

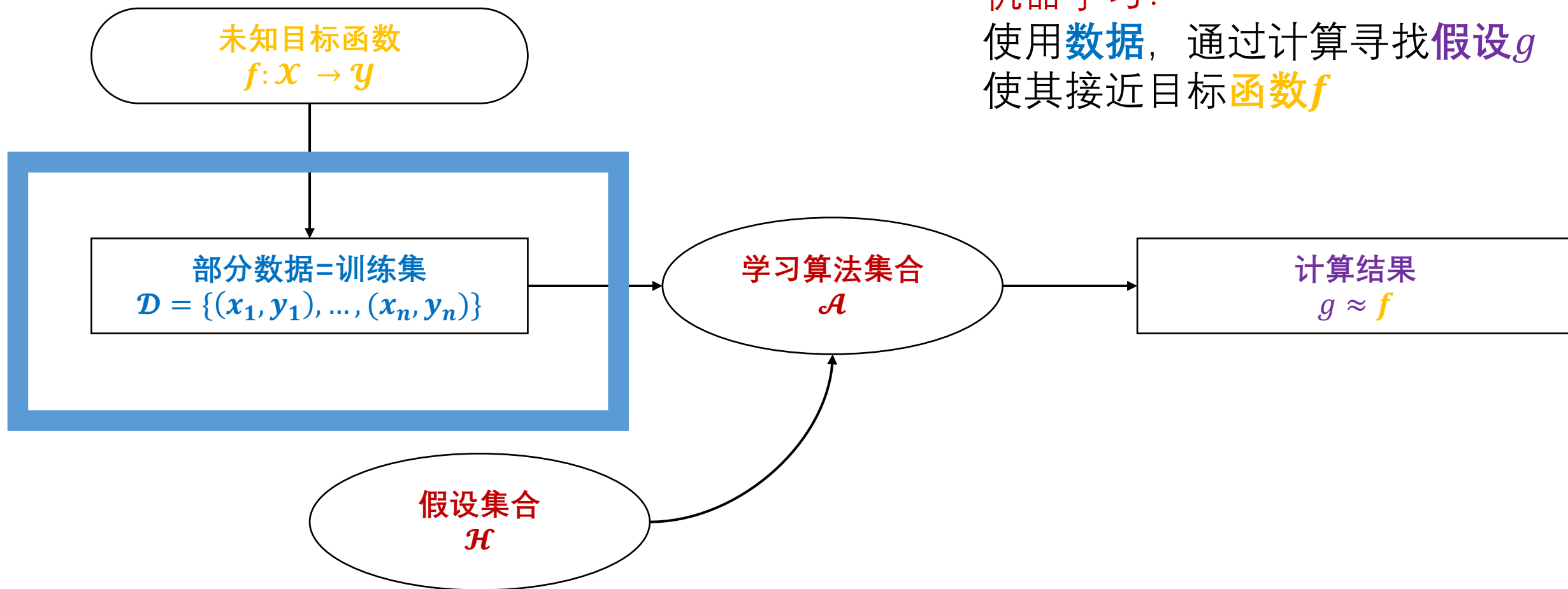
- 决策树
  - 最美的形式
  - 高度的灵活性与表示力
  - 容易过拟合、容易过敏
- 随机森林
  - 鲁棒性很强的算法
  - 良好的能力、难过拟合
  - 能力有时不够尤其回归
  - 简单融合+强个体能力
  - Bagging 算法
  - 将一个个小的**强**算法
  - 通过**简单方式**进行融合
  - 当发现一个灵活算法容易过拟合时
- 梯度下降树
  - 极其敏锐的算法
  - 担当底牌的能力
  - 过拟合、难训练
  - 复杂融合+弱个体能力
  - Boosting 算法
  - 将一个个小的**弱**算法
  - 通过**复杂方式**进行融合
  - 当发现一个问题难求解时



	Ridge	Lasso	SVM	RF	GBDT
解析解	存在	不存在 可近似	存在	随机	随机
算法透明度	高	高	高	较低	较低
算法开销	低	低	较低	较高	高
变量数敏感	是	否	否	是, 可降维	是, 可降维
变量选择	否	是, 线性	是	是	是, 但不用
数据缺失值、分类	否	否	否	是	是
算法灵活性	差	差	适中	高	高
变量个数	一般	很高	很高	高	高
特色	简单有效	有效易懂	有效时的首选 大量稀疏变量 理解数据 核的选择很重要	特征选择 高度灵活的关系 好训但上限低	高度灵活 结构化数据的 State-of-art 但难训

# Call Back

机器学习：  
使用数据，通过计算寻找假设 $g$   
使其接近目标函数 $f$





$2^{0.2}$   
3

# 特征工程初步



# 目录

## CONTENT

01

特征工程  
理论

02

结构化与  
缺失值

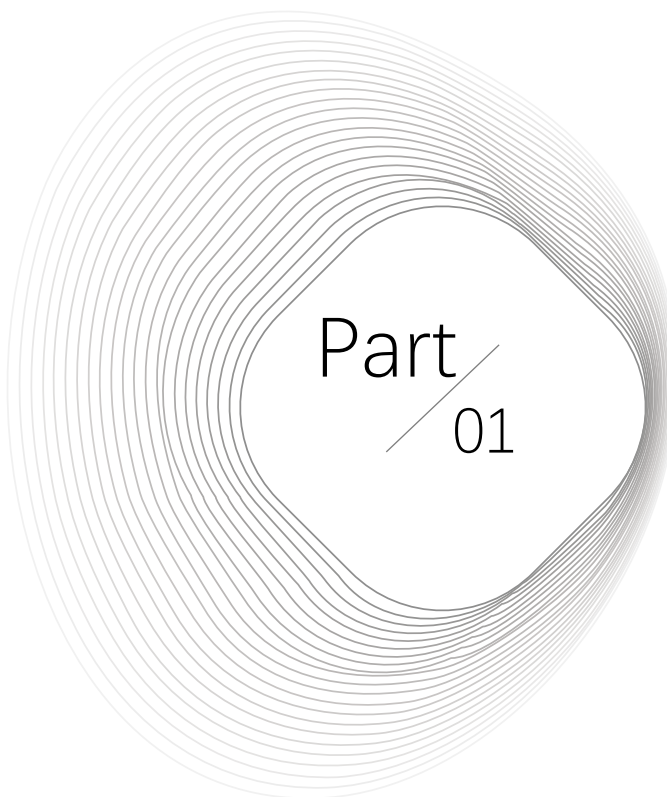
03

改变分布

04

特征选择





Part  
01

# 特征工程理论

- 什么是特征工程
- 为什么要特征工程
- 特征工程主要内容



# 什么是特征工程

## 定义：简而言之，从数据到变量

特征工程是利用数据所在领域的相关知识来构建特征，使得机器学习算法发挥其最佳的过程。它是机器学习中的一个基本应用，实现难度大且代价高。

## 地位：数据和特征是上限，算法和训练是逼近这个上限

Kaggle名言

“挖掘特征是困难、费时且需要专业知识的事，应用机器学习其实基本上是在做特征工程。”

## 实质：连接原始数据与模型，一系列分析的“起手式”

或者说，有无特征工程，是区别实验室玩具学习和现实世界机器学习的界限

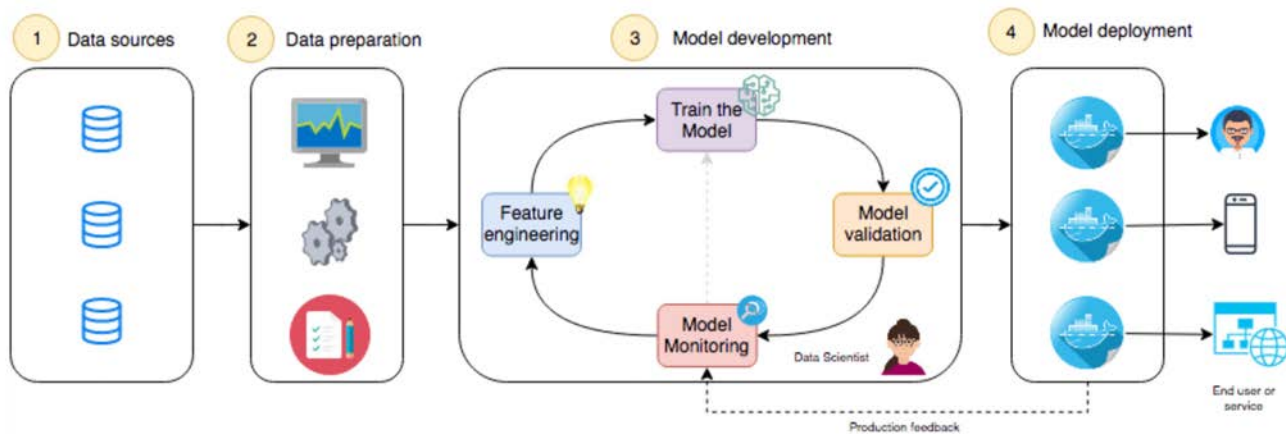


figure [1]: Data science pipeline

# 为什么之前没有学特征工程

## 实际上，已经做了一小部分

在计量中常见的：为什么要汇报描述统计表？为什么要检查相关性？为什么有的变量要取log？  
为什么我们汇报的计量表要长那个样子？

I Just Ran Two Million Regressions Sala-i-Martin 1997 AER

## 目的角度：解释性优先于预测性

变量本身：大部分时间，变量原值有很好的解释性（确实有使用标准化的趋势）

构造模型：我们只关心某个变量（核心解释变量、控制变量）所以一般情况下b很小

综合以上：降维、锦标赛基本上不存在

## 实践角度：没有挨过社会的毒打

模型评估：当我们用机器学习的时候，是否能为之负责？

总是很好看的样本内误差，总能调好的样本外，从不细究的稳定性

计算机论文的行规：在成熟数据库的基础上，默认可以汇报最好的结果并以之为benchmark

## 教学角度：道术巧、纸上谈、大潮至

方法是道，实现是术，优化是巧

其实很多老师没有搞过工程实践，当然也包括我

深度学习的出现让整个特征工程的存在意义受到质疑（基于经验——统计——模型——炼丹）





# 特征工程的主要内容

## 结构化与缺失值：encoding & embedding

将非结构化数据（文本、图像、语言、音乐）结构化

Encoding 和 embedding之间的差别？

## 变量加工：preprocessing

将数据变形为更适合预测的形态

## 特征选择：feature selection

基于理论/经验的

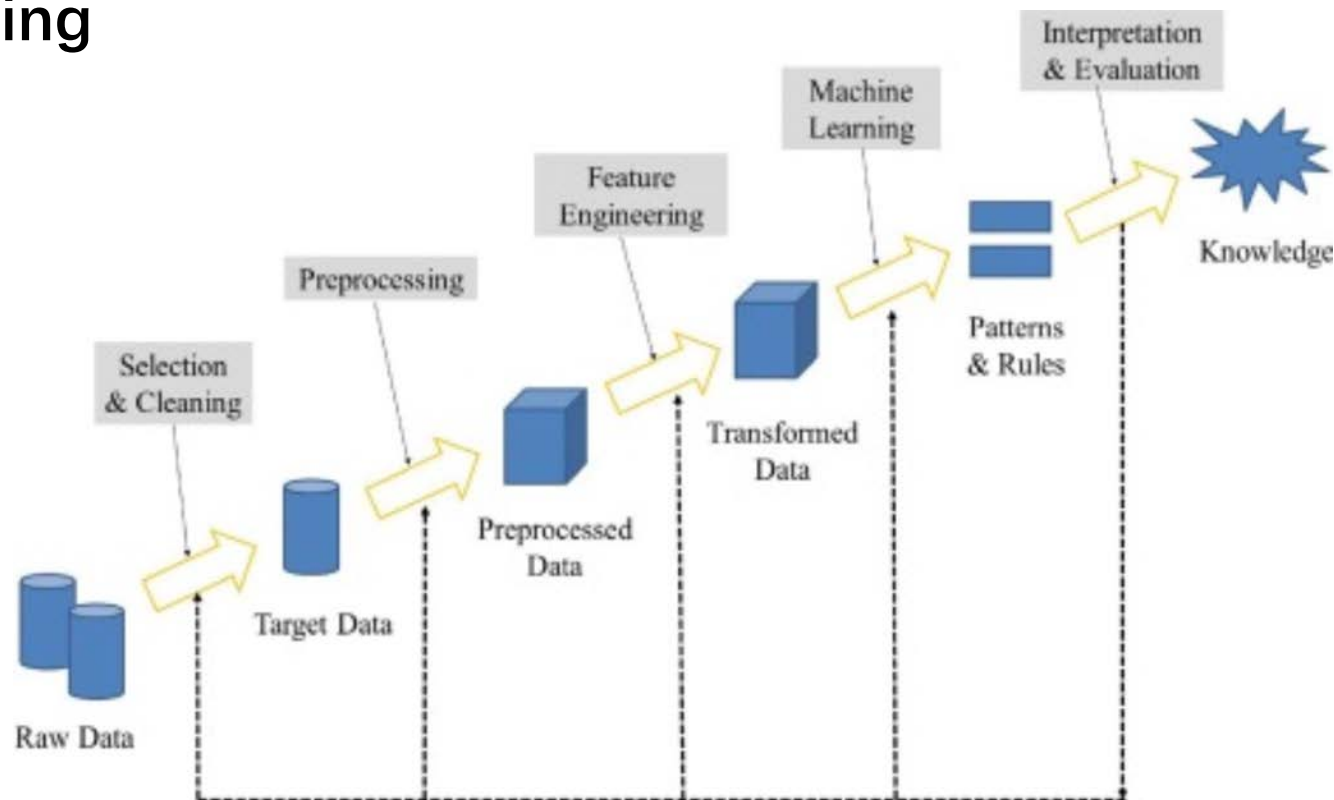
单变量评估

基于统计的多变量评估

基于机器学习的多变量评估

## 特征工程：为最终目的服务

虽然是为了预测，但其实特征工程是一个在预测能力与可解释性之间权衡的“遗老”





Part  
02

# 结构化与缺失值

- 缺失值处理
- encoding
- embedding



# 缺失值处理：先思考为什么缺失

## 缺失在哪个环节造成

原始收集环节？采集环节？清洗环节？  
有没有可能补齐？有没有合适的填补

## 缺失本身是不是就是一种信息

数据缺失在很多时候就是体现价值的  
如：网贷平台的注册自动投标信息

## 是不是一定要承担缺失值带来的影响

计量分析中的主要变量与控制变量  
如果核心目的变成了预测？  
如果缺失超过阈值，就应该删除该变量。（一般为60%）

## 类似于缺失值带来的影响

如果一个变量本身取值没有变化？



# 缺失值处理：一般方法

## Sklearn 专门用于缺失值处理

当然，pandas乃至excel也可以使用类似的功能，  
<https://scikit-learn.org/stable/modules/impute.html>

## 单变量填充

即，对某个变量的填充和其他变量没有关系  
 可以是一个固定值、统计值、中位数、频繁值  
 简单有效，考验对数据的理解

## 多元填充

寻找多个变量（多列之间的关系）

关键参数：

estimator：使用什么方法拟合，可组合，设置超参数

max\_iter=10：最大迭代次数； $\frac{\max |X_t - X_{t-1}|}{\max |X_{\text{known values}}|} < \text{tol}$

tol=1e-3：停止条件阈值

## 近邻填充

缺失：找邻居借一个——不放生跳变

关键参数：

n\_neighbors = 5

Weights = uniform 等权重 distance 距离倒数

```
>>> import numpy as np
>>> from sklearn.impute import SimpleImputer
>>> imp = SimpleImputer(missing_values=np.nan, strategy='mean')
>>> imp.fit([[1, 2], [np.nan, 3], [7, 6]])
SimpleImputer()
>>> X = [[np.nan, 2], [6, np.nan], [7, 6]]
>>> print(imp.transform(X))
[[4.      2.      ]
 [6.      3.666...]
 [7.      6.      ]]
```

```
>>> import numpy as np
>>> from sklearn.experimental import enable_iterative_imputer
>>> from sklearn.impute import IterativeImputer
>>> imp = IterativeImputer(max_iter=10, random_state=0)
>>> imp.fit([[1, 2], [3, 6], [4, 8], [np.nan, 3], [7, np.nan]])
IterativeImputer(random_state=0)
>>> X_test = [[np.nan, 2], [6, np.nan], [np.nan, 6]]
>>> # the model learns that the second feature is double the first
>>> print(np.round(imp.transform(X_test)))
[[ 1.  2.]
 [ 6. 12.]
 [ 3.  6.]]
```

```
>>> import numpy as np
>>> from sklearn.impute import KNNImputer
>>> nan = np.nan
>>> X = [[1, 2, nan], [3, 4, 3], [nan, 6, 5], [8, 8, 7]]
>>> imputer = KNNImputer(n_neighbors=2, weights="uniform")
>>> imputer.fit_transform(X)
array([[1. , 2. , 4. ],
       [3. , 4. , 3. ],
       [5.5, 6. , 5. ],
       [8. , 8. , 7. ]])
```



# 缺失值处理实例：加州房价数据

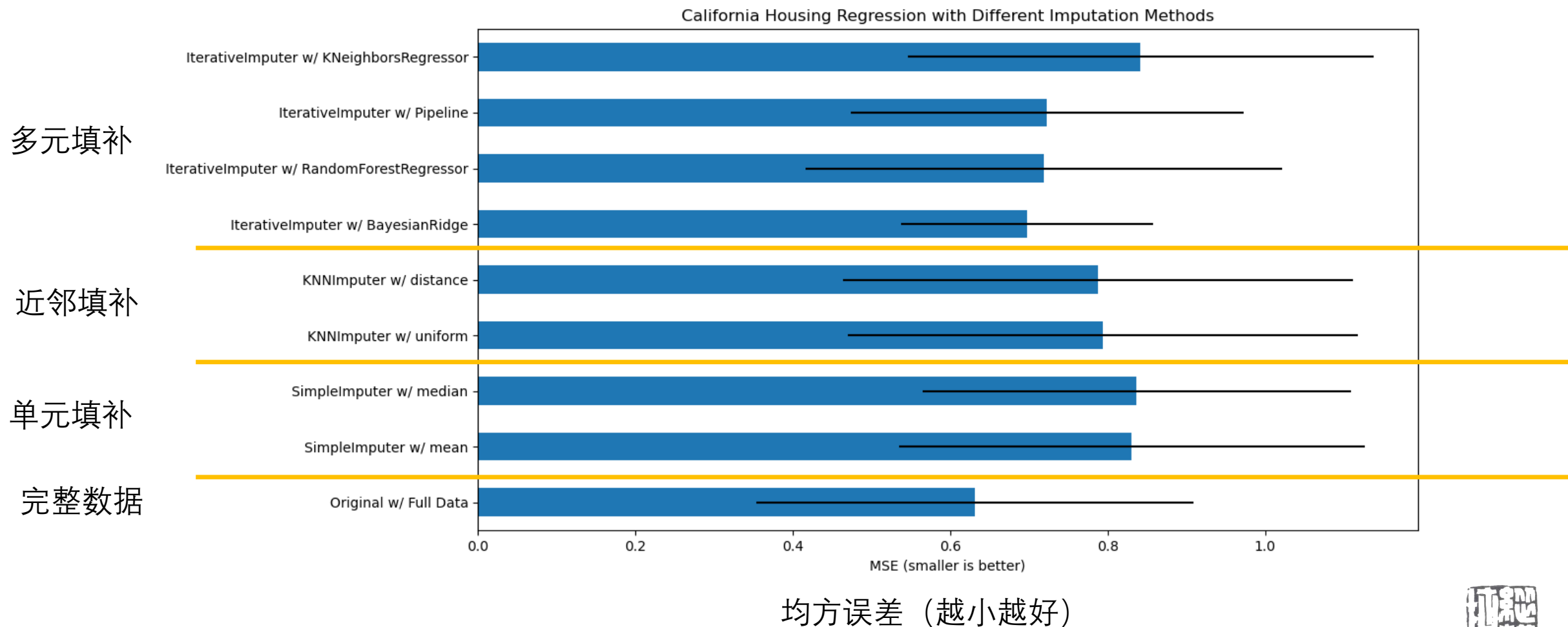
```
"MedInc": "Median income in block in $1,000",
"HouseAge": "Median house age in block",
"AveRooms": "Average number of rooms",
"AveBedrms": "Average number of bedrooms",
"Population": "Block population",
"AveOccup": "Average house occupancy",
"Latitude": "House block latitude",
"Longitude": "House block longitude",
"y": "Median House Price in $100,000"
```

```
# Add a single missing value to each row
#这里给每一行、每一列都添加了缺失值，这种程度的缺失其实比较罕见的
rng = np.random.RandomState(0)
X_missing = X_full.copy()
y_missing = y_full
missing_samples = np.arange(n_samples)
missing_features = rng.choice(n_features, n_samples, replace=True)
X_missing[missing_samples, missing_features] = np.nan
```

```
[np.isnan(X_missing[:, col_i]).sum() for col_i in range(n_features)]
[134, 130, 133, 153, 128, 127, 120, 107]
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	y
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070655	35.631861	-119.569704	2.068558
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386050	2.135952	2.003532	1.153956
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692308	32.540000	-124.350000	0.149990
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429741	33.930000	-121.800000	1.196000
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818116	34.260000	-118.490000	1.797000
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282261	37.710000	-118.010000	2.647250
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333333	41.950000	-114.310000	5.000010



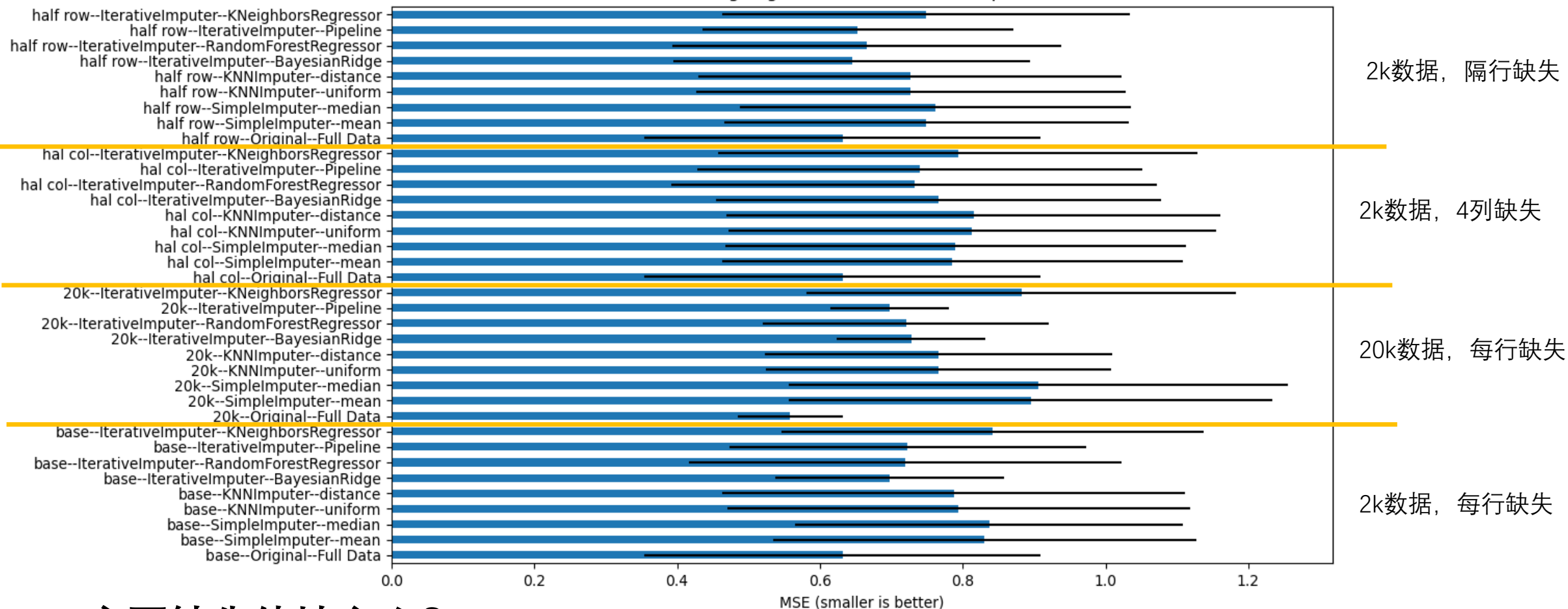




# 缺失值处理实例：不同数据情况

单元、多元、近邻的稳健性

California Housing Regression with Different Imputation Methods



一定要缺失值填充么？



# 结构化：encoding

## 结构化是为了让计算机能够处理部分数据

分类信息：性别、学历

文本信息：one-hot

## 类似处理在计量中已经使用（fixed effect）

在部分方法下，依然是可以使用的（基于树的方法）

但是注意变量的数量与正则化（why？）

## Ordinary encoder vs. one-hot encoder

Ordinary是我们数据存储中常用的，对每一个变量做区分，可以是0, 1, 2, 3……

One-hot就是类似于做城市fixed effect中的

```
In [118]: enc_oh = preprocessing.OneHotEncoder()
enc_oh.fit(X)
enc_oh.transform(X)
```

```
Out[118]: <4x10 sparse matrix of type '<class 'numpy.float64'>'
          with 12 stored elements in Compressed Sparse Row form
```

```
In [120]: enc_oh.transform(X).toarray()
```

```
Out[120]: array([[0., 1., 0., 1., 0., 0., 0., 0., 0., 1.],
                 [1., 0., 0., 0., 1., 0., 0., 1., 0., 0.],
                 [0., 1., 1., 0., 0., 0., 0., 0., 1., 0.],
                 [1., 0., 0., 0., 0., 1., 1., 0., 0., 0.]])
```

```
In [115]: X = [['男', '学士', '程序员'],
               ['女', '硕士', '公务员'],
               ['男', '博士', '外卖员'],
               ['女', np.nan, '交易员']]
```

```
In [116]: from sklearn import preprocessing
enc_ord = preprocessing.OrdinalEncoder()
enc_ord.fit(X)
enc_ord.transform(X)
```

```
Out[116]: array([[ 1.,  1.,  3.],
                 [ 0.,  2.,  1.],
                 [ 1.,  0.,  2.],
                 [ 0., nan,  0.]])
```

```
In [117]: enc_ord = preprocessing.OrdinalEncoder(encoded_missing_value=-1)
enc_ord.fit(X)
enc_ord.transform(X)
```

```
Out[117]: array([[ 1.,  1.,  3.],
                 [ 0.,  2.,  1.],
                 [ 1.,  0.,  2.],
                 [ 0., -1.,  0.]])
```

```
In [123]: enc_ord.transform([['女', '硕士', '教师']])
```

```
-----
ValueError                                Traceback
Cell In[123], line 1
----> 1 enc_ord.transform([['女', '硕士', '教师']])
```

```
In [121]: enc_oh.categories_
```

```
Out[121]: [array(['女', '男'], dtype=object),
           array(['博士', '学士', '硕士', nan], dtype=object),
           array(['交易员', '公务员', '外卖员', '程序员'], dtype=object)]
```

```
In [127]: enc_oh = preprocessing.OneHotEncoder(handle_unknown='infrequent_if_exist')
enc_oh.fit(X)
enc_oh.transform(X)
enc_oh.transform([['女', '硕士', '教师']]).toarray()
```

```
Out[127]: array([[1., 0., 0., 0., 1., 0., 0., 0., 0., 0.]])
```





Part  
03

# 特征放缩

- 为什么要改变变量分布
- 不同放缩技巧
- 为了正态分布
- 连续值to分段值



# 为什么要改变数据的分布？

## 一个熟悉的情景：为什么GDP要取log

一个数据的分布如果不符合方法的假设？

## 我们再去想一下回归参数的意义

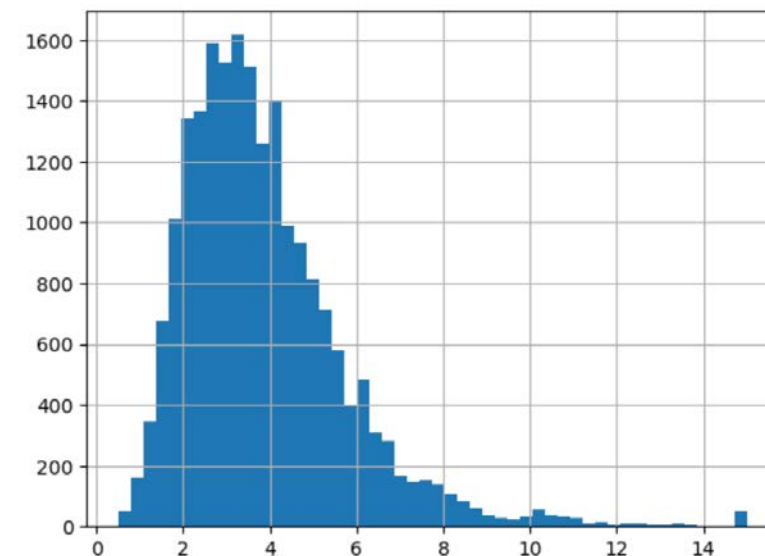
单位在OLS等模型中没有关系，但是解释时会存在一定困难  
“如何解释增长一个单位”=> 增长一个标准差

## 参数对于带正则项的算法的影响：

正则项（ridge、lasso、svm）和系数有关系，而系数又和数据有关系  
我们想让各个数据站在同一条起跑线上->做标准化

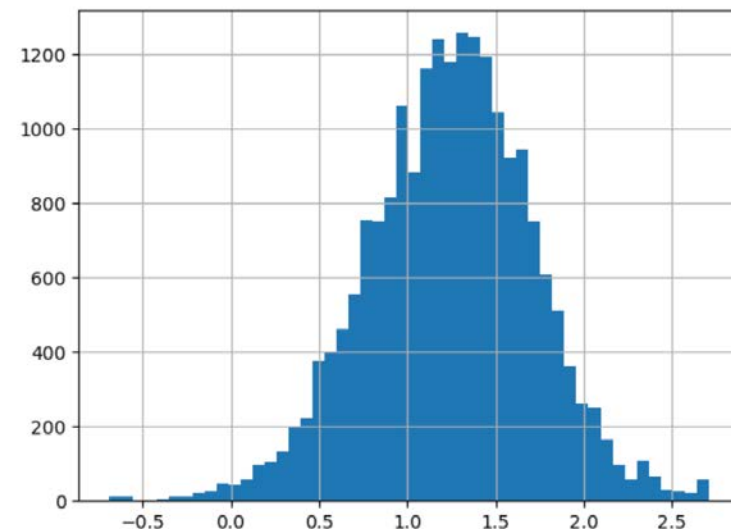
```
#收入中位数分布图  
xdata['MedInc'].hist(bins=50)
```

<Axes: >



```
np.log(xdata['MedInc']).hist(bins=50)
```

<Axes: >



# 为什么要改变数据的分布?

当我们加入惩罚项 &  
需要判断特征重要性

一个隐含问题：超参数的搜寻空间

```
dataset['MedInc_amp'] = dataset['MedInc'] * 1000
```

对于预测准确性的影响

算法对于数据分布可能有所要求

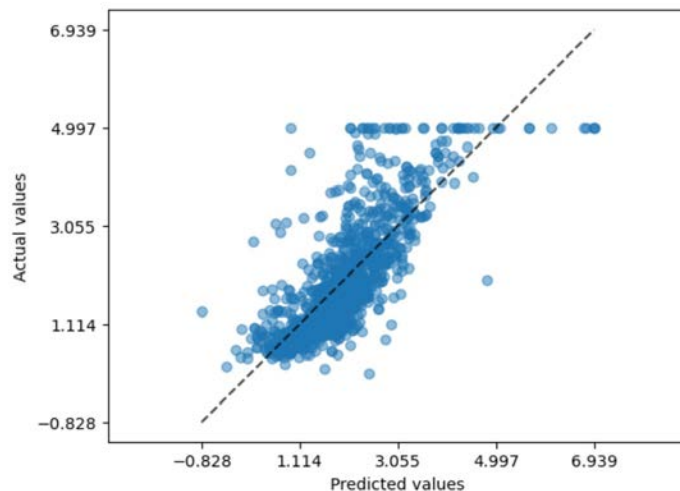
是不是所有算法都需要这一步?

并不是，基于树的方法是基于比较的  
树方法的正则化也并不是来自于对于系数的惩罚  
还是一种balance：方法和数据分布

```
print(ridge_cv.alpha_)
print(ridge_cv.coef_)
compute_score(y_test, y_pred_ride)
```

135.0  
[ 4.26013872e-01 9.88011065e-03 -7.81382469e-02 4.84545338e-01  
-6.41220844e-06 -4.59026085e-03 -4.12482727e-01 -4.23757968e-01]

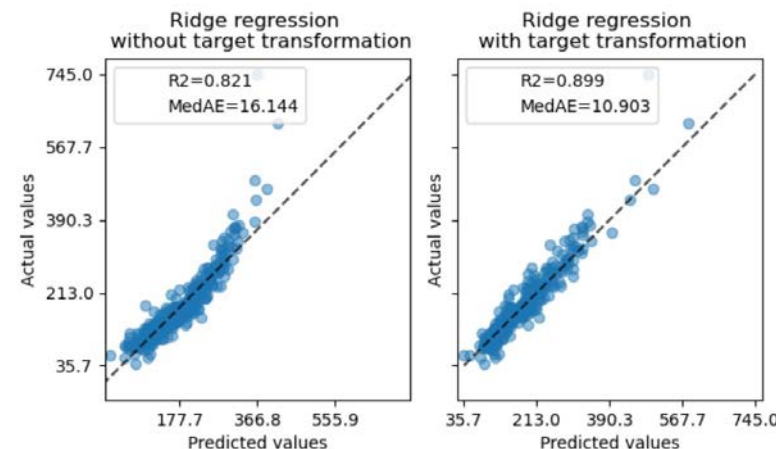
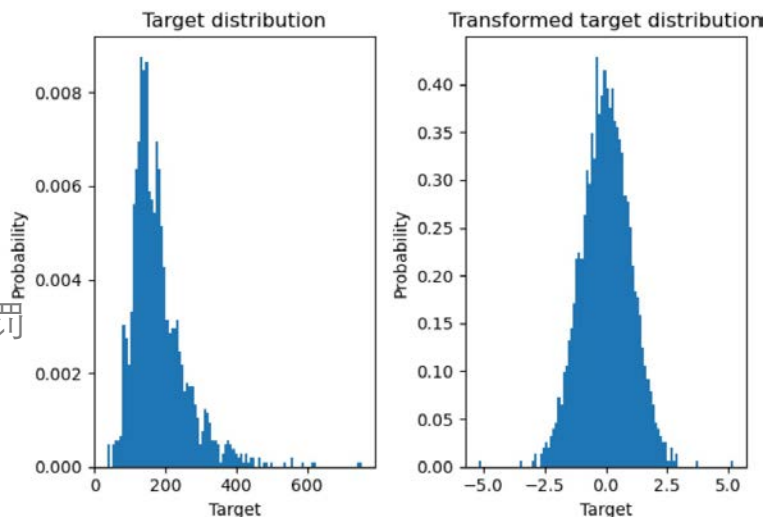
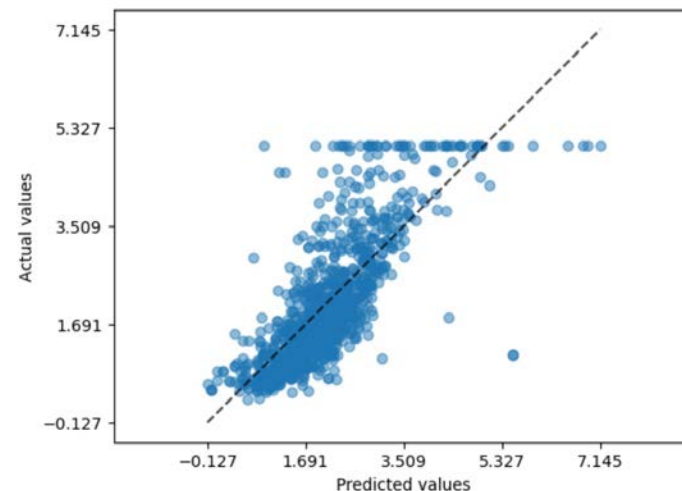
{'R2': '0.590', 'MedAE': '0.419'}



```
print(ridge_cv_amp.alpha_)
print(ridge_cv_amp.coef_)
compute_score(y_test_amp, y_pred_ride_amp)
```

143.5  
[ 4.27827457e-04 9.92719824e-03 -7.92108785e-02 4.87831974e-01  
-6.28759122e-06 -4.59731774e-03 -4.10334492e-01 -4.21583251e-01]

{'R2': '0.590', 'MedAE': '0.420'}

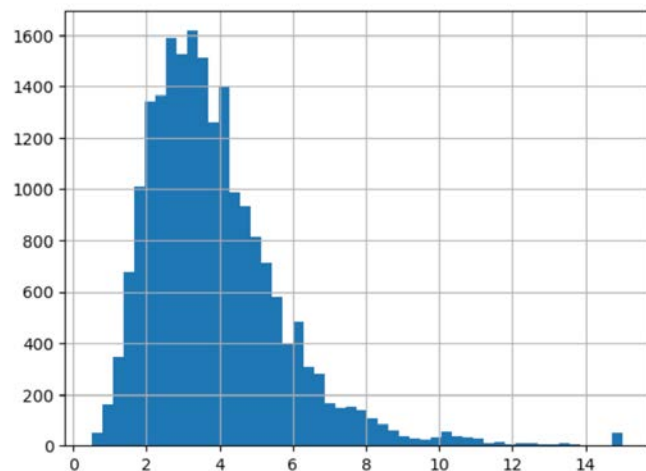


## 3.2

## 放缩方法：原始数据（见代码）

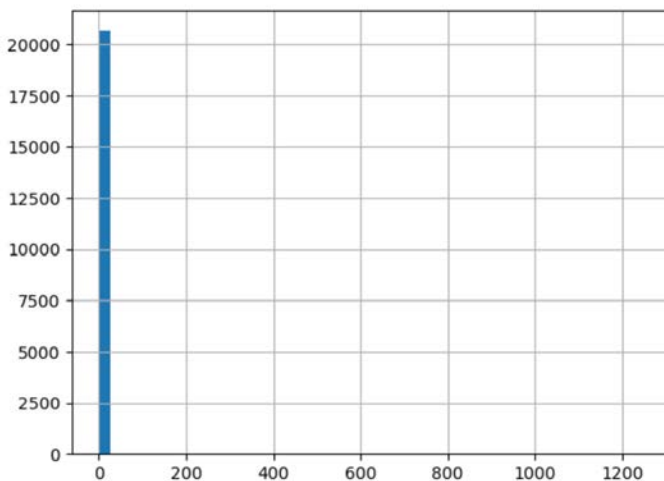
```
#收入中位数分布图  
xdata['MedInc'].hist(bins=50)
```

<Axes: >



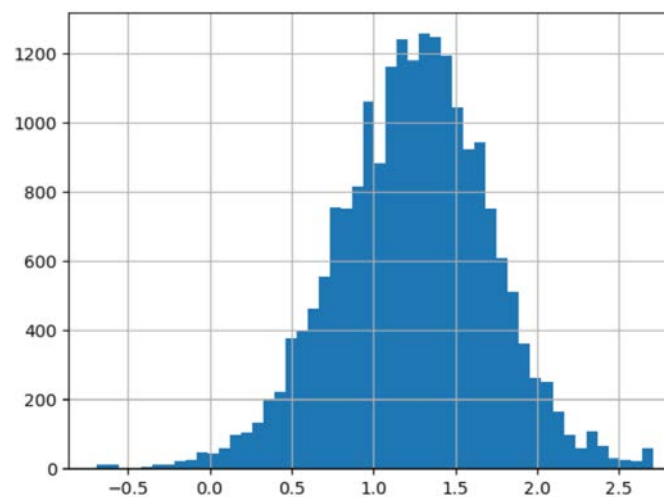
```
#平均房屋入住率存在异常值，使得分布很奇怪  
xdata['AveOccup'].hist(bins=50)
```

<Axes: >



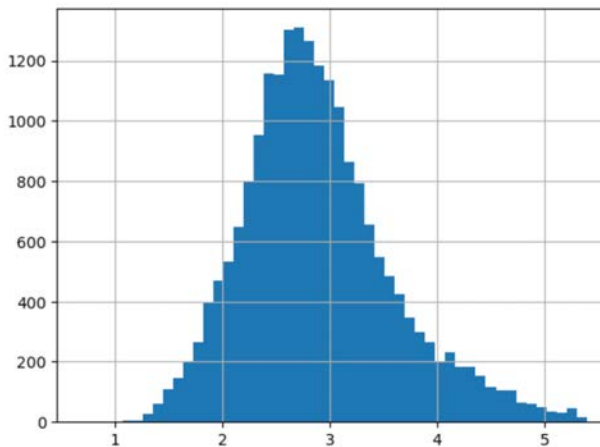
```
np.log(xdata['MedInc']).hist(bins=50)
```

<Axes: >



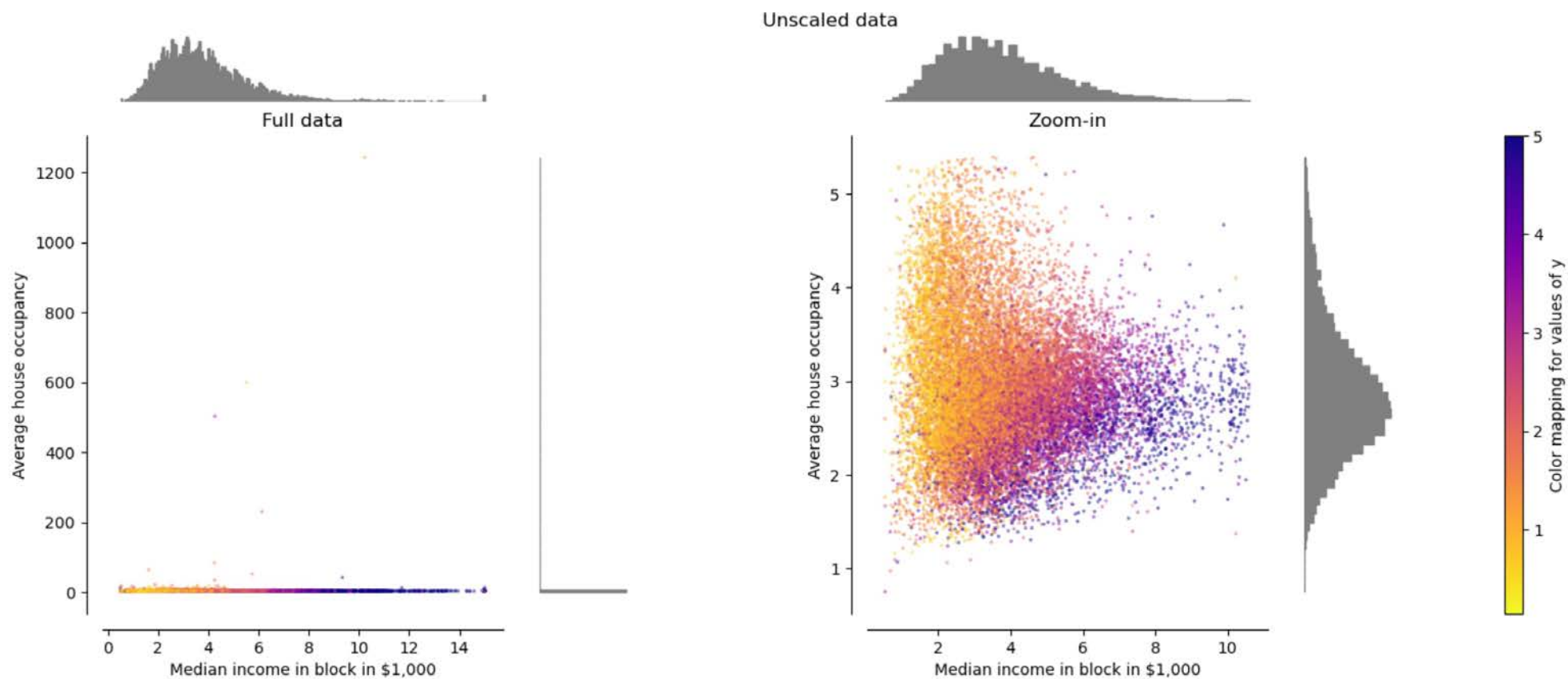
```
#排除1%的点后  
xdata[xdata['AveOccup'] <= xdata['AveOccup'].quantile(0.99)].hist(bins=50)
```

<Axes: >



## 3.2

## 放缩方法：原始数据：解读作图



## 3.2

## 放缩方法 (1) : 标准化

$$X' = \frac{X - \text{mean}(X)}{\sqrt{\text{Var}(X)}}$$

## 简单粗暴且好用

把数据分布改写为均值为0, 标准差为1

适应正则、适应变量选择

线性变化, 即有  $f(x + y) = f(x) + f(y)$   $f(nx) = nf(x)$

经济解释性很好, 甚至更好

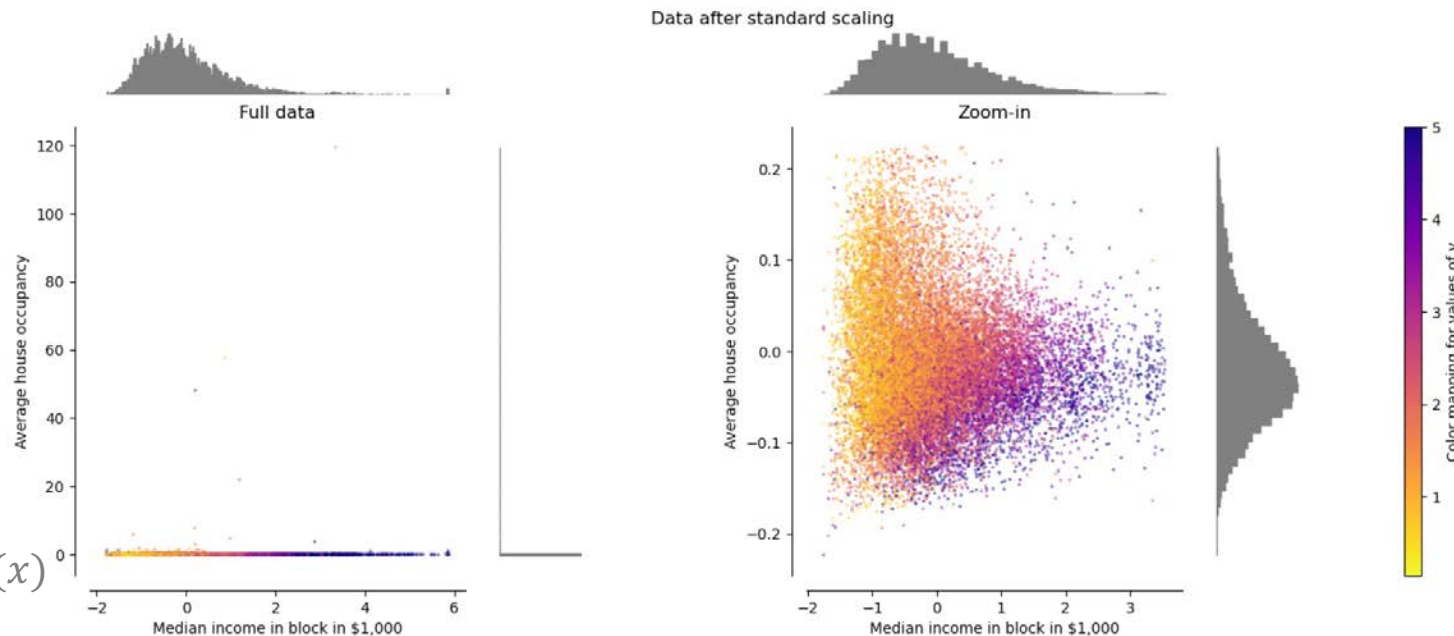
数据取值的意义也很好

## 局限

无法处理极端偏离值

数据分布可能依然集中

无法把数据变得更像正态分布





## 放缩方法（2）：最大最小放缩 & 最大绝对值放缩

### 最大最小放缩

把数据改写成 $[0,1]$

线性变换。不改变数据原有分布特征

不能处理极端值情况，敏感于极端值，变得拥挤

### 最大绝对值放缩

如果数据都是正数，则放缩至 $[0,1]$

如果数据都是负数，则放缩至 $[-1,0]$

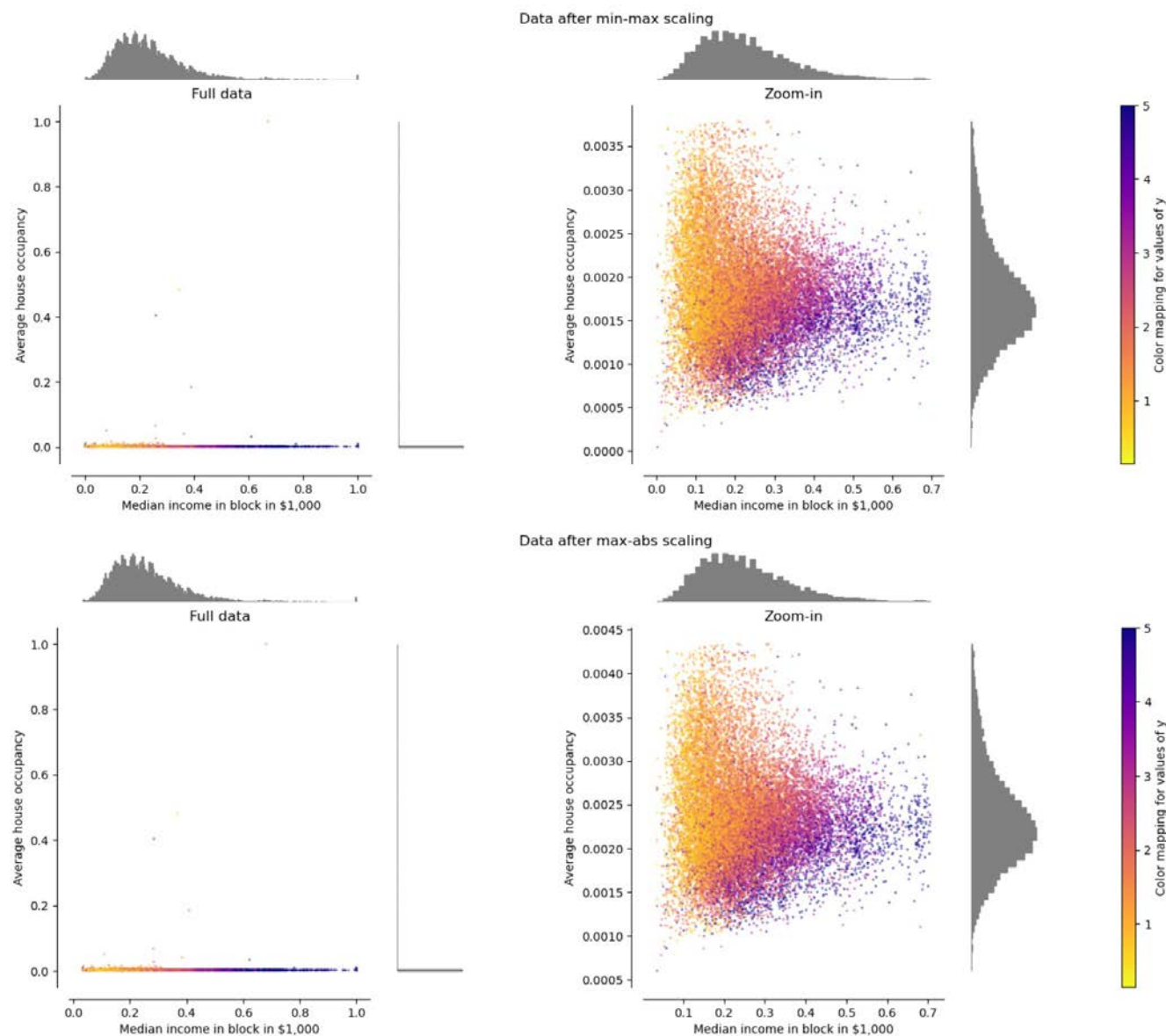
如果数据有正有负，则放缩至 $[-1,1]$

线性变换。不改变数据原有分布特征

不能处理极端值情况，敏感于极端值，变得拥挤

### 为encoder设计

保持0不被变化



## 3.2 放缩方法（3）：缩尾方法与稳健放缩

### 数据缩尾winsorize

数据可能由于噪音出现极端值

（注意一些有意义的堆积：右（左）截断、断点、bunching）

对于无意义极端值，我们可以使用缩尾（95,99, 99.9）

### 缩尾后的数据变形：稳健缩放

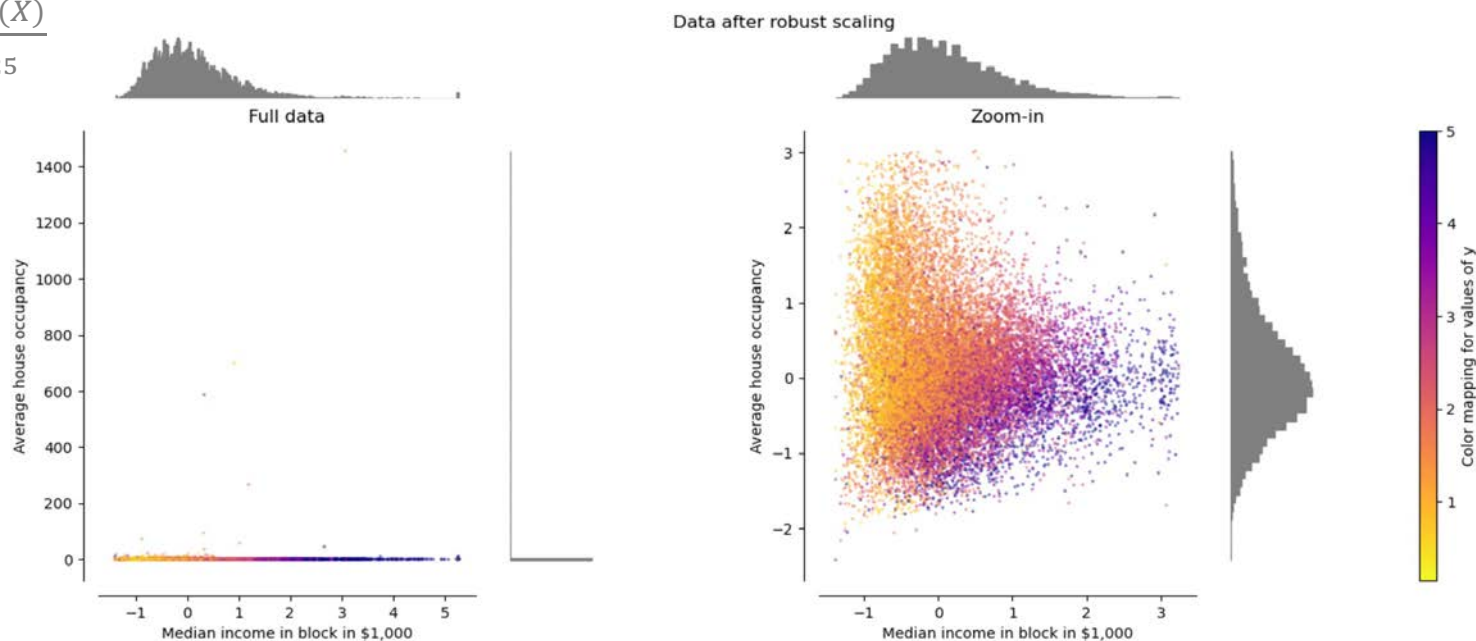
一种不受极端值影响的放缩方法  $X' = \frac{X - \text{median}(X)}{X_{q75} - X_{q25}}$

线性变化

不改变数据分布

不能改变极端值，但数据不会拥挤

极端值加入不会无法取值（过曝）





## 放缩方法（4）：非线性变化

### 非线性：

只保持序数性质： $x > y \Leftrightarrow f(x) > f(y)$

但是无法保证线性性质

$f(x + y) \neq f(x) + f(y)$   $f(nx) \neq nf(x)$

为什么对于线性方法很重要？

### 幂分布

数据严格为正 Box-Cox

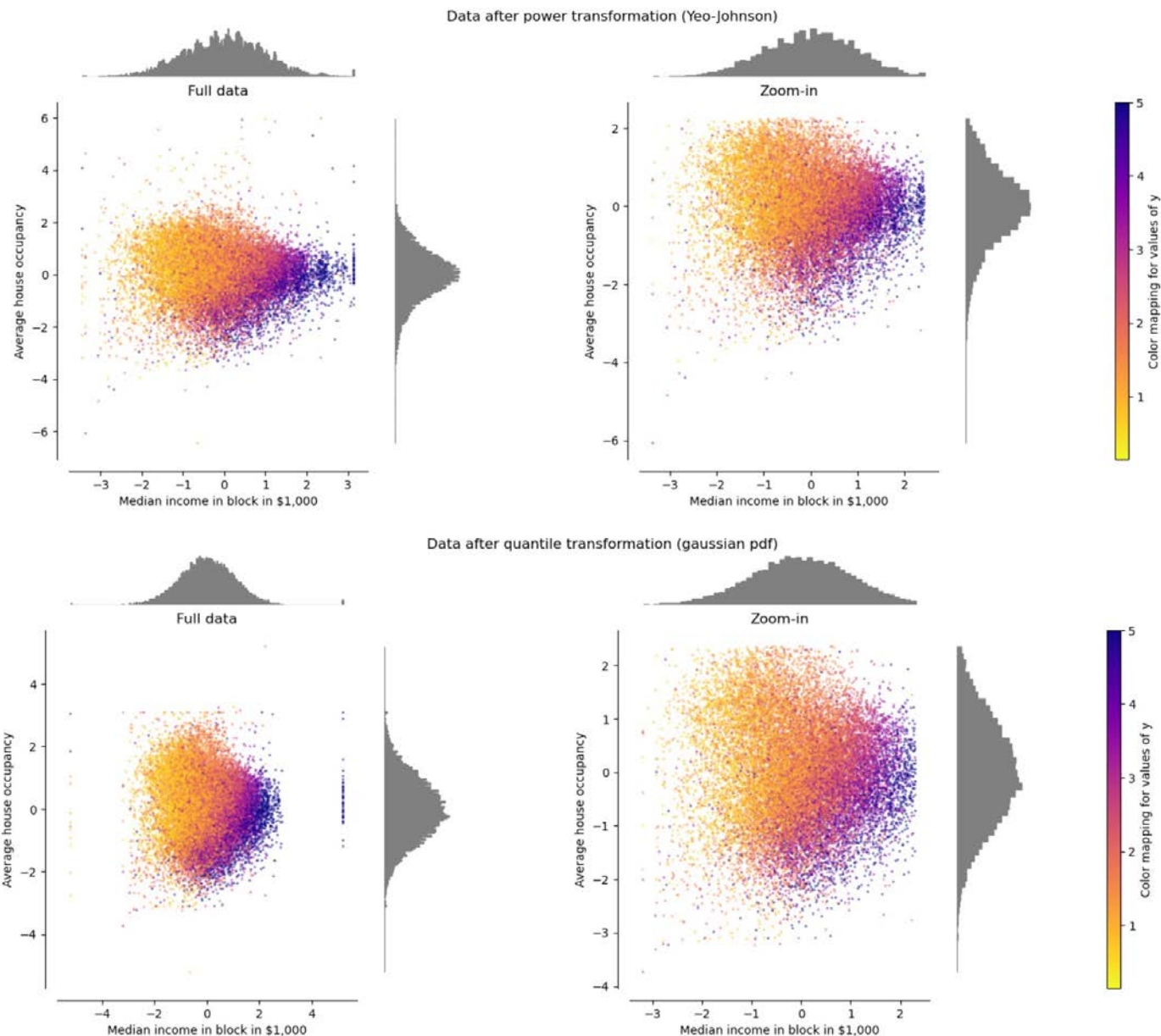
数据可能为负 Yeo-Johnson

### 分位数变换

将数据依据分位数强行变化成高斯分布

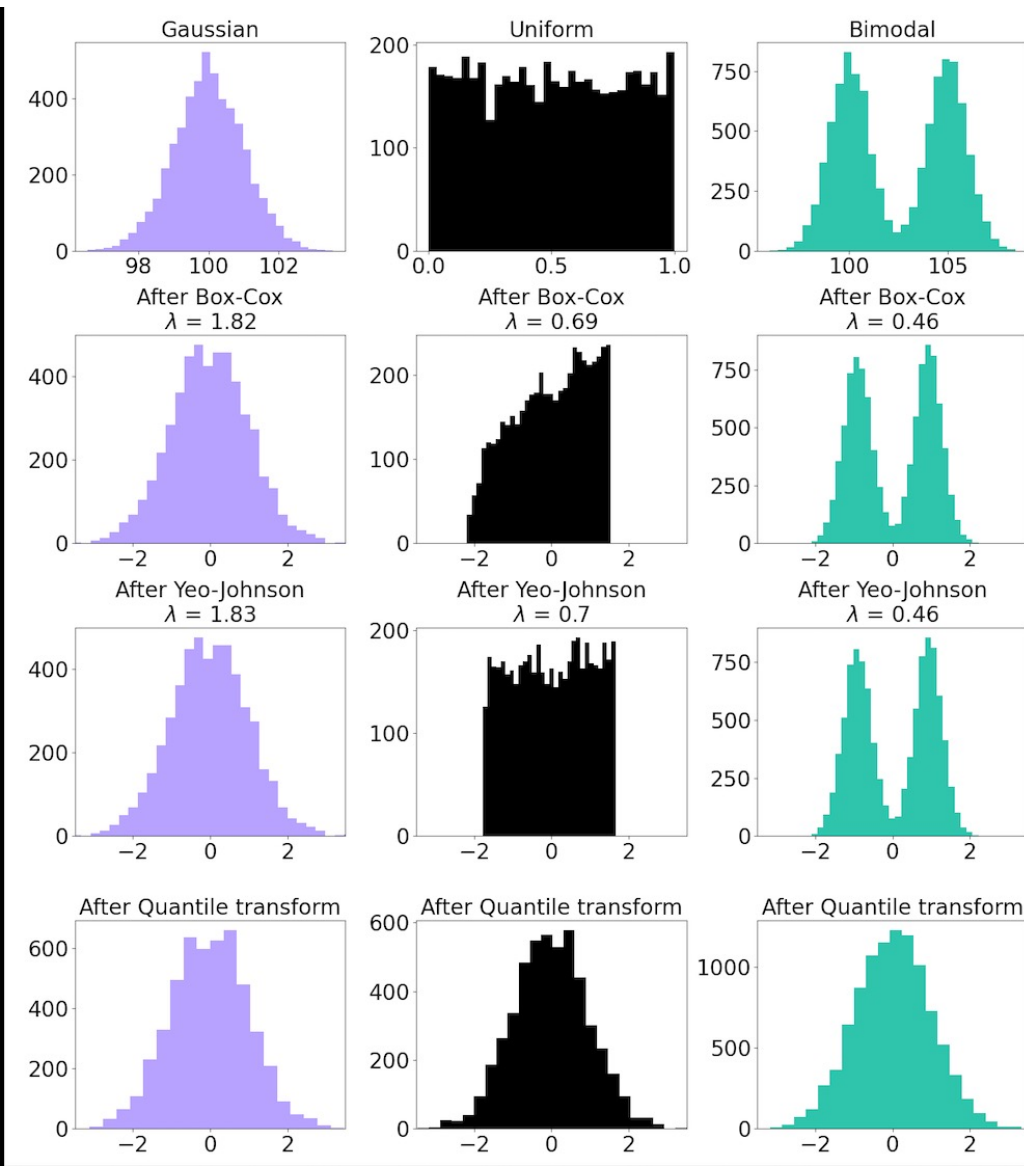
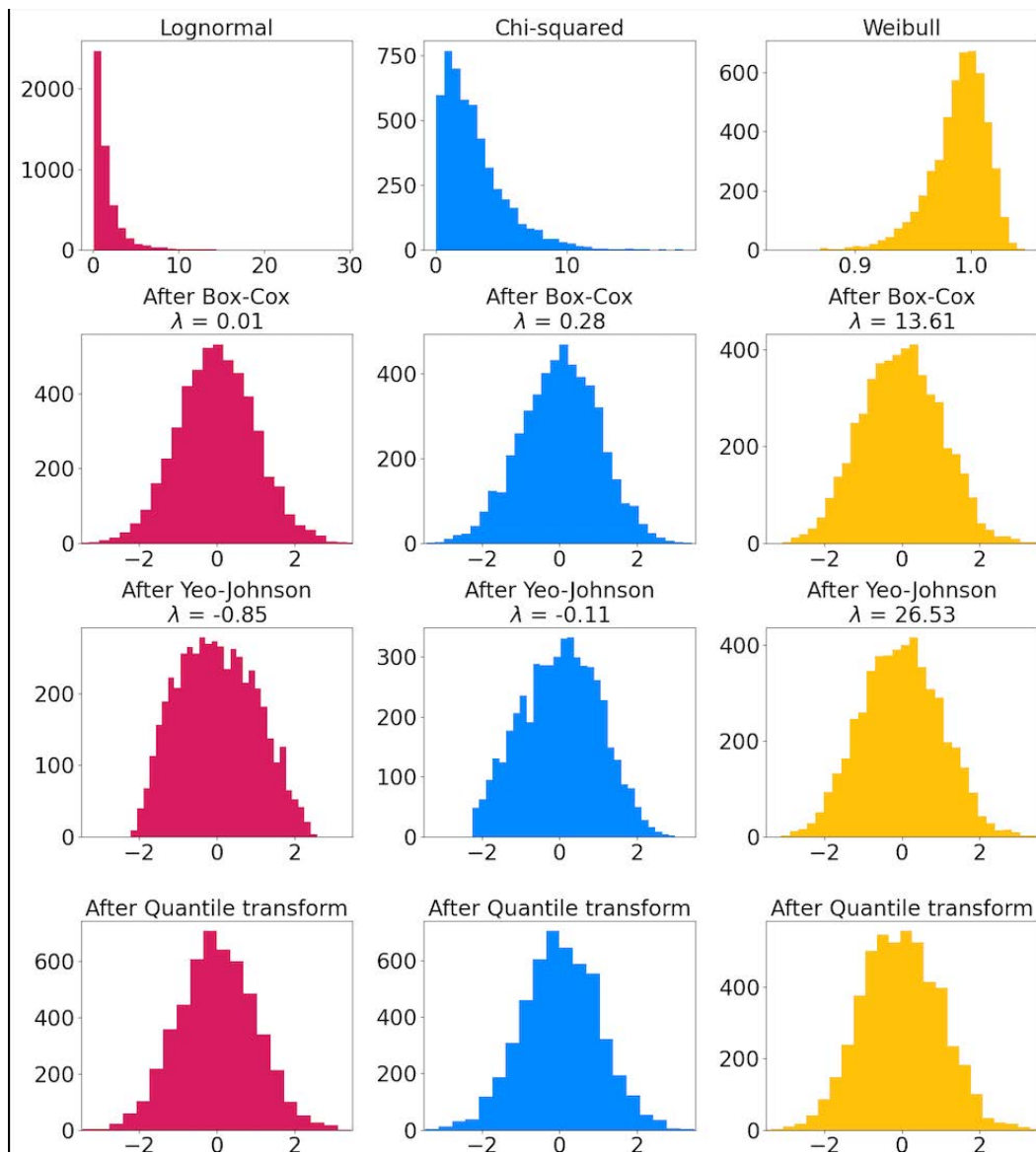
过于强力，样本量小于1000别用

如果其他方法试用，则不用



## 3.3

## 非线性变化：不同情况下的适应



# 连续值变成分段值

## Kbin: 把数据分为k个罐子

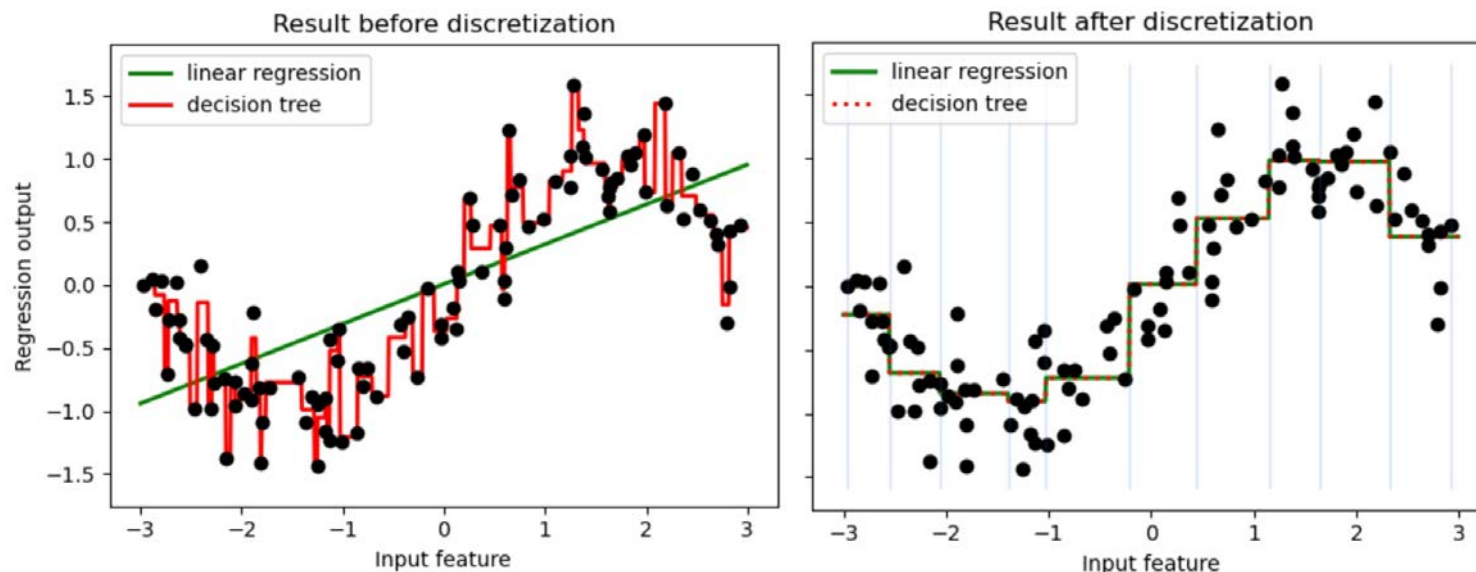
$n\_bins = 5$  分为几个罐子

Strategy = quantile 相同个数 uniform 同宽分段 kmeans 一维kmeans聚类

例如, 把百分制换为ABCD成绩档/显示排名分位数/赋分制

#这两句为变形代码, 分为十个bin, 同时使用onehot代码

```
enc = KBinsDiscretizer(n_bins=10, encode="onehot")
X_binned = enc.fit_transform(X)
```



## 增强稳健性 & 平衡算法能力 & 优化分类模型

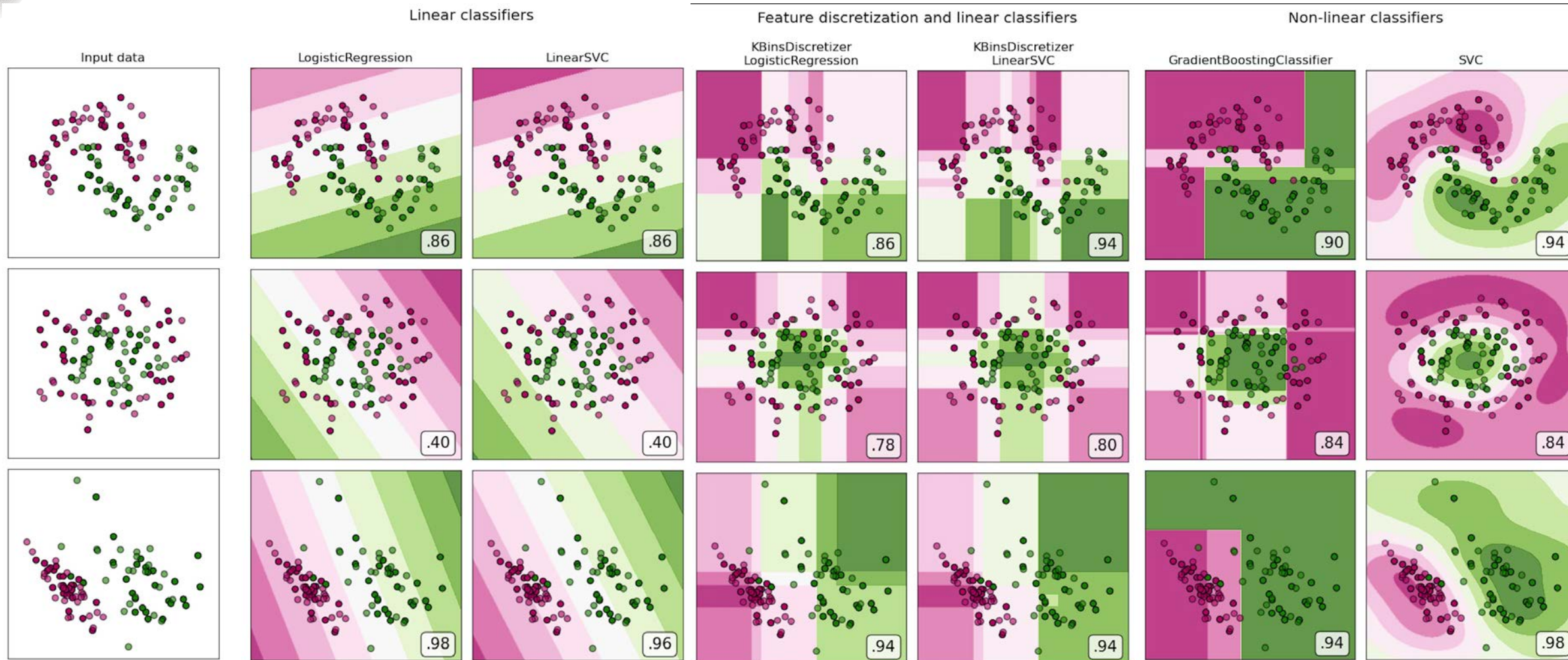
稳健性: 异常值、非线性;

增强线性算法, 弱化非线性算法

分类算法: 避免线性和logit的弱点, 征信模型



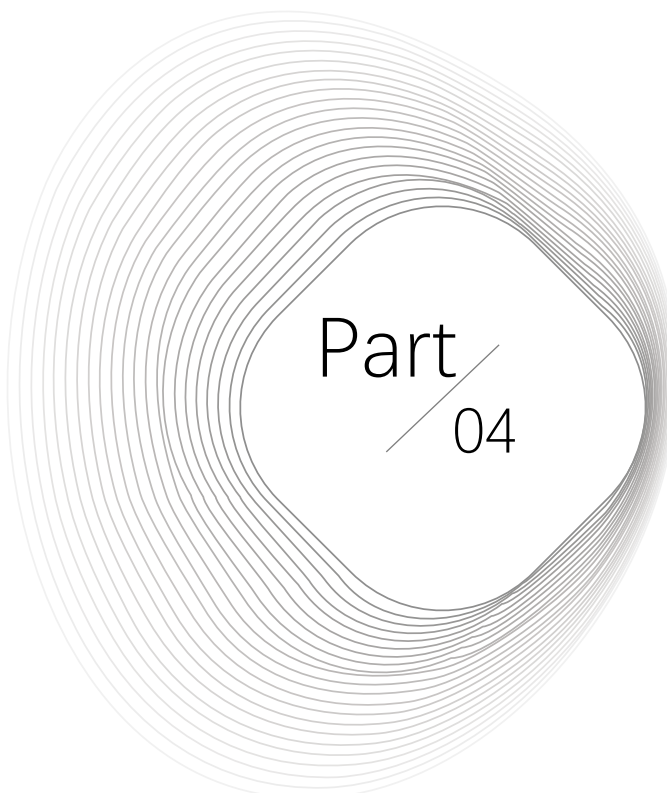




**Kbin（非线性变化）对于线性算法很重要，对于复杂方法不重要**

模型复杂度：可解释性、直观性、启动数据、计算力、训练方向





Part  
04

# 特征选择

- 单变量选择
- 多变量选择
- 算法选择
- WOE与IV



# 为什么要进行（预先）变量选择？

## 方法制约

许多模型对于多变量的效果不好（简单线性模型、树模型）

## 解释制约

当我们向别人解释一个模型的时候，变量数过多削弱解释力

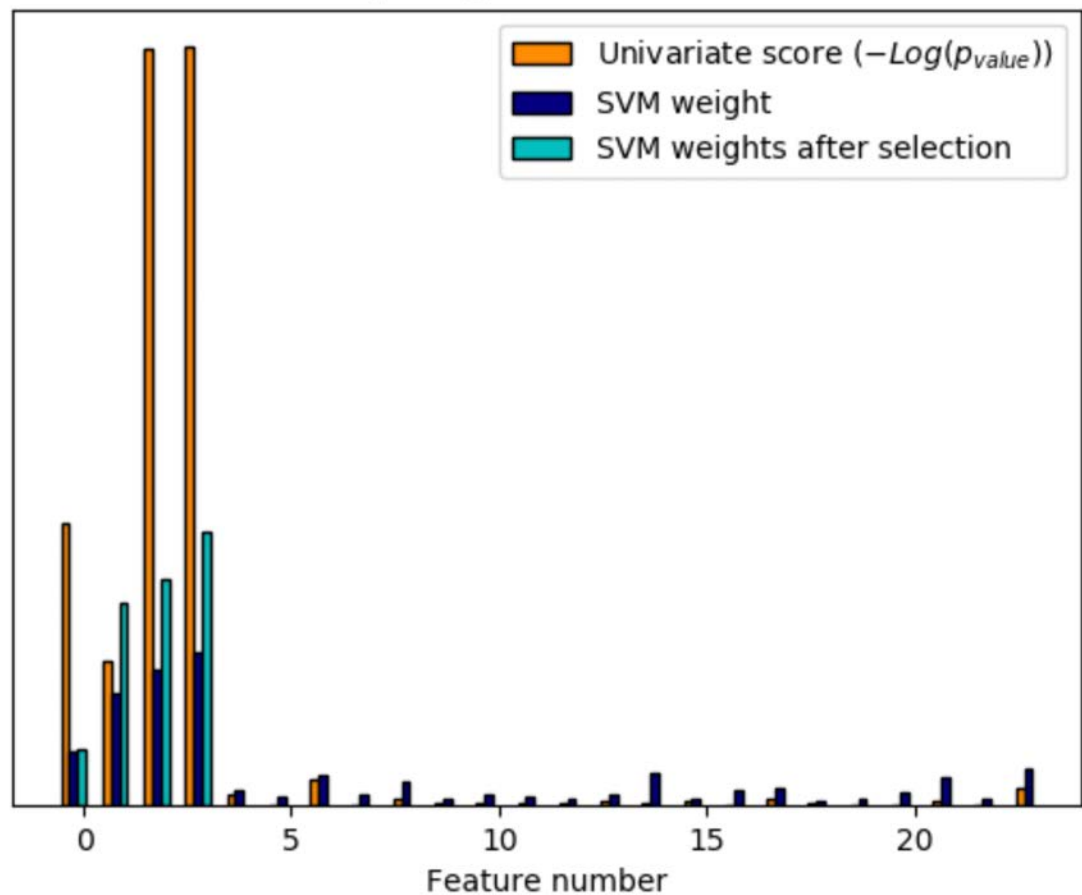
## 预测制约

预测方差、对极端值的敏感程度、随时间的模型衰弱

## 实践制约

变量数如果过大、模型如果过度复杂，对于计算压力比较大

Comparing feature selection



[https://scikit-learn.org/stable/modules/feature\\_selection.html](https://scikit-learn.org/stable/modules/feature_selection.html)



# 单变量选择

## 方差筛选：单纯地从一个变量数据的分布进行筛选

一个变量有用：variation大

常用阈值：0

## 解释力筛选：单变量x对于y的解释性

[SelectKBest](#)：筛选出K个得分最高的

[SelectPercentile](#)：筛选高于某个阈值的

用于回归方法的分布：[r\\_gression](#),

[f\\_regression](#), [mutual info regression](#)

用于分类方法的分布：

[chi2](#), [f\\_classif](#), [mutual info classif](#)

## 1.2 评分选择

```
In [26]: from sklearn.feature_selection import SelectKBest
          from sklearn.feature_selection import r_regression
          uni_kbest = SelectKBest(r_regression, k=4)
          X_sel_uniscore = uni_kbest.fit_transform(X_full, y_full)
```

```
In [27]: X_sel_uniscore.shape
```

```
Out[27]: (20226, 4)
```

```
In [31]: for i in range(len(feature_names)):
          print(feature_names[i], uni_kbest.scores_[i])
```

```
MedInc 0.6979380404221668
HouseAge 0.10259838052973475
AveRooms 0.1573837928644086
AveBedrms -0.047668303659016316
Population -0.024650453380082885
AveOccup -0.28261728437724043
Latitude -0.148334660582551
Longitude -0.04355869312189106
```



## 多变量选择

### 多重共线性的影响

对于模型的损害；数据的特征；程序包与软件的区别；

### 如何批量去除多重共线性？

VIF variance inflation factor 方差膨胀系数

$$VIF_i = \frac{1}{1 - R_i^2}$$

**实践：当VIF大于10（or 5）时，可以删去**

一方面看指标，一方面看变量本身的意义

[https://etav.github.io/python/vif\\_factor\\_python.html](https://etav.github.io/python/vif_factor_python.html)





4.3

## 算法选择

### 最稳健的方法

实践是检验真理的唯一标准

[SelectFromModel](#)统一的接口

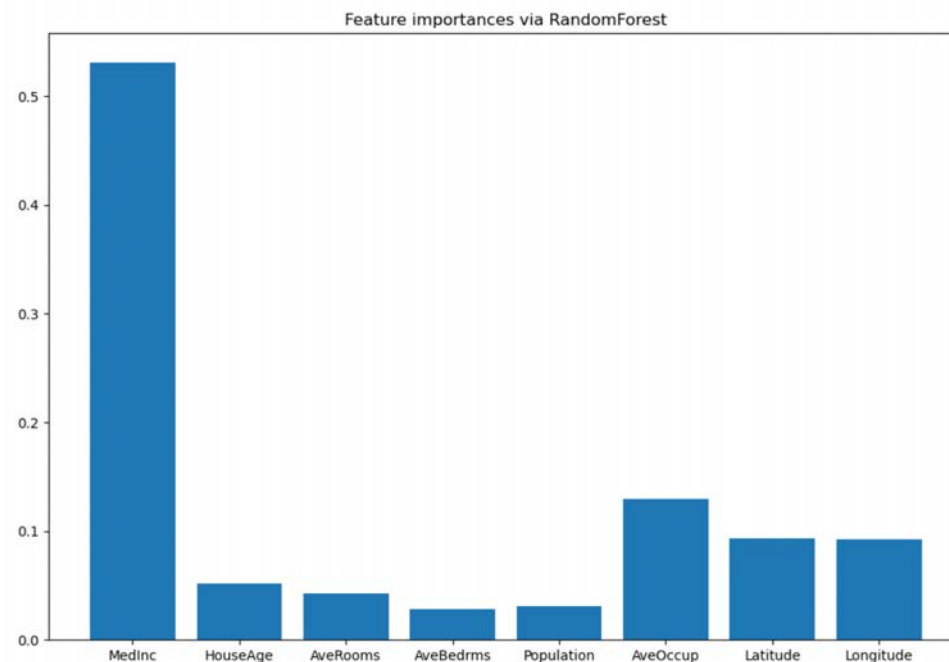
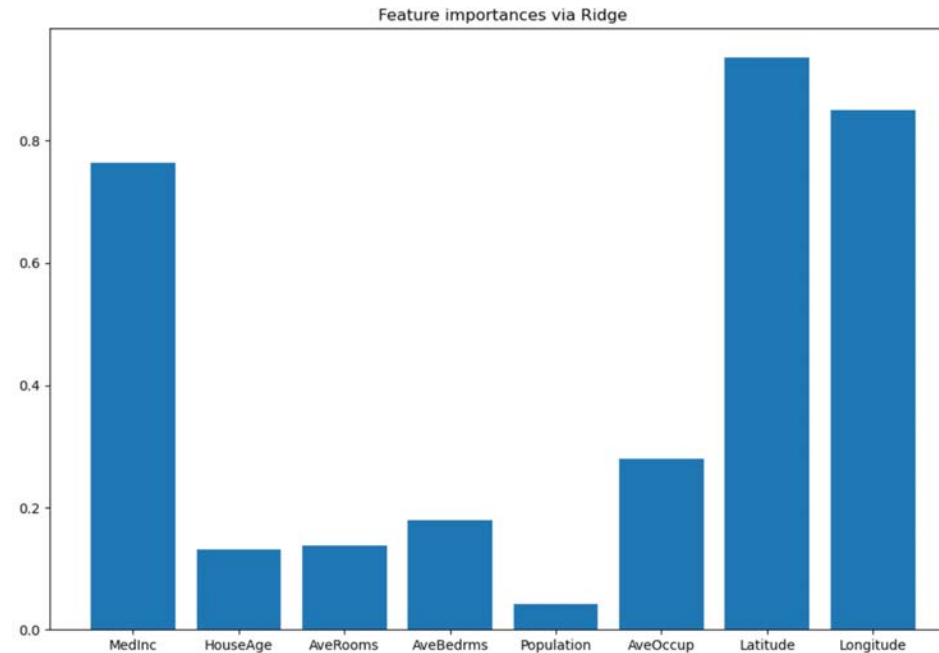
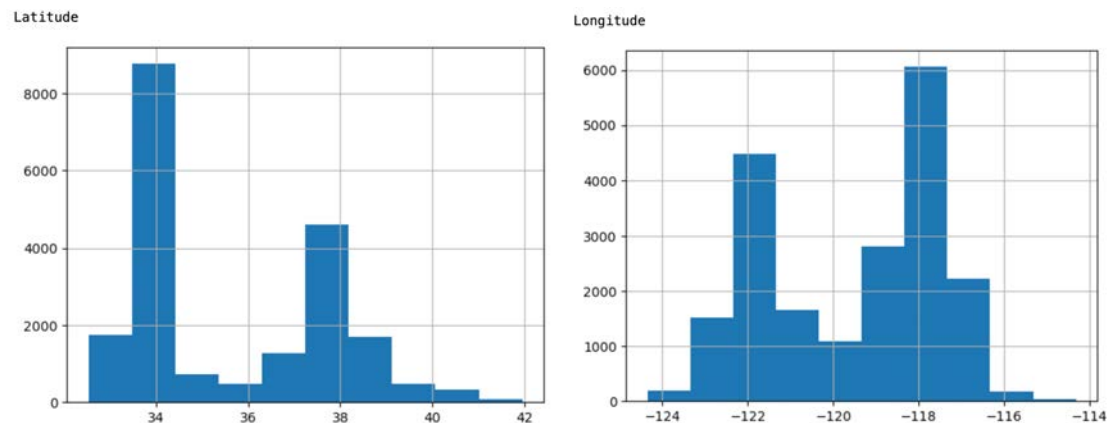
可以参考随堂代码

### 后台的算法

L1-based

Support vector based

Tree based



## 实践：金融风控

### 方法基础：评分卡模型、logistics 回归

后续很多操作，都是因为这模型框架的局限

### 变量处理：离散化

线性模型无法搞定非线性连续数据；避免极端值的影响；抵抗时间流逝的影响

### 变量筛选：单变量、多变量

Chi-2的单变量筛选，VIF删除多重共线性

### 重要方法：WOE与IV

从指标生成到变量重要性，为二分类而生

WOE weight of evidence 证据权重

IV Information Value 信息量



(2) 最近一次购买金额:

最近一次购买金额	响应	未响应	合计	响应比例
<100 元	2500	47500	50000	5%
[100,200)	3000	27000	30000	10%
[200,500)	3000	12000	15000	20%
>=500 元	1500	3500	5000	30%
合计	10000	90000	100000	10%

$$WOE_i = \ln \left( \frac{\Pr(\text{yes in bin}_i | \text{yes})}{\Pr(\text{no in bin}_i | \text{no})} \right) = \ln \left( \frac{\frac{\#y_i}{\#y_T}}{\frac{\#n_i}{\#n_T}} \right) = \ln \left( \frac{\frac{\#y_i}{\#n_i}}{\frac{\#y_T}{\#n_T}} \right)$$

$$<100 \text{ 元} : WOE_1 = \ln \left( \frac{2500/47500}{10000/90000} \right) = -0.74721$$

$$[100,200) : WOE_2 = \ln \left( \frac{3000/27000}{10000/90000} \right) = 0$$

$$[200,500) : WOE_3 = \ln \left( \frac{3000/12000}{10000/90000} \right) = 0.81093$$

$$>=500 \text{ 元} : WOE_4 = \ln \left( \frac{1500/3500}{10000/90000} \right) = 1.349927$$

- 当前分组中，响应的比例越大，WOE值越大；
- 当前分组WOE的正负，由当前分组响应和未响应的比例，与样本整体响应和未响应的比例的大小关系决定，当前分组的比例小于样本整体比例时，WOE为负，当前分组的比例大于整体比例时，WOE为正，当前分组的比例和整体比例相等时，WOE为0。
- WOE的取值范围是全体实数。

最近一次购买金额	响应	未响应	合计	响应比例	WOE
<100 元	2500	47500	50000	5%	-0.74721
[100,200)	3000	27000	30000	10%	0
[200,500)	3000	12000	15000	20%	0.81093
>=500 元	1500	3500	5000	30%	1.349927
合计	10000	90000	100000	10%	0

$$IV_i = (\Pr(\text{yes in bin}_i | \text{yes}) - \Pr(\text{no in bin}_i | \text{no})) * WOE_i$$

$$IV_i = \left( \frac{\#y_i}{\#y_T} - \frac{\#n_i}{\#n_T} \right) * WOE_i \quad \text{IV取值范围为}[0, +\infty)$$

IV范围	预测效果	英文描述
小于0.02	几乎没有	Useless for prediction
0.02~0.1	弱	Weak predictor
0.1~0.3	中等	Medium predictor
0.3~0.5	强	Strong predictor
大于0.5	难以置信，需确认	Suspicious or too good to be true

$$<100 \text{ 元} : IV_1 = \left( \frac{2500}{10000} - \frac{47500}{90000} \right) * \ln \left( \frac{\frac{2500}{10000}}{\frac{47500}{90000}} \right) = 0.20756$$

$$[100,200) : IV_2 = \left( \frac{3000}{10000} - \frac{27000}{90000} \right) * \ln \left( \frac{\frac{3000}{10000}}{\frac{27000}{90000}} \right) = 0$$

$$[200,500) : IV_3 = \left( \frac{3000}{10000} - \frac{12000}{90000} \right) * \ln \left( \frac{\frac{3000}{10000}}{\frac{12000}{90000}} \right) = 0.135155$$

$$>=500 \text{ 元} : IV_4 = \left( \frac{1500}{10000} - \frac{3500}{90000} \right) * \ln \left( \frac{\frac{1500}{10000}}{\frac{3500}{90000}} \right) = 0.149992$$

$$IV_{var} = IV_1 + IV_2 + IV_3 + IV_4$$

# 总结讨论

## 首先，今天讲的只是技术性皮毛

明确原则，尊重现实，好的feature需要大量的时间与数据才能喂出来  
不符合原则的特征可能一时好用，但不可能永远好用，巨大收益必然伴随巨大风险

## 其次，特征工程看起来愈发不重要

数据分布只有部分方法关心，k-bin在rbf-svm、树方法中都不重要  
标准化对于树方法无效，深度学习自己就自带特征工程

## 再次，特征工程在业界依然重要

一方面要明白，feature 本身的重要性，对于经典feature的充分尊重  
另一方面要明白，为什么看起来silly的方法依然主导实践与研究  
稳定性；白盒与易解释性；和既往标准的比较（冷启动）；在时间衰减中的抵抗（热维护）；话语沟通

## 最后，君不密则失臣，臣不密则失身

这是一个大的方法体系，特别是在合作、对别人的钱负责的时候  
想明白这个项目是对什么、谁在负责  
金融永远是人的事情

