aisquared

The AI Squared Team

Feb 12, 2024

DOCUMENTATION

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: `0.3.11`

`View this Documentation in PDF Format`.

CHAPTER ONE

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 Subpackages

aisquared.base package

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.css module

aisquared.base.endpoints module

NOT MEANT TO BE CALLED BY THE END USER - configuration parameters for the different endpoints in the platform

aisquared.base.harvesting module

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

aisquared.base.platform module

aisquared.base.preprocessing module

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

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

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.

aisquared.config package

Subpackages

aisquared.config.analytic package

Submodules

aisquared.config.analytic.DeployedAnalytic module

class `aisquared.config.analytic.DeployedAnalytic.`**`DeployedAnalytic`**(*url: str*, *method: str*, *input_type: str*, *headers: dict | None = None*, *body: dict | None = None*)

Bases: `BaseObject`

Interaction with a remote endpoint.

Example usage:

```
>>> import aisquared
>>> analytic = aisquared.config.analytic.DeployedAnalytic(
    'model_url',
    'POST',
    'text',
    {
        'Content-Type' : 'application/json'
    },
    {
        'data_to_be_sent' : '{{input}}'
    }
)
>>> analytic.to_dict()
{'className': 'DeployedAnalytic',
'params': {'url': 'model_url',
'method': 'POST',
'inputType': 'text',
'headers': {'Content-Type': 'application/json'},
'body': {'data_to_be_sent': '{{input}}'}}}
```

property body

property headers

property input_type

property method

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*, *headers: dict | None = None*, *body_key: str | None = None*, *return_key: str | None = None*, *body_setup: dict | None = None*, *body_setup_replace_value: str = '{DATAPOINT}'*)

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',
'headers': None,
'bodyKey': None,
'returnKey': None,
'bodySetup': None,
'bodySetupReplaceValue': None
}}
```

property body_key

property body_setup

property body_setup_replace_value

property headers

property input_type

property return_key

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`**(*label: str*, *connector_type: str*, *connector_action: str = 'read'*, *input_type: str = 'text'*, *filter_type: str = 'input'*, *file_names: list = []*, *bucket: str = ''*, *filter_by_columns: list = []*, *all: bool = False*, *arn: str = ''*, *host: str = ''*, *path: str = ''*, *port: str = ''*, *role: str = ''*, *soql: str = ''*, *query: str = ''*, *token: str = ''*, *column: str = ''*, *db_name: str = ''*, *db_user: str = ''*, *period: int | None = None*, *schema: str = ''*, *secret: str = ''*, *account: str = ''*, *data_map: list = []*, *db_table: str = ''*, *client_id: str = ''*, *file_name: str = ''*, *password: str = ''*, *schedule: str = ''*, *sync_keys: dict = {'destination': '', 'source': ''}*, *warehouse: str = ''*, *data_source: str = ''*, *cluster_name: str = ''*, *client_secret: str = ''*, *organization: str = ''*, *authentication_type: str = ''*)

Bases: `BaseObject`

Creation of a ReverseML Workflow to interact with remote data sources

`to_dict``()` → dict

Get the configuration object as a dictionary

Module contents

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

aisquared.config.feedback package

Submodules

aisquared.config.feedback.BinaryFeedback module

`class aisquared.config.feedback.BinaryFeedback.`**`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()
{'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

Module contents

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

aisquared.config.harvesting package

Submodules

aisquared.config.harvesting.ChatbotHarvester module

class `aisquared.config.harvesting.ChatbotHarvester.`**`ChatbotHarvester`**(*title*, *harvest_history=False*, *input_type='text'*, *features=None*, *max_length=None*)

Bases: `BaseObject`

Harvesting for a chatbot

`to_dict()`

Return the configuration object as a dictionary

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: int | None = None*, *features: list | None = 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: str | list*, *url_locations: str | list*, *attributes: 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: str | None = None, flags: str = 'gu', body_only: bool = False, keywords: str | list | None = None, limit: int | None = 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

Module contents

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

aisquared.config.postprocessing package

Submodules

aisquared.config.postprocessing.BinaryClassification module

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

Bases: `BaseObject`

Postprocesssing 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: int | float | None = None*, *max: int | float | None = 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

Module contents

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

aisquared.config.preprocessing package

Subpackages

aisquared.config.preprocessing.image package

Submodules

aisquared.config.preprocessing.image.ImagePreprocessing module

class aisquared.config.preprocessing.image.ImagePreprocessing.**ImagePreprocesser**(*steps: list | None = None*)

Bases: `BaseObject`

Preprocesser object for image data

Example usage:

```
>>> import aisquared
>>> preprocesser = aisquared.config.preprocessing.image.ImagePreprocesser()
>>> preprocesser.add_step(
    aisquared.config.preprocessing.image.AddValue(255.0)
)
```

add_step(*step*)

Add a step to the preprocesser 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: int | float*)

Bases: `BaseObject`

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

Example usage:

```
>>> import aisquared
>>> preprocesser = aisquared.config.preprocessing.image.ImagePreprocesser()
>>> preprocesser.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
>>> preprocesser = aisquared.config.preprocessing.image.ImagePreprocesser()
>>> preprocesser.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: int | float*)

Bases: `BaseObject`

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

Example usage:

```
>>> import aisquared
>>> preprocesser = aisquared.config.preprocessing.image.ImagePreprocesser()
>>> preprocesser.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: int | float*)

Bases: `BaseObject`

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

Example usage:

```
>>> import aisquared
>>> preprocesser = aisquared.config.preprocessing.image.ImagePreprocesser()
>>> preprocesser.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
>>> preprocesser = aisquared.config.preprocessing.image.ImagePreprocesser()
>>> preprocesser.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: int | float*)

Bases: *BaseObject*

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

Example usage:

```
>>> import aisquared
>>> preprocesser = aisquared.config.preprocessing.image.ImagePreprocesser()
>>> preprocesser.add_step(
    aisquared.config.preprocessing.image.SubtractValue(255.0)
)
```

to_dict() → dict

Get the configuration object as a dictionary

property value

Module contents

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

aisquared.config.preprocessing.tabular package

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
>>> preprocesser = aisquared.config.preprocessing.tabular.TabularPreprocesser()
>>> preprocesser.add_step(
```

(continues on next page)

(continued from previous page)

```
    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: list | None = 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
>>> preprocesser = aisquared.config.preprocessing.tabular.TabularPreprocesser()
>>> preprocesser.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
>>> preprocesser = aisquared.config.preprocessing.tabular.TabularPreprocesser()
>>> preprocesser.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: int | list | None = 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
>>> preprocesser = aisquared.config.preprocessing.tabular.TabularPreprocesser()
>>> preprocesser.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.**TabularPreprocesser**(*steps: list | None = None*)

Bases: `BaseObject`

Preprocesser object for tabular data

Example usage:

Example usage:

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

add_step(*step*)

Add a step to the preprocesser object

to_dict()

Get the configuration object as a dictionary

Module contents

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

aisquared.config.preprocessing.text package

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
>>> preprocesser = aisquared.config.preprocessing.text.TextPreprocesser()
>>> preprocesser.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: int | None = None*)

Bases: `BaseObject`

Text preprocessing object to convert tokens to integer vocabularies

Example usage:

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

(continues on next page)

(continued from previous page)

```
    )
)
```

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
>>> preprocesser = aisquared.config.preprocessing.text.TextPreprocesser()
>>> preprocesser.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
>>> preprocesser = aisquared.config.preprocessing.text.TextPreprocesser()
>>> preprocesser.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
>>> preprocesser = aisquared.config.preprocessing.text.TextPreprocesser()
>>> preprocesser.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
>>> preprocesser = aisquared.config.preprocessing.text.TextPreprocesser()
>>> preprocesser.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.**TextPreprocesser**(*steps: list | None = None*)

Bases: `BaseObject`

Preprocesser object for natural language

Example usage:

```
>>> import aisquared
>>> preprocesser = aisquared.config.preprocessing.text.TextPreprocesser()
>>> preprocesser.add_step(
    aisquared.config.preprocessing.text.Tokenize()
)
```

add_step(*step*)

Add a step to the preprocesser object

property step_dict

to_dict() → dict

Get the configuration object as a dictionary

Module contents

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

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.

aisquared.config.rendering package

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: str | None = None*, *width: str = 'auto'*, *height: str = 'auto'*, *xOffset: str = '0'*, *yOffset: str = '0'*, *labels: list | None = None*, *consolidate_rows: bool = True*, *css_params: dict | None = None*)

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.ChatRendering module

class aisquared.config.rendering.ChatRendering.**ChatRendering**(*return_key*, *prediction_value_key=None*, *sender_name='You'*, *responder_name='Chatbot'*)

Bases: *BaseObject*

Rendering for a chatbot use case

property prediction_value_key

property responder_name

property return_key

property sender_name

to_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: str | None = None*, *width: str = 'auto'*, *height: str = 'auto'*, *display: str = 'flex'*, *xOffset: str = '0'*, *yOffset: str = '0'*, *orientation: str = 'column'*, *css_params: dict | None = None*)

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.CustomRendering module

`class aisquared.config.rendering.CustomRendering.`**CustomRendering**(*id: str*, *content_html: str*, *content_script: str*, *content_style: str*, *query_selector: str | None = None*)

Bases: `BaseObject`

property content_html

property content_script

property content_style

property id

property query_selector

to_dict()

Get the object as a dictionary

aisquared.config.rendering.DashboardRendering module

THIS MODULE IS IN DEVELOPMENT AND NOT STABLE. PLEASE USE WITH CAUTION AND DO NOT USE FOR ANY PRODUCTION WORKLOADS

`class aisquared.config.rendering.DashboardRendering.`**DashboardRendering**

Bases: `BaseObject`

THIS CLASS IS IN DEVELOPMENT AND IS NOT STABLE. PLEASE USE WITH CAUTION AND DO NOT USE FOR ANY PRODUCTION WORKLOADS

add_bar_chart(*container_id: str*, *prediction_name_key: str*, *prediction_value_key: str*, *prediction_name_value: str*, *chart_colors: list*, *chart_labels: list*, *width: str = 'auto'*, *height: str = 'auto'*, *xOffset: str = '0'*, *yOffset: str = '0'*, *id: str | None = None*, *label: str | None = None*, *chart_name: str | None = None*)

add_container(*query_selector: str, width: str = 'auto', height: str = 'auto', display: str = 'flex', xOffset: str = '0', yOffset: str = '0', position: str = '', orientation: str = 'column', id: str | None = None, label: str | None = None*)

add_doughnut_chart(*container_id: str, prediction_name_key: str, prediction_value_key: str, prediction_name_value: str, chart_colors: list, chart_labels: list, display_legend: bool = True, legend_icon: str = 'circle', width: str = 'auto', height: str = 'auto', xOffset: str = '0', yOffset: str = '0', id: str | None = None, label: str | None = None, chart_name: str | None = None*)

add_html_tag(*container_id: str, html_content: str, prediction_name_key: str = '', prediction_value_key: str = '', prediction_name_value: str = '', extra_content_tag: str = 'strong', injection_action: str = 'prepend', id: str | None = None, content: str = '', label: str | None = None*)

add_line_chart(*container_id: str, prediction_name_key: str, prediction_value_key: str, prediction_name_value: str, chart_colors: list, chart_labels: list, width: str = 'auto', height: str = 'auto', xOffset: str = '0', yOffset: str = '0', id: str | None = None, label: str | None = None, chart_name: str | None = None*)

add_pie_chart(*container_id: str, prediction_name_key: str, prediction_value_key: str, prediction_name_value: str, chart_colors: list, chart_labels: list, display_legend: bool = True, legend_icon: str = 'circle', width: str = 'auto', height: str = 'auto', xOffset: str = '0', yOffset: str = '0', id=None, label: str | None = None, chart_name: str | None = None*)

add_table(*container_id: str, prediction_name_key: str, prediction_value_key: str, prediction_name_values: str, table_name: str = '', id: str | None = None, label: str | None = None*)

property steps

to_dict()

Get the object as a dictionary

aisquared.config.rendering.DashboardReplacementRendering module

class aisquared.config.rendering.DashboardReplacementRendering.**DashboardReplacementRendering**(*anchor_selec str, where_replac str = '', la- bel: str = ''*)

Bases: *BaseObject*

Rendering for dashboard replacement

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.rendering.DashboardReplacementRendering(
    'test_anchor_selector'
```

(continues on next page)

(continued from previous page)

```
)
>>> 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: list | dict | str | None = None, documents: list | dict | str | None = None, include_probability: bool = False, probability_key: str = 'probability', underline_color: str = 'blue', classes: list | None = None, threshold_key: str | None = None, threshold_value: int | float | None = 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*, *prediction_name_key: str*, *prediction_value_key: str*, *prediction_name_value: str*, *display_legend: bool*, *legend_icon: str*, *labels_key: str | None = None*, *width: str = 'auto'*, *height: str = 'auto'*, *xOffset: str = '0'*, *yOffset: str = '0'*, *labels: list | None = None*, *consolidate_rows: bool = True*, *css_params: dict | None = None*)

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,
    '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: 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',
```

(continues on next page)

(continued from previous page)

```
    '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 = ''*, *css_params: dict | None = None*)

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',
```

(continues on next page)

(continued from previous page)

```
'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: list | None = None*, *threshold_key: str | None = None*, *threshold_value: int | float | None = 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.**LineChartRendering**(*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: str*, *width: str = 'auto'*, *height: str = 'auto'*, *xOffset: str = '0'*, *yOffset: str = '0'*, *labels: list | None = None*, *consolidate_rows: bool = True*, *css_params: dict | None = None*)

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()
```

(continues on next page)

(continued from previous page)

```
{'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,
    'predictionNameKey': 'name',
    'predictionValueKey': 'value',
    'predictionNameValue': 'name_value'}]}}
```

to_dict() → dict

Get the configuration object as a dictionary

aisquared.config.rendering.ObjectRendering module

class aisquared.config.rendering.ObjectRendering.**ObjectRendering**(*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'*)

Bases: *BaseObject*

Object which dictates how to render object detection in images

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.rendering.ObjectRendering()
>>> my_obj.to_dict()
{'className': 'ObjectRendering',
'params': {'color': 'blue',
'thickness': '5',
'placement': 'bottomleft',
'includeProbability': False,
'badgeColor': 'white',
'fontColor': 'black',
'fontSize': '5'}}
```

property badge_color

property color

property font_color

property font_size

property include_probability

property placement

property thickness

to_dict() → dict

Get the configuration object as a dictionary

aisquared.config.rendering.PieChartRendering module

class aisquared.config.rendering.PieChartRendering.**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: str | None = None*, *width: str = 'auto'*, *height: str = 'auto'*, *xOffset: str = '0'*, *yOffset: str = '0'*, *labels: list | None = None*, *consolidate_rows: bool = True*, *css_params: dict | None = None*)

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',
```

(continues on next page)

(continued from previous page)

```
'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 = ''*, *css_params: dict | None = None*)

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

to_dict() → dict

Get the configuration object as a dictionary

aisquared.config.rendering.TextRendering module

class aisquared.config.rendering.TextRendering.**TextRendering**(*prediction_value_key: str | None = None*)

Bases: *BaseObject*

Class for rendering text

Example usage:

```
>>> import aisquared
>>> my_obj = aisquared.config.rendering.TextRendering(
    prediction_value_key = 'my_key'
)
>>> my_obj.to_dict()
{'className': 'TextRendering',
'params': {'predictionValueKey': 'my_key'}}
```

property prediction_value_key

to_dict()

Get the object as a dictionary

aisquared.config.rendering.WordRendering module

class aisquared.config.rendering.WordRendering.**WordRendering**(*word_list: str = 'input', result_key: str | None = None, content_key: str | None = None, badge_shape: str = 'star', badge_color: str = 'blue', classes: list | None = None, threshold_key: str | None = None, threshold_value: int | float | None = None, position: str = 'after'*)

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
'position': 'after'}}
```

property badge_color

property badge_shape

property classes

property content_key

property position

property result_key

property threshold_key

property threshold_value

to_dict() → dict

Get the configuration object as a dictionary

property word_list

aisquared.config.rendering.utils module

`aisquared.config.rendering.utils.`**`save_default_css`**`()`

Save default CSS so that default CSS can be edited and automatically utilized with changes

Notes

- Saves all CSS files to the *~/.aisquared/* directory

Module contents

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

Submodules

aisquared.config.CustomObject module

`class aisquared.config.CustomObject.`**`CustomObject`**`(`*class_name: str*, *top_level_kwargs: dict | None = 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.config.GraphConfiguration module

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

Bases: `BaseObject`

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

add_node(*step: BaseObject, dependencies: int | list | None = 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: str | None = None, dtype: str | None = 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

property documentation_link

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: BaseObject | list | None = None*, *preprocessing_steps: BaseObject | list | None = None*, *analytic: BaseObject | list | None = None*, *postprocessing_steps: BaseObject | list | None = None*, *rendering_steps: BaseObject | list | None = None*, *feedback_steps: BaseObject | list | None = None*, *stage: str = 'experimental'*, *version: int | None = None*, *description: str = ''*, *mlflow_uri: str | None = None*, *mlflow_user: str | None = None*, *mlflow_token: str | None = None*, *owner: str | None = None*, *url: str = '*'*, *auto_run: bool = False*, *documentation_link: str = ''*, *warnings: list | None = None*)

Bases: `BaseObject`

Configuration object for deploying a model or analytic

`property analytic`

`property analytic_dict`

`property auto_run`

`compile`(*filename: str | None = None*, *dtype: str | None = 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 documentation_link`

`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 postprocesser_dict

property postprocessing_steps

property preprocesser_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

property warnings

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

aisquared.logging package

Module contents

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

This functionality is inhereted from MLFlow. Please see the MFLow documentatation at https://mlflow.org.

aisquared.platform package

Submodules

aisquared.platform.AISquaredAPIException module

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

Bases: Exception

aisquared.platform.AISquaredPlatformClient module

class aisquared.platform.AISquaredPlatformClient.**AISquaredPlatformClient**(*use_port: bool = False*)

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()
True
```

add_users_to_group(*group_id: str*, *user_ids: list*, *port: int = 8086*, *use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
success – Returns True if operation was successful

