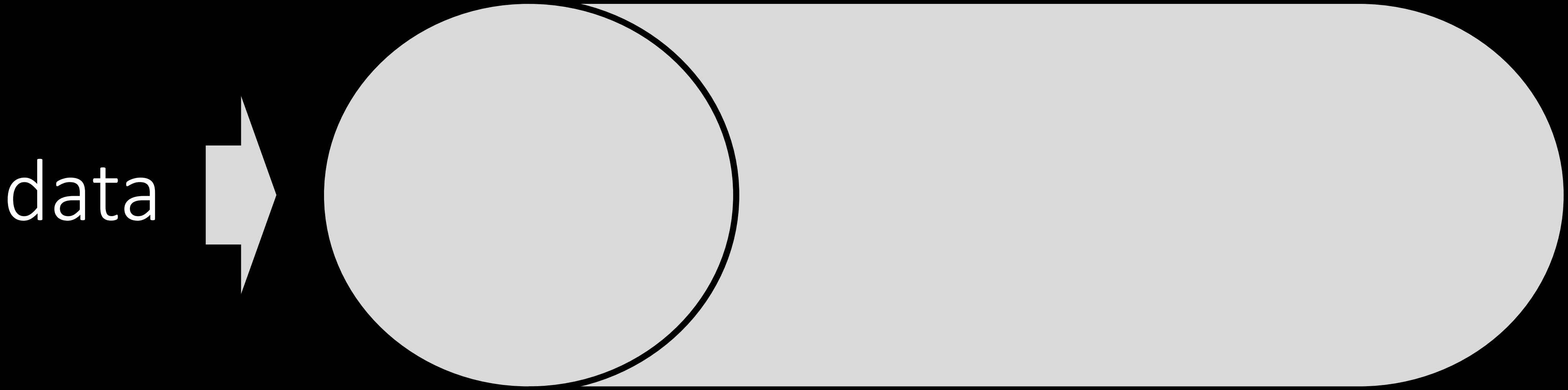
A Simple Machine Learning Pipeline

Pipeline ?

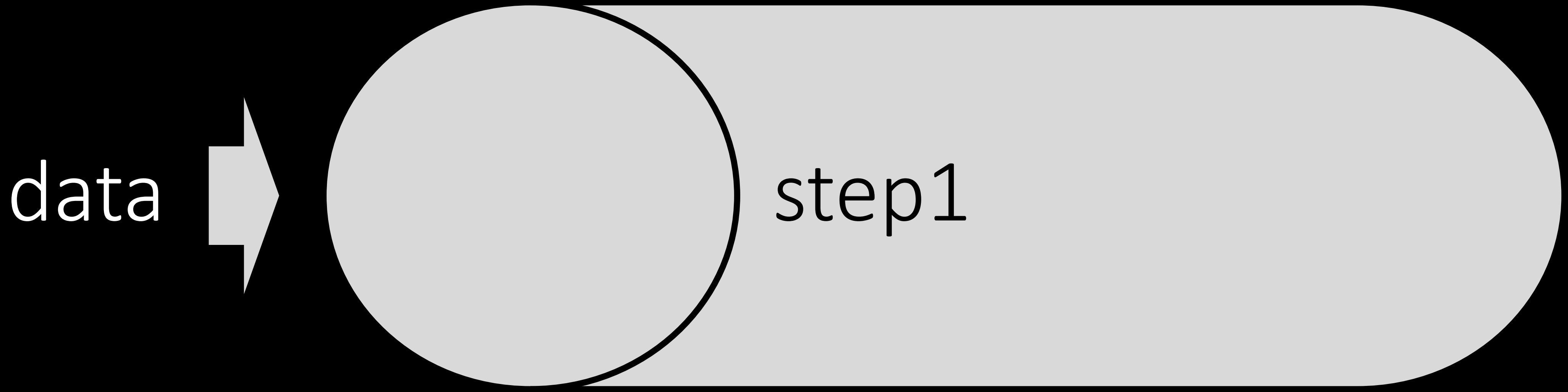
sklearn.pipeline.Pipeline

Pipeline as Algorithm Abstraction

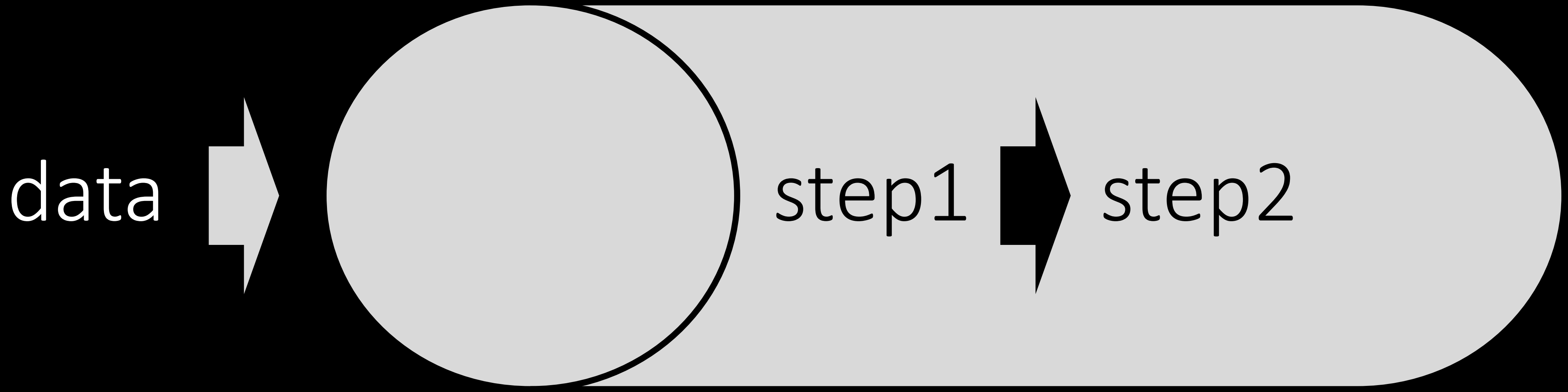
Pipeline as Algorithm Abstraction



Pipeline as Algorithm Abstraction



Pipeline as Algorithm Abstraction

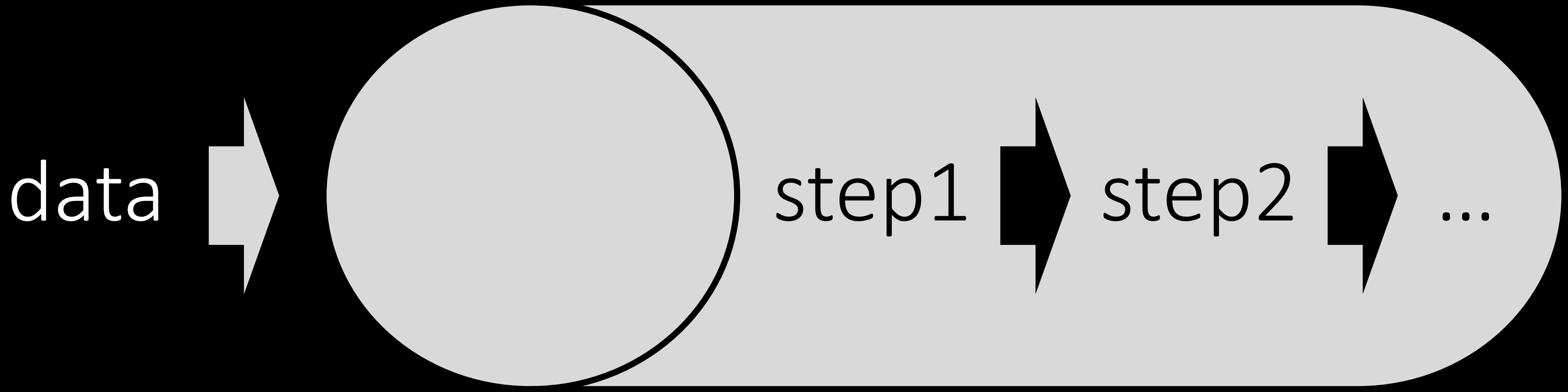


data

step1

step2

Pipeline as Algorithm Abstraction



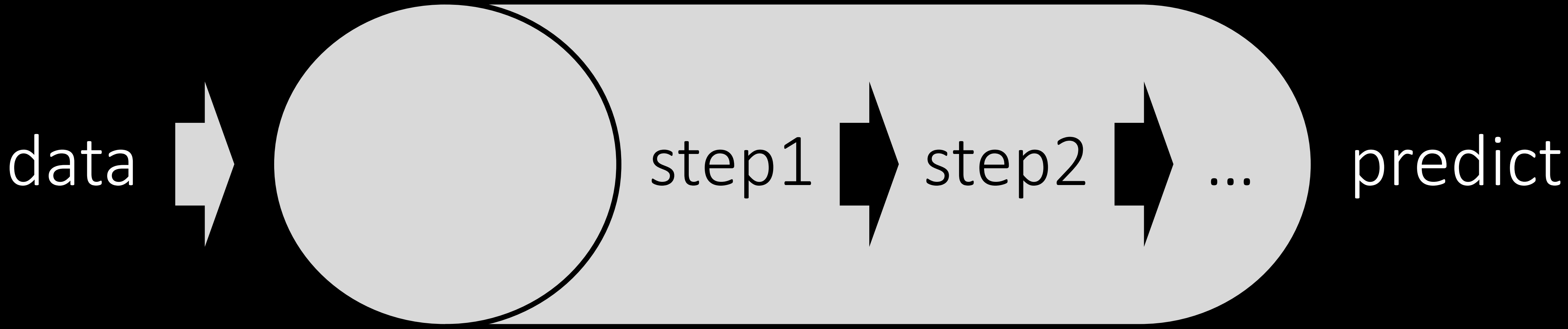
data

step1

step2

…

Pipeline as Algorithm Abstraction

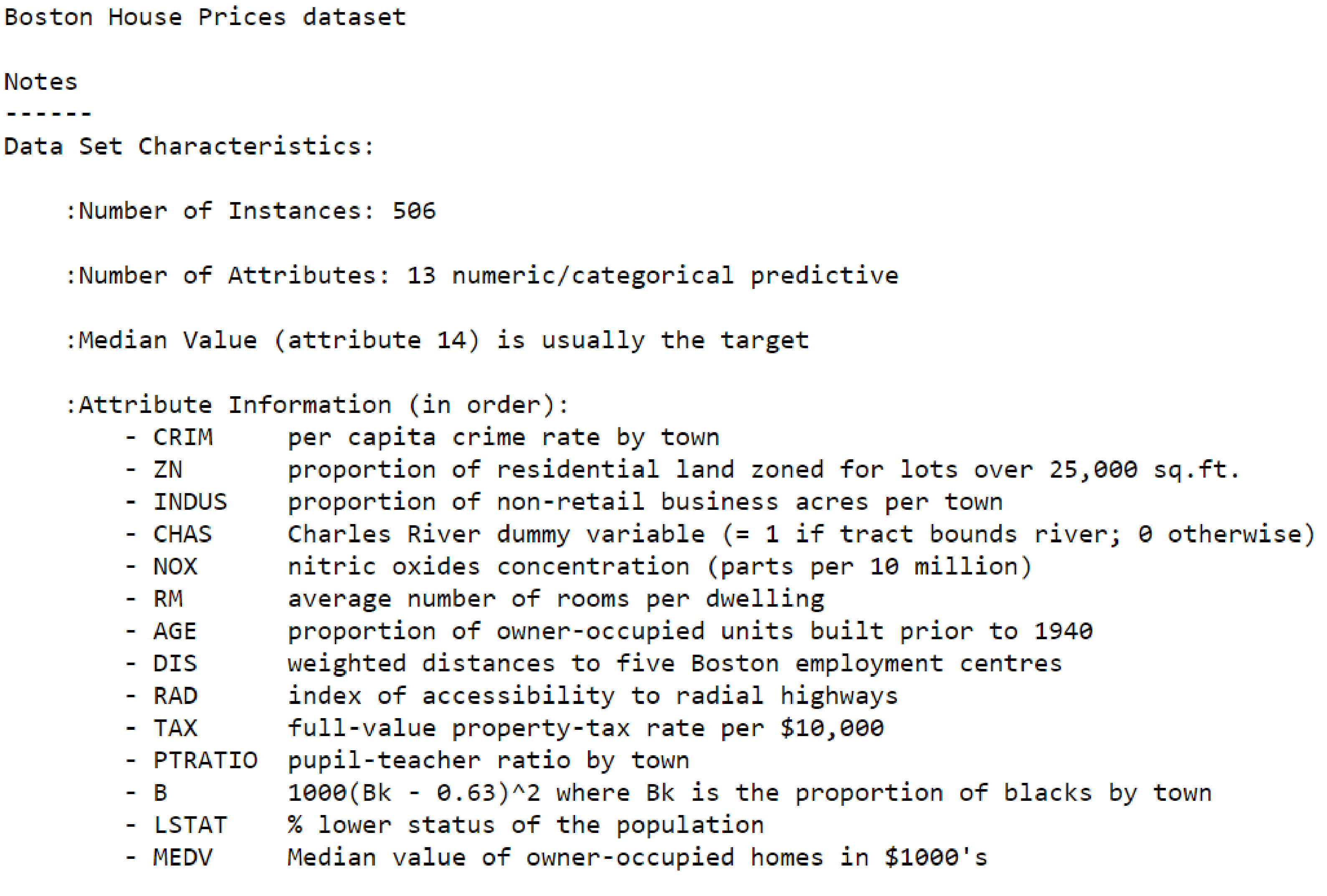


Boston Housing Dataset

https://archive.ics.uci.edu/ml/datasets/Housing

http://bostinno.streetwise.co/2014/12/18/most-expensive-boston-neighborhood-zillow-2014-boston-housing-data/

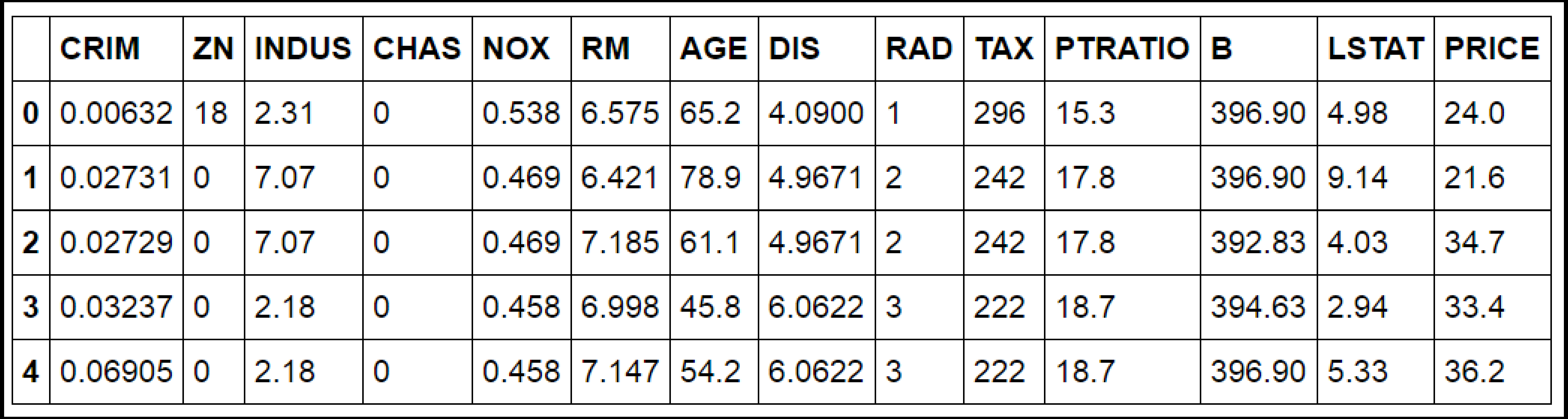
```
from sklearn.datasets import load_boston
housing_data = load_boston()
print housing_data.DESCR
```



```
Boston House Prices dataset

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's
```

```
housing_data.data
housing_data.target
```

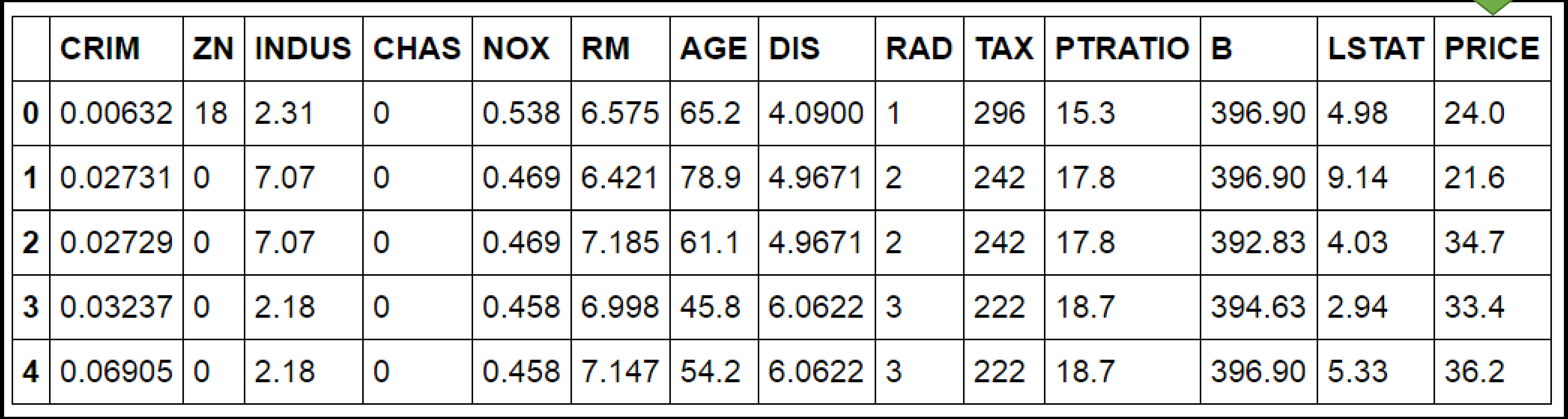
```
import pandas as pd

df = pd.DataFrame(housing_data.data)
df.columns = housing_data.feature_names
df['PRICE'] = housing_data.target
df.head()
```



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

```
import pandas as pd

df = pd.DataFrame(housing_data.data)
df.columns = housing_data.feature_names
df['PRICE'] = housing_data.target
df.head()
```

goal



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

Before implementing any algorithm…

Before implementing any algorithm…

… define the metric!

```
import numpy as np
X = df.drop('PRICE', axis=1)
y = df['PRICE']
```

ML-friendly notation
X: feature matrix (506 x 13)
y: target vector (506,)

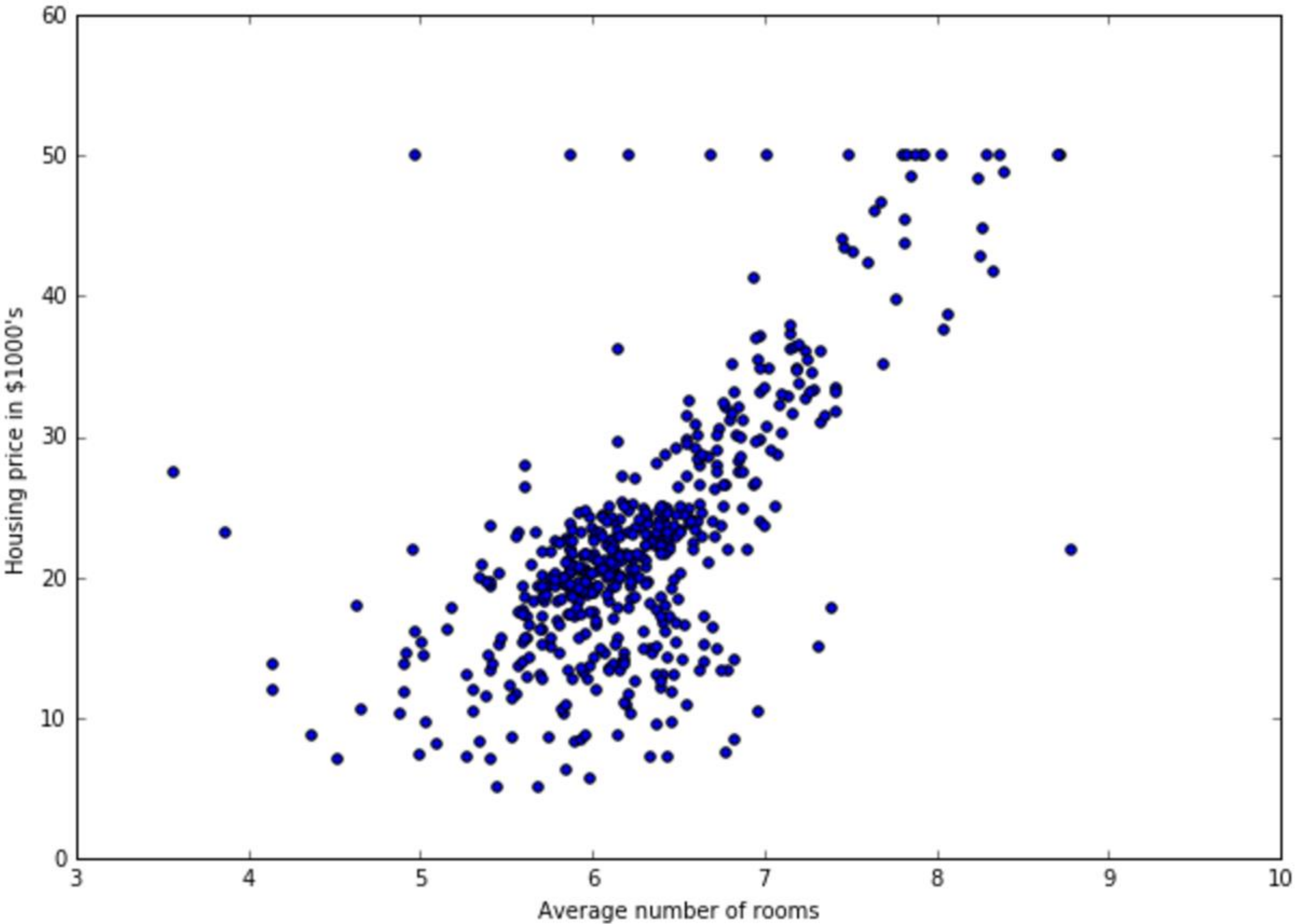
Cross Validation

	fold 1	fold 2	fold 3
run 1	test		
run 2		test	
run 3			test

```
from sklearn import cross_validation

def evaluate_model(X, y, algorithm):
    print 'Mean Square Error'
    scores = cross_validation.cross_val_score(algorithm, X, y, scoring='mean_squared_error')
    print -scores
    print 'Accuracy: %0.2f' % -scores.mean()
```

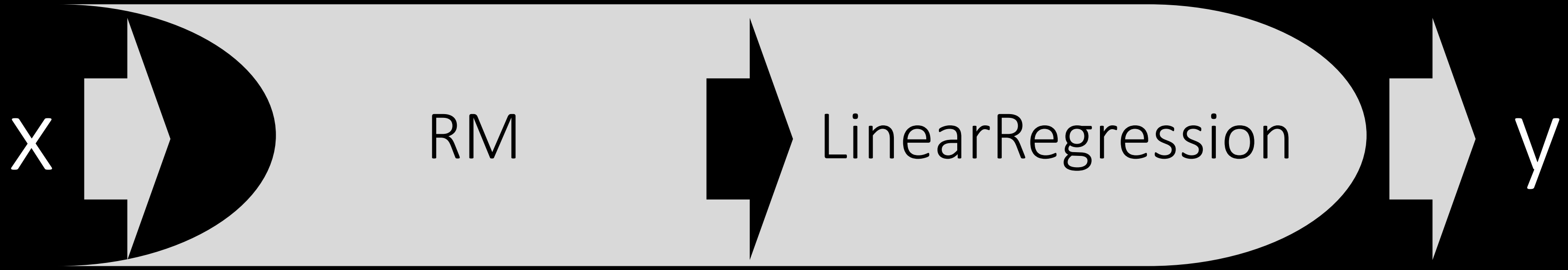
```
from sklearn import cross_validation

def evaluate_model(X, y, algorithm):
    print 'Mean Square Error'
    scores = cross_validation.cross_val_score(algorithm, X, y, scoring='mean_squared_error')
    print -scores
    print 'Accuracy: %0.2f' % -scores.mean()
```

The algorithm – v01


0
10
20
30
40
50
60
Housing price in $1000's
3
4
5
6
7
8
9
10
Average number of rooms

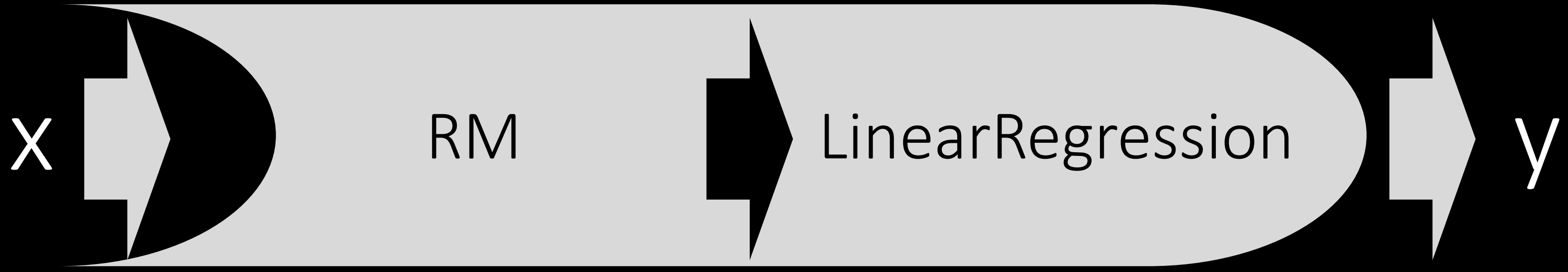
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.linear_model import LinearRegression

def just_RM_column(X):
    RM_col_index = 5
    return X[:, [RM_col_index]]

pipe = make_pipeline(
    FunctionTransformer(just_RM_column),
    LinearRegression()
)
```

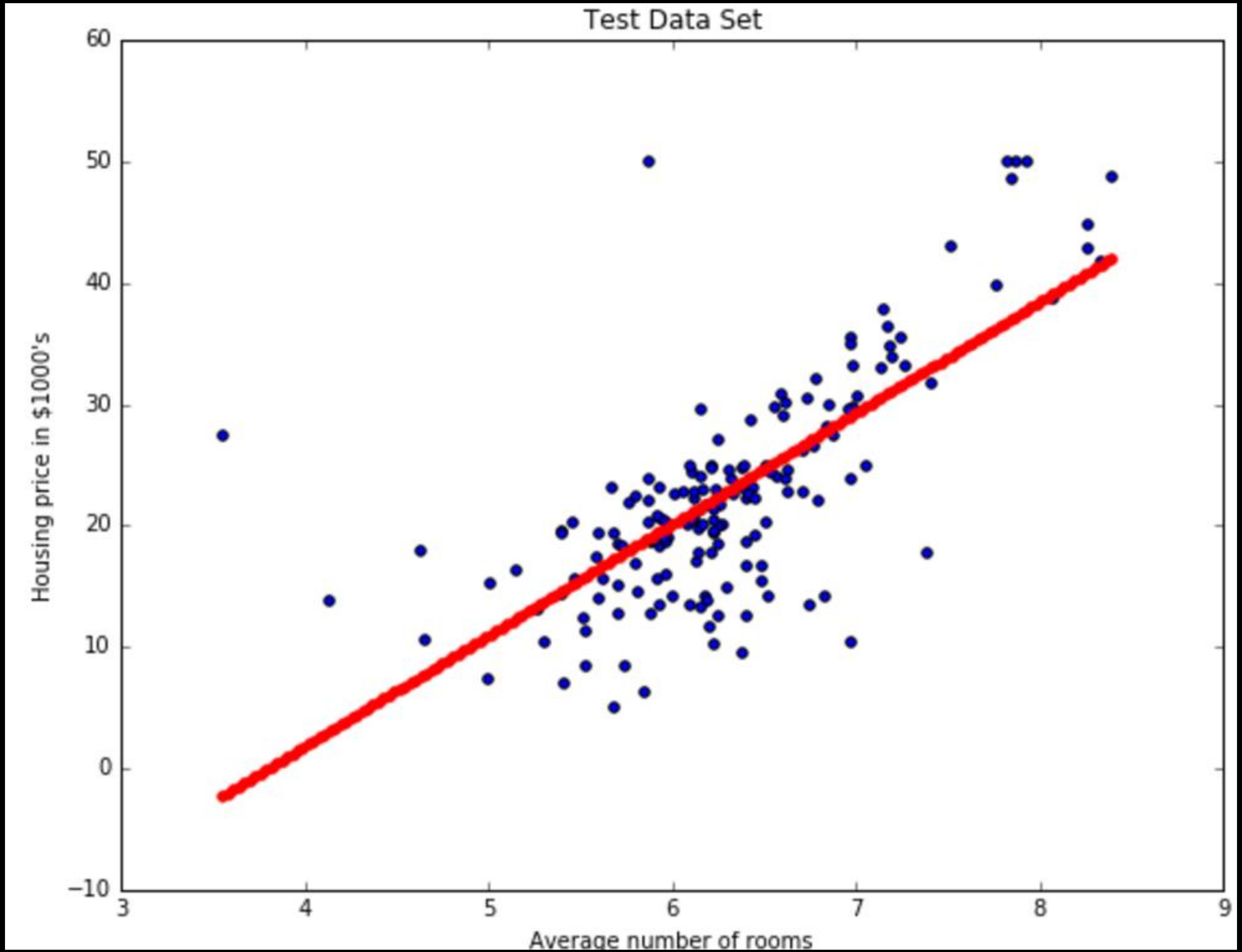
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.linear_model import LinearRegression

def just_RM_column(X):
    RM_col_index = 5
    return X[:, [RM_col_index]]

pipe = make_pipeline(
    FunctionTransformer(just_RM_column),
    LinearRegression()
)
```



```
evaluate_model(X, y, pipe)
```

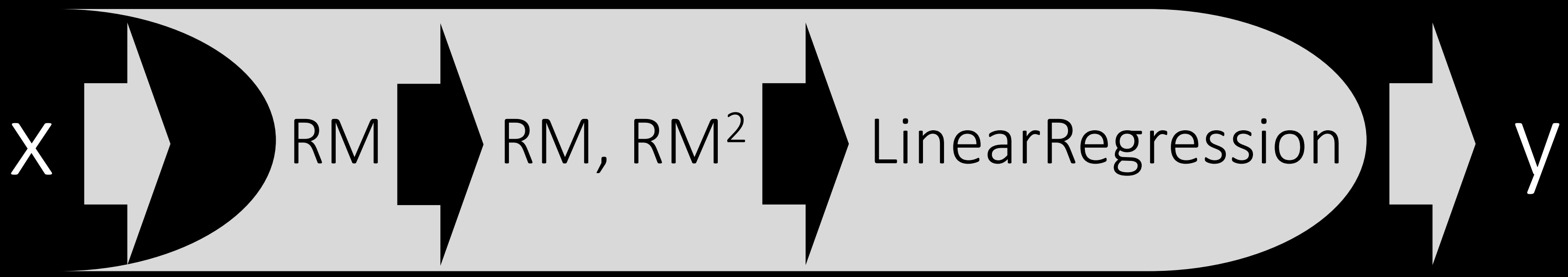
```
Mean Square Error
[ 43.19492771  41.72813479  46.89293772]
Accuracy: 43.94
```




Test Data Set
Housing price in $1000's
Average number of rooms
60
50
40
30
20
10
0
−10
3
4
5
6
7
8
9

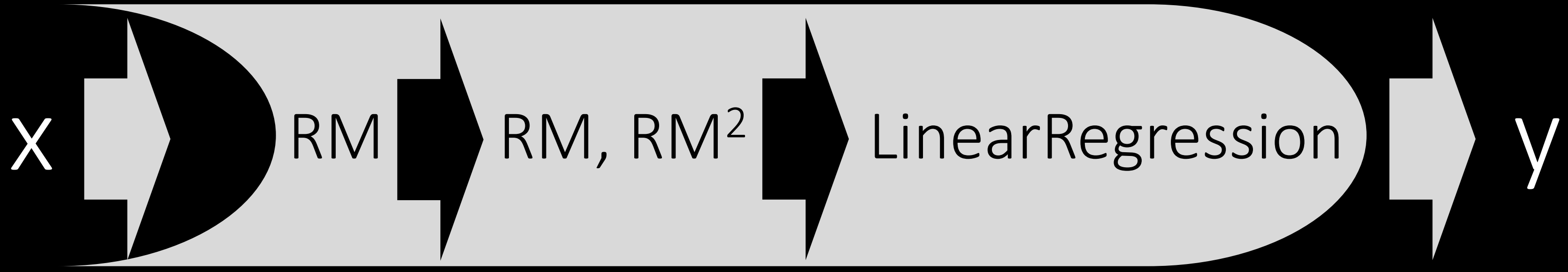
The algorithm – v02

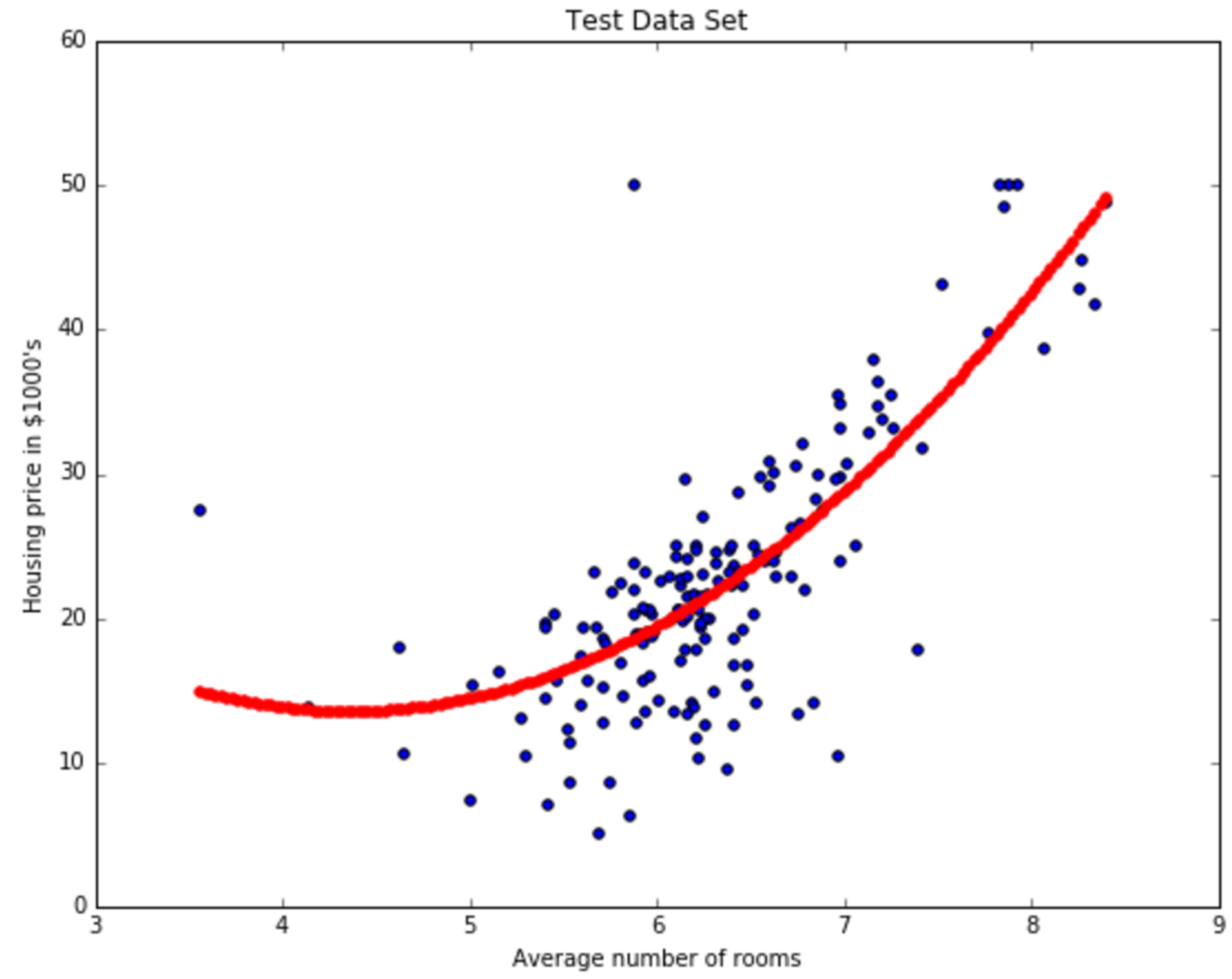
```
def add_squared_col(X):
    return np.hstack((X, X**2))

pipe = make_pipeline(
    FunctionTransformer(just_RM_column),
    FunctionTransformer(add_squared_col),
    LinearRegression()
)
```



```
evaluate_model(X, y, pipe)
```

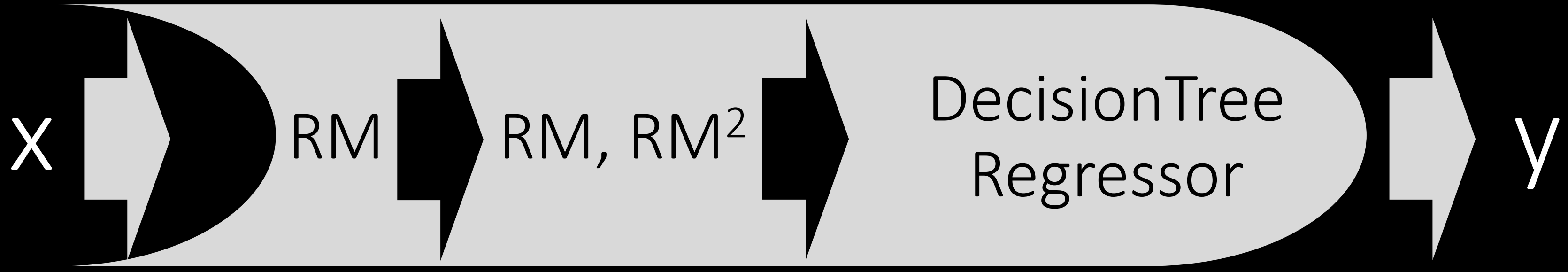
```
Mean Square Error
[ 40.31207562  36.75642688  40.75444834]
Accuracy: 39.27
```




Test Data Set
Housing price in $1000's
Average number of rooms
0
10
20
30
40
50
60
3
4
5
6
7
8
9

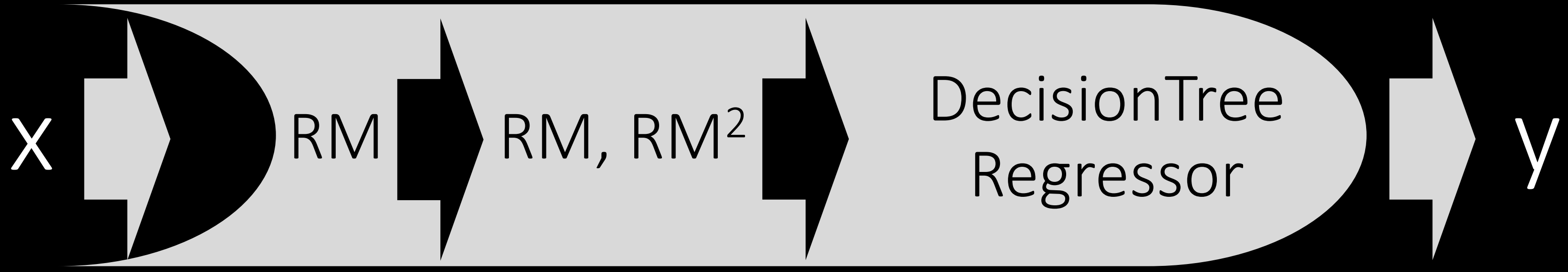
The algorithm – v03

```
from sklearn.tree import DecisionTreeRegressor

pipe = make_pipeline(
   FunctionTransformer(just_RM_column),
   FunctionTransformer(add_squared_col),
   DecisionTreeRegressor(max_depth=3)
)
```



```
evaluate_model(X, y, pipe)
```

```
Mean Square Error
[ 34.75236642  38.48146015  45.16635916]
Accuracy: 39.47
```



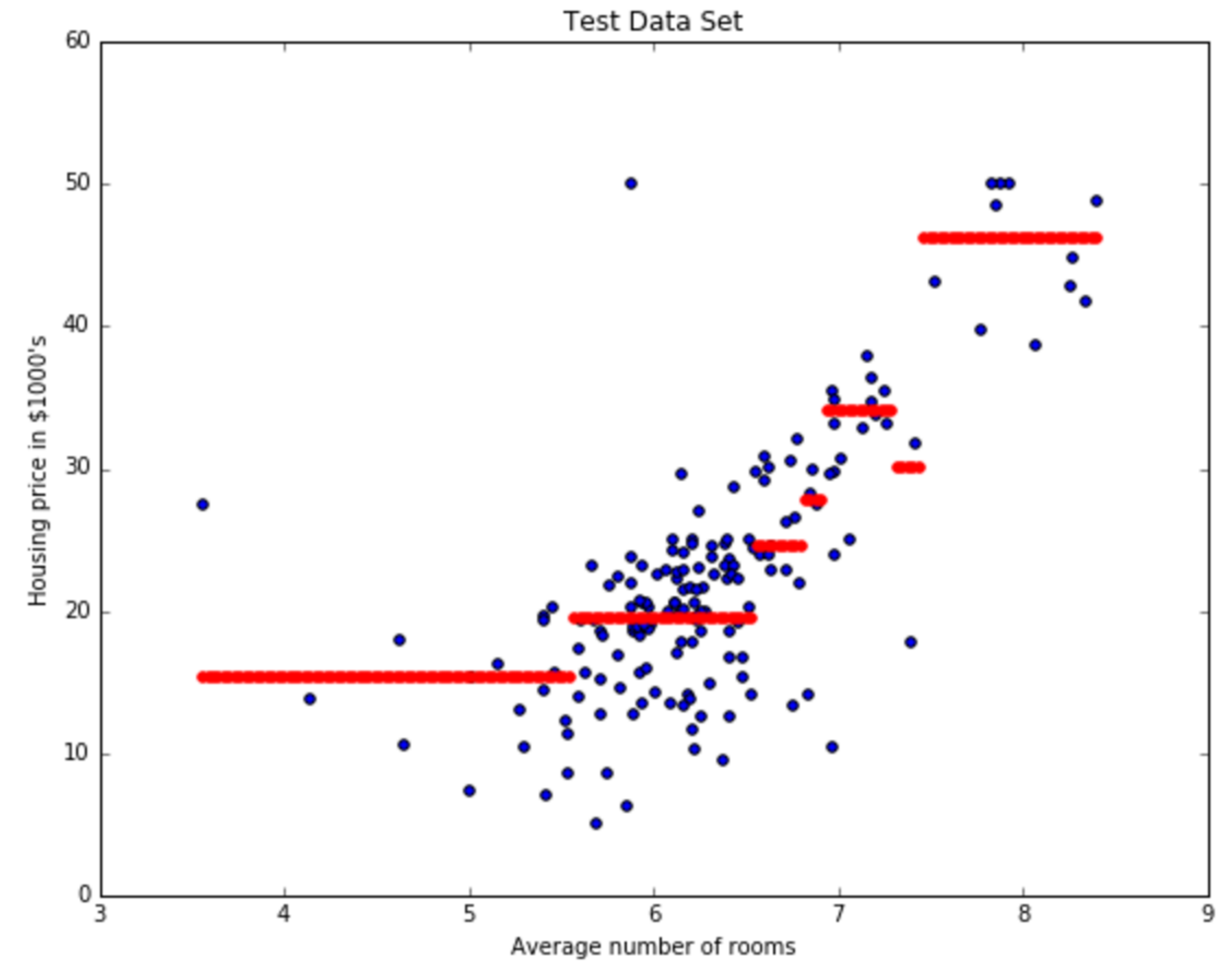
X

RM

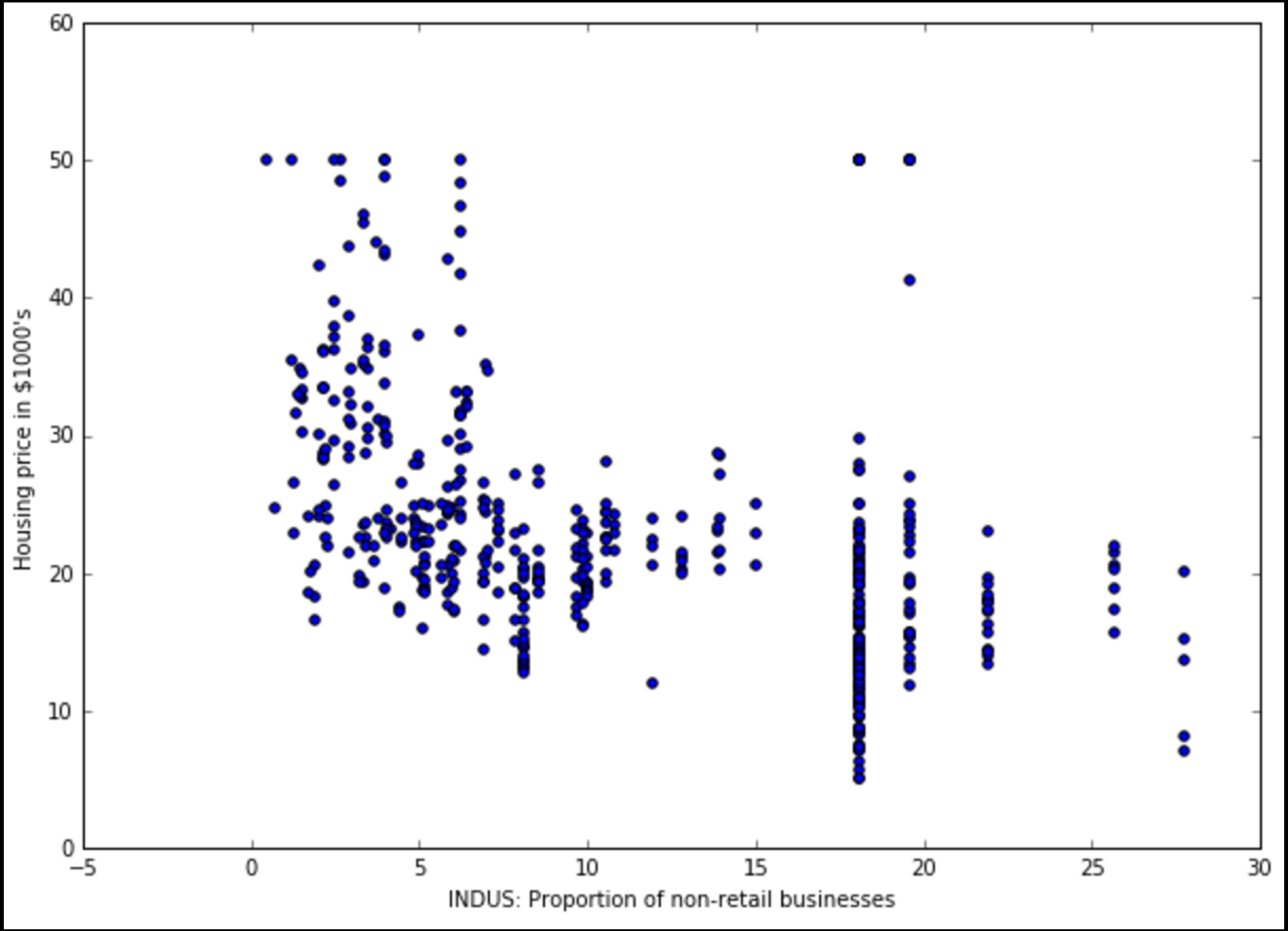
RM, RM^2

DecisionTree Regressor

y

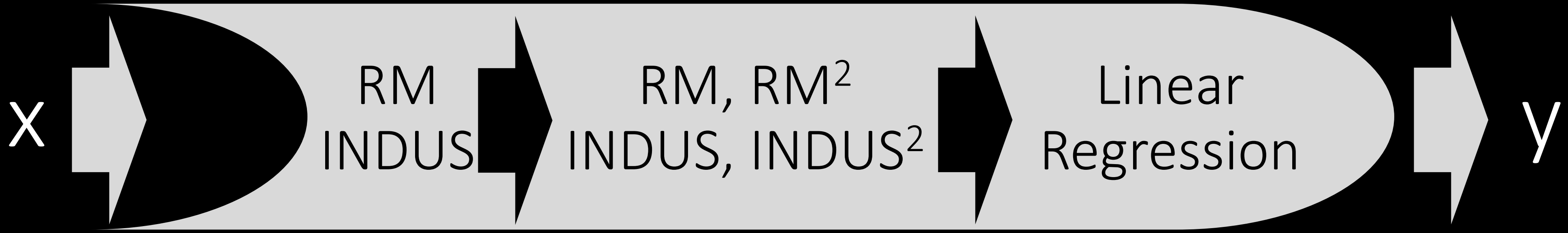

Test Data Set
Housing price in $1000's
Average number of rooms
0
10
20
30
40
50
60
3
4
5
6
7
8
9

The algorithm – v04



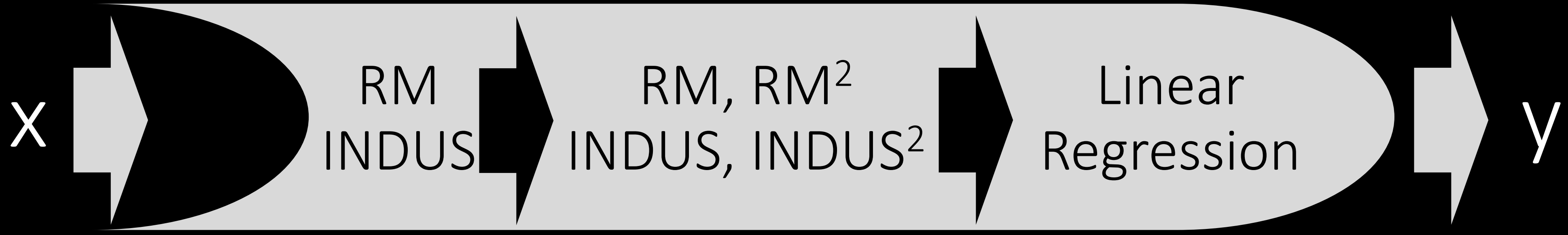
60
50
40
30
20
10
0
Housing price in $1000's
−5
0
5
10
15
20
25
30
INDUS: Proportion of non-retail businesses

```
def RM_and_INDUS_cols(X):
    RM_col_index = 5
    INDUS_col_index = 2
    return X[:, [RM_col_index, INDUS_col_index]]

pipe = make_pipeline(
    FunctionTransformer(RM_and_INDUS_cols),
    FunctionTransformer(add_squared_col),
    LinearRegression()
)
```

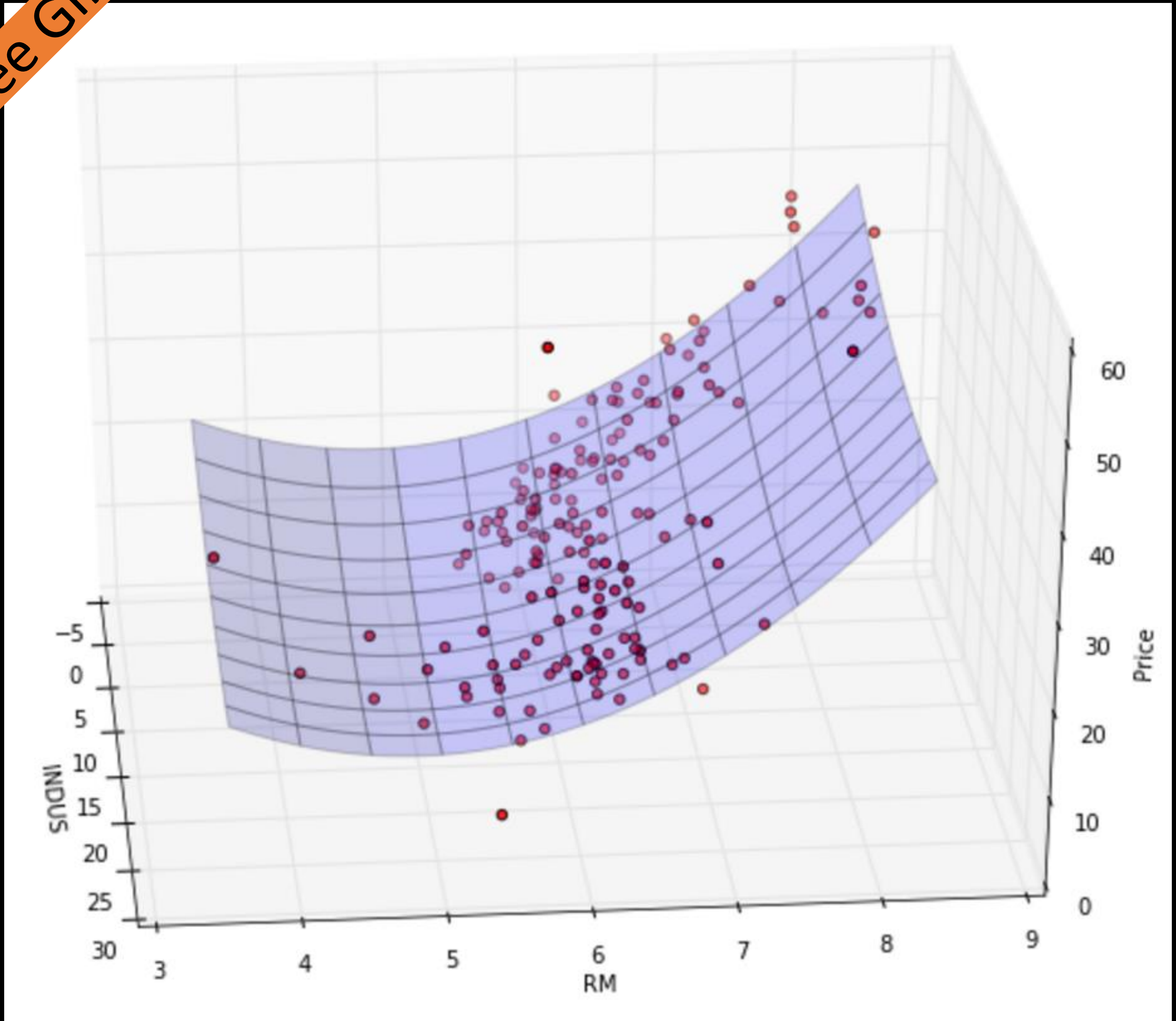


```
evaluate_model(X, y, pipe)
```

```
Mean Square Error
[ 32.3420789   31.4260901   35.95835866]
Accuracy: 33.24
```

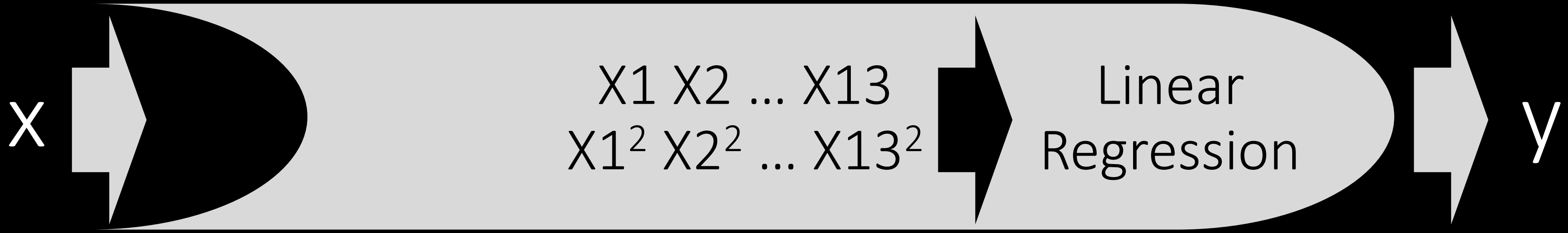


see GIF



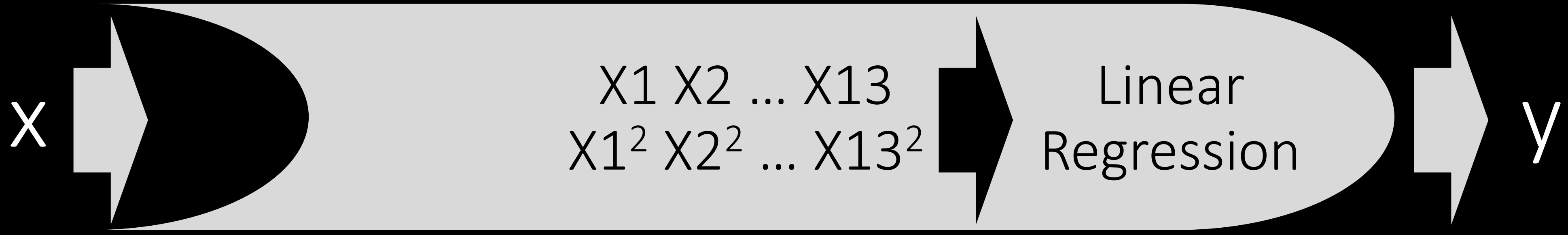
The algorithm – v05

```
pipe = make_pipeline(
   FunctionTransformer(add_squared_col),
   LinearRegression()
)
```



```
evaluate_model(X, y, pipe)
```

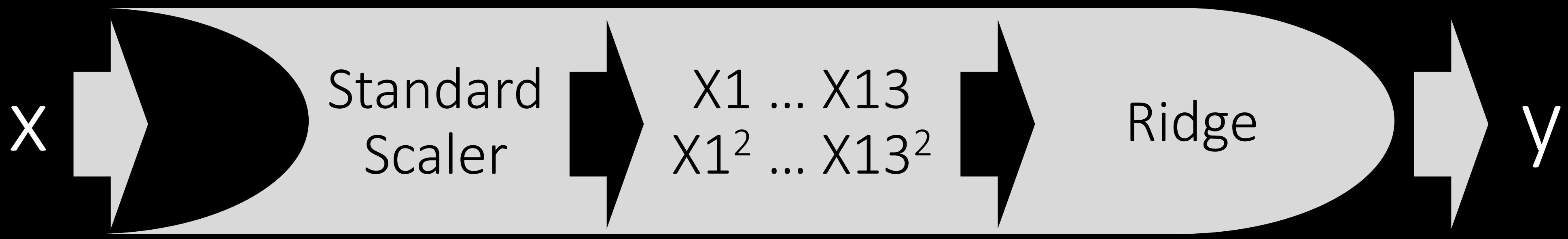
```
Mean Square Error
[ 16.7819682   14.599869    18.17785453]
Accuracy: 16.52
```



The plot??

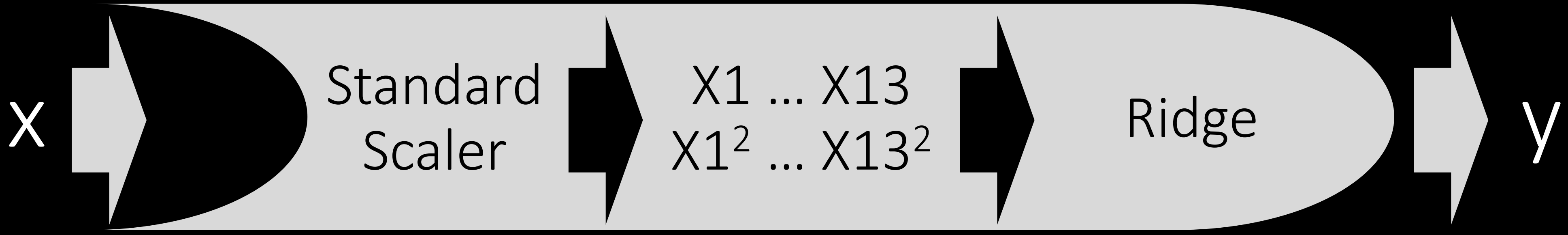
The algorithm – v06

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge

pipe = make_pipeline(
    StandardScaler(),
    FunctionTransformer(add_squared_col),
    Ridge(alpha=3)
)
```



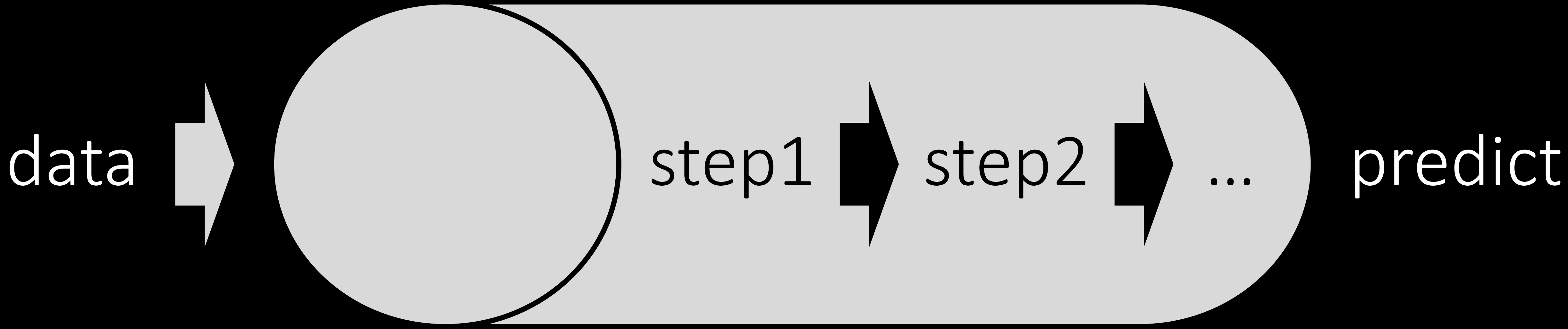
```
evaluate_model(X, y, pipe)
```

```
Mean Square Error
[ 16.4292824   14.50522561  18.27167008]
Accuracy: 16.40
```



In conclusion…

Pipeline as Algorithm Abstraction



data

step1

step2

…

predict

Questions?

Want more?
Data Science Milan

tw: @MrSantoni
marcosantoni.com