COSC 3337 : Data Science I

N. Rizk

College of Natural and Applied Sciences

Department of Computer Science

University of Houston

Why Data Preprocessing is Beneficial to Data Science ?

- Less data
  - data mining methods can learn faster
- Higher accuracy
  - data mining methods can generalize better
- Simple results
  - they are easier to understand
- Fewer attributes
  - For the next round of data collection, saving can be made by removing redundant and irrelevant features

Data preprocessing

Data cleaning

- **Missing values**
  - Use the most probable value to fill in the missing value (and five other methods)
- **Noisy data**
  - Binning; Regression; Clusttering

Data integration

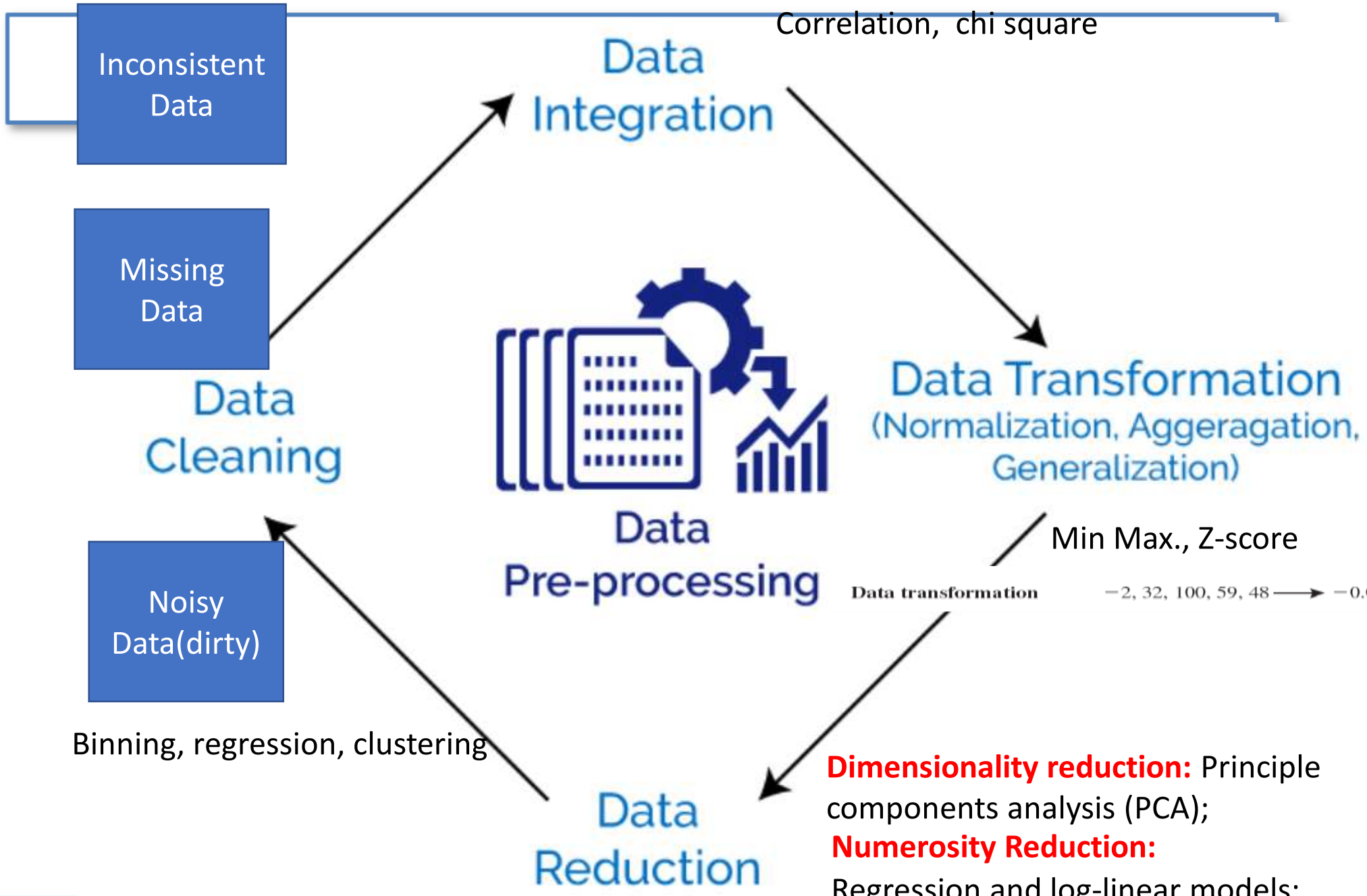
- **Entity ID problem**
  - Metadata
- **Redundancy**
  - Correlation analysis (Correlation coefficient, chi-square test)

Data trasnformation

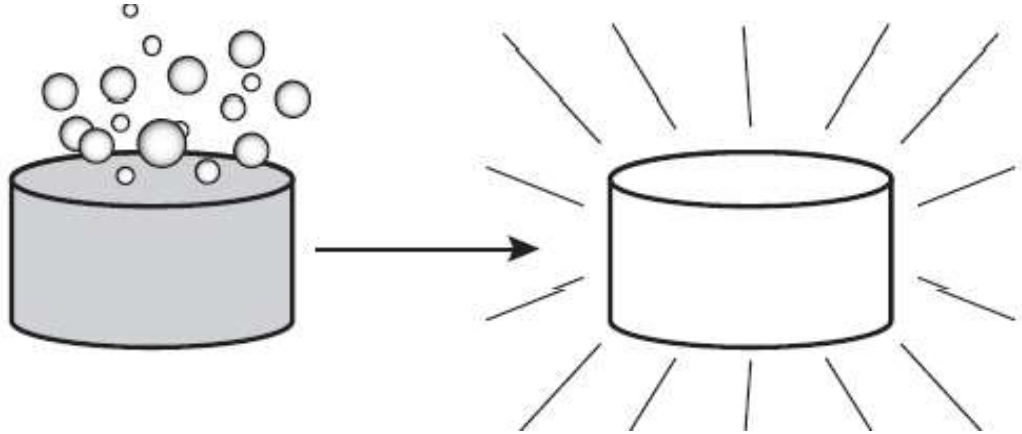
- **Smoothing**
  - **Data cleaning**
- **Aggregation**
  - **Data reduction**
- **Generailization**
  - **Data reduction**
- **Normalization**
  - Min-max; z-score; decimal scaling
- **Attribute Construction**

Data reduction

- **Data cube aggregation**
  - Data cube store multidimensional aggregated information
- **Attribute subset selection**
  - Stepwise forward selection; stepwise backward selection; combination; decision tree induction
- **Dimensionality reduction**
  - Discrete wavelet trasnforms (DWT); Principle components analysis (PCA);
- **Numerosity Reduction**
  - Regression and log-linear models; histograms; clustering; sampling
- **Data discretization**
  - Binning; historgram analysis; entropy-based discretization;
  - Interval merging by chi-square analysis; cluster analysis; intuitive partitioning
- **Concept hierarchy**


Inconsistent Data
Missing Data
Noisy Data(dirty)
Data Cleaning
Data Integration
Correlation, chi square
Data Pre-processing
Data Transformation (Normalization, Aggeragation, Generalization)
Min Max., Z-score
Data transformation
−2, 32, 100, 59, 48 ⟶ −0.02, 0.32, 1.00, 0.59, 0.4
Binning, regression, clustering
Data Reduction
Dimensionality reduction: Principle components analysis (PCA);
Numerosity Reduction:
Regression and log-linear models; histograms; clustering; sampling

Data cleaning



Preprocessing: Missing Data

There are various ways for us to handle it

1.Ignore
2.Fill manually
3.Imputation replacing missing value

Preprocessing: Missing Data

- Handling Null Values
- Handling Categorical Variables
- One-Hot Encoding
- Multicollinearity
- Standardization

Missing Data: **Handling Null Values**

```
#check if there is null values
df.isnull()
# Returns a boolean matrix, if the value is NaN then
True otherwise False
df.isnull().sum()
# Returns the column names along with the number of
NaN values in that particular column
```

df.dropna()

Missing Data: **Imputation replacing missing value**

Perform imputation by using the SimpleImputer class provided by sklearn.
from sklearn.imput

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer = imputer.fit(df[['Weight']])
df['Weight'] = imputer.transform(df[['Weight']])
```

Handling Categorical Variables

Categorical variables are basically the variables that are discrete and not continuous. Ex — color of an item is a discrete variables where as its price is a continuous variable.

Categorical variables are further divided into 2 types —

1. Ordinal categorical variables — These variables can be ordered. Ex — Size of a T-shirt. We can say that M<L<XL.
2. Nominal categorical variables — These variables can't be ordered. Ex — Color of a T-shirt. We can't say that Blue<Green as it doesn't make any sense to compare the colors as they don't have any relationship.

Handling Ordinal Categorical Variables –

First of all, we need to create a dataframe.

```
df_cat = pd.DataFrame(data =
                [['green','M',10.1,'class1'],
                 ['blue','L',20.1,'class2'],
                 ['white','M',30.1,'class1']])
df_cat.columns = ['color','size','price','classlabel']
```

Here the columns 'size' and 'classlabel' are ordinal categorical variables whereas 'color' is a nominal categorical variable.

There are 2 pretty simple and neat techniques to transform ordinal CVs.

1. Using map() function —

```
size_mapping = {'M':1,'L':2}
df_cat['size'] = df_cat['size'].map(size_mapping)
```

Here M will be replaced with 1 and L with 2.

2. Using Label Encoder —

```
from sklearn.preprocessing import LabelEncoder
class_le = LabelEncoder()
df_cat['classlabel'] =
class_le.fit_transform(df_cat['classlabel'].values)
```

Here class1 will be represented with 0 and class2 with 1 .

CategoricalEncoder + ColumnTransformer

```
categorical = df.dtypes == object

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

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

Nominal Categorical Variables —

The correct way of handling nominal CVS is to use One-Hot Encoding. The easiest way to use One-Hot Encoding is to use the get_dummies() function.

df_cat = pd.get_dummies(df_cat[['color','size','price']])

```
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
		142
		54
		63
		219

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

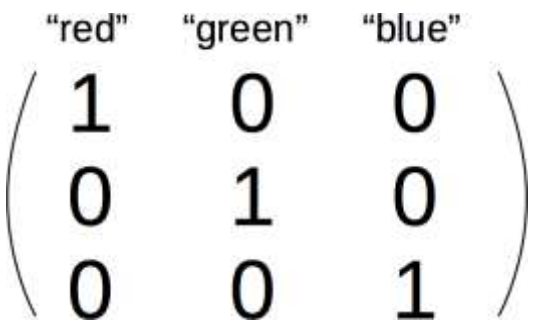
	boro	salary
0	0	103
1	1	89
2	0	142
3	2	54
4	2	63
5	3	219

```
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

One-Hot Encoding —

- One-Hot Encoding what we essentially do is that we create ’n’ columns where n is the number of unique values that the nominal variable can take.
- Ex — Here if color can take Blue,Green and White then we will just create three new columns namely — color_blue,color_green and color_white and if the color is green then the values of color_blue and color_white column will be 0 and value of color_green column will be 1 .
- So out of the n columns only **one column can have value = 1** and the rest all will have value = 0.



- One-Hot Encoding is a pretty cool and neat hack but there is only one problem associated with it and that is Multicollinearity.

OneHotEncoder

```
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.]])
```

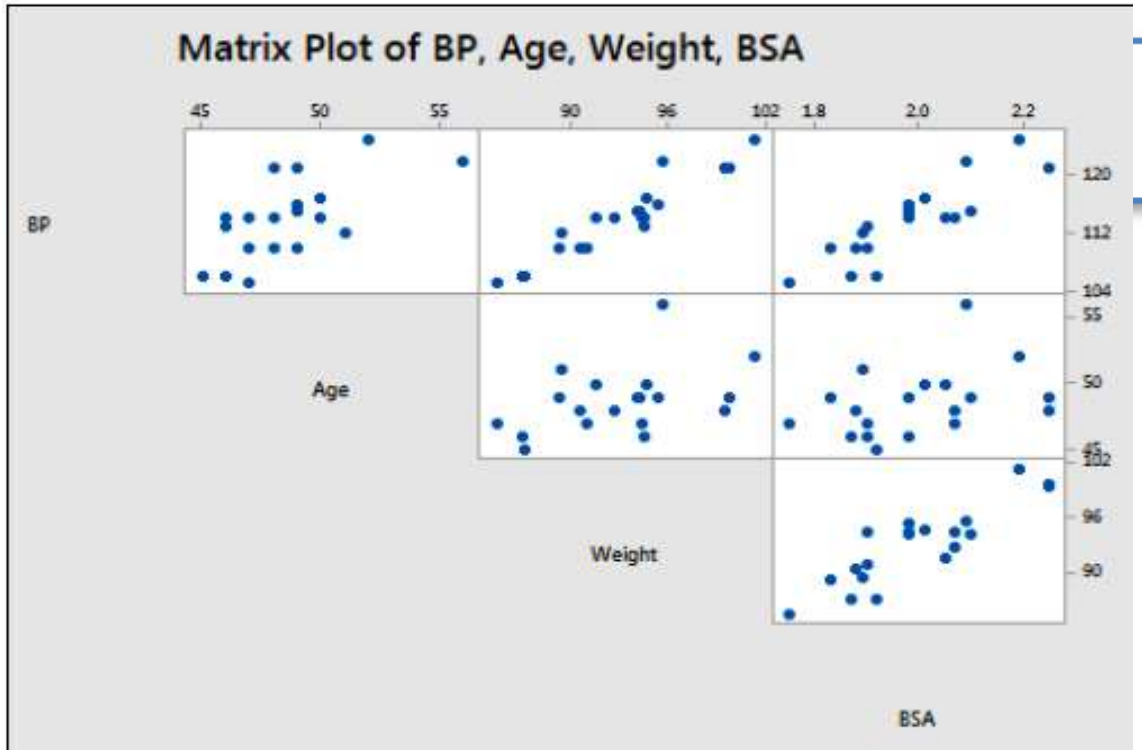
Multicollinearity and its impact —

Multicollinearity occurs in our dataset when we have features which are strongly dependent on each other. Ex- In this case we have features - color_blue,color_green and color_white which are all dependent on each other and it can impact our model.

If we have multicollinearity in our dataset **then we won't be able to use our weight vector to calculate the feature importance**.

Multicollinearity impacts the interpretability of our model.

The easiest method to identify Multicollinearity is to just plot a pairplot



Simple hack to avoid Multicollinearity-
We can use drop_first=True in order to avoid the problem of Multicollinearity

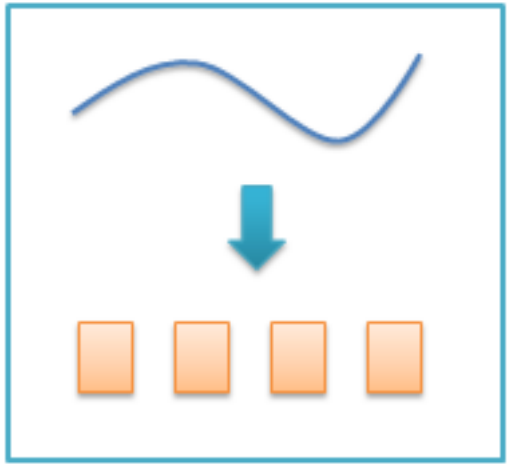
```
df_cat = pd.get_dummies(df_cat[['color','size','price']],drop_first=True)
```

we can infer the whole information with the help of only these 2 columns, hence the strong co-relation between these three columns is broken.

Noisy Data: **Binning**

Binning or discretization is the process of transforming numerical variables into categorical counterparts. An example is to bin values for Age into categories such as 20-39, 40-59, and 60-79.

Numerical variables are usually discretized in the modeling method based on frequency tables (e.g., decision trees).



Binning may improve accuracy of the predictive models by reducing the noise or non-linearity.

Binning allows easy identification of outliers, invalid and missing values of numerical variables.

Unsupervised Binning

Unsupervised binning methods transform numerical variables into categorical counterparts but do not use the target (class) information. Equal Width and Equal Frequency are two unsupervised binning methods.

1- Equal Width Binning

The algorithm divides the data into k intervals of equal size. The width of intervals is:

w = (max-min)/k

The interval boundaries are:min+w, min+2w, ... , min+(k-1)w

2- Equal Frequency Binning

The algorithm divides the data into k groups which each group contains approximately same number of values.

- **Data :** 0, 4, 12, 16, 16, 18, 24, 26, 28
- **Equal width**
  - Bin 1: 0, 4 [-,10)
  - Bin 2: 12, 16, 16, 18 [10,20)
  - Bin 3: 24, 26, 28 [20,+)
- **Equal frequency**
  - Bin 1: 0, 4, 12 [-, 14)
  - Bin 2: 16, 16, 18 [14, 21)
  - Bin 3: 24, 26, 28 [21,+)

For the both methods, the best way of determining k is by looking at the histogram and try different intervals or groups.

Supervised Binning

Supervised binning methods transform numerical variables into categorical counterparts and refer to the target (class) information when selecting discretization cut points.

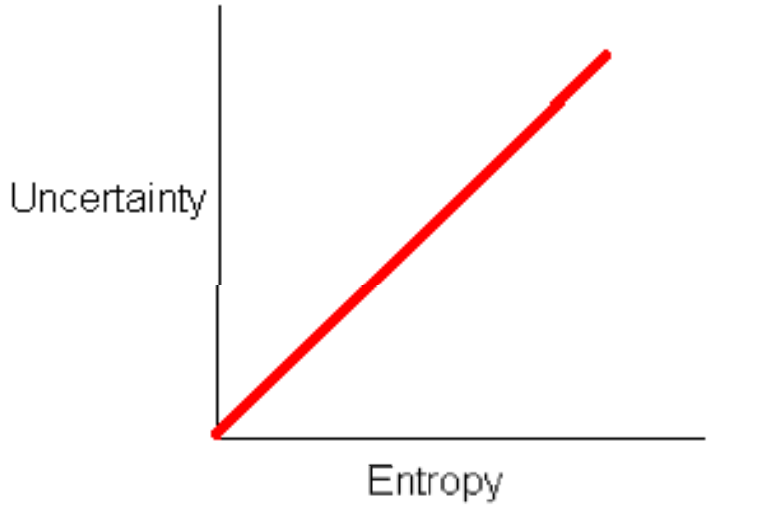
Entropy-based Binning

Entropy based method uses a split approach. The entropy (or the information content) is calculated based on the class label. Intuitively, it finds the best split so that the bins are as pure as possible that is the majority of the values in a bin correspond to have the same class label. Formally, it is characterized by finding the split with the maximal information gain.

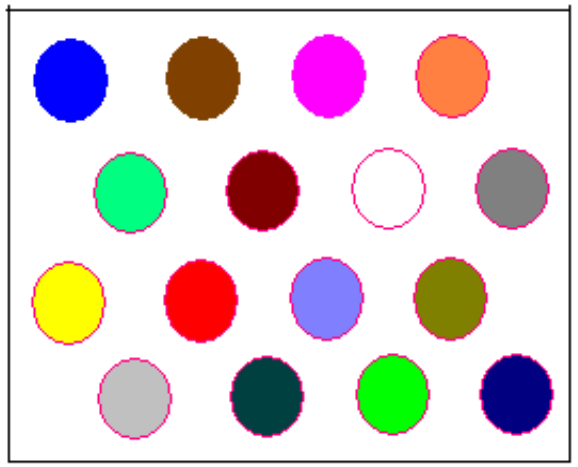
Example:
Discretize the temperature variable using entropy-based binning algorithm.

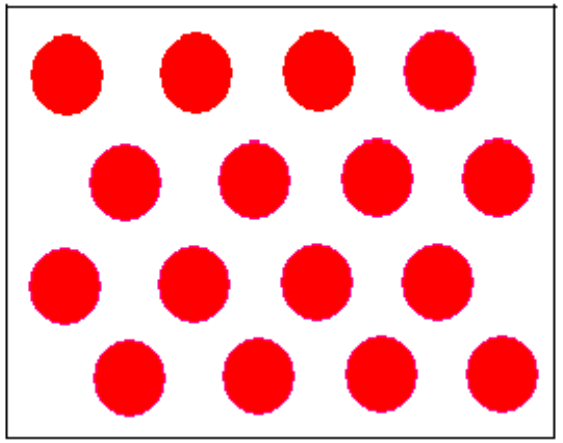
What is Entropy?

- The entropy is a measure of the uncertainty associated with a random variable
- As uncertainty and or randomness increases for a result set so does the entropy
- Values range from 0 – 1 to represent the entropy of information

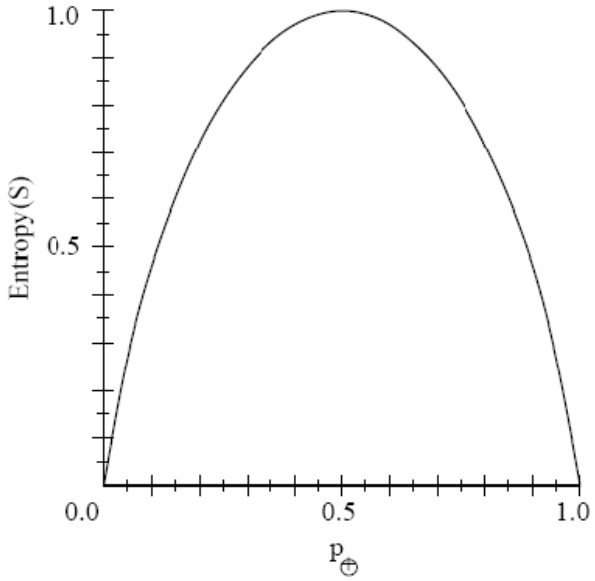


Entropy Example





$$Entropy(D) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$



Step 1: Calculate "Entropy" for $E(S) = \sum_{i=1}^{c} -p_i log_2 p_i$

O-Ring Failure	
Y	N
7	17

E (Failure) = E(7, 17) = E(0.29, .71) = -0.29 x log2(0.29) - 0.71 x log2(0.71) = 0.871

Step 2: Calculate "Entropy" for the target given a bin

$$E(S,A) = \sum_{v \in A} \frac{|S_v|}{|S|} E(S_v)$$

E (Failure,Temperature) = P(<=60) x E(3,0) + P(>60) x E(4,17) = 3/24 x 0 + 21/24 x 0.7= 0.615

		O-Ring Failure	
		Y	N
Temperature	<= 60	3	0
	> 60	4	17

Step 3: Calculate "Information Gain" given

$$\textit{Information Gain} = E(S) - E(S,A)$$

Information Gain (Failure, Temperature) = 0.256

The information gains for all three bins show that the best interval for "Temperature" is (<=60, >60) because it returns the highest gain.

O-Ring Failure	Temperature
Y	53
Y	56
Y	57
N	63
N	66
N	67
N	67
N	67
N	68
N	69
N	70
Y	70
Y	70
Y	70
N	72
N	73
N	75
Y	75
N	76
N	76
N	78
N	79
N	80
N	81

Gain = 0.256

		O-Ring Failure	
		Y	**N**
Temperature	**<= 60**	3	0
	> 60	4	17

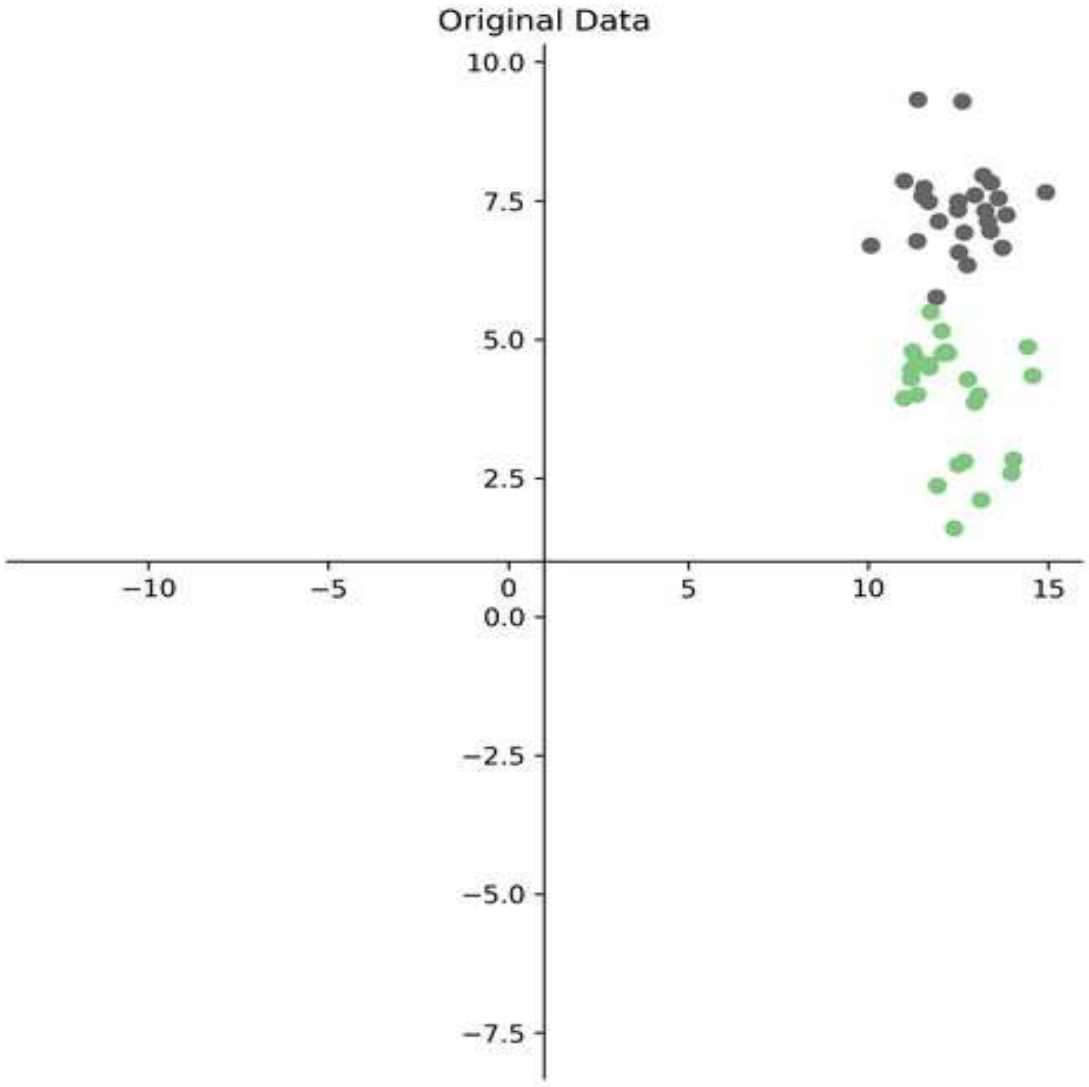
Gain = 0.101

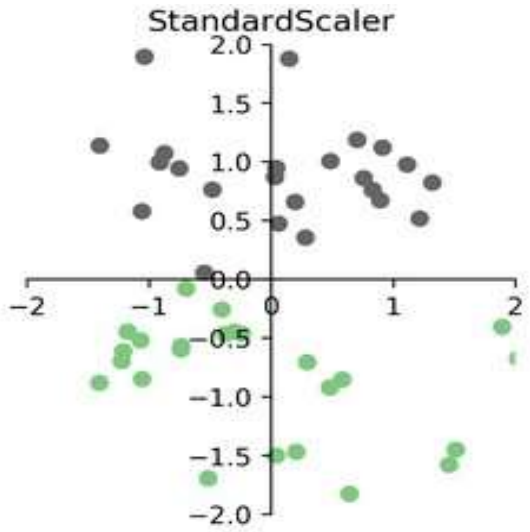
		O-Ring Failure	
		Y	**N**
Temperature	**<= 70**	6	8
	> 70	1	9

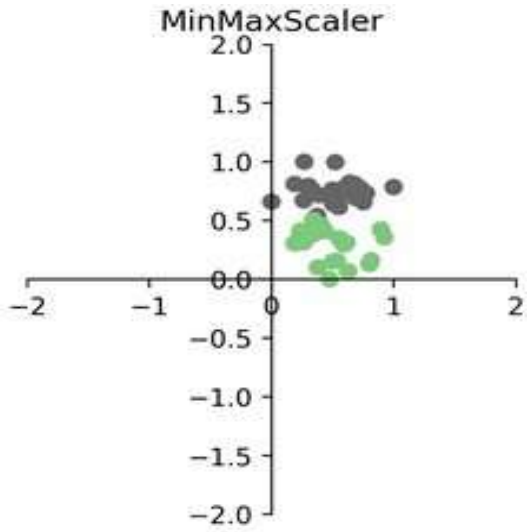
Gain = 0.148

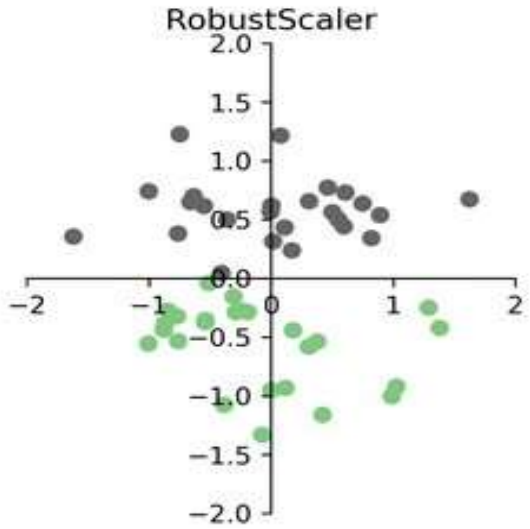
		O-Ring Failure	
		Y	**N**
Temperature	**<= 75**	7	11
	> 75	0	6

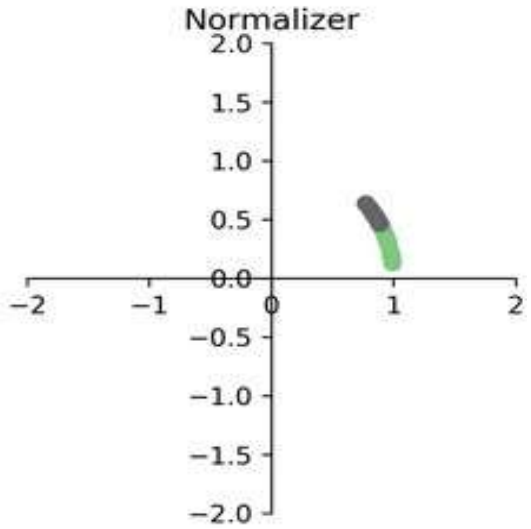
Data transformation: **Ways to scale**











Standardization — Mean 0 Sd 1

It is another integral preprocessing step.In Standardization we transform our values such that the mean of the values is 0 and the standard deviation is 1.

$$z = \frac{x_i - \mu}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
X = std.fit_transform(df[['Age','Weight']])
```

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

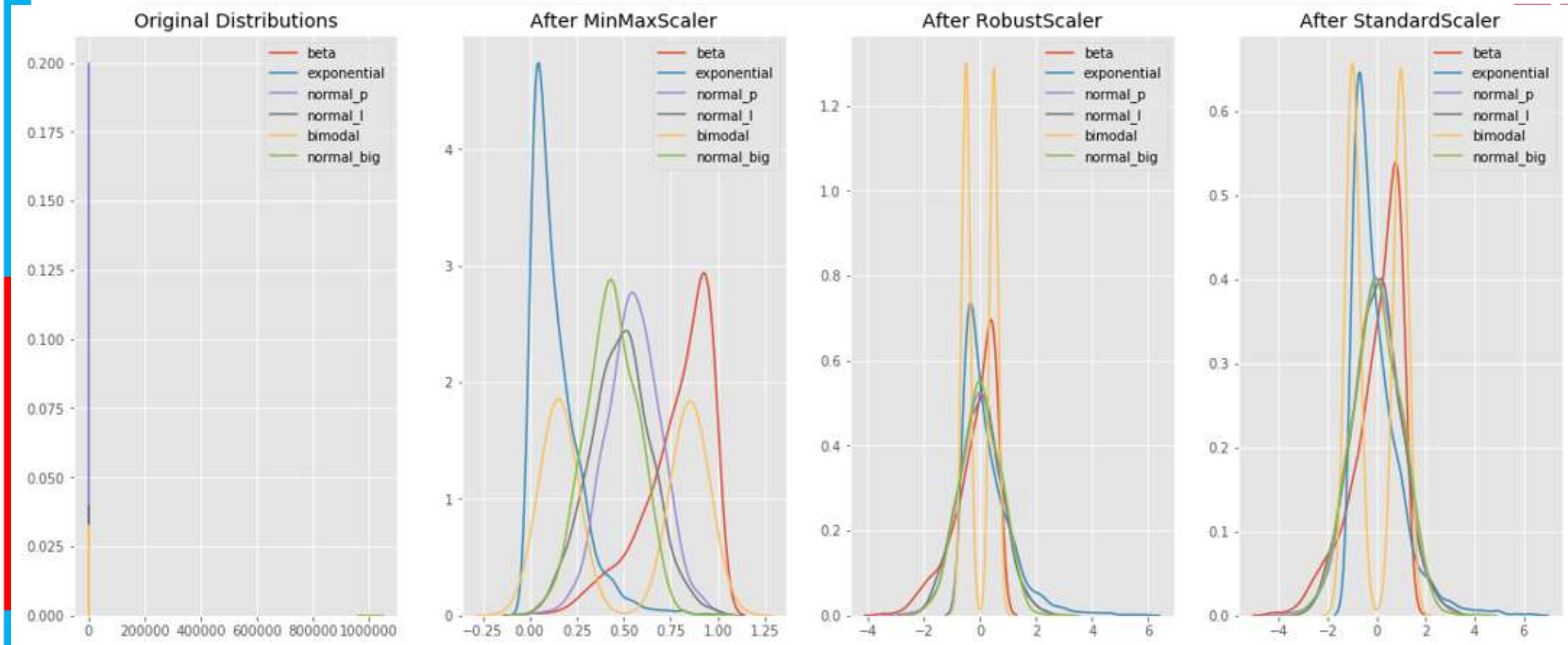
MinMAxScaler — Mean 0 Sd 1

```
from sklearn import preprocessing
```

MinMaxScaler, *RobustScaler*, *StandardScaler*, and *Normalizer are* methods to preprocess data for machine learning.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

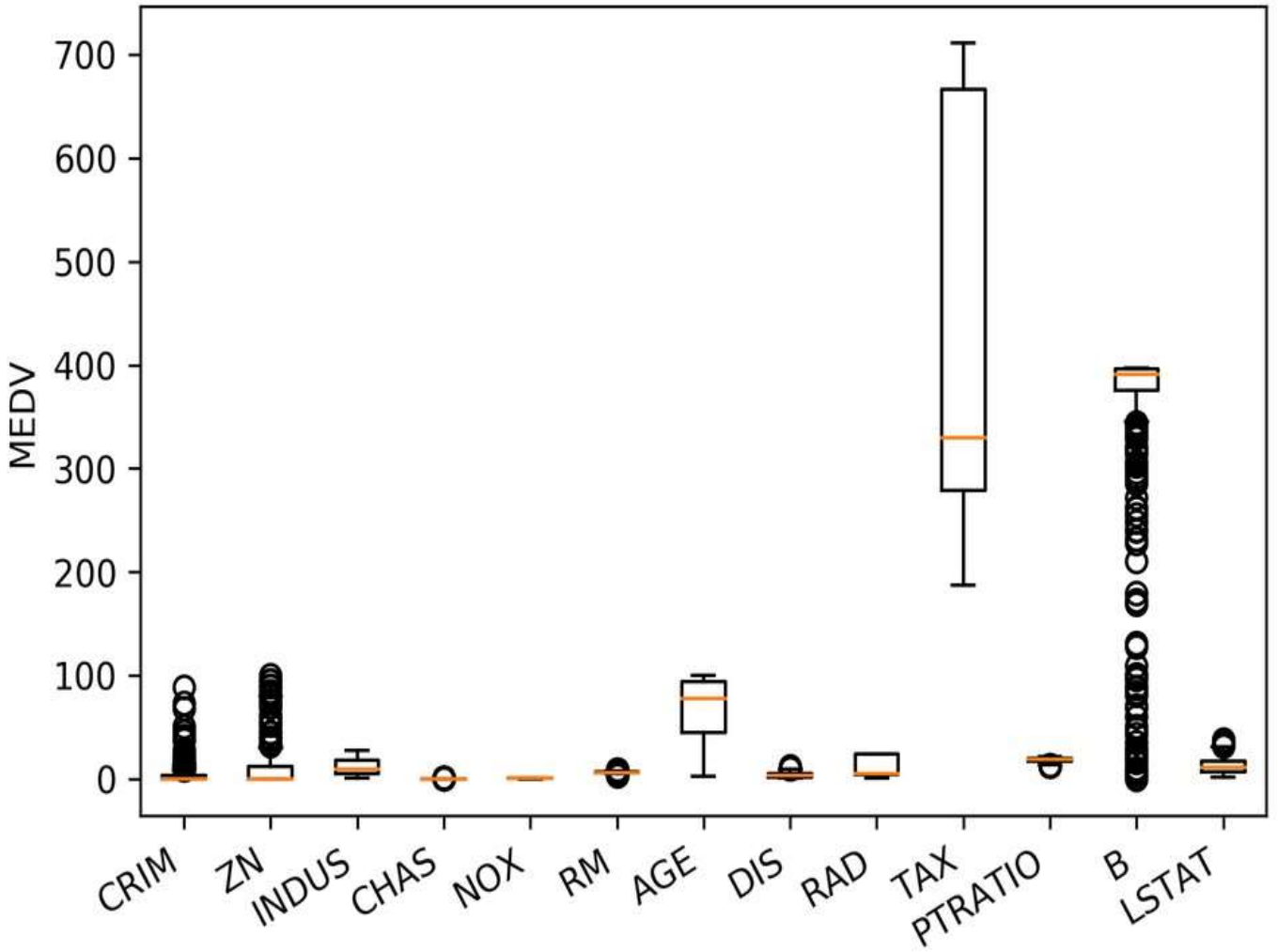
Which method you need, if any, depends on your model type and your feature values



- Use MinMaxScaler as your default
- Use RobustScaler if you have outliers and can handle a larger range
- Use StandardScaler if you need normalized features
- Use Normalizer sparingly - it normalizes rows, not columns

```
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>
```



Data Accuracy : cross_val_score

```
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)
```

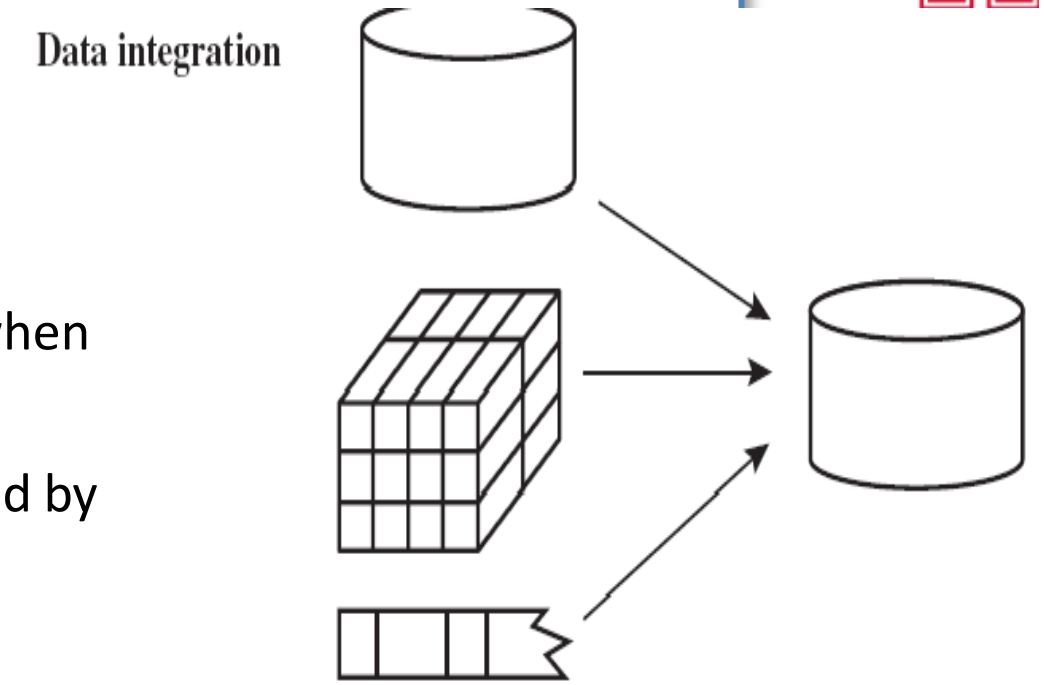
Data Integration

1-Schema integration and object matching

- *Entity identification problem*

2- Redundant data (between attributes) occur often when integration of multiple databases

- Redundant attributes may be able to be detected by correlation analysis, and chi-square method



30

Solution:Correlation, chi square

Schema Integration and Object Matching

- *custom_id* and *cust_number*
  - Schema conflict
- *"H"* and *"S"*, and *1* and *2* for *pay_type* in one database
  - Value conflict
- Solutions
  - meta data (data about data)

Detecting Redundancy

- If an attributed can be “derived” from another attribute or a set of attributes, it may be redundant

- Some redundancies can be detected by correlation analysis
  - Correlation coefficient for numeric data
  - Chi-square test for categorical data
- These can be also used for data reduction

Chi-square Test

- For categorical (discrete) data, a correlation relationship between two attributes, A and B, can be discovered by a χ^2 test
- Given the degree of freedom, the value of χ^2 is used to decide correlation based on a significance level

Chi-square Test for Categorical Data

$$\chi^2 = \sum \frac{(Observed - Expected)^2}{Expected}$$

$$\chi^2 = \sum_{i=1}^{c} \sum_{j=1}^{r} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}$$

$$e_{ij} = \frac{count(A = a_i) \times count(B = b_j)}{N}$$

The larger the X^2 value, the more likely the variables are related.

Chi-square Test

	male	*female*	**Total**
fiction	250	200	450
non_fiction	50	1000	1050
Total	300	1200	1500

contingency table
Are *gender* and *preferred_reading* correlated?

The χ^2 statistic tests the hypothesis that *gender* and *preferred_reading* are independent. The test is based on a significant level, with (r - 1) x (c - 1) degree of freedom.

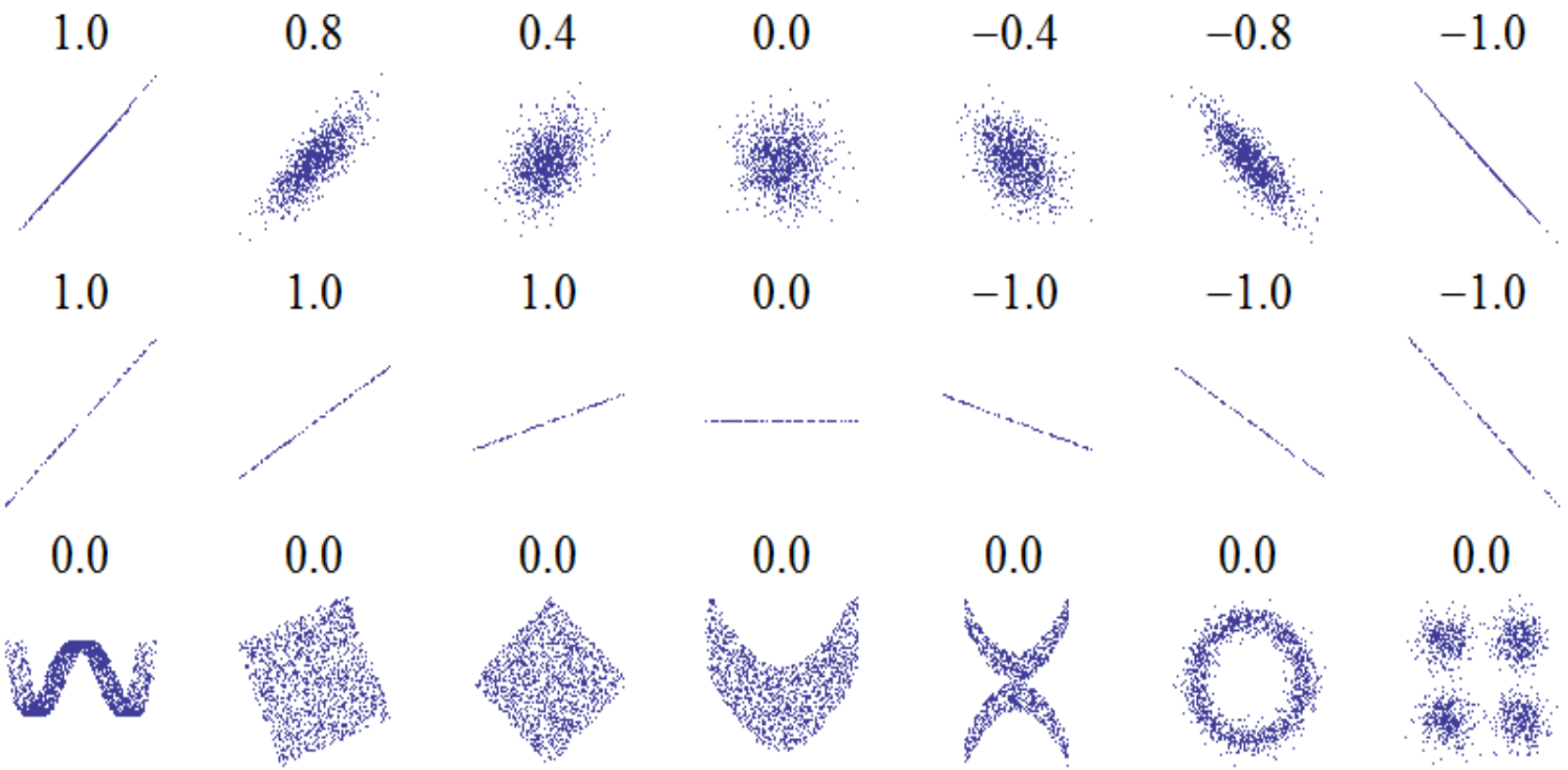
Table of Percentage Points of the χ^2 Distribution

df \ p	0.995	0.975	0.9	0.5	0.1	0.05	0.025	0.01	0.005	df
1	.000	.000	0.016	0.455	2.706	3.841	5.024	6.635	7.879	1
2	0.010	0.051	0.211	1.386	4.605	5.991	7.378	9.210	10.597	2
3	0.072	0.216	0.584	2.366	6.251	7.815	9.348	11.345	12.838	3
4	0.207	0.484	1.064	3.357	7.779	9.488	11.143	13.277	14.860	4
5	0.412	0.831	1.610	4.351	9.236	11.070	12.832	15.086	16.750	5
6	0.676	1.237	2.204	5.348	10.645	12.592	14.449	16.812	18.548	6
7	0.989	1.690	2.833	6.346	12.017	14.067	16.013	18.475	20.278	7
8	1.344	2.180	3.490	7.344	13.362	15.507	17.535	20.090	21.955	8
9	1.735	2.700	4.168	8.343	14.684	16.919	19.023	21.666	23.589	9
10	2.156	3.247	4.865	9.342	15.987	18.307	20.483	23.209	25.188	10
11	2.603	3.816	5.578	10.341	17.275	19.675	21.920	24.725	26.757	11
12	3.074	4.404	6.304	11.340	18.549	21.026	23.337	26.217	28.300	12
13	3.565	5.009	7.042	12.340	19.812	22.362	24.736	27.688	29.819	13
14	4.075	5.629	7.790	13.339	21.064	23.685	26.119	29.141	31.319	14
15	4.601	6.262	8.547	14.339	22.307	24.996	27.488	30.578	32.801	15

Correlation Coefficient

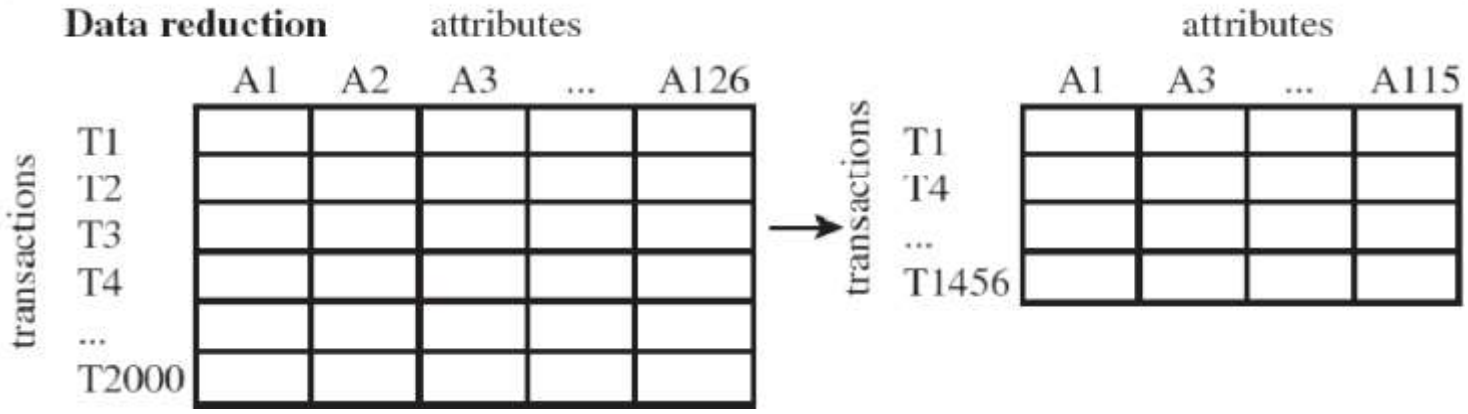
$$r_{A,B} = \frac{\sum_{i=1}^{N}(a_i - \overline{A})(b_i - \overline{B})}{N\sigma_A\sigma_B} = \frac{\sum_{i=1}^{N}(a_i b_i) - N\overline{AB}}{N\sigma_A\sigma_B}$$

$$-1 \leq r_{A,B} \leq +1$$



http://upload.wikimedia.org/wikipedia/commons/0/02/Correlation_examples.png

Data Reduction



Data Reduction

- Data reduction techniques can be applied to obtain a reduced representation of the data set that is much smaller in volume, yet closely maintains the integrity of the original data

  - Attribute (Subset) Selection
  - (Data Cube)Aggregation
  - Dimensionality Reduction
  - Numerosity Reduction
  - Data Discretization
  - Concept Hierarchy Generation

"The Curse of Dimensionality"

- Size
  - The size of a data set yielding the same density of data points in an n-dimensional space increase exponentially with dimensions
- Radius
  - A larger radius is needed to enclose a faction of the data points in a high-dimensional space
- Distance
  - Almost every point is closer to an edge than to another sample point in a high-dimensional space
- Outlier
  - Almost every point is an outlier in a high-dimensional space