

W4995 Applied Machine Learning

# Preprocessing and Feature Transformations

02/06/19

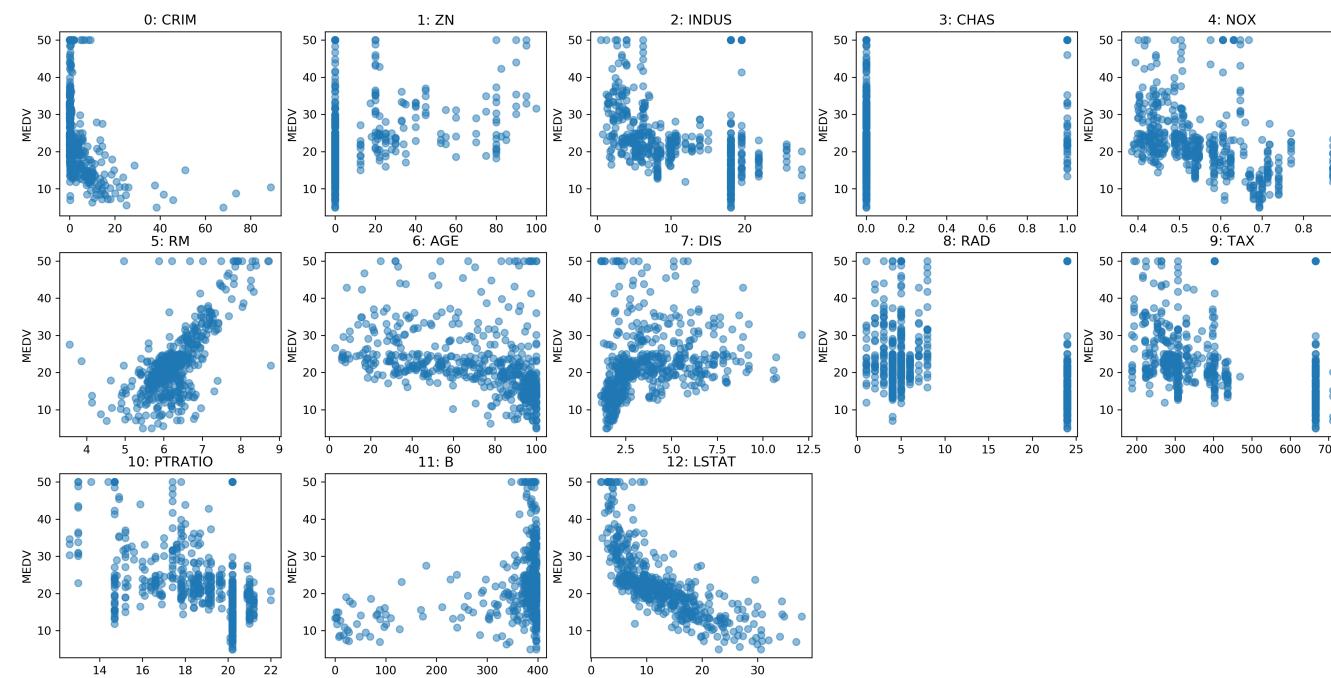
Andreas C. Müller

1 / 49

Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.

Andrew Ng

2 / 49



# Scaling

4 / 49

```
plt.boxplot(X)
plt.xticks(np.arange(1, X.shape[1] + 1), boston.feature_names, rotation=30, ha="right")
plt.ylabel("MEDV")
```

5 / 49

# Scaling and Distances

KNN w/o scaling

KNN with scaling

6 / 49

# Scaling and Distances

KNN w/o scaling



KNN with scaling



7 / 49

# Ways to Scale Data



8 / 49

# Sparse Data

- Data with many zeros – only store non-zero entries.
- Subtracting anything will make the data “dense” (no more zeros) and blow the RAM.
- Only scale, don’t center (use MaxAbsScaler)