Return type
bool

property base_url: str

The base URL associated with the client

create_group(*display_name: str*, *role_id: str*, *port: int = 8086*, *use_port: bool | None = None*) → dict

Create a group in the platform

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

Parameters

- **display_name** (*str*) – The display name of the group
- **role_id** (*str*) – The role ID for the group
- **port** (*int (default 8086)*) – The API port for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

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: str | None = None*, *company_id: str | None = None*, *password: str | None = None*, *port: int = 8085*, *use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
user_data – Metadata about the user

Return type
dict

delete_group(*group_id*, *port=8086*, *use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
result – Returns True if successful

Return type
bool

delete_model(*id: str*, *port: int = 8080*, *use_port: bool | None = None*) → 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. This can be handled automatically by the platform ALB

- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
success – Whether the action was successful

Return type
bool

delete_user(*user_id: str*, *port: int = 8085*, *use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
result – Returns True if the call is successful

Return type
bool

get_group(*group_id: str*, *port: int = 8086*, *use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
group_info – The information about the group

Return type
dict

get_group_id_by_name(*group_name: str*, *port: int = 8083*, *use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
group_id – The ID of the group

Return type
str

get_model(*id: str*, *port: int = 8080*, *use_port: bool | None = None*) → 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. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
model – 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*, *use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
model_id – The model's ID

Return type
str

get_role_id_by_role_name(*role_name: str*, *port: int = 8086*, *use_port: bool | None = None*) → str

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

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

Parameters

- **role_name** (*str*) – The name of the role
- **port** (*int (default 8086)*) – The API port for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
role_id – The ID of the role

Return type
str

get_user(*user_id: str*, *port: int = 8085*, *use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
user_info – The information about the user

Return type
dict

get_user_id_by_name(*name: str*, *port: int = 8080*, *use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

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*, *use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
users – The response from the API

Return type
pandas DataFrame or dictionary

list_groups(*max_count: int = 100*, *as_df: bool = True*, *port: int = 8083*, *use_port: bool | None = None*) → DataFrame | dict

List all groups

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

Parameters

- **max_count** (*int (default 100)*) – The maximum number of groups to return
- **as_df** (*bool (default True)*) – Whether to return the result as a pandas DataFrame
- **port** (*int (default 8083)*) – The API port for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

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*, *use_port: bool | None = None*) → 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. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
feedback – The feedback

Return type
dict or pandas DataFrame

list_model_prediction_feedback(*model_id: str*, *as_df: bool = True*, *port: int = 8080*, *use_port: bool | None = None*) → 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. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

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*, *use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

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*, *use_port: bool | None = None*) → 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. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
model_users – The users for the model

Return type
pandas DataFrame or dictionary

`list_models`(*as_df: bool = True*, *port: int = 8080*, *use_port: bool | None = None*) → 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** (*default None*) – The API port for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
models – The models

Return type
pandas DataFrame or dictionary

`list_prediction_feedback`(*prediction_id: str*, *as_df: bool = True*, *port: int = 8080*, *use_port: bool | None = None*) → 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. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
results – The results from the platform

Return type
pandas DataFrame or dict

list_roles(*as_df: bool = True*, *port: int = 8086*, *use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

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*, *use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
results – The results from the platform

Return type
pandas DataFrame or dict

list_users(*max_count: int = 100*, *as_df: bool = True*, *port: int = 8080*, *use_port: bool | None = None*) → DataFrame | dict

List all users

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

Parameters

- **max_count** (*int (default 100)*) – The maximum number of users to return
- **as_df** (*bool (default True)*) – Whether to return the data as a Pandas DataFrame
- **port** (*int (default 8080)*) – The API port for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
users – The response from the API

Return type
pandas DataFrame or dictionary

login(*url: str | None = None*, *port: int = 8080*, *username: str | None = None*, *password: str | None = None*, *use_port: bool | None = 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
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 or None (default 8080)*) – The API port for the call. This can be handled automatically by the platform ALB
- **username** (*str or None (default None)*) – The username

- **password** (*str or None (default None)*) – The password
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

property password: str

The password associated with the client

remove_users_from_group(*group_id: str, user_ids: list, port: int = 8086, use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns

success – Returns True if successful

Return type

bool

share_model_with_group(*model_id: str, group_id: str, port: int = 8080, use_port: bool | None = None*) → 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. This can be handled automatically by the platform ALB
- **port** (*int (default 8080)*) – The API port to use. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns

success – Returns True if successful

Return type
bool

share_model_with_user(*model_id: str*, *user_id: str*, *port: int = 8080*, *use_port: bool | None = None*) → 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. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
success – Whether the action was successful

Return type
bool

test_connection(*port: int = 8080*, *use_port: bool | None = None*) → bool

Test whether there is a healthy connection to the platform

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.test_connection()
True
```

Parameters

- **port** (*int (default 8080)*) – The API port for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
success – True if connection was successful

Return type
bool

property token: str

The token associated with the client

unshare_model_with_group(*model_id: str*, *group_id: str*, *port: int = 8080*, *use_port: bool | None = None*) → 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. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
success – Returns True if successful

Return type
bool

unshare_model_with_user(*model_id: str*, *user_id: str*, *port: int = 8080*, *use_port: bool | None = None*) → 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. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
success – Whether the action was successful

Return type
bool

update_group(*group_id: str*, *display_name: str*, *role_id: str*, *port: int = 8086*, *use_port: bool | None = None*) → bool

Update information about a group

```
>>> import aisquared
>>> client = aisquared.platform.AISquaredPlatformClient()
>>> client.update_group(
```

(continues on next page)

(continued from previous page)

```
    '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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

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: str | None = None*, *company_id: str | None = None*, *password: str | None = None*, *port: int = 8085*, *use_port: bool | None = None*) → 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 for the call. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
success – Returns True if update is successful

Return type
bool

upload_model(*model_file: str*, *port: int = 8081*, *use_port: bool | None = None*) → 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. This can be handled automatically by the platform ALB
- **use_port** (*bool or None (default None)*) – Whether to use port in URL formatting. If None, defaults to class value

Returns
successful – Whether the action was successful

Return type
bool

property use_port

property username: str

The username associated with the client

aisquared.platform.DatabricksAPIException module

exception aisquared.platform.DatabricksAPIException.**DatabricksAPIException**

Bases: Exception

aisquared.platform.DatabricksClient module

`class aisquared.platform.DatabricksClient.`**`DatabricksClient`**

Bases: `object`

Client for working with a connected Databricks environment

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

```
>>> import aisquared
>>> client = aisquared.platform.DatabricksClient()
>>> # If you have never logged in before, run the following code:
>>> client.login()
>>> # Interactive session requesting required information
```

`property base_url: str`

The base URL for the workspace

`create_compute`(*compute_name: str*, *spark_version: str*, *node_type_id: str*) → dict

Create a compute resource

Parameters

- **`compute_name`** (*`str`*) – The name of the compute to create
- **`spark_version`** (*`str`*) – The spark version to use for the compute resource
- **`node_type_id`** (*`str`*) – The node type ID to use

Returns

compute_info – The information about the created compute resource

Return type

dict

`create_job`(*job_name: str*, *tasks: list*, *libraries: list*, *compute_name: str*, *spark_version: str*, *node_type_id: str*, *cron_syntax: str | None = None*, *timezone: str | None = None*) → bool

Create a job using notebooks and/or scripts in the workspace

Parameters

- **`job_name`** (*`str`*) – The name for the job
- **`tasks`** (*`list of dict`*) – List of {task_name : task_script} dictionary pairs to run in the job
- **`libraries`** (*`list of str`*) – The dependent libraries to install on all compute
- **`compute_name`** (*`str`*) – The name of the compute to provision specifically for this job
- **`spark_version`** (*`str`*) – The version of Spark to use on the compute instances
- **`node_type_id`** (*`str`*) – The node type to use
- **`cron_syntax`** (*`str or None (default None)`*) – If the job is to be set to a schedule, the cron syntax for that schedule
- **`timezone`** (*`str or None (default None)`*) – The timezone to set the schedule to, if cron syntax is provided

Returns

success – Whether the create job call was successful

Return type
bool

`create_served_model`(*model_name: str*, *model_version: str*, *workload_size: str*, *scale_to_zero_enabled: bool = True*, *workload_type: str = 'CPU'*) → dict

Create a model serving endpoint

Parameters

- `model_name` (*str*) – The name of the model to serve
- `model_version` (*str*) – The version of the model to serve
- `workload_size` (*str*) – The workload size of the serving endpoint
- `scale_to_zero_enabled` (*bool (default True)*) – Whether to allow for scaling the endpoint to zero
- `type` (*workload*) – The workload type - either 'CPU' or 'GPU'

