Wind Stats

Release 0.3.1

Jules Chéron

Jan 29, 2026

CONTENTS:

1 Getting Started 3

2 User Guide 9

3 API Reference 13

Python Module Index 19

Index 21

wind stats is a Python package that provides you basic functions to evaluate your wind turbine projects.

CHAPTER

ONE

GETTING STARTED

1.1 Installation

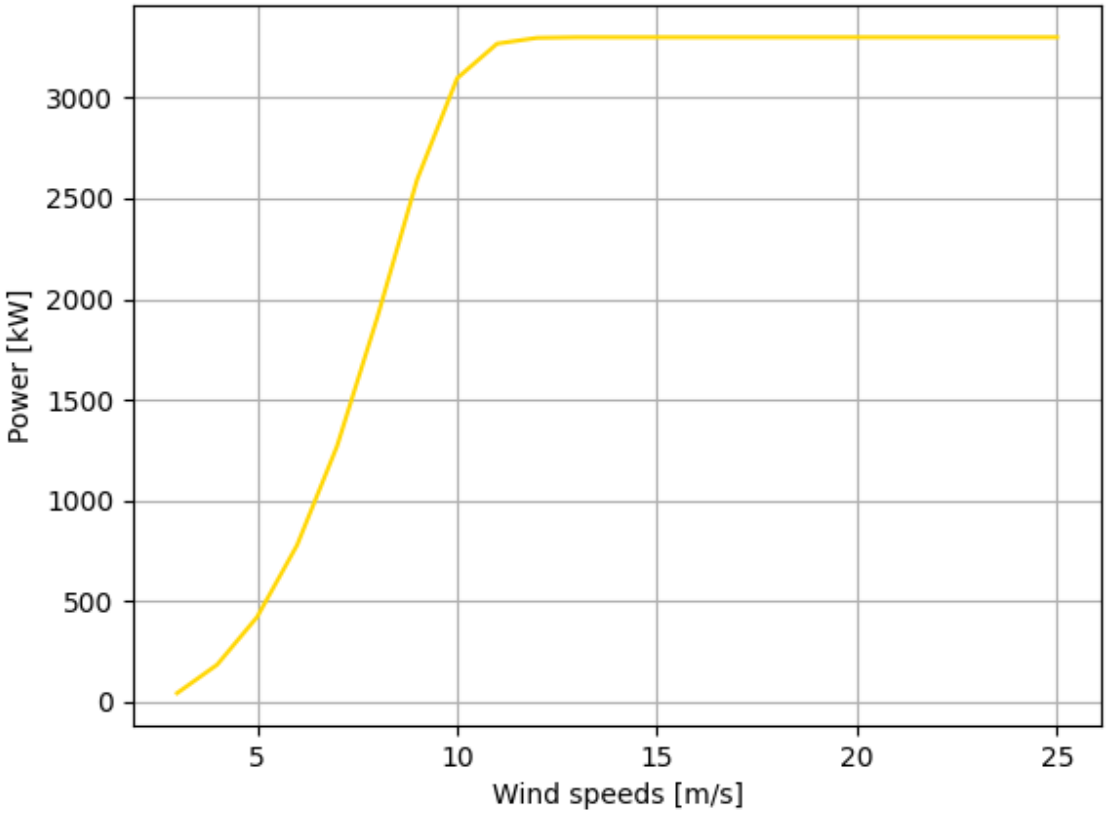
```
pip install wind-stats
```

1.2 Wind Turbine

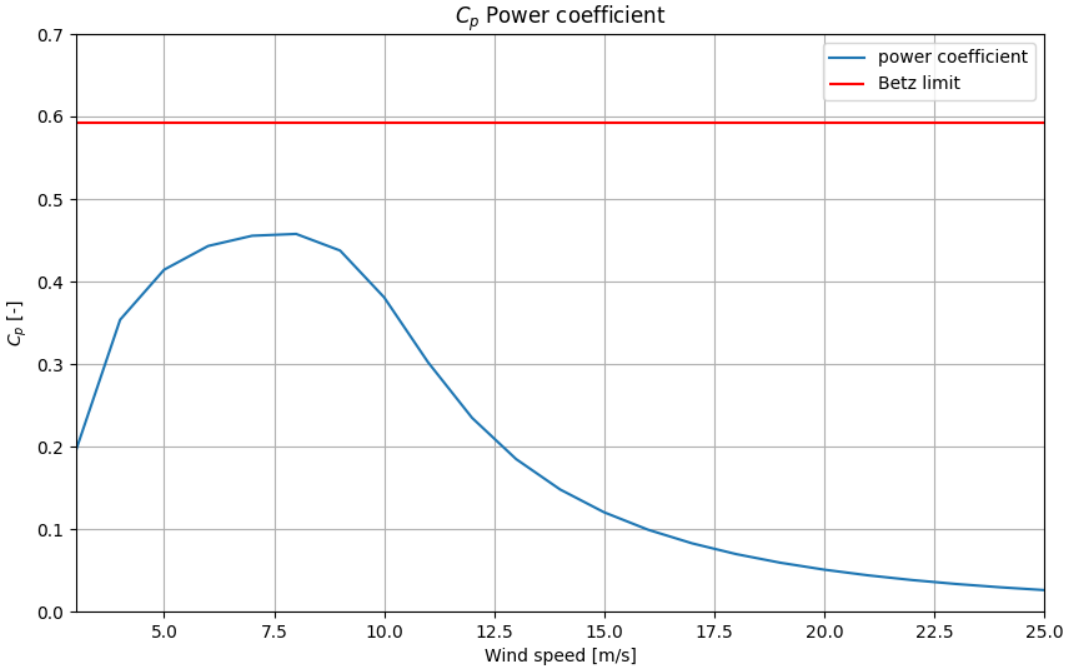
Wind turbine can be defined as follow :

```
In [1]: from wind_stats import WindTurbine

In [2]: from wind_stats.units import units

# https://en.wind-turbine-models.com/turbines/1467-siemens-swt-3.3-130-ln
# Power curve data
In [3]: wind_speed = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,␣
↪21, 22, 23, 24, 25] * units("m/s")

In [4]: power = [43., 184., 421., 778., 1270., 1905., 2593., 3096., 3268., 3297., 3300.,␣
↪3300., 3300., 3300., 3300., 3300., 3300., 3300., 3300., 3300., 3300., 3300. ] *␣
↪units.kW

In [5]: wind_turbine = WindTurbine("Siemens SWT-3.3-130 LN",(wind_speed, power), 130,␣
↪135)

In [6]: wind_turbine
