

---

**aisquared**

**The AI Squared Team**

**Feb 10, 2023**



# DOCUMENTATION

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	aisquared . . . . .	3
1.1.1	aisquared package . . . . .	3
1.1.1.1	Module contents . . . . .	3
1.1.1.2	Subpackages . . . . .	3
<b>2</b>	<b>Changelog</b>	<b>59</b>
	<b>Python Module Index</b>	<b>63</b>
	<b>Index</b>	<b>65</b>





This package contains utilities to interact with the AI Squared technology stack, particularly with developing and deploying models to the AI Squared Platform or other applications developed through the AI Squared JavaScript SDK.

**Current Production Version: 0.3.2**

[View this Documentation in PDF Format.](#)



## INSTALLATION

This package is available through [Pypi](#) and can be installed by running the following command:

```
pip install aisquared
```

Alternatively, the latest version of the software can be installed directly from GitHub using the following command:

```
pip install git+https://github.com/AISquaredInc/aisquared
```

### 1.1 aisquared

#### 1.1.1 aisquared package

##### 1.1.1.1 Module contents

This package contains utilities to interact with the AI Squared technology stack, particularly with developing and deploying models to the AI Squared Browser Extension or other applications developed through the AI Squared JavaScript SDK.

##### 1.1.1.2 Subpackages

##### **aisquared.base package**

##### **Module contents**

The aisquared.base package contains both some basic objects that are used across the aisquared package backend and some objects which are designed to facilitate simple use cases of the technology.

##### **Submodules**

##### **aisquared.base.BaseObject module**

**class** aisquared.base.BaseObject.BaseObject

Bases: object

Base class used for all other classes within the aisquared package. This class is not meant to be used by any end user of this package, but is rather used throughout this package as a parent class.

**to\_dict()** → dict

Get the object as a dictionary

**to\_json()** → str

Return the object as a json string

### aisquared.base.CustomObject module

```
class aisquared.base.CustomObject.CustomObject(class_name: str, top_level_kwargs: Optional[dict] = None, **kwargs)
```

Bases: *BaseObject*

Custom class that allows the user to define custom classes for configuration

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.base.CustomObject(
    'MyClass',
    key1 = 'foo',
    key2 = 'bar'
)
>>> my_obj.to_dict()
{'className': 'MyClass', 'params': {'key1': 'foo', 'key2': 'bar'}}
```

**to\_dict()** → dict

Get the object as a dictionary

### aisquared.base.rendering module

Some allowed configuration parameters - not meant to be directly called by the end user

### aisquared.base.stages module

Some allowed configuration parameters - not designed to be directly called by the user

### aisquared.config package

#### Module contents

The aisquared.config subpackage contains utilities and objects for packaging aisquared configuration steps and models.

For in-depth examples of how to build out .air files using the utilities and classes in this library, please visit our GitHub repository at <https://github.com/AISquaredInc/airFiles>



## Subpackages

### aisquared.config.analytic package

## Module contents

The aisquared.config.analytic subpackage contains objects for packaging individual analytics.

## Submodules

### aisquared.config.analytic.DeployedAnalytic module

**class** aisquared.config.analytic.DeployedAnalytic.**DeployedAnalytic**(*url: str, input\_type: str, secret: str = 'request', header: Optional[dict] = None*)

Bases: *BaseObject*

Interaction with a remote analytic

Example usage:

```
>>> import aisquared
>>> analytic = aisquared.config.analytic.DeployedAnalytic(
    'analytic_url',
    'text'
)
>>> analytic.to_dict()
{'className': 'DeployedAnalytic',
 'params': {'url': 'analytic_url',
 'inputType': 'text',
 'secret': 'request',
 'header': None}}
```

**property** header

**property** input\_type

**property** secret

**to\_dict()** → dict

Get the object as a dictionary

**property** url

**aisquared.config.analytic.DeployedModel module**

**class** aisquared.config.analytic.DeployedModel.**DeployedModel**(url: str, input\_type: str, secret: str = 'request', header: Optional[dict] = None)

Bases: *BaseObject*

Interaction with a remote model

Example usage:

```
>>> import aisquared
>>> analytic = aisquared.config.analytic.DeployedModel(
    'model_url',
    'text'
)
>>> analytic.to_dict()
{'className': 'DeployedModel',
 'params': {'url': 'model_url',
 'inputType': 'text',
 'secret': 'request',
 'header': None}}
```

**property header**

**property input\_type**

**property secret**

**to\_dict()** → dict

Get the config object as a dictionary

**property url**

**aisquared.config.analytic.LocalAnalytic module**

**class** aisquared.config.analytic.LocalAnalytic.**LocalAnalytic**(path: str, input\_type: str, all: bool = False)

Bases: *BaseObject*

Interaction with an analytic (lookup table) saved to the local file system

Example usage:

```
>>> import aisquared
>>> analytic = aisquared.config.analytic.LocalAnalytic(
    'analytic_path',
    'text'
)
>>> analytic.to_dict()
{'className': 'LocalAnalytic',
 'params': {'path': 'analytic_path',
 'inputType': 'text',
 'all': False}}
```

**property** all

**property** input\_type

**property** path

**to\_dict()** → dict

Get the configuration object as a dictionary

### aisquared.config.analytic.LocalModel module

**class** aisquared.config.analytic.LocalModel.**LocalModel**(*path: str, input\_type: str*)

Bases: *BaseObject*

Interaction with a model currently saved to the local file system

Example usage:

```

>>> import aisquared
>>> analytic = aisquared.config.analytic.LocalModel(
    'model_path',
    'text'
)
>>> analytic.to_dict()
{'className': 'LocalModel',
 'params': {'path': 'model_path',
            'inputType': 'text'}}

```

**property** input\_type

**property** path

**to\_dict()** → dict

Get the configuration object as a dictionary

### aisquared.config.analytic.ReverseMLWorkflow module

**class** aisquared.config.analytic.ReverseMLWorkflow.**ReverseMLWorkflow**(*bucket: str, filename: str, column: str, input\_type: str, period: Optional[int] = None, secret: str = ""*)

Bases: *BaseObject*

Interaction with a ReverseML CSV stored in S3

Example usage:

```

>>> import aisquared
>>> analytic = aisquared.config.analytic.ReverseMLWorkflow(
    'bucket_name',
    'file_name',
    'column_name',
    'text'
)

```

(continues on next page)

(continued from previous page)

```
>>> analytic.to_dict()
{'className': 'ReverseMLWorkflow',
 'params': {'bucket': 'bucket_name',
 'fileName': 'file_name',
 'inputType': 'text',
 'column': 'column_name',
 'period': None,
 'secret': ''}}
```

**property bucket**

**property column**

**property filename**

**property input\_type**

**property period**

**property secret**

**to\_dict()** → dict

Get the configuration object as a dictionary

## aisquared.config.feedback package

### Module contents

The `aisquared.config.feedback` subpackage contains objects for configuring feedback in aisquared models.

### Submodules

#### aisquared.config.feedback.BinaryFeedback module

**class** `aisquared.config.feedback.BinaryFeedback`(*label\_map: list*)

Bases: *BaseObject*

Feedback for binary classification

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.feedback.BinaryFeedback(['class1', 'class2'])
>>> my_obj.to_dict()
{'className': 'BinaryFeedback', 'params': {'labelMap': ['class1', 'class2']}}
```

**property label\_map**

**to\_dict()** → dict

Return the object as a dictionary

**aisquared.config.feedback.ModelFeedback module****class** aisquared.config.feedback.ModelFeedback.**ModelFeedback**Bases: *BaseObject*

Feedback object for questions and answers for an individual model.

Example usage:

```

>>> import aisquared
>>> my_obj = aisquared.config.feedback.ModelFeedback()
>>> my_obj.add_question(
    'How is the model performing?',
    choices = ['very poorly', 'poorly', 'neutral', 'well', 'very well']
)
>>> my_obj.add_question(
    'Any additional feedback?',
    'text'
)
>>> my_obj.to_dict()
{'className': 'ModelFeedback',
 'params': {'questions': [{'question': 'How is the model performing?',
 'answerType': 'singleChoice',
 'choices': ['very poorly', 'poorly', 'neutral', 'well', 'very well']},
 {'question': 'Any additional feedback?', 'answerType': 'text'}]}}

```

**add\_question**(*question: str, answer\_type: str = 'singleChoice', choices: list = []*)

Add a question to be asked.

**Parameters**

- **question** (*str*) – The question to be asked.
- **answer\_type** (*str* (default 'singleChoice')) – One of either 'singleChoice', 'multiChoice', or 'text'
- **choices** (*list* (default [])) – The choices to be provided, if *answer\_type* is 'singleChoice' or 'multiChoice'

**to\_dict**() → dict

Return the object as a dictionary

**aisquared.config.feedback.MulticlassFeedback module****class** aisquared.config.feedback.MulticlassFeedback.**MulticlassFeedback**(*label\_map: list*)Bases: *BaseObject*

Feedback for multiclass classification

Example Usage:

```

>>> import aisquared
>>> my_obj = aisquared.config.feedback.MulticlassFeedback(['class1', 'class2',
↪ 'class3'])
>>> my_obj.to_dict()

```

(continues on next page)

(continued from previous page)

```
{'className': 'MulticlassFeedback',  
'params': {'labelMap': ['class1', 'class2', 'class3']}}
```

**property** `label_map`

**to\_dict()** → dict

Return the object as a dictionary

## aisquared.config.feedback.QualitativeFeedback module

**class** `aisquared.config.feedback.QualitativeFeedback.QualitativeFeedback`

Bases: `BaseObject`

Feedback object for questions and answers for individual predictions.

Example usage:

```
>>> import aisquared  
>>> my_obj = aisquared.config.feedback.QualitativeFeedback()  
>>> my_obj.add_question('Any additional feedback?', 'text')  
>>> my_obj.to_dict()  
{'className': 'QualitativeFeedback',  
'params': {'questions': [{'question': 'Any additional feedback?',  
'answerType': 'text'}]}}
```

**add\_question**(*question: str, answer\_type: str = 'singleChoice', choices: list = []*)

Add a question to be asked.

### Parameters

- **question** (*str*) – The question to be asked.
- **answer\_type** (*str* (default 'singleChoice')) – One of either 'singleChoice', 'multiChoice', or 'text'
- **choices** (*list* (default [])) – The choices to be provided, if *answer\_type* is 'singleChoice' or 'multiChoice'

**to\_dict()** → dict

Return the object as a dictionary

## aisquared.config.feedback.RegressionFeedback module

**class** `aisquared.config.feedback.RegressionFeedback.RegressionFeedback`

Bases: `BaseObject`

Feedback for regression

Example usage:

```
>>> import aisquared  
>>> my_obj = aisquared.config.feedback.RegressionFeedback()  
>>> my_obj.to_dict()  
{'className': 'RegressionFeedback', 'params': {}}
```

**to\_dict()** → dict

Return the object as a dictionary

### aisquared.config.feedback.SimpleFeedback module

**class** aisquared.config.feedback.SimpleFeedback.**SimpleFeedback**

Bases: *BaseObject*

Simple thumbs-up/thumbs-down feedback for predictions

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.feedback.SimpleFeedback()
>>> my_obj.to_dict()
{'className': 'SimpleFeedback', 'params': {}}
```

**to\_dict()** → dict

Return the object as a dictionary

### aisquared.config.harvesting package

#### Module contents

The aisquared.config.harvesting subpackage contains objects for configuring harvesting of data.

#### Submodules

### aisquared.config.harvesting.ImageHarvester module

**class** aisquared.config.harvesting.ImageHarvester.**ImageHarvester**(*how: str = 'all'*)

Bases: *BaseObject*

Object to harvest images

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.harvesting.ImageHarvester()
>>> my_obj.to_dict()
{'className': 'ImageHarvester', 'params': {'how': 'all'}}
```

**property** how

**to\_dict()** → dict

Get the configuration object as a dictionary

**aisquared.config.harvesting.InputHarvester module**

```
class aisquared.config.harvesting.InputHarvester.InputHarvester(input_type: str = 'text',  
                                                                max_length: Optional[int] =  
                                                                None, features: Optional[list] =  
                                                                None)
```

Bases: *BaseObject*

Object to harvest user-input text

Example usage:

```
>>> import aisquared  
>>> my_obj = aisquared.config.harvesting.InputHarvester()  
>>> my_obj.to_dict()  
{'className': 'InputHarvester',  
'params': {'inputType': 'text', 'maxLength': None, 'features': None}}
```

**property features**

**property input\_type**

**property max\_length**

**to\_dict()** → dict

Get the configuration object as a dictionary

**aisquared.config.harvesting.QueryParameterHarvester module**

```
class aisquared.config.harvesting.QueryParameterHarvester.QueryParameterHarvester(query_keys:  
                                                                Union[str,  
                                                                list],  
                                                                url_locations:  
                                                                Union[str,  
                                                                list], at-  
                                                                tributes:  
                                                                Union[str,  
                                                                list])
```

Bases: *BaseObject*

Harvester for Query Parameters

Example usage:

```
>>> import aisquared  
>>> my_obj = aisquared.config.harvesting.QueryParameterHarvester(  
    'test_key',  
    'test_url',  
    'test_attribute'  
)  
>>> my_obj.to_dict()  
{'className': 'QueryParameterHarvester',  
'params': {'queryKeys': ['test_key'],  
'urlLocations': ['test_url'],  
'attributes': ['test_attribute']}}
```



**property attributes**

**property query\_keys**

**to\_dict()** → dict

Get the configuration object as a dictionary

**property url\_locations**

## aisquared.config.harvesting.TextHarvester module

```
class aisquared.config.harvesting.TextHarvester.TextHarvester(how: str = 'all', regex: Optional[str] = None, flags: str = 'gu', body_only: bool = False, keywords: Optional[Union[str, list]] = None, limit: Optional[int] = None)
```

Bases: *BaseObject*

Object to harvest text

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.harvesting.TextHarvester(
    how = 'all',
    body_only = True
)
>>> my_obj.to_dict()
{'className': 'TextHarvester',
 'params': {'how': 'all',
 'regex': None,
 'flags': 'gu',
 'bodyOnly': True,
 'limit': None}}
```

**property body\_only**

**property flags**

**property how**

**property limit**

**property regex**

**to\_dict()** → dict

Get the configuration object as a dictionary

## **aisquared.config.postprocessing package**

### **Module contents**

The `aisquared.config.postprocessing` subpackage contains objects for configuring how predictions are postprocessed.

### **Submodules**

#### **aisquared.config.postprocessing.BinaryClassification module**

```
class aisquared.config.postprocessing.BinaryClassification.BinaryClassification(label_map:  
                                                                    list,  
                                                                    threshold:  
                                                                    float = 0.5)
```

Bases: *BaseObject*

Postprocessing configuration object for binary classification

Example usage

```
>>> import aisquared  
>>> my_obj = aisquared.config.postprocessing.BinaryClassification(  
    ['class1', 'class2']  
)  
>>> my_obj.to_dict()  
{'className': 'BinaryClassification',  
 'params': {'labelMap': ['class1', 'class2'], 'threshold': 0.5}}
```

**property** `label_map`

**property** `threshold`

**to\_dict()** → dict

Get the configuration object as a dictionary

#### **aisquared.config.postprocessing.MulticlassClassification module**

```
class aisquared.config.postprocessing.MulticlassClassification.MulticlassClassification(label_map:  
                                                                    list)
```

Bases: *BaseObject*

Postprocessing configuration object for multiclass classification

Example usage:

```
>>> import aisquared  
>>> my_obj = aisquared.config.postprocessing.MulticlassClassification(  
    ['class1', 'class2', 'class3']  
)  
>>> my_obj.to_dict()  
{'className': 'MulticlassClassification',  
 'params': {'labelMap': ['class1', 'class2', 'class3']}}
```

property **label\_map**

**to\_dict()** → dict

Get the configuration object as a dictionary

### aisquared.config.postprocessing.ObjectDetection module

**class** aisquared.config.postprocessing.ObjectDetection.**ObjectDetection**(*label\_map: list, threshold: float = 0.5*)

Bases: *BaseObject*

Postprocessing configuration object for object detection

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.postprocessing.ObjectDetection(
    ['class1', 'class2', 'class3']
)
>>> my_obj.to_dict()
{'className': 'ObjectDetection',
 'params': {'labelMap': ['class1', 'class2', 'class3'], 'threshold': 0.5}}
```

property **label\_map**

property **threshold**

**to\_dict()** → dict

Get the configuration object as a dictionary

### aisquared.config.postprocessing.Regression module

**class** aisquared.config.postprocessing.Regression.**Regression**(*min: Optional[Union[int, float]] = None, max: Optional[Union[int, float]] = None, round: bool = False*)

Bases: *BaseObject*

Postprocessing configuration object for Regression

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.postprocessing.Regression(
    10,
    100
)
>>> my_obj.to_dict()
{'className': 'Regression', 'params': {'min': 10, 'max': 100, 'round': False}}
```

property **max**

property **min**

property **round**

`to_dict()` → dict

Get the configuration object as a dictionary

### aisquared.config.preprocessing package

#### Module contents

The `aisquared.config.preprocessing` subpackage contains utilities to configure the preprocessing of data in the data pipeline. It contains

three separate subpackages, `aisquared.config.preprocessing.text`, `aisquared.config.preprocessing.image`, and `aisquared.config.preprocessing.tabular`, which configure the preprocessing of different types of data.

#### Subpackages

### aisquared.config.preprocessing.image package

#### Module contents

The `aisquared.config.preprocessing.image` subpackage contains objects for configuring image preprocessing.

#### Submodules

### aisquared.config.preprocessing.image.ImagePreprocessing module

**class** `aisquared.config.preprocessing.image.ImagePreprocessing.ImagePreprocessor` (*steps: Optional[list] = None*)

Bases: `BaseObject`

Preprocessor object for image data

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.image.ImagePreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.image.AddValue(255.0)
)
```

**add\_step**(*step*)

Add a step to the preprocessor object

**property** `step_dict`

`to_dict()` → dict

Get the configuration object as a dictionary

## aisquared.config.preprocessing.image.Steps module

**class** aisquared.config.preprocessing.image.Steps.**AddValue**(value: Union[int, float])

Bases: *BaseObject*

Preprocessing step to add a value to all pixels in an image

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.image.ImagePreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.image.AddValue(255.0)
)
```

**to\_dict()** → dict

Get the configuration object as a dictionary

**property value**

**class** aisquared.config.preprocessing.image.Steps.**ConvertToColor**(color: str)

Bases: *BaseObject*

Preprocessing step to convert images to a color scheme

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.image.ImagePreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.image.ConvertToColor('RGB')
)
```

**property color**

**to\_dict()** → dict

Get the configuration object as a dictionary

**class** aisquared.config.preprocessing.image.Steps.**DivideValue**(value: Union[int, float])

Bases: *BaseObject*

Preprocessing step to divide all pixels in an image by a value

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.image.ImagePreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.image.DivideValue(255.0)
)
```

**to\_dict()** → dict

Get the configuration object as a dictionary

**property value**

**class** aisquared.config.preprocessing.image.Steps.**MultiplyValue**(value: Union[int, float])

Bases: *BaseObject*

Preprocessing step to multiply all pixels in an image by a value

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.image.ImagePreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.image.MultiplyValue(2.0)
)
```

**to\_dict()** → dict

Get the configuration object as a dictionary

**property value**

**class** aisquared.config.preprocessing.image.Steps.**Resize**(size: list, method: str = 'bilinear',  
preserve\_aspect\_ratio: bool = False)

Bases: *BaseObject*

Preprocessing step to resize an image

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.image.ImagePreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.image.Resize([100, 100])
)
```

**property method**

**property preserve\_aspect\_ratio**

**property size**

**to\_dict()** → dict

Get the configuration object as a dictionary

**class** aisquared.config.preprocessing.image.Steps.**SubtractValue**(value: Union[int, float])

Bases: *BaseObject*

Preprocessing step to subtract a value from all pixels in an image

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.image.ImagePreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.image.SubtractValue(255.0)
)
```

**to\_dict()** → dict

Get the configuration object as a dictionary

**property value**

## aisquared.config.preprocessing.tabular package

### Module contents

The aisquared.config.preprocessing.tabular subpackage contains objects for preprocessing tabular data.

### Submodules

#### aisquared.config.preprocessing.tabular.Steps module

**class** aisquared.config.preprocessing.tabular.Steps.**DropColumn**(column: int)

Bases: *BaseObject*

Drop a column from tabular data

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.tabular.TabularPreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.tabular.DropColumn(
        3
    )
)
```

**property** column

**to\_dict**() → dict

Get the configuration object as a dictionary

**class** aisquared.config.preprocessing.tabular.Steps.**MinMax**(mins: list, maxs: list, columns: Optional[list] = None)

Bases: *BaseObject*

Min-Max Scaling preprocessing step

Min-Max Scaling takes all associated columns and maps values relative to the minimum and maximum values of the training data.

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.tabular.TabularPreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.tabular.MinMax(
        [0, 1.1, 2],
        [0.2, 14, 18.3]
    )
)
```

**property** columns

**property** maxs

**property mins**

**to\_dict()** → dict

Get the configuration object as a dictionary

**class** aisquared.config.preprocessing.tabular.Steps.**OneHot**(*column: int, values: list*)

Bases: *BaseObject*

One Hot encoding preprocessing step

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.tabular.TabularPreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.tabular.OneHot(
        6,
        ['one', 'two', 'three']
    )
)
```

**property column**

**to\_dict()** → dict

Get the configuration object as a dictionary

**property values**

**class** aisquared.config.preprocessing.tabular.Steps.**ZScore**(*means: list, stds: list, columns: Optional[Union[int, list]] = None*)

Bases: *BaseObject*

Z-Score normalization preprocessing step

Z-Score normalization takes each supplied column value, subtracts that column's provided mean, and divides by the provided standard deviation.

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.tabular.TabularPreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.tabular.ZScore(
        [0, 1, 2],
        [0.2, 0.4, 0.6]
    )
)
```

**property columns**

**property means**

**property stds**

**to\_dict()** → dict

Get the configuration object as a dictionary



**aisquared.config.preprocessing.tabular.TabularPreprocessing module**

**class** aisquared.config.preprocessing.tabular.TabularPreprocessing.**TabularPreprocessor**(*steps: Optional[list] = None*)

Bases: *BaseObject*

Preprocessor object for tabular data

Example usage:

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.tabular.TabularPreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.tabular.ZScore(
        [0, 1, 2],
        [0.2, 0.4, 0.6]
    )
)
```

**add\_step**(*step*)

Add a step to the preprocessor object

**to\_dict**()

Get the configuration object as a dictionary

**aisquared.config.preprocessing.text package****Module contents**

The aisquared.config.preprocessing.text subpackage contains objects for preprocessing text data.

**Submodules****aisquared.config.preprocessing.text.Steps module**

**class** aisquared.config.preprocessing.text.Steps.**ConvertToCase**(*lowercase: bool = True*)

Bases: *BaseObject*

Text preprocessing object to convert inputs to all lowercase or all uppercase

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.text.TextPreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.text.ConvertToCase()
)
```

**property lowercase**

**to\_dict()** → dict

Get the configuration object as a dictionary

```
class aisquared.config.preprocessing.text.Steps.ConvertToVocabulary(vocabulary: dict,  
                                                                start_character: int = 1,  
                                                                oov_character: int = 2,  
                                                                max_vocab: Optional[int]  
                                                                = None)
```

Bases: *BaseObject*

Text preprocessing object to convert tokens to integer vocabularies

Example usage:

```
>>> import aisquared  
>>> preprocessor = aisquared.config.preprocessing.text.TextPreprocessor()  
>>> preprocessor.add_step(  
    aisquared.config.preprocessing.text.ConvertToVocabulary(  
        {  
            'test' : 3,  
            'vocabulary' : 4  
        }  
    )  
)
```

**property max\_vocab**

**property oov\_character**

**property start\_character**

**to\_dict()** → dict

Get the configuration object as a dictionary

**property vocabulary**

```
class aisquared.config.preprocessing.text.Steps.PadSequences(pad_character: int = 0, length: int =  
                                                            128, pad_location: str = 'post',  
                                                            truncate_location: str = 'post')
```

Bases: *BaseObject*

Text preprocessing object to pad sequences

Example usage:

```
>>> import aisquared  
>>> preprocessor = aisquared.config.preprocessing.text.TextPreprocessor()  
>>> preprocessor.add_step(  
    aisquared.config.preprocessing.text.PadSequences()  
)
```

**property length**

**property pad\_character**

**property pad\_location**

**to\_dict()** → dict

Get the configuration object as a dictionary

**property truncate\_location**

```
class aisquared.config.preprocessing.text.Steps.RemoveCharacters(remove_digits: bool = True,
                                                                remove_punctuation: bool =
                                                                True)
```

Bases: *BaseObject*

Preprocessing step to remove characters from text

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.text.TextPreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.text.RemoveCharacters()
)
```

**property remove\_digits**

**property remove\_punctuation**

**to\_dict()** → dict

Get the configuration object as a dictionary

```
class aisquared.config.preprocessing.text.Steps.Tokenize(split_sentences: bool = False,
                                                           split_words: bool = True, token_pattern:
                                                           str = '\x08\\w\\w+\x08')
```

Bases: *BaseObject*

Preprocessing Step to tokenize text

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.text.TextPreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.text.Tokenize()
)
```

**property split\_sentences**

**property split\_words**

**to\_dict()** → dict

Get the configuration object as a dictionary

**property token\_pattern**

```
class aisquared.config.preprocessing.text.Steps.Trim
```

Bases: *BaseObject*

Text preprocessing class to trim whitespace from text

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.text.TextPreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.text.Trim()
)
```

**to\_dict()** → dict

Get the configuration object as a dictionary

## **aisquared.config.preprocessing.text.TextPreprocessing module**

**class** aisquared.config.preprocessing.text.TextPreprocessing.**TextPreprocessor**(*steps:*  
*Optional[list] =*  
*None*)

Bases: *BaseObject*

Preprocessor object for natural language

Example usage:

```
>>> import aisquared
>>> preprocessor = aisquared.config.preprocessing.text.TextPreprocessor()
>>> preprocessor.add_step(
    aisquared.config.preprocessing.text.Tokenize()
)
```

**add\_step**(*step*)

Add a step to the preprocessor object

**property** **step\_dict**

**to\_dict()** → dict

Get the configuration object as a dictionary

## **aisquared.config.rendering package**

### **Module contents**

The aisquared.config.rendering subpackage contains objects for configuring how rendering of predictions is to occur.

### **Submodules**

#### **aisquared.config.rendering.BarChartRendering module**

