pyech

Release 0.1.0

CPA Ferrere | Data Analytics

Feb 24, 2022

CONTENTS

Process INE's ECH surveys in Python.

CHAPTER

ONE

USER DOCUMENTATION

Brief introduction.

1.1 Overview

A simple package that streamlines the download-read-wrangling process needed to analyze the Encuesta Continua de Hogares survey carried out by the Instituto Nacional de Estadística (Uruguay).

Here's what PyECH can do:

- Download survey compressed files.
- Unrar, rename and move the SAV (SPSS) file to a specified path.
- Read surveys from SAV files, keeping variable and value labels.
- Download and process variable dictionaries.
- Search through variable dictionaries.
- Summarize variables.
- Calculate variable n-tiles.
- Convert variables to real terms or USD.

PyECH does not attempt to estimate any indicators in particular, or facilitate any kind of modelling, or concatenate surveys from multiple years. Instead, it aims at providing a hassle-free experience with as simple a syntax as possible.

Surprisingly, PyECH covers a lot of what people tend to do with the ECH survey without having to deal with software licensing.

For R users, check out ech.

1.2 Installation

```
pip install pyech
```

1.2.1 Dependencies

In order to unpack downloaded survey files you will need to have unrar in your system. This should be covered if you have WinRAR or 7zip installed. Otherwise `sudo apt-get install unrar` or what's appropiate for your system.

1.3 Usage

- Full documentation, including this readme.
- Run the examples notebook in your browser |

Loading a survey is as simple as using `ECH.load`, which will download it if it cannot be found at `dirpath` (by default the current working directory).

```
from pyech import ECH

survey = ECH()
survey.load(year=2019, weights="pesoano")
```

Optionally, `load` accepts `from_repo=True`, which downloads survey data from the PyECH Github repository (HDFS+JSON). Loading data this way is significantly faster.

`ECH.load` also downloads the corresponding variable dictionary, which can be easily searched.

```
survey.search_dictionary("ingreso", ignore_case=True, regex=True)
```

This will return a pandas DataFrame where every row matches the search term in any of its columns.

Calculating aggregations is as simple as using `ECH.summarize`.

```
survey.summarize("ht11", by="dpto", aggfunc="mean", household_level=True)
```

Which returns a pandas DataFrame with the mean of "ht11" grouped by `ECH.splitter` and `by` (both are optional). Cases are weighted by the column defined in `ECH.load`.

CHAPTER

TWO

API DOCUMENTATION

Or read the API documentation (automatically generated from source code) for the specifics.

2.1 API documentation

2.1.1 ECH class

class pyech.core.**ECH**(*dirpath=None*, *categorical_threshold=50*, *splitter=None*)

Bases: `object`

Download, read and process the 2006-2020 Encuesta Continua de Hogares survey carried out by Uruguay's Instituto Nacional de Estadística.

Parameters

- **dirpath** (`Union[Path, str, None]`) – Path where to download new surveys or read existing ones, by default ".".
- **categorical_threshold** (`int`) – Number of unique values below which the variable is considered categorical, by default 50.
- **splitter** (*Optional[str]*) –

data
Survey data, by default pd.DataFrame().

Type pd.DataFrame

metadata
Survey metadata, by default None.

Type metadata_container

weights
Column in *data* used to weight cases. Generally "pesoano" for annual weighting, by default None

Type Optional[str]

splitter
Variable(s) to use for grouping in methods (*summarize*, *assign_ptile*), by default [].

Type Union[str, List[str]]

dictionary
Variable dictionary, by default pd.DataFrame().

Type pd.DataFrame

cpi
Monthly CPI data, by default pd.DataFrame().

Type pd.DataFrame

nxr
Monthly nominal exchange rate data, by default pd.DataFrame().

Type pd.DataFrame

classmethod from_data(*data*, *metadata*, *splitter=None*, *weights=None*)
Build *ECH* from *data* and *metadata* as created by *pyreadstat.read_sav()*.

Parameters

- **data** (DataFrame) – Survey data.
- **metadata** (metadata_container) – Survey metadata.
- **weights** (Optional[str]) – Column in *data* used to weight cases. Generally "pesoano" for annual weighting, by default None
- **splitter** (Union[str, List[str], None]) – Variable(s) to use for grouping in methods (*summarize*, *assign_ptile*), by default [].

Returns

Return type *ECH*

property splitter

property year: int

Return type int

property weights: Optional[str]

Return type Optional[str]

load(*year*, *from_repo=False*, *weights=None*, *splitter=None*, *missing='\\s+\\.'*, *missing_regex=True*, *lower=True*, *multiprocess=False*)
Load a ECH survey and dictionary from a specified year.

First attempt to read a survey by looking for "*year*.sav" in dirpath. If it cannot be found, download the .rar file, extract it to a temporary directory, move the renamed .sav file to dirpath and then read. Optionally replace missing values with *numpy.nan*, lower all variable names and download the corresponding variable dictonary.

For the 2020 survey a new column called "pesoano" is calculated according to the following formula: pesoano = pesomen / 12. The result is rounded and converted to *int*. This is because 2020 is the first survey that does not have annual weights ("pesoano"). However, they can be calculated from monthly weights ("pesomen").

