Apode

Release 0.1.0

Néstor Grión and Sofía Sappia

Dec 10, 2020

CONTENTS:

Apode is a package that contains a set of indicators that are applied in economic analysis. It contains measures of poverty, inequality, polarization, welfare and concentration.

CHAPTER

ONE

REQUIREMENTS

You need Python 3.8 to run Apode.

CHAPTER

TWO

CODE REPOSITORY & ISSUES

https://github.com/ngrion/apode

CHAPTER

THREE

BASIC INSTALL

Execute

```
$ pip install apode
```

CHAPTER

FOUR

DEVELOPMENT INSTALL

Clone this repo and install with pip

```
$ git clone https://github.com/ngrion/apode.git
$ cd pycf3
$ pip install -e .
```

CHAPTER

FIVE

FEATURES

Objects are created using:

```
>>> ad = ApodeData(DataFrame, income_column)
```

Where income_column is the name of the desired analysis column in the dataframe.

Methods that calculate indicators:

```
>>> ad.poverty(method,*args)
>>> ad.ineq(method,*args)
>>> ad.welfare(method,*args)
>>> ad.polarization(method,*args)
>>> ad.concentration(method,*args)
```

Graphical representations:

```
>>> ad.plot.hist()
>>> ad.plot.tip(**kwargs)
>>> ad.plot.lorenz(**kwargs)
>>> ad.plot.pen(**kwargs)
```

For examples on how to use apode, please refer to the Tutorial.

CHAPTER

SIX

CONTRIBUTORS

Thanks to the following people who have contributed to this project:

- @ngrion
- @sofisappia

CHAPTER

SEVEN

SUPPORT

If you want to contact me you can reach me at ngrion@gmail.com. If you have issues please report them as a issue here.

CHAPTER

EIGHT

LICENSE

Distributed under the MIT License. See LICENSE for more information.

CHAPTER

NINE

CONTENTS:

9.1 Requirements

You need Python 3.8 or later to run Apode. You can have multiple Python versions (3.x) installed on the same system without problems.

9.2 Installation

The easiest way to install is using pip:

```
$ pip install apode
```

This will install the latest stable version on PIPy.

If you want to use the latest development version from github, unpack or clone the repo on your local machine, change the directory to where setup.py is, and install using setuptools:

```
$ python setup.py install
```

or pip:

```
$ pip install -e .
```

9.3 Licence

MIT License

OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

9.4 Apode Tutorial

This tutorial shows the current functionalities of the Apode package. Apode contains various methods to calculate measures and generate graphs on the following topics:

- Poverty
- Inequality
- Welfare
- Polarization
- Concentration

9.4.1 Getting Started

ApodeData class

The objects are created as:

```
ad = ApodeData(DataFrame,income_column)
```

Where income_column is the name of the column of interest for the analysis in the dataframe.

Methods that calculate indicators:

```
ad.poverty(method,*args)
ad.ineq(method,*args)
ad.welfare(method,*args)
ad.polarization(method,*args)
ad.concentration(method,*args)
```

Métodos that generate graphs:

```
ad.plot.hist()
ad.plot.tip(**kwargs)
ad.plot.lorenz(**kwargs)
ad.plot.pen(**kwargs)
```

For an introduction to the Python language see Beginner Guide.

Data Creation and Description

- Data can be generated manually or by means of a simulation. They are contained in a DataFrame object.
- Other categoric variables that allow for the indicators to be applied by groups (groupby) might be available.

[1]:
```
import numpy as np
import pandas as pd

from apode import ApodeData
from apode import datasets
```

Manual data loading

An object can be generated from a DataFrame or from a valid argument in the DataFrame method.

[2]:
```
x = [23, 10, 12, 21, 4, 8, 19, 15, 11, 9]
df1 = pd.DataFrame({'x':x})
ad1 = ApodeData(df1, income_column="x")
ad1
```

[2]:
```
    x
0  23
1  10
2  12
3  21
4   4
5   8
6  19
7  15
8  11
9   9
ApodeData(income_column='x') - 10 rows x              1 columns
```

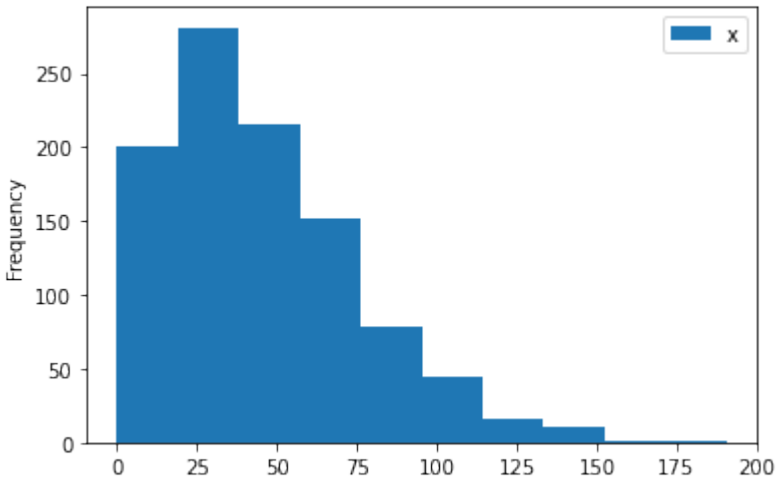
Data simulation

The module *datasets* contains some distribution examples commonly used to model the income distribution. For example:

[3]:
```
# Generate data
n = 1000 # observations
seed = 12345
ad2 = datasets.make_weibull(seed=seed,size=n)
ad2.describe()
```

[3]:
```
                 x
count  1000.000000
mean     45.408376
std      30.207372
min       0.112876
25%      22.363891
50%      39.404779
75%      62.690946
max     190.596705
```

[4]:
```
# Plotting the distribution
ad2.plot();
```



Other distributions are: uniform, lognormal, exponential, pareto, chisquare and gamma. For help type:

[5]:
```
help(datasets.make_weibull)
```

```
Help on function make_weibull in module apode.datasets:

make_weibull(seed=None, size=100, a=1.5, c=50, nbin=None)
    Weibull Distribution.

    Parameters
    ----------
    seed: int, optional(default=None)

    size: int, optional(default=100)

    a: float, optional(default=1.5)

    c: float, optional(default=50)

    nbin: int, optional(default=None)

    Return
    ------
    out: float array
        Array of random numbers.
```

9.4.2 Poverty

The following poverty measures are implemented:

- `headcount`: Headcount Index
- `gap`: Poverty gap Index
- `severity`: Poverty Severity Index
- `fgt`: Foster–Greer–Thorbecke Indices
- `sen`: Sen Index

- `sst`: Sen-Shorrocks-Thon Index
- `watts`: Watts Index
- `cuh`: Clark, Ulph and Hemming Indices
- `takayama`: Takayama Index
- `kakwani`: Kakwani Indices
- `thon`: Thon Index
- `bd`: Blackorby and Donaldson Indices
- `hagenaars`: Hagenaars Index
- `chakravarty`: Chakravarty Indices

Also the TIP curve, which allows for a graphic comparison of poverty amongst different distributions.

Numerical measures

All the methods require the poverty line (*pline*) as argument for get a absolute measure of poverty.