Out[6]: <WindTurbine>(Siemens SWT-3.3-130 LN, 3.3 MW, height:135 m, diameter:130 m)
```

```
In [7]: In [1]: import matplotlib.pyplot as plt
   ...:     ...: plt.xlabel("Wind speeds [m/s]")
   ...:     ...: plt.ylabel("Power [kW]")
   ...:     ...: plt.grid()
   ...:     ...: plt.plot(wind_speed.m, power.m, color="gold")
   ...:
Out[7]: [<matplotlib.lines.Line2D at 0x71a361961bd0>]
```




```
In [8]: wind_speed, cp = wind_turbine.get_power_coefficients()
   ...: plt.figure(figsize=(10,6))
   ...: plt.plot(wind_speed, cp, label="power coefficient")
   ...: plt.hlines(16/27, 0, 25,color = "red", label="Betz limit")
   ...: plt.title("$C_p$ Power coefficient")
   ...: plt.xlabel("Wind speed [m/s]")
   ...: plt.ylabel("$C_p$ [-]")
   ...: plt.xlim(3, 25)
   ...: plt.ylim(0, 0.7)
   ...: plt.grid()
   ...: plt.legend();
   ...:

In [9]: plt.show()
```



1.3 Wind Distribution

To evaluate energy production, one need to know the wind distribution at the site the wind turbine is going to be installed.

To do that **wind-stats** is making things easy for you with using WASP compatible files that can be downloaded with Global Wind Atlas.

GWC (Global Wind Climate) File can be downloaded through the Global Wind Atlas or with `get_gwc_data` function.

Note

The generalized wind climate file, also known as a wind atlas file, contains the sector-wise frequency of occurrence of the wind (the wind rose) as well as the wind speed frequency distributions in the same sectors (as Weibull A- and k-parameters).

Important

You need to install `requests` to use `get_gwc_data` function.

See also

More options are available to generate wind distribution.

```
In [10]: from wind_stats import get_gwc_data

In [11]: latitude, longitude = 48.4569 , 5.583

In [12]: gwc_data = get_gwc_data(latitude, longitude)

In [13]: gwc_data
Out[13]:
<xarray.Dataset> Size: 6kB
Dimensions:    (roughness: 5, height: 6, sector: 12)
Coordinates:
  * roughness  (roughness) float64 40B 0.0 0.03 0.1 0.4 3.0
  * height     (height) float64 48B 10.0 50.0 100.0 150.0 200.0 250.0
  * sector     (sector) float64 96B 0.0 30.0 60.0 90.0 ... 270.0 300.0 330.0
Data variables:
    A          (roughness, height, sector) float64 3kB 5.43 6.77 ... 6.33 6.58
    k          (roughness, height, sector) float64 3kB 2.308 2.439 ... 2.51
    frequency  (roughness, sector) float64 480B 4.12 5.4 9.48 ... 6.52 4.66
Attributes:
    coordinates:  (48.45, 5.575)
```

Or with a downloaded GWC file through Global Wind Atlas interface:

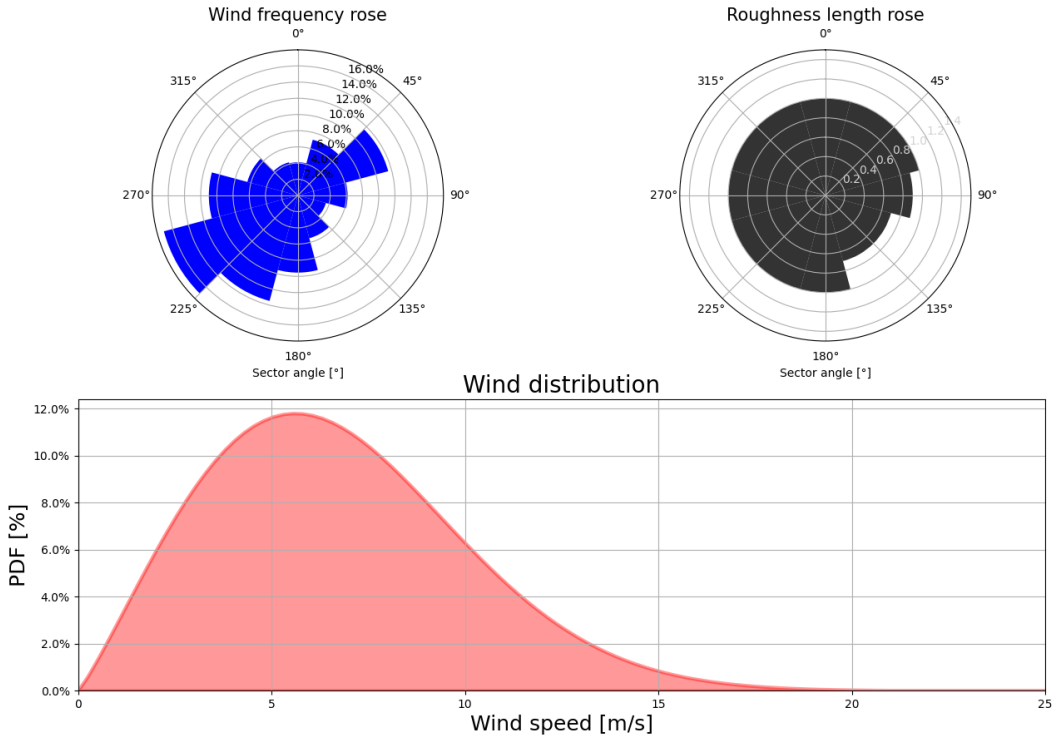
```
In [14]: from wind_stats import GWAReader

In [15]: with open("gwa3_gwc_j9mubhkw.lib") as f:
   ....:     gwc_data = GWAReader.load(f)
   ....:
```

Weibull A- and k-parameters & frequencies are interpolated provided hub-height & roughness length for each wind sector to generate the Weibull wind distribution.

See also

Roughness length user guide can help you evaluate roughness length at your site.

```
In [16]: from wind_stats import WindDistribution

In [17]: import matplotlib.pyplot as plt

In [18]: roughness_lengths_distribution = [1, 1, 1, 0.9, 0.7, 0.7, 1, 1 , 1, 1, 1, 1]

In [19]: wind_distribution = WindDistribution.from_gwc(
   ....:     gwc_data,
   ....:     roughness_lengths_distribution,
   ....:     wind_turbine.hub_height.m
   ....: )
   ....:

In [20]: wind_distribution
Out[20]: <WindDistribution>(type: weibull_min, mean: 6.6961 m/s)
```



1.4 Site

A Site is defined with GPS coordinates (latitude, longitude) & wind distribution.

```
In [21]: from wind_stats import Site

In [22]: site = Site(latitude, longitude, wind_distribution)
```

Tip

You can generate site from Global Wind Atlas wind distribution :

```
from wind_stats import Site

Site.from_gwc(latitude, longitude, roughness_lengths_distribution, height)
```

After defining our site we can now evaluate our annual energy output:

```
In [23]: annual_energy = wind_turbine.get_annual_energy_production(site).to("MWh")

In [24]: print(annual_energy)
11707 MWh
```

```
In [25]: energy = wind_turbine.get_energy_production(site, 3 * units.months).to("MWh")

In [26]: print(energy)
2926.7 MWh
```

CHAPTER

TWO

USER GUIDE

The User Guide covers different topics concerning wind-stats.

Be sure to check our *Getting Started* guide to grasp basic usage of the library.

Further information on any specific method or class can be obtained in the api.

2.1 Roughness length

Roughness length is an important parameter in the evaluation of vertical wind profile.

$$u(z) = \frac{u_*}{\kappa}\left[\ln\left(\frac{z-d}{z_0}\right) + \psi\left(\frac{z-d-z_0}{L}\right)\right] \quad (2.1)(2.1)$$

where :

- u_* is friction velocity.
- κ is Von Kármán constant (~0.4).
- z_0 is roughness length.
- L is the Monin-Obukhov length.

On standard condition (2.1) becomes :

$$u(z) = \frac{u_*}{\kappa}\ln\left(\frac{z}{z_0}\right) \quad (2.2)(2.2)$$

Deriving (2.2) we obtain the **log wind profile** to derive mean wind speed knowing the roughness length & wind speed at z_1.

$$u(z_2) = u(z_1)\frac{\ln(z_2-d)/z_0}{\ln(z_1-d)/z_0}$$

2.1.1 Roughness length evaluation

The surface roughness length over land depends on the surface cover and land use and is often difficult to estimate.

Class index	Short terrain description	z0 (m)
1	Open sea, fetch at least 5 km	0.0002
2	Mud flats, snow; no vegetation, no obstacles	0.005
3	Open flat terrain; grass, few isolated obstacles	0.03
4	Low crops; occasional large obstacles, x/H > 20	0.10
5	High crops; scattered obstacles, 15 < x/H < 20	0.25
6	Parkland, bushes; numerous obstacles, x/H 10	0.5
7	Regular large obstacle coverage (suburb, forest)	1.0
8	City centre with high- and low-rise buildings	2

Todo

Roughness length user guide under construction.

2.2 Wind Distribution

Statistical wind speed distribution are used to compute energy output from a wind turbine.

The well known weibull distribution is mostly used to fit data.

2.2.1 Use your own data

Anemometer data to create a wind distribution that will fit more precisely your own data.

