

# Data Preprocessing

Instructor: Yiyang (Ian) Wang

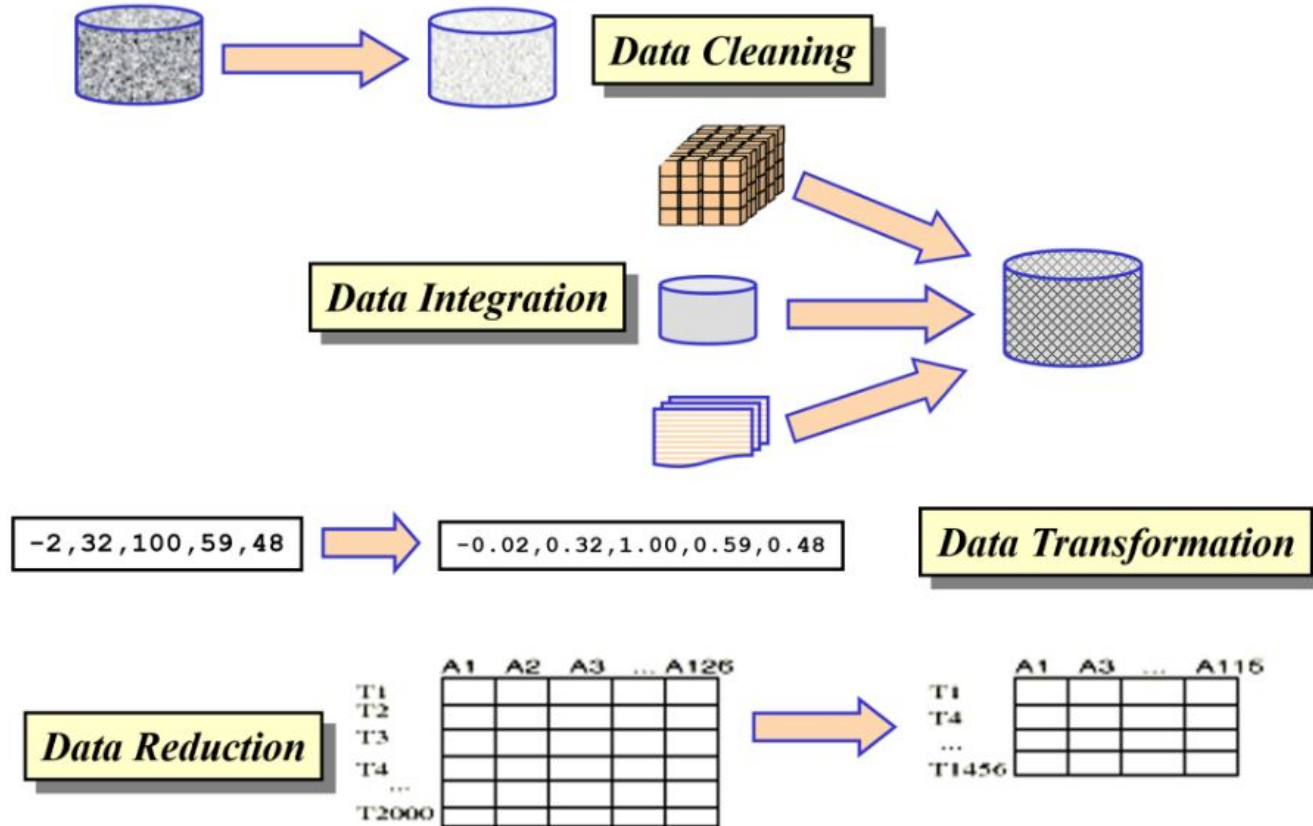
# Why do we need to prepare the data?

In real world applications, data can be inconsistent, incomplete and/or noisy

- Data entry, data transmission, or data collection problems
- Discrepancy in naming conventions
- Duplicate records
- Incomplete or missing data
- Contradictions in data

**“Garbage in, garbage out”**

# Data Preprocessing



# Data Cleaning: Missing Values

*How would you deal with missing data in a CSV file?*

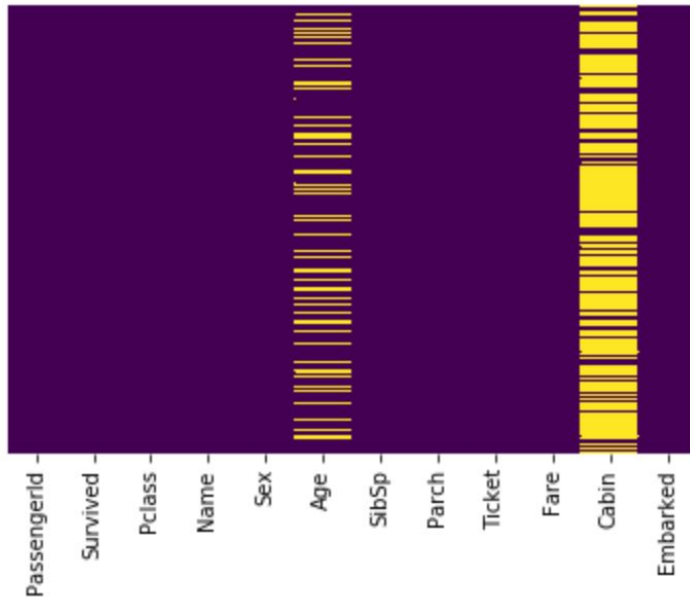
# Options to Solve the Missing Data Problem

- Ignore the record with missing values
- Fill in the missing values manually
- Use a global constant to fill in missing values (NULL, unknown, etc)
- Use the attribute value mean to filling missing values of that attribute
- Use the attribute mean for all samples belonging to the same class to fill in the missing values
- Infer the most probable value to fill in the missing value
  - may need to use methods such as Bayesian classification or decision trees to automatically infer missing attribute values