```
class aisquared.config.rendering.BarChartRendering.BarChartRendering(label: str, id: str,
                                                                    chart_name: str,
                                                                    container_id: str,
                                                                    prediction_name_key: str,
                                                                    prediction_value_key: str,
                                                                    prediction_name_value:
                                                                    str, display_legend: bool,
                                                                    legend_icon: str,
                                                                    labels_key: Optional[str]
                                                                    = None, width: str =
                                                                    'auto', height: str = 'auto',
                                                                    xOffset: str = '0', yOffset:
                                                                    str = '0', labels:
                                                                    Optional[list] = None,
                                                                    consolidate_rows: bool =
                                                                    True)
```

Bases: [BaseObject](#)

Rendering class for rendering a Bar Chart

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.rendering.BarChartRendering(
    'my_label',
    'my_id',
    'my_bar_chart',
    'my_container_id',
    'name',
    'value',
    'name_value',
    True,
    'circle'
)
>>> my_obj.to_dict()
{'className': 'BarChartRendering',
 'label': 'my_label',
 'params': {'id': 'my_id',
 'chartName': 'my_bar_chart',
 'containerId': 'my_container_id',
 'displayLegend': True,
 'legendIcon': 'circle',
 'width': 'auto',
 'height': 'auto',
 'xOffset': '0',
 'yOffset': '0',
 'datasource': [{ 'labels': None,
 'labelsKey': None,
 'consolidateRows': True,
 'predictionNameKey': 'name',
 'predictionValueKey': 'value',
 'predictionNameValue': 'name_value' } ]}}
```

**to\_dict()** → dict

Get the configuration object as a dictionary

**aisquared.config.rendering.ContainerRendering module**

```
class aisquared.config.rendering.ContainerRendering.ContainerRendering(label: str, id: str,  
query_selector: str,  
position: str =  
'absolute',  
static_position:  
Optional[str] = None,  
width: str = 'auto',  
height: str = 'auto',  
display: str = 'flex',  
xOffset: str = '0',  
yOffset: str = '0',  
orientation: str =  
'column')
```

Bases: *BaseObject*

Rendering for a container

Example usage:

```
>>> import aisquared  
>>> my_obj = aisquared.config.rendering.ContainerRendering(  
    'my container',  
    'myContainerID',  
    "[data-id='tabpanel-general']"  
)  
>>> my_obj.to_dict()  
{'className': 'ContainerRendering',  
 'label': 'my container',  
 'params': {'id': 'myContainerID',  
 'width': 'auto',  
 'height': 'auto',  
 'display': 'flex',  
 'xOffset': '0',  
 'yOffset': '0',  
 'position': 'absolute',  
 'orientation': 'column',  
 'querySelector': "[data-id='tabpanel-general']",  
 'staticPosition': None}}
```

**property display**

**property height**

**property id**

**property label**

**property orientation**

**property position**

**property query\_selector**

**property static\_position**

**to\_dict()** → dict

Get the configuration object as a dictionary

**property width**

**property xOffset**

**property yOffset**

## aisquared.config.rendering.DashboardReplacementRendering module

```
class aisquared.config.rendering.DashboardReplacementRendering(anchor_selector,
                                                             str,
                                                             where_replace,
                                                             str
                                                             =
                                                             ",
                                                             la-
                                                             bel:
                                                             str
                                                             =
                                                             ")
```

Bases: *BaseObject*

Rendering for dashboard replacement

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.rendering.DashboardReplacementRendering(
    'test_anchor_selector'
)
>>> my_obj.to_dict()
{'className': 'DashboardReplacementRendering',
 'label': '',
 'params': {'anchorSelector': 'test_anchor_selector', 'whereReplace': ''}}
```

**property anchor\_selector**

**property label**

**to\_dict()** → dict

Get the configuration object as a dictionary

**property where\_replace**

**aisquared.config.rendering.DocumentRendering module**

```
class aisquared.config.rendering.DocumentRendering.DocumentRendering(prediction_key: str =  
    'className', words:  
    Optional[Union[list, dict,  
    str]] = None, documents:  
    Optional[Union[list, dict,  
    str]] = None,  
    include_probability: bool  
    = False, probability_key:  
    str = 'probability',  
    underline_color: str =  
    'blue', classes:  
    Optional[list] = None,  
    threshold_key:  
    Optional[str] = None,  
    threshold_value:  
    Optional[Union[int,  
    float]] = None)
```

Bases: *BaseObject*

Object which dictates how to render predictions on entire documents

Example usage:

```
>>> import aisquared  
>>> my_obj = aisquared.config.rendering.DocumentRendering()  
>>> my_obj.to_dict()  
{'className': 'DocumentRendering',  
'params': {'predictionKey': 'className',  
'words': None,  
'documents': None,  
'includeProbability': False,  
'probabilityKey': 'probability',  
'underlineColor': 'blue',  
'classes': None,  
'thresholdKey': None,  
'thresholdValue': None}}
```

**property** classes

**property** documents

**property** include\_probability

**property** prediction\_key

**property** probability\_key

**property** threshold\_key

**property** threshold\_value

**to\_dict()** → dict

Get the configuration object as a dictionary



property underline\_color

property words

## aisquared.config.rendering.DoughnutChartRendering module

```
class aisquared.config.rendering.DoughnutChartRendering.DoughnutChartRendering(label: str, id: str,
chart_name: str,
container_id: str, predic-
tion_name_key: str, predic-
tion_value_key: str, predic-
tion_name_value: str, dis-
play_legend: bool,
legend_icon: str,
labels_key: Optional[str]
= None,
width: str = 'auto', height: str = 'auto',
xOffset: str = '0', yOffset: str = '0',
labels: Op-
tional[list] = None,
consoli-
date_rows: bool = True)
```

Bases: [BaseObject](#)

Rendering class for rendering a Doughnut Chart

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.rendering.DoughnutChartRendering(
    'my_label',
    'my_id',
    'my_doughnut_chart',
    'my_container_id',
    'name',
    'value',
    'name_value',
    True,
```

(continues on next page)

(continued from previous page)

```

        'circle'
    )
    >>> my_obj.to_dict()
    {'className': 'DoughnutChartRendering',
     'label': 'my_label',
     'params': {'id': 'my_id',
                'chartName': 'my_doughnut_chart',
                'containerId': 'my_container_id',
                'displayLegend': True,
                'legendIcon': 'circle',
                'width': 'auto',
                'height': 'auto',
                'xOffset': '0',
                'yOffset': '0',
                'datasource': [{'labels': None,
                                'labelsKey': None,
                                'consolidateRows': True,
                                'predictionNameKey': 'name',
                                'predictionValueKey': 'value',
                                'predictionNameValue': 'name_value'}]}}
```

**to\_dict()** → dict

Get the configuration object as a dictionary

## aisquared.config.rendering.FilterRendering module

**class** aisquared.config.rendering.FilterRendering.**FilterRendering**(*source: str, key: str, qualifier: str, value: Union[list, str, int, float]*)

Bases: *BaseObject*

Object which dictates how predictions are to be passed to downstream analytics

Example usage:

```

>>> import aisquared
>>> my_obj = aisquared.config.rendering.FilterRendering(
    'inputs',
    'key',
    'gt',
    0.2
)
>>> my_obj.to_dict()
{'className': 'FilterRendering',
 'params': {'source': 'inputs', 'key': 'key', 'qualifier': 'gt', 'value': 0.2}}
```

**property** key

**property** qualifier

**property** source

**to\_dict()** → dict

Get the configuration object as a dictionary

**property value**

## aisquared.config.rendering.HTMLTagRendering module

```
class aisquared.config.rendering.HTMLTagRendering.HTMLTagRendering(label: str, id: str,
                                                                    container_id: str,
                                                                    html_content: str,
                                                                    extra_content_tag: str,
                                                                    injection_action: str,
                                                                    prediction_name_key: str,
                                                                    prediction_value_key: str,
                                                                    prediction_name_value: str,
                                                                    content: str = ")

```

Bases: *BaseObject*

Rendering for HTML tags

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.rendering.HTMLTagRendering(
    'my HTML tag',
    'MyHTMLTagRenderingID',
    'MyContainerID',
    '<p>Example Text</p>',
    'extra_tag',
    'append',
    'name_key',
    'value_key',
    'name_value'
)
>>> my_obj.to_dict()
{'className': 'HTMLTagRendering',
 'label': 'my HTML tag',
 'params': {'id': 'MyHTMLTagRenderingID',
            'containerId': 'MyContainerID',
            'htmlContent': '<p>Example Text</p>',
            'extraContentTag': 'extra_tag',
            'injectionAction': 'append',
            'predictionNameKey': 'name_key',
            'predictionValueKey': 'value_key',
            'predictionNameValue': 'name_value',
            'content': ' '}}
```

**to\_dict()** → dict

Return the configuration object as a dictionary

**aisquared.config.rendering.ImageRendering module**

```
class aisquared.config.rendering.ImageRendering.ImageRendering(color: str = 'blue', thickness: str = '5', placement: str = 'bottomleft', include_probability: bool = False, badge_color: str = 'white', font_color: str = 'black', font_size: str = '5', classes: Optional[list] = None, threshold_key: Optional[str] = None, threshold_value: Optional[Union[int, float]] = None)
```

Bases: *BaseObject*

Object which dictates how to render images

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.rendering.ImageRendering()
>>> my_obj.to_dict()
{'className': 'ImageRendering',
 'params': {'color': 'blue',
 'thickness': '5',
 'placement': 'bottomleft',
 'includeProbability': False,
 'badgeColor': 'white',
 'fontColor': 'black',
 'fontSize': '5',
 'classes': None,
 'thresholdKey': None,
 'thresholdValue': None}}
```

**property** badge\_color

**property** classes

**property** color

**property** font\_color

**property** font\_size

**property** include\_probability

**property** placement

**property** thickness

**property** threshold\_key

**property** threshold\_value

**to\_dict()** → dict

Get the configuration object as a dictionary

**aisquared.config.rendering.LineChartRendering module**

```
class aisquared.config.rendering.LineChartRendering(label: str, id: str,
                                                    chart_name: str,
                                                    container_id: str,
                                                    prediction_name_key:
                                                    str,
                                                    prediction_value_key:
                                                    str, predic-
                                                    tion_name_value: str,
                                                    display_legend: bool,
                                                    legend_icon: str,
                                                    labels_key: str, width:
                                                    str = 'auto', height: str =
                                                    'auto', xOffset: str =
                                                    '0', yOffset: str = '0',
                                                    labels: Optional[list] =
                                                    None,
                                                    consolidate_rows: bool
                                                    = True)
```

Bases: *BaseObject*

Rendering class for rendering a Line Chart

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.rendering.LineChartRendering(
    'my_label',
    'my_id',
    'my_line_chart',
    'my_container_id',
    'name',
    'value',
    'name_value',
    True,
    'circle',
    'labels'
)
>>> my_obj.to_dict()
{'className': 'LineChartRendering',
 'label': 'my_label',
 'params': {'id': 'my_id',
 'chartName': 'my_line_chart',
 'containerId': 'my_container_id',
 'displayLegend': True,
 'legendIcon': 'circle',
 'width': 'auto',
 'height': 'auto',
 'xOffset': '0',
 'yOffset': '0',
 'datasource': [{'labels': None,
 'labelsKey': 'labels',
 'consolidateRows': True,
```

(continues on next page)



**aisquared.config.rendering.PieChartRendering module**

```
class aisquared.config.rendering.PieChartRendering(label: str, id: str,
chart_name: str,
container_id: str,
prediction_name_key: str,
prediction_value_key: str,
prediction_name_value:
str, display_legend: bool,
legend_icon: str,
labels_key: Optional[str]
= None, width: str =
'auto', height: str = 'auto',
xOffset: str = '0', yOffset:
str = '0', labels:
Optional[list] = None,
consolidate_rows: bool =
True)
```

Bases: *BaseObject*

Rendering class for rendering a Pie Chart

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.rendering.PieChartRendering(
    'my_label',
    'my_id',
    'my_doughnut_chart',
    'my_container_id',
    'name',
    'value',
    'name_value',
    True,
    'circle'
)
>>> my_obj.to_dict()
{'className': 'PieChartRendering',
 'label': 'my_label',
 'params': {'id': 'my_id',
 'chartName': 'my_doughnut_chart',
 'containerId': 'my_container_id',
 'displayLegend': True,
 'legendIcon': 'circle',
 'width': 'auto',
 'height': 'auto',
 'xOffset': '0',
 'yOffset': '0',
 'datasource': [{'labels': None,
 'labelsKey': None,
 'consolidateRows': True,
 'predictionNameKey': 'name',
 'predictionValueKey': 'value',
 'predictionNameValue': 'name_value'}]}}
```

**to\_dict()** → dict

Get the configuration object as a dictionary

### aisquared.config.rendering.SOSRendering module

**class** aisquared.config.rendering.SOSRendering.SOSRendering(*can\_toggle: bool, label: str = ""*)

Bases: *BaseObject*

Rendering of an SOS dashboard

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.rendering.SOSRendering(True)
>>> my_obj.to_dict()
{'className': 'SOSRendering', 'label': '', 'params': {'canToggle': True}}
```

**property** can\_toggle

**property** label

**to\_dict()** → dict

Get the configuration object as a dictionary

### aisquared.config.rendering.TableRendering module

**class** aisquared.config.rendering.TableRendering.TableRendering(*label: str, id: str, container\_id: str, prediction\_name\_key: str, prediction\_value\_key: str, prediction\_name\_values: str, table\_name: str = ""*)

Bases: *BaseObject*

Class for rendering tables

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.rendering.TableRendering(
    'my table',
    'MyTableID',
    'MyContainerID',
    'name_key',
    'value_key',
    'name_values'
)
>>> my_obj.to_dict()
{'className': 'TableRendering',
 'label': 'my table',
 'params': {'id': 'MyTableID',
            'containerId': 'MyContainerID',
            'predictionNameKey': 'name_key',
            'predictionValueKey': 'value_key',
```

(continues on next page)



