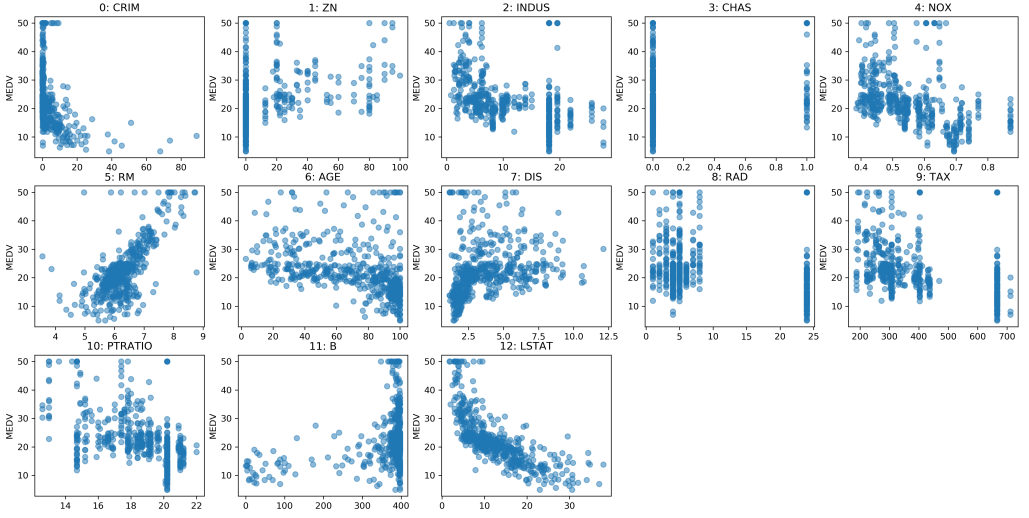
Applied Machine Learning

Preprocessing and Feature Engineering

03/04/19

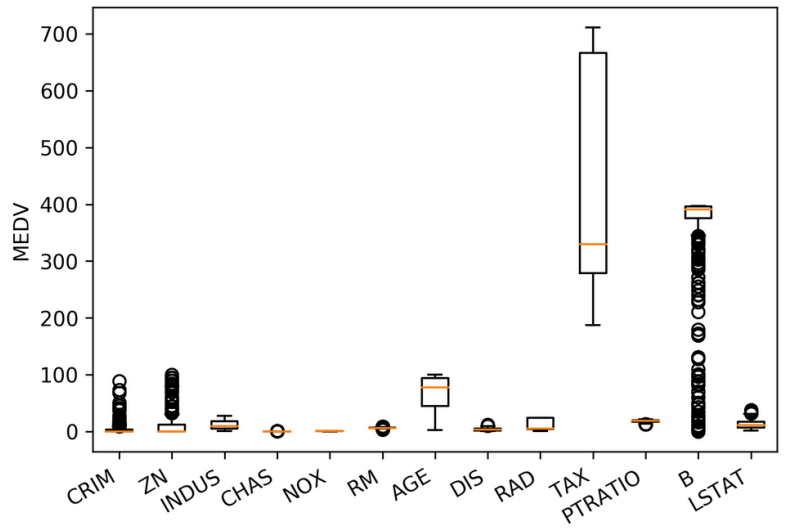
Yasin Ceran


0: CRIM
1: ZN
2: INDUS
3: CHAS
4: NOX
5: RM
6: AGE
7: DIS
8: RAD
9: TAX
10: PTRATIO
11: B
12: LSTAT
MEDV

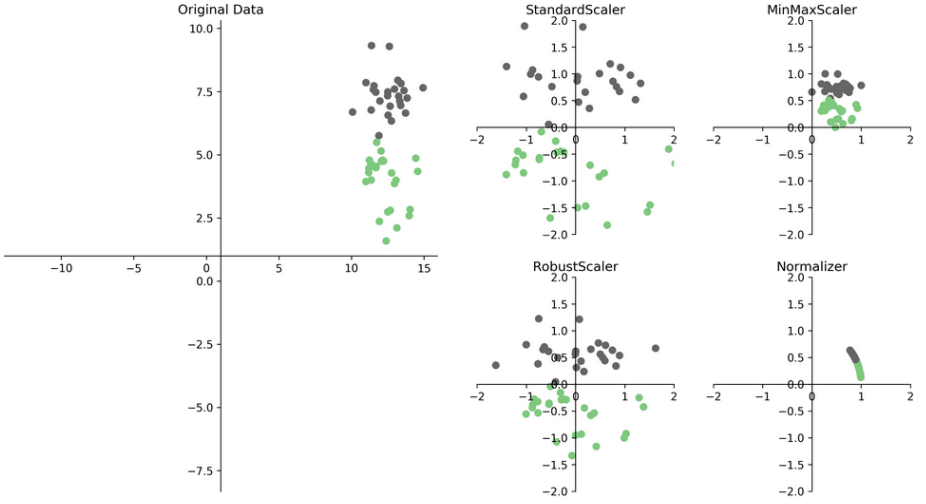
Scaling

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

```
<matplotlib.text.Text at 0x7f580303eac8>
```



Ways to Scale Data



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)

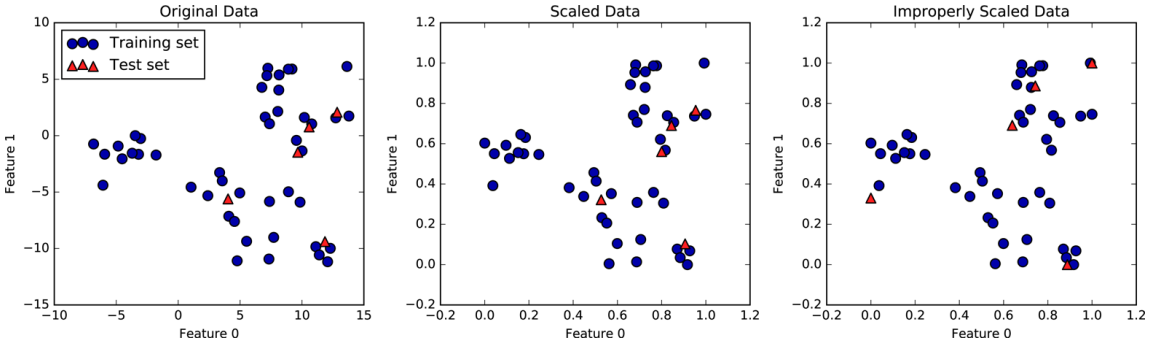
Standard Scaler Example

```
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
```


Original Data
Training set
Test set
Feature 1
Feature 0
Scaled Data
Feature 1
Feature 0
Improperly Scaled Data
Feature 1
Feature 0

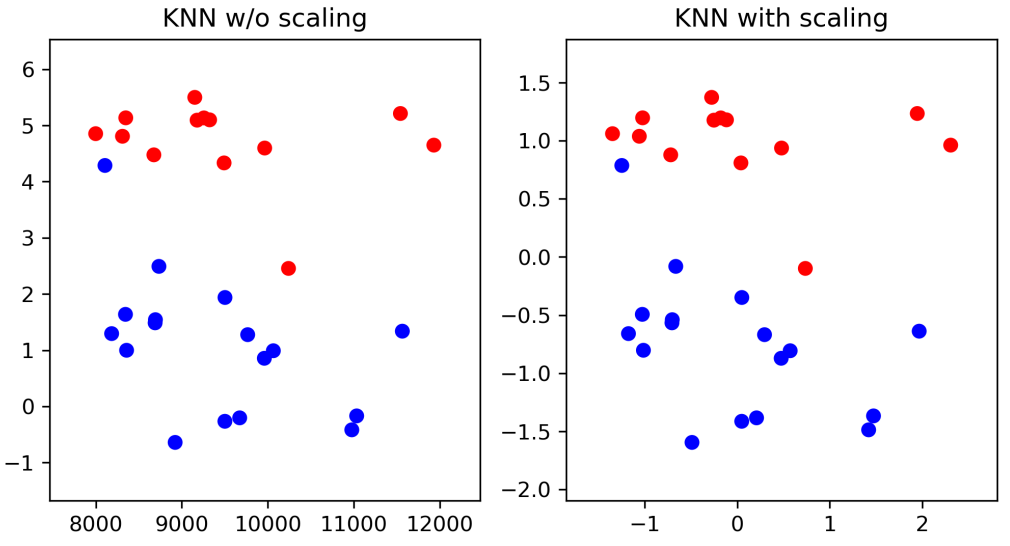
Sckit-Learn API Summary

estimator.fit(X_train, [y_train])	
estimator.predict(X_test)	**estimator.transform(X_test)**
Classification	Preprocessing
Regression	Dimensionality Reduction
Clustering	Feature Extraction
	Feature selection

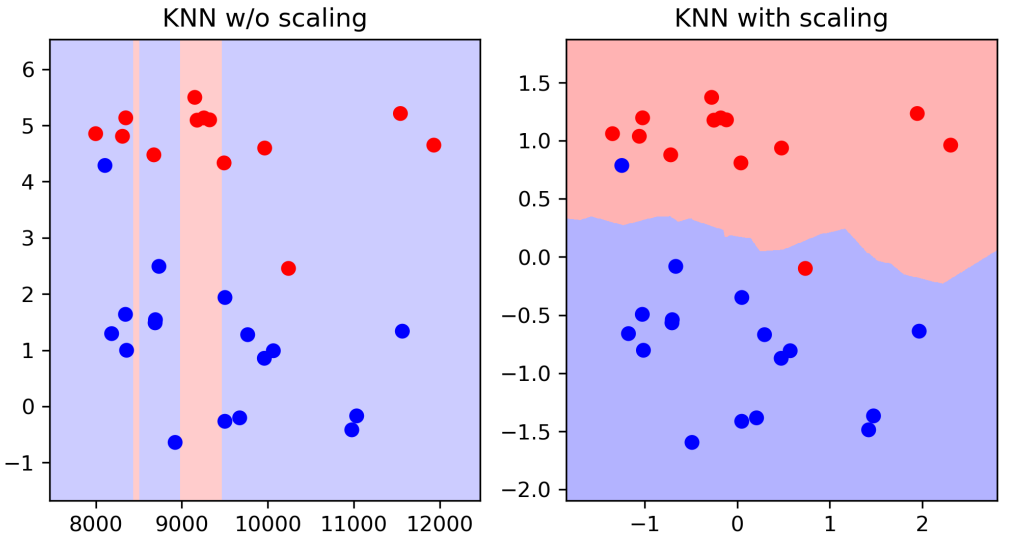
Efficient shortcuts:

```
est.fit_transform(X) == est.fit(X).transform(X)  # mostly
est.fit_predict(X) == est.fit(X).predict(X)   # mostly
```

Scaling and Distances



Scaling and Distances



```
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)
```

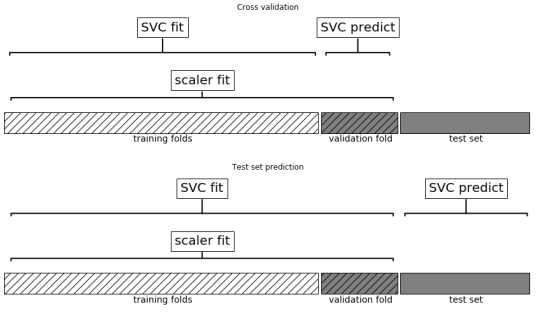
```
from sklearn.neighbors import KNeighborsRegressor
scores = cross_val_score(KNeighborsRegressor(), X_train_scaled, y_train, cv=10)
np.mean(scores), np.std(scores)
```

```
(0.750, 0.106)
```

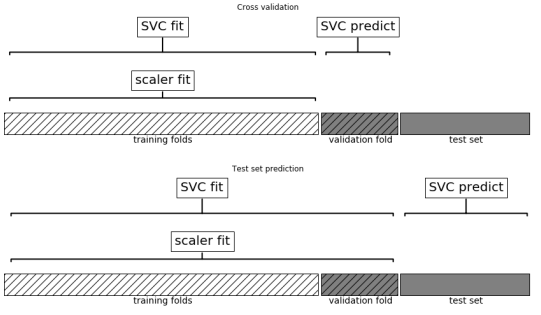
A note on preprocessing (and pipelines)

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)
```

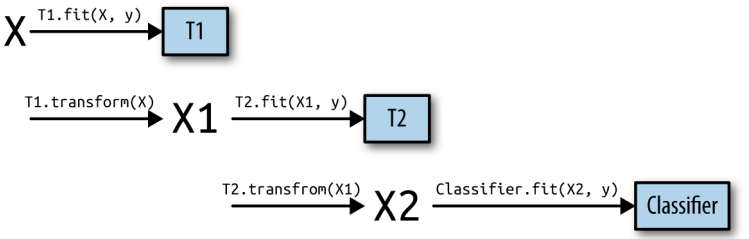
```
0.634
```

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

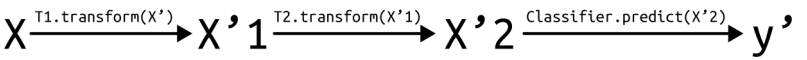
```
0.634
```

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


```
pipe.fit(X, y)
```




```
pipe.predict(X')
```



```
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)))
```

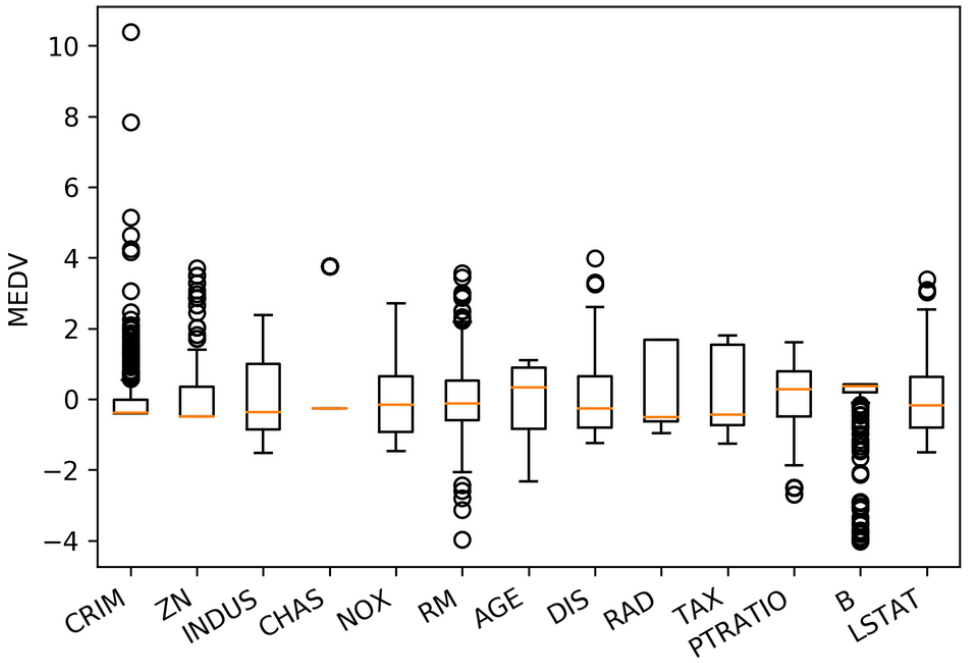
Pipeline and GridSearchCV

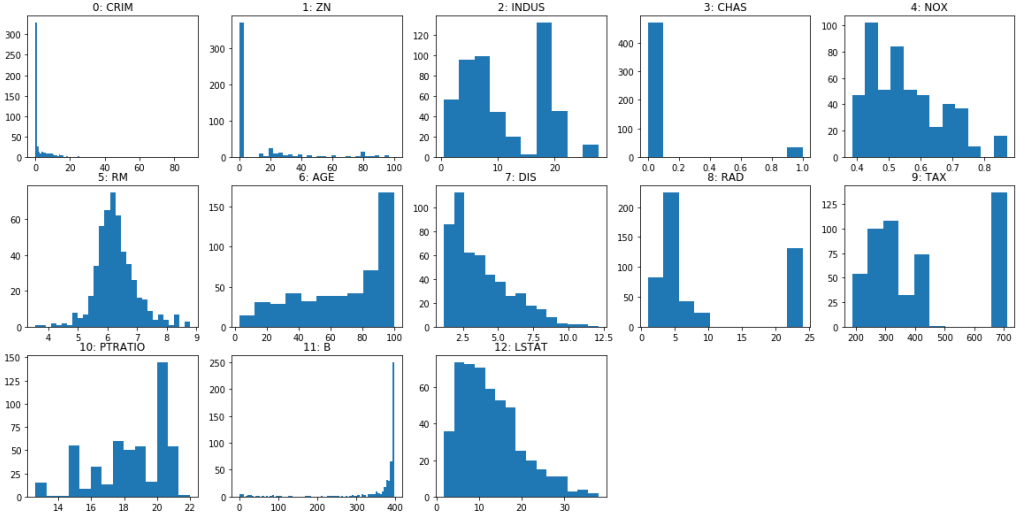
```
from sklearn.model_selection import GridSearchCV

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
```

Feature Distributions

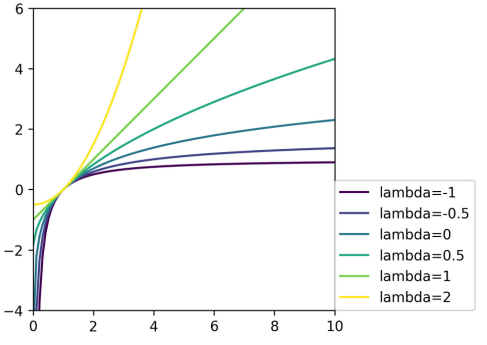

MEDV
10
8
6
4
2
0
−2
−4
CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PTRATIO
B
LSTAT


0: CRIM
1: ZN
2: INDUS
3: CHAS
4: NOX
5: RM
6: AGE
7: DIS
8: RAD
9: TAX
10: PTRATIO
11: B
12: LSTAT

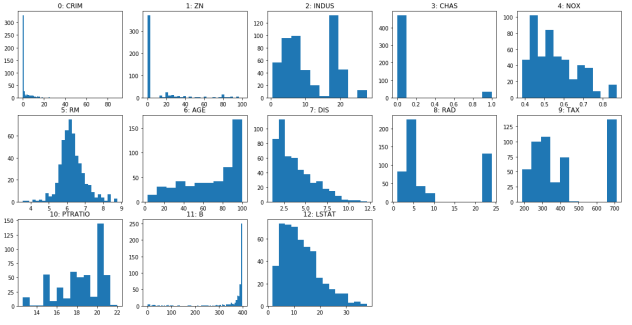
Box-Cox Transform

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

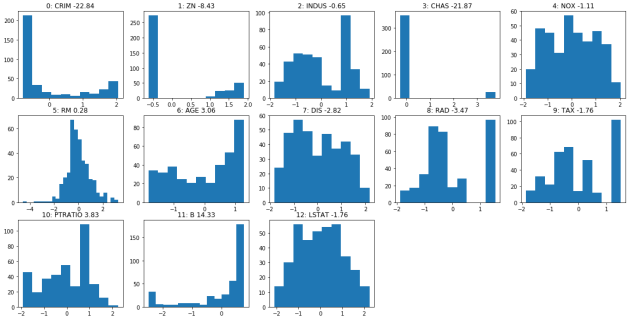
Only applicable for positive x!



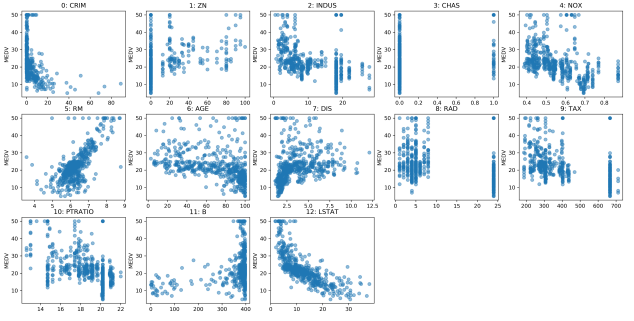

```
# sklearn 0.20-dev
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer(method='box-cox')
# soon: Yeo-Johnson
pt.fit(X)
```



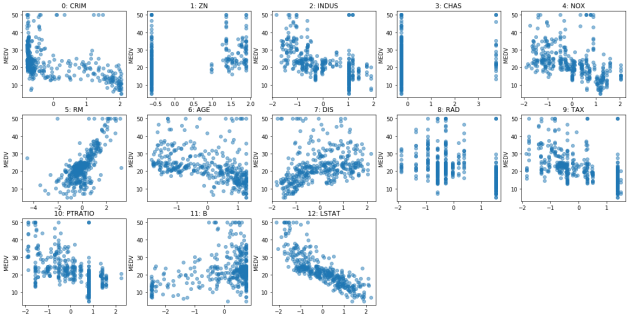
Before



After



Before



After

Discrete features

Categorical Variables

$\{'red', 'green', 'blue'\} \subset \mathbb{R}^p$?

Categorical Variables

"red"	"green"	"blue"
1	0	0
0	1	0
0	0	1

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

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

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

	salary	boro_Bronx	boro_Brooklyn	boro_Manhatten	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

```
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	Manhatten	103
1	Queens	89
2	Manhatten	142
3	Brooklyn	54
4	Brooklyn	63
5	Bronx	219

	salary	boro_Bronx	boro_Brooklyn	boro_Manhatten	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	Manhatten	98
2	Brooklyn	204
3	Bronx	54

	salary	boro_Bronx	boro_Brooklyn	boro_Manhatten	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

- Deprecated mode, don't use.

```
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.]])
```

OneHotEncoder

- New mode: always transforms all columns

```
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.]])
```

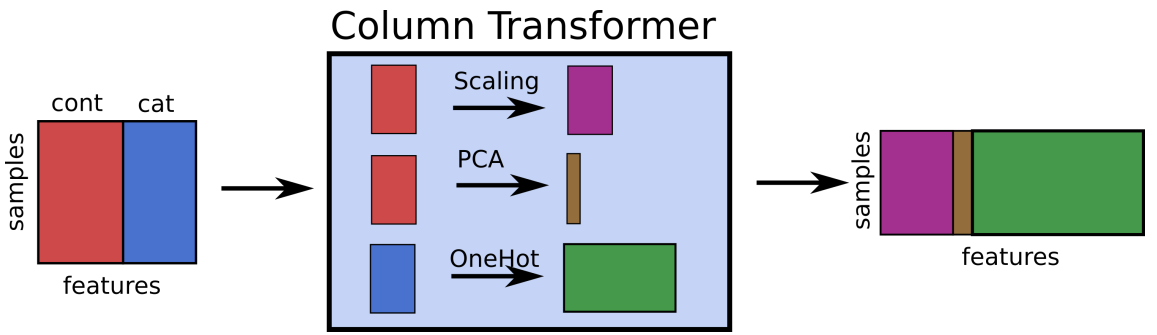
The Future (it's here)

OneHotEncoderEncoder + ColumnTransformer

```
categorical = df.dtypes == object

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

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


Column Transformer
cont
cat
samples
features
Scaling
PCA
OneHot
samples
features

Count-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()
```

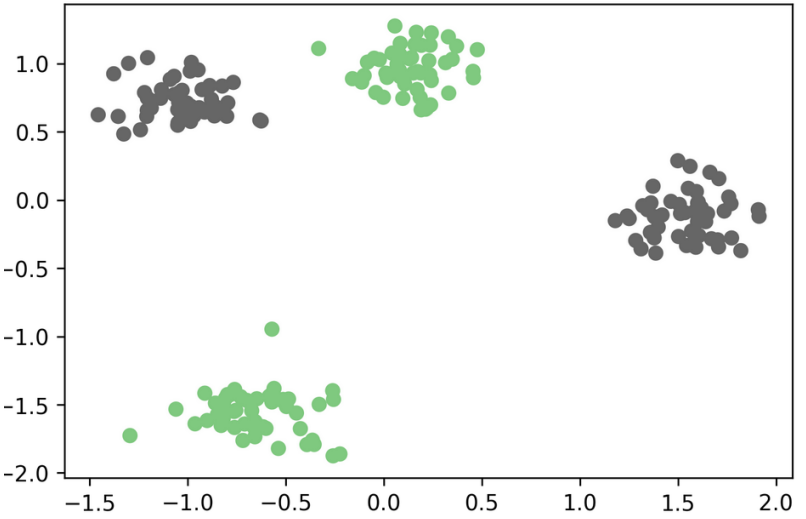
```
United-States                 29170
Mexico                          643
?                               583
Philippines                     198
Germany                         137
Canada                          121
Puerto-Rico                     114
El-Salvador                     106
India                           100
Cuba                             95
England                          90
Jamaica                          81
South                            80
China                            75
Italy                            73
Dominican-Republic               70
Vietnam                          67
Guatemala                        64
Japan                            62
Poland                           60
Columbia                         59
Taiwan                           51
Haiti                            44
Iran                             43
Portugal                         37
Nicaragua                        34
Peru                             31
France                           29
Greece                           29
Ecuador                          28
Ireland                          24
Hong                             20
```

	<=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

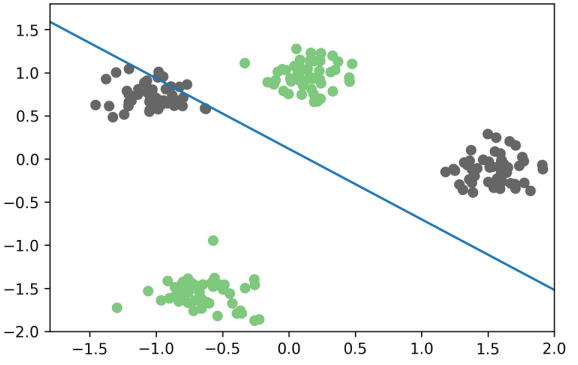
	<=50K	>50K	frequency	native-country	income
?	0.749571	0.250429	0.245835	United-States	<=50K
Cambodia	0.631579	0.368421	0.245835	United-States	<=50K
Canada	0.677686	0.322314	0.245835	United-States	<=50K
China	0.733333	0.266667	0.245835	United-States	<=50K
Columbia	0.966102	0.033898	0.263158	Cuba	<=50K
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

Feature Engineering

Interaction Features



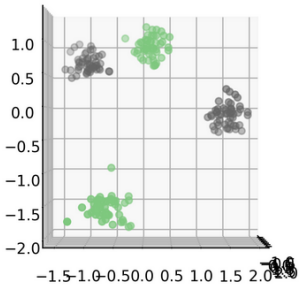
Interaction Features

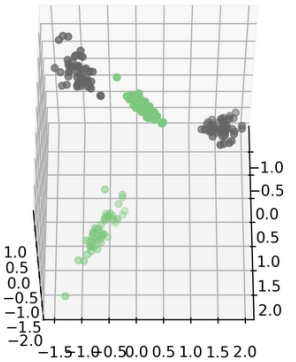


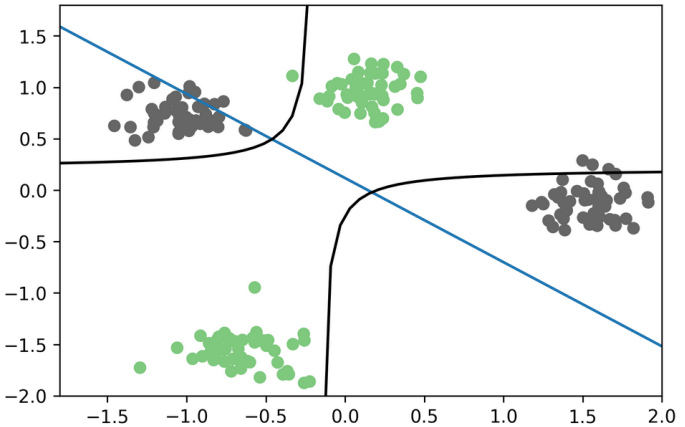

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
logreg = LogisticRegressionCV().fit(X_train, y_train)
logreg.score(X_test, y_test)
```

```
0.5
```

```
# Same as PolynomialFeatures(order=2, interactions_only=True)
X_interaction = np.hstack([X, X[:, 0:1] * X[:, 1:]])
```








```
X_i_train, X_i_test, y_train, y_test = train_test_split(
    X_interaction, y, random_state=0)
logreg3 = LogisticRegressionCV().fit(X_i_train, y_train)
logreg3.score(X_i_test, y_test)
```

```
0.960
```

	age	articles_bought	gender	spend$	time_online
0	14	5	M	70	269
1	16	10	F	12	1522
2	12	2	M	42	235
3	25	1	F	64	63
4	22	1	F	93	21

	age	articles_bought	spend$	time_online	gender_F	gender_M
0	14	5	70	269	0.0	1.0
1	16	10	12	1522	1.0	0.0
2	12	2	42	235	0.0	1.0
3	25	1	64	63	1.0	0.0
4	22	1	93	21	1.0	0.0

	age_M	articles_bought_M	spend$_M	time_online_M	gender_M_M	age_F	articles_bought_F	spend$_F	time_online_F	gender_F_F
0	14.0	5.0	70.0	269.0	1.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	16.0	10.0	12.0	1522.0	1.0
2	12.0	2.0	42.0	235.0	1.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	25.0	1.0	64.0	63.0	1.0
4	0.0	0.0	0.0	0.0	0.0	22.0	1.0	93.0	21.0	1.0

- One model per gender!
- Keep original: common model + model for each gender to adjust.
- Product of multiple categoricals: common model + multiple models to adjust for combinations

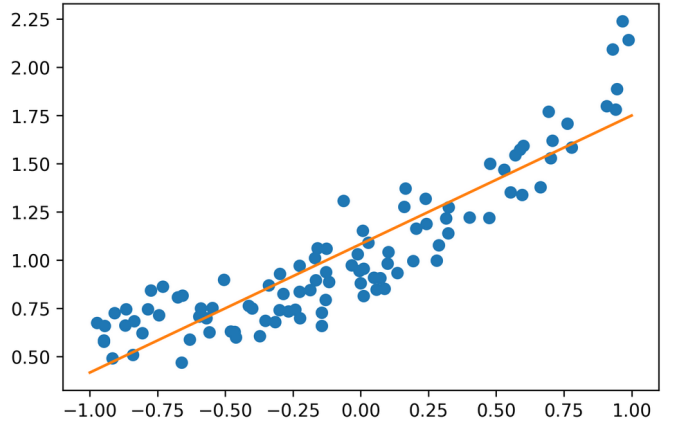
More interactions

```
age articles_bought gender spend$ time_online
 + Male * (age articles_bought spend$ time_online )
 + Female * (age articles_bought spend$ time_online )
 + (age > 20) * (age articles_bought gender spend$ time_online)
 + (age <= 20) * (age articles_bought gender spend$ time_online)
 + (age <= 20) * Male * (age articles_bought gender spend$ time_online)
```

Polynomial Features

```
from sklearn.linear_model import LinearRegression
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
lr = LinearRegression().fit(X_train, y_train)
line = np.linspace(-1, 1, 100).reshape(-1, 1)
plt.plot(x, y, 'o')
plt.plot(line, lr.predict(line))
lr.score(X_test, y_test)
```

```
0.763
```



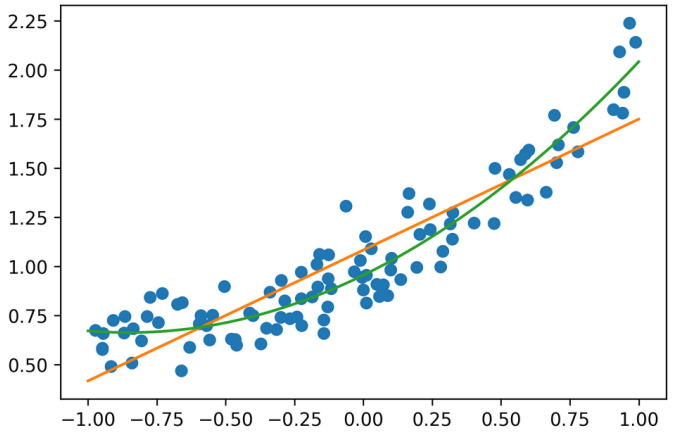
Polynomial Features

```
poly_lr = make_pipeline(PolynomialFeatures(include_bias=False), LinearRegression())

poly_lr.fit(X_train, y_train)

plt.plot(x, y, 'o')
plt.plot(line, lr.predict(line))
plt.plot(line, poly_lr.predict(line))
poly_lr.score(X_test, y_test)
```

```
0.833
```



Polynomial Features

- PolynomialFeatures() adds polynomials and interactions.
- Transformer interface like scalers etc.
- Create polynomial algorithms with make_pipeline!

Polynomial Features

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures()
X_bc_poly = poly.fit_transform(X_bc_scaled)
print(X_bc_scaled.shape)
print(X_bc_poly.shape)
```

```
(379, 13)
(379, 105)
```

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

```
(0.759, 0.081)
```

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

```
(0.865, 0.080)
```

Other Features?

- Plot the data, see if there are periodic patterns!