APRENDIZAJE NO SUPERVISADO

Transformaciones del dataset

Aprendizaje no supervisado

Problemas con el aprendizaje no supervisado

- Usamos datasets sin etiquetas (porque no contamos con ellas o porque no nos interesan)

- Hay dos tipos
  - Transformaciones
  - Clustering

- Es difícil saber cuando el algoritmo aprende cosas útiles

No tenemos manera de decirle al algoritmo que estamos buscando

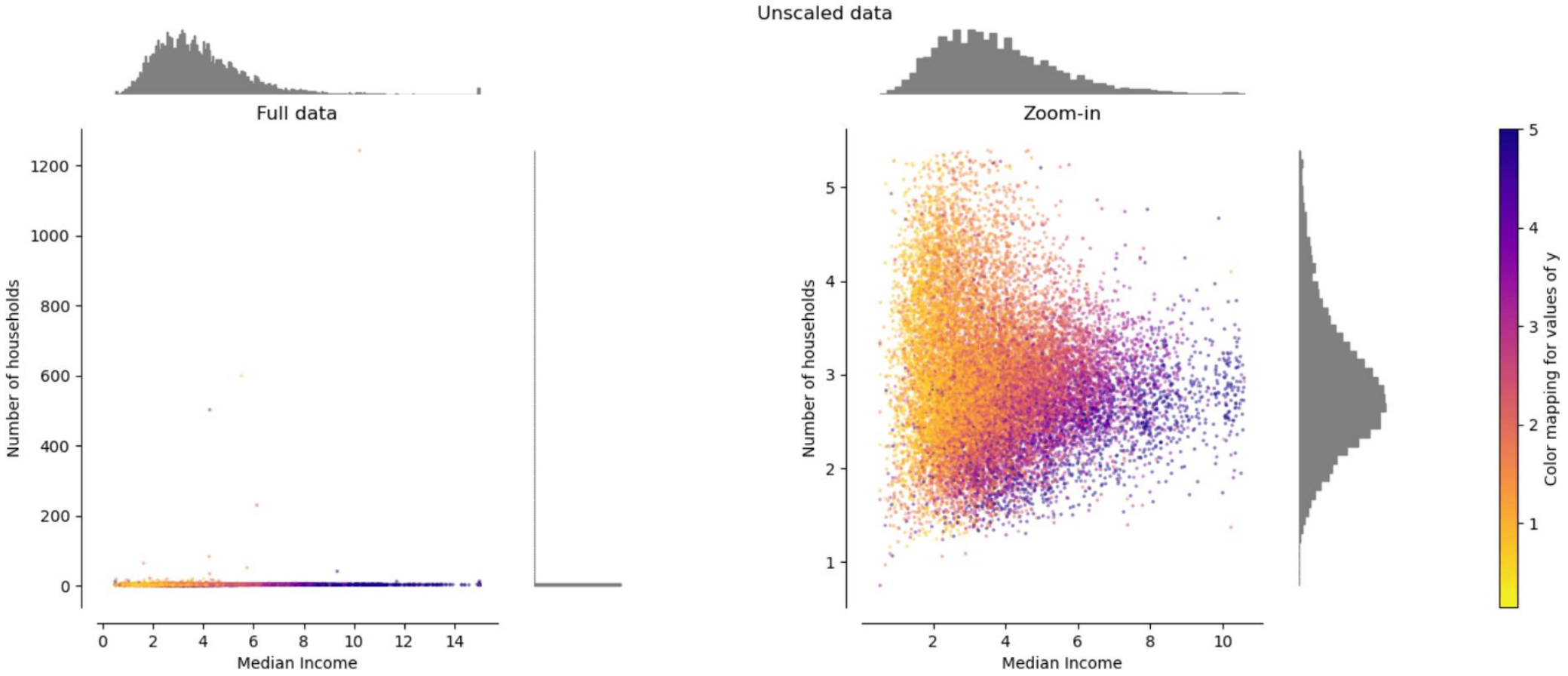
Transformaciones no supervisadas

Crean nuevas representaciones de los datos para hacerlos más fáciles de trabajar, ya sea para las personas o para otros algoritmos.

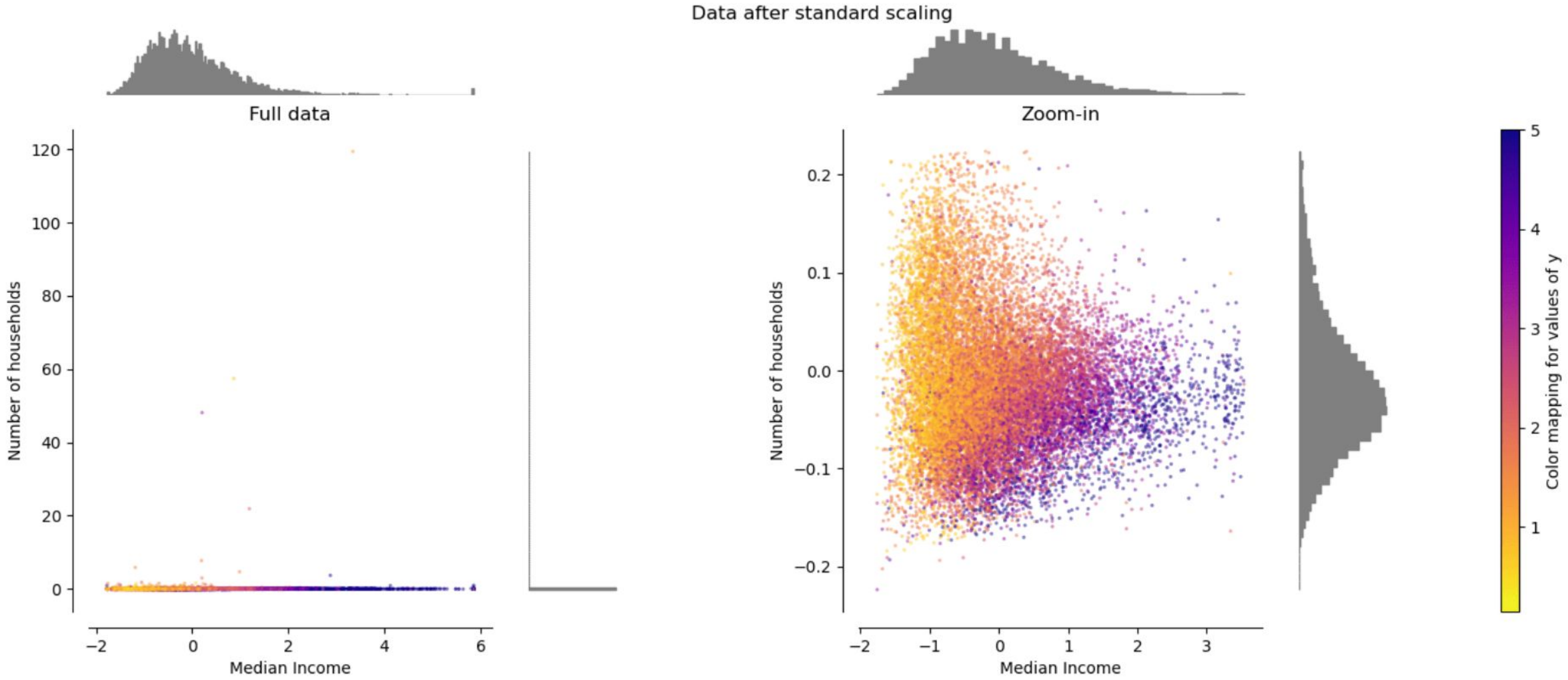
Aplicaciones

- Reducir la dimensión
- Encontrar las partes que "componen" los datos
- Reducir memoria y tiempos