Returns
configuration – Configuration information about the serving endpoint

Return type
dict

`delete_compute`(*compute_id: str*) → bool

Delete a compute resource in the workspace

Parameters
`compute_id` (*str*) – The ID for the compute to delete

Returns
success – Whether the operation was successful

Return type
bool

`delete_from_workspace`(*filename: str*) → bool

Delete a file from the workspace

Parameters
`filename` (*str*) – The name of the file to delete

Returns
success – Whether the operation is successful

Return type
bool

`delete_job`(*job_id: str*) → bool

Delete a job from the workspace

Parameters
`job_id` (*str*) – The ID of the job to delete

Returns
success – Whether the delete operation was successful

Return type
bool

`delete_registered_model`(*model_name: str*) → bool

Delete a registered model

Parameters
`model_name` (*str*) – The name of the model to delete

Returns
success – Whether the delete operation was successful

Return type
bool

`delete_served_model`(*model_name: str*) → bool

Delete a served model in the workspace

Parameters
`model_name` (*str*) – The name of the model to delete

Returns
success – Whether the delete operation was successful

Return type
bool

`download_from_workspace`(*filename: str*) → str

Download a file from the workspace

Parameters
`filename` (*str*) – The filename of the file to download

Returns
contents – The contents of the file

Return type
str

`property headers: dict`

API headers for calls to the API

`list_compute`(*as_df: bool = True*) → dict | DataFrame

List compute in the workspace

Parameters
`as_df` (*bool (default True)*) – Whether to return a pandas DataFrame

Returns
compute – The compute resources in the workspace

Return type
dict or pd.DataFrame

`list_jobs`(*as_df: bool = True*) → dict | DataFrame

List all jobs in the workspace

Parameters
`as_df` (*bool (default True)*) – Whether to return a pandas DataFrame

Returns
jobs – The jobs that exist in the workspace

Return type
dict or pandas DataFrame

list_registered_models(*as_df: bool = True*) → dict | DataFrame

List registered models in the workspace

Parameters
as_df (*bool (default True)*) – Whether to return a pandas DataFrame

Returns
models – The models in the workspace

Return type
dict or pandas DataFrame

list_served_models(*as_df: bool = True*) → dict | DataFrame

List served models in the workspace

Parameters
as_df (*bool (default True)*) – Whether to return results as a pandas DataFrame

Returns
models – The models served in the workspace

Return type
dict or pandas DataFrame

list_workspace(*as_df: bool = True*) → DataFrame | dict

List files in the connected Databricks workspace

Parameters
as_df (*bool (default True)*) – Whether to return the results as a pandas DataFrame

Returns
results – The files in the workspace

Return type
dict or pd.DataFrame

login(*url: str | None = None*, *username: str | None = None*, *token: str | None = None*, *persist: bool = True*) → None

Log in to the Databricks environment programmatically

```
>>> import aisquared
>>> client = aisquared.platform.DatabricksClient()
>>> client.login()
Enter URL: {Databricks_workspace_url}
Enter Username: your.email@your_domain.com
Enter Secret Token: <hidden>
```

Parameters

- **url** (*str or None (default None)*) – The URL of the Databricks workspace
- **username** (*str or None (default None)*) – The username in the Databricks workspace
- **token** (*str or None (default None)*) – The secret token for the Databricks workspace
- **persist** (*bool (default True)*) – Whether to persist the login information, eliminating the need to run this command again in the future

`run_job`(*job_id: str*) → int

Run a job

Parameters
`job_id` (*`str`*) – The ID of the job to run

Returns
run_id – The ID of the specific run that was created

Return type
int

`start_compute`(*compute_id: str*) → bool

Start a compute resource

Parameters
`compute_id` (*`str`*) – The ID of the compute to start

Returns
success – Whether the start operation was successful

Return type
bool

`stop_compute`(*compute_id: str*) → bool

Stop a compute resource

Parameters
`compute_id` (*`str`*) – The ID of the compute to start

Returns
success – Whether the stop operation was successful

Return type
bool

`property token: str`

The token to use for the workspace

`update_job`(*job_id: int*, *job_name: str*, *tasks: list*, *libraries: list*, *compute_name: str*, *spark_version: str*, *node_type_id: str*, *cron_syntax: str | None = None*, *timezone: str | None = None*) → bool

Update a job by Job ID using notebooks and/or scripts in the workspace

Parameters

- **`job_id`** (*`int`*) – The unique identifier of the job to update
- **`job_name`** (*`str`*) – The new name for the job
- **`tasks`** (*`list of dict`*) – List of {task_name : task_script} dictionary pairs to run in the updated job
- **`libraries`** (*`list of str`*) – The dependent libraries to install on all compute for the new job
- **`compute_name`** (*`str`*) – The name of the compute to provision specifically for the new job
- **`spark_version`** (*`str`*) – The version of Spark to use on the compute instances
- **`node_type_id`** (*`str`*) – The node type to use
- **`cron_syntax`** (*`str or None (default None)`*) – If the new job is to be set to a schedule, the cron syntax for that schedule