# Example 1: Titanic Dataset

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

<matplotlib.axes.\_subplots.AxesSubplot at 0xc732278>

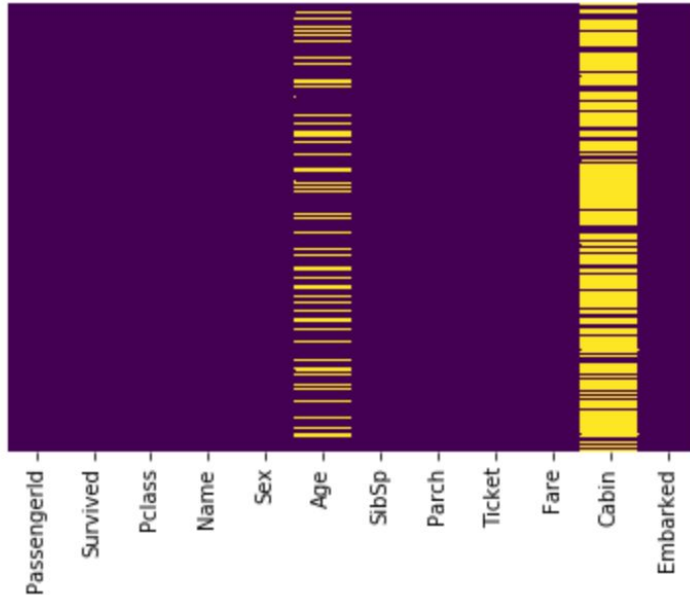


*How would you fill in the missing values of the 'Age' and 'Cabin' columns?*

# Example 1: Titanic Dataset

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

<matplotlib.axes.\_subplots.AxesSubplot at 0xc732278>



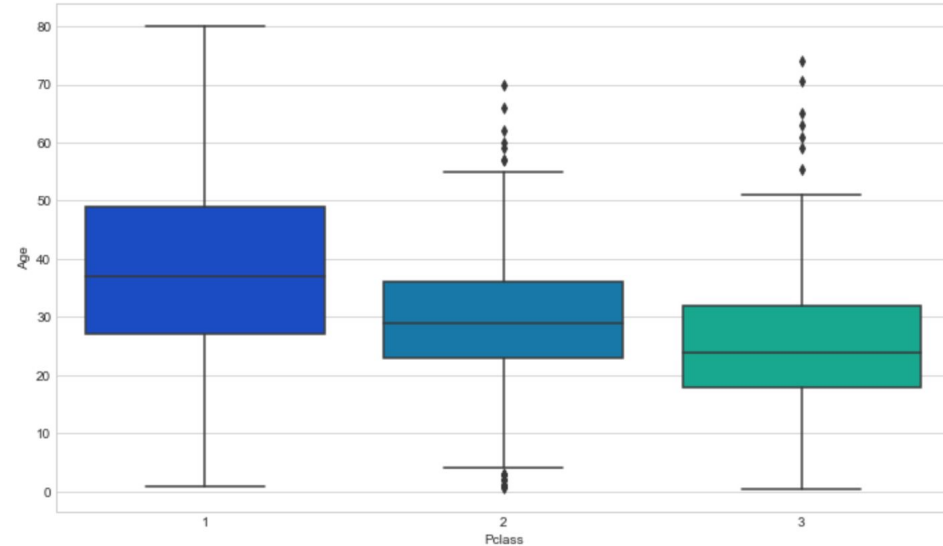
- Roughly 20 percent of the Age data is missing. The proportion of Age missing is likely small enough for reasonable replacement with some form of imputation
- Looking at the Cabin column, it looks like we are just missing too much of that data to do something useful with at a basic level
  - We'll probably drop this later, or change it to another feature like "Cabin Known": 1 or 0



*What is the best way to fill in missing age data?*

One way to do this is by filling in the mean age of all the passengers (**imputation**). However we can be smarter about this and check the average age by passenger class.

# Example 1: Titanic Dataset



```
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24

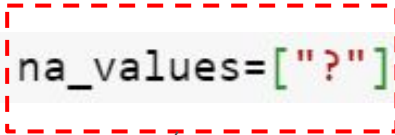
    else:
        return Age
```

Now apply that function!

```
train['Age'] = train[['Age', 'Pclass']].apply(impute_age, axis=1)
```

## Example 2: Video\_Store Data

```
vstable = pd.read_csv("Video_Store_3.csv", index_col=0, na_values=["?"])
```



convert any cells containing the  
"?" value to NaN.

# Example 2: Video\_Store Data

```
1 vstable.head(10)
```

	Gender	Income	Age	Rentals	Avg Per Visit	Genre	Incidentals
Cust ID							
1	M	45000	25.0	32	2.5	Action	Yes
2	F	54000	33.0	12	3.4	Drama	No
3	F	32000	NaN	42	1.6	Comedy	No
4	NaN	59000	70.0	16	4.2	Drama	Yes
5	M	37000	35.0	25	3.2	Action	Yes
6	M	18000	20.0	29	1.7	Action	No
7	F	29000	NaN	19	3.8	Drama	No
8	M	74000	25.0	31	2.4	Action	Yes
9	NaN	38000	21.0	18	2.1	Comedy	No
10	F	65000	40.0	21	3.3	Drama	No

## Example 2: Video\_Store Data

```
1 vstable.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50 entries, 1 to 50
Data columns (total 7 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Gender                45 non-null    object
1   Income                50 non-null    int64
2   Age                  43 non-null    float64
3   Rentals              50 non-null    int64
4   Avg Per Visit        50 non-null    float64
5   Genre                50 non-null    object
6   Incidentals          50 non-null    object
dtypes: float64(2), int64(2), object(3)
memory usage: 3.1+ KB
```

# Example 2: Video\_Store Data

```
1 vstable.describe(include="all")
```

	Gender	Income	Age	Rentals	Avg Per Visit	Genre	Incidentals
count	45	50.000000	43.000000	50.000000	50.000000	50	50
unique	2	NaN	NaN	NaN	NaN	3	2
top	F	NaN	NaN	NaN	NaN	Drama	Yes
freq	23	NaN	NaN	NaN	NaN	20	26
mean	NaN	42300.000000	30.930233	26.320000	2.748000	NaN	NaN
std	NaN	21409.753642	11.650455	10.047723	0.898125	NaN	NaN
min	NaN	1000.000000	16.000000	9.000000	1.100000	NaN	NaN
25%	NaN	26750.000000	22.000000	19.000000	2.125000	NaN	NaN
50%	NaN	41000.000000	29.000000	25.000000	2.750000	NaN	NaN
75%	NaN	56750.000000	35.000000	32.000000	3.375000	NaN	NaN
max	NaN	89000.000000	70.000000	48.000000	4.700000	NaN	NaN

# Example 2: Video\_Store Data

```
1 vstable[vstable.isnull().any(axis=1)]
```

	Gender	Income	Age	Rentals	Avg Per Visit	Genre	Incidentals
Cust ID							
3	F	32000	NaN	42	1.6	Comedy	No
4	NaN	59000	70.0	16	4.2	Drama	Yes
7	F	29000	NaN	19	3.8	Drama	No
9	NaN	38000	21.0	18	2.1	Comedy	No
14	M	45000	NaN	24	2.7	Drama	No
15	NaN	68000	30.0	36	2.7	Comedy	Yes
23	F	2000	NaN	30	2.5	Comedy	No



# Example 2: Video\_Store Data

```
1 vstable[vstable.Age.isnull()]
```

	Gender	Income	Age	Rentals	Avg Per Visit	Genre	Incidentals
Cust ID							
3	F	32000	NaN	42	1.6	Comedy	No
7	F	29000	NaN	19	3.8	Drama	No
14	M	45000	NaN	24	2.7	Drama	No
23	F	2000	NaN	30	2.5	Comedy	No
31	F	49000	NaN	15	3.2	Comedy	No
41	F	50000	NaN	17	1.4	Drama	No
46	F	57000	NaN	9	1.1	Drama	No

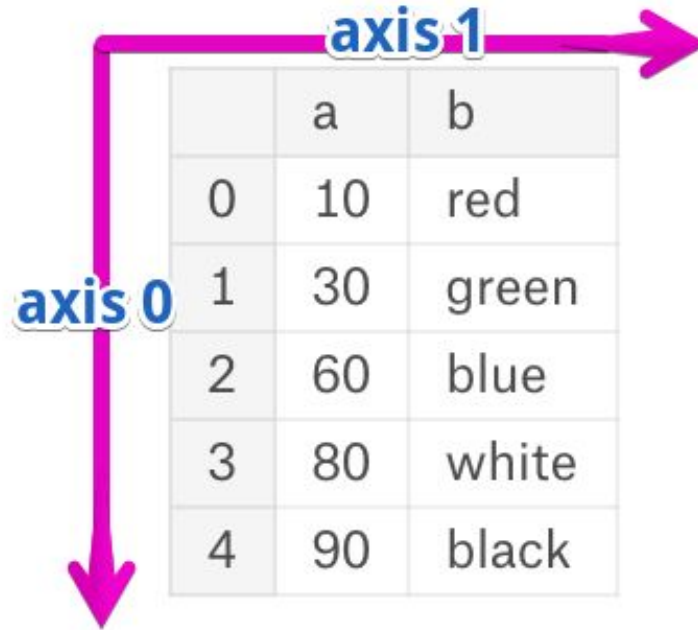
## Example 2: Video\_Store Data

```
1 age_mean = vstable.Age.mean()
2 vstable.Age.fillna(age_mean, axis=0, inplace=True)
```

```
1 vstable.head(5)
```

	Gender	Income	Age	Rentals	Avg Per Visit	Genre	Incidentals
Cust ID							
1	M	45000	25.000000	32	2.5	Action	Yes
2	F	54000	33.000000	12	3.4	Drama	No
3	F	32000	30.930233	42	1.6	Comedy	No
4	NaN	59000	70.000000	16	4.2	Drama	Yes
5	M	37000	35.000000	25	3.2	Action	Yes

Be careful about “axis” in Pandas!



The diagram illustrates the axes of a Pandas DataFrame. A vertical pink arrow labeled 'axis 0' points downwards, indicating the row index. A horizontal pink arrow labeled 'axis 1' points to the right, indicating the column names.

	a	b
0	10	red
1	30	green
2	60	blue
3	80	white
4	90	black

# Example 2: Video\_Store Data

```
1 vstable.drop(vstable[vstable.Gender.isnull()].index, axis=0).head(10)
```

	Gender	Income	Age	Rentals	Avg Per Visit	Genre	Incidentals
Cust ID							
1	M	45000	25.000000	32	2.5	Action	Yes
2	F	54000	33.000000	12	3.4	Drama	No
3	F	32000	30.930233	42	1.6	Comedy	No
5	M	37000	35.000000	25	3.2	Action	Yes
6	M	18000	20.000000	29	1.7	Action	No
7	F	29000	30.930233	19	3.8	Drama	No
8	M	74000	25.000000	31	2.4	Action	Yes
10	F	65000	40.000000	21	3.3	Drama	No
11	F	41000	22.000000	48	2.3	Drama	Yes
12	F	26000	22.000000	32	2.9	Action	Yes

To permanently remove the rows with *NaN* Gender values, use the "*inplace*" parameter in the "drop" function.

```
1 vstable.drop(vstable[vstable.Gender.isnull()].index, axis=0, inplace=True)
2 vstable.head(10)
```

	Gender	Income	Age	Rentals	Avg Per Visit	Genre	Incidentals
Cust ID							
1	M	45000	25.000000	32	2.5	Action	Yes
2	F	54000	33.000000	12	3.4	Drama	No
3	F	32000	30.930233	42	1.6	Comedy	No
5	M	37000	35.000000	25	3.2	Action	Yes
6	M	18000	20.000000	29	1.7	Action	No
7	F	29000	30.930233	19	3.8	Drama	No
8	M	74000	25.000000	31	2.4	Action	Yes
10	F	65000	40.000000	21	3.3	Drama	No
11	F	41000	22.000000	48	2.3	Drama	Yes
12	F	26000	22.000000	32	2.9	Action	Yes

# Easy Approach: Using '*dropna*' Function to Remove Rows with NaN Values

```
1 vstable2.dropna(axis=0, inplace=True)  
2 vstable2.shape
```

```
(38, 7)
```

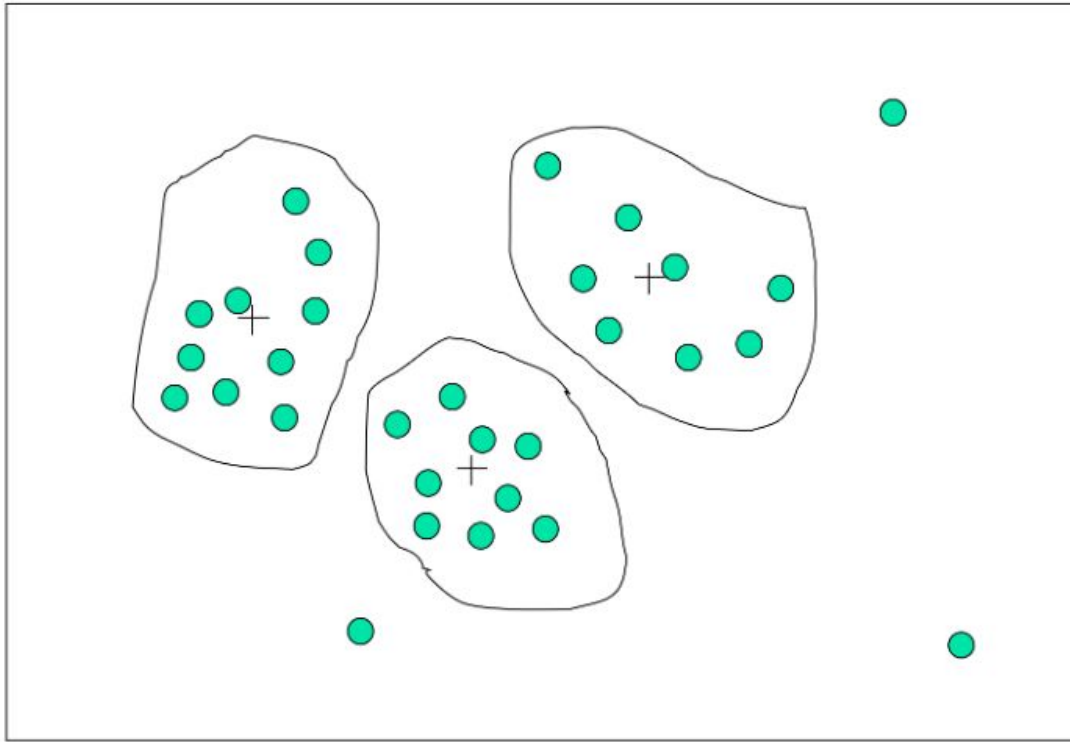
# Data Cleaning: Smoothing

# Noisy Data

- Noise: Random Error or variance in a measured variable
- Incorrect **attribute** values may be due to:
  - Faulty data collection instruments
  - Data Entry Problems
  - Data Transmission Problems
  - Technology Limitation
  - Inconsistency in naming convention
- **Class label** noise is hard to deal with
  - Sometimes we don't know whether the class label is correct, or it is simply unexpected



# Cluster Analysis for Data Smoothing



**Similar values are organized into groups (clusters). Values falling outside of clusters may be considered “outliers” and may be candidates for elimination.**

# Smoothing Noisy Data – Example

Want to smooth "Temperature" by bin means with bins of size 3:

1. First sort the values of the attribute (keep track of the ID or key so that the transformed values can be replaced in the original table).
2. Divide the data into bins of size 3 (or less in case of last bin).
3. Convert the values in each bin to the mean value for that bin
4. Put the resulting values into the original table


ID	Outlook	Temperature	Humidity	Windy
1	sunny	85	85	FALSE
2	sunny	80	90	TRUE
3	overcast	83	78	FALSE
4	rain	70	96	FALSE
5	rain	68	80	FALSE
6	rain	65	70	TRUE
7	overcast	58	65	TRUE
8	sunny	72	95	FALSE
9	sunny	69	70	FALSE
10	rain	71	80	FALSE
11	sunny	75	70	TRUE
12	overcast	73	90	TRUE
13	overcast	81	75	FALSE
14	rain	75	80	TRUE



ID	Temperature	
7	58	Bin1
6	65	
5	68	
9	69	Bin2
4	70	
10	71	
8	72	Bin3
12	73	
11	75	
14	75	Bin4
2	80	
13	81	
3	83	Bin5
1	85	

# Smoothing Noisy Data – Example

ID	Temperature	
7	58	Bin1
6	65	
5	68	
9	69	Bin2
4	70	
10	71	
8	72	Bin3
12	73	
11	75	
14	75	Bin4
2	80	
13	81	
3	83	Bin5
1	85	



ID	Temperature	
7	64	Bin1
6	64	
5	64	
9	70	Bin2
4	70	
10	70	
8	73	Bin3
12	73	
11	73	
14	79	Bin4
2	79	
13	79	
3	84	Bin5
1	84	

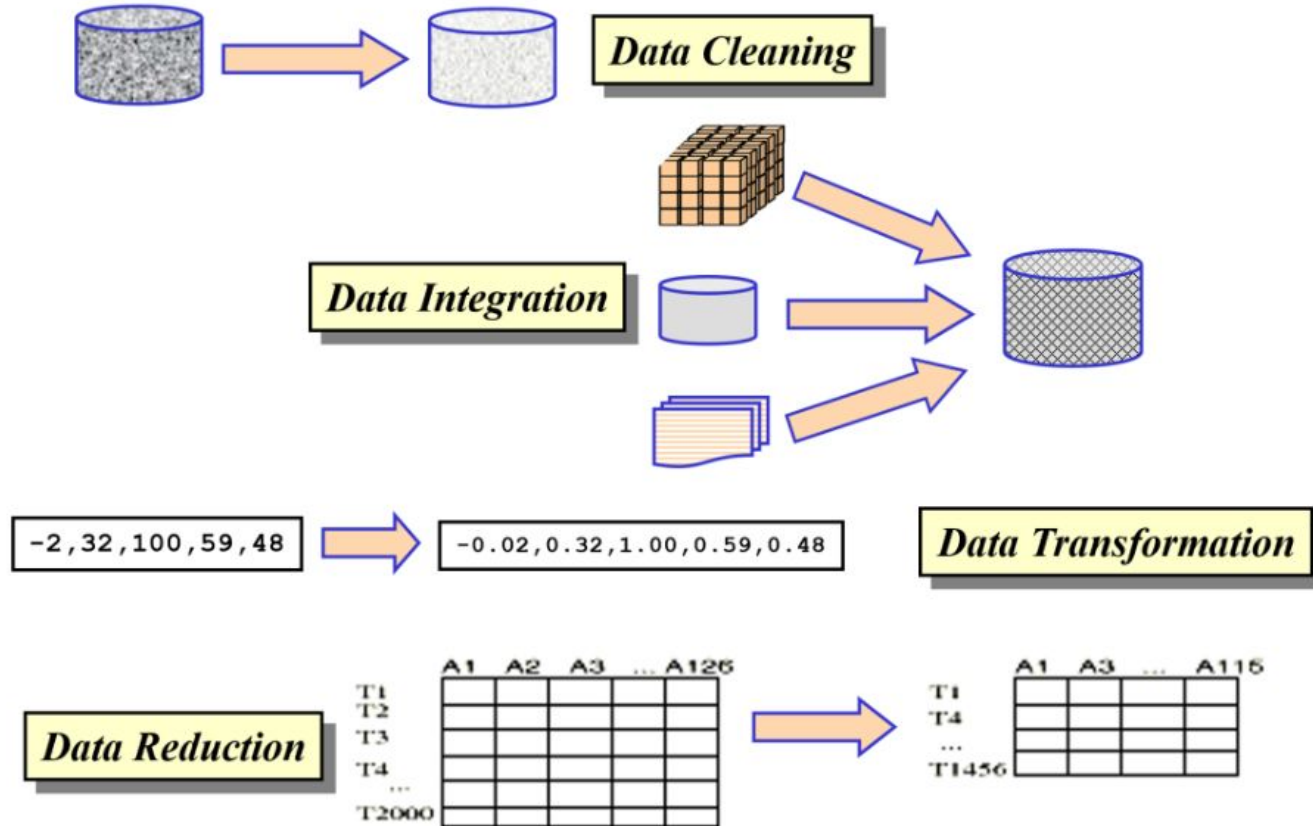
Value of every record in each bin is changed to the mean value for that bin. If it is necessary to keep the value as an integer, then the mean values are rounded to the nearest integer.

# Smoothing Noisy Data – Example

The final table with the new values for the Temperature attribute.

ID	Outlook	Temperature	Humidity	Windy
1	sunny	84	85	FALSE
2	sunny	79	90	TRUE
3	overcast	84	78	FALSE
4	rain	70	96	FALSE
5	rain	64	80	FALSE
6	rain	64	70	TRUE
7	overcast	64	65	TRUE
8	sunny	73	95	FALSE
9	sunny	70	70	FALSE
10	rain	70	80	FALSE
11	sunny	73	70	TRUE
12	overcast	73	90	TRUE
13	overcast	79	75	FALSE
14	rain	79	80	TRUE

# Data Preprocessing



# Data Transformation: Normalization

# Why we need to normalize features / attributes?

- Some machine learning algorithms are **sensitive to the scale of the input features**, and may not perform well if the features are not normalized.
  - *Can you think of any machine learning model algorithms that are sensitive to feature scales?*

# Two Popular Normalization Techniques

- **Min-Max normalization:** This technique scales the values of a feature to a *range between 0 and 1* (most of the time).
  - This is done by subtracting the minimum value of the feature from each value, and then dividing by the range of the feature
- **Z-score normalization:** This technique scales the values of a feature to have a mean of 0 and a standard deviation of 1.
  - This is done by subtracting the mean of the feature from each value, and then dividing by the standard deviation



# Min-max Normalization (Rescaling)

## General Formula

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

To rescale a range between an arbitrary set of values [a, b]

$$x' = a + \frac{(x - \min(x))(b - a)}{\max(x) - \min(x)}$$

## Z-score normalization (Standardization)

$$x' = \frac{x - \bar{x}}{\sigma}$$

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}, \text{ where } \mu = \frac{1}{N} \sum_{i=1}^N x_i.$$

Question: When do you prefer Min-Max normalization over Z-score normalization, or vice versa?

# Min-Max Normalization v.s. Z-score Normalization

Min-Max Normalization	Z-score Normalization
Scales data to a specific range	Scales data to have a mean (average) of 0 and a standard deviation of 1
<b>Sensitive to outliers</b>	<b>Relatively robust to outliers</b>
Easily interpretable because it directly represents the proportion of the original data's range.	Don't have a direct, intuitive interpretation like Min-Max normalization.
No assumption of a normal distribution	<b>Assumes normal distribution</b>

# Normalization Example

- **Min-Max normalization on an employee database**

- ▶ max distance for salary:  $100000 - 19000 = 81000$
- ▶ max distance for age:  $52 - 27 = 25$
- ▶ New min for age and salary = 0; new max for age and salary = 1

$$x'_i = \frac{x_i - \min x_i}{\max x_i - \min x_i} (\text{new max} - \text{new min}) + \text{new min}$$

ID	Gender	Age	Salary
1	F	27	19,000
2	M	51	64,000
3	M	52	100,000
4	F	33	55,000
5	M	45	45,000

ID	Gender	Age	Salary
1	1	0.00	0.00
2	0	0.96	0.56
3	0	1.00	1.00
4	1	0.24	0.44
5	0	0.72	0.32

# Data Transformation: Discretization

# Discretization Example

**Example: discretizing the “Humidity” attribute using 3 bins.**

Humidity
85
90
78
96
80
70
65
95
70
80
70
90
75
80



Low = 60-69  
Normal = 70-79  
High = 80+



Humidity
High
High
Normal
High
High
Normal
Low
High
Normal
High
Normal
High
Normal
High

# Data Transformation: Converting Categorical Attributes to Numerical Attributes (One-Hot Encoding)



# One-Hot Encoding Motivation

Some machine learning algorithms can work directly with categorical data depending on implementation, such as a decision tree, but most require any inputs or outputs variables to be a number, or numeric in value. This means that any categorical data must be mapped to integers.

Type	Onehot encoding →	Type	AA_Onehot	AB_Onehot	CD_Onehot
AA		AA	1	0	0
AB		AB	0	1	0
CD		CD	0	0	1
AA		AA	0	0	0

# One-Hot Encoding Example

ID	Outlook	Temperature	Humidity	Windy
1	sunny	85	85	FALSE
2	sunny	80	90	TRUE
3	overcast	83	78	FALSE
4	rain	70	96	FALSE
5	rain	68	80	FALSE
6	rain	65	70	TRUE
7	overcast	58	65	TRUE
8	sunny	72	95	FALSE
9	sunny	69	70	FALSE
10	rain	71	80	FALSE
11	sunny	75	70	TRUE
12	overcast	73	90	TRUE
13	overcast	81	75	FALSE
14	rain	75	80	TRUE

## Attributes:

Outlook (overcast, rain, sunny)

Temperature real

Humidity real

Windy (true, false)

## Standard Spreadsheet Format

Create separate columns for each value of a categorical attribute (e.g., 3 values for the Outlook attribute and two values of the Windy attribute). There is no change to the numerical attributes.

Outlook	Outlook	Outlook	Temp	Humidity	Windy	Windy
overcast	rain	sunny			TRUE	FALSE
0	0	1	85	85	0	1
0	0	1	80	90	1	0
1	0	0	83	78	0	1
0	1	0	70	96	0	1
0	1	0	68	80	0	1
0	1	0	65	70	1	0
1	0	0	64	65	1	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.

For the previous example, why not convert the  
    'Outlook' column into numerical values:  
0 for 'Overcast,' 1 for 'Rain,' and 2 for 'Sunny'?

Discussion: What are the potential disadvantages of using one-hot encoding?