# Standard Scaler Example

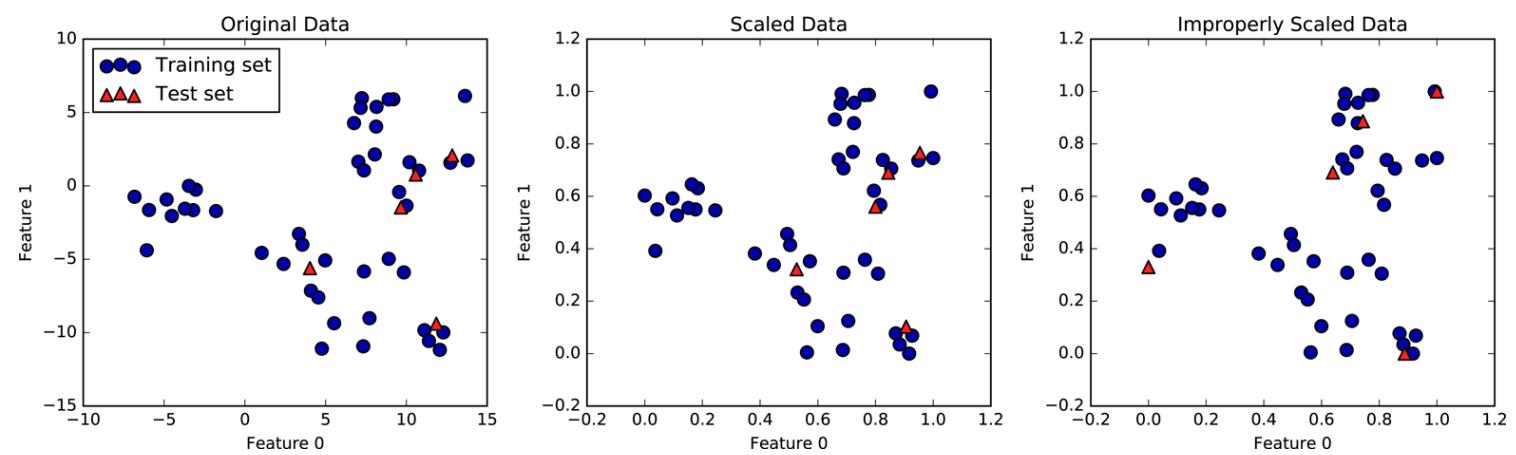
```
from sklearn.linear_model import Ridge
X, y = boston.data, boston.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=0)

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)

ridge = Ridge().fit(X_train_scaled, y_train)
X_test_scaled = scaler.transform(X_test)
ridge.score(X_test_scaled, y_test)
```

0.634

10 / 49



# Sckit-Learn API Summary

**estimator.fit(X, [y])**

12 / 49

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import RidgeCV
scores = cross_val_score(RidgeCV(), X_train, y_train, cv=10)
np.mean(scores), np.std(scores)

(0.717, 0.125)

scores = cross_val_score(RidgeCV(), X_train_scaled, y_train, cv=10)
np.mean(scores), np.std(scores)

(0.718, 0.127)

from sklearn.neighbors import KNeighborsRegressor
scores = cross_val_score(KNeighborsRegressor(), X_train, y_train, cv=10)
np.mean(scores), np.std(scores)

(0.499, 0.146)

from sklearn.neighbors import KNeighborsRegressor
scores = cross_val_score(KNeighborsRegressor(), X_train_scaled, y_train, cv=10)
np.mean(scores), np.std(scores)

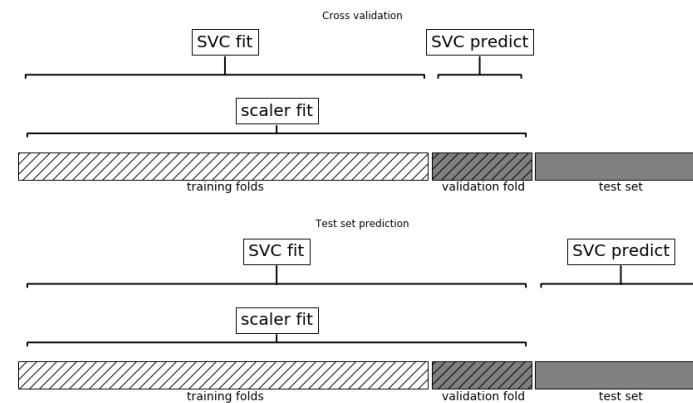
(0.750, 0.106)
```

# A note on preprocessing (and pipelines)

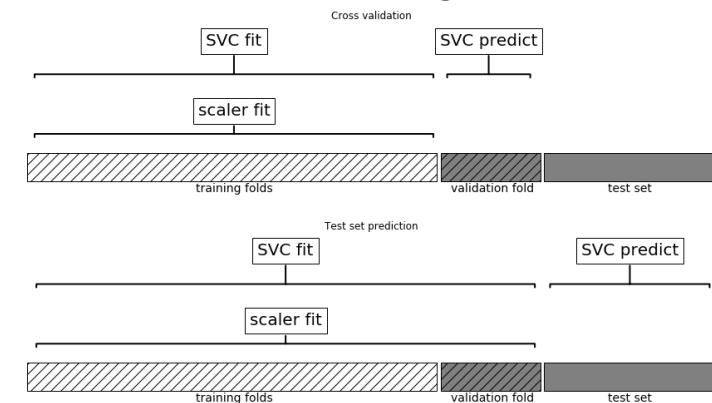
14 / 49

# Leaking Information

Information Leak



No Information leakage



Need to include preprocessing in cross-validation !