- **timezone** (*str or None (default None)*) – The timezone to set the schedule to, if cron syntax is provided

Returns
success – Whether the update job call was successful

Return type
bool

upload_to_workspace(*filename: str*, *overwrite: bool = False*) → bool

Upload a file to the workspace

Parameters

- **filename** (*str*) – The name of the file to upload
- **overwrite** (*bool (default False)*) – Whether to overwrite the file if one of the same name already exists in the workspace

Returns
success – Whether the upload was successful

Return type
bool

property username: str

The user's username

aisquared.platform.NoResultsFoundError module

exception aisquared.platform.NoResultsFoundError.**NoResultsFoundError**

Bases: Exception

aisquared.platform.additional_utils module

aisquared.platform.crudl module

aisquared.platform.feedback module

aisquared.platform.metrics module

aisquared.platform.sharing module

aisquared.platform.user_group module

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.

aisquared.serving package

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: dict | None = None*, *additional_functions_file: str | None = 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 BeyondML 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_beyondml_model`**(*model: str*, *custom_objects: dict*)

Load a BeyondML model with custom objects

aisquared.serving.get_remote_prediction module

`aisquared.serving.get_remote_prediction.`**`get_remote_prediction`**(*data: 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

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*
```

aisquared.utils package

Submodules

aisquared.utils.utils module

`aisquared.utils.utils.`**`get_model`**(*model_type: str*, *input_shape: int | tuple*, *num_outputs: int*, *output_activation: str*, *size: str = 'small'*, *vocab_size: 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: 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

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

1.1.1.2 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.

CHAPTER

TWO

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 postprocesser
  - 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 delting 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*
- **Version 0.3.3**
  - Updated functionality of the *AISquaredPlatformClient* to interact directly with the platform ALB
  - Changed function names in support of change from MANN to BeyondML
  - Added documentation surrounding global configuration objects
  - Removed redundant additional dependencies
- **Version 0.3.4**
  - Added support for custom CSS strings to appropriate rendering classes
  - Refactored *AISquaredPlatformClient* to import functions from support files
  - Fixed documentation errors for the documentation site
  - Checked whether responses returned OK status code rather than 200
  - Moved *CustomObject* to *aisquared.config* from *aisquared.base*
  - Changed endpoint used to list platform users
  - Fixed response behaviors where no data was returned from *AISquaredPlatformClient*
- **Version 0.3.5**
  - Changed *file_name* parameter in *ReverseMLWorkflow* to *file_names*
  - Added *documentation_link* parameter to *ModelConfiguration* class
- **Version 0.3.6**
  - Fixed issue with type checking for *ModelConfiguration* Rendering classes
  - Restricted TensorFlow version to below *2.12.0* to prevent import issues
  - Added *position* parameter to *WordRendering* class
  - Changed default CSS styling for rendering classes
  - Changed name of all *processor* classes to *processer*

- **Version 0.3.7**
  - Changed schema of the *DeployedAnalytic* class to include API key management
  - Changed JSON schema of Preprocesser classes
  - Allowed .keras files to be saved and loaded with the *ModelConfiguration* and *GraphConfiguration* APIs into *.air* files
  - Relaxed TensorFlow requirements enforced in version *0.3.6*
- **Version 0.3.8**
  - Created *ChatbotHarvester* class
  - Created *TextRendering* class
  - Changed location of reference lists of classes to clean up code
  - Updated class schemas to ensure compliance with expectations
  - Updated test cases
- **Version 0.3.9**
  - Created *CustomRendering* class
  - Changed to full import of *CustomObject* in *aisquared.base* subpackage
- **Version 0.3.10**
  - Added *DatabricksClient* to the *aisquared.platform* subpackage
- **Version 0.3.11**
  - Updated *DeployedModel* class configuration to conform to AIRJS
  - Updated *DatabricksClient* class to include *update_job* function
  - Updated custom CSS fields in rendering classes
  - Reconfigured *ReverseMLWorkflow* class
  - Added *'User-Agent'* to headers for *AISquaredPlatformClient* and *DatabricksClient*
  - Added *llmlink* as a dependency to the 'full' installation of *aisquared* and added it as a top-level package
- **Version 0.3.12**
  - Updated *DeployedModel* class to support more abstract API calls
  - Updated *ChatbotHarvester*, *DeployedAnalytic*, and *ChatRendering* classes
  - Updated *ModelConfiguration* class with *warnings* and *documentURL*

PYTHON MODULE INDEX

INDEX