94p95         5603
p93p94         5603
               …
p38p39         5603
p37p38         5603
p36p37         5603
p35p36         5603
p99.999p100    5603
Name: percentile, Length: 130, dtype: int64
```

Post-tax national income For the post-tax national income distribution there are less cases:

[10]:
```
posttax_nat_count = wid_posttax_nat.groupby(['country', 'year',␣
↪'country_year']).nunique() #counts unique values for the variables
posttax_nat_not130 = posttax_nat_count[posttax_nat_count['percentile']!=130].
↪reset_index() #new dataframe with new indices
posttax_nat_not130
```

[10]:
```
   country  year country_year  percentile  p  threshold  average  share  \
0       FR  1900       FR1900           1  1          0        0      1
1       FR  1910       FR1910           1  1          0        0      1
2       FR  1915       FR1915           1  1          0        0      1
3       FR  1916       FR1916           1  1          0        0      1
4       FR  1917       FR1917           1  1          0        0      1
..     …   …          …         … ..        …      …    …
59      FR  1973       FR1973           1  1          0        0      1
60      FR  1974       FR1974           1  1          0        0      1
61      FR  1976       FR1976           1  1          0        0      1
62      FR  1977       FR1977           1  1          0        0      1
63      FR  1978       FR1978           1  1          0        0      1

    inv_paretolorenz  age  pop
0                  0    1    1
1                  0    1    1
2                  0    1    1
3                  0    1    1
4                  0    1    1
..               …  …  …
59                 0    1    1
60                 0    1    1
61                 0    1    1
62                 0    1    1
63                 0    1    1

[64 rows x 11 columns]
```

64 different distributions do not have 130 percentiles.

[11]:
```
posttax_nat_not130.describe(include='all')
```

[11]:
```
       country         year country_year  percentile     p  threshold  \
count       64    64.000000           64        64.0  64.0       64.0
unique       1          NaN           64         NaN   NaN        NaN
top         FR          NaN       FR1900         NaN   NaN        NaN
freq        64          NaN            1         NaN   NaN        NaN
mean       NaN  1944.390625          NaN         1.0   1.0        0.0
std        NaN    19.389587          NaN         0.0   0.0        0.0
min        NaN  1900.000000          NaN         1.0   1.0        0.0
25%        NaN  1928.750000          NaN         1.0   1.0        0.0
50%        NaN  1944.500000          NaN         1.0   1.0        0.0
75%        NaN  1960.250000          NaN         1.0   1.0        0.0
max        NaN  1978.000000          NaN         1.0   1.0        0.0

        average  share  inv_paretolorenz   age   pop
count      64.0   64.0              64.0  64.0  64.0
unique      NaN    NaN               NaN   NaN   NaN
top         NaN    NaN               NaN   NaN   NaN
freq        NaN    NaN               NaN   NaN   NaN
mean        0.0    1.0               0.0   1.0   1.0
std         0.0    0.0               0.0   0.0   0.0
min         0.0    1.0               0.0   1.0   1.0
25%         0.0    1.0               0.0   1.0   1.0
50%         0.0    1.0               0.0   1.0   1.0
75%         0.0    1.0               0.0   1.0   1.0
max         0.0    1.0               0.0   1.0   1.0
```

Only one country is in this situation (France), with a range of years from 1900 to 1978 (median 1944). There is always 1 percentile for each of these distributions.

All of these 64 percentiles are **p99p100, the top 1%**:

[12]:
```
posttax_nat_not130_list = list(posttax_nat_not130.country_year.unique()) #Gets␣
↪the list of country-year which do not have 130 quantiles

#Dataframe with only the country-year in the list:
wid_posttax_nat_not130 = wid_posttax_nat[wid_posttax_nat['country_year'].
↪isin(posttax_nat_not130_list)].reset_index(drop=True)
#"Clean" dataframe, excluding countries-year with less than 130 quantiles
wid_posttax_nat_clean = wid_posttax_nat[~wid_posttax_nat['country_year'].
↪isin(posttax_nat_not130_list)].reset_index(drop=True)

wid_posttax_nat_not130.percentile.value_counts(dropna=False) #counts unique␣
↪values of quantiles in the country-years with issues
```

[12]:
```
p99p100    64
Name: percentile, dtype: int64
```

And filtering out the exceptions, now all the percentiles are represented uniformly in 1469 distributions:

[13]:
```
wid_posttax_nat_clean.percentile.value_counts(dropna=False)
```

[13]:
```
p0p1           1469
p97p98         1469
p95p96         1469
p94p95         1469
p93p94         1469
               …
p38p39         1469
p37p38         1469
p36p37         1469
p35p36         1469
p99.999p100    1469
Name: percentile, Length: 130, dtype: int64
```

Post-tax disposable income There are more post-tax disposable income distributions that follow this category:

[14]:
```
posttax_dis_count = wid_posttax_dis.groupby(['country', 'year',␣
↪'country_year']).nunique() #counts unique values for the variables
posttax_dis_not130 = posttax_dis_count[posttax_dis_count['percentile']!=130].
↪reset_index() #new dataframe with new indices
posttax_dis_not130
```

[14]:
```
    country  year country_year  percentile  p  threshold  average  share  \
0        FR  1900       FR1900           1  1          0        0      1
1        FR  1910       FR1910           1  1          0        0      1
2        FR  1915       FR1915           1  1          0        0      1
3        FR  1916       FR1916           1  1          0        0      1
4        FR  1917       FR1917           1  1          0        0      1
..      …   …          …          … ..        …      …    …
165      US  2014       US2014           3  3          0        0      3
166      US  2015       US2015           3  3          0        0      3
167      US  2016       US2016           3  3          0        0      3
168      US  2017       US2017           3  3          0        0      3
169      US  2018       US2018           3  3          0        0      3

     inv_paretolorenz  age  pop
0                   0    1    1
1                   0    1    1
2                   0    1    1
```

```
3                     0    1    1
4                     0    1    1
..                 …   …   …
165                   0    1    1
166                   0    1    1
167                   0    1    1
168                   0    1    1
169                   0    1    1