Parameters

- **year** (int) – Survey year
- **from_repo** (bool) – If True, download the survey from the Github repo as a HDFS+JSON combo.
- **weights** (Optional[str]) – Variable used for weighting cases, by default None.
- **splitter** (Union[str, List[str], None]) – Variable(s) to use for grouping in methods (*summarize*, *assign_ptile*), by default []

- **missing** (Optional[str]) – Missing values to replace with *numpy.nan*. Can be a regex with *missing_regex=True*, by default r"s+.".
- **missing_regex** (bool) – Whether to parse *missing* as regex, by default True.
- **lower** (bool) – Whether to turn variable names to lower case. This helps with analyzing surveys for several years, by default True.
- **multiprocess** (bool) – Whether to use multiprocessing to read the file. It will use all available CPUs, by default False.

Return type None

static download(*dirpath*, *year*)

Download a ECH survey, unpack the .rar, extract the .sav, rename as "*year*.sav" and place in dirpath.

Parameters

- **dirpath** (Union[Path, str]) – Download location.
- **year** (int) – Survey year.

Return type None

get_dictionary(*year*)

Download and process variable dictionary for a specified year.

Parameters **year** (int) – Survey year.

Return type None

search_dictionary(*term*, *ignore_case=True*, *regex=True*)

Return rows in *dictionary* with matching terms.

Parameters

- **term** (str) – Search term.
- **ignore_case** (bool) – Whether to search for upper and lower case. Requires *regex=True*, by default True.
- **regex** (bool) – Whether to parse *term* as regex, by default True.

Returns DataFrame containing matching rows.

Return type pd.DataFrame

summarize(*variable*, *by=None*, *is_categorical=None*, *aggfunc='mean'*, *household_level=False*, *prequery=None*, *variable_labels=False*, *value_labels=True*, *dropna=False*)

Summarize a variable in *data*.

Parameters

- **variable** (str) – Variable to summarize.
- **by** (Union[str, List[str], None]) – Summarize by these groups, as well as those in *splitter*, by default None.
- **is_categorical** (Optional[bool]) – Whether *value* should be treated as categorical. If None, compare with categorical_threshold, by default None.
- **aggfunc** (Union[str, Callable]) – Aggregating function. Possible values are "mean", "sum", "count", or any function that works with pd.DataFrame.apply. If *values* is categorical will force *aggfunc="count"*, by default "mean".

- **prequery** (Optional[str]) – Pass a string representing a boolean expression to query the survey before summarizing. For example, 'e27 >= 18' would filter out observations where the "e27" variable (age) is lower than 18, and then carry on with summarization. Leverages pandas' query.
- **household_level** (bool) – If True, summarize at the household level (i.e. consider only *data* ["nper"] == 1), by default False.
- **variable_labels** (bool) – Whether to use variable labels from *metadata*, by default True.
- **value_labels** (bool) – Whether to use value labels from *metadata*, by default True.
- **dropna** (bool) – Whether to drop groups with no observations, by default False.

Returns Summarized variable.

Return type pd.DataFrame

Raises **AttributeError** – If *weights* is not defined.

assign_ptile(*variable*, *n*, *labels=False*, *by=None*, *result_weighted=False*, *name=None*, *household_level=False*)

Calculate n-tiles for a variable. By default add as new column to *data*.

Parameters

- **variable** (str) – Reference variable.
- **n** (int) – Number of bins to calculate.
- **labels** (Union[bool, Sequence[str]]) – Passed to *pandas.qcut*. If False, use *int* labels for the resulting bins. If True, name bins by their edges. Otherwise pass a sequence of length equal to *n*, by default False.
- **by** (Union[str, List[str], None]) – Calculate bins for each of the groups, as well as those in *splitter*, by default None.
- **result_weighted** (bool) – If True, return a pd.DataFrame with the weighted result. Else, add as a column to *data*, by default False
- **name** (Optional[str]) – Name for the new column. If None, set as "*variable*`_`*n*", by default None:
- **household_level** (bool) – If True, calculate at the household level (i.e. consider only *data* ["nper"] == 1), by default False.

Returns

Return type Optional[pd.DataFrame]

Raises **AttributeError** – If *weights* is not defined.

convert_real(*variables*, *start=None*, *end=None*)

Convert selected monetary variables to real terms.

Parameters

- **variables** (Union[str, List[str]]) – Column(s) in *data*. Can be a string or a sequence of strings for multiple columns.
- **start** (Union[str, datetime, date, None]) – Set prices to either of these dates or the mean between them, by default None.
- **end** (Union[str, datetime, date, None]) – Set prices to either of these dates or the mean between them, by default None.

Return type None

convert_usd(*variables*)
Convert selected monetary variables to USD.

Parameters **variables** (Union[str, List[str]]) – Column(s) in *data*. Can be a string or a sequence of strings for multiple columns.

Return type None

apply_weights(*variables*)
Repeat rows as many times as defined in *weights*.

Parameters **variables** (Union[str, List[str]]) – Columns for which weights should be applied. In general it is a good idea to avoid applying weights to all columns since this can result in a large DataFrame.

Returns

Return type pd.DataFrame

Raises **AttributeError** – If *weights* is not defined.

save(*base_filename*, *key='df'*, *complevel=9*, *complib='blosc'*, ***kwargs*)

Parameters

- **base_filename** (*str*) –
- **key** (*str*) –
- **complevel** (*Optional[int]*) –
- **complib** (*Optional[str]*) –

2.1.2 External utility functions

pyech.external.**get_cpi**()
Download and process CPI data.

Returns

Return type pd.DataFrame

pyech.external.**get_nxr**()
Download and process USDUYU nominal exchange rate.

Returns

Return type pd.DataFrame

PYTHON MODULE INDEX

INDEX