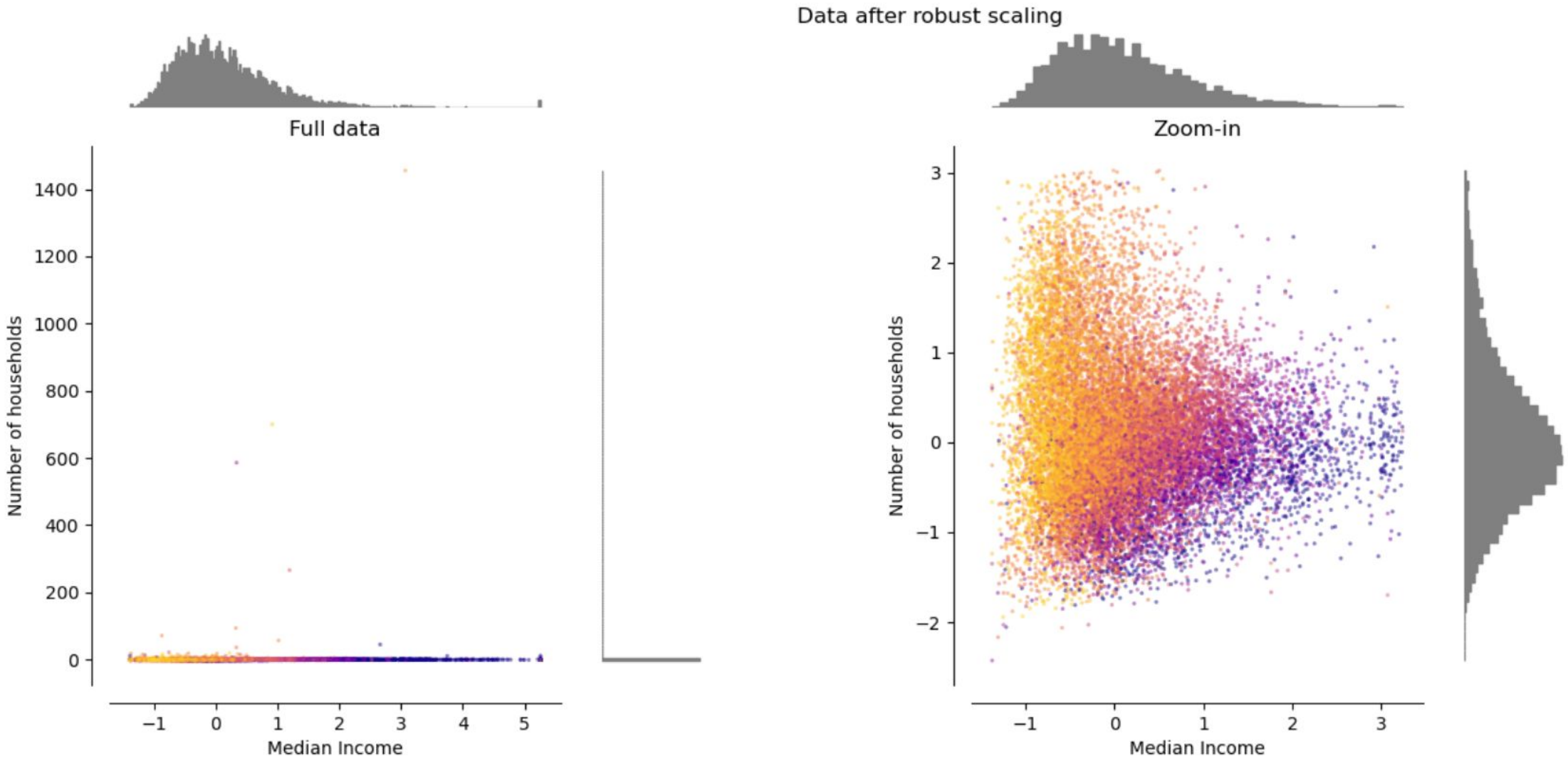
Formas de procesar y escalar los datos



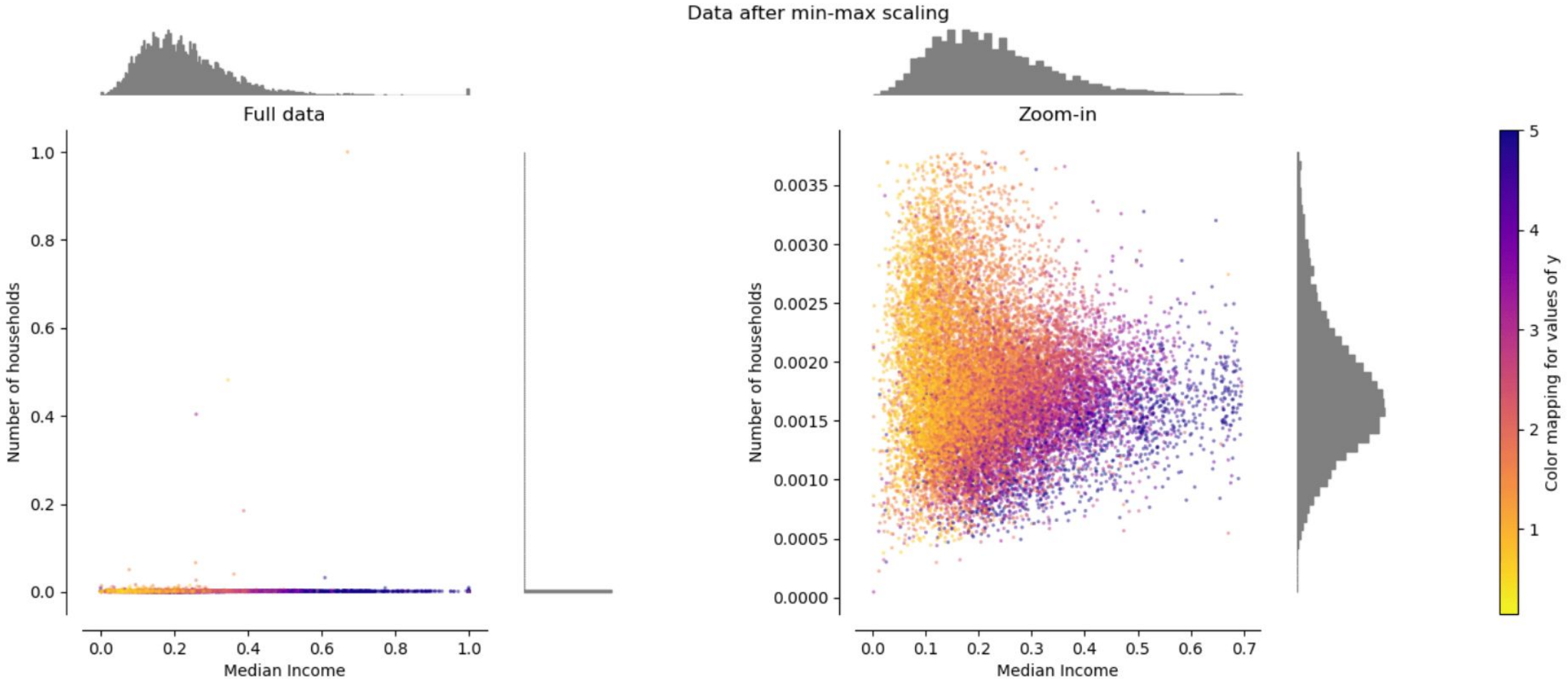
StandardScaling



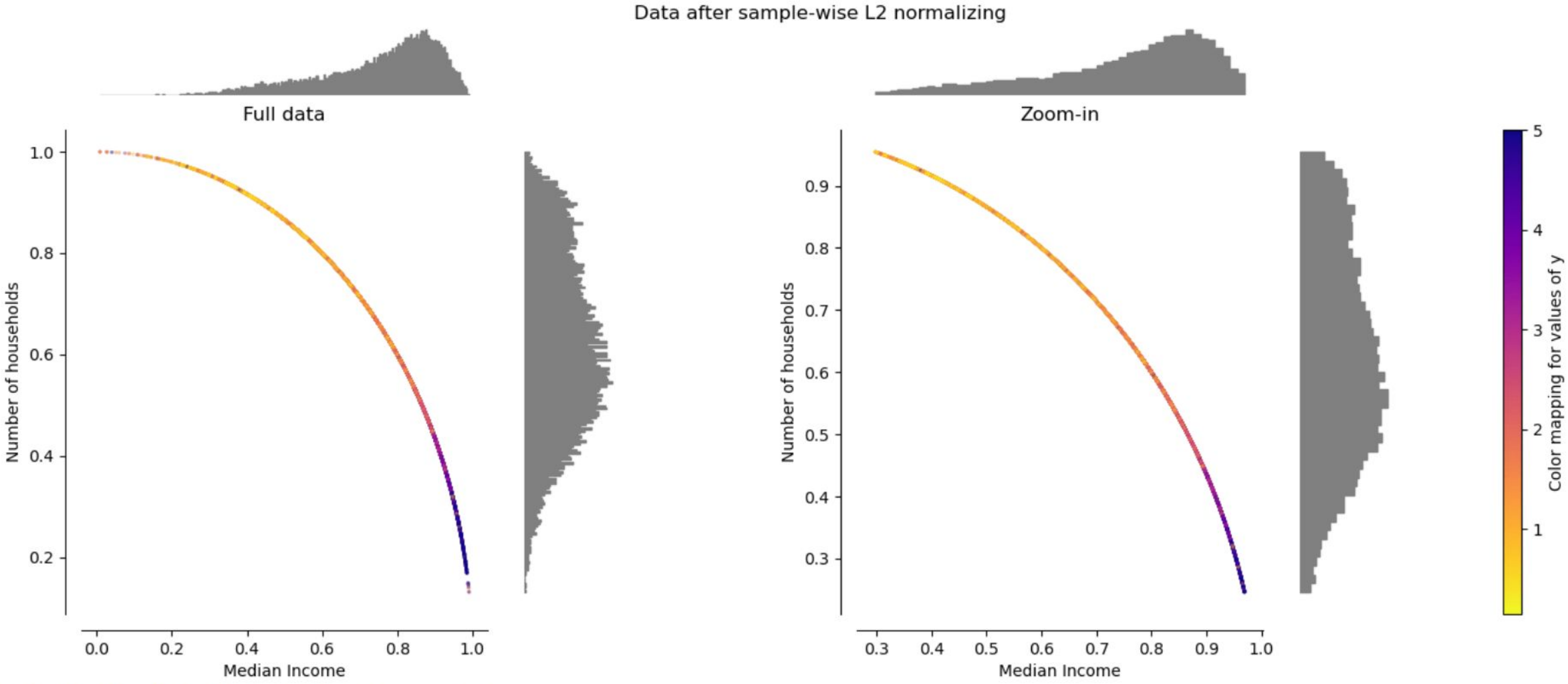
RobustScaling



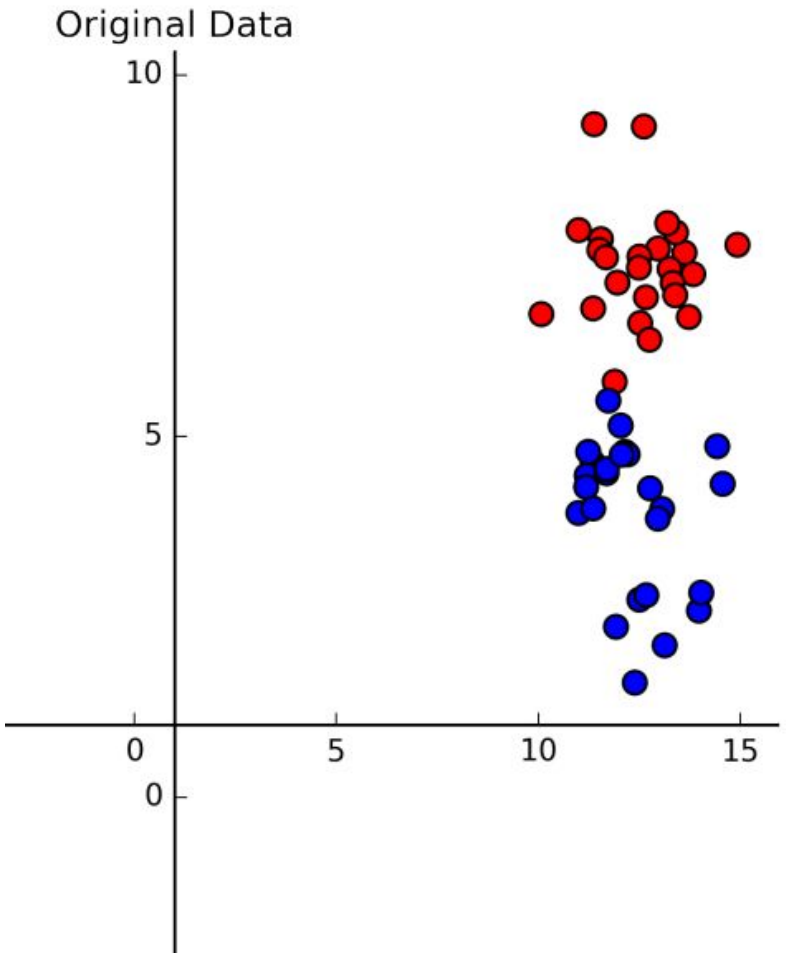
MinMaxScaling

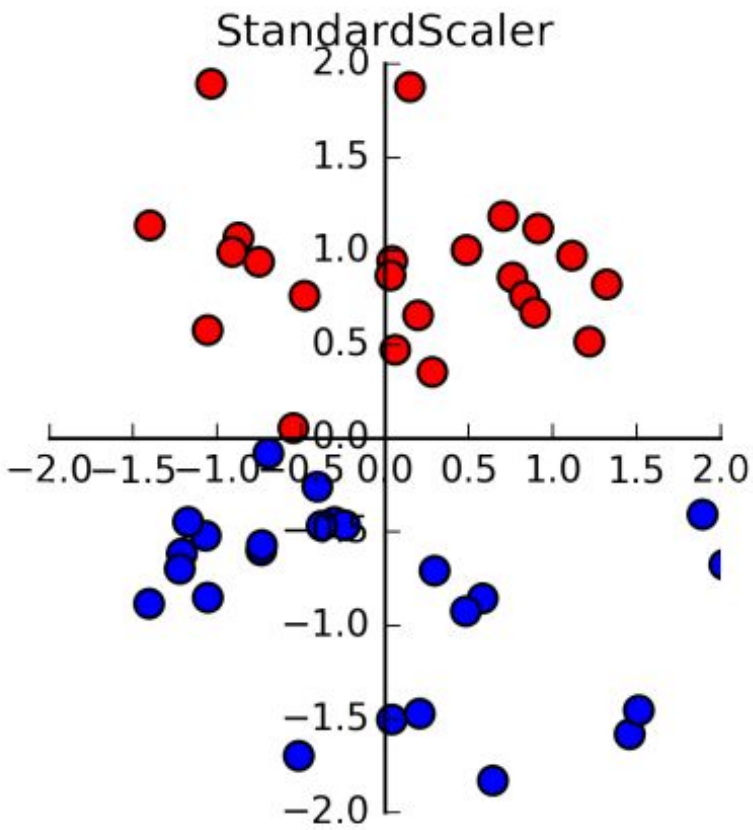


Normilizer

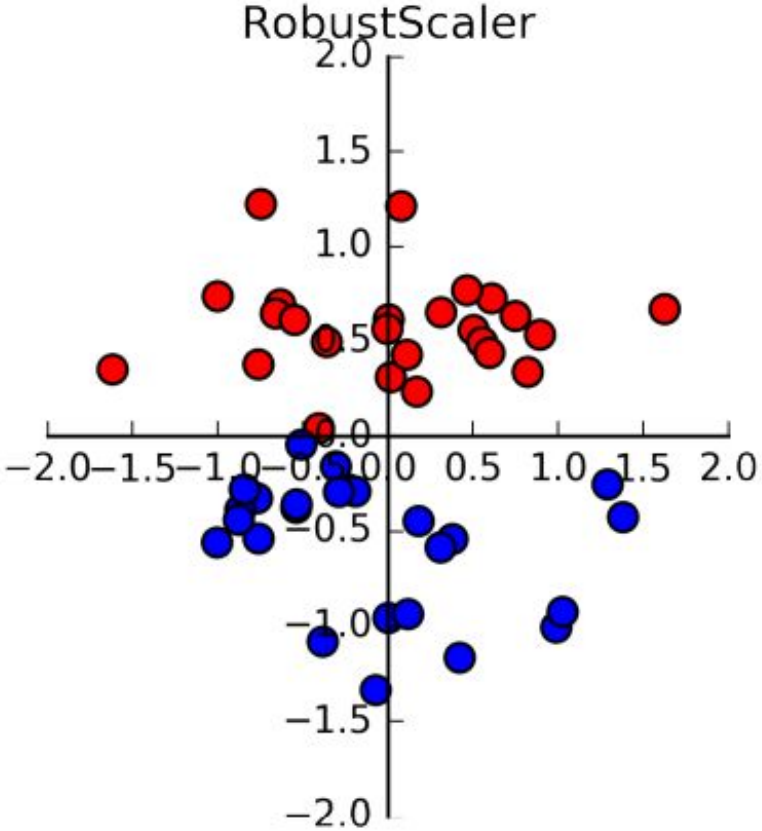

Data after sample-wise L2 normalizing
Full data
Zoom-in
Number of households
Median Income
Color mapping for values of y

Formas de procesar y escalar los datos



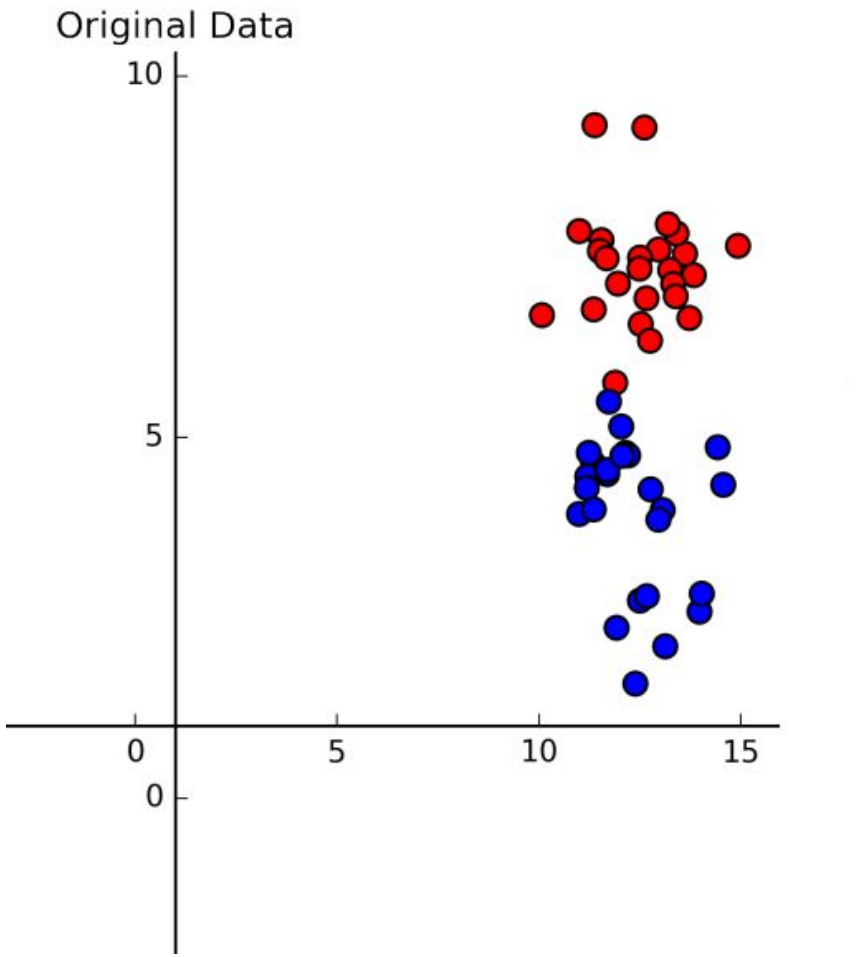


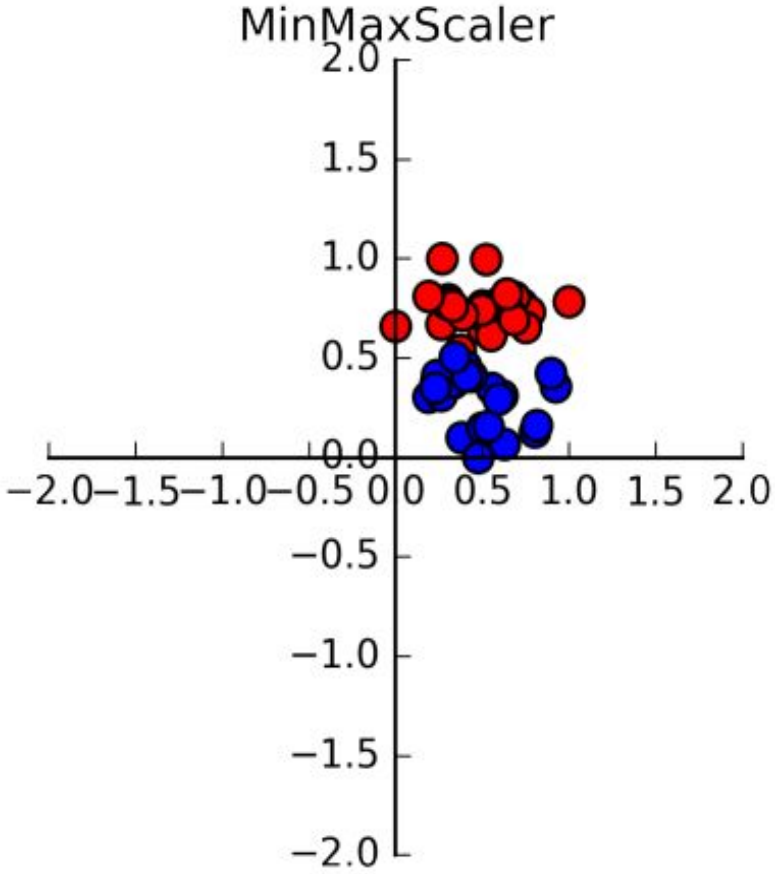
Todo alrededor del cero



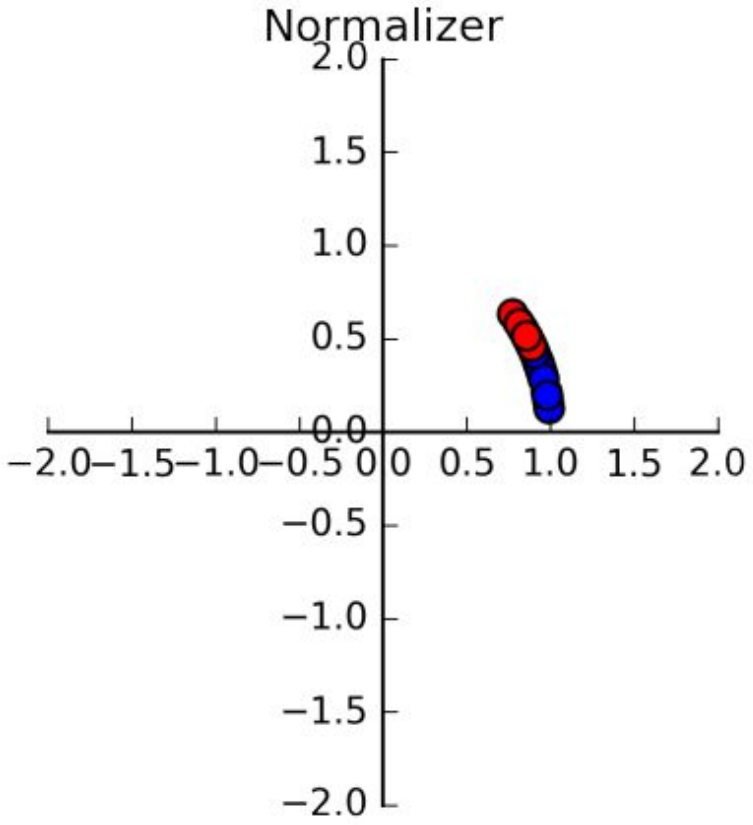
Es bueno si existen puntos atípicos

Formas de procesar y escalar los datos



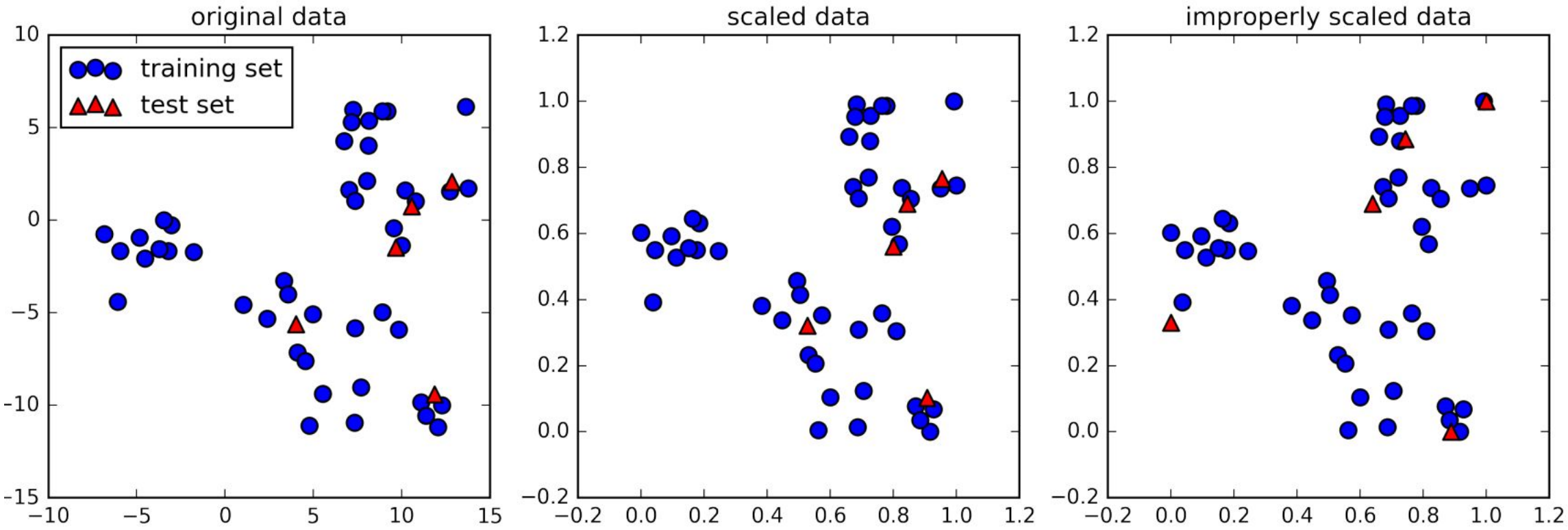


Entre cero y uno



No le importan las longitudes

Hay que transformar el conjunto prueba con cuidado



Ejemplo de MinMaxScaler

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
 random_state=1)
print(X_train.shape)
print(X_test.shape)
```

```
(426, 30)
(143, 30)
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

```
scaler.fit(X_train)
MinMaxScaler(copy=True, feature_range=(0, 1))
```

```
MinMaxScaler()
```

```
# don't print using scientific notation
np.set_printoptions(suppress=True, precision=2)
# transform data
X_train_scaled = scaler.transform(X_train)
# print data set properties before and after scaling
print("transformed shape: %s" % (X_train_scaled.shape,))
print("per-feature minimum before scaling:\n %s" % X_train.min(axis=0))
print("per-feature maximum before scaling:\n %s" % X_train.max(axis=0))
print("per-feature minimum after scaling:\n %s" % X_train_scaled.min(axis=0))
print("per-feature maximum after scaling:\n %s" % X_train_scaled.max(axis=0))
```

```
transformed shape: (426, 30)
per-feature minimum before scaling:
 [  6.98   9.71  43.79 143.5    0.05   0.02   0.     0.     0.11   0.05
   0.12   0.36   0.76   6.8    0.     0.     0.     0.     0.01   0.
   7.93  12.02  50.41 185.2    0.07   0.03   0.     0.     0.16   0.06]
per-feature maximum before scaling:
 [  28.11   39.28  188.5  2501.      0.16    0.29    0.43    0.2     0.3
    0.1     2.87    4.88   21.98  542.2     0.03    0.14    0.4     0.05
    0.06    0.03   36.04   49.54  251.2  4254.      0.22    0.94    1.17
    0.29    0.58    0.15]
per-feature minimum after scaling:
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.]
per-feature maximum after scaling:
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1.]
```

```
# transform test data
X_test_scaled = scaler.transform(X_test)
# print test data properties after scaling
print("per-feature minimum after scaling: %s" % X_test_scaled.min(axis=0))
print("per-feature maximum after scaling: %s" % X_test_scaled.max(axis=0))
```

```
per-feature minimum after scaling: [ 0.03  0.02  0.03  0.01  0.14  0.04  0.    0.    0.15 -0.01 -0.    0.01
  0.    0.    0.04  0.01  0.    0.   -0.03  0.01  0.03  0.06  0.02  0.01
  0.11  0.03  0.    0.   -0.   -0.  ]
per-feature maximum after scaling: [0.96 0.82 0.96 0.89 0.81 1.22 0.88 0.93 0.93 1.04 0.43 0.5  0.44 0.28
 0.49 0.74 0.77 0.63 1.34 0.39 0.9  0.79 0.85 0.74 0.92 1.13 1.07 0.92
 1.21 1.63]
```

Diferencias al escalar los datos

```
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
 random_state=4)
svm = SVC(C=100)
svm.fit(X_train, y_train)
print(svm.score(X_test, y_test))
```

```
0.8881118881118881
```

Con los datos originales

```
# preprocessing using 0-1 scaling
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
# learning an SVM on the scaled training data
svm.fit(X_train_scaled, y_train)
# scoring on the scaled test set
svm.score(X_test_scaled, y_test)
```

```
0.951048951048951
```

Al escalar los datos

Escalar los datos de otra manera

Con MinMaxScaler

```
# preprocessing using 0-1 scaling
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
# learning an SVM on the scaled training data
svm.fit(X_train_scaled, y_train)
# scoring on the scaled test set
svm.score(X_test_scaled, y_test)
```

```
0.951048951048951
```

Con StandardScaler

```
# preprocessing using zero mean and unit variance scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
# learning an SVM on the scaled training data
svm.fit(X_train_scaled, y_train)
# scoring on the scaled test set
svm.score(X_test_scaled, y_test)
```

```
0.958041958041958
```

Escalar con RobustScaler y Normilizer

Con RobustScaler

```
# preprocessing using robustScaler
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
# learning an SVM on the scaled training data
svm.fit(X_train_scaled, y_train)
# scoring on the scaled test set
svm.score(X_test_scaled, y_test)
```

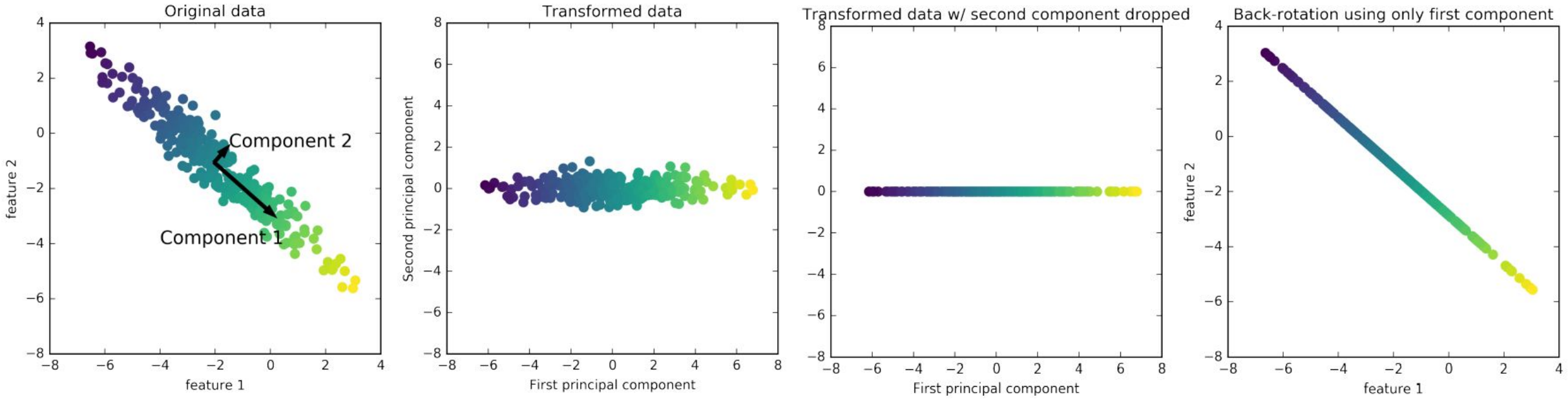
```
0.972027972027972
```

Con Normalizer

```
# preprocessing using normalizer
from sklearn.preprocessing import Normalizer
scaler = Normalizer()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
# learning an SVM on the scaled training data
svm.fit(X_train_scaled, y_train)
# scoring on the scaled test set
svm.score(X_test_scaled, y_test)
```

```
0.8671328671328671
```

Principal Component Analysis (PCA)
Análisis de componentes principales



PCA para visualizar datos

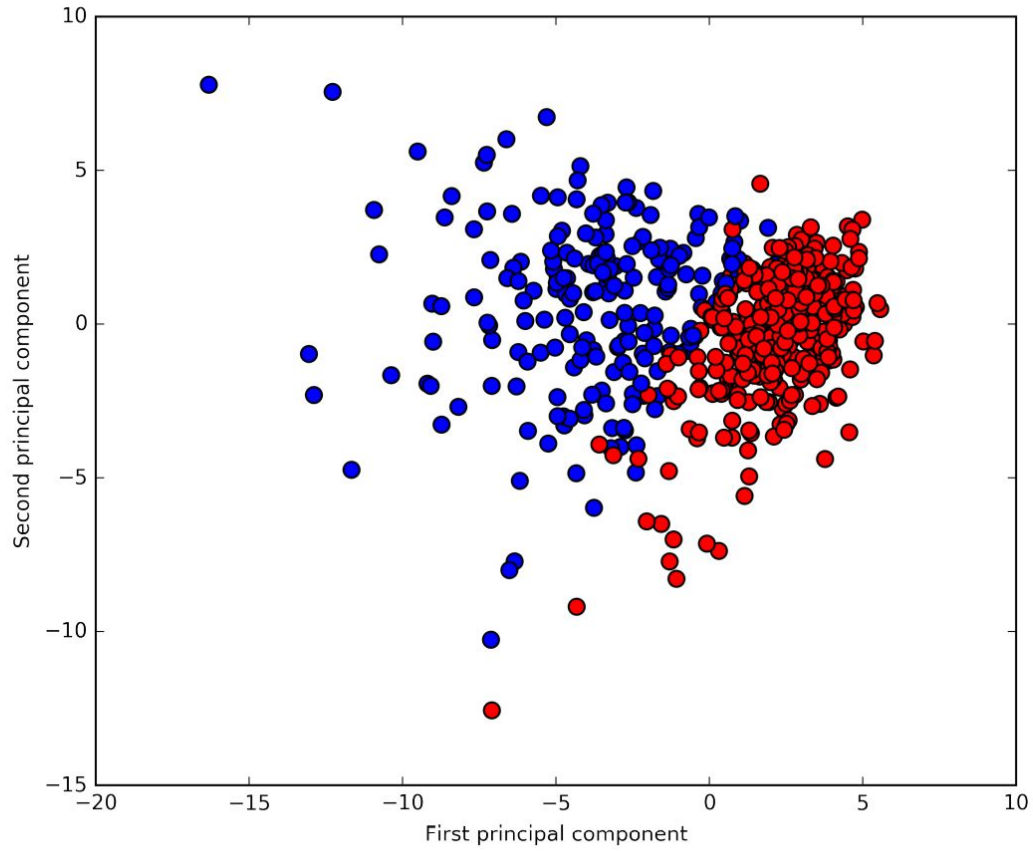
```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
scaler = StandardScaler()
scaler.fit(cancer.data)
X_scaled = scaler.transform(cancer.data)
```

```
from sklearn.decomposition import PCA
# keep the first two principal components of the data
pca = PCA(n_components=2)
# fit PCA model to beast cancer data
pca.fit(X_scaled)
# transform data onto the first two principal components
X_pca = pca.transform(X_scaled)
print("Original shape: %s" % str(X_scaled.shape))
print("Reduced shape: %s" % str(X_pca.shape))
```

```
Original shape: (569, 30)
Reduced shape: (569, 2)
```

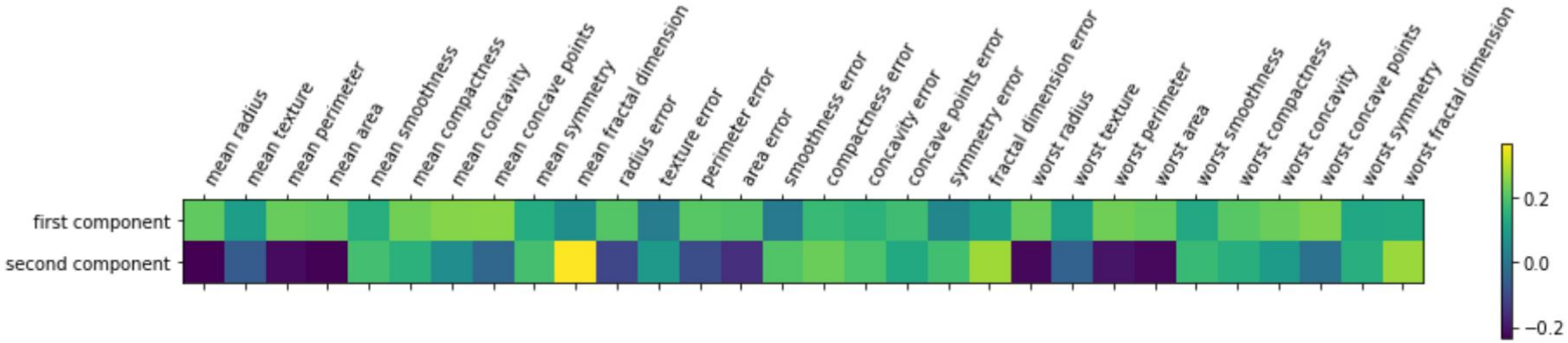
PCA para visualizar datos

```
# plot fist vs second principal component, color by class
plt.figure(figsize=(8, 8))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cancer.target, cmap=mglearn.tools.cm, s=60)
plt.gca().set_aspect("equal")
plt.xlabel("First principal component")
plt.ylabel("Second principal component")
```



Interpretando las componentes

```
plt.matshow(pca.components_, cmap='viridis')
plt.yticks([0, 1], ["first component", "second component"])
plt.colorbar()
plt.xticks(range(len(cancer.feature_names)),
 cancer.feature_names, rotation=60, ha='left');
#plt.suptitle("pca_components_cancer")
```



Feature extraction
Reducción de dimensionalidad

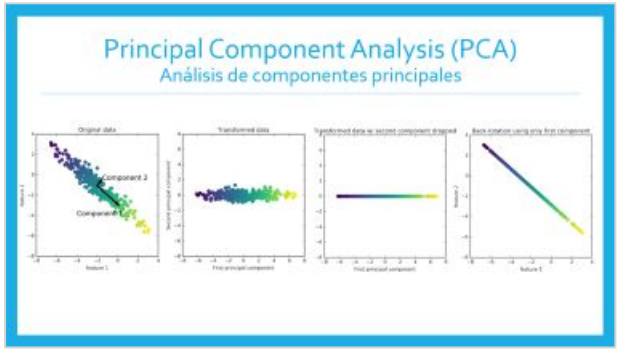
Es muy útiles con imágenes, donde cada pixel es una característica

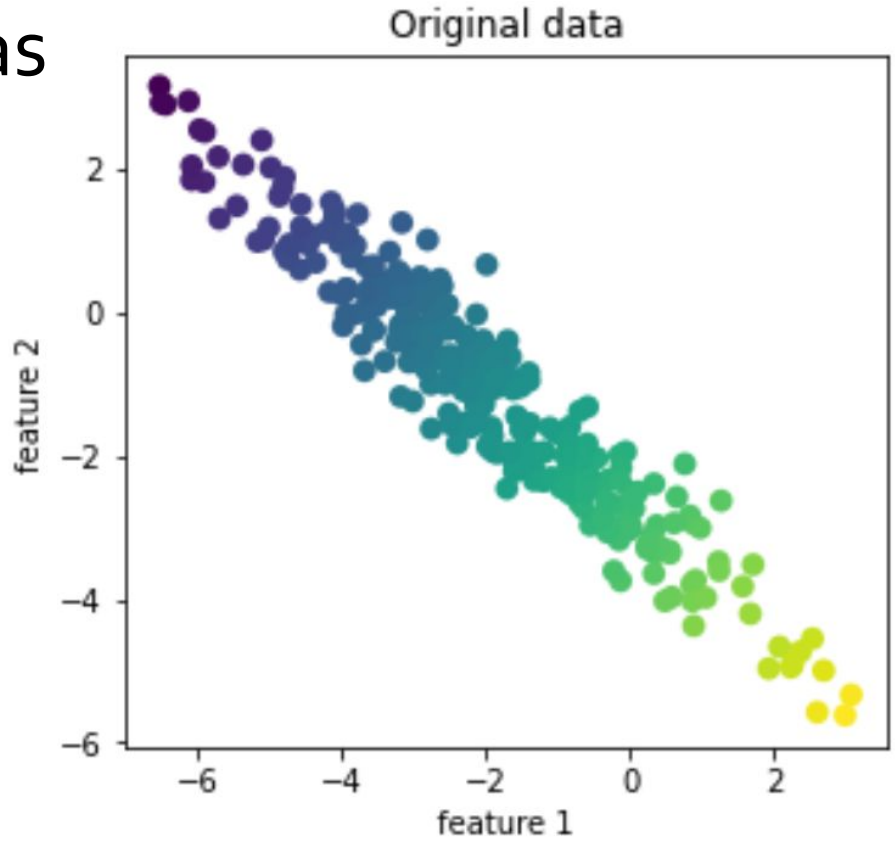
Queremos saber si una cara pertenece a una persona en la base de datos

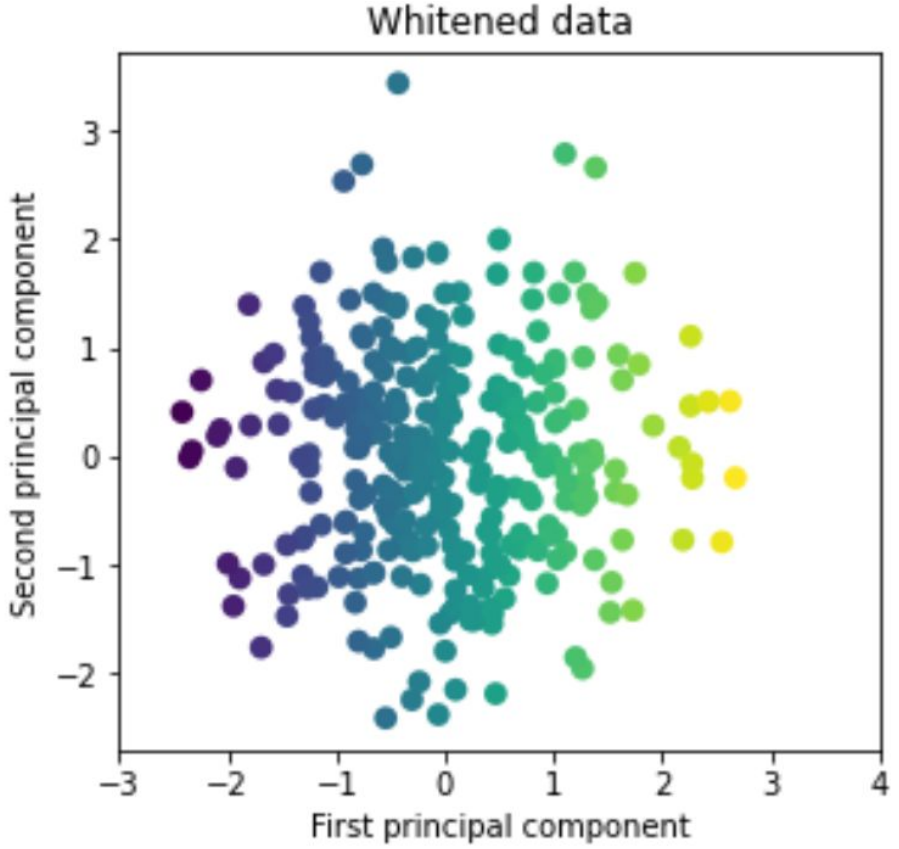
¿Cómo hacemos tal cosa?

Usando KNeighborsClassifier

Nos conviene tener buenas distancias







Diferencias al no usar PCA y al si usarlo

```
from sklearn.neighbors import KNeighborsClassifier
# split the data in training and test set
X_train, X_test, y_train, y_test = train_test_split(
 X_people, y_people, stratify=y_people, random_state=0)
# build a KNeighborsClassifier with using one neighbor:
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
```

```
0.23255813953488372
```

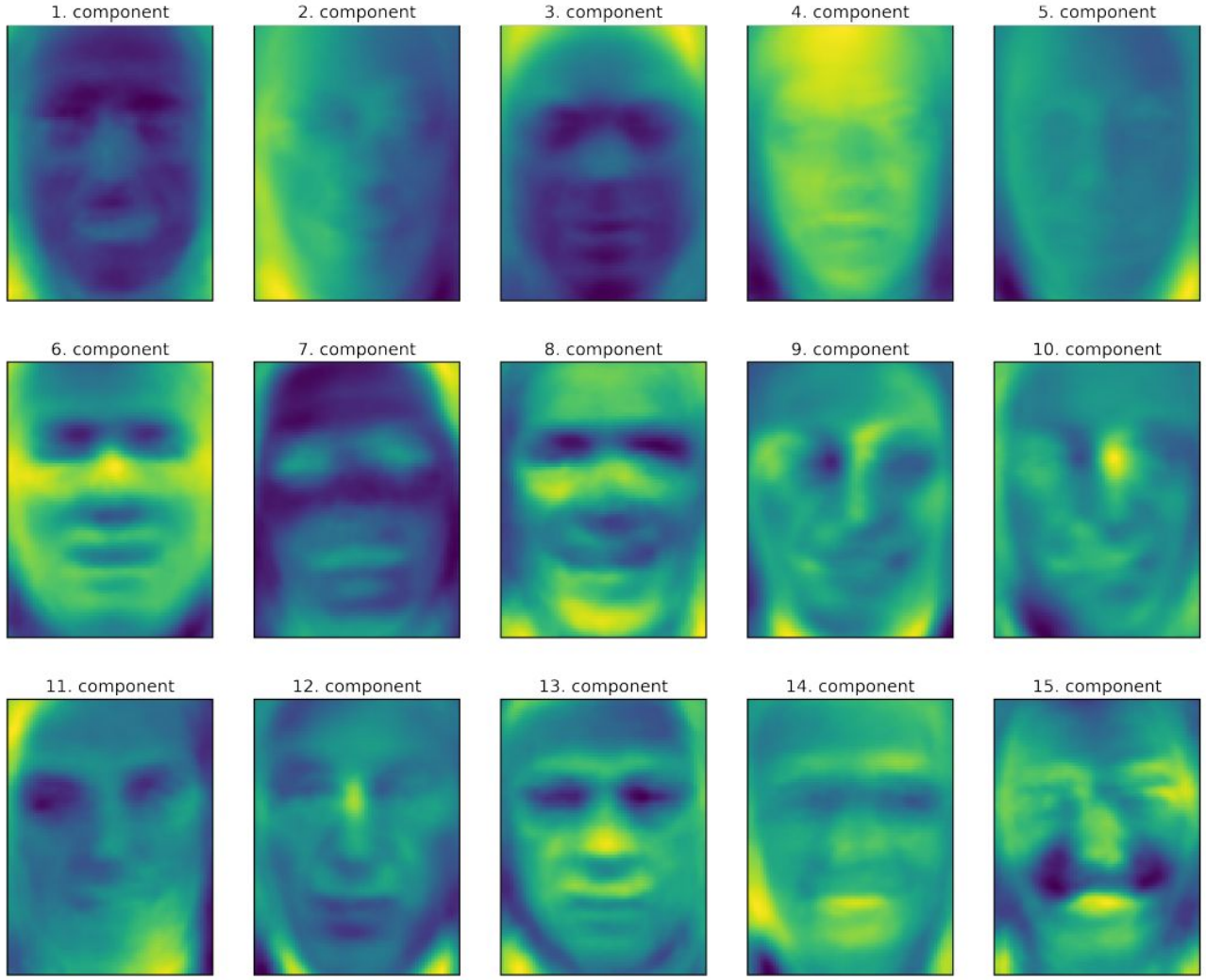
```
pca = PCA(n_components=100, whiten=True).fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print(X_train_pca.shape)
```

```
(1547, 100)
```

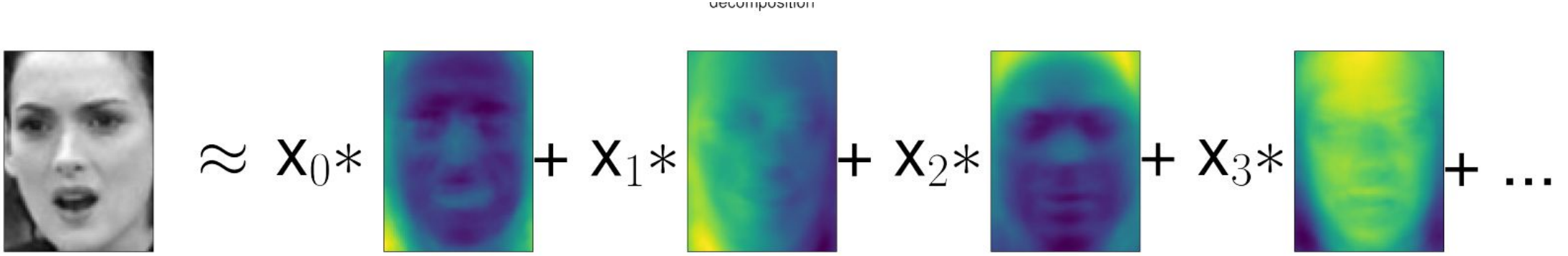
```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_pca, y_train)
knn.score(X_test_pca, y_test)
```

```
0.31007751937984496
```

Interpretamos las componentes



Entendiendo el modelo de otro modo

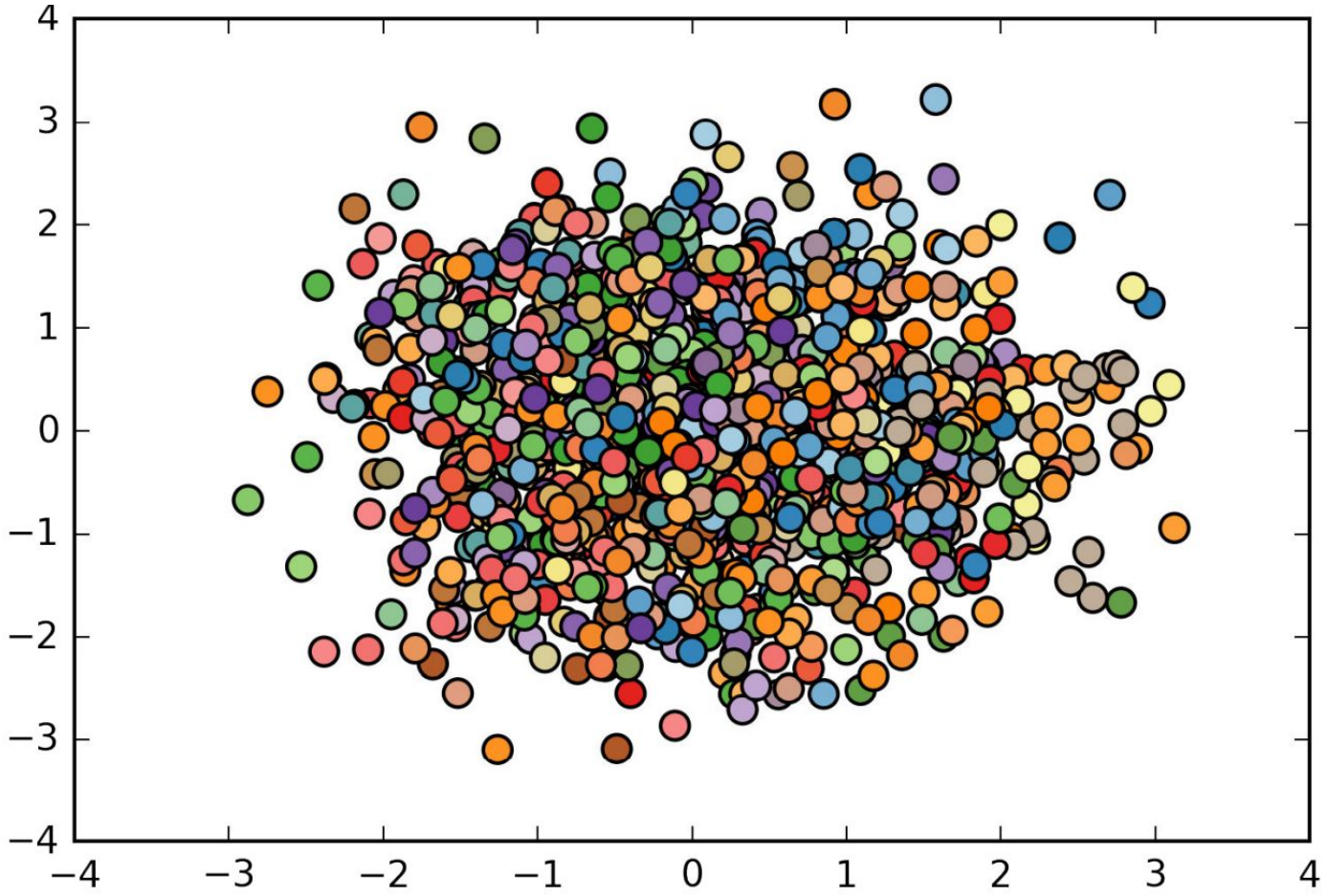


La influencia de las componentes

```
mglearn.plots.plot_pca_faces(X_train, X_test, image_shape)
plt.suptitle("pca_reconstructions")
```

¿Podremos visualizar los datos en dos dimensiones?

¿Podremos visualizar los datos en dos dimensiones?



No.