Spines

Release 0.0.5+0.g3c14eee.dirty

Douglas Daly

Apr 19, 2019

GETTING STARTED

Backbones for parameterized models.

CHAPTER

ONE

ABOUT

Spines was built to provide a skeleton for Model classes: a common interface for users to build models around (with some tools and utilities which take advantage of those commonalities). It's core Model class is similar, in structure, to some of scikit-learn's underlying Estimator classes - but with a single set of unified functions for all models, namely:

- Build
- Fit
- Predict
- Score
- Error

The predict method is the only one that's required to be implemented, though the others are likely useful most of the time (and often required to take advantage of some of the additional utilities provided by spines).

Spines also incorporates automatic version management for your models - something akin to a very lightweight git - but for individual models. It also caches results generated during various iterations of the development/fitting process so that they're not lost during - something that can (and often does) happen during very iterative model development work.

CHAPTER TWO

INSTALLING

Install with your favorite package manager, like `pipenv`:

```
$ pipenv install spines
```

CHAPTER

THREE

SIMPLE EXAMPLE

To demonstrate how to build a model with spines we'll use a toy example of a simple linear regression model. First we import what we'll need:

```
import numpy as np

from spines import Model, Parameter
```

Now we'll create the model class:

```
class LinearRegression(Model):
    """
    Simple linear regression model

        y = mx + b

    """
    m = Parameter(float)
    b = Parameter(float)

    def fit(self, x, y):
        covs = np.cov(x, y)
        self.m = (covs[0, 1] / np.var(x))
        self.b = np.mean(y) - (self.m * np.mean(x))

    def predict(self, x):
        return (self.m * x) + self.b
```

Now that we have the model we can generate some random data to fit it with:

```
x = np.random.rand(10)
y = 3.0 * x
x += np.random.normal(scale=0.05, size=(10,))
```

Then create and fit the model:

```
model = LinearRegression()
model.fit(x, y)
```

If we look at the `model.parameters` attribute we should see something like `b` being a small-ish number around 0 and `m` being close to 3.

See the *quick start* page for a slightly more in-depth example.

3.1 Installation

You can install `spines` in one of two ways: either from the PyPI package (the most common way) or from the source code.

3.1.1 Package

To install the `spines` package follow the instructions for your package manager below.

pip

```
$ pip install spines
```

pipenv

```
$ pipenv install spines
```

3.1.2 Source

To install the `spines` package from the source code first clone the git repository:

```
$ git clone https://github.com/douglasdaly/spines.git
```

Then install with your preferred package manager.

3.2 Quick Start

To demonstrate some of the features of the spines package we'll begin by constructing a simple OLS regression model.

3.2.1 Creating a Model

First we'll import the libraries we'll need, in our case here just numpy and spines:

[1]:
```
import numpy as np

from spines import Model, Parameter
```

Now we'll construct the OLS Regression model class:

[2]:
```
class OLSRegression(Model):
    """
    OLS Regression model
    """
    betas = Parameter(np.ndarray)
    intercept = Parameter(bool, default=False)

    def fit(self, X, y):
```

(continues on next page)

(continued from previous page)

```
        """Fits the model"""
        if self.intercept:
            X = np.hstack((X, np.full((X.shape[0], 1), 1.0)))
        self.betas = np.matmul(np.matmul(np.linalg.inv(np.matmul(X.T, X)), X.T), y)

    def predict(self, X):
        return np.matmul(X, self.betas)

    def error(self, X, y):
        y_hat = self.predict(X)
        return np.mean((y-y_hat)**2.0)
```

Let's generate some random, slightly noisy data to fit the model with:

[3]:
```
X = np.random.rand(100, 3)
y = (X * np.array([1.0, 0.5, 2.0])).sum(axis=1)
X += np.random.normal(scale=0.01, size=X.shape)
y += np.random.normal(scale=0.05, size=y.shape)
```

Now we can create our model instance and fit it:

[4]:
```
ols_model = OLSRegression()
ols_model.fit(X, y)
```

The results:

[5]:
```
ols_model.parameters
```

[5]:
```
<ParameterStore final=True> {
  <Parameter betas [type=ndarray] (required)>: [0.99080627 0.51589418 2.00767143],
  <Parameter intercept [type=bool]>: False,
}
```

[6]:
```
ols_model.error(X, y)
```

[6]:
```
0.002558902684175054
```

3.3 Parameters

Spines models hold stores of *spines.Parameter* objects. These objects specify parameters that the model requires as well as any restrictions or constraints on them. There are different types of parameter classes (aside from the base class), but all of them share these common attributes:

value_type The type(s) of data that are allowed for the value of the parameter.

default An optional default value for the parameter if it's not otherwise specified.

desc An optional description for the parameter.

