W4995 Applied Machine Learning

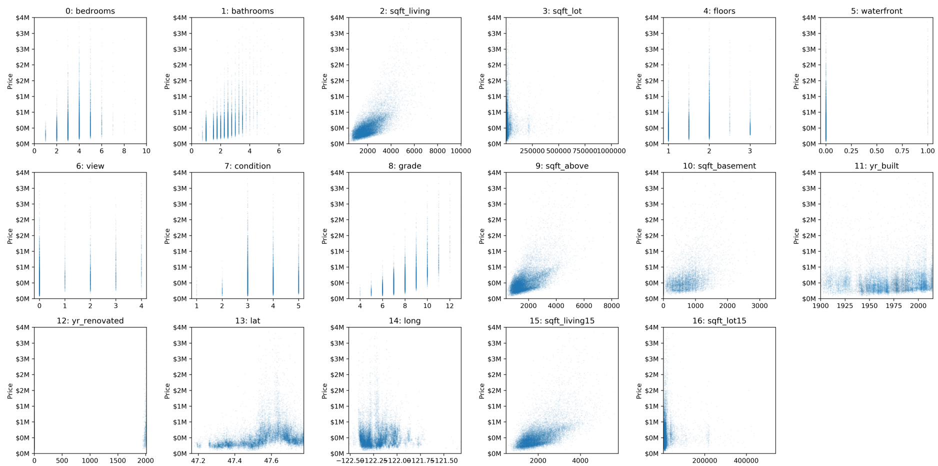
Preprocessing and Feature Transformations

02/05/20

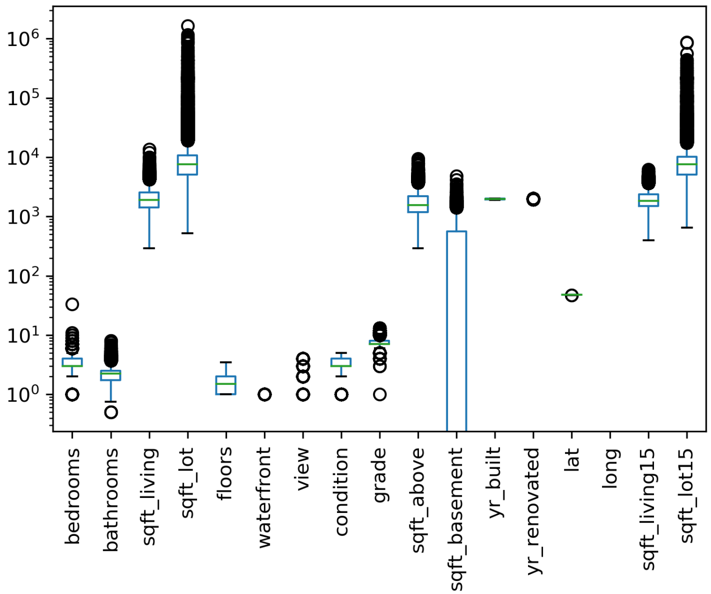
Andreas C. Müller

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

Andrew Ng

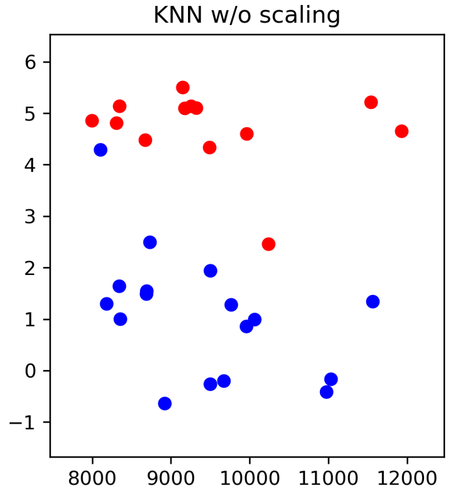

0: bedrooms
1: bathrooms
2: sqft_living
3: sqft_lot
4: floors
5: waterfront
6: view
7: condition
8: grade
9: sqft_above
10: sqft_basement
11: yr_built
12: yr_renovated
13: lat
14: long
15: sqft_living15
16: sqft_lot15
Price

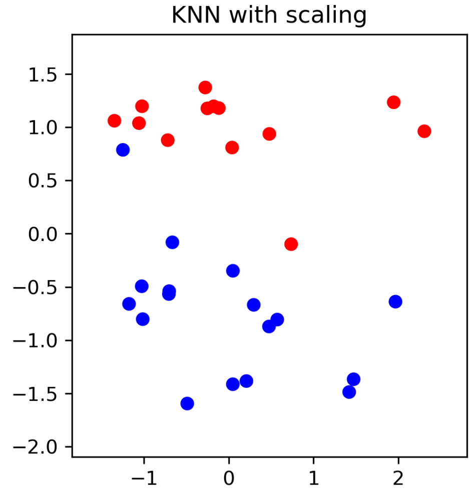
Scaling



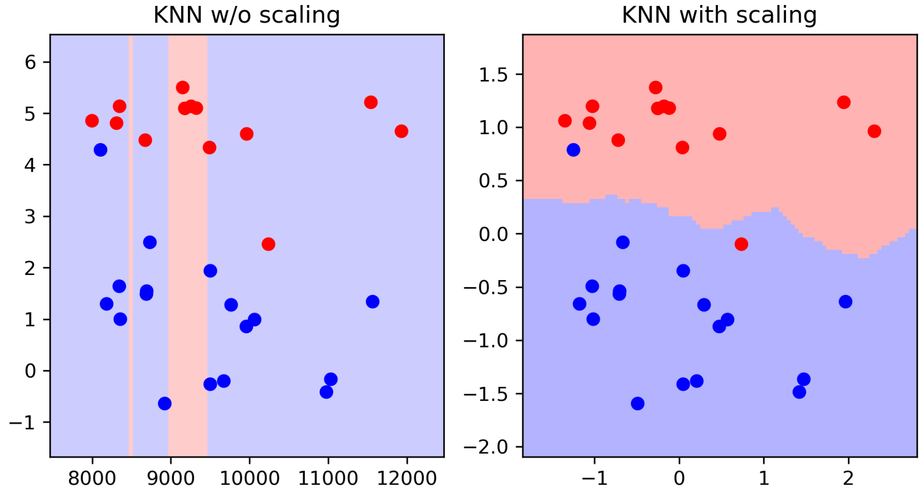
10^6
10^5
10^4
10^3
10^2
10^1
10^0
bedrooms
bathrooms
sqft_living
sqft_lot
floors
waterfront
view
condition
grade
sqft_above
sqft_basement
yr_built
yr_renovated
lat
long
sqft_living15
sqft_lot15

Scaling and Distances

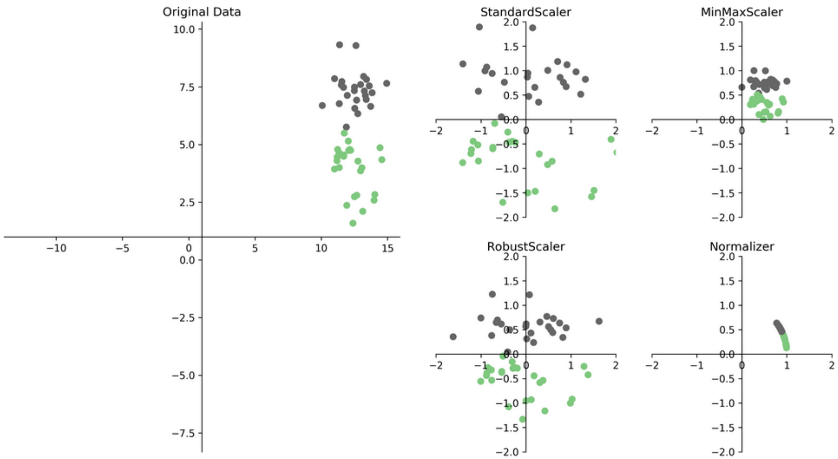




Scaling and Distances



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)

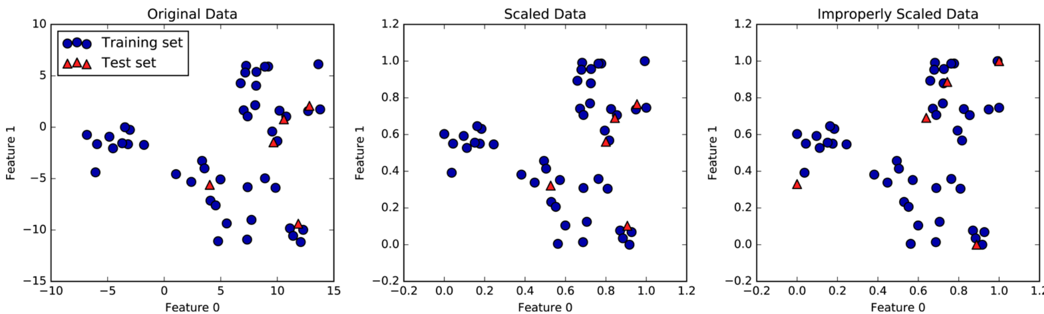
Standard Scaler Example

```python
from sklearn.linear_model import Ridge
# Back to King Country house prices
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)
```

```
0.684
```

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.~~fit_~~transform(X_test)



Sckit-Learn API Summary

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

`estimator.predict`	`estimator.transform`
Classification	Preprocessing
Regression	Dimensionality reduction
Clustering	Feature selection
	Feature extraction

```python
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)
```

```
(0.694, 0.027)
```

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

```
(0.694, 0.027)
```

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

```
(0.500, 0.039)
```

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

```
(0.786, 0.030)
```

A note on preprocessing (and pipelines)

A common errror

```
print(X.shape)
```

```
(100, 10000)
```

```
# select most informative 5% of features
from sklearn.feature_selection import SelectPercentile, f_regression
select = SelectPercentile(score_func=f_regression, percentile=5)
select.fit(X, y)
X_selected = select.transform(X)
print(X_selected.shape)
```

```
(100, 500)
```

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Ridge
np.mean(cross_val_score(Ridge(), X_selected, y))
```

```
0.90
```

```
ridge = Ridge().fit(X_selected, y)
X_test_selected = select.transform(X_test)
ridge.score(X_test_selected, y_test)
```

```
-0.18
```

Leaking Information

```python
# BAD!
select.fit(X, y)  # includes the cv test parts!
X_sel = select.transform(X)
scores = []
for train, test in cv.split(X, y):
    ridge = Ridge().fit(X_sel[train], y[train])
    score = ridge.score(X_sel[test], y[test])
    scores.append(score)
```

```python
# GOOD!
scores = []
for train, test in cv.split(X, y):
    select.fit(X[train], y[train])
    X_sel_train = select.transform(X[train])
    ridge = Ridge().fit(X_sel_train, y[train])
    X_sel_test = select.transform(X[test])
    score = ridge.score(X_sel_test, y[test])
    scores.append(score)
```

Need to include preprocessing in cross-validation !

```python
# Housing data example
from sklearn.linear_model import Ridge
X, y = df, target

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)
```

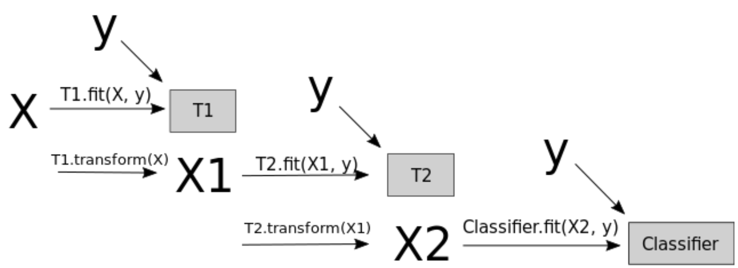
```
0.684
```

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

```
0.684
```

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

pipe.fit(X, y)



pipe.predict(X')

$$X' \xrightarrow{\text{T1.transform(X')}} X'1 \xrightarrow{\text{T2.transform(X'1)}} X'2 \xrightarrow{\text{Classifier.predict(X'2)}} y'$$

Undoing our feature selection mistake

```
# BAD!
select.fit(X, y)  # includes the cv test parts!
X_sel = select.transform(X)
scores = []
for train, test in cv.split(X, y):
    ridge = Ridge().fit(X_sel[train], y[train])
    score = ridge.score(X_sel[test], y[test])
    scores.append(score)
```

Same as:

```
select.fit(X, y)
X_selected = select.transform(X, y)
np.mean(cross_val_score(Ridge(), X_selected, y))
```

```
0.90
```

```
# GOOD!
scores = []
for train, test in cv.split(X, y):
    select.fit(X[train], y[train])
    X_sel_train = select.transform(X[train])
    ridge = Ridge().fit(X_sel_train, y[train])
    X_sel_test = select.transform(X[test])
    score = ridge.score(X_sel_test, y[test])
    scores.append(score)
```

Same as:

```
pipe = make_pipeline(select, Ridge())
np.mean(cross_val_score(pipe, X, y))
```

```
-0.079
```

Naming Steps

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

```
[('standardscaler', StandardScaler()),
 ('kneighborsregressor', KNeighborsRegressor())]
```

```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}
0.60
```

Going wild with Pipelines

```
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
X_train, X_test, y_train, y_test = train_test_split(
    diabetes.data, diabetes.target, random_state=0)

from sklearn.preprocessing import PolynomialFeatures
pipe = make_pipeline(
    StandardScaler(),
    PolynomialFeatures(),
    Ridge())

param_grid = {'polynomialfeatures__degree': [1, 2, 3],
              'ridge__alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
grid = GridSearchCV(pipe, param_grid=param_grid,
                    n_jobs=-1, return_train_score=True)
grid.fit(X_train, y_train)
```

Going wilder with Pipelines

```
pipe = Pipeline([('scaler', StandardScaler()),
                 ('regressor', Ridge())])

param_grid = {'scaler': [StandardScaler(), MinMaxScaler(),
                         'passthrough'],
              'regressor': [Ridge(), Lasso()],
              'regressor__alpha': np.logspace(-3, 3, 7)}


grid = GridSearchCV(pipe, param_grid)
grid.fit(X_train, y_train)
grid.score(X_test, y_test)
```

Going wildest with Pipelines

```
from sklearn.tree import DecisionTreeRegressor
pipe = Pipeline([('scaler', StandardScaler()),
                 ('regressor', Ridge())])

# check out searchgrid for more convenience
param_grid = [{'regressor': [DecisionTreeRegressor()],
               'regressor__max_depth': [2, 3, 4],
               'scaler': ['passthrough']},
              {'regressor': [Ridge()],
               'regressor__alpha': [0.1, 1],
               'scaler': [StandardScaler(), MinMaxScaler(),
                          'passthrough']}
             ]
grid = GridSearchCV(pipe, param_grid)
grid.fit(X_train, y_train)
grid.score(X_test, y_test)
```

Categorical Variables

Categorical Variables

```
import pandas as pd
df = pd.DataFrame({
  'boro': ['Manhattan', 'Queens', 'Manhattan', 'Brooklyn', 'Brooklyn', 'Bronx'],
  'salary': [103, 89, 142, 54, 63, 219],
     'vegan': ['No', 'No','No','Yes', 'Yes', 'No']})
```

	boro	salary	vegan
0	Manhattan	103	No
1	Queens	89	No
2	Manhattan	142	No
3	Brooklyn	54	Yes
4	Brooklyn	63	Yes
5	Bronx	219	No

Ordinal encoding

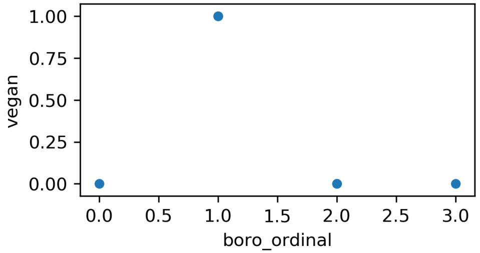
```
df['boro_ordinal'] = df.boro.astype("category").cat.codes
df
```

	boro	salary	vegan
0	2	103	No
1	3	89	No
2	2	142	No
3	1	54	Yes
4	1	63	Yes
5	0	219	No

Ordinal encoding

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

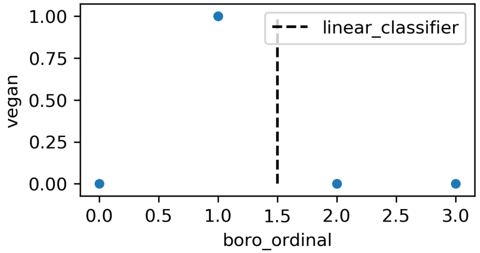
	boro	salary	vegan
0	2	103	No
1	3	89	No
2	2	142	No
3	1	54	Yes
4	1	63	Yes
5	0	219	No



Ordinal encoding

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

	boro_ordinal	salary	vegan
0	2	103	No
1	3	89	No
2	2	142	No
3	1	54	Yes
4	1	63	Yes
5	0	219	No



One-Hot (Dummy) Encoding

	boro	salary	vegan
0	Manhattan	103	No
1	Queens	89	No
2	Manhattan	142	No
3	Brooklyn	54	Yes
4	Brooklyn	63	Yes
5	Bronx	219	No

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

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

One-Hot (Dummy) Encoding

	boro	salary	vegan
0	Manhattan	103	No
1	Queens	89	No
2	Manhattan	142	No
3	Brooklyn	54	Yes
4	Brooklyn	63	Yes
5	Bronx	219	No

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

	salary	vegan	boro_Bronx	boro_Brooklyn	boro_Manhattan	boro_Queens
0	103	No	0	0	1	0
1	89	No	0	0	0	1
2	142	No	0	0	1	0
3	54	Yes	0	1	0	0
4	63	Yes	0	1	0	0
5	219	No	1	0	0	0

One-Hot (Dummy) Encoding

	boro	salary	vegan
0	2	103	No
1	3	89	No
2	2	142	No
3	1	54	Yes
4	1	63	Yes
5	0	219	No

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

	salary	vegan	boro_0	boro_1	boro_2	boro_3
0	103	No	0	0	1	0
1	89	No	0	0	0	1
2	142	No	0	0	1	0
3	54	Yes	0	1	0	0
4	63	Yes	0	1	0	0
5	219	No	1	0	0	0

```
df = pd.DataFrame({
                   'boro': ['Manhattan', 'Queens', 'Manhattan'
           'Brooklyn', 'Brooklyn', 'Bronx'],
  'salary': [103, 89, 142, 54, 63, 219],
  'vegan': ['No', 'No','No','Yes', 'Yes', 'No']})
df_dummies = pd.get_dummies(df, columns=['boro']
```

	salary	vegan	boro_Bronx	boro_Brooklyn	boro_Manhattan	boro_Queens
0	103	No	0	0	1	0
1	89	No	0	0	0	1
2	142	No	0	0	1	0
3	54	Yes	0	1	0	0
4	63	Yes	0	1	0	0
5	219	No	1	0	0	0

```
df = pd.DataFrame({
  'boro': ['Brooklyn', 'Manhattan', 'Brooklyn',
           'Queens', 'Brooklyn', 'Staten Island'],
  'salary': [61, 146, 142, 212, 98, 47],
  'vegan': ['Yes', 'No','Yes','No', 'Yes', 'No']})
df_dummies = pd.get_dummies(df, columns=['boro'])
```

	salary	vegan	boro_Brooklyn	boro_Manhattan	boro_Queens	boro_Staten Island
0	61	Yes	1	0	0	0
1	146	No	0	1	0	0
2	142	Yes	1	0	0	0
3	212	No	0	0	1	0
4	98	Yes	1	0	0	0
5	47	No	0	0	0	1

Pandas Categorical Columns

```
df = pd.DataFrame({
   'boro': ['Manhattan', 'Queens', 'Manhattan', 'Brooklyn', 'Brooklyn', 'Bronx'],
   'salary': [103, 89, 142, 54, 63, 219],
   'vegan': ['No', 'No','No','Yes', 'Yes', 'No']})

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

	salary	vegan	boro_Manhattan	boro_Queens	boro_Brooklyn	boro_Bronx	boro_Staten Island
0	103	No	1	0	0	0	0
1	89	No	0	1	0	0	0
2	142	No	1	0	0	0	0
3	54	Yes	0	0	1	0	0
4	63	Yes	0	0	1	0	0
5	219	No	0	0	0	1	0

OneHotEncoder

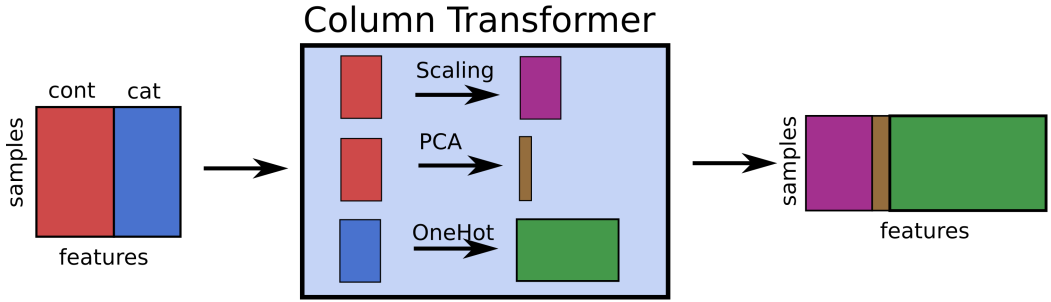
```
import pandas as pd
df = pd.DataFrame({'salary': [103, 89, 142, 54, 63, 219],
                   'boro': ['Manhattan', 'Queens', 'Manhattan',
                            'Brooklyn', 'Brooklyn', 'Bronx']})

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

OneHotEncoder + 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

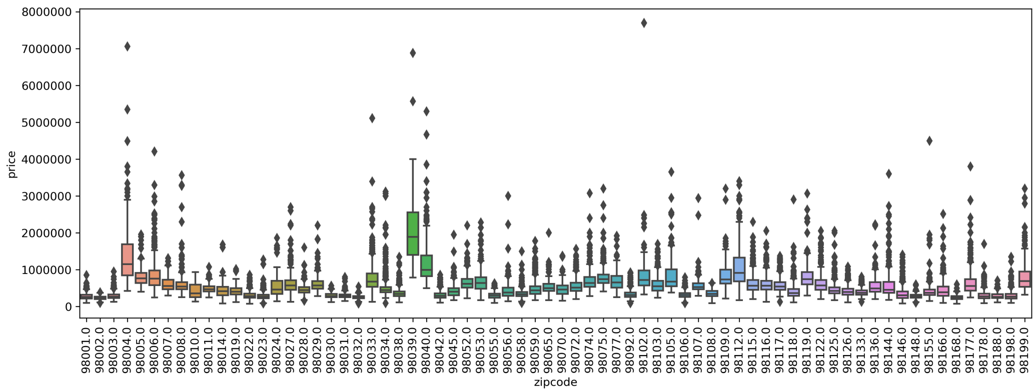
Dummy variables and colinearity

- 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:
  - Some Naive Bayes classifiers.
- In scikit-learn "soon":
  - Decision trees, random forests, gradient boosting

Target Encoding (Impact Encoding)



Target Encoding (Impact Encoding)

- For high cardinality categorical features
- Instead of 70 one-hot variables, one “response encoded” variable.
- For regression:
  - "average price in zip code”
- Binary classification: – “building in this zip code have a likelihood p for class 1”
- Multiclass: – One feature per class: probability distribution

More encodings for categorical features:

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

Load data, include ZIP code

```
data = fetch_openml("house_sales", as_frame=True)
X = data.frame.drop(['date', 'price'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, target)
X_train.columns
```

```
Index(['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat',
       'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

```python
X_train.head()
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	...	zipcode	lat	long	sqft_living15	sqft_lot15
10666	4.0	2.50	2160.0	7000.0	2.0	...	98029.0	47.566	-122.013	2300.0	7440.0
19108	4.0	4.25	3250.0	11780.0	2.0	...	98004.0	47.632	-122.203	1800.0	9000.0
20132	3.0	2.50	1280.0	1920.0	3.0	...	98105.0	47.662	-122.324	1450.0	1900.0
16169	4.0	1.50	1220.0	9600.0	1.0	...	98014.0	47.646	-121.909	1180.0	9000.0
16890	3.0	1.50	2120.0	6290.0	1.0	...	98108.0	47.566	-122.318	1620.0	5400.0

```python
te = TargetEncoder(cols='zipcode').fit(X_train, y_train)
te.transform(X_train).head()
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	...	zipcode	lat	long	sqft_living15	sqft_lot15
10666	4.0	2.50	2160.0	7000.0	2.0	...	6.164e+05	47.566	-122.013	2300.0	7440.0
19108	4.0	4.25	3250.0	11780.0	2.0	...	1.357e+06	47.632	-122.203	1800.0	9000.0
20132	3.0	2.50	1280.0	1920.0	3.0	...	8.503e+05	47.662	-122.324	1450.0	1900.0
16169	4.0	1.50	1220.0	9600.0	1.0	...	4.464e+05	47.646	-121.909	1180.0	9000.0
16890	3.0	1.50	2120.0	6290.0	1.0	...	3.604e+05	47.566	-122.318	1620.0	5400.0

```python
y_train.groupby(X_train.zipcode).mean()[X_train.head().zipcode])
```

zipcode	98029.0	98004.0	98105.0	98014.0	98108.0
price	616356.941	1.357e+06	850306.816	446448.065	360416.811

```python
X = data.frame.drop(['date', 'price', 'zipcode'], axis=1)
scores = cross_val_score(Ridge(), X, target)
np.mean(scores)
```

```
0.69
```

```python
X = data.frame.drop(['date', 'price', 'zipcode'], axis=1)
scores = cross_val_score(Ridge(), X, target)
np.mean(scores)
```

```
0.69
```

```python
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder
X = data.frame.drop(['date', 'price'], axis=1)

ct = make_column_transformer((OneHotEncoder(), ['zipcode']), remainder='passthrough')
pipe_ohe = make_pipeline(ct, Ridge())
scores = cross_val_score(pipe_ohe, X, target)
np.mean(scores)
```

```
0.52
```

```
X = data.frame.drop(['date', 'price', 'zipcode'], axis=1)
scores = cross_val_score(Ridge(), X, target)
np.mean(scores)
```

```
0.69
```

```
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder
X = data.frame.drop(['date', 'price'], axis=1)

ct = make_column_transformer((OneHotEncoder(), ['zipcode']), remainder='passthrough')
pipe_ohe = make_pipeline(ct, Ridge())
scores = cross_val_score(pipe_ohe, X, target)
np.mean(scores)
```

```
0.52
```

```
from category_encoders import TargetEncoder
X = data.frame.drop(['date', 'price'], axis=1)
pipe_target = make_pipeline(TargetEncoder(cols='zipcode'), Ridge())
scores = cross_val_score(pipe_target, X, target)
np.mean(scores)
```

```
0.78
```

Questions?