

---

# RandomSDSS

*Release 0.5.0*

**Martin Chalela**

Sep 10, 2021



**CONTENTS:**

<b>1</b>	<b>Basic Usage</b>	<b>3</b>
<b>2</b>	<b>Repository and Issues</b>	<b>5</b>
2.1	RandomSDSS . . . . .	5
2.2	Requirements . . . . .	8
2.3	Installation . . . . .	8
2.4	Licence . . . . .	8
	<b>Python Module Index</b>	<b>9</b>
	<b>Index</b>	<b>11</b>



Generate random points in Sloan Digital Sky Survey (SDSS) DR8 to DR16 footprints.

This is a small wrapper around the package `pymangle` that facilitates the creation of random points in the SDSS fields. I included SDSS polygons for its different data releases (DR8 to DR16).

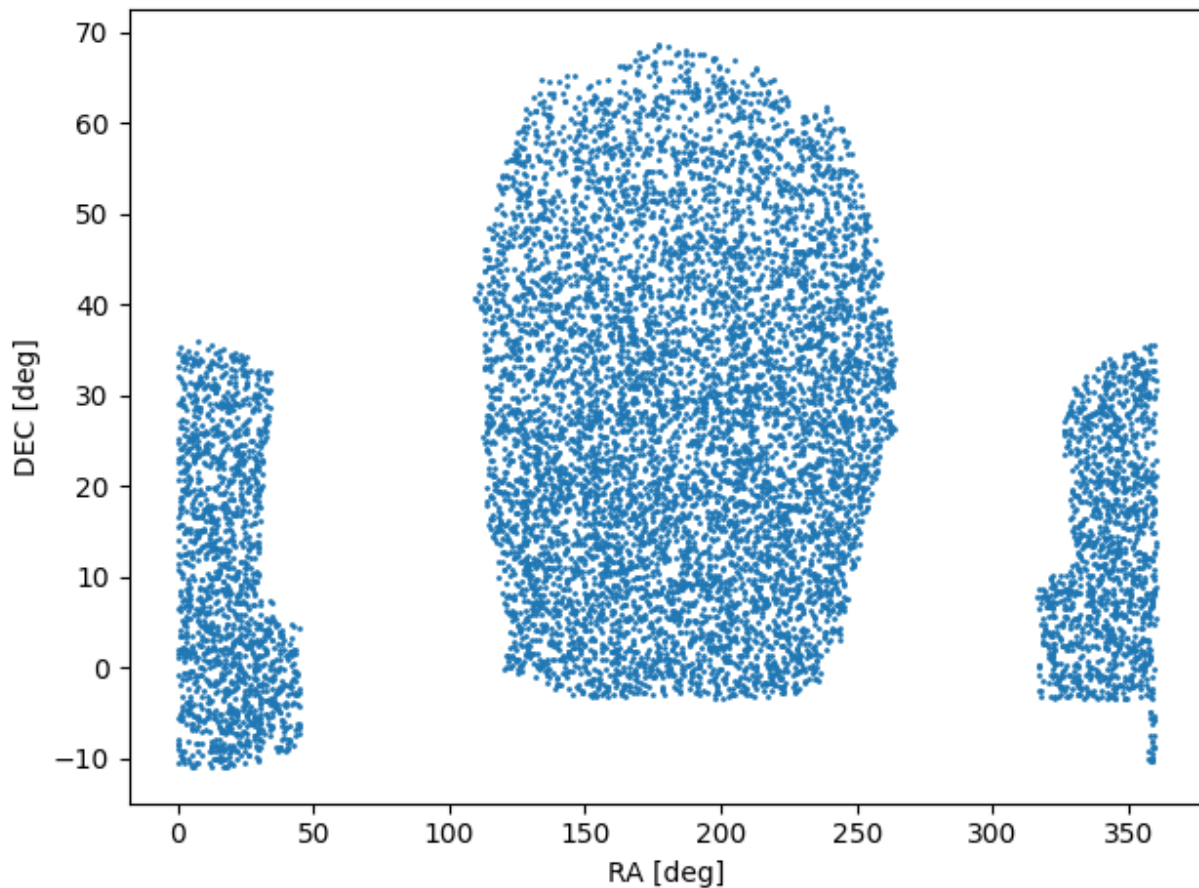


## BASIC USAGE

```
import matplotlib.pyplot as plt
import randomsdss

dr12 = randomsdss.DR12(catalog="BOSS")
ra, dec = dr12.sky_random(size=10_000)

plt.figure()
plt.scatter(ra, dec, s=1)
plt.xlabel('RA [deg]')
plt.ylabel('DEC [deg]')
```



Alternatively, you can get the same result without the need to instantiate an object using:

```
ra, dec = randomsdss.sky_random(dr="DR12", catalog="BOSS", size=10_000)
```

If you also need a random redshift distribution you can provide a sample of redshifts and a random set will be generated from the underlying Probability Density Function (PDF):

```
z = randomsdss.z_random(z_array, size=10_000)
```

The `z_random` is a complementary function since it does not use any information regarding the SDSS catalogs, only the provided redshift array.

### Author

Martin Chalela (E-mail: [tinchochalela@gmail.com](mailto:tinchochalela@gmail.com))



## REPOSITORY AND ISSUES

<https://github.com/mchalela/RandomSDSS>

### 2.1 RandomSDSS

Generate random points within SDSS DR8 to DR16 footprint.

**exception** `randomsdss.PolygonNotFoundError`

Bases: `FileNotFoundError`

Raised when the .ply file doesn't exist.

**class** `randomsdss.DR(dr, catalog)`

Bases: `object`

Base Data Release class.

#### Parameters

- **dr** (*str*) – Data Release name: e.g DR16.
- **catalog** (*str*) – Catalog name within the specified data release: e.g. BOSS.

**box\_random**(*ra\_min, ra\_max, dec\_min, dec\_max, size*)

Generate random RA, DEC points within a box.

#### Parameters

- **ra\_min** (*float*) – Right Ascension lower bound in degrees.
- **ra\_max** (*float*) – Right Ascension upper bound in degrees.
- **dec\_min** (*float*) – Declination lower bound in degrees.
- **dec\_max** (*float*) – Declination upper bound in degrees.
- **size** (*int*) – Number of random points to generate.

#### Returns

- **ra** (*numpy.ndarray*) – Right Ascension in degrees.
- **dec** (*numpy.ndarray*) – Declination in degrees.

**contains**(*ra, dec*)

Check if point is inside the catalog area.

#### Parameters

- **ra** (*numpy.ndarray*) – Right Ascension in degrees.

- **dec** (*numpy.ndarray*) – Declination in degrees.

**Returns** True if inside, False otherwise.

**Return type** bool

**polyid**(*ra, dec*)

Get polygon id of input point.

**Parameters**

- **ra** (*numpy.ndarray*) – Right Ascension in degrees.
- **dec** (*numpy.ndarray*) – Declination in degrees.

**Returns** **pid** – Polygon id. -1 if outside of catalog area.

**Return type** *numpy.ndarray*

**polyid\_and\_weight**(*ra, dec*)

Get polygon id and weight of input point.

**Parameters**

- **ra** (*numpy.ndarray*) – Right Ascension in degrees.
- **dec** (*numpy.ndarray*) – Declination in degrees.

**Returns**

- **pid** (*numpy.ndarray*) – Polygon id. -1 if outside of catalog area.
- **weight** (*numpy.ndarray*) – Polygon weight. 0 if outside of catalog area.

**set\_weights**(*weights*)

Set new weights for polygons.

**Parameters** **weight** (*float or numpy.ndarray*) – Polygons weights.

**sky\_random**(*size*)

Generate random RA, DEC points.

**Parameters** **size** (*int*) – Number of random points to generate.

**Returns**

- **ra** (*numpy.ndarray*) – Right Ascension in degrees.
- **dec** (*numpy.ndarray*) – Declination in degrees.

**weight**(*ra, dec*)

Get polygon weight of input point.

**Parameters**

- **ra** (*numpy.ndarray*) – Right Ascension in degrees.
- **dec** (*numpy.ndarray*) – Declination in degrees.

**Returns** **weight** – Polygon weight. 0 if outside of catalog area.

**Return type** *numpy.ndarray*

**property area**

Get the area of the catalog.

**property npoly**

Get the number of polygons.

**property weights**

Array of polygons weights.