There are a number of helpers and mixins so that you can create parameter classes to suit your particular use case.

3.4 Models

The core class of the `spines` package is the *spines.Model* class. All models have four primary functions in common: `build`, `fit`, `predict` and `error`. You can implement as many additional functions as needed but these lie at the heart of the `spines` library.

3.4.1 build

The `build` function is optional and called prior to any fitting or predicting. It's job is to do any initialization required for the model prior to use.

3.4.2 fit

The `fit` function (aka train) takes input data and it's corresponding output data and fits the model. This function is not required (though it is likely implemented for most use cases).

3.4.3 predict

The `predict` function takes input data and generates it's corresponding outputs based on the parameters of the model.

3.4.4 error

The `error` function takes input and output data and calculates an error measure for the model.

3.5 API Reference

3.5.1 spines

Spines

Backbones for parameterized models.

class `spines.`**`Model`**(**args*, ***kwargs*)

Bases: `object`

Model class

`build`(**args*, ***kwargs*) → None

Builds the model

Parameters

- **`args`** (*`optional`*) – Arguments to use in building the model.
- **`kwargs`** (*`optional`*) – Keyword arguments to use in building the model.

`error`(**args*, ***kwargs*) → float

Returns the error measure of the model for the given data

Parameters

- **`args`** (*`optional`*) – Additional arguments to pass to the error call.
- **`kwargs`** (*`optional`*) – Additional keyword-arguments to pass to the error call.

Returns Error for the model on the given inputs and outputs.

Return type float

fit(**args*, ***kwargs*) → None
Fits the model

Parameters

- **args** (*optional*) – Arguments to use in fit call.
- **kwargs** (*optional*) – Any additional keyword arguments to use in fit call.

get_hyper_params() → Dict[str, object]
Gets the current hyper-parameter values

Returns Copy of the currently set hyper-parameter values.

Return type dict

See also:

hyper_parameters(), *set_hyper_params()*

get_params() → dict
Gets a copy of this models parameters

Returns Copy of currently set parameter names and values.

Return type dict

hyper_parameters
Hyper-parameters which are currently set.

Type *ParameterStore*

classmethod load(*path: str*, *fmt: [<class 'str'>, None] = None*, *new: bool = False*) → Type[Model]
Loads a saved model instance

Loads saved model *parameters* and *hyper_params* as well as any serialized model-specific objects from a saved version with the *tag* specified (from the base *project_dir*).

Parameters

- **path** (*str*) – Path to load the model from.
- **fmt** (str or None) – Format of the file to load (if None it will be inferred).
- **new** (*bool, optional*) – Whether to create a new object from this class or use the saved object class (default is False).

Returns Model loaded from disk.

Return type *Model*

parameters
Parameters which are currently set.

Type *ParameterStore*

predict(**args*, ***kwargs*)
Predict outputs for the given inputs

Parameters

- **args** (*optional*) – Additional arguments to pass to predict call.
- **kwargs** (*optional*) – Additional keyword arguments to pass to predict call.

Returns Predictions from the given data.

Return type object

save (*path: str*, *fmt: [<class 'str'>, None] = None*, *overwrite_existing: bool = False*) → str
Saves this model

Saves this model's *parameters*, *hyper_params* as well as any other data required to reconstruct this model. Saves this data with the given unique *tag* name.

Parameters

- **path** (`str`) – File path to save the model to.
- **fmt** (`str`) – File output format to use.
- **overwrite_existing** (`bool, optional`) – Whether to overwrite any existing saved model with the same *path* (Default is False).

Returns The path to the saved file.

Return type str

Raises **NotImplementedError** – If the specified *fmt* is not supported.

set_hyper_parameter (*name: str*, *value*) → None
Sets a hyper-parameter value

Sets a hyper-parameter's value if the given *hyper_param* and *value* are valid.

Parameters

- **name** (`str`) – Hyper-parameter to set value for.
- **value** – Value to set.

Raises

- ***MissingParameterException*** – If the given *name* hyper-parameter does not exist.
- ***InvalidParameterException*** – If the given *value* is not valid for the specified hyper-parameter.

See also:

hyper_parameters(), *set_hyper_params()*

set_hyper_params (***hyper_params*) → None
Sets the values of this model's hyper-parameters

Parameters **hyper_params** – Hyper-parameter values to set.

Raises ***InvalidParameterException*** – If one of the given hyper-parameter values is not valid.

set_parameter (*name: str*, *value*) → None
Sets a parameter value

Will add the given *param* and *value* to the parameters if they are valid, throws an exception if they are not.

