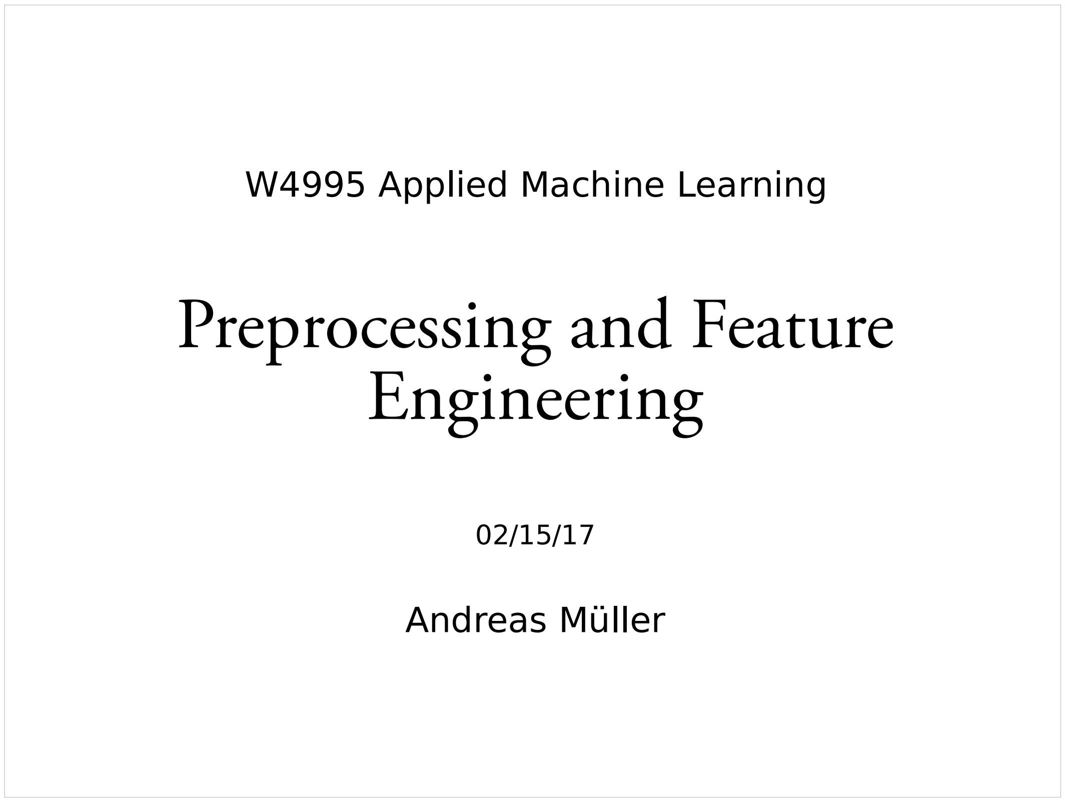


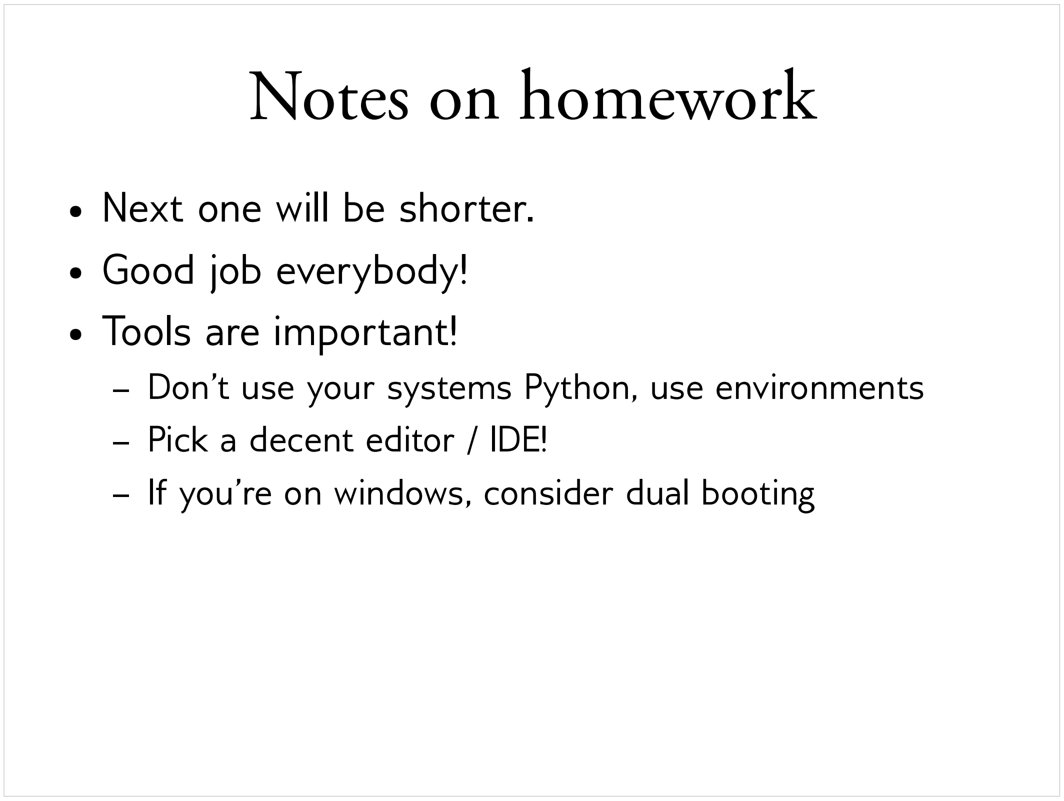
W4995 Applied Machine Learning

Preprocessing and Feature Engineering

02/15/17

Andreas Müller

Today we’ll talk about preprocessing and feature-engineering. What we’re talking about today mostly applies to linear models, and not to tree-based models, but it also applies to neural nets and kernel SVMs.



Notes on homework

- Next one will be shorter.
- Good job everybody!
- Tools are important!
  - Don't use your systems Python, use environments
  - Pick a decent editor / IDE!
  - If you're on windows, consider dual booting

I realized the homework was a bit much for some of you who haven't been familiar with git and testing.

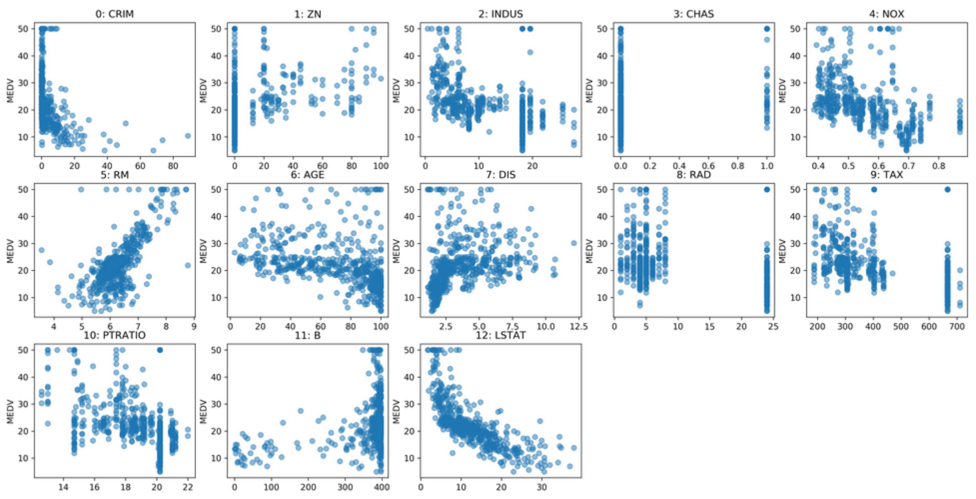
I saw many of you made really great progress and got everything together, so good on you. I promise the next homework will be shorter.

While this might have been much all at once, all the pieces are important. Who wants to go to industry? You'll fight these fights all the time!

I gave some advice at the second lecture, and I noticed not everybody took it. If you want be a serious data scientist in industry, start now. Less applicable to researchers (but also).

Don't use system python. Know which environment you're using. Pick an editor.

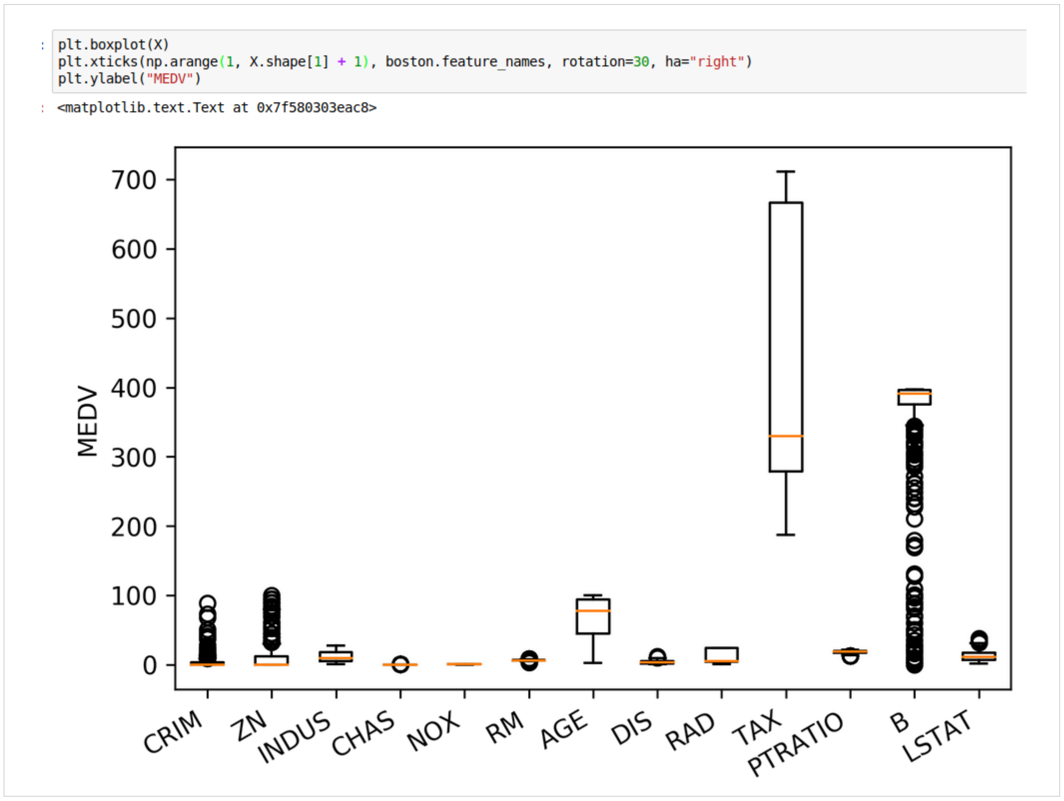
If you're on windows, get comfy with cygwin or better set up dual boot. You need to know POSIX!



Let’s go back to the boston housing dataset. The idea was to predict house prices. Here are the features on the x axis and the response, so price, on the y axis.

What are some thing you can notice? (concentrated distributions, skewed distibutions, discrete variable, linear and non-linear effects, different scales)

Scaling



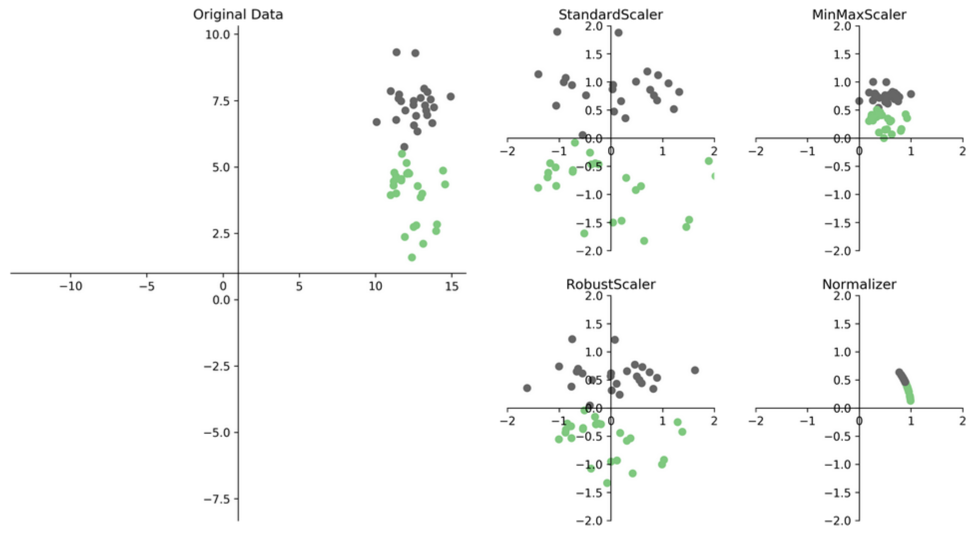
```
plt.boxplot(X)
plt.xticks(np.arange(1, X.shape[1] + 1), boston.feature_names, rotation=30, ha="right")
plt.ylabel("MEDV")
```

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

Let's start with the different scales.
Many model want data that is on the same scale.
  KNearestNeighbors: If the distance in TAX is between 300 and 400 then the distance difference in CHArS doesn't matter!
Linear models: the different scales mean different penalty. L2 is the same for all!

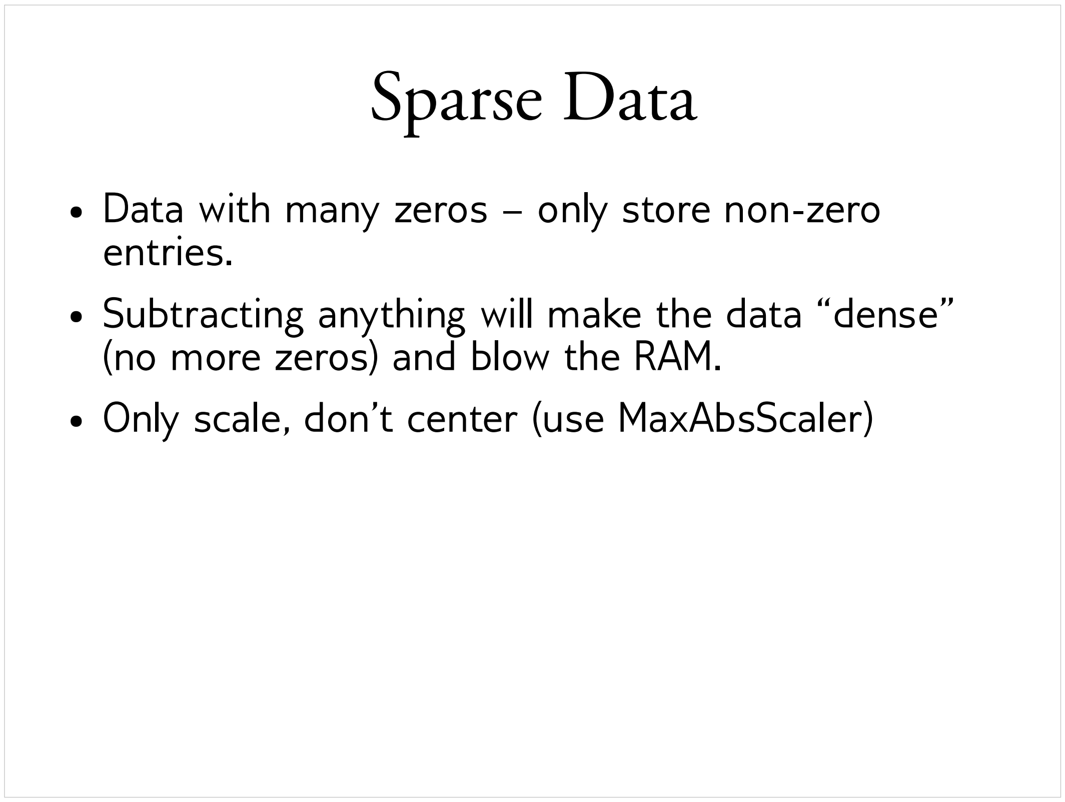
We can also see non-gaussian distributions here btw!

Ways to Scale Data



StandardScaler: subtract mean, divide by standard deviation.
MinMaxScaler: subtract minimum, divide by range. Afterwards between 0 and 1.
Robust Scaler: uses median and quantiles, therefore robust to outliers. Similar to StandardScaler.
Normalizer: only considers angle, not length. Helpful for histograms, not that often used.

StandardScaler is usually good, but doesn’t guarantee particular min and max values



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)

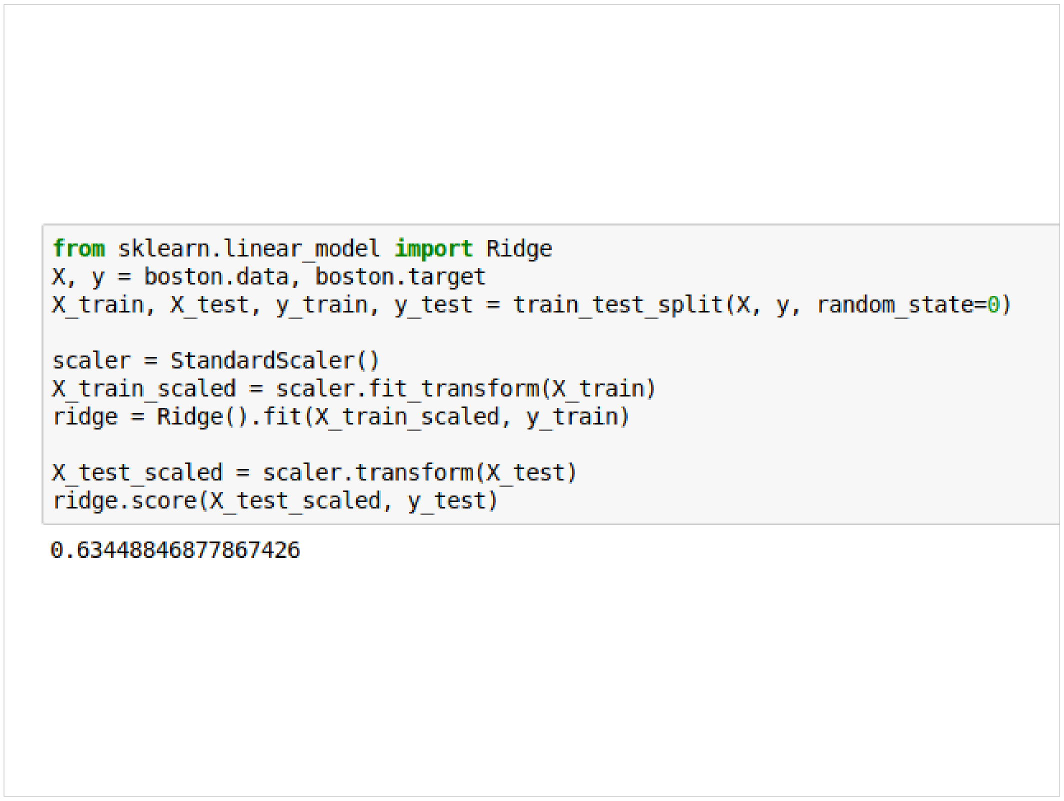
You have to be careful if you have sparse data. Sparse data is data where most entries of the data-matrix X are zero – often only 1% or less are not zero.

You can store this efficiently by only storing the non-zero elements.

Subtracting the mean results in all features becoming non-zero!

So don’t subtract anything, but you can still scale.

MaxAbsScaler scales between -1 and 1 by dividing with the maximum absolute value for each feature.



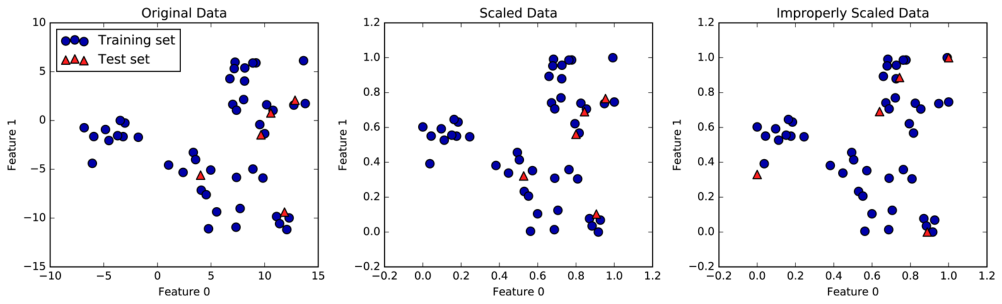
```
from sklearn.linear_model import Ridge
X, y = boston.data, boston.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

scaler = StandardScaler()
X_train_scaled = scaler.fit_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)
```

```
0.63448846877867426
```

Here’s how you do the scaling with StandardScaler in scikit-learn. Similar interface to models, but “transform” instead of “predict”. “transform” is always used when you want a new representation of the data.

Here I was lazy and did fit and transform in one.

Fit on training set, transform training set, fit ridge on scaled data, transform test data, score scaled test data.


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

Scikit-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 short cuts:

est.fit_transform(X) == est.fit(X).transform(X)
est.fit_predict(X) == est.fit(X).predict(X)

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

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

```
(0.71718655233314066, 0.12521148650633437)
```

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

```
(0.71789046947346136, 0.12695447250917097)
```

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

```
(0.49871865806668803, 0.14628381664585244)
```

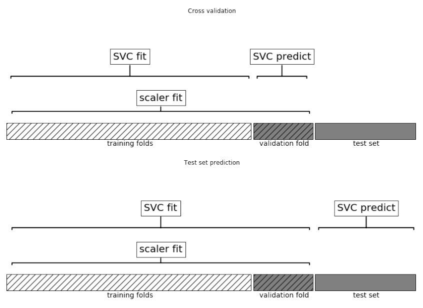
```python
from sklearn.neighbors import KNeighborsRegressor
scores = cross_val_score(KNeighborsRegressor(), X_train_scaled, y_train, cv=10)
np.mean(scores), np.std(scores)
```

```
(0.74979219238995831, 0.10573244928159024)
```

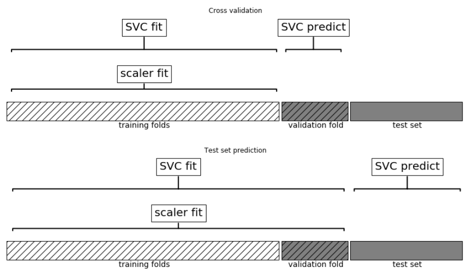
A note on preprocessing (and pipelines)

Leaking information

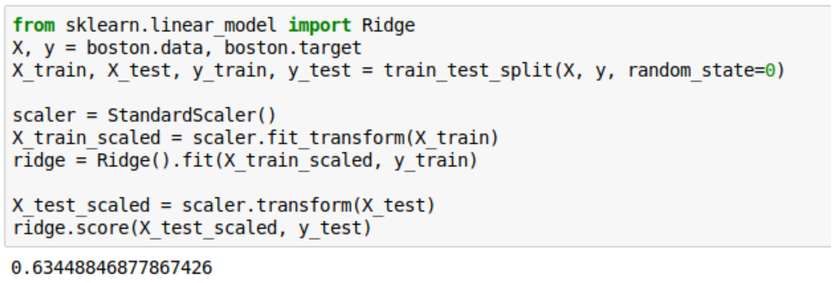
Information leak



No information leak

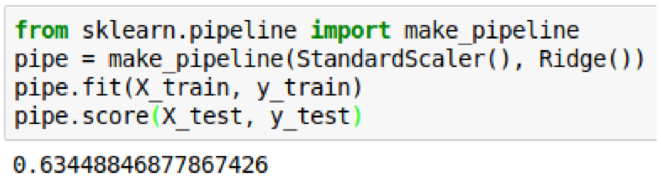


Need to include preprocessing in cross-validation!



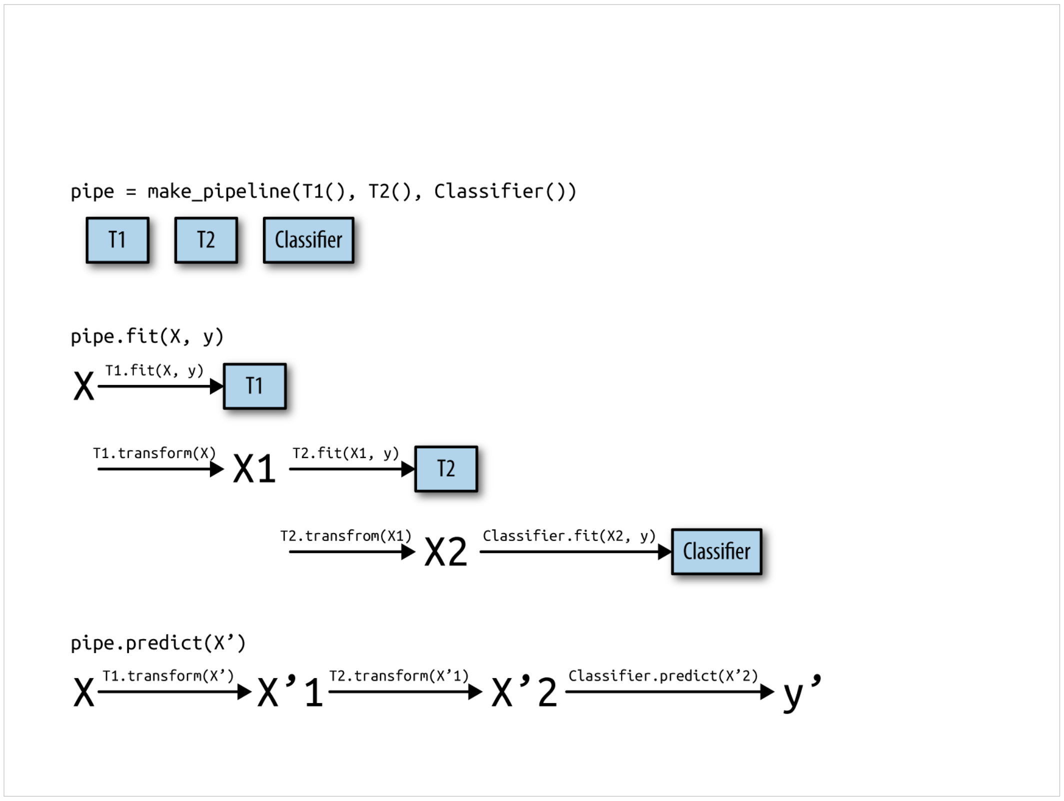
```python
from sklearn.linear_model import Ridge
X, y = boston.data, boston.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

scaler = StandardScaler()
X_train_scaled = scaler.fit_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)
```

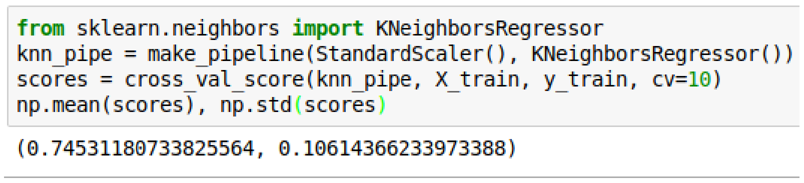
```
0.63448846877867426
```



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

```
0.63448846877867426
```

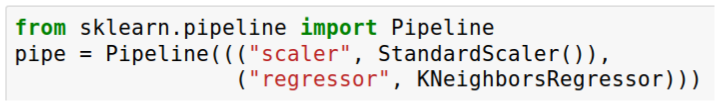

pipe = make_pipeline(T1(), T2(), Classifier())
T1
T2
Classifier
pipe.fit(X, y)
X
T1.fit(X, y)
T1
T1.transform(X)
X1
T2.fit(X1, y)
T2
T2.transfrom(X1)
X2
Classifier.fit(X2, y)
Classifier
pipe.predict(X’)
X
T1.transform(X’)
X’1
T2.transform(X’1)
X’2
Classifier.predict(X’2)
y’



```python
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)
```

```
(0.74531180733825564, 0.10614366233973388)
```

```
print(knn_pipe.steps)

[('standardscaler', StandardScaler(copy=True, with_mean=True, with_std=True)),
 ('kneighborsregressor', KNeighborsRegressor(algorithm='auto', leaf_size=30,
metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=5, p=2,
weights='uniform'))]
```



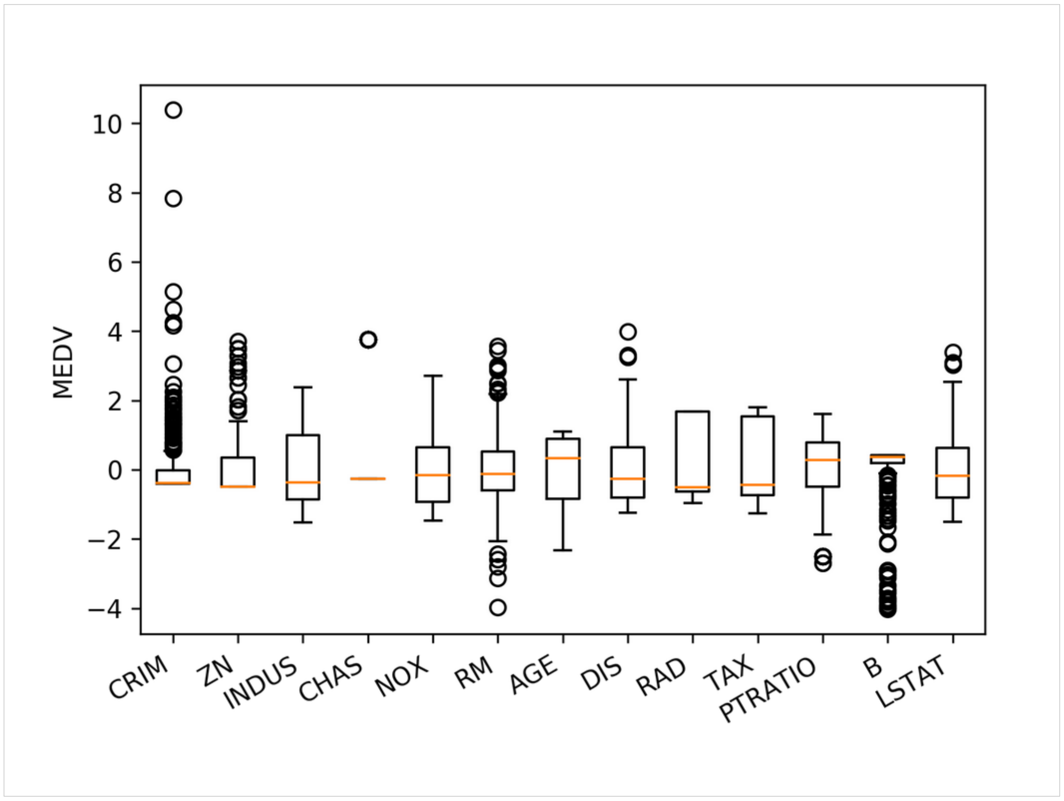
```python
from sklearn.pipeline import Pipeline
pipe = Pipeline((("scaler", StandardScaler()),
                 ("regressor", KNeighborsRegressor())))
```

Pipeline and GridSearchCV

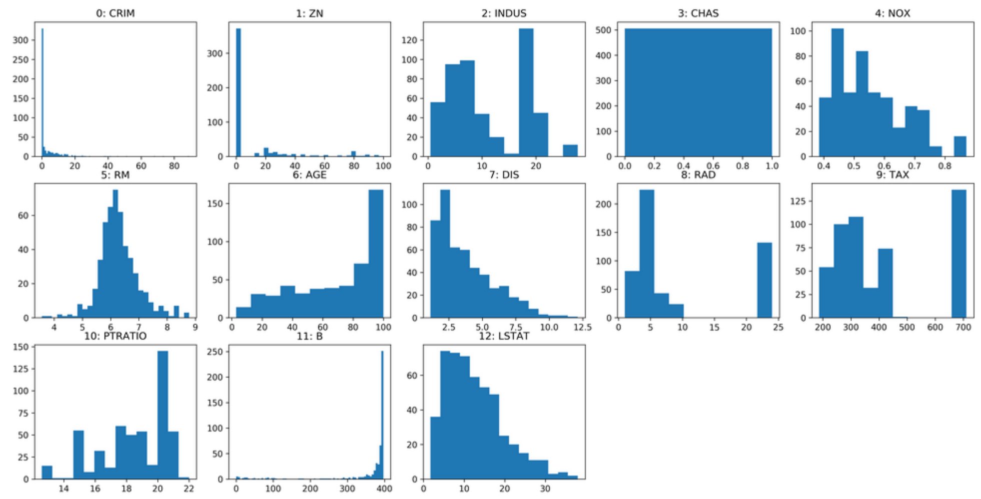
```
from sklearn.model_selection import GridSearchCV

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}
0.600015753391
```

Feature Distributions


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

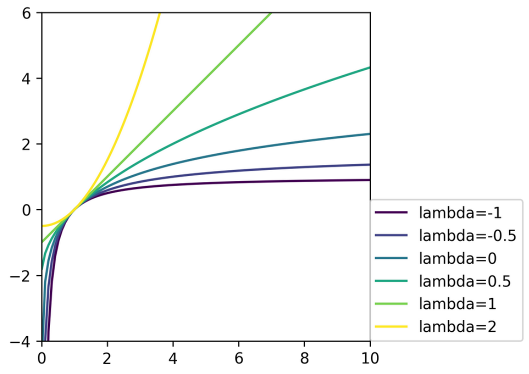
```
fig, axes = plt.subplots(3, 5, figsize=(20, 10))
for i, ax in enumerate(axes.ravel()):
    if i > 12:
        ax.set_visible(False)
        continue
    ax.hist(X[:, i], bins="auto")
    ax.set_title("{}: {}".format(i, boston.feature_names[i]))
```

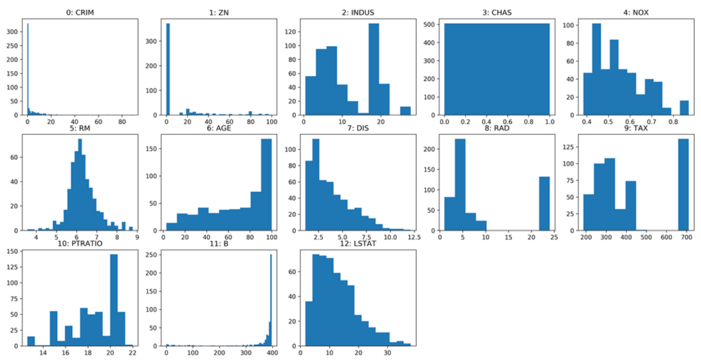


Box-Cox Transform

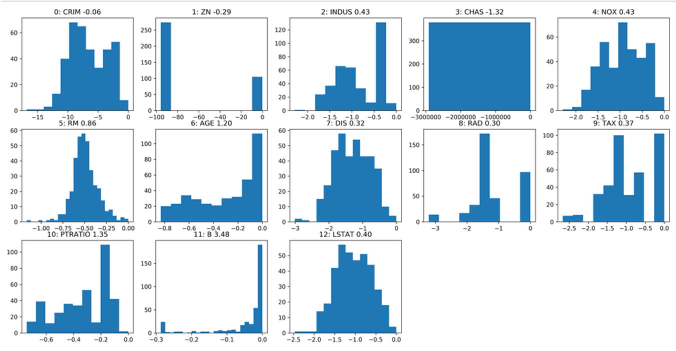
$$bc_\lambda(x) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log(x) & \text{if } \lambda = 0 \end{cases}$$

Only applicable for positive x!

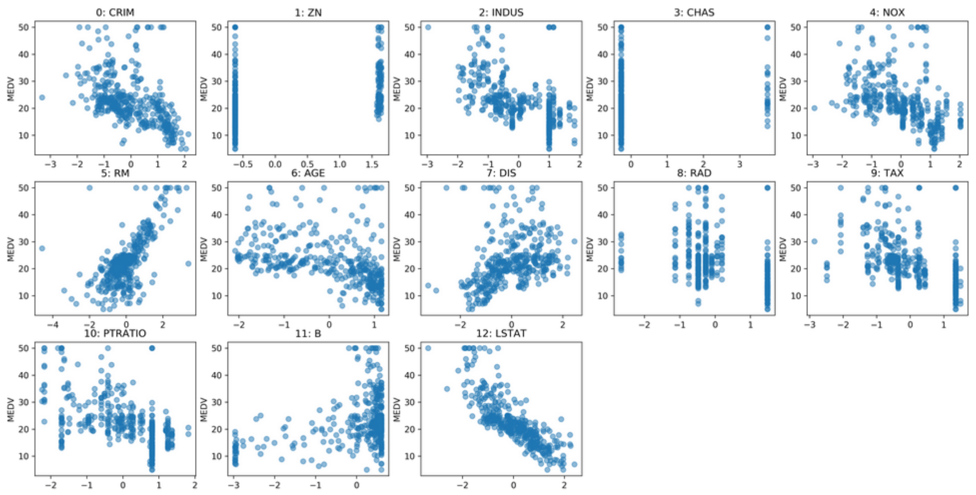




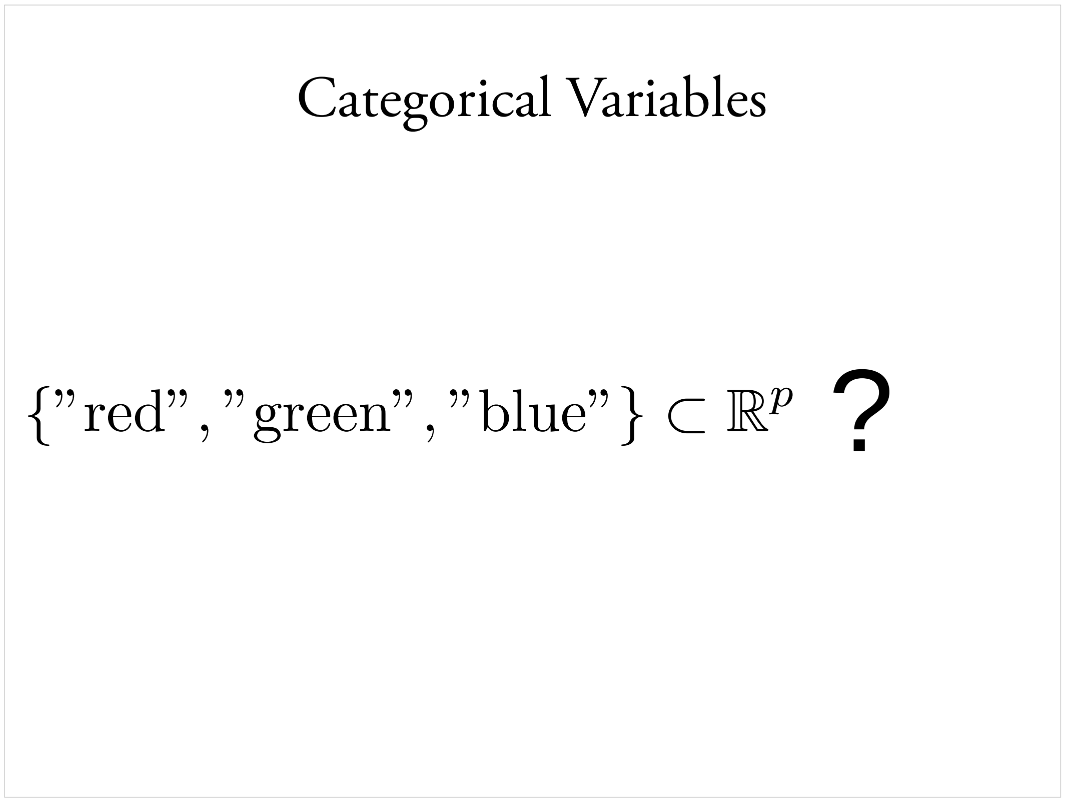
Before



After


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

Discrete Features

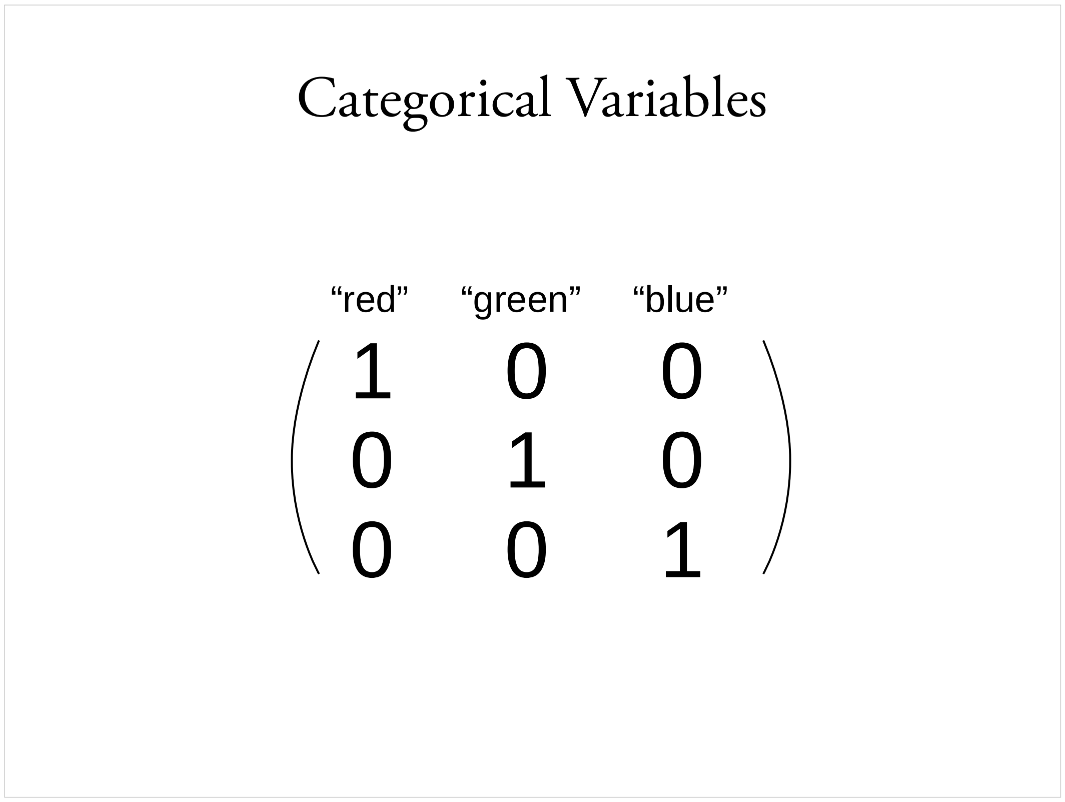


Categorical Variables

$$\{\text{"red"}, \text{"green"}, \text{"blue"}\} \subset \mathbb{R}^p \quad ?$$

Before we can apply a machine learning algorithm, we first need to think about how we represent our data.
Earlier, I said x \in R^n. That's not how you usually get data. Often data has units, possibly different units for different sensors, it has a mixture of continuous values and discrete values, and different measurements might be on totally different scales.

First, let me explain how to deal with discrete input variables, also known as categorical features. They come up in nearly all applications.
Let's say you have three possible values for a given measurement, whether you used setup1 setup2 or setup3. You could try to encode these into a single real number, say 0, 1 and 2, or e, \pi, \tau.
However, that would be a bad idea for algorithms like linear regression.



Categorical Variables

$$\begin{array}{ccc} \text{“red”} & \text{“green”} & \text{“blue”} \\ \hline 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}$$

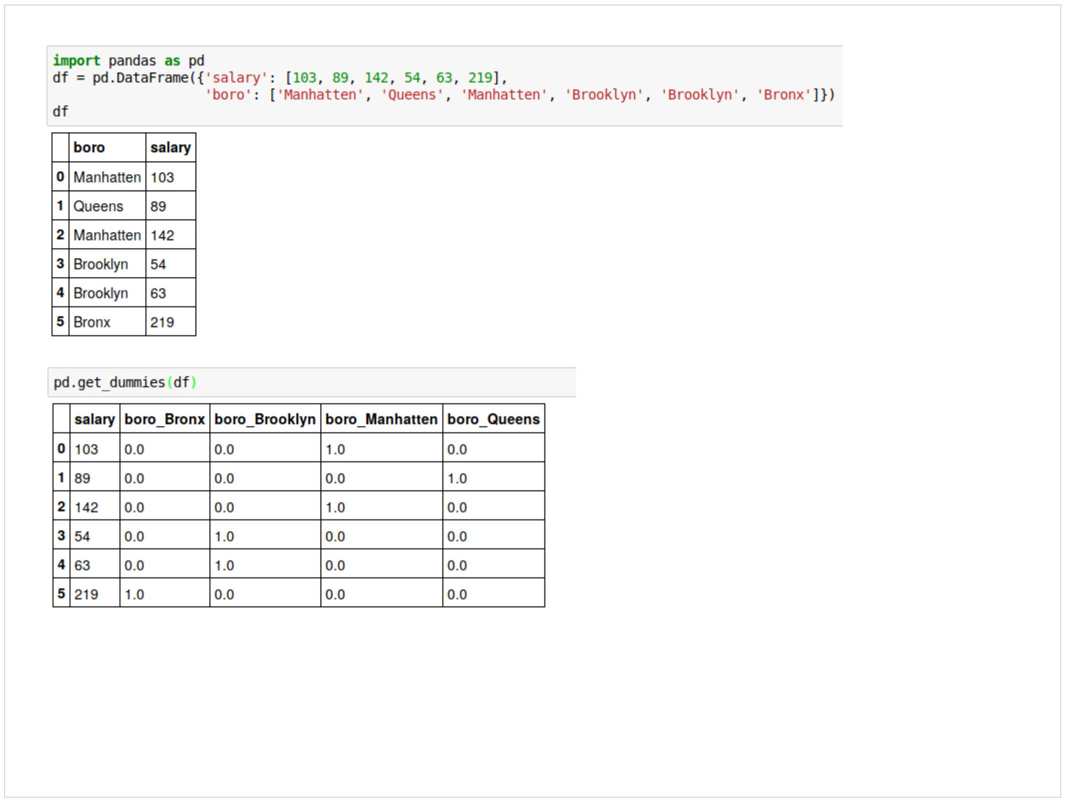
If you encode all three values using the same feature, then you are imposing a linear relation between them, and in particular you define an order between the categories. Usually, there is no semantic ordering of the categories, and so we shouldn’t introduce one in our representation of the data.

Instead, we add one new feature for each category,

And that feature encodes whether a sample belongs to this category or not.

That’s called a one-hot encoding, because only one of the three features in this example is active at a time.

You could actually get away with n-1 features, but in machine learning that usually doesn’t matter.

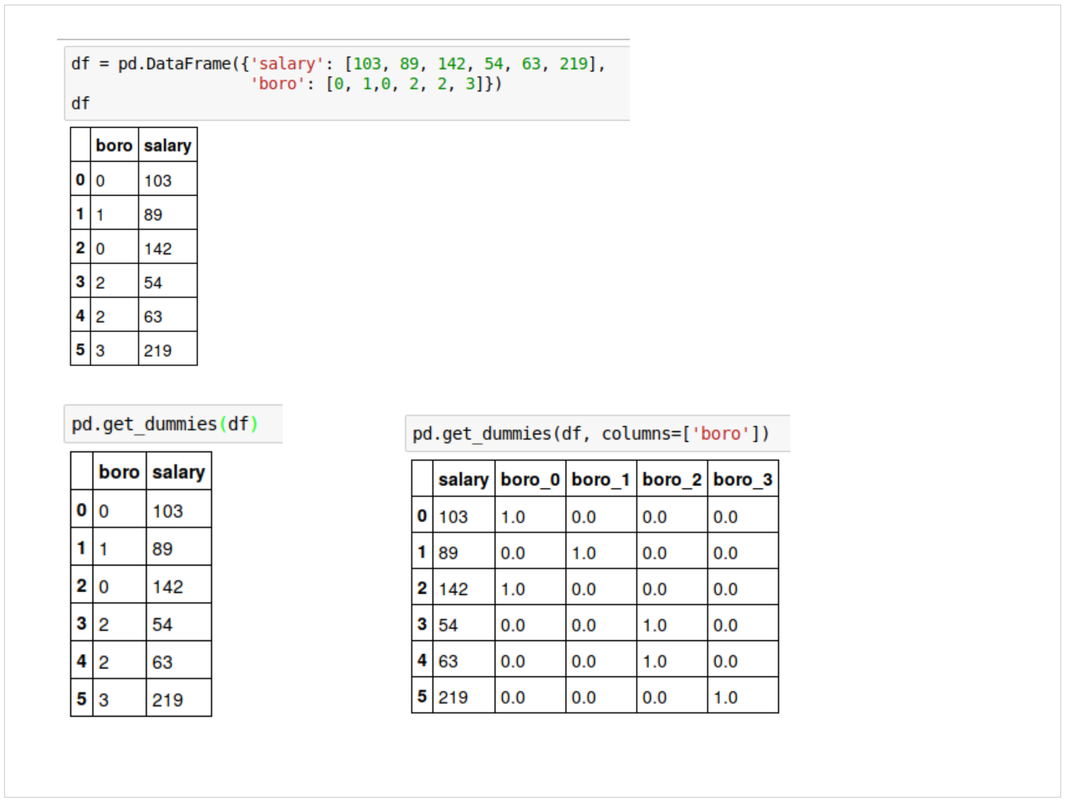


```python
import pandas as pd
df = pd.DataFrame({'salary': [103, 89, 142, 54, 63, 219],
                   'boro': ['Manhatten', 'Queens', 'Manhatten', 'Brooklyn', 'Brooklyn', 'Bronx']})
df
```

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

```python
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



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

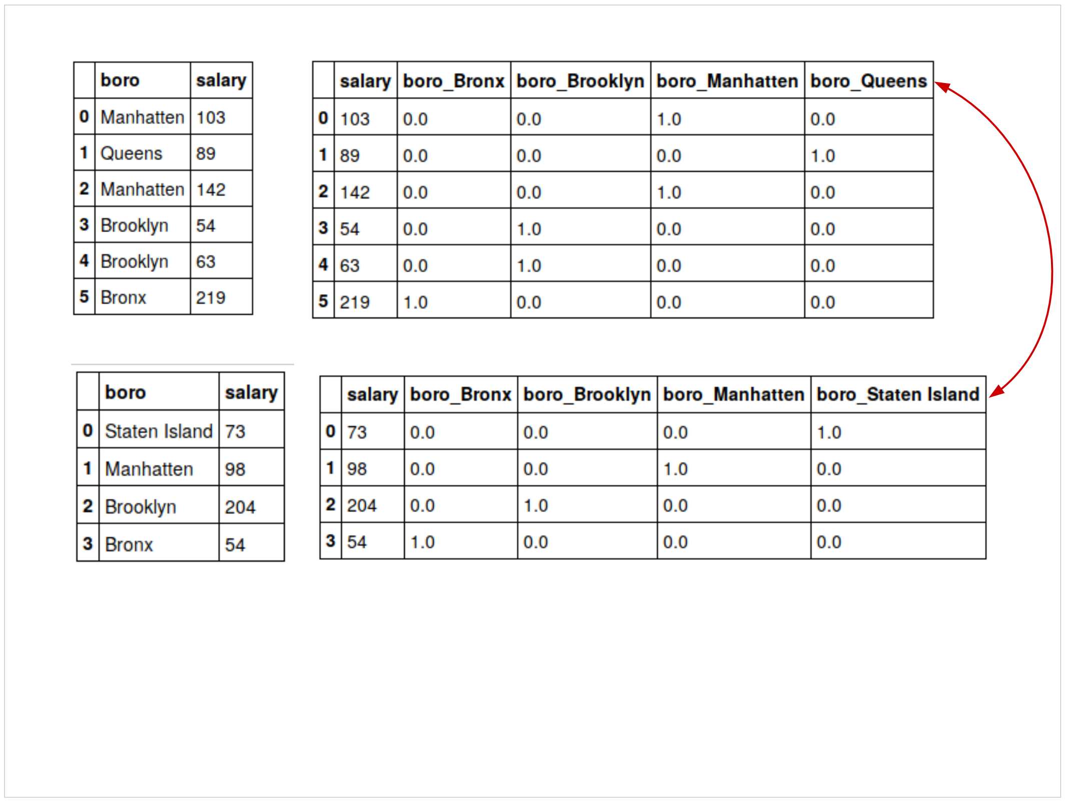
	boro	salary
0	0	103
1	1	89
2	0	142
3	2	54
4	2	63
5	3	219

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

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

```python
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



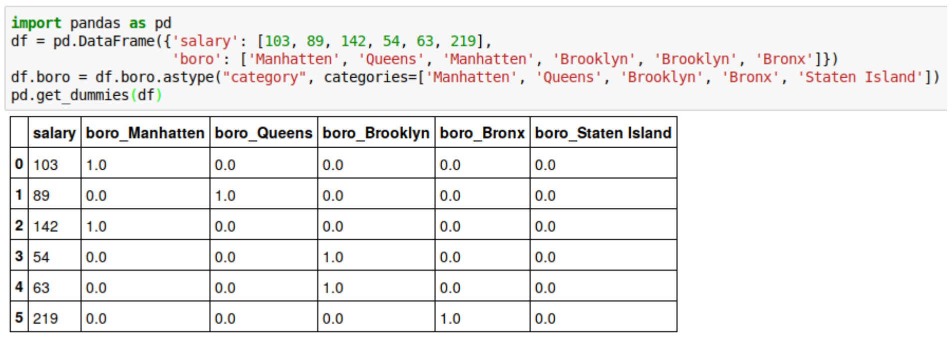
	boro	salary
0	Manhatten	103
1	Queens	89
2	Manhatten	142
3	Brooklyn	54
4	Brooklyn	63
5	Bronx	219

	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

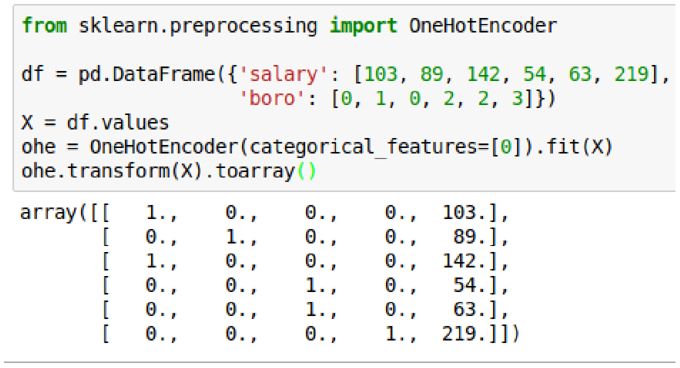
Pandas Categorical Columns



```python
import pandas as pd
df = pd.DataFrame({'salary': [103, 89, 142, 54, 63, 219],
                   'boro': ['Manhatten', 'Queens', 'Manhatten', 'Brooklyn', 'Brooklyn', 'Bronx']})
df.boro = df.boro.astype("category", categories=['Manhatten', 'Queens', 'Brooklyn', 'Bronx', 'Staten Island'])
pd.get_dummies(df)
```

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

OneHotEncoder



```
from sklearn.preprocessing import OneHotEncoder

df = pd.DataFrame({'salary': [103, 89, 142, 54, 63, 219],
                   'boro': [0, 1, 0, 2, 2, 3]})
X = df.values
ohe = OneHotEncoder(categorical_features=[0]).fit(X)
ohe.transform(X).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.]])
```

- Fit-transform paradigm ensures train and test-set categories correspond.
- Only works for integers right now, not strings (we're fixing this).

One-Hot vs statisticians

- 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
- In scikit-learn:
  - None
- In scikit-learn soon:
  - Decision trees, random forests

Count-Based Encoding

- For high cardinality categorical features
- Example: US states, given low samples

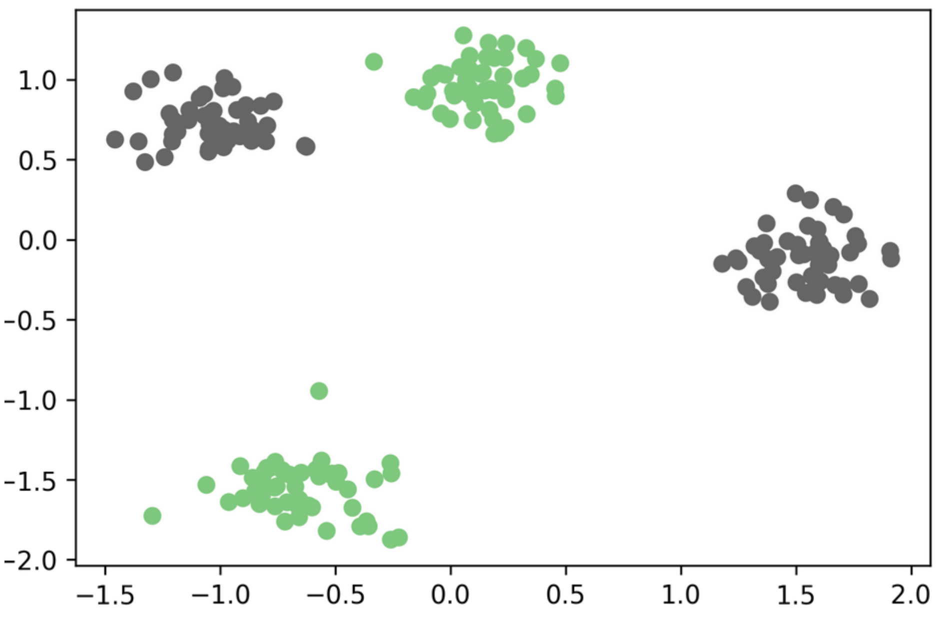
- Instead of 50 one-hot variables, one “response encoded” variable.
- For regression:
  “people in this state have an average response of y”
- Binary classification:
  - “people in this state have likelihood p for class 1”
- Multiclass:
  - One feature per class: probability distribution

Example

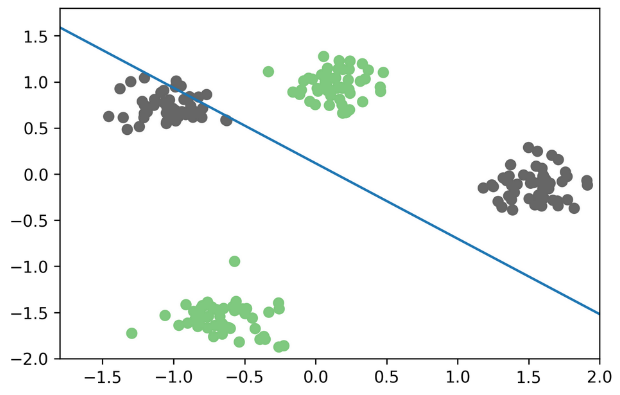
- FIXME

Feature Engineering

Interaction Features



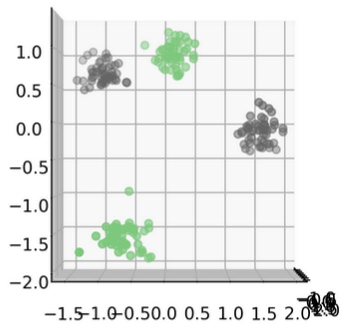
Interaction Features

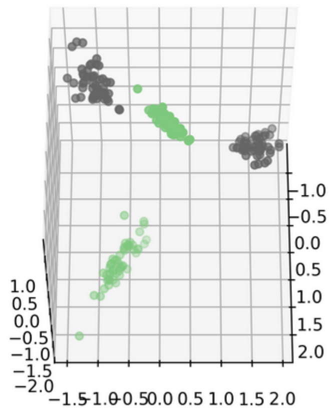


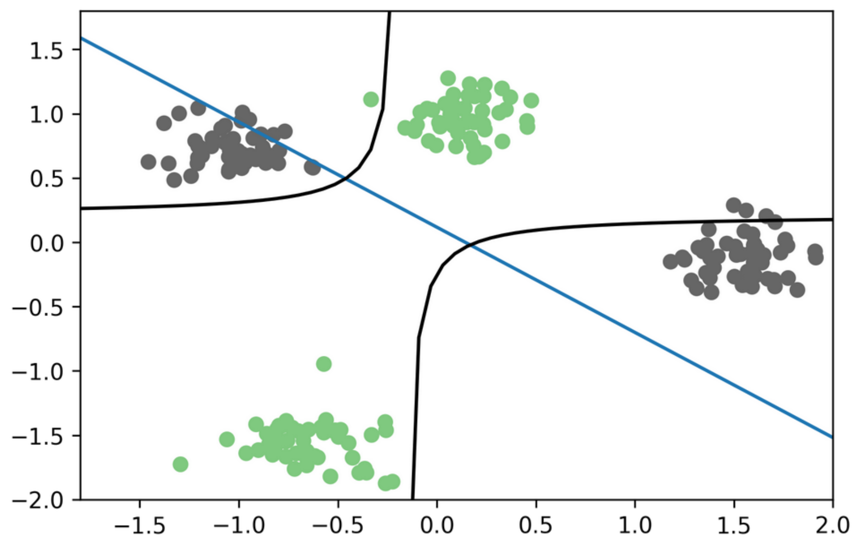
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
logreg = LogisticRegressionCV().fit(X_train, y_train)
logreg.score(X_test, y_test)
```

```
0.5
```

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

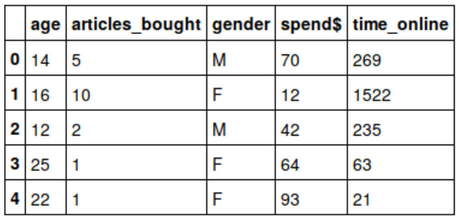




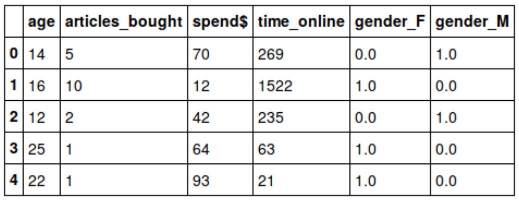


```
: X_i_train, X_i_test, y_train, y_test = train_test_split(X_interaction, y, random_state=0)
  logreg3 = LogisticRegressionCV().fit(X_i_train, y_train)
  logreg3.score(X_i_test, y_test)
```

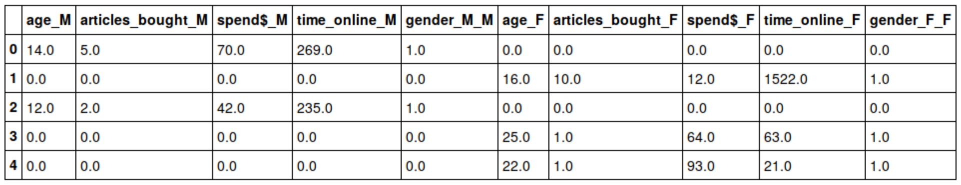
```
: 0.95999999999999996
```



	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

age articles_bought gender spend$ time_online   + Male * (age articles_bought spend$ time_online)

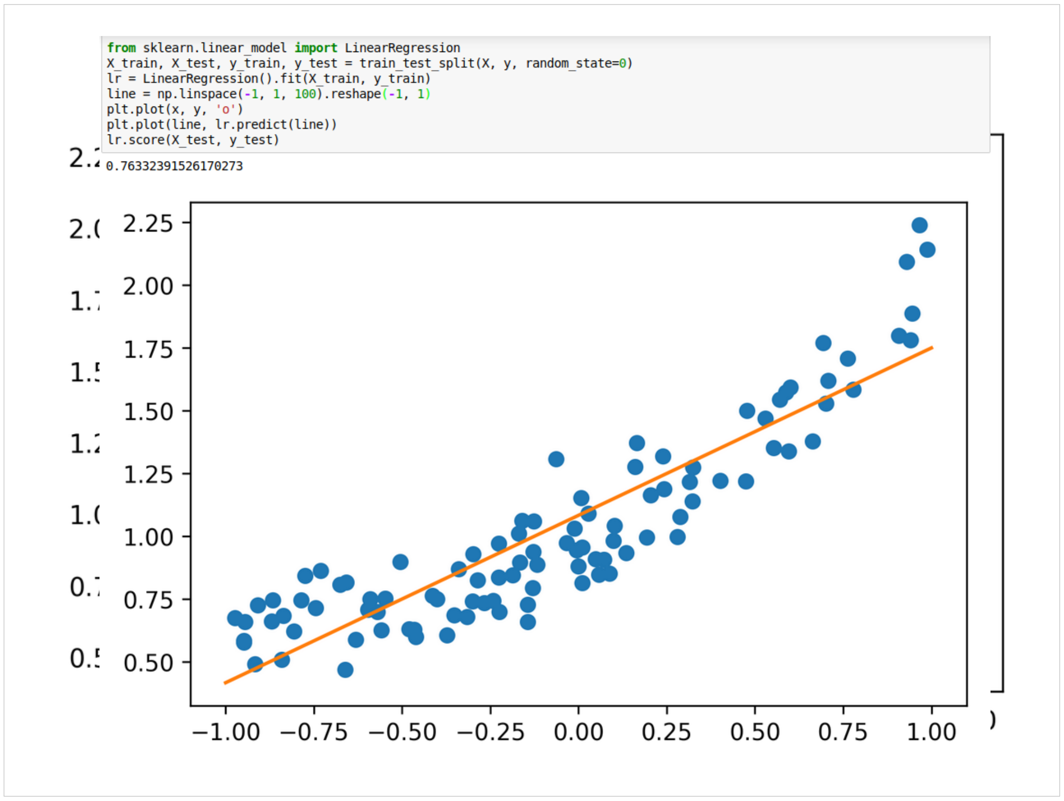
+ Female * (age articles_bought spend$ time_online)

+ (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)

….



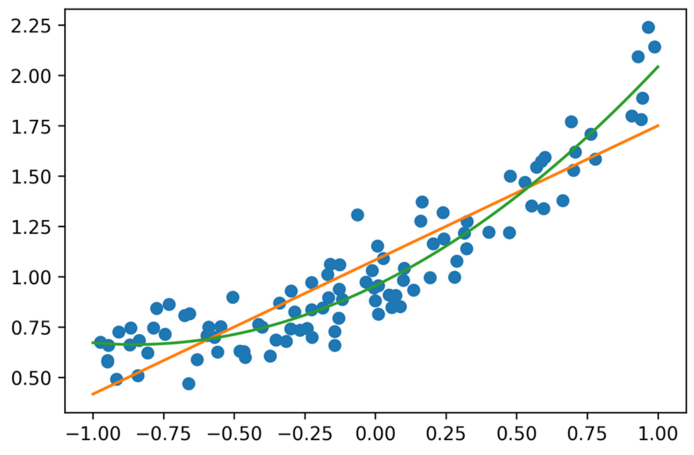
```python
from sklearn.linear_model import LinearRegression
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
lr = LinearRegression().fit(X_train, y_train)
line = np.linspace(-1, 1, 100).reshape(-1, 1)
plt.plot(x, y, 'o')
plt.plot(line, lr.predict(line))
lr.score(X_test, y_test)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

```
0.76332391526170273
```

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)
```

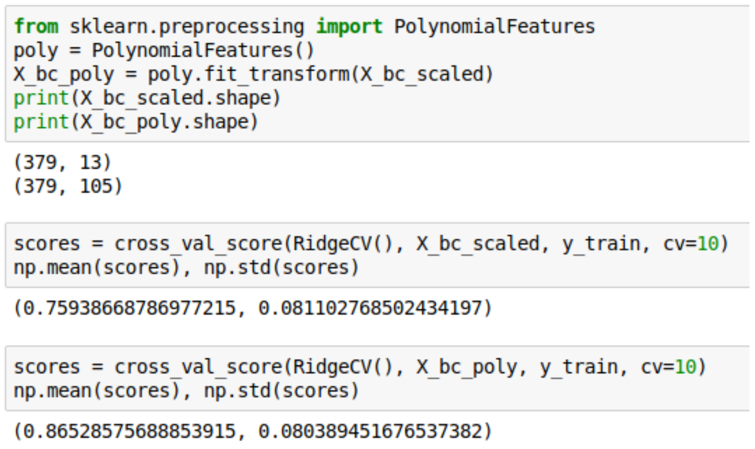
```
: 0.83367862697542183
```



Polynomial Features

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

Polynomial Features



```python
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)
```

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

```
(0.75938668786977215, 0.081102768502434197)
```

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

```
(0.86528575688853915, 0.080389451676537382)
```

Other features?

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

Discretization and Binning

- Loses data.
- Target-independent might be bad
- Powerful combined with interactions to create new features!