[170 rows x 11 columns]
```

In the case of the post-tax disposable data there are 170 different distributions that do not have the 130 quantiles. The main stats of this group are in the following table.

[15]:
```
posttax_dis_not130.describe(include='all')
```

[15]:
```
       country         year country_year  percentile           p  threshold  \
count      170   170.000000          170  170.000000  170.000000      170.0
unique       2          NaN          170         NaN         NaN        NaN
top         US          NaN       FR1900         NaN         NaN        NaN
freq       106          NaN            1         NaN         NaN        NaN
mean       NaN  1957.552941          NaN    2.247059    2.247059        0.0
std        NaN    28.854873          NaN    0.971863    0.971863        0.0
min        NaN  1900.000000          NaN    1.000000    1.000000        0.0
25%        NaN  1934.000000          NaN    1.000000    1.000000        0.0
50%        NaN  1955.000000          NaN    3.000000    3.000000        0.0
75%        NaN  1977.000000          NaN    3.000000    3.000000        0.0
max        NaN  2018.000000          NaN    3.000000    3.000000        0.0

        average        share  inv_paretolorenz    age    pop
count     170.0   170.000000             170.0  170.0  170.0
unique      NaN          NaN               NaN    NaN    NaN
top         NaN          NaN               NaN    NaN    NaN
freq        NaN          NaN               NaN    NaN    NaN
mean        0.0     2.247059               0.0    1.0    1.0
std         0.0     0.971863               0.0    0.0    0.0
min         0.0     1.000000               0.0    1.0    1.0
25%         0.0     1.000000               0.0    1.0    1.0
50%         0.0     3.000000               0.0    1.0    1.0
75%         0.0     3.000000               0.0    1.0    1.0
max         0.0     3.000000               0.0    1.0    1.0
```

Only two countries are in this situation (France, US), with a range of years from 1900 to 2018 (median 1955). The amount of different percentiles for these groups range between 1 and 3. The cases are distributed as this table shows:

[16]:
```
posttax_dis_not130.country.value_counts(dropna=False)
```

[16]: US   106
FR   64
Name: country, dtype: int64

In this case the percentiles are the **top 1%, 0.1% and 0.01%:**

```
[17]: posttax_dis_not130_list = list(posttax_dis_not130.country_year.unique()) #Gets␣
 ↪the list of country-year which do not have 130 quantiles

#Dataframe with only the country-year in the list:
wid_posttax_dis_not130 = wid_posttax_dis[wid_posttax_dis['country_year'].
 ↪isin(posttax_dis_not130_list)].reset_index(drop=True)
#"Clean" dataframe, excluding countries-year with less than 130 quantiles
wid_posttax_dis_clean = wid_posttax_dis[~wid_posttax_dis['country_year'].
 ↪isin(posttax_dis_not130_list)].reset_index(drop=True)

wid_posttax_dis_not130.percentile.value_counts(dropna=False) #counts unique␣
 ↪values of quantiles in the country-years with issues
```

```
[17]: p99p100       170
p99.9p100     106
p99.99p100    106
Name: percentile, dtype: int64
```

And filtering out the exceptions, now all the percentiles are represented uniformly in 1363 distributions:

```
[18]: wid_posttax_dis_clean.percentile.value_counts(dropna=False)
```

```
[18]: p0p1           1363
p97p98         1363
p95p96         1363
p94p95         1363
p93p94         1363
               …
p38p39         1363
p37p38         1363
p36p37         1363
p35p36         1363
p99.999p100    1363
Name: percentile, Length: 130, dtype: int64
```

1.3.2 Monotonicity

When ordered by **p**, the threshold and average values for each country-year should not decrease. These can increase or stay the same, but never decrease. If this happens the construction of the distribution failed.

Pretax income distribution

[19]:
```
#These three quantiles are excluded to get an entirely continous series
excl_list = ['p99p100', 'p99.9p100', 'p99.99p100']
wid_pretax_monotonicity = wid_pretax_clean[~wid_pretax_clean['percentile'].
 ↪isin(excl_list)].reset_index(drop=True)
```

In the following code code the monotonicity is checked for the variables **average** and **threshold**.

[20]:
```
#The average and threshold values are lagged by one row in the lagged_average␣
 ↪and lagged_threshold variables for them to be compared

wid_pretax_monotonicity['lagged_average'] =  wid_pretax_monotonicity['average'].
 ↪shift(1) #average shifted by 1 row
wid_pretax_monotonicity['monotonicity_check_avg'] = ␣
 ↪wid_pretax_monotonicity['average'] >=␣
 ↪wid_pretax_monotonicity['lagged_average'] #True if average is greater than␣
 ↪or equal than the previous
wid_pretax_monotonicity.loc[(wid_pretax_monotonicity['percentile'] == 'p0p1'),␣
 ↪'monotonicity_check_avg'] = True #The first percentile gets automatically a␣
 ↪True value, as it cannot be compared
wid_pretax_monotonicity.loc[(wid_pretax_monotonicity['average'].isnull()) |␣
 ↪(wid_pretax_monotonicity['lagged_average'].isnull()),␣
 ↪'monotonicity_check_avg'] = np.nan #If values are null the comparison is null

#Same steps for threshold
wid_pretax_monotonicity['lagged_threshold'] = ␣
 ↪wid_pretax_monotonicity['threshold'].shift(1)
wid_pretax_monotonicity['monotonicity_check_thr'] = ␣
 ↪wid_pretax_monotonicity['threshold'] >=␣
 ↪wid_pretax_monotonicity['lagged_threshold']
wid_pretax_monotonicity.loc[(wid_pretax_monotonicity['percentile'] == 'p0p1'),␣
 ↪'monotonicity_check_thr'] = True
wid_pretax_monotonicity.loc[(wid_pretax_monotonicity['threshold'].isnull()) |␣
 ↪(wid_pretax_monotonicity['lagged_threshold'].isnull()),␣
 ↪'monotonicity_check_thr'] = np.nan
```

99.85% of the values for average and **99.64%** of the values for threshold **pass the test**:

[21]:
```
wid_pretax_monotonicity.monotonicity_check_avg.value_counts(normalize=True)
```

[21]:
```
True     0.998449
False    0.001551
Name: monotonicity_check_avg, dtype: float64
```

[22]:
```
wid_pretax_monotonicity.monotonicity_check_thr.value_counts(normalize=True)
```

[22]:
```
True     0.996365
False    0.003635
Name: monotonicity_check_thr, dtype: float64
```

These are the countries showing discontinuities in some of their distributions:

[23]:
```
#Dataframe keeping only the False checks (average)
pretax_avgfalse =␣
 ↪wid_pretax_monotonicity[wid_pretax_monotonicity['monotonicity_check_avg']==False].
 ↪reset_index(drop=True)
pretax_avgfalse.country.value_counts(dropna=False)
```

[23]:
```
SG    249
TW    246
PE     55
SV     46
AR     43
EC     31
CR     30
UY     22
BR     21
MX     19
CL     13
CO     11
HK      9
UA      7
NZ      1
Name: country, dtype: int64
```

[24]:
```
#Dataframe keeping only the False checks (threshold)
pretax_thrfalse =␣
 ↪wid_pretax_monotonicity[wid_pretax_monotonicity['monotonicity_check_thr']==False].
 ↪reset_index(drop=True)
pretax_thrfalse.country.value_counts(dropna=False)
```

[24]:
```
AU    520
NZ    318
TW    300
SG    297
CA    271
NO     25
UA     20
CZ     20
BY     16
IN     10
KR      9
RU      9
```

```
AM       9
DD       8
DK       6
BG       6
FI       4
PL       4
AZ       3
SK       3
CN       3
EG       2
KZ       2
KG       2
IE       2
HU       2
ZA       2
LT       1
LV       1
NP       1
EE       1
RO       1
AT       1
SI       1
BD       1
HK       1
Name: country, dtype: int64
```

The discontinuities are more concentrated in the subdivisions of the top 1% for the average and is more mixed for threshold:

[25]:
```
pretax_avgfalse.percentile.value_counts(dropna=False)
```

[25]:
```
p99.997p99.998    62
p99.995p99.996    60
p99.994p99.995    59
p99.998p99.999    57
p99.993p99.994    56
p99.992p99.993    54
p99.991p99.992    37
p99.95p99.96      36
p99.996p99.997    36
p99.92p99.93      35
p99.8p99.9        31
p99.91p99.92      30
p99.98p99.99      28
p99.94p99.95      25
p99.93p99.94      24
p99.97p99.98      22
```

```
p98p99            22
p99.1p99.2        19
p99.9p99.91       19
p99.7p99.8        14
p99.96p99.97      13
p99.5p99.6        12
p99.6p99.7         9
p99.2p99.3         9
p99.999p100        6
p99.4p99.5         5
p99.99p99.991      3
p92p93             3
p99.3p99.4         3
p24p25             2
p90p91             2
p97p98             1
p96p97             1
p67p68             1
p34p35             1
p56p57             1
p94p95             1
p66p67             1
p30p31             1
p31p32             1
p83p84             1
Name: percentile, dtype: int64
```

[26]:
```
#pd.set_option("display.max_rows", None)
pretax_thrfalse.percentile.value_counts(dropna=False)
```

[26]:
```
p21p22    402
p26p27    108
p22p23    107
p23p24    106
p28p29     83
         …
p60p61      1
p73p74      1
p86p87      1
p55p56      1
p98p99      1
Name: percentile, Length: 87, dtype: int64
```

Post-tax national income distribution

[27]:
```
#These three quantiles are excluded to get an entirely continous series
excl_list = ['p99p100', 'p99.9p100', 'p99.99p100']
```

```
wid_posttax_nat_monotonicity =␣
 ↪wid_posttax_nat_clean[~wid_posttax_nat_clean['percentile'].isin(excl_list)].
 ↪reset_index(drop=True)
```

[28]:
```
#The average and threshold values are lagged by one row in the lagged_average␣
 ↪and lagged_threshold variables for them to be compared

wid_posttax_nat_monotonicity['lagged_average'] = ␣
 ↪wid_posttax_nat_monotonicity['average'].shift(1) #average shifted by 1 row
wid_posttax_nat_monotonicity['monotonicity_check_avg'] = ␣
 ↪wid_posttax_nat_monotonicity['average'] >=␣
 ↪wid_posttax_nat_monotonicity['lagged_average'] #True if average is greater␣
 ↪than or equal than the previous
wid_posttax_nat_monotonicity.loc[(wid_posttax_nat_monotonicity['percentile'] ==␣
 ↪'p0p1'), 'monotonicity_check_avg'] = True #The first percentile gets␣
 ↪automatically a True value, as it cannot be compared
wid_posttax_nat_monotonicity.loc[(wid_posttax_nat_monotonicity['average'].
 ↪isnull()) | (wid_posttax_nat_monotonicity['lagged_average'].isnull()),␣
 ↪'monotonicity_check_avg'] = np.nan #If values are null the comparison is null

#Same steps for threshold
wid_posttax_nat_monotonicity['lagged_threshold'] = ␣
 ↪wid_posttax_nat_monotonicity['threshold'].shift(1)
wid_posttax_nat_monotonicity['monotonicity_check_thr'] = ␣
 ↪wid_posttax_nat_monotonicity['threshold'] >=␣
 ↪wid_posttax_nat_monotonicity['lagged_threshold']
wid_posttax_nat_monotonicity.loc[(wid_posttax_nat_monotonicity['percentile'] ==␣
 ↪'p0p1'), 'monotonicity_check_thr'] = True
wid_posttax_nat_monotonicity.loc[(wid_posttax_nat_monotonicity['threshold'].
 ↪isnull()) | (wid_posttax_nat_monotonicity['lagged_threshold'].isnull()),␣
 ↪'monotonicity_check_thr'] = np.nan
```

99.995% of the values for average and **100%** of the values of threshold **pass the test**:

[29]:
```
wid_posttax_nat_monotonicity.monotonicity_check_avg.value_counts(normalize=True)
```

[29]:
```
True     0.999952
False    0.000048
Name: monotonicity_check_avg, dtype: float64
```

[30]:
```
wid_posttax_nat_monotonicity.monotonicity_check_thr.value_counts(normalize=True)
```

[30]:
```
True    1.0
Name: monotonicity_check_thr, dtype: float64
```

```
[31]: #Dataframe keeping only the False checks (average)
posttax_nat_avgfalse =␣
 ↪wid_posttax_nat_monotonicity[wid_posttax_nat_monotonicity['monotonicity_check_avg']==False]
 ↪reset_index(drop=True)
posttax_nat_avgfalse.country.value_counts(dropna=False)
```

```
[31]: SK    6
FR    1
LU    1
Name: country, dtype: int64
```

```
[32]: #Dataframe keeping only the False checks (threshold)
posttax_nat_thrfalse =␣
 ↪wid_posttax_nat_monotonicity[wid_posttax_nat_monotonicity['monotonicity_check_thr']==False]
 ↪reset_index(drop=True)
posttax_nat_thrfalse.country.value_counts(dropna=False)
```

```
[32]: Series([], Name: country, dtype: int64)
```

```
[33]: posttax_nat_avgfalse.percentile.value_counts(dropna=False)
```

```
[33]: p2p3              3
p1p2              2
p99.997p99.998    1
p3p4              1
p4p5              1
Name: percentile, dtype: int64
```

```
[34]: posttax_nat_thrfalse.percentile.value_counts(dropna=False)
```

```
[34]: Series([], Name: percentile, dtype: int64)
```

Post-tax disposable income distribution

```
[35]: #These three quantiles are excluded to get an entirely continous series
excl_list = ['p99p100', 'p99.9p100', 'p99.99p100']
wid_posttax_dis_monotonicity =␣
 ↪wid_posttax_dis_clean[~wid_posttax_dis_clean['percentile'].isin(excl_list)].
 ↪reset_index(drop=True)
```

```
[36]: #The average and threshold values are lagged by one row in the lagged_average␣
 ↪and lagged_threshold variables for them to be compared

wid_posttax_dis_monotonicity['lagged_average'] = ␣
 ↪wid_posttax_dis_monotonicity['average'].shift(1) #average shifted by 1 row
```

```
wid_posttax_dis_monotonicity['monotonicity_check_avg'] = ␣
 ↪wid_posttax_dis_monotonicity['average'] >=␣
 ↪wid_posttax_dis_monotonicity['lagged_average'] #True if average is greater␣
 ↪than or equal than the previous
wid_posttax_dis_monotonicity.loc[(wid_posttax_dis_monotonicity['percentile'] ==␣
 ↪'p0p1'), 'monotonicity_check_avg'] = True #The first percentile gets␣
 ↪automatically a True value, as it cannot be compared
wid_posttax_dis_monotonicity.loc[(wid_posttax_dis_monotonicity['average'].
 ↪isnull()) | (wid_posttax_dis_monotonicity['lagged_average'].isnull()),␣
 ↪'monotonicity_check_avg'] = np.nan #If values are null the comparison is null

#Same steps for threshold
wid_posttax_dis_monotonicity['lagged_threshold'] = ␣
 ↪wid_posttax_dis_monotonicity['threshold'].shift(1)
wid_posttax_dis_monotonicity['monotonicity_check_thr'] = ␣
 ↪wid_posttax_dis_monotonicity['threshold'] >=␣
 ↪wid_posttax_dis_monotonicity['lagged_threshold']
wid_posttax_dis_monotonicity.loc[(wid_posttax_dis_monotonicity['percentile'] ==␣
 ↪'p0p1'), 'monotonicity_check_thr'] = True
wid_posttax_dis_monotonicity.loc[(wid_posttax_dis_monotonicity['threshold'].
 ↪isnull()) | (wid_posttax_dis_monotonicity['lagged_threshold'].isnull()),␣
 ↪'monotonicity_check_thr'] = np.nan
```

100% of the values for average and threshold **pass the test**:

[37]:
```
wid_posttax_dis_monotonicity.monotonicity_check_avg.value_counts(normalize=True)
```

[37]:
```
True    1.0
Name: monotonicity_check_avg, dtype: float64
```

[38]:
```
wid_posttax_dis_monotonicity.monotonicity_check_thr.value_counts(normalize=True)
```

[38]:
```
True    1.0
Name: monotonicity_check_thr, dtype: float64
```

[39]:
```
#Dataframe keeping only the False checks (average)
posttax_dis_avgfalse =␣
 ↪wid_posttax_dis_monotonicity[wid_posttax_dis_monotonicity['monotonicity_check_avg']==False]
 ↪reset_index(drop=True)
posttax_dis_avgfalse.country.value_counts(dropna=False)
```

[39]:
```
Series([], Name: country, dtype: int64)
```

[40]:
```
#Dataframe keeping only the False checks (threshold)
posttax_dis_thrfalse =␣
 ↪wid_posttax_dis_monotonicity[wid_posttax_dis_monotonicity['monotonicity_check_thr']==False]
 ↪reset_index(drop=True)
```

```
posttax_dis_thrfalse.country.value_counts(dropna=False)
```

[40]: Series([], Name: country, dtype: int64)

[41]:
```
posttax_dis_avgfalse.percentile.value_counts(dropna=False)
```

[41]: Series([], Name: percentile, dtype: int64)

[42]:
```
posttax_dis_thrfalse.percentile.value_counts(dropna=False)
```

[42]: Series([], Name: percentile, dtype: int64)

This is important to check the robustness of the **threshold** and **average** data across the years, to see a logical evolution of these numbers and not sudden jumps which might due to errors in the construction or due to the quality of the microdata.

1.3.3 Negative values

Negative income values, although common in the construction of distributions, usually are bottom coded to 0. In this section, negative values for average and threshold are checked.

As expected, negative values occur only in the first percentiles of the distribution (max p6p7 in post-tax disposable income). All this values are bottom coded to 0 in the *_positive* dataframes

Pretax income

[43]:
```
pretax_negative_avg = wid_pretax_clean[wid_pretax_clean['average'] < 0].
  ↪reset_index(drop=True) #keeps only average < 0
pretax_negative_thr = wid_pretax_clean[wid_pretax_clean['threshold'] < 0].
  ↪reset_index(drop=True) #keeps only threshold < 0

#This dataframe changes the negative values of average and threshold to 0
pretax_positive = wid_pretax_clean.copy()
pretax_positive.loc[(pretax_positive['average'] < 0), 'average'] = 0
pretax_positive.loc[(pretax_positive['threshold'] < 0), 'threshold'] = 0
```

[44]:
```
pretax_negative_thr.percentile.value_counts(dropna=False)
```

[44]:
```
p0p1    336
p1p2    277
p2p3    177
p3p4     53
Name: percentile, dtype: int64
```

[45]:
```
pretax_negative_thr.country.value_counts(dropna=False)
```

[45]:
```
WO       124
US       104
XB        96
```

```
QP         96
QF         62
OA         59
XL         55
QB         27
QT         27
XF         27
XR         21
QO         19
OI         16
QW         15
OE         12
OD         12
OB         11
QV         10
QK          8
QN          8
QE          8
CN-RU       6
CN-UR       6
BR          6
QJ          4
OJ          4
Name: country, dtype: int64
```

[46]:
```
pretax_negative_thr.percentile.value_counts(dropna=False)
```

[46]:
```
p0p1    336
p1p2    277
p2p3    177
p3p4     53
Name: percentile, dtype: int64
```

[47]:
```
pretax_negative_thr.country.value_counts(dropna=False)
```

[47]:
```
WO        124
US        104
XB         96
QP         96
QF         62
OA         59
XL         55
QB         27
QT         27
XF         27
XR         21
QO         19
```

```
OI        16
QW        15
OE        12
OD        12
OB        11
QV        10
QK         8
QN         8
QE         8
CN-RU      6
CN-UR      6
BR         6
QJ         4
OJ         4
Name: country, dtype: int64
```

Post-tax national income

[48]:
```
posttax_nat_negative_avg =␣
  ↪wid_posttax_nat_clean[wid_posttax_nat_clean['average'] < 0].
  ↪reset_index(drop=True) #keeps only average < 0
posttax_nat_negative_thr =␣
  ↪wid_posttax_nat_clean[wid_posttax_nat_clean['threshold'] < 0].
  ↪reset_index(drop=True) #keeps only threshold < 0

#This dataframe changes the negative values of average and threshold to 0
posttax_nat_positive = wid_posttax_nat_clean.copy()
posttax_nat_positive.loc[(posttax_nat_positive['average'] < 0), 'average'] = 0
posttax_nat_positive.loc[(posttax_nat_positive['threshold'] < 0), 'threshold']␣
  ↪= 0
```

[49]:
```
posttax_nat_negative_thr.percentile.value_counts(dropna=False)
```

[49]:
```
p0p1    496
p1p2    179
p2p3     18
p3p4      5
p4p5      1
Name: percentile, dtype: int64
```

[50]:
```
posttax_nat_negative_thr.country.value_counts(dropna=False)
```

[50]:
```
US    102
CH     86
QX     67
QY     59
QE     56
```

```
IS     45
PT     28
LV     22
DE     19
QM     18
EE     18
NL     18
BG     17
MD     16
GB     12
HR     10
DK     10
CY     10
MK     10
LU      8
ME      8
FI      7
ES      7
KS      7
SE      6
SK      5
RO      4
AL      4
PL      3
HU      3
CZ      3
BA      3
SI      2
LT      1
AT      1
IE      1
FR      1
BE      1
IT      1
Name: country, dtype: int64
```

[51]:
```
posttax_nat_negative_thr.percentile.value_counts(dropna=False)
```

[51]:
```
p0p1    496
p1p2    179
p2p3     18
p3p4      5
p4p5      1
Name: percentile, dtype: int64
```

[52]:
```
posttax_nat_negative_thr.country.value_counts(dropna=False)
```

[52]:
```
US     102
CH      86
QX      67
QY      59
QE      56
IS      45
PT      28
LV      22
DE      19
QM      18
EE      18
NL      18
BG      17
MD      16
GB      12
HR      10
DK      10
CY      10
MK      10
LU       8
ME       8
FI       7
ES       7
KS       7
SE       6
SK       5
RO       4
AL       4
PL       3
HU       3
CZ       3
BA       3
SI       2
LT       1
AT       1
IE       1
FR       1
BE       1
IT       1
Name: country, dtype: int64
```

Post-tax disposable income

[53]:
```
posttax_dis_negative_avg =␣
↪wid_posttax_dis_clean[wid_posttax_dis_clean['average'] < 0].
↪reset_index(drop=True) #keeps only average < 0
```

```
posttax_dis_negative_thr =␣
  ↪wid_posttax_dis_clean[wid_posttax_dis_clean['threshold'] < 0].
  ↪reset_index(drop=True) #keeps only threshold < 0

#This dataframe changes the negative values of average and threshold to 0
posttax_dis_positive = wid_posttax_dis_clean.copy()
posttax_dis_positive.loc[(posttax_dis_positive['average'] < 0), 'average'] = 0
posttax_dis_positive.loc[(posttax_dis_positive['threshold'] < 0), 'threshold']␣
  ↪= 0
```

[54]:
```
posttax_dis_negative_thr.percentile.value_counts(dropna=False)
```

[54]:
```
p0p1    892
p1p2    426
p2p3    153
p3p4     70
p4p5     41
p5p6     24
p6p7      9
Name: percentile, dtype: int64
```

[55]:
```
posttax_dis_negative_thr.country.value_counts(dropna=False)
```

[55]:
```
PT    120
CH    119
QX     85
HR     80
QY     75
QE     72
DE     69
MD     66
EE     62
IS     58
GR     55
LV     53
NL     53
QM     47
ES     44
ME     43
FR     38
SE     35
BG     35
DK     34
NO     32
FI     31
GB     31
SK     28
```

```
IE      23
LT      22
PL      22
HU      20
MK      19
RO      19
BE      19
SI      17
KS      17
CY      14
BA      14
LU      13
MT       9
AL       9
CZ       6
RS       4
IT       2
AT       1
Name: country, dtype: int64
```

[56]:
```
posttax_dis_negative_thr.percentile.value_counts(dropna=False)
```

[56]:
```
p0p1    892
p1p2    426
p2p3    153
p3p4     70
p4p5     41
p5p6     24
p6p7      9
Name: percentile, dtype: int64
```

[57]:
```
posttax_dis_negative_thr.country.value_counts(dropna=False)
```

[57]:
```
PT     120
CH     119
QX      85
HR      80
QY      75
QE      72
DE      69
MD      66
EE      62
IS      58
GR      55
LV      53
NL      53
QM      47
```

```
ES     44
ME     43
FR     38
SE     35
BG     35
DK     34
NO     32
FI     31
GB     31
SK     28
IE     23
LT     22
PL     22
HU     20
MK     19
RO     19
BE     19
SI     17
KS     17
CY     14
BA     14
LU     13
MT      9
AL      9
CZ      6
RS      4
IT      2
AT      1
Name: country, dtype: int64
```

1.3.4 Total sum of shares equalling 1

The shares are all part of a total which have to sum 1 (if the percentile brackets represent the entire population analysed). Four different checks can be done here, playing with the tenths, hundreds and thousands of percentile at the 1%: - The share of the percentiles p0p1 to p99p100 should sum 1. - The share of the percentiles p0p1 to p98p99 and p99p99.1 to p99.9p100 should sum 1. - The share of the percentiles p0p1 to p98p99.9 and p99.9p99.91 to p99.99p100 should sum 1. - The share of the percentiles p0p1 to p98p99.99 and p99.99p99.991 to p99.999p100 should sum 1.

Consequentially, four different lists of percentiles are generated to apply them to the "clean" datasets:

[58]:
```
file = Path('Percentile names.xlsx')
percentiles = pd.read_excel(file, sheet_name='percentiles')
percentiles_list = percentiles['pXpY'].to_list()

tenths = pd.read_excel(file, sheet_name='tenths')
tenths_list = tenths['pXpY'].to_list()
```