Note

Look at WMO recommendation on how to measure winds. WMO recommends 10 min averages data points.

Averaging periods shorter than a few minutes do not sufficiently smooth the usually occurring natural turbulent fluctuations of wind

Wind stats can generate a wind distribution from your data using a Kernel Density Estimator (KDE). Wind speed distribution is scaled with vertical wind log profile if anemometer height & wind turbine height are different.

```
from wind_stats import WindDistribution

WindDistribution.from_data(data, roughness_length, measurement_height, height)
```

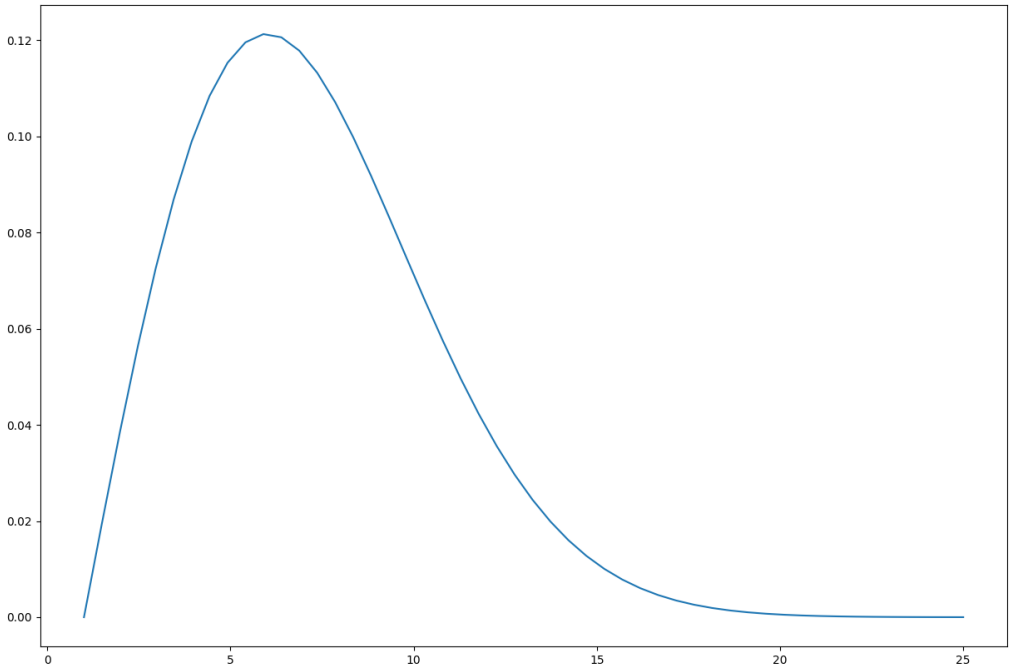
2.2.2 Other Statistical distribution

Wind stats uses `scipy` under the hood, so if another statistical distribution fits your need you can create it. Just create a `WindDistribution` with any continuous distribution in scipy.

https://docs.scipy.org/doc/scipy/reference/stats.html#continuous-distributions

```
In [1]: from scipy.stats import rayleigh

In [2]: from wind_stats import WindDistribution

In [3]: wind_distribution = WindDistribution(rayleigh(1, 5))

In [4]: wind_distribution
Out[4]: <WindDistribution>(type: rayleigh, mean: 7.2666 m/s)
```

```
In [5]: from matplotlib import pyplot as plt

In [6]: import numpy as np

In [7]: x = np.linspace(1, 25)

In [8]: y = wind_distribution.pdf(x)

In [9]: plt.plot(x, y)
Out[9]: [<matplotlib.lines.Line2D at 0x71a361abe950>]
```



Todo

Wind Distribution user guide under construction.

Todo

This section is under construction.

CHAPTER

THREE

API REFERENCE

class wind_stats.**GWAReader**

Methods

load(fp)	Deserialize *fp* GWC file-like object to dataset.
loads(s[, encoding])	Deserialize *s* (a str, bytes or bytearray instance containing a GWC document) to Dataset.

static load(*fp: SupportsRead*) → Dataset

Deserialize *fp* GWC file-like object to dataset.

static loads(*s: str | bytes*, *encoding: str = 'ASCII'*) → Dataset

Deserialize *s* (a str, bytes or bytearray instance containing a GWC document) to Dataset.

class wind_stats.**PowerCurve**(*wind_speed: Quantity*, *power: Quantity*)

Power curve.

Parameters

wind_speed: `pint.Quantity`
wind speed data

wind_speed: `pint.Quantity`
power data

Methods

__call__(x)	Linear interpolation on the power curve.

Examples

```
>>> power_data = [
        43.0,
        184.0,
        421.0,
        778.0,
        1270.0,
        1905.0,
        2593.0,
        3096.0,
        3268.0,
        3297.0,
        3300.0,
    ] * units.kW
>>> wind_speeds = np.arange(3, 13) * units("m/s")
>>> power_curve = PowerCurve(wind_speeds, power_data)
>>> power_curve(8  * units("m/s"))
<Quantity(1905.0, 'kilowatt')>
>>> power_curve(40  * units("m/s"))
<Quantity(0.0, 'kilowatt')>
```

class wind_stats.**Site**(*latitude: float*, *longitude: float*, *distribution: ~wind_stats.models.WindDistribution*, *elevation: float = 0.0*, *avg_temperature=<Quantity(15, 'degree_Celsius')>*, *avg_humidity: float = 0.0*)

Site location.

Attributes

latitude
longitude
distribution: WindDistribution
elevation
avg_temperature

Methods

create_gwa_data(latitude, longitude, ...)	Create Site with Global Wind Atlas Data.
get_mean_power_density()	Get mean power density in W/m^2.

property air_density

classmethod create_gwa_data(*latitude: float*, *longitude: float*, *roughness_length: float | List[float]*, *height: float*, ***kwargs*) → *Site*

Create Site with Global Wind Atlas Data.

Retrieve GWA data & initiate Site with Wind distribution.

get_mean_power_density() → Quantity

Get mean power density in W/m^2.

Notes

Mean power density :

$$\bar{P}_{density} = \frac{1}{2} \cdot \rho \cdot \int_0^\infty [v^3 pdf(v)] dv$$

property mean_power_density: Quantity

property mean_wind: Quantity

Mean wind speeds at site in m/s.

class wind_stats.**WindDistribution**(*distribution: rv_continuous*)

Wind distribution.

Attributes

mean_wind_speed

Methods

from_data(wind_speed_data, roughness_length, ...)	Create KDE WindDistribution based on measurement data.
from_gwc(gwc_dataset, roughness_length, height)	Create Weibull WindDistribution based on GWC file dataset.
moment(n)	Get n-raw moment of the distribution.
pdf(x)	Probability density function.
weibull(A, k)	Create Weibull WindDistribution.

classmethod from_data(*wind_speed_data*, *roughness_length: float | List[float]*, *measurement_height: float*, *height: float*) → *WindDistribution*

Create KDE WindDistribution based on measurement data.

classmethod from_gwc(*gwc_dataset: xr.Dataset*, *roughness_length: float | List[float]*, *height: float*) → *WindDistribution*

Create Weibull WindDistribution based on GWC file dataset.

property mean_wind_speed: Quantity

moment(*n: int*) → float

Get n-raw moment of the distribution.

pdf(*x: float*) → float

Probability density function.

classmethod weibull(*A: float*, *k: float*) → *WindDistribution*

Create Weibull WindDistribution.

class wind_stats.**WindTurbine**(*name: str*, *power_curve: Tuple[Quantity, Quantity]*, *diameter: float*, *height: float*)

Attributes

rotor_area	Swept rotor area.

Methods

get_annual_energy_production(site)	Get annual energy production.
get_energy_production(site, time)	Calculate energy output over a period of time.
get_mean_power(site)	Mean power output.
get_power_coefficients()	Get Cp coefficients for wind speeds defined in the power curve.

get_annual_energy_production(*site:* Site) → Quantity

Get annual energy production.

Integrate wind frequency with power curve & annual hours.

Parameters

site: Site
Site where the wind turbine is located.

Returns

energy: *pint.Quantity*
Annual energy to be expected in Wh.

See also

get_energy_production
Get energy production over any period of time.

get_energy_production(*site:* Site, *time: Quantity*) → Quantity

Calculate energy output over a period of time.

Parameters

site: Site
Site where the wind turbine is located.

time: `pint.Quantity`
period of time the result energy is produced.

Returns

energy_production: *pint.Quantity*

See also

get_annual_energy_production
Get energy output over a year.

Notes

The energy produced is the integration of the power production on the wind speed probability density function over time:

$$E = \bar{P} \cdot t$$

$$E = \int_0^\infty [P(v) \cdot pdf(v) \cdot t] dv$$

get_mean_power(*site:* Site) → Quantity

Mean power output.

$$\bar{P} = \int_0^\infty [P(v) \cdot pdf(v)] dv$$

Parameters

site: Site

Returns

mean_power: *pint.Quantity*

get_power_coefficients() → Tuple[ndarray, ndarray]

Get Cp coefficients for wind speeds defined in the power curve.

Notes

The power coefficient C_p is the ratio between the power extracted & the theoretical wind power available going through the swept area.

$$C_p = \frac{P}{\frac{1}{2}\rho A V^2}$$

property rotor_area: Quantity

Swept rotor area.

wind_stats.**get_gwc_data**(*latitude: float*, *longitude: float*) → Dataset

Get GWC file from Global Wind Atlas API.

Notes

See Global Wind Atlas Term of Use : https://globalwindatlas.info/about/TermsOfUse

wind_stats.**get_weibull_parameters**(*ds: Dataset*, *roughness_length: list[float] | float*, *height: float*) → tuple[float, float, list[float]]

Get A, k Weibull parameters based on GWA dataset files.

It computes based on roughness length & height for each wind rose sector.

Parameters

ds: xr.Dataset
dataset read from GWC file.

roughness_length: float or list[float]
array of 12 roughness lengths for each azimuthal wind sector (30°) or a global roughness_length.

height: float
height at which wind is measured.

Returns

parameters: tuple
Global weibull parameters & normalized frequencies for each sector.

To the getting started guide

To the reference guide

PYTHON MODULE INDEX

W

wind_stats, 13

INDEX

A

air_density (*wind_stats.Site property*), 14

C

create_gwa_data() (*wind_stats.Site class method*), 14

F

from_data() (*wind_stats.WindDistribution class method*), 15
from_gwc() (*wind_stats.WindDistribution class method*), 15

G

get_annual_energy_production() (*wind_stats.WindTurbine method*), 16
get_energy_production() (*wind_stats.WindTurbine method*), 16
get_gwc_data() (*in module wind_stats*), 17
get_mean_power() (*wind_stats.WindTurbine method*), 17
get_mean_power_density() (*wind_stats.Site method*), 14
get_power_coefficients() (*wind_stats.WindTurbine method*), 17
get_weibull_parameters() (*in module wind_stats*), 17
GWAReader (*class in wind_stats*), 13

L

load() (*wind_stats.GWAReader static method*), 13
loads() (*wind_stats.GWAReader static method*), 13

M

mean_power_density (*wind_stats.Site property*), 15
mean_wind (*wind_stats.Site property*), 15
mean_wind_speed (*wind_stats.WindDistribution property*), 15
module
    wind_stats, 13
moment() (*wind_stats.WindDistribution method*), 15

P

pdf() (*wind_stats.WindDistribution method*), 15
PowerCurve (*class in wind_stats*), 13

R

rotor_area (*wind_stats.WindTurbine property*), 17

S

Site (*class in wind_stats*), 14

W

weibull() (*wind_stats.WindDistribution class method*), 15
wind_stats
    module, 13
WindDistribution (*class in wind_stats*), 15
WindTurbine (*class in wind_stats*), 15