```
from sklearn.linear_model import Ridge
X, y = boston.data, boston.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
ridge = Ridge().fit(X_train_scaled, y_train)

X_test_scaled = scaler.transform(X_test)
ridge.score(X_test_scaled, y_test)
```

0.634

```
from sklearn.pipeline import make_pipeline
pipe = make_pipeline(StandardScaler(), Ridge())
pipe.fit(X_train, y_train)
pipe.score(X_test, y_test)
```

0.634

```
pipe = make_pipeline(T1(), T2(), Classifier())
```

17 / 49

```
from sklearn.neighbors import KNeighborsRegressor
knn_pipe = make_pipeline(StandardScaler(), KNeighborsRegressor())
scores = cross_val_score(knn_pipe, X_train, y_train, cv=10)
np.mean(scores), np.std(scores)
```

(0.745, 0.106)

# Naming Steps

```
from sklearn.pipeline import make_pipeline
knn_pipe = make_pipeline(StandardScaler(), KNeighborsRegressor())
print(knn_pipe.steps)
```

```
[('standardscaler', StandardScaler(with_mean=True, with_std=True)),
 ('kneighborsregressor', KNeighborsRegressor(algorithm='auto', ...))]
```

```
from sklearn.pipeline import Pipeline
pipe = Pipeline([('scaler', StandardScaler()),
                 ('regressor', KNeighborsRegressor())])
```

# Pipeline and GridSearchCV

```
from sklearn.model_selection import GridSearchCV

knn_pipe = make_pipeline(StandardScaler(), KNeighborsRegressor())
param_grid = {'kneighborsregressor__n_neighbors': range(1, 10)}
grid = GridSearchCV(knn_pipe, param_grid, cv=10)
grid.fit(X_train, y_train)
print(grid.best_params_)
print(grid.score(X_test, y_test))

{'kneighborsregressor__n_neighbors': 7}
0.60
```

# Discrete features

21 / 49

# Categorical Variables

```
import pandas as pd
df = pd.DataFrame(
    {'boro': ['Manhattan', 'Queens', 'Manhattan', 'Brooklyn', 'Brooklyn', 'Bronx'],
     'vegan': ['No', 'No', 'No', 'Yes', 'Yes', 'No']})
```

	boro	vegan
0	Manhattan	No
1	Queens	No
2	Manhattan	No
3	Brooklyn	Yes
4	Brooklyn	Yes
5	Bronx	No

# Ordinal encoding

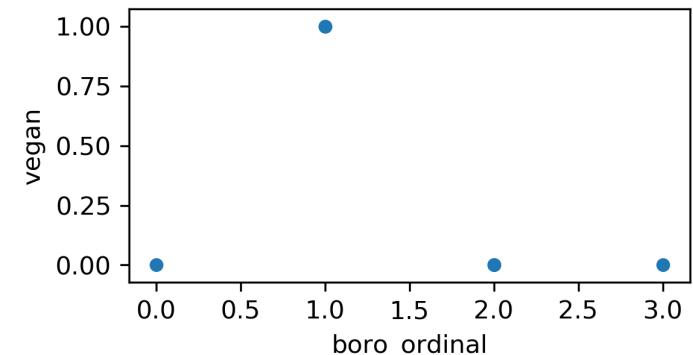
```
df['boro_ordinal'] = df.boro.astype("category").cat.codes  
df
```

	boro	boro_ordinal	vegan
0	Manhattan	2	No
1	Queens	3	No
2	Manhattan	2	No
3	Brooklyn	1	Yes
4	Brooklyn	1	Yes
5	Bronx	0	No

# Ordinal encoding

```
df['boro_ordinal'] = df.boro.astype("category").cat.codes  
df
```

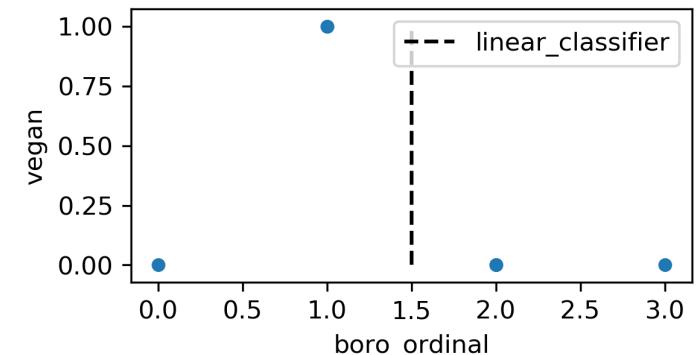
	boro	boro_ordinal	vegan
0	Manhattan	2	No
1	Queens	3	No
2	Manhattan	2	No
3	Brooklyn	1	Yes
4	Brooklyn	1	Yes
5	Bronx	0	No



# Ordinal encoding

```
df['boro_ordinal'] = df.boro.astype("category").cat.codes  
df
```

	boro	boro_ordinal	vegan
0	Manhattan	2	No
1	Queens	3	No
2	Manhattan	2	No
3	Brooklyn	1	Yes
4	Brooklyn	1	Yes
5	Bronx	0	No



# One-Hot (Dummy) Encoding

	boro	vegan
0	Manhattan	No
1	Queens	No
2	Manhattan	No
3	Brooklyn	Yes
4	Brooklyn	Yes
5	Bronx	No

```
pd.get_dummies(df)
```

	boro_Bronx	boro_Brooklyn	boro_Manhattan	boro_Queens	vegan_No	vegan_Yes
0	0	0	1	0	1	0
1	0	0	0	1	1	0
2	0	0	1	0	0	1
3	0	1	0	0	0	1
4	0	1	0	0	0	1
5	1	0	0	0	1	0

# One-Hot (Dummy) Encoding

	boro	vegan
0	Manhattan	No
1	Queens	No
2	Manhattan	No
3	Brooklyn	Yes
4	Brooklyn	Yes
5	Bronx	No

```
pd.get_dummies(df, columns=['boro'])
```

	vegan	boro_Bronx	boro_Brooklyn	boro_Manhattan	boro_Queens
0	No	0	0	1	0
1	No	0	0	0	1
2	No	0	0	1	0
3	Yes	0	1	0	0
4	Yes	0	1	0	0
5	No	1	0	0	0

# One-Hot (Dummy) Encoding

	boro	vegan
0	2	No
1	3	No
2	2	No
3	1	Yes
4	1	Yes
5	0	No

```
pd.get_dummies(df_ordinal, columns=['boro'])
```

	vegan	boro_0	boro_1	boro_2	boro_3
0	No	0	0	1	0
1	No	0	0	0	1
2	No	0	0	1	0
3	Yes	0	1	0	0
4	Yes	0	1	0	0
5	No	1	0	0	0

```
df = pd.DataFrame({'salary': [103, 89, 142, 54, 63, 219],  
                  'boro': [0, 1, 0, 2, 2, 3]})  
df
```

	boro	salary
0	0	103
1	1	89
2	0	142
3	2	54
4	2	63
5	3	219

```
pd.get_dummies(df)
```

	boro	salary
0	0	103
1	1	89
2	0	142
3	2	54
4	2	63
5	3	219

```
pd.get_dummies(df, columns=['boro'])
```

	salary	boro_0	boro_1	boro_2	boro_3
0	103	1.0	0.0	0.0	0.0
1	89	0.0	1.0	0.0	0.0
2	142	1.0	0.0	0.0	0.0
3	54	0.0	0.0	1.0	0.0
4	63	0.0	0.0	1.0	0.0
5	219	0.0	0.0	0.0	1.0

	boro	salary
0	Manhattan	103
1	Queens	89
2	Manhattan	142
3	Brooklyn	54
4	Brooklyn	63
5	Bronx	219

	salary	boro_Bronx	boro_Brooklyn	boro_Manhattan	boro_Queens
0	103	0.0	0.0	1.0	0.0
1	89	0.0	0.0	0.0	1.0
2	142	0.0	0.0	1.0	0.0
3	54	0.0	1.0	0.0	0.0
4	63	0.0	1.0	0.0	0.0
5	219	1.0	0.0	0.0	0.0

	boro	salary
0	Staten Island	73
1	Manhattan	98
2	Brooklyn	204
3	Bronx	54

	salary	boro_Bronx	boro_Brooklyn	boro_Manhattan	boro_Staten Island
0	73	0.0	0.0	0.0	1.0
1	98	0.0	0.0	1.0	0.0
2	204	0.0	1.0	0.0	0.0
3	54	1.0	0.0	0.0	0.0

# Pandas Categorical Columns

```
import pandas as pd
df = pd.DataFrame({'salary': [103, 89, 142, 54, 63, 219],
                    'boro': ['Manhattan', 'Queens', 'Manhattan',
                             'Brooklyn', 'Brooklyn', 'Bronx']})

df['boro'] = pd.Categorical(df.boro, categories=['Manhattan', 'Queens', 'Brooklyn',
                                                 'Bronx', 'Staten Island'])
pd.get_dummies(df)
```

	salary	boro_Manhattan	boro_Queens	boro_Brooklyn	boro_Bronx	boro_Staten Island
0	103	1	0	0	0	0
1	89	0	1	0	0	0
2	142	1	0	0	0	0
3	54	0	0	1	0	0
4	63	0	0	1	0	0
5	219	0	0	0	1	0

# OneHotEncoder

```
from sklearn.preprocessing import OneHotEncoder

df = pd.DataFrame({'salary': [103, 89, 142, 54, 63, 219],
                   'boro': [0, 1, 0, 2, 2, 3]})

ohe = OneHotEncoder(categorical_features=[0]).fit(df)
ohe.transform(df).toarray()
```

```
array([[ 1.,    0.,    0.,    0.,   103.],
       [ 0.,    1.,    0.,    0.,    89.],
       [ 1.,    0.,    0.,    0.,   142.],
       [ 0.,    0.,    1.,    0.,    54.],
       [ 0.,    0.,    1.,    0.,    63.],
       [ 0.,    0.,    0.,    1.,   219.]])
```

- Legacy mode: Only works for integers right now, not strings.
- New mode: works on strings and integers, but always transforms all columns.

# OneHotEncoder

```
import pandas as pd
df = pd.DataFrame({'salary': [103, 89, 142, 54, 63, 219],
                    'boro': ['Manhattan', 'Queens', 'Manhattan',
                             'Brooklyn', 'Brooklyn', 'Bronx']})

ce = OneHotEncoder().fit(df)
ce.transform(df).toarray()
```

```
array([[ 0.,  0.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  1.,  0.],
       [ 0.,  1.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.]])
```

- Always transforms all columns

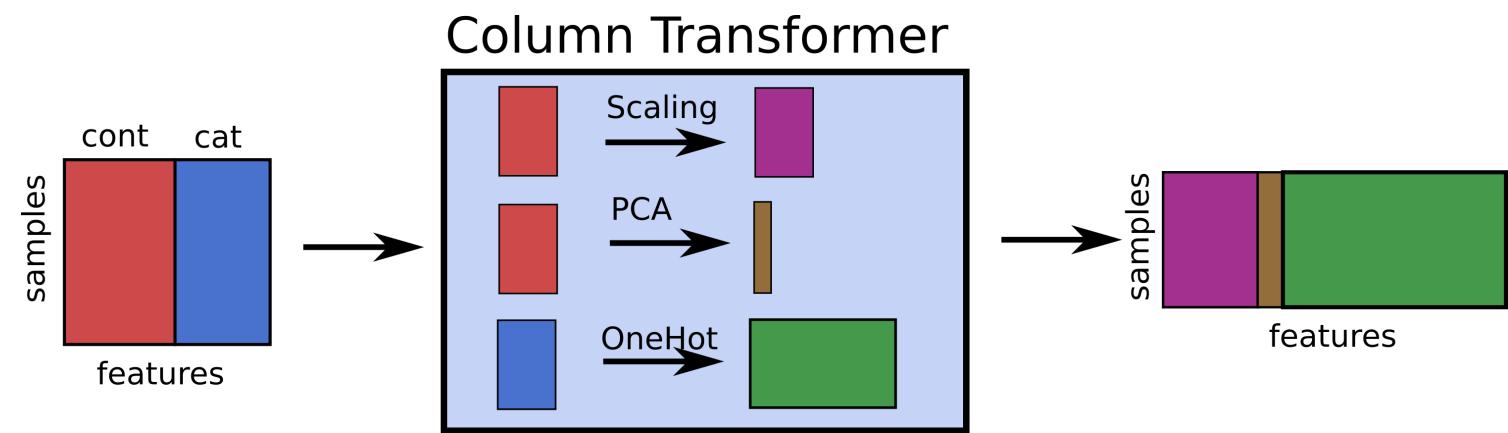
# The Future (it's here)

## OneHotEncoderEncoder + ColumnTransformer

```
categorical = df.dtypes == object

preprocess = make_column_transformer(
    (StandardScaler(), ~categorical),
    (OneHotEncoder(), categorical))

model = make_pipeline(preprocess, LogisticRegression())
```



# Dummy variables and colinearity

- One-hot is redundant (last one is  $1 - \sum$  of others)
- Can introduce co-linearity
- Can drop one
- Choice which one matters for penalized models
- Keeping all can make the model more interpretable

# Models Supporting Discrete Features

- In principle:
  - All tree-based models, naive Bayes
- In scikit-learn:
  - None
- In scikit-learn "soon":
  - Decision trees, random forests, gradient boosting

# More encodings for categorical features:

<http://contrib.scikit-learn.org/categorical-encoding/>

# Target-Based Encoding

- For high cardinality categorical features
- Example: US states, given low samples
- Instead of 50 one-hot variables, one “response encoded” variable.
- For regression:
  - “people in this state have an average response of  $y$ ”
- Binary classification: – “people in this state have likelihood  $p$  for class 1”
- Multiclass: – One feature per class: probability distribution

# Example: Adult census, native-country

```
data = pd.read_csv("adult.csv")
data.columns
Index(['age', 'workclass', 'education', 'education-num', 'marital-status',
       'occupation', 'relationship', 'race', 'gender', 'capital-gain',
       'capital-loss', 'hours-per-week', 'native-country', 'income'], dtype='object')
data['native-country'].value_counts()
```

United-States	29170	Vietnam	67
Mexico	643	Guatemala	64
?	583	Japan	62
Philippines	198	Poland	60
Germany	137	Columbia	59
Canada	121	Taiwan	51
Puerto-Rico	114	Haiti	44
El-Salvador	106	Iran	43
India	100	Portugal	37
Cuba	95	Nicaragua	34
England	90	Peru	31
Jamaica	81	France	29
South	80	Greece	29
China	75	Ecuador	28

40 / 49

	<=50K	>50K
?	0.749571	0.250429
Cambodia	0.631579	0.368421
Canada	0.677686	0.322314
China	0.733333	0.266667
Columbia	0.966102	0.033898
Cuba	0.736842	0.263158
Dominican-Republic	0.971429	0.028571
Ecuador	0.857143	0.142857
El-Salvador	0.915094	0.084906
England	0.666667	0.333333
France	0.586207	0.413793
Germany	0.678832	0.321168
Greece	0.724138	0.275862
Guatemala	0.953125	0.046875
Haiti	0.909091	0.090909
Holand-Netherlands	1.000000	0.000000
Honduras	0.923077	0.076923
Hong	0.700000	0.300000
Hungary	0.769231	0.230769
India	0.600000	0.400000
Iran	0.581395	0.418605
Ireland	0.791667	0.208333
Italy	0.657534	0.342466
Jamaica	0.876543	0.123457
Japan	0.612903	0.387097
Laos	0.888889	0.111111
Mexico	0.948678	0.051322
Nicaragua	0.941176	0.058824
Outlying-US(Guam-USVI-etc)	1.000000	0.000000
Peru	0.935484	0.064516
Philippines	0.691919	0.308081

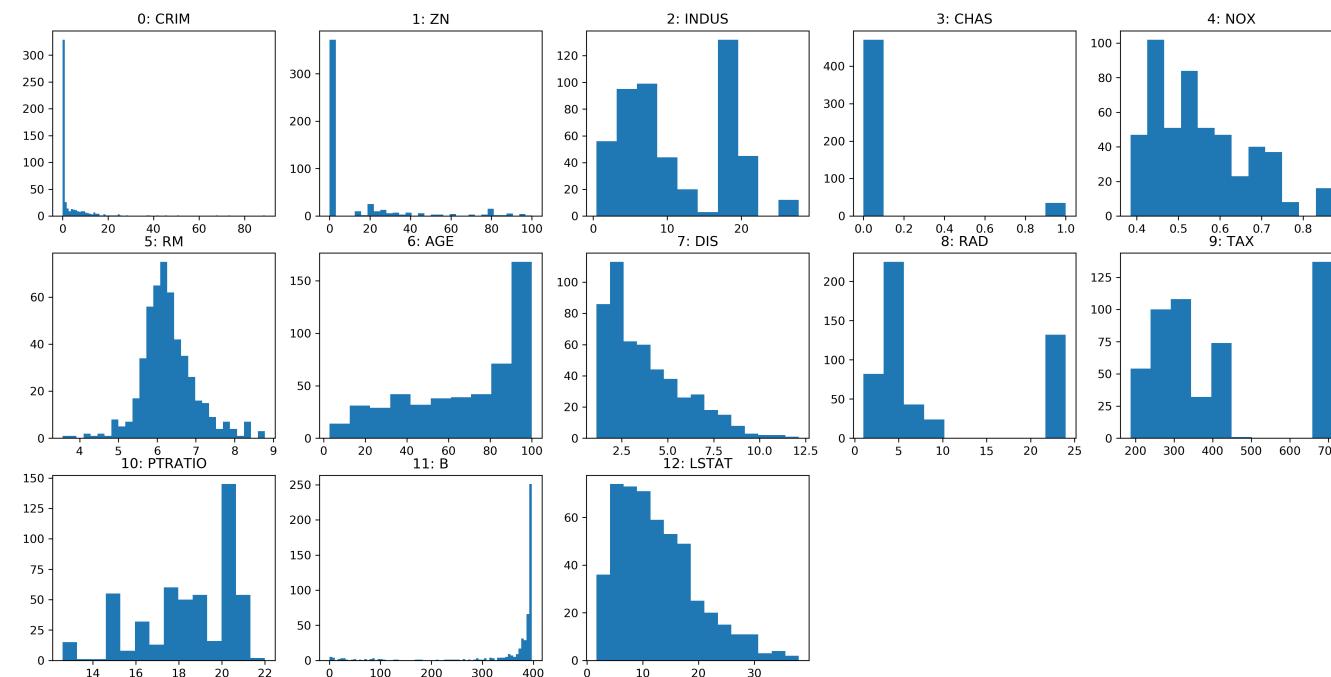
	<=50K	>50K	frequency	native-country	income
?	0.749571	0.250429	0.245835	United-States	<=50K
Cambodia	0.631579	0.368421	0.245835	United-States	<=50K
Canada	0.677686	0.322314	0.245835	United-States	<=50K
China	0.733333	0.266667	0.245835	United-States	<=50K
Columbia	0.966102	0.033898	0.263158	Cuba	<=50K
Cuba	0.736842	0.263158	0.245835	United-States	<=50K
Dominican-Republic	0.971429	0.028571	0.123457	Jamaica	<=50K
Ecuador	0.857143	0.142857	0.245835	United-States	>50K
El-Salvador	0.915094	0.084906	0.245835	United-States	>50K
England	0.666667	0.333333	0.245835	United-States	>50K
France	0.586207	0.413793	0.245835	United-States	>50K