Parameters

- **name** (`str`) – Parameter to set the value for.
- **value** – Value to set.

Raises ***InvalidParameterException*** – If the given *name* or *value* are not valid.

See also:

parameters()

`set_params`(***params*) → None

Sets the values for this model's parameters

Parameters **`params`** – Parameters and values to set.

Raises *`InvalidParameterException`* – If the given *name* or *value* are not valid.

`unset_hyper_parameter`(*name: str*)

Un-sets a hyper-parameter

Un-sets the specified hyper-parameter's value from the set of hyper-parameters and returns the previously set value.

Parameters **`name`** (*`str`*) – Name of the hyper-parameter to clear the value for.

Returns Previously set value of the hyper-parameter.

Return type object

Raises *`MissingParameterException`* – If the given *name* hyper-parameter does not exist.

See also:

`hyper_parameters()`, *`set_hyper_params()`*

`unset_parameter`(*name: str*) → object

Unsets a parameter value

Removes the specified parameter's value from the parameter values if it is part of the parameter set and returns its current value.

Parameters **`name`** (*`str`*) – Name of the parameter whose value needs to be un-set.

Returns Previously set value of the parameter.

Return type object

Raises *`MissingParameterException`* – If the parameter to remove does not exist in the set of parameters.

See also:

`parameters()`

`class` `spines.`**`Parameter`**(**value_type*, *default=None*, *desc: str = None*)

Bases: `object`

Parameter class

Parameters

- **`value_type`** (`type` or `Iterable` of `type`) – The type(s) of values allowed for this parameter.
- **`default`** (*`object, optional`*) – Default value for this parameter, if any.
- **`desc`** (*`str, optional`*) – Description for this parameter, if any.

`check`(*value*) → bool

Checks the given *value* for validity

Parameters **`value`** – Parameter value to check validity of.

Returns Whether or not the value is valid for the parameter.

Return type bool

default
Default value to use for this parameter.

Type object

desc
Description of this parameter.

Type str

name
Name of this parameter.

Type str

required
Whether or not this parameter is required to be set.

Type bool

value_type
The types of values allowed for this option.

Type tuple

class spines.**HyperParameter**(**value_type*, *default=None*, *desc: str = None*)
Bases: *spines.parameters.base.Parameter*

Hyper-parameter

class spines.**Bounded**(**args*, ***kwargs*)
Bases: *spines.parameters.mixins.Minimum*, *spines.parameters.mixins.Maximum*, *spines.parameters.base.Parameter*

Bounded parameter (min/max)

class spines.**HyperBounded**(**args*, ***kwargs*)
Bases: *spines.parameters.mixins.Minimum*, *spines.parameters.mixins.Maximum*, *spines.parameters.base.HyperParameter*

Bounded hyper-parameter (min/max)

spines.models

Models for the spines library.

class spines.models.**Model**(**args*, ***kwargs*)
Bases: object

Model class

build(**args*, ***kwargs*) → None
Builds the model

Parameters

- **args** (*optional*) – Arguments to use in building the model.
- **kwargs** (*optional*) – Keyword arguments to use in building the model.

error(**args*, ***kwargs*) → float
Returns the error measure of the model for the given data

Parameters

- **args** (*optional*) – Additional arguments to pass to the error call.
- **kwargs** (*optional*) – Additional keyword-arguments to pass to the error call.

Returns Error for the model on the given inputs and outputs.

Return type float

fit(**args*, ***kwargs*) → None

Fits the model

Parameters

- **args** (*optional*) – Arguments to use in fit call.
- **kwargs** (*optional*) – Any additional keyword arguments to use in fit call.

get_hyper_params() → Dict[str, object]

Gets the current hyper-parameter values

Returns Copy of the currently set hyper-parameter values.

Return type dict

See also:

hyper_parameters(), *set_hyper_params()*

get_params() → dict

Gets a copy of this models parameters

Returns Copy of currently set parameter names and values.

Return type dict

hyper_parameters

Hyper-parameters which are currently set.

Type *ParameterStore*

classmethod load(*path: str*, *fmt: [<class 'str'>, None] = None*, *new: bool = False*) → Type[Model]

Loads a saved model instance

Loads saved model *parameters* and *hyper_params* as well as any serialized model-specific objects from a saved version with the *tag* specified (from the base *project_dir*).

Parameters

- **path** (*str*) – Path to load the model from.
- **fmt** (str or None) – Format of the file to load (if None it will be inferred).
- **new** (*bool, optional*) – Whether to create a new object from this class or use the saved object class (default is False).

