uncertain_panda Documentation

Release 0.2.0

Nils Braun

Feb 02, 2020

Contents

1 Content **3**
- 1.1 Examples . . . 3
- 1.2 Bootstrapping . . . 4
- 1.3 API . . . 5

2 Why is the panda uncertain? **7**

3 How to use it? **9**
- 3.1 Comparison in A/B testing . . . 9

4 Features **11**

5 How does it work? **13**

6 Other packages **15**

Index **17**

`uncertain_panda` helps you with constructing uncertainties of quantities calculated on your `pandas` data frames, by applying the method of bootstrapping.

CHAPTER 1

Content

1.1 Examples

1.1.1 A/B Testing

Your colleague John has developed a super-hot new feature for your companies website. But of course before turning it on, your boss wants you to check its quality. So you do some A/B-testing: a fraction of the users gets the new feature whereas most of the users still see the old website. The keep things simple: the users rate their experience with a single number between 0 and 1 afterwards.

Your boss has recently read a statistics book and now wants a different performance indicator than usual:

```
Instead of using the mean, we will calculate the median this time
as is is not so sensitive to outliers!
```

Your colleague John loads the data into his jupyter and quickly calculates the result:

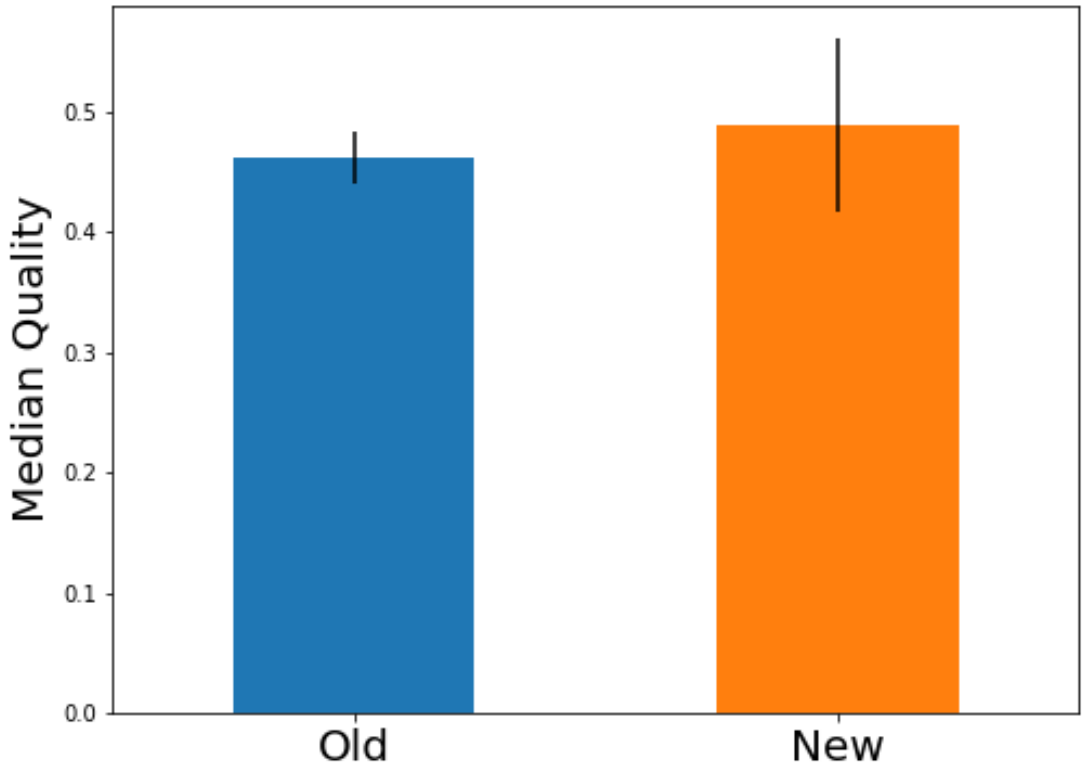
```
>> df.groupby("group").quality.median()
group
A    0.462222
B    0.487938
Name: quality, dtype: float64
```

"Nice", John says. The new method (method B) is much better than the old one! Let use it from now on! "Not so fast", you say, "how large is the statistical uncertainty on the result". "Statistical what?", asks John. "Well, your new method was only tested on a smaller sample. It could be, that you are measuring just an upwards fluctuation. Lets check it." So you also load the data into your notebook and just do a small change:

```
>> from uncertain_panda import pandas as pd
>> df.groupby("group").quality.unc.median()
group
A    0.462+/-0.021
B      0.49+/-0.07
Name: quality, dtype: object
```

"That was easy.", says John. "What are those additional numbers?" "This is the statistical uncertainty on the result." The smaller the sample size is, the less 'curtain' you can be on the result. And in this case it seems that the statistics is too small to make a reasonable statement. The two groups are compatible withing their uncertainties." "Very good you pointed that out", says John. "But how to tell our boss?" "I know he always likes figures", you respond. So you continue in the notebook

```
>> plt.figure(figsize=(8, 6))
>> df.groupby("group").quality.unc.median().\
        plot_with_uncertainty(kind="bar", ax=plt.gca())
```



"The error bars indicate the statistical uncertainty. You see, the new method seems better but with the high uncertainty you can not give a statement." "Nice", says John. "Let's show this to our boss!"

1.2 Bootstrapping

Suppose you want to calculate a quantity $f(X)$ on your data frame X. Bootstrapping samples multiple versions Y_i of X by drawing elements with replacement from the data frame with the same length the data frame itself. On all these Y_i, the function f is evaluated, creating a distribution of possible values for $f(X)$. The standard deviation of this distribution is the (symmetric) uncertainty returned by `uncertain_panda`. If you request the asymmetric uncertainty, the 1 sigma quantile in both directions around the median is returned. You can find some more information on bootstrapping in the net, e.g. on wikipedia.

1.3 API

1.3.1 First things first

If you want to use `uncertain_panda`, you need to call the following line at the beginning of your script/notebook:

```
from uncertain_panda import pandas as pd
```

This will load `pandas` as normal, but will add the uncertainty calculations described in the following. It is basically equivalent to do

```
import pandas as pd
```

and then add the additional features of `uncertain_panda` by hand.

1.3.2 Calculating a quantity with uncertainty

In the following, lets suppose you have a pandas object (could be a `pandas.Series`, a `pandas.DataFrame` or even the result of a `groupby` operation) which we call `df`. You want to calculate a function `f` on them and normally you would call

```
df.f()
```

or with arguments

```
df.f(some_arg=value)
```

To do the same calculation, but this time with uncertainties, just add an *unc* in between:

```
df.unc.f()
df.unc.f(some_arg=value)
```

The return value is a number/series/data frame (whatever `f` normally returns) with uncertainties. Thanks to the uncertainties package (make sure to star this great package), these results behave just as normal numbers. The error is even propagated correctly! Remember, `df` can be any pandas object and `f` can be any pandas function, so you can do things like

```
df.groupby("group").unc.median()
df[df.x > 3].y.unc.quantile(0.25)
(df + 2).loc["A"].unc.sum()
```

The results will behave as the normal function call - just with the uncertainties added!

1.3.3 Advanced functionality

Actually, the return values of the uncertainty calculations are not only "bare numbers with uncertainties". They are instances of *BootstrapResult* and have a bunch of additional functionality:

class uncertain_panda.**BootstrapResult**(*nominal_value*, *bootstrap*)

Result of any calculation performed with the *unc* wrapper. It is an instance of `uncertainties.core.Variable`, so it behaves like a normal number with uncertainties.

bs()

Return the full data sample of bootstrapped results. Usually used for visualisations, such as:

```
df["var"].unc.mean().bs().plot(kind="hist")
```

ci(*a=0.682689*, *b=None*)
Return the confidence interval between `a` and `b`. This is the pair of values [left, right], so that a fraction `a` of the bootstrapped results is left of `left` and `b` of the results is right of `right`. If you only give one parameter, the symmetric interval with `b = a` is returned. :return: a `pd.Series` with the columns `value`, `left` and `right`.

compare_ge(*rhs*)
How many of the values are >= than `rhs`?

compare_gt(*rhs*)
How many of the values are > than `rhs`?

compare_le(*rhs*)
How many of the values are <= than `rhs`?

compare_lt(*rhs*)
How many of the values are < than `rhs`?

prob(*value*)
Return the probability to have a resut equal or greater than `value`.

If we assume the bootstrapped results are a probability density function, this is equivalent to the p-value.

strip()
This result still includes the full sample of bootstrapped results. So it can be quite heavy (in terms of memory). The function returns an uncertainty number without the bootstrapped histogram.

1.3.4 Plotting

The package also adds another function to the basic pandas objects (`Series`, `DataFrame`) to plot values with uncertainties correctly. You can call it with

```
df.plot_with_uncertainty()
```

It is 1:1 equivalent to the normal call to `plot`, so you can put in the same argument

```
df.plot_with_uncertainty(kind="bar", label="Something", figsize=(20, 10))
```

CHAPTER 2

Why is the panda uncertain?

Have you ever calculated quantities on your pandas data frame/series and wanted to know their uncertainty? Did you ever wondered if the difference in the average of two methods is significant?

Then you want to have an uncertain panda!

`uncertain_panda` helps you calculate uncertainties on arbitrary quantities related to your pandas data frame e.g. `mean`, `median`, `quantile` or `min`/`max` and every other arbitrary function on pandas data frames!

You can use any measured data (e.g. from A/B testing, recorded data from an experiment or any type of tabular data) and calculate any quantity using `pandas` and `uncertain_panda` will give you the uncertainty on this quantity.

CHAPTER 3

How to use it?

First, install the package

```
pip install uncertain_panda
```

Now, just import pandas from the `uncertain_panda` package and prefix `unc` before every calculation to get the value with the uncertainty:

```
from uncertain_panda import pandas as pd

series = pd.Series([1, 2, 3, 4, 5, 6, 7])
series.unc.mean()
```

That's it! The return value is an instance of the uncertainty `Variable` from the superb uncertainties package. As this package already knows how to calculate with uncertainties, you can use the results as if they were normal numbers in your calculations.

```
series.unc.mean() + 2 * series.unc.std()
```

Super easy!

You can find some more examples in *Examples*.

3.1 Comparison in A/B testing

Suppose you have done some A/B testing with a brand new feature you want to introduce. You have measured the quality of your service before (A) and after (B) the feature introduction. The averge quality is better, but is the change significant?

A first measure for this problem might be the uncertainty of the average, so lets calculate it:

```
data_frame.groupby("feature_introduced").quality.unc.mean()
```

which will not only give you the two average qualities but also their uncertainties.

CHAPTER 4

Features

The development has just started and there is a lot that can still be added. Here is a list of already implemented features

- Automatic calculation of uncertainties of every built in pandas function for
  - data frames
  - series
  - grouped data frames

  using the prefix `unc` before the function name, e.g.

```
df.unc.mean()
```

  In the background, it used the method of bootstrapping (see below) to calculate the uncertainties.

- Calculate confidence intervals (instead of symmetric one-sigma uncertainties) or get back the basic bootstrapping distribution with

```
df.unc.mean().bs()  # for the bootstrap distribution
df.unc.mean().ci(0.3, 0.8)  # for the confidence interval between 0.3 and 0.8
```

- Opional usage of `dask` for large data samples. Enable it with

```
df.unc.mean(pandas=False)
```

  to use `dask` instead of pandas.

- Plotting functionality for uncertainties with

```
df.unc.mean().plot_with_uncertainties(kind="bar")
```

  for a nice error-bar plot.

- Full configurable bootstrapping with either using pandas built-in methods or `dask` (optionally enabled). Just pass the options to your called method, e.g.

```
df.unc.mean(number_of_draws=300)
```

to use 300 draws in the bootstrapping.

CHAPTER 5

How does it work?

Under the hood, `uncertain_panda` is using bootstrapping for calculating the uncertainties. Find more information on bootstrapping in *Bootstrapping*.

CHAPTER 6

Other packages

There are probably plenty of packages out there for this job, that I am not aware of. The best known is probably the bootstrapped package. Compared to this package, `uncertain_panda` tries to automate the quantity calculation and works for arbitrary functions. Also, it can use `dask` for the calculation. `bootstrapped` on the other hand is very nice for sparse arrays, which is not (yet) implemented in `uncertain_panda`.

Index

B

BootstrapResult (*class in uncertain_panda*), 5
bs() (*uncertain_panda.BootstrapResult method*), 5

C

ci() (*uncertain_panda.BootstrapResult method*), 6
compare_ge() (*uncertain_panda.BootstrapResult method*), 6
compare_gt() (*uncertain_panda.BootstrapResult method*), 6
compare_le() (*uncertain_panda.BootstrapResult method*), 6
compare_lt() (*uncertain_panda.BootstrapResult method*), 6

P

prob() (*uncertain_panda.BootstrapResult method*), 6

S

strip() (*uncertain_panda.BootstrapResult method*), 6