Germany	0.678832	0.321168	0.400000	India	>50K
Greece	0.724138	0.275862	0.245835	United-States	<=50K
Guatemala	0.953125	0.046875	0.245835	United-States	<=50K
Haiti	0.909091	0.090909	0.250429	?	>50K
Holand-Netherlands	1.000000	0.000000	0.051322	Mexico	<=50K
Honduras	0.923077	0.076923	0.245835	United-States	<=50K
Hong	0.700000	0.300000	0.245835	United-States	<=50K
Hungary	0.769231	0.230769	0.245835	United-States	<=50K
India	0.600000	0.400000	0.245835	United-States	>50K
Iran	0.581395	0.418605	0.245835	United-States	>50K
Ireland	0.791667	0.208333	0.245835	United-States	<=50K
Italy	0.657534	0.342466	0.245835	United-States	<=50K
Jamaica	0.876543	0.123457	0.245835	United-States	<=50K
Japan	0.612903	0.387097	0.245835	United-States	<=50K
Laos	0.888889	0.111111	0.245835	United-States	>50K
Mexico	0.948678	0.051322	0.245835	United-States	<=50K
Nicaragua	0.941176	0.058824	0.200000	South	>50K
Outlying-US(Guam-USVI-etc)	1.000000	0.000000	0.245835	United-States	<=50K
Peru	0.935484	0.064516	0.245835	United-States	<=50K
Philippines	0.691919	0.308081	0.245835	United-States	<=50K

12 / 49

# Feature Distributions

43 / 49

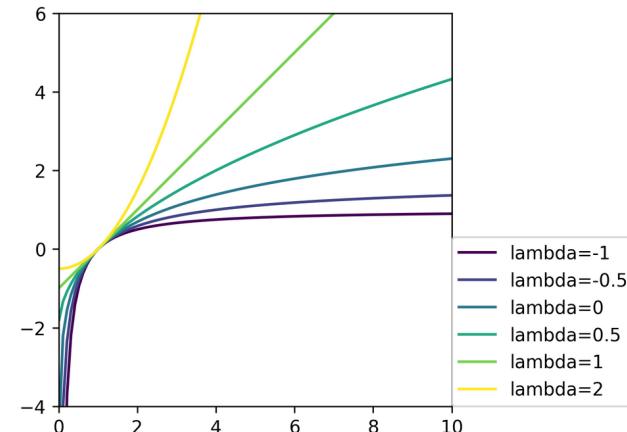




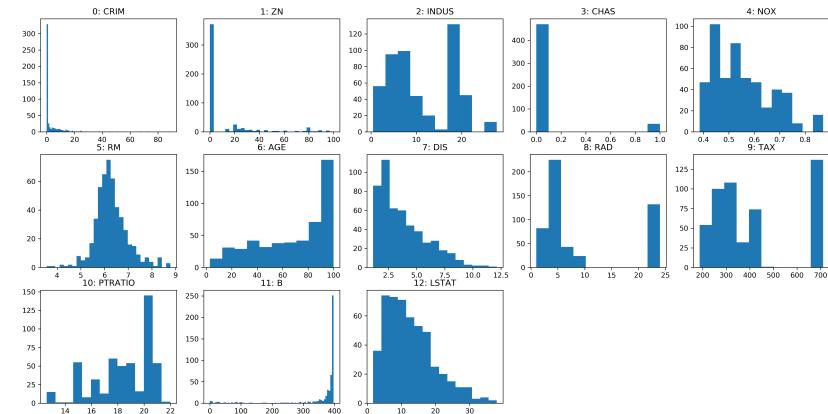
# Power Transformations

$$bc_\lambda(x) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log(x) & \text{if } \lambda = 0 \end{cases}$$

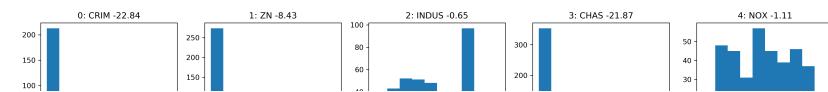
Only applicable for positive x!



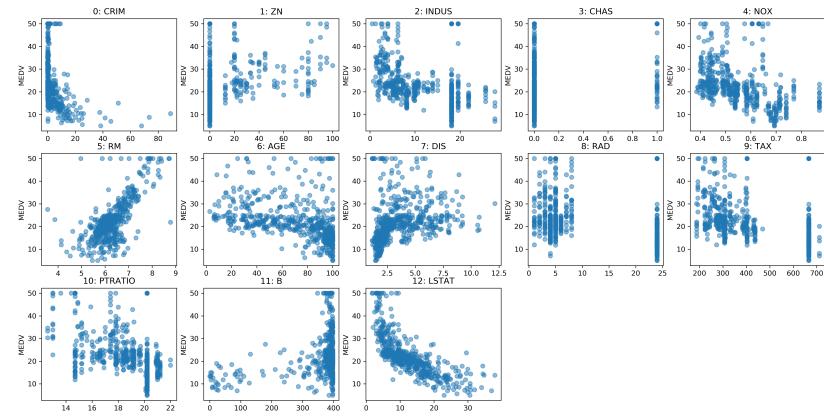
```
from sklearn.preprocessing import PowerTransformer  
pt = PowerTransformer(method='box-cox')  
# for any data: Yeo-Johnson  
pt.fit(X)
```



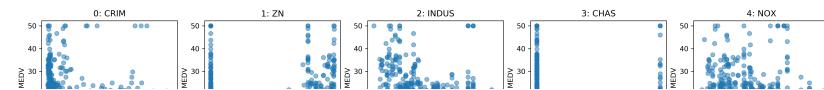
Before



After



Before



After

48 / 49