**class** randomsdss.DR10(*catalog*='SDSS')

Bases: [randomsdss.randomsdss.DR](#)

**class** randomsdss.DR11(*catalog*='SDSS')

Bases: [randomsdss.randomsdss.DR](#)

**class** randomsdss.DR12(*catalog*='SDSS')

Bases: [randomsdss.randomsdss.DR](#)

**class** randomsdss.DR13(*catalog*='SDSS')

Bases: [randomsdss.randomsdss.DR](#)

**class** randomsdss.DR14(*catalog*='SDSS')

Bases: [randomsdss.randomsdss.DR](#)

**class** randomsdss.DR15(*catalog*='SDSS')

Bases: [randomsdss.randomsdss.DR](#)

**class** randomsdss.DR16(*catalog*='SDSS')

Bases: [randomsdss.randomsdss.DR](#)

**class** randomsdss.DR8(*catalog*='SDSS')

Bases: [randomsdss.randomsdss.DR](#)

**class** randomsdss.DR9(*catalog*='SDSS')

Bases: [randomsdss.randomsdss.DR](#)

randomsdss.sky\_random(*dr*='DR16', *catalog*='SDSS', *size*=10000)

Generate random RA, DEC values within the specified DR and catalog.

**Parameters**

- **dr** (*str*) – Data Release name: e.g DR16.
- **catalog** (*str*) – Catalog name within the specified data release: e.g. BOSS.
- **size** (*int*) – Number of random points to generate.

**Returns**

- **ra** (*numpy.ndarray*) – Right Ascension in degrees.
- **dec** (*numpy.ndarray*) – Declination in degrees.

randomsdss.z\_random(*z*, *size*=10000, *weights*=None, *seed*=None)

Generate random redshift values following the input distribution.

This function uses `scipy.stats.gaussian_kde` to compute the Probability Density Distribution (PDF).

**Parameters**

- **z** (*numpy.ndarray*) – Redshift sample to generate a PDF and extract random points.
- **size** (*int*) – Number of random points to generate.
- **weights** (*numpy.ndarray*) – Weights of each redshift value to compute a weighted PDF.
- **seed** (*int*) – Set random seed.

**Returns** **z\_rand** – Random redshifts.

**Return type** `numpy.ndarray`

## 2.2 Requirements

For the moment you will need to install numpy before installing RandomSDSS.

```
$ pip install numpy
```

## 2.3 Installation

The easiest way to install is using pip:

```
$ pip install randomsdss
```

This will install the latest stable version on PyPI.

If you want to use the latest development version from github, unpack or clone the *repo* <<https://github.com/mchalela/RandomSDSS>> 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 .
```

## 2.4 Licence

### MIT License

Copyright (c) 2021 Martin Chalela

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## PYTHON MODULE INDEX

**r**

randomsdss, [5](#)



## INDEX

### A

`area` (*randomsdss.DR property*), 6

### B

`box_random()` (*randomsdss.DR method*), 5

### C

`contains()` (*randomsdss.DR method*), 5

### D

`DR` (*class in randomsdss*), 5  
`DR10` (*class in randomsdss*), 7  
`DR11` (*class in randomsdss*), 7  
`DR12` (*class in randomsdss*), 7  
`DR13` (*class in randomsdss*), 7  
`DR14` (*class in randomsdss*), 7  
`DR15` (*class in randomsdss*), 7  
`DR16` (*class in randomsdss*), 7  
`DR8` (*class in randomsdss*), 7  
`DR9` (*class in randomsdss*), 7

### M

`module`  
    *randomsdss*, 5

### N

`npoly` (*randomsdss.DR property*), 6

### P

`PolygonNotFoundError`, 5  
`polyid()` (*randomsdss.DR method*), 6  
`polyid_and_weight()` (*randomsdss.DR method*), 6

### R

`randomsdss`  
    *module*, 5

### S

`set_weights()` (*randomsdss.DR method*), 6  
`sky_random()` (*in module randomsdss*), 7  
`sky_random()` (*randomsdss.DR method*), 6

### W

`weight()` (*randomsdss.DR method*), 6  
`weights` (*randomsdss.DR property*), 6

### Z

`z_random()` (*in module randomsdss*), 7