[6]:
```
pline = 50 # Poverty line
p = ad2.poverty('headcount',pline=pline)
p
```

[6]:
```
0.626
```

If the argument is omitted 0.5*median is used. Other options for a relative measure of poverty are:

[7]:
```
# pline = factor*stat  [stat: mean, median, quantile_q]
p1 = ad2.poverty('headcount')  # pline= 0.5*median(y)
p2 = ad2.poverty('headcount', pline='median', factor=0.5)
p3 = ad2.poverty('headcount', pline='quantile', q=0.5, factor=0.5)
p4 = ad2.poverty('headcount', pline='mean', factor=0.5)
p1, p2, p3, p4
```

[7]:
```
(0.214, 0.214, 0.214, 0.256)
```

Some methods require an additional parameter, *alpha*. In some cases, a default value is set for it. The summary function shows the result of various methods:

[17]:
```
df_p = poverty_summary(ad2)
df_p
```

[17]:

	method	measure
0	headcount	0.214000
1	gap	0.088038
2	severity	0.052488
3	fgt(1.5)	0.066208
4	sen	0.134042
5	sst	0.164471
6	watts	0.155405
7	cuh(0)	0.917773
8	cuh(0.5)	0.112144
9	takayama	0.083811
10	kakwani	0.139643
11	thon	0.164395
12	bd(1.0)	0.110478

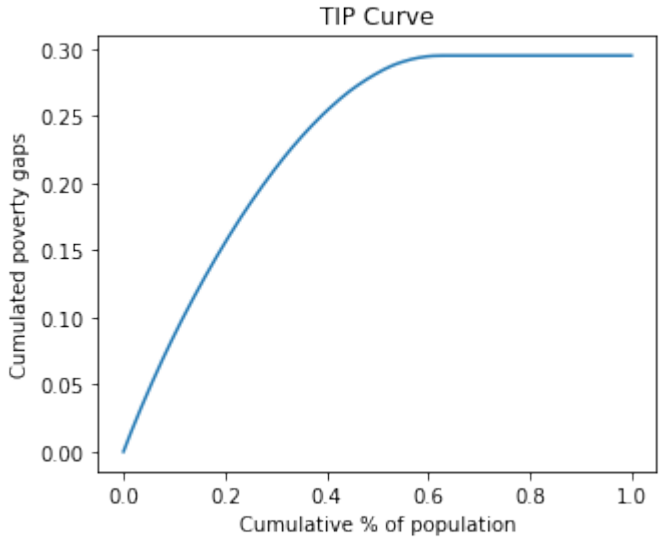
(continues on next page)

(continued from previous page)

```
13           bd(2.0)  0.158095
14         hagenaars  0.052136
15  chakravarty(0.5)  0.056072
```

Graph measures

[9]:
```
# TIP curve
ad2.plot.tip(pline=pline);
```



Exercise: Comparison of poverty measures

In this exercise some properties of three poverty measures are compared: headscount, poverty gap and severity index (they are part of the *FGT* class). The exercise compares three distributions:

[21]:
```
x1 = np.array([5,20,35,60])
x2 = x1 +  np.array([10, 10, 10, 0])
x3 = x2 +  np.array([10,0,-10, 0])
dfe = pd.DataFrame({'x1':x1,'x2':x2,'x3':x3})
dfe
```

[21]:

	x1	x2	x3
0	5	15	25
1	20	30	30
2	35	45	35
3	60	60	60

[11]:
```
pline = 50
ade1 = ApodeData(dfe,income_column='x1')
ade2 = ApodeData(dfe,income_column='x2')
ade3 = ApodeData(dfe,income_column='x3')

# Headcount
ade1.poverty('headcount',pline=pline), ade2.poverty('headcount',pline=pline), ade3.
↪poverty('headcount',pline=pline)
```

[11]:
```
(0.75, 0.75, 0.75)
```

The greatest virtues of the *headcount index* are that it is simple to construct and easy to understand. However, the measure does not take the intensity of poverty into account (the three indices are equal).

[12]:
```
# Poverty Gap
ade1.poverty('gap',pline=pline), ade2.poverty('gap',pline=pline), ade3.poverty('gap',
↪pline=pline)
```

[12]:
```
(0.45, 0.30000000000000004, 0.3)
```

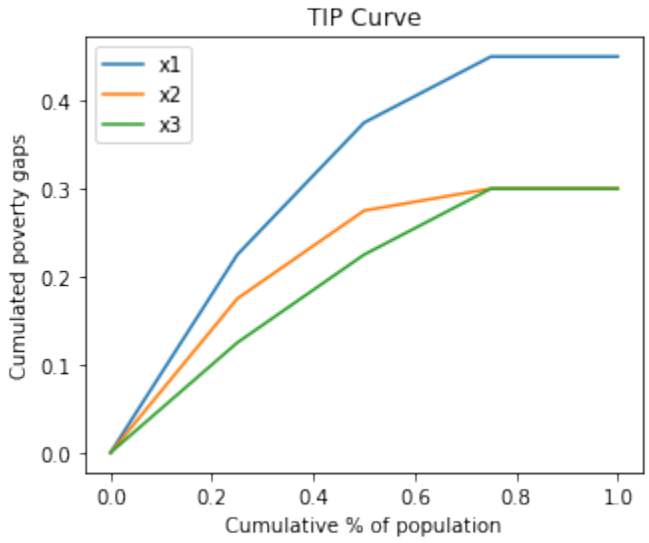
The *poverty gap* index gives an idea of the cost to eliminate poverty (in relation to the poverty line). But the previous indices are unsatisfactory because they violate the *transfer principle* which states that transfers from a richer person to a poorer person should improve the welfare measure (x2 versus x3). This property is captured by the *severity* index.

[13]:
```
# Severity
ade1.poverty('severity',pline=pline), ade2.poverty('severity',pline=pline), ade3.
↪poverty('severity',pline=pline)
```

[13]:
```
(0.315, 0.16499999999999998, 0.125)
```

TIP curves are cumulative poverty gap curves used for representing the three different aspects of poverty: incidence, intensity and inequality (severity). Non-intersection of TIP curves is recognized as a criterion to compare income distributions in terms of poverty.

[20]:
```
ax = ade1.plot.tip(pline=pline)
ade2.plot.tip(ax=ax, pline=pline)
ade3.plot.tip(ax=ax, pline=pline)
ax.legend(['x1','x2','x3']);
```



Exercise: Contribution of each subgroup to aggregate poverty

Another convenient feature of the *FGT* class of poverty measures is that they can be disaggregated for population subgroups and the contribution of each subgroup to aggregate poverty can be calculated. For instance:

[36]:
```
x = [23, 10, 12, 21, 4, 8, 19, 15, 5, 7]
y = [10,10,20,10,10,20,20,20,10,10]
region = ['urban','urban','rural','urban','urban','rural','rural','rural','urban',
↪'urban']
dfa = pd.DataFrame({'x':x,'region':region})
ada1 = ApodeData(dfa,income_column='x')
ada1
```

[36]:
```
    x region
0  23  urban
1  10  urban
2  12  rural
3  21  urban
4   4  urban
5   8  rural
6  19  rural
7  15  rural
8   5  urban
9   7  urban
ApodeData(income_column='x') - 10 rows x             2 columns
```

[39]:
```
# group calculation according to variable "y"
pline = 11
pg = poverty_groupby(ada1,'headcount',group_column='region', pline=pline)
pg
```

[39]:
```
       x_measure  x_weight
rural   0.250000         4
urban   0.666667         6
```

[40]:
```
# simple calculation
ps = ada1.poverty('headcount',pline=pline)
ps
```

[40]:
```
0.5
```

[41]:
```
# If the indicator is decomposable, the same result is attained:
pg_p = sum(pg['x_measure']*pg['x_weight']/sum(pg['x_weight']))
pg_p
```

[41]:
```
0.5
```

9.4.3 Inequality

The following inequality measures are implemented:

- `gini`: Gini Index
- `entropy`: Generalized Entropy Index
- `atkinson`: Atkinson Index
- `rrange`: Relative Range

- `rad`: Relative average deviation
- `cv`: Coefficient of variation
- `sdlog`: Standard deviation of log
- `merhan`: Merhan index
- `piesch`: Piesch Index
- `bonferroni`: Bonferroni Indices
- `kolm`: Kolm Index

Also the Lorenz and Pen curves, which allows for a graphic comparison of inequality among different distributions.