Returns Model loaded from disk.

Return type *Model*

parameters

Parameters which are currently set.

Type *ParameterStore*

predict(**args*, ***kwargs*)

Predict outputs for the given inputs

Parameters

- **args** (*optional*) – Additional arguments to pass to predict call.
- **kwargs** (*optional*) – Additional keyword arguments to pass to predict call.

Returns Predictions from the given data.

Return type object

save (*path: str*, *fmt: [<class 'str'>, None] = None*, *overwrite_existing: bool = False*) → str
Saves this model

Saves this model's *parameters*, *hyper_params* as well as any other data required to reconstruct this model. Saves this data with the given unique *tag* name.

Parameters

- **path** (*str*) – File path to save the model to.
- **fmt** (*str*) – File output format to use.
- **overwrite_existing** (*bool, optional*) – Whether to overwrite any existing saved model with the same *path* (Default is False).

Returns The path to the saved file.

Return type str

Raises **NotImplementedError** – If the specified *fmt* is not supported.

set_hyper_parameter (*name: str*, *value*) → None
Sets a hyper-parameter value

Sets a hyper-parameter's value if the given *hyper_param* and *value* are valid.

Parameters

- **name** (*str*) – Hyper-parameter to set value for.
- **value** – Value to set.

Raises

- ***MissingParameterException*** – If the given *name* hyper-parameter does not exist.
- ***InvalidParameterException*** – If the given *value* is not valid for the specified hyper-parameter.

See also:

hyper_parameters(), *set_hyper_params()*

set_hyper_params (***hyper_params*) → None
Sets the values of this model's hyper-parameters

Parameters **hyper_params** – Hyper-parameter values to set.

Raises ***InvalidParameterException*** – If one of the given hyper-parameter values is not valid.

set_parameter (*name: str*, *value*) → None
Sets a parameter value

Will add the given *param* and *value* to the parameters if they are valid, throws an exception if they are not.

Parameters

- **name** (*str*) – Parameter to set the value for.
- **value** – Value to set.

Raises *`InvalidParameterException`* – If the given *name* or *value* are not valid.

See also:

`parameters()`

`set_params`(***params*) → None

Sets the values for this model's parameters

Parameters **`params`** – Parameters and values to set.

Raises *`InvalidParameterException`* – If the given *name* or *value* are not valid.

`unset_hyper_parameter`(*name: str*)

Un-sets a hyper-parameter

Un-sets the specified hyper-parameter's value from the set of hyper-parameters and returns the previously set value.

Parameters **`name`** (*`str`*) – Name of the hyper-parameter to clear the value for.

Returns Previously set value of the hyper-parameter.

Return type object

Raises *`MissingParameterException`* – If the given *name* hyper-parameter does not exist.

See also:

`hyper_parameters()`, *`set_hyper_params()`*

`unset_parameter`(*name: str*) → object

Unsets a parameter value

Removes the specified parameter's value from the parameter values if it is part of the parameter set and returns its current value.

Parameters **`name`** (*`str`*) – Name of the parameter whose value needs to be un-set.

Returns Previously set value of the parameter.

Return type object

Raises *`MissingParameterException`* – If the parameter to remove does not exist in the set of parameters.

See also:

`parameters()`

spines.models.base

Base classes for the spines package.

`class` `spines.models.base.`**`Model`**(**args*, ***kwargs*)

Bases: `object`

Model class

`build`(**args*, ***kwargs*) → None

Builds the model

Parameters

- **`args`** (*`optional`*) – Arguments to use in building the model.

- **kwargs** (*optional*) – Keyword arguments to use in building the model.

error(**args*, ***kwargs*) → float

Returns the error measure of the model for the given data

Parameters

- **args** (*optional*) – Additional arguments to pass to the error call.
- **kwargs** (*optional*) – Additional keyword-arguments to pass to the error call.

Returns Error for the model on the given inputs and outputs.

Return type float

fit(**args*, ***kwargs*) → None

Fits the model

Parameters

- **args** (*optional*) – Arguments to use in fit call.
- **kwargs** (*optional*) – Any additional keyword arguments to use in fit call.

get_hyper_params() → Dict[str, object]

Gets the current hyper-parameter values

Returns Copy of the currently set hyper-parameter values.

Return type dict

See also:

hyper_parameters(), *set_hyper_params()*

get_params() → dict

Gets a copy of this models parameters

Returns Copy of currently set parameter names and values.

Return type dict

hyper_parameters

Hyper-parameters which are currently set.

