Data Encoding and Preprocessing

Discrete Transformations

Ordinal Encoder

In [ ]:

```
from sklearn.preprocessing import OrdinalEncoder
enc = OrdinalEncoder()
X = [['Male', 1], ['Female', 3], ['Female', 2]]
enc.fit(X)


enc.categories_

enc.transform([['Female', 3], ['Male', 1]])
```

In [ ]:

```
enc.inverse_transform([[1, 0], [0, 1]])
```

One-Hot Encoder

In [ ]:

```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(handle_unknown='ignore')
X = [['Male', 1], ['Female', 3], ['Female', 2]]
enc.fit(X)

enc.categories_

enc.transform([['Female', 1], ['Male', 4]]).toarray()


enc.inverse_transform([[0, 1, 1, 0, 0], [0, 0, 0, 1, 0]])


enc.get_feature_names()

drop_enc = OneHotEncoder(drop='first').fit(X)
drop_enc.categories_

drop_enc.transform([['Female', 1], ['Male', 2]]).toarray()
```

Label Encoder

In [ ]:

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit([1, 2, 2, 6])

le.classes_

le.transform([1, 1, 2, 6])

le.inverse_transform([0, 0, 1, 2])
```

In [ ]:

```
le = preprocessing.LabelEncoder()
le.fit(["paris", "paris", "tokyo", "amsterdam"])

list(le.classes_)

le.transform(["tokyo", "tokyo", "paris"])

list(le.inverse_transform([2, 2, 1]))
```

Binarizer

In [ ]:

```
from sklearn.preprocessing import Binarizer
X = [[ 1., -1.,  2.],
     [ 2.,  0.,  0.],
     [ 0.,  1., -1.]]
transformer = Binarizer().fit(X)  # fit does nothing.
transformer

transformer.transform(X)
```

Continuous Transformations

Normalizer

In [ ]:

```
from sklearn.preprocessing import Normalizer
X = [[4, 1, 2, 2],
     [1, 3, 9, 3],
     [5, 7, 5, 1]]
transformer = Normalizer().fit(X)  # fit does nothing.
transformer

transformer.transform(X)
```

Scalar

In [ ]:

```
from sklearn.preprocessing import StandardScaler
data = [[0, 0], [0, 0], [1, 1], [1, 1]]
scaler = StandardScaler()
print(scaler.fit(data))

print(scaler.mean_)

print(scaler.transform(data))




print(scaler.transform([[2, 2]]))
```

Preparation Comparison

In [2]:

```
from sklearn.preprocessing import *

transforms = [
    FunctionTransformer(lambda x: x, validate=False),
    StandardScaler(),
    MinMaxScaler(),
    MaxAbsScaler(),
    RobustScaler(quantile_range=(25, 75)),
    PowerTransformer(method='yeo-johnson'),
    PowerTransformer(method='box-cox'),
    QuantileTransformer(output_distribution='normal'),
    QuantileTransformer(output_distribution='uniform'),
    Normalizer()
 ]
```

In [29]:

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_california_housing

housing =  fetch_california_housing()
```

In [30]:

```
print(housing['DESCR'])
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the
target

    :Attribute Information:
        - MedInc        median income in block
        - HouseAge      median house age in block
        - AveRooms      average number of rooms
        - AveBedrms     average number of bedrooms
        - Population    block population
        - AveOccup      average house occupancy
        - Latitude      house block latitude
        - Longitude     house block longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
http://lib.stat.cmu.edu/datasets/

The target variable is the median house value for California distric
ts.

This dataset was derived from the 1990 U.S. census, using one row pe
r census
block group. A block group is the smallest geographical unit for whi
ch the U.S.
Census Bureau publishes sample data (a block group typically has a p
opulation
of 600 to 3,000 people).

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. topic:: References

    - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressio
ns,
      Statistics and Probability Letters, 33 (1997) 291-297
```

In [28]:

```
X_full, y_full, *_ = housing.values()

X = X_full[:, [0, 5]]

# scale the output between 0 and 1 for the colorbar
y = minmax_scale(y_full)
```

```
---------------------------------------------------------------------
-------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-28-1ce70835a5de> in <module>
      1
----> 2 X_full, y_full, *_ = housing.values()
      3
      4 X = X_full[:, [0, 5]]
      5

NameError: name 'housing' is not defined
```

In [32]:

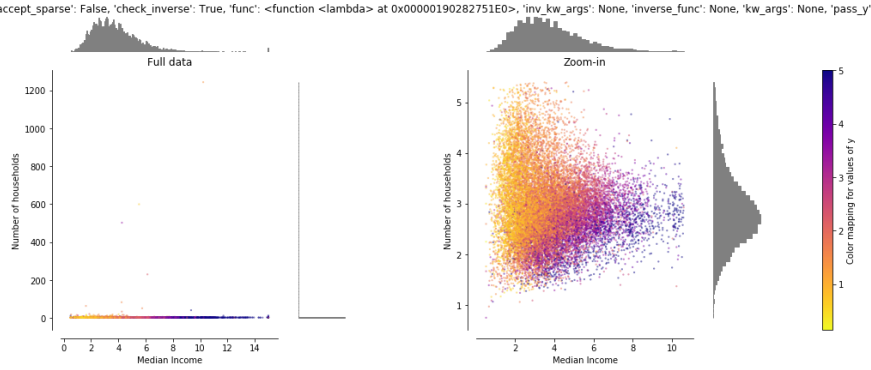
```
def create_axes(title, figsize=(16, 6)):
    fig = plt.figure(figsize=figsize)
    fig.suptitle(title)

    # define the axis for the first plot
    left, width = 0.1, 0.22
    bottom, height = 0.1, 0.7
    bottom_h = height + 0.15
    left_h = left + width + 0.02

    rect_scatter = [left, bottom, width, height]
    rect_histx = [left, bottom_h, width, 0.1]
    rect_histy = [left_h, bottom, 0.05, height]

    ax_scatter = plt.axes(rect_scatter)
    ax_histx = plt.axes(rect_histx)
    ax_histy = plt.axes(rect_histy)

    # define the axis for the zoomed-in plot
    left = width + left + 0.2
    left_h = left + width + 0.02

    rect_scatter = [left, bottom, width, height]
    rect_histx = [left, bottom_h, width, 0.1]
    rect_histy = [left_h, bottom, 0.05, height]

    ax_scatter_zoom = plt.axes(rect_scatter)
    ax_histx_zoom = plt.axes(rect_histx)
    ax_histy_zoom = plt.axes(rect_histy)

    # define the axis for the colorbar
    left, width = width + left + 0.13, 0.01

    rect_colorbar = [left, bottom, width, height]
    ax_colorbar = plt.axes(rect_colorbar)

    return ((ax_scatter, ax_histy, ax_histx),
            (ax_scatter_zoom, ax_histy_zoom, ax_histx_zoom),
            ax_colorbar)


def plot_distribution(axes, X, y, hist_nbins=50, title="", x0_label="", x1_label
=""):
    ax, hist_X1, hist_X0 = axes

    ax.set_title(title)
    ax.set_xlabel(x0_label)
    ax.set_ylabel(x1_label)

    # The scatter plot
    colors = mpl.cm.plasma_r(y)
    ax.scatter(X[:, 0], X[:, 1], alpha=0.5, marker='o', s=5, lw=0, c=colors)

    # Removing the top and the right spine for aesthetics
    # make nice axis layout
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()
```

```
    ax.spines['left'].set_position(('outward', 10))
    ax.spines['bottom'].set_position(('outward', 10))

    # Histogram for axis X1 (feature 5)
    hist_X1.set_ylim(ax.get_ylim())
    hist_X1.hist(X[:, 1], bins=hist_nbins, orientation='horizontal',
                 color='grey', ec='grey')
    hist_X1.axis('off')

    # Histogram for axis X0 (feature 0)
    hist_X0.set_xlim(ax.get_xlim())
    hist_X0.hist(X[:, 0], bins=hist_nbins, orientation='vertical',
                 color='grey', ec='grey')
    hist_X0.axis('off')

def plot_transform(tf, X):
    title = tf.__class__.__name__ + ' ' + str(tf.get_params())[1:-1]
    X = tf.fit_transform(X)

    ax_zoom_out, ax_zoom_in, ax_colorbar = create_axes(title)
    axarr = (ax_zoom_out, ax_zoom_in)
    plot_distribution(axarr[0], X, y, hist_nbins=200,
                      x0_label="Median Income",
                      x1_label="Number of households",
                      title="Post-Transform (Full)")

    # zoom-in
    zoom_in_percentile_range = (0, 99)
    cutoffs_X0 = np.percentile(X[:, 0], zoom_in_percentile_range)
    cutoffs_X1 = np.percentile(X[:, 1], zoom_in_percentile_range)

    non_outliers_mask = (
        np.all(X > [cutoffs_X0[0], cutoffs_X1[0]], axis=1) &
        np.all(X < [cutoffs_X0[1], cutoffs_X1[1]], axis=1))
    plot_distribution(axarr[1], X[non_outliers_mask], y[non_outliers_mask],
                      hist_nbins=50,
                      x0_label="X0: Median Income",
                      x1_label="X5: Number of households",
                      title="Post-Transform (Zoom)")

    norm = mpl.colors.Normalize(y_full.min(), y_full.max())
    mpl.colorbar.ColorbarBase(ax_colorbar, cmap=mpl.cm.plasma_r,
                              norm=norm, orientation='vertical',
                              label='Color mapping for values of MedianHouseValu
e')
```

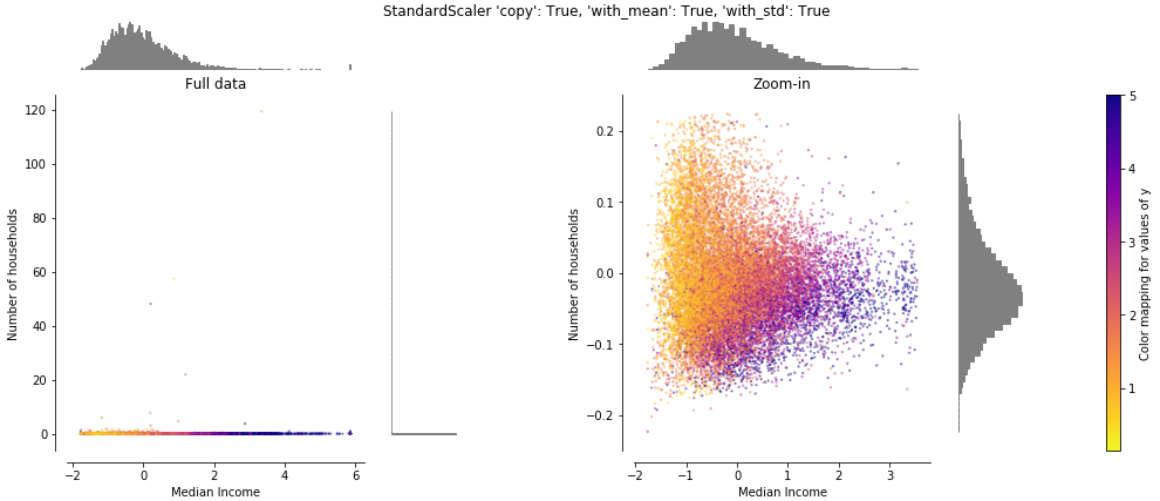
In [20]:

```
for t in transforms:
    plot_transform(t, X)
```

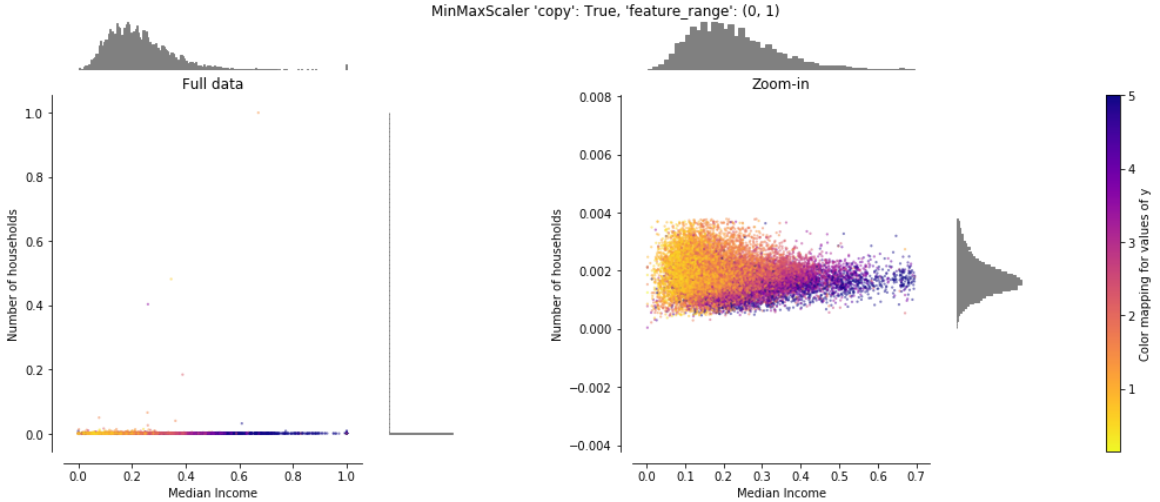
FunctionTransformer 'accept_sparse': False, 'check_inverse': True, 'func': <function <lambda> at 0x000001902827 51E0>, 'inv_kw_args': None, 'inverse_func': None, 'kw_args': None, 'pass_y': 'deprecated', 'validate': False

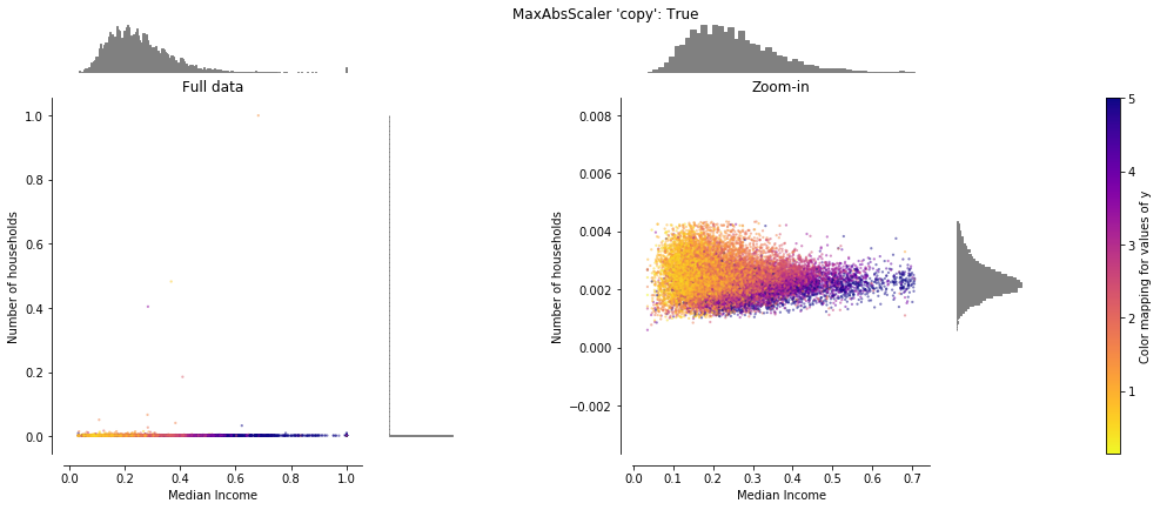


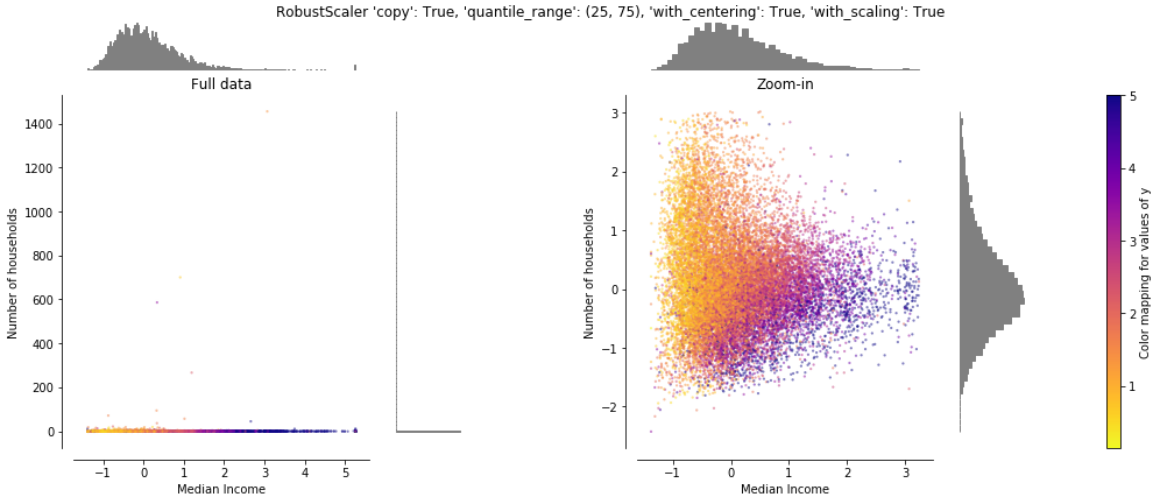
StandardScaler 'copy': True, 'with_mean': True, 'with_std': True

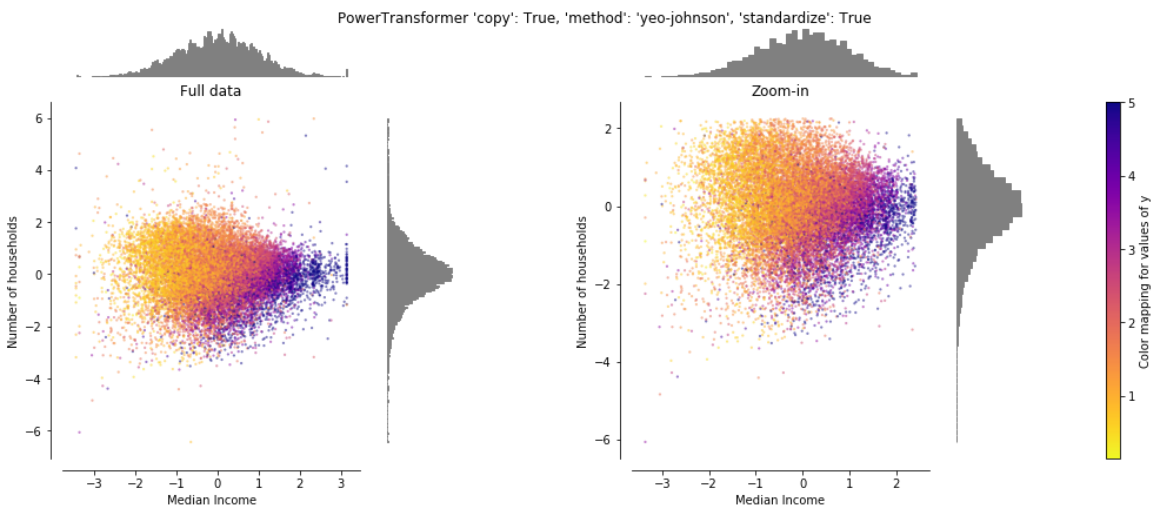


MinMaxScaler 'copy': True, 'feature_range': (0, 1)

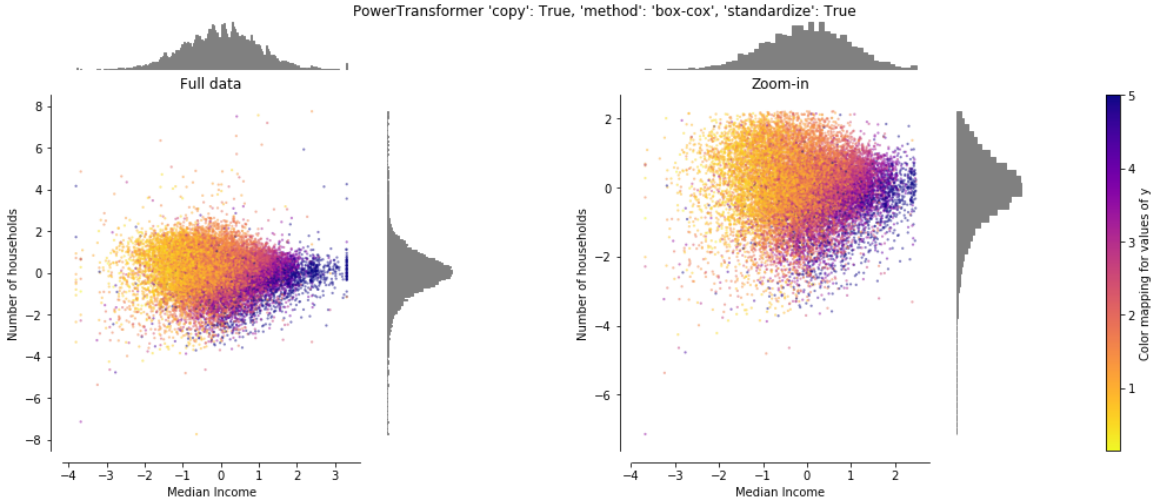



MaxAbsScaler 'copy': True
Full data
Zoom-in
Number of households
Median Income
Color mapping for values of y

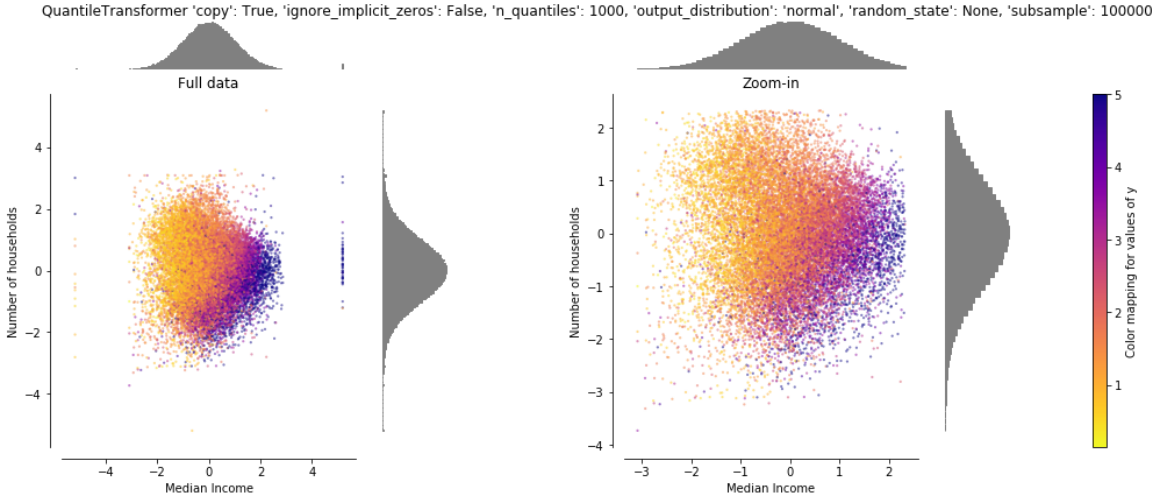

RobustScaler 'copy': True, 'quantile_range': (25, 75), 'with_centering': True, 'with_scaling': True
Full data
Zoom-in
Number of households
Median Income
Color mapping for values of y


PowerTransformer 'copy': True, 'method': 'yeo-johnson', 'standardize': True
Full data
Zoom-in
Number of households
Median Income
Color mapping for values of y

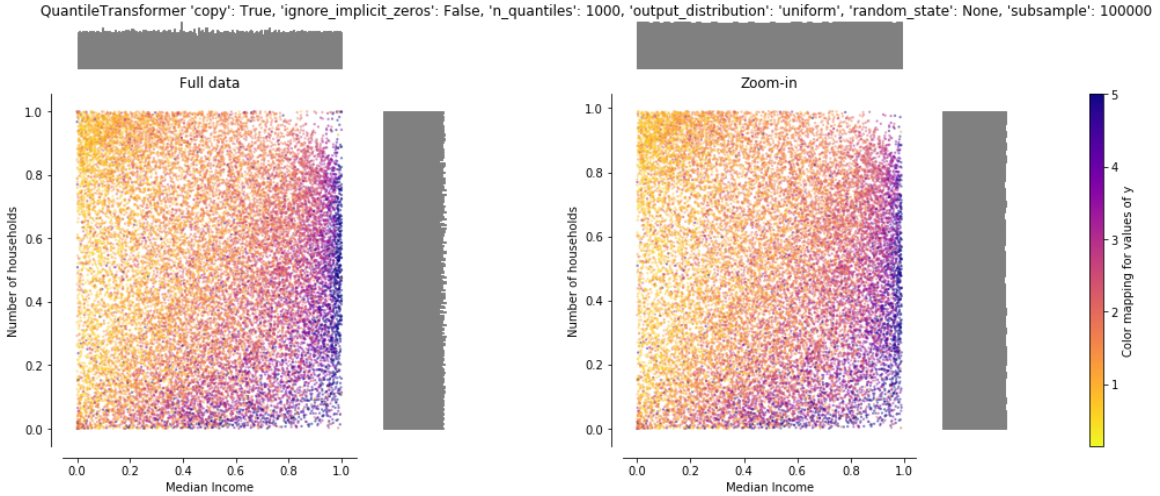
PowerTransformer 'copy': True, 'method': 'box-cox', 'standardize': True

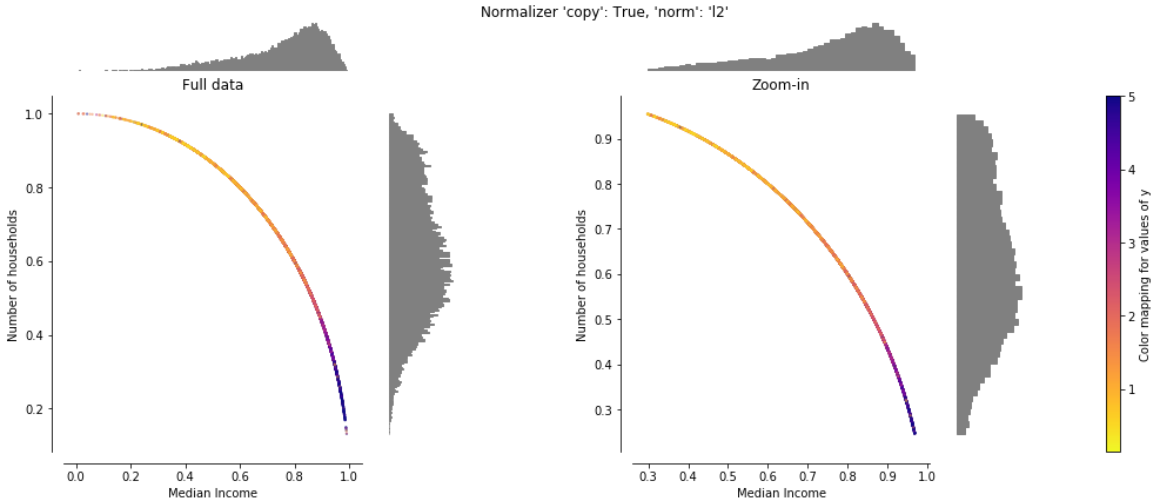


QuantileTransformer 'copy': True, 'ignore_implicit_zeros': False, 'n_quantiles': 1000, 'output_distribution': 'normal', 'random_state': None, 'subsample': 100000



QuantileTransformer 'copy': True, 'ignore_implicit_zeros': False, 'n_quantiles': 1000, 'output_distribution': 'uniform', 'random_state': None, 'subsample': 100000





Licences

- Partly derived from
  - work by

    ```
    Raghav RV <rvraghav93@gmail.com>
    Guillaume Lemaitre <g.lemaitre58@gmail.com>
    Thomas Unterthiner
    Licence: BSD 3 clause
    ```

  - work by

    ```
    Copyright (c) 2007–2019 The scikit-learn developers.
    Licence: New BSD
    ```