Analyzing Complex Survey Data Using Python

Introduction to the samplics Package

PYDATA, CAMBRIDGE MEETUP

28 APRIL 2021

About me

Statistician at UNICEF

Previously worked at Westat and Statistics Canada

Hold Ph.D. in Statistics

Social media handle

- Twitter / GitHub / LinkedIn: @MamadouSDiallo

Plan of the Presentation

- Introduction
- A short tour of samplics
  - Sample size calculation
  - Sample selection
  - Sample weighting
  - Population parameter estimation
  - T-test

Introduction

Why samplics?

- Python is missing a comprehensive survey sampling package
- Allow Python users to stay in the Python ecosystem when analysing survey data
- Help reduce the gap between official statistics and machine learning / data science

Disclaimer: current version of samplics is a beta stage and the APIs are not stable yet. While the code base has extensive testing, users should expect bugs and improvements that may break their code. Many features are being developed and may influence the design of existing APIs.

Samplics documentation: https://samplics.readthedocs.io/en/latest/index.html

@MamadouSDiallo <samplics.org>

Introduction

What is Survey Sampling?

- Random selection of a subset from a finite population
- Known probabilities of selection
- The sampling strategy or sampling design is often complex for operational, cost, or efficiency reasons
  - Stratification
  - Clustering, Stage selection
  - Phase selection
  - Calibration
  - Etc.

Survey sampling techniques are the set of statistical methods for estimating population parameters (e.g., mean, total, regression coefficients, etc.) under the sampling design.

Introduction

Research questions

- **what is the household poverty rate in the USA?**
- **is poverty rate the same between households headed by women vs men?**

To answer the research questions, let's consider a **subset** of the ACS 2019 as **our target population**.

Note

- American Community Survey (ACS) 2019 from IPUMS (https://usa.ipums.org/usa).
- Only a subset of the ACS 2019 data was used. Hence, the numbers do not represent the full ACS2019.
- All the analysis in this presentation are just for illustration purpose, not a proper statistical analysis of the 2019 ACS.

Introduction

As mentioned before, we use a subset of the ACS 2019 as our **universe / frame / census**

We cluster the households in the frame into **2,351 primary sampling units (PSUs)**

Each cluster is a geographic area composed of a few hundred of households

hhid	region	psu	sex	race	education	family_income	poverty
612101	Midwest	1116	Female	White	College	42000	0
1022602	South	1877	Female	White	No college	61400	0
715033	Northeast	1304	Male	White	College	108000	0
1207701	South	2229	Female	White	No college	22300	0
912305	Midwest	1683	Male	White	College	20110	0
356101	South	656	Male	White	College	42000	0
1193372	South	2204	Female	White	No college	43300	0
1090718	South	1999	Male	White	College	227000	0
1014851	South	1865	Female	White	College	171100	0
378475	West	701	Male	White	No college	47300	0
294309	South	543	Female	Black	College	27000	0
326362	South	603	Female	Black	No college	87500	0
788672	Northeast	1452	Female	Black	College	47200	0
245608	South	463	Female	White	No college	14300	1
645861	South	1181	Female	Black	College	59800	0

Sample Size


```python
from samplics.sampling import SampleSize

# Expected poverty poverty rate
expected_pov_rate = {
    "Midwest": 0.11,
    "Northeast": 0.09,
    "South": 0.15,
    "West": 0.13,
}

# Declare the sample size calculation parameters
pov_rate_sample = SampleSize(
    parameter="proportion", method="wald", stratification=True
)

# calculate the sample size
pov_rate_sample.calculate(target=expected_pov_rate, half_ci=0.06)

# show the calculated sample size
pov_rate_sample.samp_size
```

```
{'Midwest': 105, 'Northeast': 88, 'South': 137, 'West': 121}
```

```python
# Convert sample sizes to a pandas dataframe
pov_rate_sample.to_dataframe()
```

	_stratum	_target	_half_ci	_samp_size
0	Midwest	0.11	0.06	105.0
1	Northeast	0.09	0.06	88.0
2	South	0.15	0.06	137.0
3	West	0.13	0.06	121.0

Determine the number of clusters needed
Total: **32 PSUs**

```python
psu_sample_size = {}
for key in pov_rate_sample.samp_size:
    psu_sample_size[key] = np.ceil(pov_rate_sample.samp_size[key] / 15).astype(
        int
    )

psu_sample_size
```

```
{'Midwest': 7, 'Northeast': 6, 'South': 10, 'West': 9}
```

@MamadouSDiallo <samplics.org>

Selection

The **selection** module in samplics provides

- Simple random selection (SRS)
- Systematic selection
- Probability proportional to size (PPS)
  - Systematic (method="pps-sys") – with and without replacement
  - Brewer (method= "pps-brewer")
  - Hanurav-Vijayan (method= "pps-hv")
  - Murphy (method= "pps-murphy")
  - Rao-Sampford (method= "pps-sampford")

Selection

Two step selection

Step 1: select the PSUs (clusters of households)

Step 2: Select the households from the selected PSUs

Note

- For this presentation, we artificially constructed the PSUs.
- Often, data collection is needed after step 1 to create the sampling frame for the stage 2 selection.

Selection


```python
from samplics.sampling import SampleSelection

stage1_design = SampleSelection(
    method="pps-sys", stratification=True, with_replacement=False
)

np.random.seed(12345)

(
    psu_frame["psu_sample"],
    psu_frame["psu_hits"],
    psu_frame["psu_probs"],
) = stage1_design.select(
    samp_unit=psu_frame["psu"],
    samp_size=psu_sample_size,
    stratum=psu_frame["region"],
    mos=psu_frame["number_households"],
    to_dataframe=False,
    sample_only=False,
)

psu_frame.sample(10)
```

select() returns a tuple of three numpy arrays

- Sample indicator
- Number of hits
- Selection probabilities

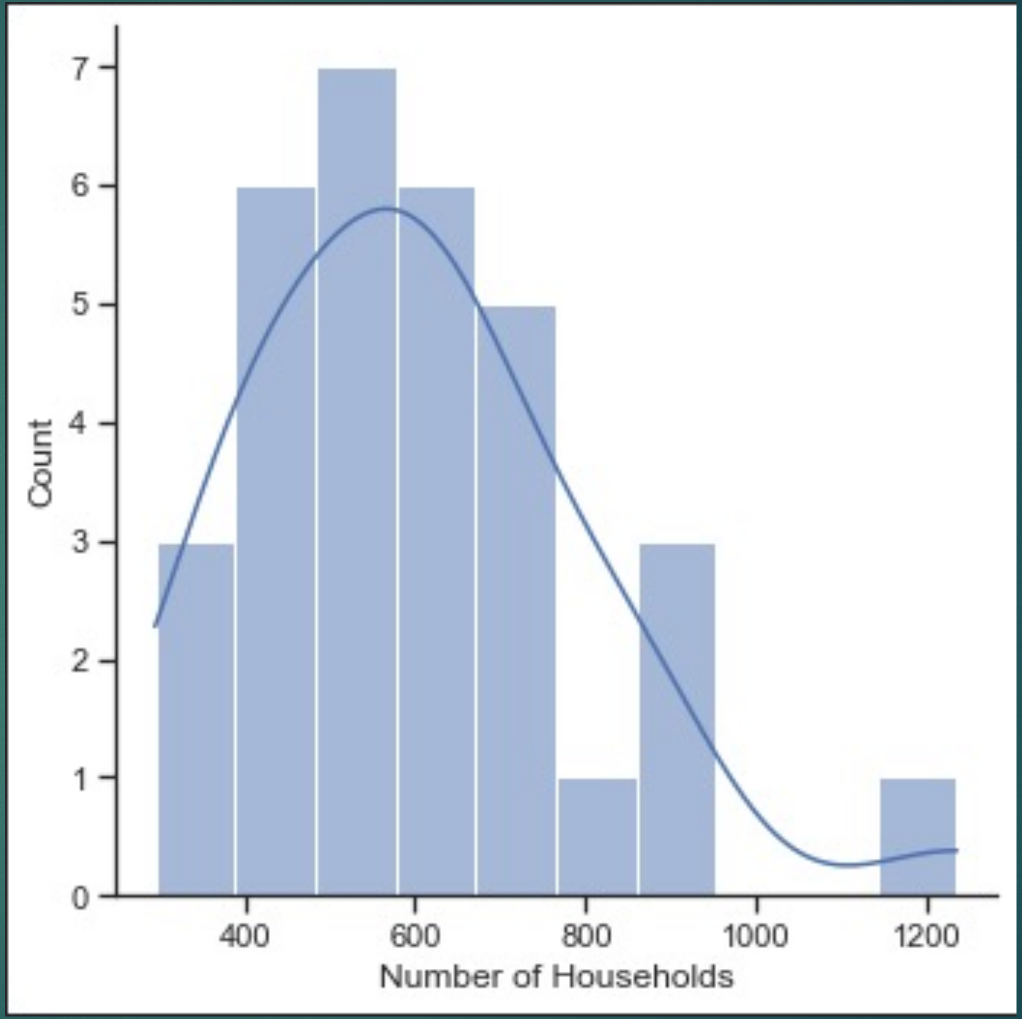
to_dataframe flag will return a pandas data frame when set to True

	region	state	psu	number_households	psu_sample	psu_hits	psu_probs
1377	South	North Carolina	1564	519	0	0	0.010641
1612	South	Texas	2033	197	0	0	0.004039
1430	South	Oklahoma	1715	485	0	0	0.009944
414	Midwest	Ohio	1663	448	0	0	0.011145
838	Northeast	Pennsylvania	1784	615	0	0	0.016539
2238	West	Oregon	1730	710	0	0	0.022461
131	Midwest	Indiana	836	562	0	0	0.013981
302	Midwest	Missouri	1202	927	0	0	0.023062
1585	South	Texas	2006	453	0	0	0.009288
1815	West	Arizona	46	500	1	1	0.015818

Selection

19,303 households listed in the 32 selected PSUs

In each PSU, we will select 15 households using SRS



Selection

Select 15 households from each PSU in the sample

We use simple random selection (srs)

All households have the same probability of selection within a stratum

The output is of type a pandas dataframe because to_dataframe=True.

```
np.random.seed(12345)

stage2_design = SampleSelection(
    method="srs", stratification=True, with_replacement=False
)

household_sample = stage2_design.select(
    samp_unit=household_frame["hhid"],
    samp_size=15,
    stratum=household_frame["psu"],
    to_dataframe=True,
    sample_only=True,
)

household_sample.sample(5)
```

	_samp_unit	_stratum	_mos	_sample	_hits	_probs
424	1119553	2057	1.0	1	1	0.030928
383	985675	1811	1.0	1	1	0.014822
301	784390	1444	1.0	1	1	0.036058
341	843148	1557	1.0	1	1	0.039164
277	702266	1279	1.0	1	1	0.029126

Weighting

Overall inclusion probabilities are the product of the probabilities of selection of each stage

Design weights are the inverse of the inclusion probabilities

```
# Probability of inclusion
household_sample["inclusion_probs"] = (
    household_sample["psu_probs"] * household_sample["hh_probs"]
)

# Base & Design weights
household_sample["design_weight"] = 1 / household_sample["inclusion_probs"]
```

Weighting

The **weighing** module in samplics provides

- Non-response adjustment
- Calibration including post-stratification
- Normalization
- Replicate weights
  - Balanced Repeated Replication (BRR)
  - Bootstrap
  - Jackknife

Weighting

Non-response weight adjustment consists of distributing the weight of non-respondents to respondents

Samplics uses a pre-codified scheme to distinguish the response status

- "in" for ineligible
- "rr" for respondent
- "nr" for non-respondent
- "uk" for unknown

If the response variable is not codified in the default scheme. (i.e., "in", "rr", "nr", "uk"), then the user must provide a mapping between the default codes and the user-defined codes.

Weighting

```python
from samplics.weighting import SampleWeight

response_status_mapping = {
    "in": "ineligible",
    "rr": "respondent",
    "nr": "non-respondent",
    "uk": "unknown",
}

household_sample["nr_weight"] = SampleWeight().adjust(
    samp_weight=household_sample["design_weight"],
    adjust_class=household_sample[["region", "race"]],
    resp_status=household_sample["response_status"],
    resp_dict=response_status_mapping,
    unknown_to_inelig=False,
)
```

	hhid	region	race	response_status	design_weight	nr_weight
406	1068887	South	White	respondent	3251.640000	3895.832830
277	703166	Northeast	Black	respondent	2478.966667	2478.966667
87	165715	West	White	respondent	2107.318519	2622.440823
455	1219025	West	White	respondent	2107.318519	2622.440823
338	841053	South	White	respondent	3251.640000	3895.832830
201	457655	Midwest	Asian	respondent	2679.761905	3573.015873
122	260577	South	White	respondent	3251.640000	3895.832830
354	905582	Midwest	White	non-respondent	2679.761905	0.000000
372	939112	West	White	respondent	2107.318519	2622.440823
251	631203	Midwest	White	respondent	2679.761905	3366.880342
325	822536	Northeast	Other	respondent	2478.966667	3305.288889
447	1179796	South	White	respondent	3251.640000	3895.832830
292	740282	Northeast	White	respondent	2478.966667	2784.070256
439	1179552	South	White	respondent	3251.640000	3895.832830
164	358293	South	White	non-respondent	3251.640000	0.000000

@MamadouSDiallo <samplics.org>

Weighting

Samplics implements the generalized regression (GREG) class for calibration

It requires known auxiliary variables control values at the population level

After the calibration adjustment, the weighted estimates of the auxiliary variables sum to the control values

```python
totals_by_domain = {
    "Midwest": {"poverty": 31414, "nb_children": 138952},
    "Northeast": {"poverty": 23280, "nb_children": 102614},
    "South": {"poverty": 64056, "nb_children": 239165},
    "West": {"poverty": 32131, "nb_children": 154986},
}

household_sample["calib_weight"] = SampleWeight().calibrate(
    samp_weight=household_sample["nr_weight"],
    aux_vars=household_sample[["poverty", "nb_children"]],
    control=totals_by_domain,
    domain=household_sample["region"],
)
```

```python
test_calib = household_sample[["region"]]

test_calib["poverty_weighted"] = (
    household_sample["poverty"] * household_sample["calib_weight"]
)
test_calib["nb_children_weighted"] = (
    household_sample["nb_children"] * household_sample["calib_weight"]
)

test_calib[["region", "poverty_weighted", "nb_children_weighted"]].groupby(
    "region"
).sum().reset_index()
```

	region	poverty_weighted	nb_children_weighted
0	Midwest	31414.0	138952.0
1	Northeast	23280.0	102614.0
2	South	64056.0	239165.0
3	West	32131.0	154986.0

@MamadouSDiallo <samplics.org>

Estimation

The **estimation** module in samplics provides

- Taylor-based estimates (class TaylorEstimator)
- Replicate-based estimates (class ReplicateEstimator)

Estimation

TaylorEstimator can estimate
- Proportions
- Means
- Totals
- Ratios
- **Quantiles (under development)**

For domain estimation, use function arguments domain.

Finite population correction (fpc) also possible

The APIs for **ReplicateEstimator** is similar with the use of rep_weight instead of samp_weight

```python
from samplics.estimation import TaylorEstimator

poverty_rate = TaylorEstimator(parameter="proportion")

poverty_rate.estimate(
    y=resp_sample["poverty"],
    samp_weight=resp_sample["calib_weight"],
    stratum=resp_sample["region"],
    psu=resp_sample["psu"],
)

print(poverty_rate)
```

```
SAMPLICS - Estimation of Proportion

Number of strata: 4
Number of psus: 32
Degree of freedom: 28

 LEVELS  PROPORTION       SE       LCI       UCI        CV
      0    0.881327  0.02047  0.832597  0.917281  0.023227
      1    0.118673  0.02047  0.082719  0.167403  0.172492
```

Tabulation and T-test

samplics provide APIs to produce survey-based tabulations and t-tests.

Tabulation() and **CrossTabulation()** classes are the main interfaces for producing one-way and two-way tables.

Rao-Scott adjustment implemented for both Pearson and Likelihood ration tests

Ttest() class is the main interface for comparison of group means.

```
from samplics.categorical import Ttest

poverty_by_sex = Ttest(samp_type="one-sample")
poverty_by_sex.compare(
    y=resp_sample["poverty"],
    group=resp_sample["sex"],
    samp_weight=resp_sample["calib_weight"],
    stratum=resp_sample["region"],
    psu=resp_sample["psu"],
)

print(poverty_by_sex)
```

```
Design-based One-Sample T-test
 Null hypothesis (Ho): mean(Female) = mean(Male)
 Equal variance assumption:
  t statictics: 0.6148
  Degrees of freedom: 391.00
  Alternative hypothesis (Ha):
   Prob(T < t) = 0.7305
   Prob(|T| > |t|) = 0.5390
   Prob(T > t) = 0.2695
 Unequal variance assumption:
  t statictics: 0.6076
  Degrees of freedom: 332.94
  Alternative hypothesis (Ha):
   Prob(T < t) = 0.7281
   Prob(|T| > |t|) = 0.5438
   Prob(T > t) = 0.2719

 Group  Nb. Obs      Mean  Std. Error  Std. Dev.  Lower CI  Upper CI
Female      203  0.128885    0.019332   0.275437  0.089285  0.168484
  Male      190  0.107717    0.028981   0.399481  0.048351  0.167082
```

Next steps

Develop more examples and training material

Add more features

- Expansion of the sample size module
- Addition of estimation for quantiles
- Next modules to be added
  - Survey-based regression modelling
  - Imputation methods

Thank you

@MamadouSDiallo
msdiallo@sampling.org