Numerical measures

[18]:
```
# Evaluate an inequality method
q = ad2.inequality('gini')
q
```

[18]:
```
0.3652184907296814
```

The summary function shows the result of various methods:

[19]:
```
df_ineq = inequality_summary(ad2)
df_ineq
```

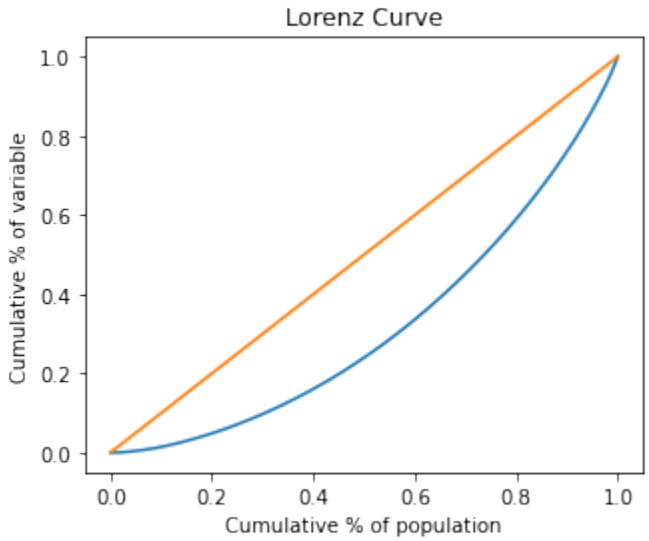
[19]:

	method	measure
0	rrange	4.194905
1	rad	0.264575
2	cv	0.664905
3	sdlog	0.916263
4	gini	0.365218
5	merhan	0.512500
6	piesch	0.288587
7	bonferroni	0.510418
8	kolm(0.5)	36.566749
9	ratio(0.05)	0.032061
10	ratio(0.2)	0.118795
11	entropy(0)	0.281897
12	entropy(1)	0.217586
13	entropy(2)	0.221049
14	atkinson(0.5)	0.114579
15	atkinson(1.0)	0.245649
16	atkinson(2.0)	0.631276

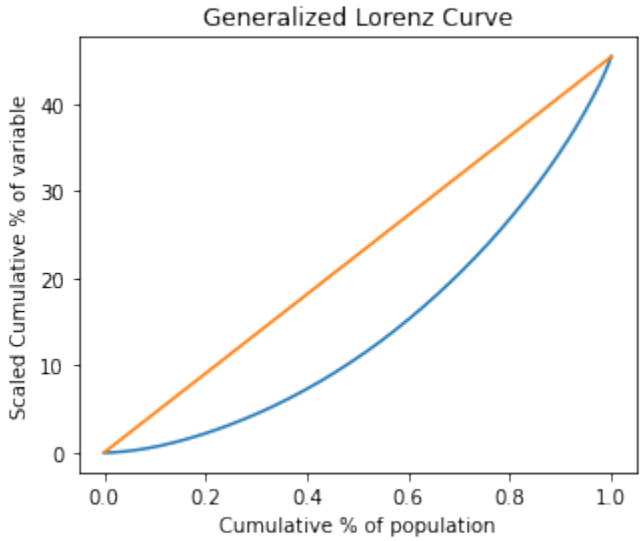
Graph measures

The relative, generalized and absolute Lorenz Curves are implemented. Alse de Pen Parade curve.

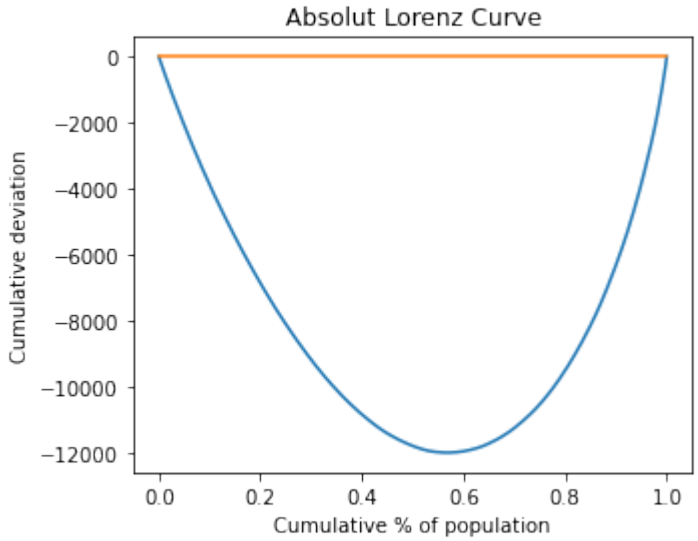
[20]:
```
# Lorenz Curves
ad2.plot.lorenz();
```



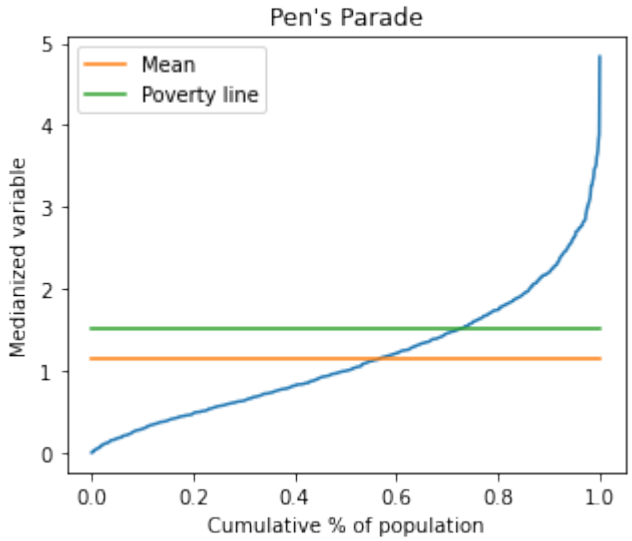
[21]:
```
# Generalized Lorenz Curve
ad2.plot.lorenz(alpha='g');
```



[22]:
```
# Absolute Lorenz Curve
ad2.plot.lorenz(alpha='a');
```



[23]:
```
# Pen's Parade
ad2.plot.pen(pline=60);
```



Exercise: Redistributive Effect of Fiscal Policy

Income pre and post fiscal policy:

[24]:
```
# Income pre fiscal policy:
y_pre = np.array([20, 30, 40, 60, 100])

# Fiscal policy
tax = 0.2*np.maximum(y_pre-35,0)   # tax formula
revenue = np.sum(tax)              # total revenue
transfers = revenue/len(y_pre)     # per capita transfers
```

(continues on next page)

(continued from previous page)

```
# Income post fiscal policy:
y_post = y_pre - tax + transfers
```

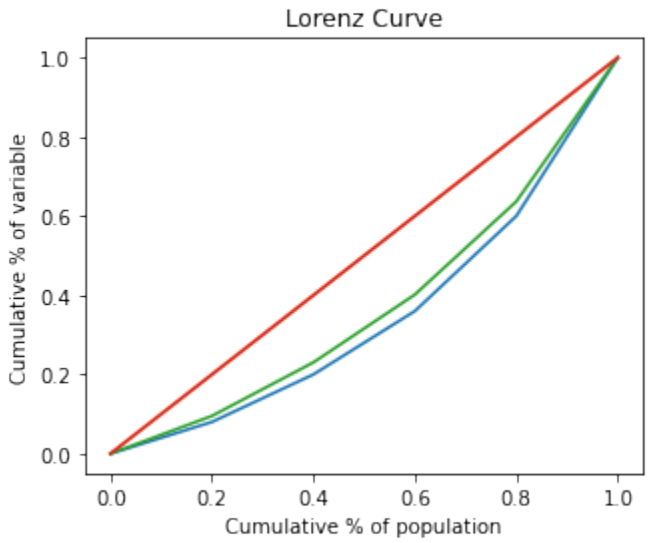
[25]:
```
# ApodeData
df_pre = pd.DataFrame({'y1':y_pre})
ad_pre = ApodeData(df_pre,income_column='y1')

df_post = pd.DataFrame({'y2':y_post})
ad_post = ApodeData(df_post,income_column='y2')
ad_post
```

[25]:
```
     y2
0  23.8
1  33.8
2  42.8
3  58.8
4  90.8
ApodeData(income_column='y2') - 5 rows x              1 columns
```

[26]:
```
# Gini
ad_pre.inequality.gini(), ad_post.inequality.gini()  # decrease inequality
```

[26]:
```
(0.30399999999999994, 0.25440000000000024)
```

[27]:
```
# Lorenz Curves
ax = ad_pre.plot.lorenz()
ad_post.plot.lorenz(ax=ax);
```



9.4.4 Welfare

The following welfare measures are implemented:

- `utilitarian`: Utilitarian utility function
- `rawlsian`: Rawlsian utility function
- `isoelastic`: Isoelastic utility function
- `sen`: Sen utility function
- `theill`: Theill utility function
- `theilt`: Theilt utility function

[28]:
```
# Evaluate a welfare method
w = ad2.welfare('sen')
w
```

[28]:
```
28.824397753868958
```

The summary function shows the result of various methods:

[29]:
```
df_wlf = welfare_summary(ad2)
df_wlf
```

[29]:
```
            method    measure
0      utilitarian  45.408376
1         rawlsian   0.112876
2              sen  28.824398
3           theill  34.253866
4           theilt  36.529160
5    isoelastic(0)  45.408376
6    isoelastic(1)   3.533799
7    isoelastic(2)  -0.059726
8  isoelastic(inf)   0.112876
```

9.4.5 Polarization

The following welfare measures are implemented:

- `ray`: Esteban and Ray index
- `wolfson`: Wolfson index

[30]:
```
# Evaluate a polarization method
p = ad2.polarization('ray')
p
```

[30]:
```
0.03316795744715791
```

The summary function shows the result of various methods:

[31]:
```
df_pz = polarization_summary(ad2)
df_pz
```

[31]:
```
    method   measure
0      ray  0.033168
1  wolfson  0.357081
```

9.4.6 Concentration

The following concentration measures are implemented:

- `herfindahl`: Herfindahl-Hirschman Index
- `rosenbluth`: Rosenbluth Index
- `concentration_ratio`: Concentration Ratio Index

[32]:
```
# Evaluate a concentration method
c = ad2.concentration('herfindahl')
c
```

[32]:
```
0.00044254144331556705
```

The summary function shows the result of various methods:

[33]:
```
df_conc = concentration_summary(ad2)
df_conc
```

[33]:
```
                   method   measure
0              herfindahl  0.000443
1        herfindahl(norm)  0.000443
2              rosenbluth  0.001575
3  concentration_ratio(1)  0.004197
4  concentration_ratio(3)  0.010900
```

9.4.7 References

Schröder, C. (2011). Cowell, F.: Measuring Inequality. London School of Economics Perspectives in Economic Analysis.

Adler, M. D., & Fleurbaey, M. (Eds.). (2016). The Oxford handbook of well-being and public policy. Oxford University Press.

Haughton, J., & Khandker, S. R. (2009). Handbook on poverty+ inequality. World Bank Publications.

Gasparini, L., Cicowiez, M., & Sosa Escudero, W. (2012). Pobreza y desigualdad en América Latina. Temas Grupo Editorial.

Araar, A., & Duclos, J. Y. (2007). DASP: Distributive analysis stata package. PEP, World Bank, UNDP and Université Laval.

Apendix

[15]:
```
# Evaluating a list of poverty methods
def poverty_summary(ad, pline=None, factor=1.0, q=None):
    import apode
    #y = ad[ad.income_column].values # ver falla
    y = ad.x.values
    pline = apode.poverty._get_pline(y, pline, factor, q)

    pov_list = [["headcount", None],
                ["gap", None],
                ["severity",None],
                ["fgt",1.5],
                ["sen",None],
```

(continues on next page)

(continued from previous page)

```
                ["sst",None],
                ["watts",None],
                ["cuh",0],
                ["cuh",0.5],
                ["takayama",None],
                ["kakwani",None],
                ["thon",None],
                ["bd",1.0],
                ["bd",2.0],
                ["hagenaars",None],
                ["chakravarty",0.5]]
    p = []
    pl = []
    for elem in pov_list:
        if elem[1]==None:
            p.append(ad.poverty(elem[0],pline=pline))
            pl.append(elem[0])
        else:
            p.append(ad.poverty(elem[0],pline=pline,alpha=elem[1]))
            pl.append(elem[0] + '(' + str(elem[1]) +')')
    df_p = pd.concat([pd.DataFrame(pl),pd.DataFrame(p)],axis=1)
    df_p.columns = ['method','measure']
    return df_p

# Evaluate a list of inequality methods
def inequality_summary(ad):
    ineq_list = [["rrange", None],
                ["rad", None],
                ["cv",None],
                ["sdlog",None],
                ["gini",None],
                ["merhan",None],
                ["piesch",None],
                ["bonferroni",None],
                ["kolm",0.5],
                ["ratio",0.05],
                ["ratio",0.2],
                ["entropy",0],
                ["entropy",1],
                ["entropy",2],
                ["atkinson",0.5],
                ["atkinson",1.0],
                ["atkinson",2.0]]
    p = []
    pl = []
    for elem in ineq_list:
        if elem[1]==None:
            p.append(ad.inequality(elem[0]))
            pl.append(elem[0])
        else:
            p.append(ad.inequality(elem[0],alpha=elem[1]))
            pl.append(elem[0] + '(' + str(elem[1]) +')')
    df_ineq = pd.concat([pd.DataFrame(pl),pd.DataFrame(p)],axis=1)
    df_ineq.columns = ['method','measure']
    return df_ineq

# Evaluate a list of welfare methods
```

(continues on next page)

(continued from previous page)

```
def welfare_summary(ad):
    wlf_list = [["utilitarian", None],
                ["rawlsian", None],
                ["sen",None],
                ["theill",None],
                ["theilt",None],
                ["isoelastic",0],
                ["isoelastic",1],
                ["isoelastic",2],
                ["isoelastic",np.Inf]]
    p = []
    pl = []
    for elem in wlf_list:
        if elem[1]==None:
            p.append(ad.welfare(elem[0]))
            pl.append(elem[0])
        else:
            p.append(ad.welfare(elem[0],alpha=elem[1]))
            pl.append(elem[0] + '(' + str(elem[1]) +')')

    df_wlf = pd.concat([pd.DataFrame(pl),pd.DataFrame(p)],axis=1)
    df_wlf.columns = ['method','measure']
    return df_wlf

# Evaluate a list of polarization methods
def polarization_summary(ad):
    pol_list = [["ray", None],
                ["wolfson", None]]
    p = []
    pl = []
    for elem in pol_list:
        if elem[1]==None:
            p.append(ad2.polarization(elem[0]))
            pl.append(elem[0])
        else:
            p.append(ad2.polarization(elem[0],alpha=elem[1]))
            pl.append(elem[0] + '(' + str(elem[1]) +')')
    df_pz = pd.concat([pd.DataFrame(pl),pd.DataFrame(p)],axis=1)
    df_pz.columns = ['method','measure']
    return df_pz

# Evaluate a list of concentration methods
def concentration_summary(ad):
    conc_list = [["herfindahl", None],
                ["herfindahl", True],
                ["rosenbluth",None],
                ["concentration_ratio",1],
                ["concentration_ratio",3]]
    p = []
    pl = []
    for elem in conc_list:
        if elem[1]==None:
            p.append(ad2.concentration(elem[0]))
            pl.append(elem[0])
        else:
            if elem[0]=="herfindahl":
                p.append(ad2.concentration(elem[0],normalized=elem[1]))  # check␣
↪keyword
```

(continues on next page)

(continued from previous page)

```
                pl.append(elem[0] + '(norm)')
            elif elem[0]=="concentration_ratio":
                p.append(ad2.concentration(elem[0],k=elem[1]))  # check keyword
                pl.append(elem[0] + '(' + str(elem[1]) +')')
            else:
                p.append(ad2.concentration(elem[0],alpha=elem[1]))
                pl.append(elem[0] + '(' + str(elem[1]) +')')
    df_conc = pd.concat([pd.DataFrame(pl),pd.DataFrame(p)],axis=1)
    df_conc.columns = ['method','measure']
    return df_conc
```

[16]:
```
# receives a dataframe and applies a measure according to the column "varg"
def poverty_groupby(ad,method,group_column,**kwargs):
    return measure_groupby(ad,method,group_column,measure='poverty',**kwargs)

def measure_groupby(ad,method,group_column,measure,**kwargs):
    a = []; b = []; c = []
    for name, group in ad.groupby(group_column):
        adi = ApodeData(group,income_column=ad.income_column)
        if measure=='poverty':
            p = adi.poverty(method,**kwargs)
        elif measure=='inequality':
            p = adi.inequality(method,**kwargs)
        elif measure=='concentration':
            p = adi.concentration(method,**kwargs)
        elif measure=='polarization':
            p = adi.polarization(method,**kwargs)
        elif measure=='welfare':
            p = adi.welfare(method,**kwargs)
        a.append(name)
        b.append(p)
        c.append(group.shape[0])
    xname = ad.income_column + "_measure"
    wname = ad.income_column + "_weight"
    return pd.DataFrame({xname: b, wname: c}, index=pd.Index(a))
```

9.5 API Module

9.5.1 Module contents

Poverty and Inequality Analysis.

Apode contains a set of measures applied in economics.

class apode.**ApodeData**(*data*, *income_column*)

Bases: object

Poverty and Inequality Analysis in Python.

Apode is a package that contains a set of indicators that are applied in economic analysis. It contains measures of: - poverty - inequality - welfare - polarization - concentration

Parameters

- **data** (*dataframe*) – Dataset to be analiced.

- **income_column** (*str*) – Column name

data, income_column

9.5.2 apode.basic module

ApodeData class for Apode.

class apode.basic.**ApodeData**(*data*, *income_column*)
Bases: object

Poverty and Inequality Analysis in Python.

Apode is a package that contains a set of indicators that are applied in economic analysis. It contains measures of: - poverty - inequality - welfare - polarization - concentration

Parameters

- **data** (*dataframe*) – Dataset to be analiced.
- **income_column** (*str*) – Column name

data, income_column

9.5.3 apode.concentration module

Concentration measures for Apode.

class apode.concentration.**ConcentrationMeasures**(*idf*)
Bases: object

Concentration Measures.

The following concentration measures are implemented:

- herfindahl : Herfindahl-Hirschman Index
- rosenbluth : Rosenbluth Index
- concentration_ratio : Concentration Ratio Index

Parameters

- **method** (*String*) – Concentration measure.
- ****kwargs** – Arbitrary keyword arguments.

concentration_ratio(*k*)
Concentration Ratio index.

The concentration ratio is calculated as the sum of the market share percentage held by the largest specified number of firms in an industry.

Parameters **k** (*int*) – The number of firms included in the concentration ratio calculation.

Returns **out** – Index measure.

Return type float

herfindahl(*normalized=True*)
Herfindahl-Hirschman index.

The Herfindahl-Hirschman index it is defined as the sum of the squares of the market shares of the firms within the industry[1].

Parameters **normalized** (*bool (default=true)*) – The normalized index ranges from 0 to 1.

Returns **out** – Index measure.

Return type float

References

rosenbluth()
Rosenbluth index.

The Rosenbluth index measures the proportion of the population that counted as poor[2].

Returns **out** – Index measure.

Return type float

References

9.5.4 apode.datasets module

Data simulation tools for Apode.

apode.datasets.**binning**(*df*, *pos=0*, *nbin=None*)
Binning function.

Agrupa valores de un dataframe en nbin categorías.

Parameters

- **df** (*DataFrame*) –
- **pos** (*int, optional (default=0)*) – Options are r: relative, 'g': generalized, 'a': absolut.
- **nbin** (*int, optional (default=None)*) –

Returns **out** – Grouped data

Return type DataFrame

apode.datasets.**make_bimodal**(*size=100*, *nbin=None*)
Bimodal Distribution.

Parameters

- **size** (*int, optional (default=100)*) –
- **nbin** (*int, optional (default=None)*) –

Returns **out**

Return type float array

[1] Hirschman, A.O (1964), "The Paternity of an Index", American Economic Review, 54 (5), 761.
[2] Rosenbluth, G. (1955). Measures of concentration, Business Concentration and Price Policy. National Bureau of Economic Research. Special Conference Series No. 5. Princeton, 57–89.

apode.datasets.**make_chisquare**(*seed=None*, *size=100*, *df=5*, *c=10*, *nbin=None*)
Chisquare Distribution.

Parameters

- **seed**(*int, optional(default=None)*) –
- **size**(*int, optional(default=100)*) –
- **df**(*float, optional(default=5)*) –
- **c**(*float, optional(default=10)*) –
- **nbin**(*int, optional(default=None)*) –

Returns **out** – Array of random numbers.

Return type float array

apode.datasets.**make_constant**(*size=100*, *nbin=None*)
Constant value Distribution.

Parameters

- **size**(*int, optional(default=100)*) –
- **nbin**(*int, optional(default=None)*) –

Returns **out**

Return type float array

apode.datasets.**make_exponential**(*seed=None*, *size=100*, *scale=1*, *c=50*, *nbin=None*)
Exponential Distribution.

Parameters

- **seed**(*int, optional(default=None)*) –
- **size**(*int, optional(default=100)*) –
- **scale**(*float, optional(default=1.0)*) –
- **c**(*float, optional(default=50)*) –
- **nbin**(*int, optional(default=None)*) –

Returns **out** – Array of random numbers.

Return type float array

apode.datasets.**make_extreme**(*size=100*, *nbin=None*)
Extreme value Distribution.

Parameters

- **size**(*int, optional(default=100)*) –
- **nbin**(*int, optional(default=None)*) –

Returns **out**

Return type float array

apode.datasets.**make_gamma**(*seed=None*, *size=100*, *shape=1*, *scale=50.0*, *nbin=None*)
Gamma Distribution.

Parameters

- **seed**(*int, optional(default=None)*) –

- **size** (*int, optional(default=100)*) –
- **shape** (*float, optional(default=1.0)*) –
- **scale** (*float, optional(default=50.0)*) –
- **nbin** (*int, optional(default=None)*) –

Returns **out** – Array of random numbers.

Return type float array

apode.datasets.**make_linear**(*size=100*, *nbin=None*)
Linear value Distribution.

Parameters

- **size** (*int, optional(default=100)*) –
- **nbin** (*int, optional(default=None)*) –

Returns **out**

Return type float array

apode.datasets.**make_lognormal**(*seed=None*, *size=100*, *sigma=1.0*, *nbin=None*)
Lognormal Distribution.

Parameters

- **seed** (*int, optional(default=None)*) –
- **size** (*int, optional(default=100)*) –
- **sigma** (*float, optional(default=1.0)*) –
- **nbin** (*int, optional(default=None)*) –

Returns **out** – Array of random numbers.

Return type float array

apode.datasets.**make_pareto**(*seed=None*, *a=5*, *size=100*, *c=200*, *nbin=None*)
Pareto Distribution.

Parameters

- **seed** (*int, optional(default=None)*) –
- **a** (*float, optional(default=5)*) –
- **size** (*int, optional(default=100)*) –
- **c** (*int, optional(default=200)*) –
- **nbin** (*int, optional(default=None)*) –

Returns **out** – Array of random numbers.

Return type float array

apode.datasets.**make_squared**(*size=100*, *nbin=None*)
Squared value Distribution.

Parameters

- **size** (*int, optional(default=100)*) –
- **nbin** (*int, optional(default=None)*) –

Returns **out**

Return type float array

apode.datasets.**make_uniform**(*seed=None*, *size=100*, *mu=100*, *nbin=None*)
Uniform Distribution.

Parameters

- **seed**(*int, optional(default=None)*) –
- **size**(*int, optional(default=100)*) –
- **mu**(*float, optional(default=100)*) –
- **nbin**(*int, optional(default=None)*) –

Returns **out** – Array of random numbers.

Return type float array

apode.datasets.**make_unimodal**(*size=100*, *nbin=None*)
Unimodal Distribution.

Parameters

- **size**(*int, optional(default=100)*) –
- **nbin**(*int, optional(default=None)*) –

Returns **out**

Return type float array

apode.datasets.**make_weibull**(*seed=None*, *size=100*, *a=1.5*, *c=50*, *nbin=None*)
Weibull Distribution.

Parameters

- **seed**(*int, optional(default=None)*) –
- **size**(*int, optional(default=100)*) –
- **a**(*float, optional(default=1.5)*) –
- **c**(*float, optional(default=50)*) –
- **nbin**(*int, optional(default=None)*) –

Returns **out** – Array of random numbers.

Return type float array

9.5.5 apode.inequality module

Inequality measures for Apode.

class apode.inequality.**InequalityMeasures**(*idf*)
Bases: object

Inequality measures for Apode.

The following inequality measures are implemented:

- gini: Gini Index
- entropy: Generalized Entropy Index

- atkinson: Atkinson Index
- rrange: Relative Range
- rad: Relative average deviation
- cv: Coefficient of variation
- sdlog: Standard deviation of log
- merhan: Merhan index
- piesch: Piesch Index
- bonferroni: Bonferroni Indices
- kolm: Kolm Index

Parameters

- **method** (*String*) – Inequality measure.
- ****kwargs** – Arbitrary keyword arguments.

atkinson(*alpha=2*)

Atkinson index.

The Atkinson index measures the proportion of the population that counted as poor.[12]

Parameters **alpha** (*float, optional(default=2)*) –

Returns **out** – Index measure.

Return type float

References

bonferroni()

Bonferroni Coefficient.

The Bonferroni Coefficient[10].

Returns **out** – Index measure.

Return type float

References

cv()

Coefficient of variation.

It is the quotient between the standard deviation and the mean.[4]

Returns **out** – Index measure.

Return type float

[12] Atkinson, AB (1970) On the measurement of inequality. Journal of Economic Theory, 2 (3), pp. 244–263.
[10] Bonferroni, C.E. (1930), Elementi di Statistica Generale, Seeber, Firenze.
[4] Atkinson, AB (1970) On the measurement of inequality. Journal of Economic Theory, 2 (3), pp. 244–263.

References

entropy(*alpha=0*)
General Entropy index.

The entropy index measures the proportion of the population that counted as poor.

Parameters **alpha** (*float, optional(default=0)*) –

Returns **out** – Index measure.

Return type float

gini()
Gini Coefficient.

The Gini Coefficient[7].

Returns **out** – Index measure.

Return type float

References

kolm(*alpha*)
Kolm Coefficient.

The Kolm Coefficient[11]

Parameters **alpha** (*float*) –

Returns **out** – Index measure.

Return type float

References

merhan()
Merhan Coefficient.

The Merhan Coefficient[8].

Returns **out** – Merhan Coefficient.

Return type float

[7] Gini, C. (1914), 'Sulla misura della concentrazione e della variabilità dei caratteri', Atti del Reale Istituto Veneto di Scienze, Lettere ed Arti 73, 1203-1248.
[11] Kolm, S.-Ch. (1 976a). 'Unequal Inequalitites 1', Journal of Economic Theory.
[8] Mehran, Farhad, 1976. "Linear Measures of Income Inequality," Econometrica, Econometric Society, vol. 44(4), pages 805-809, July.

References

piesch()
Piesch Coefficient.

The Piesch Coefficient[9].

Returns **out** – Index measure.

Return type float

References

rad()
Relative average deviation.

Ratio of the sum of the absolute value of the distance between each income in the distribution and the mean income, to total income.[3]

Returns **out** – Index measure.

Return type float

References

ratio(*alpha*)
Dispersion Ratio (Kuznets Ratio).

This measure presents the ratio of the average income of the richest alpha percent of the population to the average income of the poorest alpha percent[6].

Parameters **alpha** (*float*) –

Returns **out** – Index measure.

Return type float

References

rrange()
Relative range.

This measure divides the difference between the highest and lowest income by the mean income.

Returns **out** – Index measure.

Return type float

sdlog()
Calculate Standard deviation of logarithms.

Attach great importance to income transfers at the lower end.[5]

Returns **out** – Index measure.

Return type float

[9] Piesch, W. (1975). Statistische Konzentrationsmasse. Mohr (Paul Siebeck), Tübingen.
[3] Atkinson, AB (1970) On the measurement of inequality. Journal of Economic Theory, 2 (3), pp. 244–263.
[6] Haughton, J., and Khandker, S. R. (2009). Handbook on poverty and inequality. Washington, DC: World Bank.
[5] Atkinson, AB (1970) On the measurement of inequality. Journal of Economic Theory, 2 (3), pp. 244–263.

References

9.5.6 apode.plots module

Plots for Apode.

class apode.plots.**PlotAccessor**(*idf*)
Bases: object

Plots for Apode.

The following plots are implemented:

- hist : Histogram (default)
- lorenz : Lorenz curve (relative, generalized, absolute)
- pen : Pen Parade
- tip : Tip curve

Parameters

- **method** (*String*) – Plot type.
- ****kwargs** – Arbitrary keyword arguments.

lorenz(*alpha='r'*, *ax=None*, ***kwargs*)
Lorenz Curve.

A Lorenz curve is a graphical representation of the distribution of income or wealth within a population. Lorenz curves graph percentiles of the population against cumulative income or wealth of people at or below that percentile.[13]

Parameters

- **alpha** (*string, optional(default='r')*) – Options are r: relative, 'g': generalized, 'a': absolut.
- **ax** (*axes object, optional*) –

Returns **out** – Matplotlib plot

Return type plot

References

pen(*pline=None*, *ax=None*, ***kwargs*)
Pen Parade Curve.

Pen's Parade or The Income Parade is a concept described in a 1971 book published by Dutch economist Jan Pen describing income distribution. The parade is defined as a succession of every person in the economy, with their height proportional to their income, and ordered from lowest to greatest.[14]

Parameters

- **pline** (*float, optional*) –
- **ax** (*axes object, optional*) –

Returns **out** – Matplotlib plot

[13] Lorenz, M. O. (1905). Methods for measuring concentration of wealth. Journal of the American Statistical Association 9, 209-219.
[14] Pen, J. (1971). Income Distribution. London: Allen Lane, The Penguin Press.

Return type plot

References

tip(*pline*, *ax=None*, ***kwargs*)
TIP Curve.

Three 'I's of Poverty (TIP) curves, based on distributions of poverty gaps, provide evocative graphical summaries of the incidence, intensity, and inequality dimensions of poverty, and a means for checking for unanimous poverty orderings according to a wide class of poverty indices.[15]

Parameters

- **pline** (*float, optional*) –
- **ax** (*axes object, optional*) –

Returns **out** – Matplotlib plot

Return type plot

References

9.5.7 apode.polarization module

Polarization measures for Apode.

class apode.polarization.**PolarizationMeasures**(*idf*)
Bases: object

Polarization Measures.

The following welfare measures are implemented:

- ray : Esteban and Ray index
- wolfson : Wolfson index

Parameters **method** (*String*) – Polarization measure.

ray()
Esteban and Ray index of polarization.

Esteban and Ray index of polarization.[16]

Returns **out** – Polarization measure.

Return type float

[15] Jenkins S. P., Lambert P., 1997. Three "I's of Poverty" Curves, with an Analysis of UK Poverty Trends, Oxford Economic Papers, 49, pp. 317-327.
[16] Esteban, J.M. y D. Ray (1994), "On the Measurement of Polarization", Econometrica, vol. 62, N. 4, julio, pp. 819-851.

References

wolfson()
Wolfson index of bipolarization.

Wolfson index of bipolarization (normalized).[17]

Returns **out** – Polarization measure.

Return type float

References

9.5.8 apode.poverty module

Poverty measures for Apode.

class apode.poverty.**PovertyMeasures**(*idf*)
Bases: object

Poverty Measures.

The following poverty measures are implemented:

- headcount: Headcount Index
- gap: Poverty gap Index
- severity: Poverty Severity Index
- fgt: Foster–Greer–Thorbecke Indices
- sen: Sen Index
- sst: Sen-Shorrocks-Thon Index
- watts: Watts Index
- cuh: Clark, Ulph and Hemming Indices
- takayama: Takayama Index
- kakwani: Kakwani Indices
- thon: Thon Index
- bd: Blackorby and Donaldson Indices
- hagenaars: Hagenaars Index
- chakravarty: Chakravarty Indices

Parameters

- **method** (*String*) – Poverty measure.
- ****kwargs** – Arbitrary keyword arguments.

bd(*pline=None*, *alpha=2*, *factor=1.0*, *q=None*)
Blackorby and Donaldson Indices.

Blackorby y Donaldson (1980) proponen una medida de pobreza de tipo normativo.[29]

[17] Wolfson, Michael C. 1994. "When Inequalities Diverge." The American Economic Review 84 (2): 353–58.
[29] Blackorby, C. y Donaldson, D. (1980). "Ethical indices for the measurement of poverty". Econometrica. Vol. 48, n 4, pp.1053–1060.

Parameters

- **pline** (*optional(default=None)*) – Absolute poverty line if pline is float. Relative poverty line if pline is 'median', 'quantile' or 'mean' If pline is None then pline = 0.5*median(y).
- **factor** (*float, optional(default=1.0)*) – Factor in pline = factor*stat
- **q** (*float, optional(default=None)*) – Cuantil q if pline is'quantile'
- **alpha** (*float, optional(default=2)*) – Aversion parameter. (ver)

Returns **out** – Index measure in [0,1].

Return type float

References

chakravarty(*pline=None*, *alpha=0.5*, *factor=1.0*, *q=None*)
Chakravarty Indices.

Chakravarty (1983) es una medida ética de pobreza. El índice de pobreza se obtiene como la suma normalizada de las carencias de utilidad de los pobres.[31]

Parameters

- **pline** (*optional(default=None)*) – Absolute poverty line if pline is float. Relative poverty line if pline is 'median', 'quantile' or 'mean' If pline is None then pline = 0.5*median(y).
- **factor** (*float, optional(default=1.0)*) – Factor in pline = factor*stat
- **q** (*float, optional(default=None)*) – Cuantil q if pline is'quantile'
- **alpha** (*float, optional(default=0.5)*) – Aversion parameter. (ver)

Returns **out** – Index measures.

Return type float

References

cuh(*pline=None*, *alpha=0.5*, *factor=1.0*, *q=None*)
Clark, Ulph and Hemming index.

Clark, Hemming y Ulph (1981) proponen utilizar en la medida de pobreza de Sen, la medida de Atkinson en lugar del índice de Gini de los pobres.[25]

Parameters

- **pline** (*optional(default=None)*) – Absolute poverty line if pline is float. Relative poverty line if pline is 'median', 'quantile' or 'mean' If pline is None then pline = 0.5*median(y).
- **factor** (*float, optional(default=1.0)*) – Factor in pline = factor*stat
- **q** (*float, optional(default=None)*) – Cuantil q if pline is'quantile'
- **alpha** (*float, optional(default=0.5)*) – Atkinson parameter.

Returns **out** – Index measure in [0,1].

[31] Chakravarty, S.R. (1983). "A new index of poverty". Mathematical Social Sciences. Vol. 6, pp.307–313.
[25] Clark, S.R.; Hemming, R. y Ulph, D. (1981). "On indices for the measurement of poverty". Economic Journal. Vol. 91, pp.515–526.

Return type float

References

fgt (*pline=None*, *alpha=0*, *factor=1.0*, *q=None*)
Foster–Greer–Thorbecke Indices.

When parameter = 0, P0 is simply the headcount index. When = 1, the index is the poverty gap index P1, and when is set equal to 2, P2 is the poverty severity index. A se le conoce con el nombre de parámetro de aversión a la pobreza y, por tanto, cuanto mayor sea , más énfasis se le da al más pobre de los pobres.[21]

Parameters

- **pline** (*optional (default=None)*) – Absolute poverty line if pline is float. Relative poverty line if pline is 'median', 'quantile' or 'mean' If pline is None then pline = 0.5*median(y).
- **factor** (*float, optional (default=1.0)*) – Factor in pline = factor*stat
- **q** (*float, optional (default=None)*) – Cuantil q if pline is'quantile'
- **alpha** (*float, optional (default=0)*) – Aversion poverty parameter.

Returns **out** – Index measure in [0, 1].

Return type float

References

gap (*pline=None*, *factor=1.0*, *q=None*)
Poverty gap index.

The poverty gap index adds up the extent to which individuals on average fall below the poverty line, and expresses it as a percentage of the poverty line.[19]

Parameters

- **pline** (*optional (default=None)*) – Absolute poverty line if pline is float. Relative poverty line if pline is 'median', 'quantile' or 'mean' If pline is None then pline = 0.5*median(y).
- **factor** (*float, optional (default=1.0)*) – Factor in pline = factor*stat
- **q** (*float, optional (default=None)*) – Cuantil q if pline is'quantile'

Returns **out** – Index measure [0, 1].

Return type float

[21] Foster, J.E.; Greer, J. y Thorbecke, E. (1984). "A class of decomposable poverty measures". Econometrica. Vol. 52, n 3, pp.761–766.
[19] Haughton, J., and Khandker, S. R. (2009). Handbook on poverty and inequality. Washington, DC: World Bank.

References

hagenaars (*pline=None*, *factor=1.0*, *q=None*)
Hagenaars Index.

Hagenaars (1984) para obtener la medida de pobreza considera la función de evaluación social de la renta como V(x) = ln(x).[30]

Parameters

- **pline** (*optional(default=None)*) – Absolute poverty line if pline is float. Relative poverty line if pline is 'median', 'quantile' or 'mean' If pline is None then pline = 0.5*median(y).
- **factor** (*float, optional(default=1.0)*) – Factor in pline = factor*stat
- **q** (*float, optional(default=None)*) – Cuantil q if pline is'quantile'

Returns **out** – Index measure [unbounded].

Return type float

References

headcount (*pline=None*, *factor=1.0*, *q=None*)
Headcount index.

The headcount index measures the proportion of the population that counted as poor.[18]

Parameters

- **pline** (*optional(default=None)*) – Absolute poverty line if pline is float. Relative poverty line if pline is 'median', 'quantile' or 'mean' If pline is None then pline = 0.5*median(y).
- **factor** (*float, optional(default=1.0)*) – Factor in pline = factor*stat
- **q** (*float, optional(default=None)*) – Cuantil q if pline is'quantile'

Returns **out** – Index measure.

Return type float

References

kakwani (*pline=None*, *alpha=2*, *factor=1.0*, *q=None*)
Kakwani Indices.

La familia de Kakwani (1980) que pondera los déficit mediante una potencia del número de orden que ocupa cada individuo dentro del subgrupo de pobres. El parámetro identifica una cierta "aversión" al lugar ocupado en la sociedad.[27]

Parameters

- **pline** (*optional(default=None)*) – Absolute poverty line if pline is float. Relative poverty line if pline is 'median', 'quantile' or 'mean' If pline is None then pline = 0.5*median(y).

[30] Hagenaars, A. (1984). "A class of poverty indices". Center for Research in Public Economics. Leyden University.
[18] Haughton, J., and Khandker, S. R. (2009). Handbook on poverty and inequality. Washington, DC: World Bank.
[27] Kakwani, Nanak (1980). "On a Class of Poverty Measures". Econometrica, vol.48, n.2, pp.437-446

- **factor** (*float, optional(default=1.0)*) – Factor in pline = factor*stat
- **q** (*float, optional(default=None)*) – Cuantil q if pline is'quantile'
- **alpha** (*float, optional(default=2)*) – Aversion parameter.

Returns **out** – Index measure in [0, 1].

Return type float

References

sen (*pline=None*, *factor=1.0*, *q=None*)
Sen Index.

Sen (1976) proposed an index that seeks to combine the effects of the number of poor, the depth of their poverty, and the distribution poverty within the group.[22]

Parameters

- **pline** (*optional(default=None)*) – Absolute poverty line if pline is float. Relative poverty line if pline is 'median', 'quantile' or 'mean' If pline is None then pline = 0.5*median(y).
- **factor** (*float, optional(default=1.0)*) – Factor in pline = factor*stat
- **q** (*float, optional(default=None)*) – Cuantil q if pline is'quantile'

Returns **out** – Index measure in [0, 1].

Return type float

References

severity (*pline=None*, *factor=1.0*, *q=None*)
Squared Poverty Gap (Poverty Severity) Index.

To construct a measure of poverty that takes into account inequality among the poor, some researchers use the squared poverty gap index. This is simply a weighted sum of poverty gaps (as a proportion of the poverty line), where the weights are the proportionate poverty gaps themselves.[20]

Parameters

- **pline** (*optional(default=None)*) – Absolute poverty line if pline is float. Relative poverty line if pline is 'median', 'quantile' or 'mean' If pline is None then pline = 0.5*median(y).
- **factor** (*float, optional(default=1.0)*) – Factor in pline = factor*stat
- **q** (*float, optional(default=None)*) – Cuantil q if pline is'quantile'

Returns **out** – Index measure in [0, 1].

Return type float

[22] Sen, A. (1976). "Poverty: an ordinal approach to measurement". Econometrica 44(2), pp.219–231.
[20] Haughton, J., and Khandker, S. R. (2009). Handbook on poverty and inequality. Washington, DC: World Bank.

References

sst (*pline=None*, *factor=1.0*, *q=None*)
Sen-Shorrocks-Thon Index Index.

The Sen index has been modified by others, and one of the most attractive versions is the Sen-Shorrocks-Thon (SST) index. One strength of the SST index is that it can help give a good sense of the sources of change in poverty over time. This is because the index can be decomposed.[23]

Parameters

- **pline** (*optional(default=None)*) – Absolute poverty line if pline is float. Relative poverty line if pline is 'median', 'quantile' or 'mean' If pline is None then pline = 0.5*median(y).
- **factor** (*float, optional(default=1.0)*) – Factor in pline = factor*stat
- **q** (*float, optional(default=None)*) – Cuantil q if pline is'quantile'

Returns **out** – Index measure.

Return type float

References

takayama (*pline=None*, *factor=1.0*, *q=None*)
Takayama Index.

Takayama (1979) define su medida de pobreza calculando el índice de Gini de la distribución censurada por la línea de pobreza.[26]

Parameters

- **pline** (*optional(default=None)*) – Absolute poverty line if pline is float. Relative poverty line if pline is 'median', 'quantile' or 'mean' If pline is None then pline = 0.5*median(y).
- **factor** (*float, optional(default=1.0)*) – Factor in pline = factor*stat
- **q** (*float, optional(default=None)*) – Cuantil q if pline is'quantile'

Returns **out** – Index measure in [0,1].

Return type float

References

thon (*pline=None*, *factor=1.0*, *q=None*)
Thon Index.

La diferencia entre esta medida (Thon,1979) y la de Sen radica en la función de ponderación. Aquí se pondera el individuo pobre por el lugar que ocupa dentro de toda la población, y no solo respecto a los pobres.[28]

Parameters

[23] Xu, K. (1998). Statistical inference for the Sen-Shorrocks-Thon index of poverty intensity. Journal of Income Distribution, 8, 143-152.
[26] Takayama, N. (1979). "Poverty, income inequality, and their measures: Professor Sen's axiomatic approach reconsidered". Econometrica. Vol. 47, n 3, pp.747–759.
[28] Thon, D. (1979). "On measuring poverty". Review of Income and Wealth. Vol. 25, pp.429–439.

- **pline** (*optional(default=None)*) – Absolute poverty line if pline is float. Relative poverty line if pline is 'median', 'quantile' or 'mean' If pline is None then pline = 0.5*median(y).
- **factor** (*float, optional(default=1.0)*) – Factor in pline = factor*stat
- **q** (*float, optional(default=None)*) – Cuantil q if pline is'quantile'

Returns **out** – Index measure.

Return type float

References

watts (*pline=None*, *factor=1.0*, *q=None*)
Watts index.

Harold Watts (1968) propuso la siguiente medida de pobreza sensible a la distribución de rentas.[24]

Parameters

- **pline** (*optional(default=None)*) – Absolute poverty line if pline is float. Relative poverty line if pline is 'median', 'quantile' or 'mean' If pline is None then pline = 0.5*median(y).
- **factor** (*float, optional(default=1.0)*) – Factor in pline = factor*stat
- **q** (*float, optional(default=None)*) – Cuantil q if pline is'quantile'

Returns **out** – Index measure [0,inf].

Return type float

References

9.5.9 apode.welfare module

Welfare measures for Apode.

class apode.welfare.**WelfareMeasures**(*idf*)
Bases: object

Welfare Measures.

The following welfare measures are implemented:

- utilitarian : Utilitarian utility function
- rawlsian : Rawlsian utility function
- isoelastic : Isoelastic utility function
- sen : Sen utility function
- theill : Theill utility function
- theilt : Theilt utility function

Parameters

[24] Watts, H. (1968). "An economic definition of poverty", en D. P. Moynihan. On Understanding Poverty. Basic Books. Inc. New York, pp.316–329.

- **method** (*String*) – Welfare measure.
- ****kwargs** – Arbitrary keyword arguments.

isoelastic(*alpha*)
Isoelastic utility function.

The isoelastic utility function.

Returns **out** – Utility value.

Return type float

rawlsian()
Rawlsian utility function.

The rawlsian utility function.

Returns **out** – Utility value.

Return type float

sen()
Sen utility function.

The Sen utility function.

Returns **out** – Utility value.

Return type float

theill()
Theil L utility function.

The Theil L utility function.

Returns **out** – Utility value.

Return type float

theilt()
Theil T utility function.

The Theil T utility function.

Returns **out** – Utility value.

Return type float

utilitarian()
Utilitarian utility function.

The utilitarian utility function.

Returns **out** – Utility value.

Return type float

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

P

R

S

T

U

W