```
hundreds = pd.read_excel(file, sheet_name='hundreds')
hundreds_list = hundreds['pXpY'].to_list()

thousands = pd.read_excel(file, sheet_name='thousands')
thousands_list = thousands['pXpY'].to_list()
```

The three following tables show the descriptive statistics for these four different checks. Overall, in the pretax and both post-tax distributions the median sum of the shares is always 1, the minimum value is 0.998600 and the maximum value is 1.001200. This means **the most "extreme" values only differ in 0.1% or 0.2% to 1, which should not be a concern.**

[59]:
```
#Generates the four different distributions:
wid_pretax_percentiles = wid_pretax_clean[wid_pretax_clean['percentile'].
 ↪isin(percentiles_list)].reset_index(drop=True)
wid_pretax_tenths = wid_pretax_clean[wid_pretax_clean['percentile'].
 ↪isin(tenths_list)].reset_index(drop=True)
wid_pretax_hundreds = wid_pretax_clean[wid_pretax_clean['percentile'].
 ↪isin(hundreds_list)].reset_index(drop=True)
wid_pretax_thousands = wid_pretax_clean[wid_pretax_clean['percentile'].
 ↪isin(thousands_list)].reset_index(drop=True)

#Grouping the sum by country and year
wid_pretax_percentiles_shares = wid_pretax_percentiles.groupby(['country',␣
 ↪'year', 'country_year']).sum().reset_index()
wid_pretax_percentiles_shares.rename(columns={"share": "share_percentiles"},␣
 ↪inplace=True)

wid_pretax_tenths_shares = wid_pretax_tenths.groupby(['country', 'year',␣
 ↪'country_year']).sum().reset_index()
wid_pretax_tenths_shares.rename(columns={"share": "share_tenths"}, inplace=True)

wid_pretax_hundreds_shares = wid_pretax_hundreds.groupby(['country', 'year',␣
 ↪'country_year']).sum().reset_index()
wid_pretax_hundreds_shares.rename(columns={"share": "share_hundreds"},␣
 ↪inplace=True)

wid_pretax_thousands_shares = wid_pretax_thousands.groupby(['country', 'year',␣
 ↪'country_year']).sum().reset_index()
wid_pretax_thousands_shares.rename(columns={"share": "share_thousands"},␣
 ↪inplace=True)

#Merging the results to show the results in one table
pretax_shares_check = pd.merge(wid_pretax_percentiles_shares,␣
 ↪wid_pretax_tenths_shares[['country_year', 'share_tenths']],␣
 ↪on='country_year', validate='one_to_one')
```

```
pretax_shares_check = pd.merge(pretax_shares_check,␣
↪wid_pretax_hundreds_shares[['country_year', 'share_hundreds']],␣
↪on='country_year', validate='one_to_one')
pretax_shares_check = pd.merge(pretax_shares_check,␣
↪wid_pretax_thousands_shares[['country_year', 'share_thousands']],␣
↪on='country_year', validate='one_to_one')

pretax_shares_check = pretax_shares_check[['country', 'year', 'country_year',␣
↪'share_percentiles', 'share_tenths', 'share_hundreds', 'share_thousands']]

pretax_shares_check.describe()
```

[59]:

```
              year  share_percentiles  share_tenths  share_hundreds  \
count  5603.000000        5603.000000   5603.000000     5603.000000
mean   1997.644833           1.000001      1.000001        0.999999
std      17.612145           0.000291      0.000305        0.000317
min    1900.000000           0.998700      0.998700        0.998700
25%    1989.000000           0.999800      0.999800        0.999800
50%    2001.000000           1.000000      1.000000        1.000000
75%    2010.000000           1.000200      1.000200        1.000200
max    2021.000000           1.001100      1.001000        1.001200

       share_thousands
count      5603.000000
mean          0.999999
std           0.000328
min           0.998600
25%           0.999800
50%           1.000000
75%           1.000200
max           1.001200
```

[60]:

```
#Generates the four different distributions:
wid_posttax_nat_percentiles =␣
↪wid_posttax_nat_clean[wid_posttax_nat_clean['percentile'].
↪isin(percentiles_list)].reset_index(drop=True)
wid_posttax_nat_tenths =␣
↪wid_posttax_nat_clean[wid_posttax_nat_clean['percentile'].isin(tenths_list)].
↪reset_index(drop=True)
wid_posttax_nat_hundreds =␣
↪wid_posttax_nat_clean[wid_posttax_nat_clean['percentile'].
↪isin(hundreds_list)].reset_index(drop=True)
wid_posttax_nat_thousands =␣
↪wid_posttax_nat_clean[wid_posttax_nat_clean['percentile'].
↪isin(thousands_list)].reset_index(drop=True)
```

```
#Grouping the sum by country and year
wid_posttax_nat_percentiles_shares = wid_posttax_nat_percentiles.
 ↪groupby(['country', 'year', 'country_year']).sum().reset_index()
wid_posttax_nat_percentiles_shares.rename(columns={"share":␣
 ↪"share_percentiles"}, inplace=True)

wid_posttax_nat_tenths_shares = wid_posttax_nat_tenths.groupby(['country',␣
 ↪'year', 'country_year']).sum().reset_index()
wid_posttax_nat_tenths_shares.rename(columns={"share": "share_tenths"},␣
 ↪inplace=True)

wid_posttax_nat_hundreds_shares = wid_posttax_nat_hundreds.groupby(['country',␣
 ↪'year', 'country_year']).sum().reset_index()
wid_posttax_nat_hundreds_shares.rename(columns={"share": "share_hundreds"},␣
 ↪inplace=True)

wid_posttax_nat_thousands_shares = wid_posttax_nat_thousands.
 ↪groupby(['country', 'year', 'country_year']).sum().reset_index()
wid_posttax_nat_thousands_shares.rename(columns={"share": "share_thousands"},␣
 ↪inplace=True)

#Merging the results to show the results in one table
posttax_nat_shares_check = pd.merge(wid_posttax_nat_percentiles_shares,␣
 ↪wid_posttax_nat_tenths_shares[['country_year', 'share_tenths']],␣
 ↪on='country_year', validate='one_to_one')
posttax_nat_shares_check = pd.merge(posttax_nat_shares_check,␣
 ↪wid_posttax_nat_hundreds_shares[['country_year', 'share_hundreds']],␣
 ↪on='country_year', validate='one_to_one')
posttax_nat_shares_check = pd.merge(posttax_nat_shares_check,␣
 ↪wid_posttax_nat_thousands_shares[['country_year', 'share_thousands']],␣
 ↪on='country_year', validate='one_to_one')

posttax_nat_shares_check = posttax_nat_shares_check[['country', 'year',␣
 ↪'country_year', 'share_percentiles', 'share_tenths', 'share_hundreds',␣
 ↪'share_thousands']]

posttax_nat_shares_check.describe()
```

[60]:

```
              year  share_percentiles  share_tenths  share_hundreds  \
count  1469.000000        1469.000000   1469.000000     1469.000000
mean   1998.041525           0.999997      0.999997        0.999996
std      16.254500           0.000307      0.000318        0.000331
min    1913.000000           0.998800      0.998800        0.998800
25%    1989.000000           0.999800      0.999800        0.999800
50%    2001.000000           1.000000      1.000000        1.000000
```

```
75%    2010.000000          1.000200      1.000200        1.000200
max    2020.000000          1.000900      1.000900        1.001000

       share_thousands
count      1469.000000
mean          0.999999
std           0.000341
min           0.998700
25%           0.999800
50%           1.000000
75%           1.000200
max           1.001000
```

[61]:

```
#Generates the four different distributions:
wid_posttax_dis_percentiles =␣
 ↪wid_posttax_dis_clean[wid_posttax_dis_clean['percentile'].
 ↪isin(percentiles_list)].reset_index(drop=True)
wid_posttax_dis_tenths =␣
 ↪wid_posttax_dis_clean[wid_posttax_dis_clean['percentile'].isin(tenths_list)].
 ↪reset_index(drop=True)
wid_posttax_dis_hundreds =␣
 ↪wid_posttax_dis_clean[wid_posttax_dis_clean['percentile'].
 ↪isin(hundreds_list)].reset_index(drop=True)
wid_posttax_dis_thousands =␣
 ↪wid_posttax_dis_clean[wid_posttax_dis_clean['percentile'].
 ↪isin(thousands_list)].reset_index(drop=True)


#Grouping the sum by country and year
wid_posttax_dis_percentiles_shares = wid_posttax_dis_percentiles.
 ↪groupby(['country', 'year', 'country_year']).sum().reset_index()
wid_posttax_dis_percentiles_shares.rename(columns={"share":␣
 ↪"share_percentiles"}, inplace=True)

wid_posttax_dis_tenths_shares = wid_posttax_dis_tenths.groupby(['country',␣
 ↪'year', 'country_year']).sum().reset_index()
wid_posttax_dis_tenths_shares.rename(columns={"share": "share_tenths"},␣
 ↪inplace=True)

wid_posttax_dis_hundreds_shares = wid_posttax_dis_hundreds.groupby(['country',␣
 ↪'year', 'country_year']).sum().reset_index()
wid_posttax_dis_hundreds_shares.rename(columns={"share": "share_hundreds"},␣
 ↪inplace=True)

wid_posttax_dis_thousands_shares = wid_posttax_dis_thousands.
 ↪groupby(['country', 'year', 'country_year']).sum().reset_index()
```

```
wid_posttax_dis_thousands_shares.rename(columns={"share": "share_thousands"},␣
↪inplace=True)

#Merging the results to show the results in one table
posttax_dis_shares_check = pd.merge(wid_posttax_dis_percentiles_shares,␣
↪wid_posttax_dis_tenths_shares[['country_year', 'share_tenths']],␣
↪on='country_year', validate='one_to_one')
posttax_dis_shares_check = pd.merge(posttax_dis_shares_check,␣
↪wid_posttax_dis_hundreds_shares[['country_year', 'share_hundreds']],␣
↪on='country_year', validate='one_to_one')
posttax_dis_shares_check = pd.merge(posttax_dis_shares_check,␣
↪wid_posttax_dis_thousands_shares[['country_year', 'share_thousands']],␣
↪on='country_year', validate='one_to_one')

posttax_dis_shares_check = posttax_dis_shares_check[['country', 'year',␣
↪'country_year', 'share_percentiles', 'share_tenths', 'share_hundreds',␣
↪'share_thousands']]

posttax_dis_shares_check.describe()
```

[61]:

```
                year  share_percentiles  share_tenths  share_hundreds  \
count  1363.000000        1363.000000   1363.000000     1363.000000
mean   2000.572267           0.999992      0.999987        0.999991
std      11.094231           0.000296      0.000317        0.000325
min    1970.000000           0.999000      0.999000        0.999000
25%    1991.000000           0.999800      0.999800        0.999800
50%    2002.000000           1.000000      1.000000        1.000000
75%    2010.000000           1.000200      1.000200        1.000200
max    2020.000000           1.001100      1.001100        1.001200

       share_thousands
count      1363.000000
mean          0.999991
std           0.000337
min           0.999000
25%           0.999800
50%           1.000000
75%           1.000200
max           1.001200
```

1.3.5 Averages between thresholds

The purpose of this check is to analyse if each bracker's average is between the same bracket's threshold and the following threshold. It should always be the case, because the threshold is defined as the lower limit of each percentile.

Pretax income distribution Only a **76.7%** of the (non-null) observations follow this requirement:

[63]:
```
excl_list = ['p99p100', 'p99.9p100', 'p99.99p100'] #these quantiles are excluded to get a continous distribution

pretax_avg_thr = wid_pretax_clean[~wid_pretax_clean['percentile'].isin(excl_list)].reset_index(drop=True) #dataframe without the list
pretax_avg_thr['threshold_next'] = pretax_avg_thr['threshold'].shift(-1) #threshold from the next row is brought
pretax_avg_thr.loc[(pretax_avg_thr['percentile'] == 'p99.999p100'), 'threshold_next'] = pretax_avg_thr.loc[(pretax_avg_thr['percentile'] == 'p99.999p100'), 'average'] #threshold_next for the last quantile is average (no available value in the next row)

pretax_avg_thr['avg_thr_check'] = ((pretax_avg_thr['average'] >= pretax_avg_thr['threshold']) & (pretax_avg_thr['average'] <= pretax_avg_thr['threshold_next'])) #check is true if average is between thresholds
pretax_avg_thr.loc[(pretax_avg_thr['threshold'].isnull()) | (pretax_avg_thr['average'].isnull()) | (pretax_avg_thr['threshold_next'].isnull()), 'avg_thr_check'] = np.nan #check is null if one of the values is null

pretax_avg_thr_false = pretax_avg_thr[pretax_avg_thr['avg_thr_check'] == False].reset_index(drop=True) #dataframe with all the false checks
```

[64]:
```
pretax_avg_thr.avg_thr_check.value_counts(normalize=True)
```

[64]:
```
True     0.766692
False    0.233308
Name: avg_thr_check, dtype: float64
```

[65]:
```
pretax_avg_thr_false.country_year.value_counts()
```

[65]:
```
UY2010    126
BR2002    126
BR2014    126
UY2011    126
UY2017    126
         …
BI1998      1
HR1998      1
CD2004      1
PG2009      1
HR1988      1
Name: country_year, Length: 1909, dtype: int64
```

[66]:
```
#pretax_avg_thr.to_csv('avgthr.csv')
```

Post-tax national income distribution The **99.99%** of the (non-null) observations follow this requirement:

[67]:
```
excl_list = ['p99p100', 'p99.9p100', 'p99.99p100'] #these quantiles are
 ↪excluded to get a continous distribution

posttax_nat_avg_thr =
 ↪wid_posttax_nat_clean[~wid_posttax_nat_clean['percentile'].isin(excl_list)].
 ↪reset_index(drop=True) #dataframe without the list
posttax_nat_avg_thr['threshold_next'] = posttax_nat_avg_thr['threshold'].
 ↪shift(-1) #threshold from the next row is brought
posttax_nat_avg_thr.loc[(posttax_nat_avg_thr['percentile'] == 'p99.999p100'),
 ↪'threshold_next'] = posttax_nat_avg_thr.
 ↪loc[(posttax_nat_avg_thr['percentile'] == 'p99.999p100'), 'average']
 ↪#threshold_next for the last quantile is average (no available value in the
 ↪next row)

posttax_nat_avg_thr['avg_thr_check'] = ((posttax_nat_avg_thr['average'] >=
 ↪posttax_nat_avg_thr['threshold']) & (posttax_nat_avg_thr['average'] <=
 ↪posttax_nat_avg_thr['threshold_next'])) #check is true if average is between
 ↪thresholds
posttax_nat_avg_thr.loc[(posttax_nat_avg_thr['threshold'].isnull()) |
 ↪(posttax_nat_avg_thr['average'].isnull()) |
 ↪(posttax_nat_avg_thr['threshold_next'].isnull()), 'avg_thr_check'] = np.nan
 ↪#check is null if one of the values is null

posttax_nat_avg_thr_false =
 ↪posttax_nat_avg_thr[posttax_nat_avg_thr['avg_thr_check'] == False].
 ↪reset_index(drop=True)
```

[68]:
```
posttax_nat_avg_thr.avg_thr_check.value_counts(normalize=True)
```

[68]:
```
True     0.999855
False    0.000145
Name: avg_thr_check, dtype: float64
```

[69]:
```
posttax_nat_avg_thr_false.country_year.value_counts()
```

[69]:
```
SK2016    2
FR1979    2
SK2005    2
LU2017    2
LU2011    1
SK2015    1
```

```
SK2014     1
SK2010     1
SK2006     1
SE2005     1
BE2004     1
BE2012     1
IT2014     1
IT2013     1
IT2012     1
BE2017     1
BE2015     1
BE2014     1
BE2013     1
LU2008     1
Name: country_year, dtype: int64
```

Post-national disposable income distribution The **86.5%** of the (non-null) observations follow this requirement:

[70]:
```
excl_list = ['p99p100', 'p99.9p100', 'p99.99p100'] #these quantiles are excluded to get a continous distribution

posttax_dis_avg_thr = wid_posttax_dis_clean[~wid_posttax_dis_clean['percentile'].isin(excl_list)].reset_index(drop=True) #dataframe without the list
posttax_dis_avg_thr['threshold_next'] = posttax_dis_avg_thr['threshold'].shift(-1) #threshold from the next row is brought
posttax_dis_avg_thr.loc[(posttax_dis_avg_thr['percentile'] == 'p99.999p100'), 'threshold_next'] = posttax_dis_avg_thr.loc[(posttax_dis_avg_thr['percentile'] == 'p99.999p100'), 'average'] #threshold_next for the last quantile is average (no available value in the next row)

posttax_dis_avg_thr['avg_thr_check'] = ((posttax_dis_avg_thr['average'] >= posttax_dis_avg_thr['threshold']) & (posttax_dis_avg_thr['average'] <= posttax_dis_avg_thr['threshold_next'])) #check is true if average is between thresholds
posttax_dis_avg_thr.loc[(posttax_dis_avg_thr['threshold'].isnull() | (posttax_dis_avg_thr['average'].isnull()) | (posttax_dis_avg_thr['threshold_next'].isnull()), 'avg_thr_check'] = np.nan #check is null if one of the values is null

posttax_dis_avg_thr_false = posttax_dis_avg_thr[posttax_dis_avg_thr['avg_thr_check'] == False].reset_index(drop=True)
```

[71]:
```
posttax_dis_avg_thr.avg_thr_check.value_counts(normalize=True)
```

[71]:
```
True     0.864592
False    0.135408
Name: avg_thr_check, dtype: float64
```

[72]:
```
posttax_dis_avg_thr_false.country_year.value_counts()
```

[72]:
```
QM2000    125
QM2013    125
QM1990    125
QM2007    125
QM1989    125
         …
DK2001      1
DK2002      1
DK2008      1
DK2017      1
SK2000      1
Name: country_year, Length: 229, dtype: int64
```

1.3.6 Comparability of values between periods

This check is to avoid having big jumps or drops between periods for certain percentiles.

[62]:
```
wid_pretax_clean[wid_pretax_clean['percentile']=='p50p51']
```

[62]:
```
       country  year percentile    p      threshold        average   share  \
50          AE  1998     p50p51  0.5   62199.207252   64027.866295  0.0041
180         AE  2009     p50p51  0.5   24589.520723   25280.989597  0.0035
310         AE  2013     p50p51  0.5   38323.344699   39099.204911  0.0052
440         AE  2014     p50p51  0.5   39391.356474   40189.676035  0.0051
570         AE  2018     p50p51  0.5   48598.654079   49490.771800  0.0059
…        …   …        …   …            …            …       …
727790      ZW  1991     p50p51  0.5  172769.499022  176172.418340  0.0036
727920      ZW  1996     p50p51  0.5  200058.803517  204327.407858  0.0042
728050      ZW  2011     p50p51  0.5  165799.888765  169682.845260  0.0050
728180      ZW  2017     p50p51  0.5  178844.984874  183319.039243  0.0046
728310      ZW  2019     p50p51  0.5  148015.957821  151078.094873  0.0039

        inv_paretolorenz  age pop country_year
50                   NaN  992   j       AE1998
180                  NaN  992   j       AE2009
310                  NaN  992   j       AE2013
440                  NaN  992   j       AE2014
570                  NaN  992   j       AE2018
…                    …   …  ..           …
```

```
727790              NaN  992   j       ZW1991
727920              NaN  992   j       ZW1996
728050              NaN  992   j       ZW2011
728180              NaN  992   j       ZW2017
728310              NaN  992   j       ZW2019

[5603 rows x 11 columns]
```

```
[ ]:
```

```
[ ]:
```