(continued from previous page)

```
'predictionNameValues': 'name_values',
'tableName': '{}'
```

**to\_dict()** → dict

Get the configuration object as a dictionary

## aisquared.config.rendering.WordRendering module

```
class aisquared.config.rendering.WordRendering.WordRendering(word_list: str = 'input', result_key:
    Optional[str] = None, content_key:
    Optional[str] = None, badge_shape:
    str = 'star', badge_color: str = 'blue',
    classes: Optional[list] = None,
    threshold_key: Optional[str] =
    None, threshold_value:
    Optional[Union[int, float]] = None)
```

Bases: *BaseObject*

Object for rendering badges on individual words

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.rendering.WordRendering()
>>> my_obj.to_dict()
{'className': 'WordRendering',
'params': {'wordList': 'input',
'resultKey': None,
'contentKey': None,
'badgeShape': 'star',
'badgeColor': 'blue',
'classes': None,
'thresholdKey': None,
'thresholdValue': None}}
```

**property badge\_color**

**property badge\_shape**

**property classes**

**property content\_key**

**property result\_key**

**property threshold\_key**

**property threshold\_value**

**to\_dict()** → dict

Get the configuration object as a dictionary

**property word\_list**

## Submodules

### aisquared.config.GraphConfiguration module

```
class aisquared.config.GraphConfiguration.GraphConfiguration(name: str, stage: str =
                                                            'experimental', version:
                                                            Optional[int] = None, description:
                                                            str = "", mlflow_uri: Optional[str] =
                                                            None, mlflow_user: Optional[str] =
                                                            None, mlflow_token: Optional[str] =
                                                            None, owner: Optional[str] = None,
                                                            url: str = '*', auto_run: bool =
                                                            False)
```

Bases: [BaseObject](#)

Configuration object for deploying a set of processing steps and/or analytics as a dependency graph

**add\_node**(step: [BaseObject](#), dependencies: Optional[Union[int, list]] = None) → int

Add a node to the configuration graph

#### Parameters

- **step** (*aisquared configuration step*) – The step to add
- **dependencies** (*int, list of int, or None*) – The ids of nodes which must be run before the added node

#### Returns

**node\_id** – The integer id of the node that is added

#### Return type

int

#### property auto\_run

**compile**(filename: Optional[str] = None, dtype: Optional[str] = None) → None

Compile the object into a '.air' file, which can then be dragged and dropped into applications using the AI Squared JavaScript SDK

#### Parameters

- **filename** (*path-like or None (default None)*) – Filename to compile to. If None, defaults to '{NAME}.air', where {NAME} is the name of the analytic
- **dtype** (*str or None (default None)*) – The datatype to use for the model weights when using a Keras model. If None, defaults to 'float32'

#### property description

**get\_filenames**() → list

Get filenames for all models in the configuration

#### property mlflow\_token

#### property mlflow\_uri

#### property mlflow\_user

#### property name

**property owner**

**property stage**

**to\_dict()** → dict

Get the object as a dictionary

**property url**

**property version**

## aisquared.config.ModelConfiguration module

```
class aisquared.config.ModelConfiguration.ModelConfiguration(name: str, harvesting_steps:
    Optional[Union[BaseObject, list]] =
    None, preprocessing_steps:
    Optional[Union[BaseObject, list]] =
    None, analytic:
    Optional[Union[BaseObject, list]] =
    None, postprocessing_steps:
    Optional[Union[BaseObject, list]] =
    None, rendering_steps:
    Optional[Union[BaseObject, list]] =
    None, feedback_steps:
    Optional[Union[BaseObject, list]] =
    None, stage: str = 'experimental',
    version: Optional[int] = None,
    description: str = "", mlflow_uri:
    Optional[str] = None, mlflow_user:
    Optional[str] = None, mlflow_token:
    Optional[str] = None, owner:
    Optional[str] = None, url: str = '*',
    auto_run: bool = False)
```

Bases: *BaseObject*

Configuration object for deploying a model or analytic

**property analytic**

**property analytic\_dict**

**property auto\_run**

**compile**(filename: Optional[str] = None, dtype: Optional[str] = None) → None

Compile the object into a '.air' file, which can then be dragged and dropped into applications using the AI Squared JavaScript SDK

### Parameters

- **filename** (*path-like or None (default None)*) – Filename to compile to. If None, defaults to '{NAME}.air', where {NAME} is the name of the analytic
- **dtype** (*str or None (default None)*) – The datatype to use for the model weights. If None, defaults to 'float32'

**property description**

**property** `feedback_dict`  
**property** `feedback_steps`  
**get\_model\_filenames()** → list  
Get filenames for all models in the configuration  
**property** `harvester_dict`  
**property** `harvesting_steps`  
**property** `mlflow_token`  
**property** `mlflow_uri`  
**property** `mlflow_user`  
**property** `name`  
**property** `owner`  
**property** `postprocessor_dict`  
**property** `postprocessing_steps`  
**property** `preprocessor_dict`  
**property** `preprocessing_steps`  
**property** `render_dict`  
**property** `rendering_steps`  
**property** `stage`  
**to\_dict()** → dict  
Get the object as a dictionary  
**property** `url`  
**property** `version`

## **aisquared.logging package**

### **Module contents**

The `aisquared.logging` subpackage contains utilities for performing experiments within `aisquared`.

This functionality is inherited from MLFlow. Please see the MLFlow documentation at <https://mlflow.org>.

## aisquared.platform package

### Module contents

Utilities for interacting with the AI Squared Platform.

The primary class within this subpackage is the *AISquaredPlatformClient* class, which has the capabilities to interact with much of the functionality in the AI Squared platform. For more information about this class, please see its documentation.

### Submodules

#### aisquared.platform.AISquaredPlatformClient module

**exception** aisquared.platform.AISquaredPlatformClient.AISquaredAPIException

Bases: Exception

**class** aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient

Bases: object

Client for interacting with the AI Squared platform programmatically

When using the client for the first time, it is important to run the *client.login()* method. When doing so, the client will ask for any required information interactively.

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> # If you have never logged in before, run the following code:
>>> client.login()
>>> # Test connection
>>> client.test_connection()
Connection successful
200
```

**add\_users\_to\_group**(group\_id: str, user\_ids: list, port: int = 8086) → bool

Add users to a group

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.add_users_to_group('group_id', ['user_id_1', 'user_id_2'])
True
```

#### Parameters

- **group\_id** (str) – The group to add the users to
- **user\_ids** (list of str) – The IDs of the users to add
- **port** (int (default 8086)) – The API port to use

#### Returns

**success** – Returns True if operation was successful

#### Return type

bool

**property base\_url:** str

The base URL associated with the client

**create\_group**(group\_name: str, role\_id: str, port: int = 8086) → dict

Create a group in the platform

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.create_group(
    'group_name',
    'role_id'
)
*dictionary containing group information*
```

#### Parameters

- **group\_name** (str) – The display name of the group
- **role\_id** (str) – The role ID for the group
- **port** (int (default 8086)) – The API port to use

#### Returns

**group\_info** – Metadata about the created group

#### Return type

dict

**create\_user**(user\_name: str, given\_name: str, family\_name: str, email: str, role\_id: str, active: bool = True, middle\_name: Optional[str] = None, company\_id: Optional[str] = None, password: Optional[str] = None, port: int = 8085) → dict

Create a user within the platform

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.create_user(
    'user name',
    'given_name',
    'family_name',
    'user_email',
    'role_id'
)
*Dictionary with user information*
```

#### Parameters

- **user\_name** (str) – The display name of the user
- **given\_name** (str) – The user's first name
- **family\_name** (str) – The user's last name
- **email** (str) – The user's email
- **role\_id** (str) – The ID of the role to be given to the user
- **active** (bool (default True)) – Whether the user is active
- **middle\_name** (str or None (default None)) – The user's middle name

- **company\_id** (*str or None (default None)*) – The user’s company ID
- **password** (*str or None (default None)*) – The user’s password
- **port** (*int (default 8085)*) – The API port to use

**Returns**

**user\_data** – Metadata about the user

**Return type**

dict

**delete\_group**(*group\_id, port=8086*) → bool

Delete a group from the platform

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.delete_group('group_id')
True
```

**Parameters**

- **group\_id** (*str*) – The ID of the group to delete
- **port** (*int (default 8086)*) – The API port to use

**Returns**

**result** – Returns True if successful

**Return type**

bool

**delete\_model**(*id: str, port: int = 8080*) → bool

Delete a model

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.delete_model('model_id')
True
```

**Parameters**

- **id** (*str*) – The ID for the model
- **port** (*int (default 8080)*) – The API port for the model

**Returns**

**success** – Whether the action was successful

**Return type**

bool

**delete\_user**(*user\_id: str, port: int = 8085*) → bool

Delete a user from the system

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.delete_user('user_id')
True
```

**Parameters**

- **user\_id** (*str*) – The user's ID
- **port** (*int* (default 8085)) – The API port to use

**Returns**

**result** – Returns True if the call is successful

**Return type**

bool

**get\_group**(*group\_id: str, port: int = 8086*) → dict

Retrieve information about a group

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.get_group('group_id')
*dictionary containing group data*
```

**Parameters**

- **group\_id** (*str*) – The ID of the group requested
- **port** (*int* (default 8086)) – The API port to use

**Returns**

**group\_info** – The information about the group

**Return type**

dict

**get\_group\_id\_by\_name**(*group\_name: str, port: int = 8083*) → str

Get the ID of a group by searching for its display name

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.get_group_id_by_name('Group Name')
*group_id*
```

**Parameters**

- **group\_name** (*str*) – The display name of the group
- **port** (*int* (default 8083)) – The API port to use

**Returns**

**group\_id** – The ID of the group

**Return type**

str

**get\_model**(*id: str, port: int = 8080*) → dict

Retrieve a model configuration

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.get_model('model_id')
*JSON Response including model data and metadata*
```



**Parameters**

- **id** (*str*) – The ID for the model
- **port** (*int* (default 8080)) – The API port for the call

**Returns**

**data** – Metadata about the model coupled with the model’s configuration information

**Return type**

dictionary

**get\_model\_id\_by\_name**(*model\_name: str, port: int = 8080*) → *str*

Retrieve a model’s ID using the name of the model

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.get_model_id_by_name('my_awesome_model')
*model_id*
```

**Parameters**

- **model\_name** (*str*) – The name of the model
- **port** (*int* (default 8080)) – The API port to use

**Returns**

**model\_id** – The model’s ID

**Return type**

*str*

**get\_role\_id\_by\_name**(*role\_name: str, port: int = 8086*) → *str*

Get the ID of a role by searching for its display name

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.get_role_id_by_name('Role Name')
*role_id*
```

**Parameters**

- **role\_name** (*str*) – The name of the role
- **port** (*int* (default 8086)) – The API port to use

**Returns**

**role\_id** – The ID of the role

**Return type**

*str*

**get\_user**(*user\_id: str, port: int = 8085*) → *dict*

Retrieve a user’s information from the platform

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.get_user('user_id')
*dictionary with results*
```

**Parameters**

- **user\_id** (*str*) – The ID of the user
- **port** (*int* (*default* 8085)) – The API port to use

**Returns**

**user\_info** – The information about the user

**Return type**

dict

**get\_user\_id\_by\_name**(*name: str, port: int = 8080*) → *str*

Get a user's ID from their display name

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.get_user_id_by_name('User Name')
*user_id*
```

**Parameters**

- **name** (*str*) – The display name of the user
- **port** (*int* (*default* 8080)) – The API port to use

**Returns**

**id** – The ID of the user

**Return type**

str

**property headers**

Headers used for authentication with the AI Squared Platform

**list\_group\_users**(*group\_id: str, as\_df: bool = True, port: int = 8083*) → Union[DataFrame, dict]

List users in a group

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.list_group_users('group_id')
*DataFrame with results*
```

**Parameters**

- **group\_id** (*str*) – The ID for the group
- **as\_df** (*bool* (*default* *True*)) – Whether to return the response as a pandas DataFrame
- **port** (*int* (*default* 8083)) – The API port to use

**Returns**

**users** – The response from the API

**Return type**

pandas DataFrame or dictionary

**list\_groups**(*as\_df: bool = True, port: int = 8083*) → Union[DataFrame, dict]

List all groups

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.list_groups()
*DataFrame with results*
```

#### Parameters

- **as\_df** (*bool (default True)*) – Whether to return the result as a pandas DataFrame
- **port** (*int (default 8083)*) – The API port for the call

#### Returns

**groups** – The response from the API

#### Return type

pandas DataFrame or dictionary

**list\_model\_feedback**(*model\_id: str, limit: int = 10, as\_df: bool = True, port: int = 8080*) → Union[dict, DataFrame]

List feedback on a model

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.list_model_feedback('model_id')
*DataFrame with Results*
```

#### Parameters

- **model\_id** (*str*) – The ID of the model
- **limit** (*int (default 10)*) – The maximum number of feedback items to return
- **port** (*int (default 8080)*) – The API port to use

#### Returns

**feedback** – The feedback

#### Return type

dict or pandas DataFrame

**list\_model\_prediction\_feedback**(*model\_id: str, as\_df: bool = True, port: int = 8080*) → Union[dict, DataFrame]

List all feedback for a model

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.list_model_prediction_feedback('model_id')
*DataFrame with Results*
```

#### Parameters

- **model\_id** (*str*) – The ID of the model requested
- **as\_df** (*bool (default True)*) – Whether to return the results as a pandas DataFrame

- **port** (*int* (default 8080)) – The API port to use

**Returns**

**results** – The results from the platform

**Return type**

dict or pandas DataFrame

**list\_model\_usage\_metrics**(*model\_id: str, period: str = 'hourly', as\_df: bool = True, port: int = 8080*) → Union[dict, DataFrame]

Get usage metrics for a model

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.get_model_usage_metrics('model_id')
*DataFrame with results*
```

**Parameters**

- **model\_id** (*str*) – The ID of the model
- **period** (*str* (default 'hourly')) – The period to group metrics into
- **as\_df** (*bool* (default True)) – Whether to return results as a pandas DataFrame
- **port** (*int* (default 8080)) – The API port to use

**Returns**

**results** – The results from the platform

**Return type**

pandas DataFrame or dict

**list\_model\_users**(*id: str, as\_df: bool = True, port: int = 8080*) → Union[DataFrame, dict]

List users for a model

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.list_model_users('model_id')
*DataFrame with results*
```

**Parameters**

- **id** (*str*) – The ID for the model
- **as\_df** (*bool* (default True)) – Whether to return the response as a Pandas DataFrame
- **port** (*int* (default 8080)) – The API port for the call

**Returns**

**model\_users** – The users for the model

**Return type**

pandas DataFrame or dictionary

**list\_models**(*as\_df: bool = True, port: int = 8080*) → Union[DataFrame, dict]

List models within the platform

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.list_models()
*DataFrame with results*
```

**Parameters**

- **as\_df** (*bool* (default *True*)) – Whether to return the response as a pandas DataFrame
- **port** (*int* (default *8080*)) – The API port for the call

**Returns**

**models** – The models

**Return type**

pandas DataFrame or dictionary

**list\_prediction\_feedback**(*prediction\_id: str, as\_df: bool = True, port: int = 8080*) → Union[DataFrame, dict]

List prediction feedback given a prediction ID

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.list_prediction_feedback('prediction_id')
*DataFrame with results*
```

**Parameters**

- **prediction\_id** (*str*) – The prediction ID
- **as\_df** (*bool* (default *True*)) – Whether to return the results as a pandas DataFrame
- **port** (*int* (default *8080*)) – The API port to use

**Returns**

**results** – The results from the platform

**Return type**

pandas DataFrame or dict

**list\_roles**(*as\_df: bool = True, port: int = 8086*) → Union[DataFrame, dict]

List the roles available in the platform

Example usage:

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.list_roles()
*DataFrame with results*
```

**Parameters**

- **as\_df** (*bool* (default *True*)) – Whether to return the results as a pandas DataFrame
- **port** (*int* (default *8086*)) – The API port to use

**Returns**

**roles** – The roles

**Return type**

pandas DataFrame or dict

**list\_user\_usage\_metrics**(*user\_id: str, period: str = 'hourly', as\_df: bool = True, port: int = 8080*) → Union[dict, DataFrame]

Get usage metrics for a user

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.get_user_usage_metrics('user_id')
*DataFrame with results*
```

**Parameters**

- **user\_id** (*str*) – The ID of the user
- **period** (*str (default 'hourly')*) – The period to group metrics into
- **as\_df** (*bool (default True)*) – Whether to return results as a pandas DataFrame
- **port** (*int (default 8080)*) – The API port to use

**Returns****results** – The results from the platform**Return type**

pandas DataFrame or dict

**list\_users**(*as\_df: bool = True, port: int = 8080*) → Union[DataFrame, dict]

List all users

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.list_users()
*DataFrame with results*
```

**Parameters**

- **as\_df** (*bool (default True)*) – Whether to return the data as a Pandas DataFrame
- **port** (*int (default 8080)*) – The API port for the call

**Returns****users** – The response from the API**Return type**

pandas DataFrame or dictionary

**login**(*url: Optional[str] = None, port: int = 8080, username: Optional[str] = None, password: Optional[str] = None*) → None

Log in to the platform programmatically. If no url, username, or password are provided, logs in interactively

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.login()
Enter URL: https://platform.squared.ai
```

(continues on next page)

(continued from previous page)

```
Enter Username: your.email@your_domain.com
Enter Password: <hidden>
```

**Parameters**

- **url** (*str or None (default None)*) – The URL for the platform API
- **port** (*int (default 8080)*) – The API port for the call
- **username** (*str or None (default None)*) – The username
- **password** (*str or None (default None)*) – The password

**property password:** *str*

The password associated with the client

**remove\_users\_from\_group**(*group\_id: str, user\_ids: list, port: int = 8086*) → bool

Remove users from a group

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.remove_users_from_group('group_id', ['user_id_1', 'user_id_2'])
True
```

**Parameters**

- **group\_id** (*str*) – The ID of the group
- **user\_ids** (*list of str*) – The IDs of the users to remove
- **port** (*int (default = 8086)*) – The API port to use

**Returns**

**success** – Returns True if successful

**Return type**

bool

**share\_model\_with\_group**(*model\_id: str, group\_id: str, port: int = 8080*) → bool

Share a model with a group

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.share_model_with_group('model_id', 'group_id')
True
```

**Parameters**

- **model\_id** (*str*) – The ID for the model to be shared
- **group\_id** (*str*) – The ID for the group to be shared with
- **port** (*int (default 8080)*) – The API port to use

**Returns**

**success** – Returns True if successful

**Return type**

bool

**share\_model\_with\_user**(*model\_id: str, user\_id: str, port: int = 8080*) → bool

Share a model with a user

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.share_model_with_user('model_id', 'user_id')
True
```

**Parameters**

- **model\_id** (*str*) – The ID for the model
- **user\_id** (*str*) – The ID for the user
- **port** (*int (default 8080)*) – The API port for the call

**Returns****success** – Whether the action was successful**Return type**

bool

**test\_connection**(*port: int = 8080*) → int

Test whether there is a healthy connection to the platform

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.test_connection()
Connection successful
200
```

**Parameters**

- **port** (*int (default 8080)*) – The API port for the call

**Returns****status\_code** – The status code when checking the health API**Return type**

int

**property token: str**

The token associated with the client

**unshare\_model\_with\_group**(*model\_id: str, group\_id: str, port: int = 8080*) → bool

Unshare a model with a group

```
>>> import aisquared
>>> client = aisquared.client.AISquaredPlatformClient()
>>> client.unshare_model_with_group('model_id', 'group_id')
True
```

**Parameters**

- **model\_id** (*str*) – The ID of the model



- **group\_id** (*str*) – The ID of the group
- **port** (*int* (default 8080)) – The API port to use

**Returns**

**success** – Returns True if successful

**Return type**

bool

**unshare\_model\_with\_user**(*model\_id: str, user\_id: str, port: int = 8080*) → bool

Unshare a model with a user

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.unshare_model_with_user('model_id', 'user_id')
True
```

**Parameters**

- **model\_id** (*str*) – The ID for the model
- **user\_id** (*str*) – The ID for the user
- **port** (*int* (default 8080)) – The API port for the call

**Returns**

**success** – Whether the action was successful

**Return type**

bool

**update\_group**(*group\_id: str, display\_name: str, role\_id: str, port: int = 8086*) → bool

Update information about a group

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.update_group(
    'group_id',
    'group display name',
    'role_id'
)
True
```

**Parameters**

- **group\_id** (*str*) – The ID of the group to update
- **display\_name** (*str*) – The display name of the group
- **role\_id** (*str*) – The ID of the role for the group
- **port** (*int* (default 8086)) – The API port to use

**Returns**

**success** – Returns True if successful

**Return type**

bool

**update\_user**(*user\_id: str, user\_name: str, given\_name: str, family\_name: str, email: str, role\_id: str, active: bool = True, middle\_name: Optional[str] = None, company\_id: Optional[str] = None, password: Optional[str] = None, port: int = 8085*) → bool

Update information about a user

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.update_user(
    'user_id',
    'user name',
    'given_name',
    'family_name',
    'user_email',
    'role_id'
)
True
```

#### Parameters

- **user\_id** (*str*) – The ID of the user to update
- **user\_name** (*str*) – The display name of the user
- **given\_name** (*str*) – The first name of the user
- **family\_name** (*str*) – The last name of the user
- **email** (*str*) – The user's email
- **role\_id** (*str*) – The ID of the user's role
- **active** (*bool* (default *True*)) – Whether the user is active
- **middle\_name** (*str or None* (default *None*)) – The user's middle name
- **company\_id** (*str or None* (default *None*)) – The user's company ID
- **password** (*str or None* (default *None*)) – The user's password
- **port** (*int* (default *8085*)) – The API port to use

#### Returns

**success** – Returns True if update is successful

#### Return type

bool

**upload\_model**(*model\_file: str, port: int = 8081*) → str

Upload a model to the platform

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.upload_model('my_model_filename.air')
True
```

#### Parameters

- **model\_file** (*path or path-like*) – The path to the model file
- **port** (*int* (default *8081*)) – The API port to use

**Returns****successful** – Whether the action was successful**Return type**

bool

**property username: str**

The username associated with the client

**aisquared.serving package****Module contents**

The aisquared.serving package contains utilities to serve models to a local REST endpoint.

Here is an example of how to serve a simple keras model using these utilities:

```
>>> # Assume model is already trained and stored in memory as model
>>> from aisquared import serving
>>> serving.save_keras_model(model, 'my_model')
>>> serving.deploy_model(
    'my_model',
    'keras',
    additional_functions_file = '<optional file containing `preprocess` and
    ↳ `postprocess` functions, if applicable>'
)
App created successfullly. Serving and awaiting requests
```

And to retrieve predictions from the model:

```
>>> # From a separate terminal, assume data is already loaded
>>> from aisquared import serving
>>> serving.get_remote_predictions(data) # Do not need to change host or port if
    ↳ predicting from the same machine
*predictions*
```

**Submodules****aisquared.serving.deploy\_model module**

```
aisquared.serving.deploy_model.deploy_model(saved_model: str, model_type: str, host: str = '127.0.0.1',
                                             port: int = 2244, custom_objects: Optional[dict] = None,
                                             additional_functions_file: Optional[str] = None)
```

Deploy a model to a Flask server on the specified host

**Parameters**

- **saved\_model** (*Path-like*) – The path to the saved model directory or model file
- **model\_type** (*str*) – The type of model
- **host** (*str* (default '127.0.0.1')) – The host to deploy to
- **port** (*int* (default 2244)) – The port to deploy to

- **custom\_objects** (*dict or None (default None)*) – Any custom objects to load when using a MANN model
- **additional\_functions\_file** (*file-like or None (default None)*) – File name containing additional functions (which have to be named *preprocess* and *postprocess*, if created) that are used during the prediction process

`aisquared.serving.deploy_model.load_mann_model(model: str, custom_objects: dict)`

Load a MANN model with custom objects

### **aisquared.serving.get\_remote\_prediction module**

`aisquared.serving.get_remote_prediction.get_remote_prediction(data: Union[dict, str, ndarray, list],  
host: str = '127.0.0.1', port: int = 2244) → list`

Send data to use for prediction

#### **Parameters**

- **data** (*dict, str, np.ndarray, or list*) – The data to be predicted on
- **host** (*str (default '127.0.0.1')*) – The host to use
- **port** (*int (default '2244')*) – The port to use

#### **Notes**

- If data is a dictionary, it is expected to already be correctly formatted
- If data is a string, it is expected to already be correctly formatted

#### **Returns**

**predictions** – The predictions from the deployed model

#### **Return type**

list

### **aisquared.utils package**

#### **Module contents**

Additional utilities to use with the *aisquared* package. These utilities currently consist of two functions, the *mimic\_model* and *get\_model* functions. They utilize functionality that exists in our open source package BeyondML to train teacher-student models

To see in-depth examples of how to use these functions, please visit our GitHub repository at <https://github.com/AISquaredInc/MimicModelExamples>

## Submodules

### aisquared.utils.utils module

`aisquared.utils.utils.get_model(model_type: str, input_shape: Union[int, tuple[int]], num_outputs: int, output_activation: str, size: str = 'small', vocab_size: Union[None, int] = None)`

Get a pre-configured model for different use cases

#### Parameters

- **model\_type** (*str*) – Either ‘cv’, ‘nlp\_embedding’, or ‘fc’, defining the model type
- **input\_shape** (*int or tuple of int*) – The input shape to the model
- **num\_outputs** (*int*) – The output shape of the model
- **output\_activation** (*str or keras activation function*) – The activation of the final layer of the model
- **size** (*str (default 'small')*) – One of either ‘small’, ‘medium’, or ‘large’
- **vocab\_size** (*str or None (default None)*) – Size of the vocab, if model\_type is ‘nlp\_embedding’

#### Returns

**model** – The model

#### Return type

TensorFlow Keras model

`aisquared.utils.utils.mimic_model(trained_model: BaseEstimator, nnet: Model, training_data: ndarray, test_data: ndarray, test_labels: ndarray, problem_type: str, loss: str, metrics: Union[str, list], optimizer: str, mimic_proba: bool = False, retention: float = 0.9, batch_size: int = 32, epochs: int = 100, starting_sparsification: int = 0, max_sparsification: int = 99, sparsification_rate: int = 5) → Model`

Train a sparse neural network to mimic a scikit-learn model

#### Parameters

- **trained\_model** (*sklearn model*) – The model that is already trained
- **nnet** (*TensorFlow keras Model*) – The neural network to train to mimic the trained model
- **training\_data** (*array or array-like*) – The input data that was used to train the trained model
- **test\_data** (*array or array-like*) – The input data to be used for testing
- **test\_labels** (*array or array-like*) – The output data used in testing
- **problem\_type** (*str*) – The type of problem, either ‘classification’ or ‘regression’
- **loss** (*str or keras loss function*) – The loss to use
- **metrics** (*str, function or list of str, function*) – Metrics to measure
- **optimizer** (*str or keras optimizer*) – The optimizer to use
- **mimic\_proba** (*bool (default False)*) – For classification, mimic the probability outputs

- **retention** (*float* (*default* 0.9)) – The retention of performance to allow further pruning
- **batch\_size** (*int* (*default* 32)) – The batch size to use while training
- **epochs** (*int* (*default* 100)) – The number of epochs (if early stopping is not met beforehand)
- **starting\_sparsification** (*int* (*default* 0)) – The starting model sparsification
- **max\_sparsification** (*int* (*default* 99)) – The maximum sparsification to allow
- **sparsification\_rate** (*int* (*default* 5)) – The sparsification rate when invoked

**Returns**

**nnet** – The trained model

**Return type**

TensorFlow keras Model

## CHANGELOG

- **Version 0.1.3**

- Added *flags* parameter to *TextHarvester* using regular expression harvesting
- Deleted *model\_feedback* parameter in *ModelConfiguration* object and included functionality in *feedback\_steps* parameter
- Changed *format* parameter to *header* for both deployed analytics
- Added feedback and stages to *DocumentPredictor* and *ImagePredictor* objects
- Non-API changes for *ALLOWED\_STAGES*
- Fixed bugs preventing Windows users from importing the package
- Updated *ModelConfiguration* to include *url* parameter
- Changed default tokenization string

- **Version 0.2.0**

- Moved preprocessing steps under subpackages for specific kinds of preprocessing steps
- Cleaned up documentation to render within programmatic access environments
- Added *aisquared.logging* subpackage
- **Created *InputHarvester***
  - \* Allows for harvesting of input text, images, and tabular data
- Created the *aisquared.serving* subpackage, specifically the *deploy\_model* and *get\_remote\_prediction* functions
- Created the *GraphConfiguration* class
- Added *auto-run* parameter to *ModelConfiguration* and *GraphConfiguration* classes
- **Created the *aisquared* CLI with the following commands:**
  - \* *aisquared deploy*, which deploys a model locally
  - \* *aisquared predict*, which predicts using a local JSON file
  - \* *aisquared airfiles*, which contains the subcommands *list*, *delete*, *download*, and *upload*
- Changed all classes within *aisquared.config.analytic* to accept 'tabular' as an *input\_type*
- Removed *aisquared.logging* and *aisquared.remote* from top-level imports
- Added *round* parameter to Regression postprocessor
- Removed *DocumentPredictor* and *ImagePredictor* classes

- Removed *ChainRendering* class
- Created *FilterRendering* class
- Altered *QUALIFIERS*
- Added advanced rendering parameters to rendering objects
- Removed *logging* and *remote* subpackages from top-level *aisquared* import
- **Version 0.2.1**
  - Added the *S3Connector* class to the *analytics* subpackage, which allows download of an analytic directly from S3
  - Updated the documentation and added the *docs* subdirectory for hosting the documentation on GitHub Pages
- **Version 0.2.2**
  - Fixed bug in *to\_dict* method within *ObjectRendering* class
  - Fixed bug in name of *MultiplyValue* step
  - Fixed bug in datatype checking for text harvester
  - Added *body\_only* parameter to *TextHarvester*
  - Added ‘underline’ to possible badges
  - Added *threshold\_key* and *threshold\_values* to relevant rendering classes
  - Added *Trim* text preprocessing class
  - Added *CustomObject* in the base package to allow for creation of custom classes
  - Added keyword harvesting capabilities
  - Added *utils* subpackage with capabilities to mimic a trained sklearn model
  - Small documentation changes
  - Changed the required imports for the package to streamline installation process, and created two installation options *aisquared* and *aisquared[full]*
- **Version 0.2.3**
  - Added functionality to add custom preprocessing and postprocessing functions to the model deployment pipeline
  - Added *all* parameter to *LocalAnalytic* class
  - Changed under-the-hood functionality of *mimic\_model* function in line with updates to *BeyondML*
  - Altered the *ReverseMLWorkflow* analytic
  - Added the *BarChartRendering*, *ContainerRendering*, *DashboardReplacementRendering*, *DoughnutChartRendering*, *HTMLTagRendering*, *LineChartRendering*, *PieChartRendering*, *SOSRendering*, and *TableRendering* rendering classes
  - Added the *QueryParameterHarvester* harvester class
  - Added the *limit* parameter to the *TextHarvester* class
- **Version 0.3.0**
  - Added type hinting to documentation strings
  - Revamped documentation to use Sphinx



- **Version 0.3.1**

- Changed Python type hints to allow for backwards compatibility with older versions of Python

- **Version 0.3.2**

- Added functionality to the *AISquaredPlatformClient*
  - Added *top\_level\_kwargs* parameter to the *CustomObject* class
  - Added *DashboardRendering* class
  - Removed ‘px’ from default values in *ImageRendering* and *ObjectRendering* classes
  - Added functionality for creating, updating, and deleting users to *AISquaredPlatformClient*
  - Added functionality for creating, updating, and deleting groups to *AISquaredPlatformClient*
  - Fixed bug related to requiring *auto\_run* parameter to be string (fix involves casting as string)
  - Altered schemas for different “Chart” *Rendering* classes to conform to JavaScript standards
  - Streamlined the *ModelConfiguration* class to allow a more functional interface to build *.air* files
  - Updated *ContainerRendering* class with parameters for *position* and *static\_position*
  - Updated across-the-board functionality of the *AISquaredPlatformClient*



## PYTHON MODULE INDEX

### a

aisquared, 3  
aisquared.base, 3  
aisquared.base.BaseObject, 3  
aisquared.base.CustomObject, 4  
aisquared.base.rendering, 4  
aisquared.base.stages, 4  
aisquared.config, 4  
aisquared.config.analytic, 5  
aisquared.config.analytic.DeployedAnalytic, 5  
aisquared.config.analytic.DeployedModel, 6  
aisquared.config.analytic.LocalAnalytic, 6  
aisquared.config.analytic.LocalModel, 7  
aisquared.config.analytic.ReverseMLWorkflow, 7  
aisquared.config.feedback, 8  
aisquared.config.feedback.BinaryFeedback, 8  
aisquared.config.feedback.ModelFeedback, 9  
aisquared.config.feedback.MulticlassFeedback, 9  
aisquared.config.feedback.QualitativeFeedback, 10  
aisquared.config.feedback.RegressionFeedback, 10  
aisquared.config.feedback.SimpleFeedback, 11  
aisquared.config.GraphConfiguration, 38  
aisquared.config.harvesting, 11  
aisquared.config.harvesting.ImageHarvester, 11  
aisquared.config.harvesting.InputHarvester, 12  
aisquared.config.harvesting.QueryParameterHarvester, 12  
aisquared.config.harvesting.TextHarvester, 13  
aisquared.config.ModelConfiguration, 39  
aisquared.config.postprocessing, 14  
aisquared.config.postprocessing.BinaryClassification, 14  
aisquared.config.postprocessing.MulticlassClassification, 14  
aisquared.config.postprocessing.ObjectDetection, 15  
aisquared.config.postprocessing.Regression, 15  
aisquared.config.preprocessing, 16  
aisquared.config.preprocessing.image, 16  
aisquared.config.preprocessing.image.ImagePreprocessing, 16  
aisquared.config.preprocessing.image.Steps, 17  
aisquared.config.preprocessing.tabular, 19  
aisquared.config.preprocessing.tabular.Steps, 19  
aisquared.config.preprocessing.tabular.TabularPreprocessing, 21  
aisquared.config.preprocessing.text, 21  
aisquared.config.preprocessing.text.Steps, 21  
aisquared.config.preprocessing.text.TextPreprocessing, 24  
aisquared.config.rendering, 24  
aisquared.config.rendering.BarChartRendering, 24  
aisquared.config.rendering.ContainerRendering, 26  
aisquared.config.rendering.DashboardReplacementRendering, 27  
aisquared.config.rendering.DocumentRendering, 28  
aisquared.config.rendering.DoughnutChartRendering, 29  
aisquared.config.rendering.FilterRendering, 30  
aisquared.config.rendering.HTMLTagRendering, 31  
aisquared.config.rendering.ImageRendering, 32  
aisquared.config.rendering.LineChartRendering, 33  
aisquared.config.rendering.ObjectRendering, 34  
aisquared.config.rendering.PieChartRendering, 35  
aisquared.config.rendering.SOSRendering, 36  
aisquared.config.rendering.TableRendering, 36  
aisquared.config.rendering.WordRendering, 37

`aisquared.logging`, [40](#)  
`aisquared.platform`, [41](#)  
`aisquared.platform.AISquaredPlatformClient`,  
    [41](#)  
`aisquared.serving`, [55](#)  
`aisquared.serving.deploy_model`, [55](#)  
`aisquared.serving.get_remote_prediction`, [56](#)  
`aisquared.utils`, [56](#)  
`aisquared.utils.utils`, [57](#)

## A

`add_node()` (*aisquared.config.GraphConfiguration.GraphConfiguration* module), 38  
`add_question()` (*aisquared.config.feedback.ModelFeedback.ModelFeedback* module), 9  
`add_question()` (*aisquared.config.feedback.QualitativeFeedback.QualitativeFeedback* module), 10  
`add_step()` (*aisquared.config.preprocessing.image.ImagePreprocessing.ImagePreprocessor* module), 16  
`add_step()` (*aisquared.config.preprocessing.tabular.TabularPreprocessing.TabularPreprocessor* module), 21  
`add_step()` (*aisquared.config.preprocessing.text.TextPreprocessing.TextPreprocessor* module), 24  
`add_users_to_group()`  
     (*aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient* module), 41  
`AddValue` (class in *aisquared.config.preprocessing.image.Steps*), 17  
`aisquared`  
     module, 3  
`aisquared.base`  
     module, 3  
`aisquared.base.BaseObject`  
     module, 3  
`aisquared.base.CustomObject`  
     module, 4  
`aisquared.base.rendering`  
     module, 4  
`aisquared.base.stages`  
     module, 4  
`aisquared.config`  
     module, 4  
`aisquared.config.analytic`  
     module, 5  
`aisquared.config.analytic.DeployedAnalytic`  
     module, 5  
`aisquared.config.analytic.DeployedModel`  
     module, 6  
`aisquared.config.analytic.LocalAnalytic`  
     module, 6  
`aisquared.config.analytic.LocalModel`  
     module, 7  
`aisquared.config.analytic.ReverseMLWorkflow`  
     module, 7  
`aisquared.config.feedback`  
     module, 8  
`aisquared.config.feedback.BinaryFeedback`  
     module, 8  
`aisquared.config.feedback.ModelFeedback`  
     module, 8  
`aisquared.config.feedback.MulticlassFeedback`  
     module, 9  
`aisquared.config.feedback.QualitativeFeedback`  
     module, 9  
`aisquared.config.feedback.RegressionFeedback`  
     module, 10  
`aisquared.config.feedback.SimpleFeedback`  
     module, 11  
`aisquared.config.GraphConfiguration`  
     module, 38  
`aisquared.config.harvesting`  
     module, 11  
`aisquared.config.harvesting.ImageHarvester`  
     module, 11  
`aisquared.config.harvesting.InputHarvester`  
     module, 12  
`aisquared.config.harvesting.QueryParameterHarvester`  
     module, 12  
`aisquared.config.harvesting.TextHarvester`  
     module, 13  
`aisquared.config.ModelConfiguration`  
     module, 39  
`aisquared.config.postprocessing`  
     module, 14  
`aisquared.config.postprocessing.BinaryClassification`  
     module, 14  
`aisquared.config.postprocessing.MulticlassClassification`  
     module, 14  
`aisquared.config.postprocessing.ObjectDetection`  
     module, 15  
`aisquared.config.postprocessing.Regression`  
     module, 15  
`aisquared.config.preprocessing`  
     module, 16

```

aisquared.config.preprocessing.image
    module, 16
aisquared.config.preprocessing.image.ImagePreprocessing
    module, 16
aisquared.config.preprocessing.image.Steps
    module, 17
aisquared.config.preprocessing.tabular
    module, 19
aisquared.config.preprocessing.tabular.Steps
    module, 19
aisquared.config.preprocessing.tabular.TabularPreprocessing
    module, 21
aisquared.config.preprocessing.text
    module, 21
aisquared.config.preprocessing.text.Steps
    module, 21
aisquared.config.preprocessing.text.TextPreprocessing
    module, 24
aisquared.config.rendering
    module, 24
aisquared.config.rendering.BarChartRendering
    module, 24
aisquared.config.rendering.ContainerRendering
    module, 26
aisquared.config.rendering.DashboardReplacementRendering
    module, 27
aisquared.config.rendering.DocumentRendering
    module, 28
aisquared.config.rendering.DoughnutChartRendering
    module, 29
aisquared.config.rendering.FilterRendering
    module, 30
aisquared.config.rendering.HTMLTagRendering
    module, 31
aisquared.config.rendering.ImageRendering
    module, 32
aisquared.config.rendering.LineChartRendering
    module, 33
aisquared.config.rendering.ObjectRendering
    module, 34
aisquared.config.rendering.PieChartRendering
    module, 35
aisquared.config.rendering.SOSRendering
    module, 36
aisquared.config.rendering.TableRendering
    module, 36
aisquared.config.rendering.WordRendering
    module, 37
aisquared.logging
    module, 40
aisquared.platform
    module, 41
aisquared.platform.AISquaredPlatformClient
    module, 41
aisquared.serving
    module, 55
aisquared.serving.deploy_model
    module, 55
aisquared.serving.get_remote_prediction
    module, 56
aisquared.utils
    module, 56
aisquared.utils.utils
    module, 57
AISquaredAPIException, 41
AISquaredPlatformClient (class in
    aisquared.platform.AISquaredPlatformClient),
    41
all (aisquared.config.analytic.LocalAnalytic.LocalAnalytic
    property), 6
analytics (aisquared.config.ModelConfiguration.ModelConfiguration
    property), 39
analytic_dict (aisquared.config.ModelConfiguration.ModelConfiguration
    property), 39
anchor_selector (aisquared.config.rendering.DashboardReplacementRe
    property), 27
attributes (aisquared.config.harvesting.QueryParameterHarvester.Query
    property), 12
auto_run (aisquared.config.GraphConfiguration.GraphConfiguration
    property), 38
auto_run (aisquared.config.ModelConfiguration.ModelConfiguration
    property), 39
B
badge_color (aisquared.config.rendering.ImageRendering.ImageRenderin
    property), 32
badge_color (aisquared.config.rendering.ObjectRendering.ObjectRenderi
    property), 34
badge_color (aisquared.config.rendering.WordRendering.WordRendering
    property), 37
badge_shape (aisquared.config.rendering.WordRendering.WordRendering
    property), 37
BarChartRendering (class in
    aisquared.config.rendering.BarChartRendering),
    24
base_url (aisquared.platform.AISquaredPlatformClient.AISquaredPlatfor
    property), 41
BaseObject (class in aisquared.base.BaseObject), 3
BinaryClassification (class in
    aisquared.config.postprocessing.BinaryClassification),
    14
BinaryFeedback (class in
    aisquared.config.feedback.BinaryFeedback), 8
body_only (aisquared.config.harvesting.TextHarvester.TextHarvester
    property), 13
bucket (aisquared.config.analytic.ReverseMLWorkflow.ReverseMLWorkflo
    property), 8

```

## C

`can_toggle` (`aisquared.config.rendering.SOSRendering.SOSRendering` method), 43  
`property`), 36  
`classes` (`aisquared.config.rendering.DocumentRendering.DocumentRendering` method), 43  
`property`), 28  
`classes` (`aisquared.config.rendering.ImageRendering.ImageRendering` method), 43  
`property`), 32  
`classes` (`aisquared.config.rendering.WordRendering.WordRendering` method), 43  
`property`), 37  
`color` (`aisquared.config.preprocessing.image.Steps.ConvertToColor` method), 17  
`property`), 17  
`color` (`aisquared.config.rendering.ImageRendering.ImageRendering` method), 32  
`property`), 32  
`color` (`aisquared.config.rendering.ObjectRendering.ObjectRendering` method), 34  
`property`), 34  
`column` (`aisquared.config.analytic.ReverseMLWorkflow.ReverseMLWorkflow` method), 8  
`property`), 8  
`column` (`aisquared.config.preprocessing.tabular.Steps.DropColumn` method), 19  
`property`), 19  
`column` (`aisquared.config.preprocessing.tabular.Steps.OneHot` method), 20  
`property`), 20  
`columns` (`aisquared.config.preprocessing.tabular.Steps.MinMax` method), 19  
`property`), 19  
`columns` (`aisquared.config.preprocessing.tabular.Steps.ZScore` method), 20  
`property`), 20  
`compile`() (`aisquared.config.GraphConfiguration.GraphConfiguration` method), 38  
`property`), 38  
`compile`() (`aisquared.config.ModelConfiguration.ModelConfiguration` method), 39  
`property`), 39  
`ContainerRendering` (class in `aisquared.config.rendering.ContainerRendering`), 26  
`aisquared.config.rendering.ContainerRendering`), 26  
`content_key` (`aisquared.config.rendering.WordRendering.WordRendering` method), 37  
`property`), 37  
`ConvertToCase` (class in `aisquared.config.preprocessing.text.Steps`), 21  
`aisquared.config.preprocessing.text.Steps`), 21  
`ConvertToColor` (class in `aisquared.config.preprocessing.image.Steps`), 17  
`aisquared.config.preprocessing.image.Steps`), 17  
`ConvertToVocabulary` (class in `aisquared.config.preprocessing.text.Steps`), 22  
`aisquared.config.preprocessing.text.Steps`), 22  
`create_group`() (`aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient` method), 42  
`property`), 42  
`create_user`() (`aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient` method), 42  
`property`), 42  
`CustomObject` (class in `aisquared.base.CustomObject`), 4  
`aisquared.base.CustomObject`), 4

## D

`DashboardReplacementRendering` (class in `aisquared.config.rendering.DashboardReplacementRendering`), 27  
`aisquared.config.rendering.DashboardReplacementRendering`), 27

font\_size(aisquared.config.rendering.ObjectRendering.ObjectRendering property), 34

**G**

get\_filenames() (aisquared.config.GraphConfiguration.GraphConfiguration method), 38

get\_group() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 44

get\_group\_id\_by\_name() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 44

get\_model() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 44

get\_model() (in module aisquared.utils.utils), 57

get\_model\_filenames() (aisquared.config.ModelConfiguration.ModelConfiguration method), 40

get\_model\_id\_by\_name() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 45

get\_remote\_prediction() (in module aisquared.serving.get\_remote\_prediction), 56

get\_role\_id\_by\_name() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 45

get\_user() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 45

get\_user\_id\_by\_name() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 46

GraphConfiguration (class in aisquared.config.GraphConfiguration), 38

**H**

harvester\_dict (aisquared.config.ModelConfiguration.ModelConfiguration property), 40

harvesting\_steps (aisquared.config.ModelConfiguration.ModelConfiguration property), 40

header (aisquared.config.analytic.DeployedAnalytic.DeployedAnalytic property), 5

header (aisquared.config.analytic.DeployedModel.DeployedModel property), 6

headers (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient property), 46

height (aisquared.config.rendering.ContainerRendering.ContainerRendering property), 26

how (aisquared.config.harvesting.ImageHarvester.ImageHarvester property), 11

how (aisquared.config.harvesting.TextHarvester.TextHarvester property), 13

HTMLTagRendering (class in aisquared.config.rendering.HTMLTagRendering), 31

id (aisquared.config.rendering.ContainerRendering.ContainerRendering property), 26

ImageHarvester (class in aisquared.config.harvesting.ImageHarvester), 11

ImagePreprocessor (class in aisquared.config.preprocessing.image.ImagePreprocessing), 16

ImageRendering (class in aisquared.config.rendering.ImageRendering), 28

include\_probability (aisquared.config.rendering.DocumentRendering.DocumentRendering property), 28

include\_probability (aisquared.config.rendering.ImageRendering.ImageRendering property), 32

include\_probability (aisquared.config.rendering.ObjectRendering.ObjectRendering property), 34

input\_type (aisquared.config.analytic.DeployedAnalytic.DeployedAnalytic property), 5

input\_type (aisquared.config.analytic.DeployedModel.DeployedModel property), 7

input\_type (aisquared.config.analytic.LocalAnalytic.LocalAnalytic property), 7

input\_type (aisquared.config.analytic.LocalModel.LocalModel property), 8

input\_type (aisquared.config.harvesting.InputHarvester.InputHarvester property), 12

InputHarvester (class in aisquared.config.harvesting.InputHarvester), 12

**K**

key (aisquared.config.rendering.FilterRendering.FilterRendering property), 30

**L**

label (aisquared.config.rendering.ContainerRendering.ContainerRendering property), 26

label (aisquared.config.rendering.DashboardReplacementRendering.DashboardReplacementRendering property), 27

label (aisquared.config.rendering.SOSRendering.SOSRendering property), 36

label\_map (aisquared.config.feedback.BinaryFeedback.BinaryFeedback property), 8

label\_map (aisquared.config.feedback.MulticlassFeedback.MulticlassFeedback property), 10

label\_map (aisquared.config.postprocessing.BinaryClassification.BinaryClassification property), 14



label\_map (aisquared.config.postprocessing.MulticlassClassificationHarvesting.InputHarvester.InputHarvester property), 14  
 label\_map (aisquared.config.postprocessing.ObjectDetectionHarvesting.InputHarvester.InputHarvester property), 15  
 length (aisquared.config.preprocessing.text.Steps.PadSequence property), 22  
 limit (aisquared.config.harvesting.TextHarvester.TextHarvester property), 13  
 LineChartRendering (class in method (aisquared.config.preprocessing.image.Steps.Resize property), 18  
 33  
 list\_group\_users() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 46  
 list\_groups() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 46  
 list\_model\_feedback() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 47  
 list\_model\_prediction\_feedback() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 47  
 list\_model\_usage\_metrics() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 48  
 list\_model\_users() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 48  
 list\_models() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 48  
 list\_prediction\_feedback() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 49  
 list\_roles() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 49  
 list\_user\_usage\_metrics() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 50  
 list\_users() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 50  
 load\_mann\_model() (in module aisquared.serving.deploy\_model), 56  
 LocalAnalytic (class in aisquared.config.analytic.LocalAnalytic), 6  
 LocalModel (class in aisquared.config.analytic.LocalModel), 7  
 login() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 50  
 lowercase (aisquared.config.preprocessing.text.Steps.ConvertToCase property), 21

**M**  
 max (aisquared.config.postprocessing.Regression.Regression property), 15

```

aisquared.config.feedback.ModelFeedback,
aisquared.config.feedback.MulticlassFeedback,
aisquared.config.feedback.QualitativeFeedback,
aisquared.config.feedback.RegressionFeedback,
aisquared.config.feedback.SimpleFeedback,
aisquared.config.GraphConfiguration,
aisquared.config.harvesting,
aisquared.config.harvesting.ImageHarvester,
aisquared.config.harvesting.InputHarvester,
aisquared.config.harvesting.QueryParameterHarvester,
aisquared.config.harvesting.TextHarvester,
aisquared.config.ModelConfiguration,
aisquared.config.postprocessing,
aisquared.config.postprocessing.BinaryClassification,
aisquared.config.postprocessing.MulticlassClassification,
aisquared.config.postprocessing.ObjectDetection,
aisquared.config.postprocessing.Regression,
aisquared.config.preprocessing,
aisquared.config.preprocessing.image,
aisquared.config.preprocessing.image.ImagePreprocessing,
aisquared.config.preprocessing.image.Steps,
aisquared.config.preprocessing.tabular,
aisquared.config.preprocessing.tabular.Steps,
aisquared.config.preprocessing.tabular.TabularPreprocessing,
aisquared.config.preprocessing.text,
aisquared.config.preprocessing.text.Steps,
aisquared.config.preprocessing.text.TextPreprocessing,
aisquared.config.rendering,
aisquared.config.rendering.BarChartRendering,
aisquared.config.rendering.ContainerRendering,
aisquared.config.rendering.DashboardReplacementRendering,
aisquared.config.rendering.DocumentRendering,
aisquared.config.rendering.DoughnutChartRendering,
aisquared.config.rendering.FilterRendering,
aisquared.config.rendering.HTMLTagRendering,
aisquared.config.rendering.ImageRendering,
aisquared.config.rendering.LineChartRendering,
aisquared.config.rendering.ObjectRendering,
aisquared.config.rendering.PieChartRendering,
aisquared.config.rendering.SOSRendering,
aisquared.config.rendering.TableRendering,
aisquared.config.rendering.WordRendering,
aisquared.logging,
aisquared.platform,
aisquared.platform.AISquaredPlatformClient,
aisquared.serving,
aisquared.serving.deploy_model,
aisquared.serving.get_remote_prediction,
aisquared.utils,
aisquared.utils.utils,
aisquared.config.postprocessing.MulticlassClassification (class in
aisquared.config.postprocessing.MulticlassClassification),
aisquared.config.feedback.MulticlassFeedback (class in
aisquared.config.feedback.MulticlassFeedback),
aisquared.config.preprocessing.image.Steps (class in
aisquared.config.preprocessing.image.Steps),
aisquared.config.postprocessing.ObjectDetection (class in
aisquared.config.postprocessing.ObjectDetection),
aisquared.config.rendering.ObjectRendering (class in
aisquared.config.rendering.ObjectRendering),

```

34  
 OneHot (class in aisquared.config.preprocessing.tabular.Steps), 20  
 oov\_character (aisquared.config.preprocessing.text.Steps.ConvertToVocabulary (property), 22  
 orientation (aisquared.config.rendering.ContainerRendering.ContainerRendering (property), 26  
 owner (aisquared.config.GraphConfiguration.GraphConfiguration (property), 38  
 owner (aisquared.config.ModelConfiguration.ModelConfiguration (property), 40  
**P**  
 pad\_character (aisquared.config.preprocessing.text.Steps.PadSequences (property), 22  
 pad\_location (aisquared.config.preprocessing.text.Steps.PadSequences (property), 22  
 PadSequences (class in aisquared.config.preprocessing.text.Steps), 22  
 password (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient (property), 51  
 path (aisquared.config.analytic.LocalAnalytic.LocalAnalytic (property), 7  
 path (aisquared.config.analytic.LocalModel.LocalModel (property), 7  
 period (aisquared.config.analytic.ReverseMLWorkflow.ReverseMLWorkflow (property), 8  
 PieChartRendering (class in aisquared.config.rendering.PieChartRendering), 35  
 placement (aisquared.config.rendering.ImageRendering.ImageRendering (property), 32  
 placement (aisquared.config.rendering.ObjectRendering.ObjectRendering (property), 34  
 position (aisquared.config.rendering.ContainerRendering.ContainerRendering (property), 26  
 postprocessor\_dict (aisquared.config.ModelConfiguration.ModelConfiguration (property), 40  
 postprocessing\_steps (aisquared.config.ModelConfiguration.ModelConfiguration (property), 40  
 prediction\_key (aisquared.config.rendering.DocumentRendering.DocumentRendering (property), 28  
 preprocessor\_dict (aisquared.config.ModelConfiguration.ModelConfiguration (property), 40  
 preprocessing\_steps (aisquared.config.ModelConfiguration.ModelConfiguration (property), 40  
 preserve\_aspect\_ratio (aisquared.config.preprocessing.image.Steps.Resize (property), 18  
 probability\_key (aisquared.config.rendering.DocumentRendering.DocumentRendering (property), 28  
**Q**  
 qualifier (aisquared.config.rendering.FilterRendering.FilterRendering (property), 30  
 QualitativeFeedback (class in aisquared.config.feedback.QualitativeFeedback), 10  
 query\_keys (aisquared.config.harvesting.QueryParameterHarvester.QueryParameterHarvester (property), 13  
 query\_selector (aisquared.config.rendering.ContainerRendering.ContainerRendering (property), 26  
 QueryParameterHarvester (class in aisquared.config.harvesting.QueryParameterHarvester), 12  
**R**  
 Regression (class in aisquared.config.postprocessing.Regression), 15  
 RegressionFeedback (class in aisquared.config.feedback.RegressionFeedback), 10  
 remove\_digits (aisquared.config.preprocessing.text.Steps.RemoveCharacters (property), 23  
 remove\_punctuation (aisquared.config.preprocessing.text.Steps.RemoveCharacters (property), 23  
 remove\_users\_from\_group() (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient (method), 51  
 RemoveCharacters (class in aisquared.config.preprocessing.text.Steps), 23  
 render\_dict (aisquared.config.ModelConfiguration.ModelConfiguration (property), 40  
 rendering\_steps (aisquared.config.ModelConfiguration.ModelConfiguration (property), 40  
 Resize (class in aisquared.config.preprocessing.image.Steps), 18  
 result\_key (aisquared.config.rendering.WordRendering.WordRendering (property), 37  
 ReverseMLWorkflow (class in aisquared.config.analytic.ReverseMLWorkflow), 15  
 round (aisquared.config.postprocessing.Regression.Regression (property), 15  
**S**  
 secret (aisquared.config.analytic.DeployedAnalytic.DeployedAnalytic (property), 5  
 secret (aisquared.config.analytic.DeployedModel.DeployedModel (property), 6  
 secret (aisquared.config.analytic.ReverseMLWorkflow.ReverseMLWorkflow (property), 8

**share\_model\_with\_group()** (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 51  
**share\_model\_with\_user()** (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 52  
**SimpleFeedback** (class in aisquared.config.feedback.SimpleFeedback), 11  
**size** (aisquared.config.preprocessing.image.Steps.Resize property), 18  
**SOSRendering** (class in aisquared.config.rendering.SOSRendering), 36  
**source** (aisquared.config.rendering.FilterRendering.FilterRendering property), 30  
**split\_sentences** (aisquared.config.preprocessing.text.Steps.Tokenize property), 23  
**split\_words** (aisquared.config.preprocessing.text.Steps.Tokenize property), 23  
**stage** (aisquared.config.GraphConfiguration.GraphConfiguration property), 39  
**stage** (aisquared.config.ModelConfiguration.ModelConfiguration property), 40  
**start\_character** (aisquared.config.preprocessing.text.Steps.ConvertToWord property), 22  
**static\_position** (aisquared.config.rendering.ContainerRendering.ContainerRendering property), 26  
**stds** (aisquared.config.preprocessing.tabular.Steps.ZScore property), 20  
**step\_dict** (aisquared.config.preprocessing.image.ImagePreprocessing.ImagePreprocessor property), 16  
**step\_dict** (aisquared.config.preprocessing.text.TextPreprocessing.TextPreprocessor property), 24  
**SubtractValue** (class in aisquared.config.preprocessing.image.Steps), 18  
**T**  
**TableRendering** (class in aisquared.config.rendering.TableRendering), 36  
**TabularPreprocessor** (class in aisquared.config.preprocessing.tabular.TabularPreprocessing), 21  
**test\_connection()** (aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient method), 52  
**TextHarvester** (class in aisquared.config.harvesting.TextHarvester), 13  
**TextPreprocessor** (class in aisquared.config.preprocessing.text.TextPreprocessing), 24  
**thickness** (aisquared.config.rendering.ImageRendering.ImageRendering property), 32  
**thickness** (aisquared.config.rendering.ObjectRendering.ObjectRendering property), 34  
**threshold** (aisquared.config.postprocessing.BinaryClassification.BinaryClassification property), 14  
**threshold** (aisquared.config.postprocessing.ObjectDetection.ObjectDetection property), 15  
**threshold\_key** (aisquared.config.rendering.DocumentRendering.DocumentRendering property), 28  
**threshold\_key** (aisquared.config.rendering.ImageRendering.ImageRendering property), 32  
**threshold\_key** (aisquared.config.rendering.WordRendering.WordRendering property), 37  
**threshold\_value** (aisquared.config.rendering.DocumentRendering.DocumentRendering property), 28  
**threshold\_value** (aisquared.config.rendering.ImageRendering.ImageRendering property), 32  
**threshold\_value** (aisquared.config.rendering.WordRendering.WordRendering property), 37  
**to\_dict()** (aisquared.base.BaseObject.BaseObject method), 3  
**to\_dict()** (aisquared.base.CustomObject.CustomObject method), 4  
**to\_dict()** (aisquared.config.analytic.DeployedAnalytic.DeployedAnalytic method), 7  
**to\_dict()** (aisquared.config.analytic.DeployedModel.DeployedModel method), 7  
**to\_dict()** (aisquared.config.analytic.LocalAnalytic.LocalAnalytic method), 7  
**to\_dict()** (aisquared.config.analytic.LocalModel.LocalModel method), 7  
**to\_dict()** (aisquared.config.analytic.ReverseMLWorkflow.ReverseMLWorkflow method), 7  
**to\_dict()** (aisquared.config.feedback.BinaryFeedback.BinaryFeedback method), 8  
**to\_dict()** (aisquared.config.feedback.ModelFeedback.ModelFeedback method), 9  
**to\_dict()** (aisquared.config.feedback.MulticlassFeedback.MulticlassFeedback method), 10  
**to\_dict()** (aisquared.config.feedback.QualitativeFeedback.QualitativeFeedback method), 10  
**to\_dict()** (aisquared.config.feedback.RegressionFeedback.RegressionFeedback method), 10  
**to\_dict()** (aisquared.config.feedback.SimpleFeedback.SimpleFeedback method), 11  
**to\_dict()** (aisquared.config.GraphConfiguration.GraphConfiguration method), 39  
**to\_dict()** (aisquared.config.harvesting.ImageHarvester.ImageHarvester method), 11  
**to\_dict()** (aisquared.config.harvesting.InputHarvester.InputHarvester method), 12  
**to\_dict()** (aisquared.config.harvesting.QueryParameterHarvester.QueryParameterHarvester method), 13  
**to\_dict()** (aisquared.config.harvesting.TextHarvester.TextHarvester method), 13



`to_dict()` (`aisquared.config.ModelConfiguration.ModelConfiguration` method), 40  
`to_dict()` (`aisquared.config.postprocessing.BinaryClassificationConfig` method), 14  
`to_dict()` (`aisquared.config.postprocessing.MulticlassClassificationConfig` method), 15  
`to_dict()` (`aisquared.config.postprocessing.ObjectDetectionConfig` method), 15  
`to_dict()` (`aisquared.config.postprocessing.RegressionConfig` method), 15  
`to_dict()` (`aisquared.config.preprocessing.image.ImagePreprocessor` method), 16  
`to_dict()` (`aisquared.config.preprocessing.image.Steps.AddValue` method), 17  
`to_dict()` (`aisquared.config.preprocessing.image.Steps.ConvertToColor` method), 17  
`to_dict()` (`aisquared.config.preprocessing.image.Steps.DivideWhite` method), 17  
`to_dict()` (`aisquared.config.preprocessing.image.Steps.Multiply` method), 18  
`to_dict()` (`aisquared.config.preprocessing.image.Steps.Rotate` method), 18  
`to_dict()` (`aisquared.config.preprocessing.image.Steps.Subtract` method), 18  
`to_dict()` (`aisquared.config.preprocessing.tabular.Steps.DropColumns` method), 19  
`to_dict()` (`aisquared.config.preprocessing.tabular.Steps.MixPattern` method), 20  
`to_dict()` (`aisquared.config.preprocessing.tabular.Steps.Normalize` method), 20  
`to_dict()` (`aisquared.config.preprocessing.tabular.Steps.Scale` method), 20  
`to_dict()` (`aisquared.config.preprocessing.tabular.TabularPreprocessor` method), 21  
`to_dict()` (`aisquared.config.preprocessing.text.Steps.ConvertToCase` method), 22  
`to_dict()` (`aisquared.config.preprocessing.text.Steps.ConvertToVocabulary` method), 22  
`to_dict()` (`aisquared.config.preprocessing.text.Steps.PadSequences` method), 23  
`to_dict()` (`aisquared.config.preprocessing.text.Steps.RemoveCharacters` method), 23  
`to_dict()` (`aisquared.config.preprocessing.text.Steps.Tokenize` method), 23  
`to_dict()` (`aisquared.config.preprocessing.text.Steps.Trim` method), 24  
`to_dict()` (`aisquared.config.preprocessing.text.TextPreprocessor` method), 24  
`to_dict()` (`aisquared.config.rendering.BarChartRendering` method), 25  
`to_dict()` (`aisquared.config.rendering.ContainerRendering` method), 26  
`to_dict()` (`aisquared.config.rendering.DashboardReplacementRendering` method), 27  
`to_dict()` (`aisquared.config.rendering.DocumentRendering` method), 28  
`to_dict()` (`aisquared.config.rendering.DoughnutChartRendering` method), 30  
`to_dict()` (`aisquared.config.rendering.FilterRendering` method), 30  
`to_dict()` (`aisquared.config.rendering.HTMLTagRendering` method), 31  
`to_dict()` (`aisquared.config.rendering.ImageRendering` method), 32  
`to_dict()` (`aisquared.config.rendering.LineChartRendering` method), 34  
`to_dict()` (`aisquared.config.rendering.ObjectRendering` method), 34  
`to_dict()` (`aisquared.config.rendering.PieChartRendering` method), 35  
`to_dict()` (`aisquared.config.rendering.SOSRendering` method), 36  
`to_dict()` (`aisquared.config.rendering.TableRendering` method), 37  
`to_dict()` (`aisquared.config.rendering.WordRendering` method), 37  
`to_dict()` (`aisquared.base.BaseObject.BaseObject` method), 4  
`token_pattern` (`aisquared.config.preprocessing.text.Steps.Tokenize` property), 23  
`Tokenize` (class in `aisquared.config.preprocessing.text.Steps`), 23  
`Trim` (class in `aisquared.config.preprocessing.text.Steps`), 23  
`truncate_sequence_length` (`aisquared.config.preprocessing.text.Steps.PadSequences` property), 23  
`underline_color` (`aisquared.config.rendering.DocumentRendering` property), 28  
`unshare_model_with_group()` (`aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient` method), 52  
`unshare_model_with_user()` (`aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient` method), 53  
`update_group()` (`aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient` method), 53  
`update_user()` (`aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient` method), 53  
`upload_model()` (`aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient` method), 54  
`url` (`aisquared.config.analytic.DeployedAnalytic.DeployedAnalytic` property), 5  
`url1` (`aisquared.config.analytic.DeployedModel.DeployedModel` property), 6

`url` (`aisquared.config.GraphConfiguration.GraphConfiguration`  
    `property`), 39

`url` (`aisquared.config.ModelConfiguration.ModelConfiguration`  
    `property`), 40

`url_locations` (`aisquared.config.harvesting.QueryParameterHarvester.QueryParameterHarvester`  
    `property`), 13

`username` (`aisquared.platform.AISquaredPlatformClient.AISquaredPlatformClient`  
    `property`), 55

## V

`value` (`aisquared.config.preprocessing.image.Steps.AddValue`  
    `property`), 17

`value` (`aisquared.config.preprocessing.image.Steps.DivideValue`  
    `property`), 17

`value` (`aisquared.config.preprocessing.image.Steps.MultiplyValue`  
    `property`), 18

`value` (`aisquared.config.preprocessing.image.Steps.SubtractValue`  
    `property`), 18

`value` (`aisquared.config.rendering.FilterRendering.FilterRendering`  
    `property`), 31

`values` (`aisquared.config.preprocessing.tabular.Steps.OneHot`  
    `property`), 20

`version` (`aisquared.config.GraphConfiguration.GraphConfiguration`  
    `property`), 39

`version` (`aisquared.config.ModelConfiguration.ModelConfiguration`  
    `property`), 40

`vocabulary` (`aisquared.config.preprocessing.text.Steps.ConvertToVocabulary`  
    `property`), 22

## W

`where_replace` (`aisquared.config.rendering.DashboardReplacementRendering.DashboardReplacementRendering`  
    `property`), 27

`width` (`aisquared.config.rendering.ContainerRendering.ContainerRendering`  
    `property`), 27

`word_list` (`aisquared.config.rendering.WordRendering.WordRendering`  
    `property`), 37

`WordRendering` (class in  
    `aisquared.config.rendering.WordRendering`),  
    37

`words` (`aisquared.config.rendering.DocumentRendering.DocumentRendering`  
    `property`), 29

## X

`xoffset` (`aisquared.config.rendering.ContainerRendering.ContainerRendering`  
    `property`), 27

## Y

`yoffset` (`aisquared.config.rendering.ContainerRendering.ContainerRendering`  
    `property`), 27

## Z

`ZScore` (class in `aisquared.config.preprocessing.tabular.Steps`),  
    20