Type *ParameterStore*

classmethod load(*path: str*, *fmt: [<class 'str'>, None] = None*, *new: bool = False*) → Type[Model]

Loads a saved model instance

Loads saved model *parameters* and *hyper_params* as well as any serialized model-specific objects from a saved version with the *tag* specified (from the base *project_dir*).

Parameters

- **path** (*str*) – Path to load the model from.
- **fmt** (`str` or `None`) – Format of the file to load (if `None` it will be inferred).
- **new** (*bool, optional*) – Whether to create a new object from this class or use the saved object class (default is False).

Returns Model loaded from disk.

Return type *Model*

parameters

Parameters which are currently set.

Type *ParameterStore*

predict (**args*, ***kwargs*)

Predict outputs for the given inputs

Parameters

- **args** (*optional*) – Additional arguments to pass to predict call.
- **kwargs** (*optional*) – Additional keyword arguments to pass to predict call.

Returns Predictions from the given data.

Return type object

save (*path: str*, *fmt: [<class 'str'>, None] = None*, *overwrite_existing: bool = False*) → str

Saves this model

Saves this model's *parameters*, *hyper_params* as well as any other data required to reconstruct this model. Saves this data with the given unique *tag* name.

Parameters

- **path** (*str*) – File path to save the model to.
- **fmt** (*str*) – File output format to use.
- **overwrite_existing** (*bool, optional*) – Whether to overwrite any existing saved model with the same *path* (Default is False).

Returns The path to the saved file.

Return type str

Raises **NotImplementedError** – If the specified *fmt* is not supported.

set_hyper_parameter (*name: str*, *value*) → None

Sets a hyper-parameter value

Sets a hyper-parameter's value if the given *hyper_param* and *value* are valid.

Parameters

- **name** (*str*) – Hyper-parameter to set value for.
- **value** – Value to set.

Raises

- ***MissingParameterException*** – If the given *name* hyper-parameter does not exist.
- ***InvalidParameterException*** – If the given *value* is not valid for the specified hyper-parameter.

See also:

hyper_parameters(), *set_hyper_params()*

set_hyper_params (***hyper_params*) → None

Sets the values of this model's hyper-parameters

Parameters **hyper_params** – Hyper-parameter values to set.

Raises ***InvalidParameterException*** – If one of the given hyper-parameter values is not valid.

set_parameter(*name: str*, *value*) → None

Sets a parameter value

Will add the given *param* and *value* to the parameters if they are valid, throws an exception if they are not.

Parameters

- **name** (*str*) – Parameter to set the value for.
- **value** – Value to set.

Raises *InvalidParameterException* – If the given *name* or *value* are not valid.

See also:

parameters()

set_params(***params*) → None

Sets the values for this model's parameters

Parameters **params** – Parameters and values to set.

Raises *InvalidParameterException* – If the given *name* or *value* are not valid.

unset_hyper_parameter(*name: str*)

Un-sets a hyper-parameter

Un-sets the specified hyper-parameter's value from the set of hyper-parameters and returns the previously set value.

Parameters **name** (*str*) – Name of the hyper-parameter to clear the value for.

Returns Previously set value of the hyper-parameter.

Return type object

Raises *MissingParameterException* – If the given *name* hyper-parameter does not exist.

See also:

hyper_parameters(), *set_hyper_params()*

unset_parameter(*name: str*) → object

Unsets a parameter value

Removes the specified parameter's value from the parameter values if it is part of the parameter set and returns its current value.

Parameters **name** (*str*) – Name of the parameter whose value needs to be un-set.

Returns Previously set value of the parameter.

Return type object

Raises *MissingParameterException* – If the parameter to remove does not exist in the set of parameters.

See also:

parameters()

exception spines.models.base.**ModelException**

Bases: `Exception`

Base class for Model exceptions.

spines.models.decorators

Decorators for Models

`spines.models.decorators.`**`finalize_post`**(*store: Type[spines.parameters.store.ParameterStore], func*)

Finalizes the store prior to executing the function

Parameters

- **`store`** (`ParameterStore`) – The parameter store to finalize.
- **`func`** (*`callable`*) – The function to wrap.

Returns The wrapped function.

Return type callable

Raises ***`MissingParameterException`*** – If there's a parameter missing from the required parameters in the given *store*.

`spines.models.decorators.`**`finalize_pre`**(*store: Type[spines.parameters.store.ParameterStore], func*)

Finalizes the store prior to executing the function

Parameters

- **`store`** (`ParameterStore`) – The parameter store to finalize.
- **`func`** (*`callable`*) – The function to wrap.

Returns The wrapped function.

Return type callable

Raises ***`MissingParameterException`*** – If there's a parameter missing from the required parameters in the given *store*.

spines.parameters

Parameters module for parameterized models.

`class` `spines.parameters.`**`Parameter`**(**value_type, default=None, desc: str = None*)

Bases: `object`

Parameter class

Parameters

- **`value_type`** (`type` or `Iterable` of `type`) – The type(s) of values allowed for this parameter.
- **`default`** (*`object, optional`*) – Default value for this parameter, if any.
- **`desc`** (*`str, optional`*) – Description for this parameter, if any.

`check`(*value*) → bool

Checks the given *value* for validity

Parameters **`value`** – Parameter value to check validity of.

Returns Whether or not the value is valid for the parameter.

Return type bool

`default`
Default value to use for this parameter.

Type object

`desc`
Description of this parameter.

Type str

`name`
Name of this parameter.

Type str

`required`
Whether or not this parameter is required to be set.

Type bool

`value_type`
The types of values allowed for this option.

Type tuple

`class` `spines.parameters.`**`HyperParameter`**(**value_type*, *default=None*, *desc: str = None*)
Bases: *`spines.parameters.base.Parameter`*

Hyper-parameter

`class` `spines.parameters.`**`Bounded`**(**args*, ***kwargs*)
Bases: *`spines.parameters.mixins.Minimum`*, *`spines.parameters.mixins.Maximum`*, *`spines.parameters.base.Parameter`*

Bounded parameter (min/max)

`class` `spines.parameters.`**`HyperBounded`**(**args*, ***kwargs*)
Bases: *`spines.parameters.mixins.Minimum`*, *`spines.parameters.mixins.Maximum`*, *`spines.parameters.base.HyperParameter`*

Bounded hyper-parameter (min/max)

`class` `spines.parameters.`**`ParameterStore`**
Bases: `collections.abc.MutableMapping`

Helper class for managing collections of Parameters.

`add`(*parameter: Type[spines.parameters.base.Parameter]*) → None
Add a *`Parameter`* specification to this store

Parameters **`option`** (`Parameter`) – *`Parameter`* specification to add to this parameter store.

Raises **`ParameterExistsError`** – If a parameter option with the same name already exists.

`copy`(*deep: bool = False*) → Type[spines.parameters.store.ParameterStore]
Returns a copy of this parameter store object.

Parameters **`deep`** (*`bool, optional`*) – Whether or not to do deep-copying of this stores contents.

Returns Copied parameter store object.

Return type *ParameterStore*

final
Whethor or not this set of parameters is finalized.

Type bool

finalize() → None
Finalizes the parameters stored

Raises *MissingParameterException* – If a required parameter is not set.

parameters
Copy of the current set of parameters.

Type dict

remove(*name: str*) → spines.parameters.base.Parameter
Removes a *Parameter* specification

Parameters **name** (*str*) – Name of the *Parameter* to remove.

Returns The removed *Parameter* specified.

Return type *Parameter*

Raises **KeyError** – If the given *name* does not exist.

reset() → None
Clears all of the parameters and options stored.

valid
Whether or not this is a fully valid set of parameters.

Type bool

values
Copy of the current set of parameter values.

Type dict

exception spines.parameters.**InvalidParameterException**
Bases: *spines.parameters.base.ParameterException*

Thrown when an invalid parameter is given.

exception spines.parameters.**MissingParameterException**
Bases: *spines.parameters.base.ParameterException*

Thrown when a required parameter is missing.

spines.parameters.base

Base classes for model parameters.

class spines.parameters.base.**HyperParameter**(**value_type*, *default=None*, *desc: str = None*)
Bases: *spines.parameters.base.Parameter*

Hyper-parameter

exception spines.parameters.base.**InvalidParameterException**
Bases: *spines.parameters.base.ParameterException*

Thrown when an invalid parameter is given.

exception spines.parameters.base.**MissingParameterException**

Bases: *spines.parameters.base.ParameterException*

Thrown when a required parameter is missing.

class spines.parameters.base.**Parameter**(**value_type*, *default=None*, *desc: str = None*)

Bases: object

Parameter class

Parameters

- **value_type** (type or Iterable of type) – The type(s) of values allowed for this parameter.
- **default** (*object, optional*) – Default value for this parameter, if any.
- **desc** (*str, optional*) – Description for this parameter, if any.

check(*value*) → bool

Checks the given *value* for validity

Parameters **value** – Parameter value to check validity of.

Returns Whether or not the value is valid for the parameter.

Return type bool

default

Default value to use for this parameter.

Type object

desc

Description of this parameter.

Type str

name

Name of this parameter.

Type str

required

Whether or not this parameter is required to be set.

Type bool

value_type

The types of values allowed for this option.

Type tuple

exception spines.parameters.base.**ParameterException**

Bases: Exception

Base class for Model parameter exceptions.

class spines.parameters.base.**ParameterMixin**

Bases: abc.ABC

Base mixin class for parameters

spines.parameters.core

Parameter classes for use in models.

class `spines.parameters.core.`**`Bounded`**(**args*, ***kwargs*)

Bases: *spines.parameters.mixins.Minimum*, *spines.parameters.mixins.Maximum*, *spines.parameters.base.Parameter*

Bounded parameter (min/max)

class `spines.parameters.core.`**`HyperBounded`**(**args*, ***kwargs*)

Bases: *spines.parameters.mixins.Minimum*, *spines.parameters.mixins.Maximum*, *spines.parameters.base.HyperParameter*

Bounded hyper-parameter (min/max)

spines.parameters.factories

Parameter factory functions

`spines.parameters.factories.`**`bound_mixin`**(*name*, *checker*, *cls_name=None*)

Creates a new mixin class for bounded parameters

This factory function makes creating bound mixins very simple, you only need to provide the *name* for the attribute on the resulting class for this particular boundary condition, and provide a callable *checker* to perform the validation. The checker call needs to look like this:

The call should return True when the value is within the boundary and False when it's not. The *checker* doesn't necessarily have to be a function, it just needs to be callable.

Parameters

- **name** (*str*) – Name of the property holding the bound's value.
- **checker** (*callable*) – Callable object/function to assess the boundary condition.
- **cls_name** (*str, optional*) – Name for the newly created class type (defaults to *name* + 'BoundMixin').

Returns New parameter mixin class for the bound specified.

Return type *ParameterMixin*

Raises **`ValueError`** – If the given *checker* is not callable.

spines.parameters.mixins

Mixin classes for Parameters.

class `spines.parameters.mixins.`**`Maximum`**(**args*, ***kwargs*)

Bases: `abc.MaximumBoundMixin`

Maximum value bound mixin class

`maximum`

Maximum allowed value for this parameter.

Parameters **maximum** (*optional*) – Maximum allowed value for this parameter.

class `spines.parameters.mixins.`**`Minimum`**`(`**args*, ***kwargs*`)`

Bases: `abc.MinimumBoundMixin`

Minimum value bound mixin class

`minimum`

Minimum allowed value for this parameter.

Parameters **`minimum`** (*`optional`*) – Minimum allowed value for this parameter.

spines.parameters.store

Parameter storage module.

class `spines.parameters.store.`**`ParameterStore`**

Bases: `collections.abc.MutableMapping`

Helper class for managing collections of Parameters.

`add``(`*parameter: Type[spines.parameters.base.Parameter]*`)` → None

Add a `Parameter` specification to this store

Parameters **`option`** (`Parameter`) – `Parameter` specification to add to this parameter store.

Raises **`ParameterExistsError`** – If a parameter option with the same name already exists.

`copy``(`*deep: bool = False*`)` → Type[spines.parameters.store.ParameterStore]

Returns a copy of this parameter store object.

Parameters **`deep`** (*`bool, optional`*) – Whether or not to do deep-copying of this stores contents.

Returns Copied parameter store object.

Return type *ParameterStore*

`final`

Whethor or not this set of parameters is finalized.

Type bool

`finalize``()` → None

Finalizes the parameters stored

Raises ***`MissingParameterException`*** – If a required parameter is not set.

`parameters`

Copy of the current set of parameters.

Type dict

`remove``(`*name: str*`)` → spines.parameters.base.Parameter

Removes a `Parameter` specification

Parameters **`name`** (*`str`*) – Name of the `Parameter` to remove.

Returns The removed `Parameter` specified.

Return type *Parameter*

Raises **`KeyError`** – If the given *name* does not exist.

reset() → None
Clears all of the parameters and options stored.

valid
Whether or not this is a fully valid set of parameters.

Type bool

values
Copy of the current set of parameter values.

Type dict

spines.parameters.store.**state_changed**(*func*)
Decorator indicating a function which changes the state

Parameters **func** (*callable*) – The function to wrap.

Returns The wrapped function.

Return type callable

spines.versioning

Versioning sub-package for spines.

spines.versioning.base

Base classes for the spines versioning package.

class spines.versioning.base.**Signature**
Bases: object

Signature objects for component change tracking and management

class spines.versioning.base.**Version**(*name*, *display_name=None*, *desc=None*)
Bases: object

Version objects for versioning of spines components

bump() → None
Bumps this version's PATCH number by one.

bump_dev() → None
Bumps the dev number for use during iterative work.

bump_major() → None
Bumps this version's MAJOR number by one.

bump_minor() → None
Bumps this version's MINOR number by one.

description
Description for the object versioned.

Type str

display_name
Display name for the object versioned.

Type str

name
Name of the object versioned.

Type str

slug
Slugified version of this version object

Type str

tag
Tag (if any) for this version.

Type str

to_dev() → None
Switches the version to development

to_post() → None
Switches the version to post-release

to_pre() → None
Switches the version to pre-release

to_release() → None
Switches the version to release

version
Version string for this version object.

Type str

spines.versioning.core

Core functionality for the spines versioning package.

spines.versioning.core.**get_changes**(*a: [<class 'str'>, typing.List[str]], b: [<class 'str'>, typing.List[str]]*) → List[tuple]
Gets the full set of changes required to go from a to b

Parameters

- **a** (str or list of str) – Text to start from.
- **b** (str or list of str) – Text to get changes to get to.

Returns List of five-tuples of operation, from start index, from end index, to start index and to end index.

Return type list of tuple

spines.versioning.core.**get_diff**(*a: [<class 'str'>, typing.List[str]], b: [<class 'str'>, typing.List[str]], n=3*)
Gets the differences between text data

Parameters

- **a** (str or list of str) – Text to compare from.
- **b** (str or list of str) – Text to compare with.
- **n** (*int, optional*) – Lines of context to show around differences.

Returns Differences between the texts.

Return type str

`spines.versioning.core.get_doc_string(obj)`

Gets the documentation string for the given object

Parameters **obj** (*object*) – Object to get docstring for.

Returns Docstring of the given object.

Return type str

`spines.versioning.core.get_function_source(func)`

Gets the source code for the given function

Parameters **func** (*callable*) – Function to get source code of.

Returns Function source code, properly formatted.

Return type str

`spines.versioning.core.slugify(value: str, allow_unicode: bool = False) → str`

Slugifys the given string

Convert to ASCII if 'allow_unicode' is False. Convert spaces to hyphens. Remove characters that aren't alphanumerics, underscores, or hyphens. Convert to lowercase. Also strip leading and trailing whitespace.

Note: Modified (barely) from Django: https://github.com/django/django/blob/master/django/utils/text.py

Parameters

- **value** (*str*) – String to slugify.
- **allow_unicode** (*bool, optional*) – Whether or not to allow unicode characters.

Returns Slugified string.

Return type str

3.6 Contributing

Contributions to the `spines` package are welcome and credit will be given for any contributions via the `AUTHORS.md` file. There are a few things to note when making contributions:

- This project uses numpy-style docstrings and all code must be thoroughly documented in order for it to be accepted. For more information on the docstring format see the numpydoc docstring guide and numpy docstring examples.
- This project follows the PEP8 standards for code style and all new code must follow the same conventions.
- Please conduct yourself in a polite and respectful manner, in accordance with the project's *code of conduct*.
- Please ensure that any and all contributions made to the project are yours and belong to you, do not violate any agreements or infringe upon the intellectual property of any other individual or organization that you may be bound to.
- This project is licensed under the MIT License (see this page for details), all contributions which are accepted and included in the project will be subject to that license's terms.

First fork the repository to create a copy to make your changes on. Once you've finished your contribution you will need to ensure that the project's unit tests all pass and that the code passes flake8 and coverage tests.

Once you've ensured the code passes all tests you can then submit a pull request on the project's github repository.

3.7 Code of Conduct

It should go without saying but please conduct yourself in a polite and respectful manner when interacting with the project. This project has adopted the code of conduct standards of the Python Software Foundation. Visit their community code of conduct page for more information.

3.8 Authors

All contributors to the project are listed in the `AUTHORS` file (and *contributions* are always welcome). The contents of that file is can also be found here.

> The basic idea for Spines was independently created by Douglas Daly in 2017 and split off into the Spines package in 2019.
>
> Lead Developer:
>
> - Douglas Daly <contact@douglasdaly.com>

3.9 Changelogs

All changes in this project are documented in the `CHANGELOG.md` file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

3.9.1 Changelogs

0.0.1

Released March 31, 2019

Added

- Initial release of alpha version.

0.0.2

Release April 01, 2019

Changed

- Transform and Build (replacing Construct and Predict).
- Documentation updates.

Fixed

- Hotfixes for documentation configuration.

0.0.3

Release April 18, 2019

Updates to documentation.

0.0.4

Release April 18, 2019

Hotfixes.

0.0.5

Release April 18, 2019

Fixed

- Travis config python 3.7 issue.

3.10 License

This project is licensed under the MIT license.

CHAPTER FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX