skippa Documentation

Release 0.1.15

Robert van Straalen

Nov 18, 2022

CONTENTS:

CHAPTER

ONE

INTRODUCTION

SciKIt-learn Pipeline in PAndas

Want to create a machine learning model using pandas & scikit-learn? This should make your life easier.

Skippa helps you to easily create a pre-processing and modeling pipeline, based on scikit-learn transformers but preserving pandas dataframe format throughout all pre-processing. This makes it a lot easier to define a series of subsequent transformation steps, while referring to columns in your intermediate dataframe.

1.1 Installation

```
$ pip install skippa
```

1.2 Basic use

Skippa helps you to easily define data cleaning & pre-processing operations on a pandas DataFrame and combine it with a scikit-learn model/algorithm into a single executable pipeline. It works roughly like this:

```
from skippa import Skippa, columns
from sklearn.linear_model import LogisticRegression
pipeline = (
   Skippa()
   .impute(columns(dtype_include='object'), strategy='most_frequent')
   .impute(columns(dtype_include='number'), strategy='median')
   .scale(columns(dtype_include='number'), type='standard')
   .onehot(columns(['category1', 'category2']))
   .model(LogisticRegression())
)
pipeline.fit(X, y)
predictions = pipeline.predict_proba(new_data)
```

CHAPTER

TWO

SKIPPA PACKAGE

2.1 Subpackages

2.1.1 skippa.transformers package

Submodules

skippa.transformers.base module

This contains base / utility classes and functions needed for defining/using transformers

class skippa.transformers.base.**ColumnSelector**(*selector*)

Bases: object

This is not a transformer, but a utility class for defining a column set.

class skippa.transformers.base.**SkippaMixin**

Bases: object

Utility class providing additional methods for custom Skippa transformers.

skippa.transformers.base.**columns**(**args*, *include=None*, *exclude=None*, ***kwargs*)

Helper function for creating a ColumnSelector

Flexible arguments: - include or exclude lists: speak for themselves - dtype_include, dtype_exclude, pattern: dispatched to sklearn's make_column_selector - otherwise: a list to include, or an existing ColumnSelector

Parameters

- **include** (*Optional[ColumnExpression], optional*) – [description]. Defaults to None.
- **exclude** (*Optional[ColumnExpression], optional*) – [description]. Defaults to None.

Returns

A callable that returns columns names, when called on a df

Return type

ColumnSelector

skippa.transformers.custom module

This defines custom transformers implementing anything other than existing skleafrn treansformers.

class skippa.transformers.custom.**SkippaApplier**(*cols*, **args*, ***kwargs*)

Bases: BaseEstimator, TransformerMixin, *SkippaMixin*

Transformer for applying arbitrary function (wraps around pandas apply)

fit(*X*, *y=None*, ***fit_params*)

Nothing to do here

transform(*X*, *y=None*, ***transform_params*)

Use pandas.DataFrame.apply method

class skippa.transformers.custom.**SkippaAssigner**(***kwargs*)

Bases: BaseEstimator, TransformerMixin, *SkippaMixin*

Transformer for selecting a subset of columns in a df.

fit(*X*, *y=None*, ***kwargs*)

transform(*X*, *y=None*, ***kwargs*)

class skippa.transformers.custom.**SkippaCaster**(*cols*, *dtype*)

Bases: BaseEstimator, TransformerMixin, *SkippaMixin*

Transformer for casting columns to another data type

fit(*X*, *y=None*, ***kwargs*)

Nothing to do here.

transform(*X*, *y=None*, ***kwargs*)

Apply the actual casting using pandas.astype

class skippa.transformers.custom.**SkippaConcat**(*left*, *right*)

Bases: BaseEstimator, *SkippaMixin*

Concatenate two pipelines.

fit(*X*, *y=None*, ***kwargs*)

transform(*X*, *y=None*, ***kwargs*)

class skippa.transformers.custom.**SkippaDateEncoder**(*cols*, ***kwargs*)

Bases: BaseEstimator, TransformerMixin, *SkippaMixin*

Derive date features using pandas datatime's .dt property.

fit(*X*, *y=None*)

transform(*X*, *y=None*, ***kwargs*)

class skippa.transformers.custom.**SkippaDateFormatter**(*cols*, ***kwargs*)

Bases: BaseEstimator, TransformerMixin, *SkippaMixin*

Data strings into pandas datetime

fit(*X*, *y=None*, ***kwargs*)

Nothing to do here

transform(*X*, *y=None*, ***kwargs*)

Apply the transformation

class skippa.transformers.custom.**SkippaOutlierRemover**(*cols*, *factor=1.5*)

Bases: BaseEstimator, TransformerMixin, *SkippaMixin*

Detect and remove outliers, based on simple IQR

fit(*X*, *y=None*)

transform(*X*, *y=None*)

class skippa.transformers.custom.**SkippaRenamer**(*mapping*)

Bases: BaseEstimator, TransformerMixin

Transformer for renaming columns

fit(*X*, *y=None*, ***kwargs*)

Look at the df to determine the mapping.

In case of a columnselector + function: evaluate the column names and apply the renaming function

transform(*X*, *y=None*, ***kwargs*)

Apply the actual renaming using pandas.rename

class skippa.transformers.custom.**SkippaReplacer**(***kwargs*)

Bases: BaseEstimator, TransformerMixin, *SkippaMixin*

fit(*X*, *y=None*, ***kwargs*)

transform(*X*, *y=None*, ***kwargs*)

class skippa.transformers.custom.**SkippaSelector**(*cols*)

Bases: BaseEstimator, TransformerMixin, *SkippaMixin*

Transformer for selecting a subset of columns in a df.

fit(*X*, *y=None*, ***kwargs*)

transform(*X*, *y=None*, ***kwargs*)

skippa.transformers.sklearn module

This implements transformers based on existing sklearn transformers

class skippa.transformers.sklearn.**SkippaColumnTransformer**(*transformers*, ***, *remainder='drop'*, *sparse_threshold=0.3*, *n_jobs=None*, *transformer_weights=None*, *verbose=False*, *verbose_feature_names_out=True*)

Bases: ColumnTransformer, *SkippaMixin*

Custom ColumnTransformer. Probably not needed anymore.

fit(*X*, *y=None*, ***kwargs*)

Fit all transformers using X.

Parameters

- **X** (*{array-like, dataframe} of shape (n_samples, n_features)*) – Input data, of which specified subsets are used to fit the transformers.
- **y** (*array-like of shape (n_samples,...), default=None*) – Targets for supervised learning.

Returns
self – This estimator.

Return type
ColumnTransformer

fit_transform(*X*, *y=None*)

Fit all transformers, transform the data and concatenate results.

Parameters

- **X** (*{array-like, dataframe} of shape (n_samples, n_features)*) – Input data, of which specified subsets are used to fit the transformers.
- **y** (*array-like of shape (n_samples,), default=None*) – Targets for supervised learning.

Returns
X_t – Horizontally stacked results of transformers. sum_n_components is the sum of n_components (output dimension) over transformers. If any result is a sparse matrix, everything will be converted to sparse matrices.

Return type
{array-like, sparse matrix} of shape (n_samples, sum_n_components)

steps: List[Any]

transform(*X*, *y=None*)

Transform X separately by each transformer, concatenate results.

Parameters
X (*{array-like, dataframe} of shape (n_samples, n_features)*) – The data to be transformed by subset.

Returns
X_t – Horizontally stacked results of transformers. sum_n_components is the sum of n_components (output dimension) over transformers. If any result is a sparse matrix, everything will be converted to sparse matrices.

Return type
{array-like, sparse matrix} of shape (n_samples, sum_n_components)

class skippa.transformers.sklearn.**SkippaLabelEncoder**(*cols*, ***kwargs*)

Bases: *SkippaMixin*, LabelEncoder

Wrapper round sklearn's LabelEncoder

fit(*X*, *y=None*, ***kwargs*)

Fit label encoder.

Parameters
y (*array-like of shape (n_samples,)*) – Target values.

Returns
self – Fitted label encoder.

Return type
returns an instance of self.

`transform`(*X*, *y=None*, ***kwargs*)

Transform labels to normalized encoding.

Parameters
y (*array-like of shape (n_samples,)*) – Target values.

Returns
y – Labels as normalized encodings.

Return type
array-like of shape (n_samples,)

`class skippa.transformers.sklearn.SkippaMinMaxScaler`(*cols*, ***kwargs*)

Bases: *SkippaMixin*, `MinMaxScaler`

Wrapper round sklearn's MinMaxScaler

`fit`(*X*, *y=None*, ***kwargs*)

Compute the minimum and maximum to be used for later scaling.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – The data used to compute the per-feature minimum and maximum used for later scaling along the features axis.
- **y** (*None*) – Ignored.

Returns
self – Fitted scaler.

Return type
object

`transform`(*X*, *y=None*, ***kwargs*)

Scale features of X according to feature_range.

Parameters
X (*array-like of shape (n_samples, n_features)*) – Input data that will be transformed.

Returns
Xt – Transformed data.

Return type
ndarray of shape (n_samples, n_features)

`class skippa.transformers.sklearn.SkippaOneHotEncoder`(*cols*, ***kwargs*)

Bases: *SkippaMixin*, `OneHotEncoder`

Wrapper round sklearn's OneHotEncoder

`fit`(*X*, *y=None*, ***kwargs*)

Fit OneHotEncoder to X.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – The data to determine the categories of each feature.
- **y** (*None*) – Ignored. This parameter exists only for compatibility with `Pipeline`.

Returns
Fitted encoder.

Return type
self

`transform`(*X*, *y=None*, ***kwargs*)

Transform X using one-hot encoding.

If there are infrequent categories for a feature, the infrequent categories will be grouped into a single category.

Parameters
X (*array-like of shape (n_samples, n_features)*) – The data to encode.

Returns
X_out – Transformed input. If *sparse=True*, a sparse matrix will be returned.

Return type
{ndarray, sparse matrix} of shape (n_samples, n_encoded_features)

`class skippa.transformers.sklearn.SkippaOrdinalEncoder`(*cols*, ***kwargs*)

Bases: *SkippaMixin*, `OrdinalEncoder`

Wrapper round sklearn's OrdinalEncoder

`fit`(*X*, *y=None*, ***kwargs*)

Fit the OrdinalEncoder to X.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – The data to determine the categories of each feature.
- **y** (*None*) – Ignored. This parameter exists only for compatibility with `Pipeline`.

Returns
self – Fitted encoder.

Return type
object

`transform`(*X*, *y=None*, ***kwargs*)

Transform X to ordinal codes.

Parameters
X (*array-like of shape (n_samples, n_features)*) – The data to encode.

Returns
X_out – Transformed input.

Return type
ndarray of shape (n_samples, n_features)

`class skippa.transformers.sklearn.SkippaPCA`(*cols*, ***kwargs*)

Bases: *SkippaMixin*, `PCA`

Wrapper round sklearn's PCA

`fit`(*X*, *y=None*, ***kwargs*)

Fit the model with X.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Training data, where *n_samples* is the number of samples and *n_features* is the number of features.
- **y** (*Ignored*) – Ignored.

Returns
self – Returns the instance itself.

Return type
object

`fit_transform`(*X*, *y=None*, ***kwargs*)

The PCA parent class has a custom .fit_transform method for some reason.

`transform`(*X*, *y=None*, ***kwargs*)

Apply dimensionality reduction to X.

X is projected on the first principal components previously extracted from a training set.

Parameters
X (*array-like of shape (n_samples, n_features)*) – New data, where *n_samples* is the number of samples and *n_features* is the number of features.

Returns
X_new – Projection of X in the first principal components, where *n_samples* is the number of samples and *n_components* is the number of the components.

Return type
array-like of shape (n_samples, n_components)

`class` skippa.transformers.sklearn.`SkippaSimpleImputer`(*cols*, ***kwargs*)

Bases: *SkippaMixin*, SimpleImputer

Wrapper round sklearn's SimpleImputer

`fit`(*X*, *y=None*, ***kwargs*)

Fit the imputer on *X*.

Parameters

- **X** (*{array-like, sparse matrix}, shape (n_samples, n_features)*) – Input data, where *n_samples* is the number of samples and *n_features* is the number of features.
- **y** (*Ignored*) – Not used, present here for API consistency by convention.

Returns
self – Fitted estimator.

Return type
object

`transform`(*X*, *y=None*, ***kwargs*)

Impute all missing values in *X*.

Parameters
X (*{array-like, sparse matrix}, shape (n_samples, n_features)*) – The input data to complete.

Returns
X_imputed – *X* with imputed values.

Return type
{ndarray, sparse matrix} of shape (n_samples, n_features_out)

class skippa.transformers.sklearn.**SkippaStandardScaler**(*cols*, ***kwargs*)

Bases: *SkippaMixin*, StandardScaler

Wrapper round sklearn's StandardScaler

fit(*X*, *y=None*, ***kwargs*)

Compute the mean and std to be used for later scaling.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – The data used to compute the mean and standard deviation used for later scaling along the features axis.
- **y** (*None*) – Ignored.
- **sample_weight** (*array-like of shape (n_samples,), default=None*) – Individual weights for each sample.

  New in version 0.24: parameter *sample_weight* support to StandardScaler.

Returns
self – Fitted scaler.

Return type
object

transform(*X*, *y=None*, ***kwargs*)

Perform standardization by centering and scaling.

Parameters

- **X** (*{array-like, sparse matrix of shape (n_samples, n_features)*) – The data used to scale along the features axis.
- **copy** (*bool, default=None*) – Copy the input X or not.

Returns
X_tr – Transformed array.

Return type
{ndarray, sparse matrix} of shape (n_samples, n_features)

skippa.transformers.sklearn.**make_skippa_column_transformer**(**transformers*, *remainder='drop'*, ***kwargs*)

Custom wrapper around sklearn's make_column_transformer

Return type
SkippaColumnTransformer

Module contents

2.2 Submodules

2.3 skippa.app module

2.4 skippa.pipeline module

Defining a Skippa pipeline

```
>>> import pandas as pd
>>> from skippa import Skippa, columns
>>> from sklearn.linear_model import LogisticRegression
```

```
>>> X = pd.DataFrame({
>>>     'q': [2, 3, 4],
>>>     'x': ['a', 'b', 'c'],
>>>     'y': [1, 16, 1000],
>>>     'z': [0.4, None, 8.7]
>>> })
>>> y = np.array([0, 0, 1])
```

```
>>> pipe = (
>>>     Skippa()
>>>         .impute(columns(dtype_include='number'), strategy='median')
>>>         .scale(columns(dtype_include='number'), type='standard')
>>>         .onehot(columns(['x']))
>>>         .select(columns(['y', 'z']) + columns(pattern='x_*'))
>>>         .model(LogisticRegression())
>>> )
```

```
>>> pipe.fit(X=X, y=y)
>>> predictions = pipe.predict_proba(X)
```

class skippa.pipeline.**Skippa**(***kwargs*)

Bases: `object`

Skippa pipeline class

A Skippa pipeline can be extended by piping transformation commands. Only a number of implemented transformations is supported. Although these transformations use existing scikit-learn transformations, each one reqwuires a specific wrapper that implements the pandas dataframe support

append(*pipe*)

Just an alias for adding

Return type
Skippa

apply(**args*, ***kwargs*)

Apply a function to the dataframe.

This is a wrapper around pandas' .apply method and uses the same syntax.

Parameters

- ***args** – first arg should be the funciton to apply
- ****kwargs** – e.g. axis to apply function on

Returns
just return itself again (so we can use piping)

Return type
Skippa

as_type(**args*, ***kwargs*)

Alias for .cast

Return type
Skippa

assign(***kwargs*)

Create new columns based on data in existing columns

This is a wrapper around pandas' .assign method and uses the same syntax.

Parameters
****kwargs** – keyword args denoting new_column=assignment_function pairs

Returns
just return itself again (so we can use piping)

Return type
Skippa

astype(**args*, ***kwargs*)

Alias for .cast

Return type
Skippa

build(***kwargs*)

Build into a scikit-learn Pipeline

Returns
An sklearn Pipeline that supports .fit, .transform

Return type
Pipeline

cast(*cols*, *dtype*)

Cast column to another data type.

Parameters

- **cols** (ColumnSelector) – [description]
- ****kwargs** – arguments for the actual transformer

Returns
just return itself again (so we can use piping)

Return type
Skippa

concat(*pipe*)

Concatenate output of this pipeline to another.

Where adding/appending extends the pipeline, concat keeps parallel pipelines and concatenates their outcomes.

Parameters
pipe (Skippa) – [description]

Returns
[description]

Return type
Skippa

encode_date(*cols*, ***kwargs*)

A date cannot be used unless you encode it into features.

This encoder creates new features out of the year, month, day etc.

Parameters

- **cols** (*[type]*) – [description]
- ****kwargs** – optional keywords like <datepart>=True/False, indicating whether to use dt.<datepart> as a new feature

Returns
[description]

Return type
Skippa

fillna(*cols*, *value*)

Alias/shortcut for impute with constant value (after pandas' .fillna).

This implementation doesn't use pandas.DataFrame.fillna(), but sklearn's SimpleImputer

Parameters
cols (ColumnSelector) – _description_

Returns
just return itself again (so we can use piping)

Return type
Skippa

impute(*cols*, ***kwargs*)

Skippa wrapper around sklearn's SimpleImputer

Parameters
cols (ColumnSelector) – [description]

Returns
just return itself again (so we can use piping)

Return type
Skippa

label_encode(*cols*, ***kwargs*)

Wrapper around sklearn's LabelEncoder

Parameters

- **cols** (ColumnSelector) – columns specification
- ****kwargs** – optional kwargs for LabelEncoder

Returns
just return itself again (so we can use piping)

Return type
Skippa

`static load`(*path*)

Load a previously saved skippa

N.B. dill is used for (de)serialization, because joblib/pickle doesn't support things like lambda functions.

Parameters
path (*PathLike*) – pathamae, either string or pathlib.Path

Returns
an sklearn Pipeline

Return type
Pipeline

`static load_pipeline`(*path*)

Load a previously saved pipeline

N.B. dill is used for (de)serialization, because joblib/pickle doesn't support things like lambda functions.

Parameters
path (*PathLike*) – pathname, either string or pathlib.Path

Returns
an extended sklearn Pipeline

Return type
SkippaPipeline

`model`(*model*)

Add a model estimator.

A model estimator is always the last step in the pipeline! Therefore this doesn't return the Skippa object (self) but calls the .build method to return the pipeline.

Parameters
model (*BaseEstimator*) – An sklearn estimator

Returns
a built pipeline

Return type
SkippaPipeline

`onehot`(*cols*, ***kwargs*)

Skippa wrapper around sklearn's OneHotEncoder

Parameters

- **cols** (ColumnSelector) – columns specification
- ****kwargs** – optional kwargs for OneHotEncoder (although 'sparse' will always be set to False)

Returns
just return itself again (so we can use piping)

Return type
Skippa

ordinal_encode(*cols*, ***kwargs*)

Wrapper around sklearn's OrdinalEncoder

Parameters

- **cols** (ColumnSelector) – columns specification
- ****kwargs** – optional kwargs for OrdinalEncoder

Returns
just return itself again (so we can use piping)

Return type
Skippa

pca(*cols*, ***kwargs*)

Wrapper around sklearn.decomposition.PCA

Parameters

- **cols** (ColumnSelector) – columns expression
- **kwargs** – any kwargs to be used by PCA's __init__

Returns
just return itself again (so we can use piping)

Return type
Skippa

rename(**args*, ***kwargs*)

Rename certain columns.

Two ways to use this: - a dict which defines a mapping {existing_col: new_col} - a column selector and a renaming function (e.g. ['a', 'b', 'c'], lambda c: f'new_{c}') It adds an XRenamer step, which wraps around pandas.rename

Returns
just return itself again (so we can use piping)

Return type
Skippa

save(*file_path*)

Save to disk using dill

Return type
None

scale(*cols*, *type='standard'*, ***kwargs*)

Skippa wrapper around sklearn's StandardScaler / MinMaxScaler

Parameters

- **cols** (ColumnSelector) – [description]
- **type** (*str, optional*) – One of ['standard', 'minmax']. Defaults to 'standard'.

Raises
ValueError – if an unknown/unsupported scaler type is passed

Returns
just return itself again (so we can use piping)

Return type
Skippa

`select`(*cols*)

Apply a column selection

Parameters
cols (ColumnSelector) – [description]

Returns
just return itself again (so we can use piping)

Return type
Skippa

`class skippa.pipeline.SkippaPipeline`(*steps*, *, *memory=None*, *verbose=False*)

Bases: `Pipeline`

Extension of sklearn's Pipeline object.

While the Skippa class is for creating pipelines, it is not a pipeline itself. Only after adding a model estimator step, or by calling *.build* explicitly, is a SkippaPipeline created. This is basically an sklearn Pipeline with some added methods.

`create_gradio_app`(***kwargs*)

Create a Gradio app for model inspection.

Parameters
****kwargs** – kwargs received by Gradio's *Interface()* initialisation

Returns
Gradio Interface object -> call .launch to start the app

Return type
gr.Interface

`fit`(*X*, *y=None*, ***kwargs*)

Inspect input data before fitting the pipeline.

Return type
SkippaPipeline

`get_data_profile`()

The DataProfile is used in the Gradio app.

The profile contains information on column names, their dtypes and value ranges.

Raises
NotFittedError – If pipeline has not been fitted there is no data profile yet.

Returns
Simple object containing necessary info

Return type
DataProfile

`get_model`()

Get the model estimator part of the pipeline.

So that you can access info like coefficients e.d.

Returns
fitted model

Return type
BaseEstimator

`get_pipeline_params`(*params*)

Translate model param grid to Pipeline param grid.

For GridSearch over a Pipeline, you need to sdupply a param grid in the form of { <step-name>__<paramname>: values } Since it's non-trivial to find the name of the model/estimator step in the Pipeline, this auto detects it and return a new param grid in the right format.

Parameters
params (*Dict*) – param grid with parameter names containing only the model parameter

Returns
param grid with parameter names relating to both the pipeline step and the model parameter

Return type
Dict

`save`(*file_path*)

Return type
None

`steps: List[Any]`

`test`(*X*, *up_to_step=-1*)

Test what happens to data in a pipeline.

This allows you to execute the pipeline up & until the last step before modeling (or any other step) and get the resulting data.

Parameters

- **X** (*_type_*) – _description_
- **up_to_step** (*int, optional*) – _description_. Defaults to -1.

Returns
description

Return type
pd.DataFrame

2.5 skippa.profile module

DataProfile is used for storing and retrieving metadata of data that is used in the pipeline. Typically the DataProfile is created during fitting of a pipeline. The profile is used by the Gradio app that can be created.

`class skippa.profile.DataProfile`(*df*, *y=None*)

Bases: `object`

`MAX_NUM_DISTINCT_VALUES = 100000`

`is_classification()`

Return type
`bool`

`is_regression()`

Return type
`bool`

2.6 skippa.utils module

`skippa.utils.get_dummy_data(nrows=100, nfloat=4, nint=2, nchar=3, ndate=1, missing=True, binary_y=True)`

Create dummy data.

Parameters

- **nrows** (*int, optional*) – Number of total rows. Defaults to 100.
- **nfloat** (*int, optional*) – Number of float columns. Defaults to 4.
- **nint** (*int, optional*) – Number of integer columns. Defaults to 2.
- **nchar** (*int, optional*) – Number of character/categorical columns. Defaults to 3.
- **ndate** (*int, optional*) – Number of date columns. Defaults to 1.
- **binary_y** (*bool, optional*) – If True, returns 0's & 1's for y, otherwise float values between 0 & 100

Returns
A pandas DataFrame for features and a numpy array for labels

Return type
Tuple[pd.DataFrame, np.ndarray]

2.7 Module contents

Top-level package for skippa.

The pipeline module defines the main Skippa methods The transformers subpackage contains various transformers used in the pipeline.

CHAPTER

THREE

INTRODUCTION

SciKIt-learn Pipeline in PAndas

Want to create a machine learning model using pandas & scikit-learn? This should make your life easier.

Skippa helps you to easily create a pre-processing and modeling pipeline, based on scikit-learn transformers but preserving pandas dataframe format throughout all pre-processing. This makes it a lot easier to define a series of subsequent transformation steps, while referring to columns in your intermediate dataframe.

3.1 Installation

```
$ pip install skippa
```

3.2 Basic use

Skippa helps you to easily define data cleaning & pre-processing operations on a pandas DataFrame and combine it with a scikit-learn model/algorithm into a single executable pipeline. It works roughly like this:

```
from skippa import Skippa, columns
from sklearn.linear_model import LogisticRegression
pipeline = (
   Skippa()
   .impute(columns(dtype_include='object'), strategy='most_frequent')
   .impute(columns(dtype_include='number'), strategy='median')
   .scale(columns(dtype_include='number'), type='standard')
   .onehot(columns(['category1', 'category2']))
   .model(LogisticRegression())
)
pipeline.fit(X, y)
predictions = pipeline.predict_proba(new_data)
```

CHAPTER

FOUR

MODULES

Top-level package for skippa.

The pipeline module defines the main Skippa methods The transformers subpackage contains various transformers used in the pipeline.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX