

# Chapter 6

## **DATA MANIPULATION**

---

DATA ANALYSIS IN R

# Outline

- data Already in R
- Examples of Reading and Formatting Datasets
- Major tasks in Data Preprocessing
- Data transformation and Data Discretization
- Manipulating Data with dplyr
- Tidying Data with tidyr

# Data Already in R

- There are some datasets already built into R or available in R packages: package dataset.

```
> library(datasets)
> library(help="datasets")
```

- To load an actual dataset into R's memory, use the `data()` function

Datasets are all relatively small

They are ideal for quickly testing the code you are working with.

```
> data(cars)
> head(cars)
  speed  dist
1     4     2
2     4    10
3     7     4
4     7    22
5     8    16
6     9    10
```

# Datasets:Example

- **Another package with several useful datasets is mlbench.**

It contains datasets for machine learning benchmarks so these datasets are aimed at testing how new methods perform on known datasets.

This package is not distributed together with R, but you can install it, load it, and get a list of the datasets in it like this:

```
install.packages("mlbench")  
library(mlbench)  
library(help="mlbench")
```

# Quickly Reviewing Data

- **head()**, which shows the first  $n$  lines of a data frame where  $n$  is an option with default 6.
- **tail()** gives you the last  $n$  lines
- To get summary statistics for all the columns in a data frame, you can use the **summary()** function
- **The summary you get depends on the types the columns have.**
  - Numerical data is summarized by their quartiles
  - Categorical and boolean data is summarized by counts of each category or TRUE/FALSE values.
- **To see the type of each column, you can use the str() function.**
  - This gives you the structure of a data type

# Example

In the iris dataset there is one column, Species, that is categorical, and its summary is the count of each level.

```
> data(iris)
> iris %>% summary
```

| Sepal.Length  | Sepal.Width   | Petal.Length  | Petal.Width   | Species       |
|---------------|---------------|---------------|---------------|---------------|
| Min. :4.300   | Min. :2.000   | Min. :1.000   | Min. :0.100   | setosa :50    |
| 1st Qu.:5.100 | 1st Qu.:2.800 | 1st Qu.:1.600 | 1st Qu.:0.300 | versicolor:50 |
| Median :5.800 | Median :3.000 | Median :4.350 | Median :1.300 | virginica :50 |
| Mean :5.843   | Mean :3.057   | Mean :3.758   | Mean :1.199   |               |
| 3rd Qu.:6.400 | 3rd Qu.:3.300 | 3rd Qu.:5.100 | 3rd Qu.:1.800 |               |
| Max. :7.900   | Max. :4.400   | Max. :6.900   | Max. :2.500   |               |

# Examples of Reading and Formatting Datasets:

## Boston Housing Dataset

BostonHousing data contains information about crime rates and some explanatory variables we can use to predict crime rates.

```
> data(BostonHousing)
> str(BostonHousing)
'data.frame':  506 obs. of  14 variables:
 $ crim    : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
 $ zn      : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
 $ indus   : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
 $ chas    : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ nox     : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
 $ rm      : num  6.58 6.42 7.18 7 7.15 ...
 $ age     : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
 $ dis     : num  4.09 4.97 4.97 6.06 6.06 ...
 $ rad     : num  1 2 2 3 3 3 5 5 5 5 ...
 $ tax     : num  296 242 242 222 222 222 311 311 311 311 ...
 $ ptratio : num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
 $ b       : num  397 397 393 395 397 ...
```

# Data Preprocessing

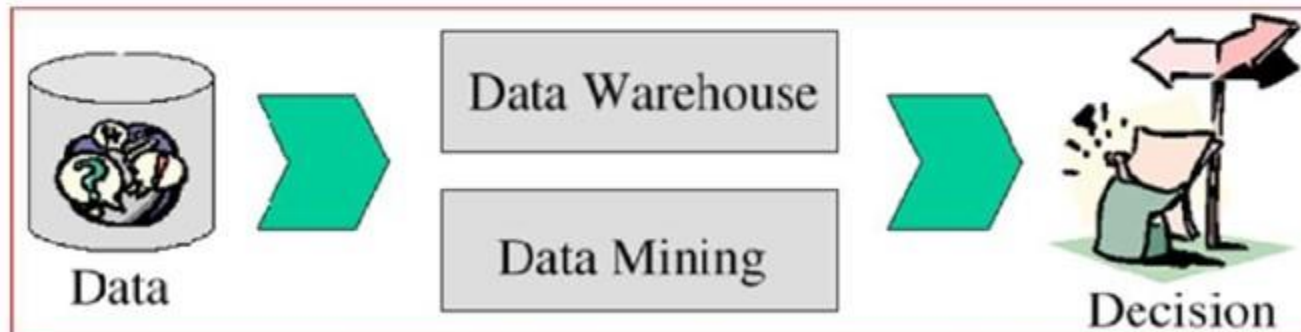
- **Why do we need to prepare the data?**

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

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

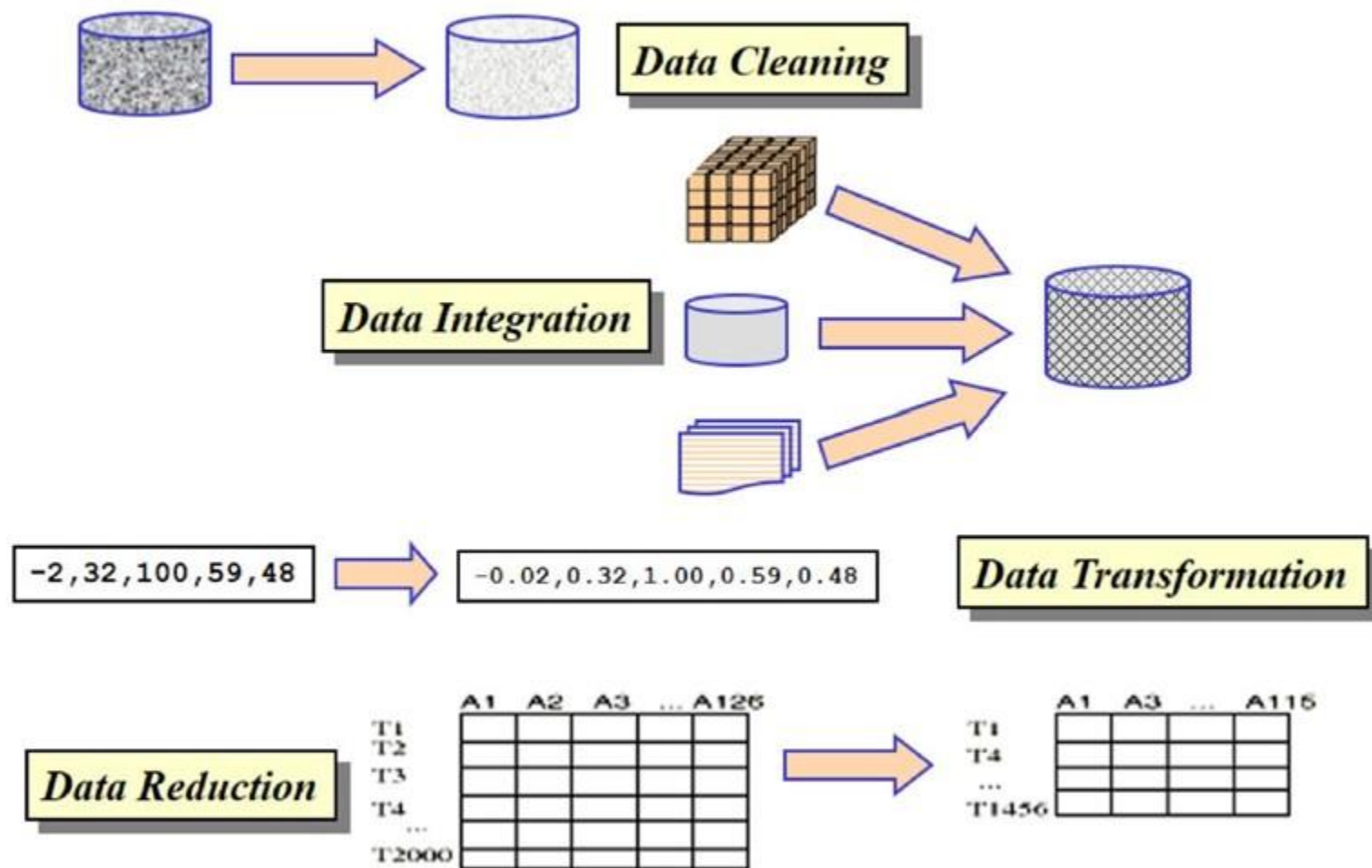
- **What happens when the data can not be trusted?**

- ▶ Can the decision be trusted? Decision making is jeopardized



- ▶ Better chance to discover useful knowledge when data is clean

# Major Tasks in Data Preprocessing



# Data Cleaning

- **Data in the Real World Is Dirty: Lots of potentially incorrect data, e.g., instrument faulty, human or computer error, transmission error**

- ❖ incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data

- e.g., *Occupation*=" " (missing data)

- ❖ noisy: containing noise, errors, or outliers

- e.g., *Salary*=="-10" (an error)

- ❖ inconsistent: containing discrepancies in codes or names, e.g.,

- Age="42", Birthday= 31/5/2015

- Was rating "1 2 3" now rating "A B C"

- discrepancy between duplicate records

- ❖ Intentional (e.g., *disguised missing data*)

- Jan. 1 as everyone's birthday?

# Data Cleaning

- **Real-world application data can be incomplete, noisy, and inconsistent**
  - ❖ No recorded values for some attributes
  - ❖ Not considered at time of entry
  - ❖ Random errors
  - ❖ Irrelevant records or fields
- **Data cleaning attempts to:**
  - ❖ Fill in missing values
  - ❖ Smooth out noisy data
  - ❖ Correct inconsistencies
  - ❖ Remove irrelevant data



# Incomplete (Missing) Data

- **Data is not always available**

E.g., many tuples have no recorded value for several attributes such as customer income in sales data

- **Missing data may be due to**

- ❖ Equipment malfunction
- ❖ inconsistent with other recorded data and thus deleted
- ❖ data not entered due to misunderstanding
- ❖ certain data may not be considered important at the time of entry
- ❖ not register history or changes of the data

- **Missing data may need to be inferred**

# Dealing with Missing Values

- **Solving 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;
- ❖ 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

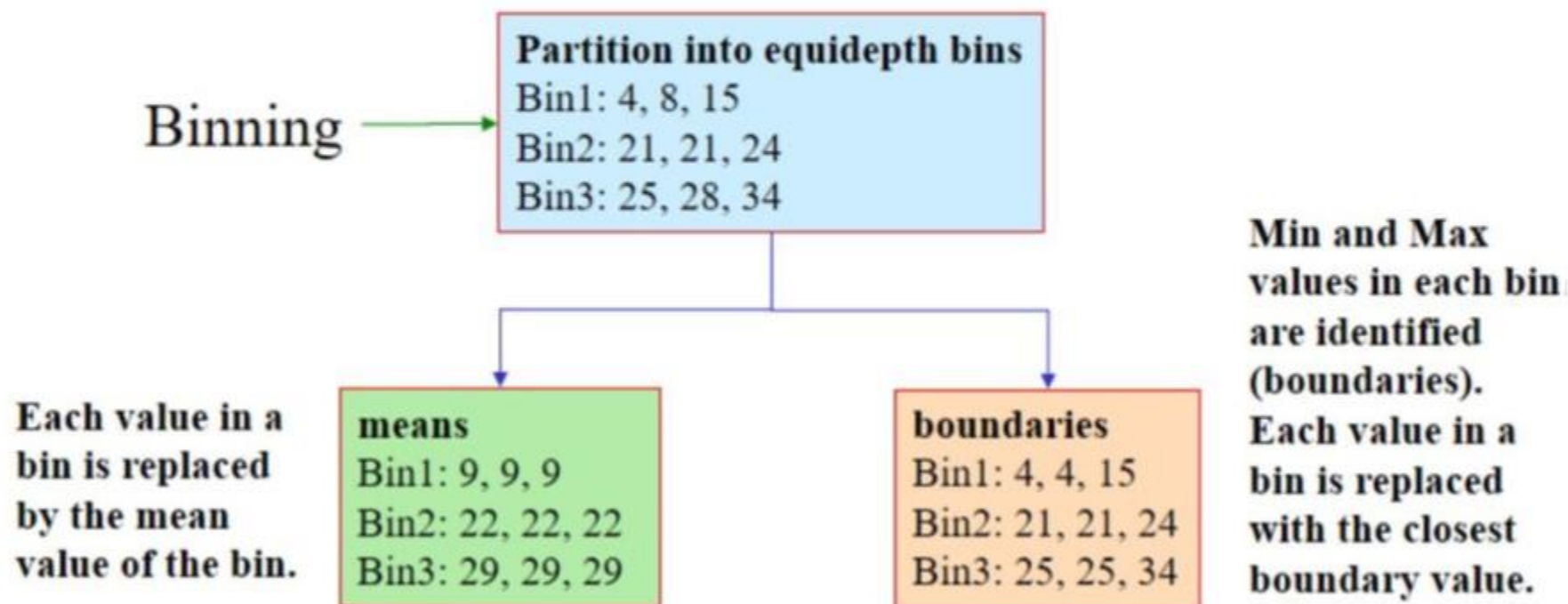
# Noisy Data

- **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
- **Other data problems which require data cleaning**
  - ▶ duplicate records
  - ▶ incomplete data
  - ▶ inconsistent data

# Smoothing Noisy Data

- The purpose of data smoothing is to eliminate noise and "smooth out" the data fluctuations

Ex: Original Data for "price" (after sorting): 4, 8, 15, 21, 21, 24, 25, 28, 34



- You can smooth noisy data using data visualization techniques.

# 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

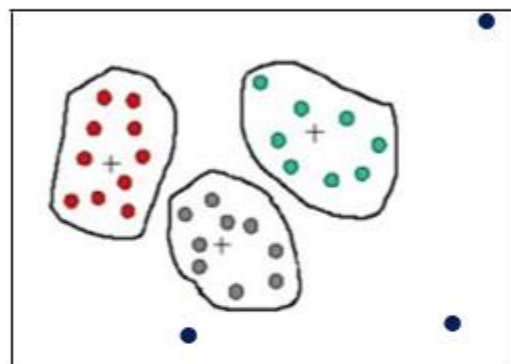
The final table with the new values for the Temperature attribute.

| 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  |

# Smoothing Noisy Data

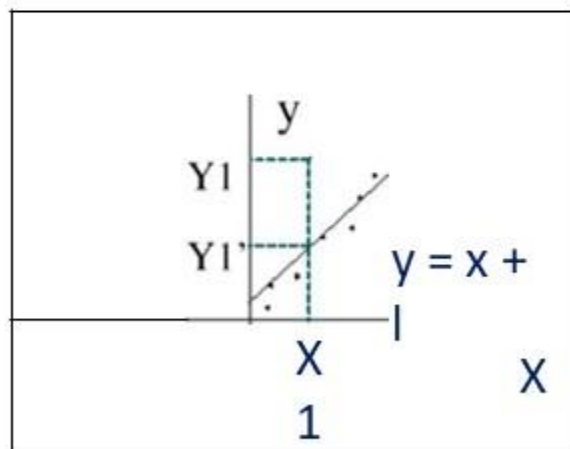
- Other Methods

Clustering



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

Regression



Fit data to a function. Linear regression finds the best line to fit two variables. Multiple regression can handle multiple variables. The values given by the function are used instead of the original values.

# Data Integration

- Data analysis may require a combination of data from multiple sources into a coherent data store
- Challenges in Data Integration:
  - ▶ Schema integration: Integrate metadata from different sources
    - CID = C\_number = Cust-id = cust#
  - ▶ Semantic heterogeneity: Identify real world entities from multiple data sources.
    - e.g., Bill Clinton = William Clinton
  - ▶ Data value conflicts (different representations or scales, etc.)
    - e.g., metric vs. British units
  - ▶ Redundant attributes (redundant if it can be derived from other attributes) -- may be able to identify redundancies via **correlation analysis** and **covariance analysis**.

# Handling Redundancy in Data Integration

- Redundant attributes may be able to be detected by *correlation analysis* and *covariance analysis*
- Careful integration of the data from multiple sources may help reduce/avoid redundancies and inconsistencies and improve mining speed and quality

# Data Transformation

- Normalization
- Discretization

# What is normalization

- **Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values.**

For machine learning, every dataset doesn't require normalization. It is required only when features have different ranges. In simple words when multiple attributes are there, but attributes have values on different scale this may lead to poor data models while performing data mining operation. So they are normalized to bring all the attributes on the same scale, usually something between  $[0,1]$ .

- **It is not always a good idea to normalize the data since we might lose information about maximum and minimum values. Sometimes, it is a good idea to do so.**

For example , ML algorithms such as linear regression or SVM typically converge faster on normalized data. But on algorithm like k-means or K-NN, normalization could be a good or bad choice depending on the use case since the distance between the points plays a key role here

# Normalizing/Standardizing Numeric Data

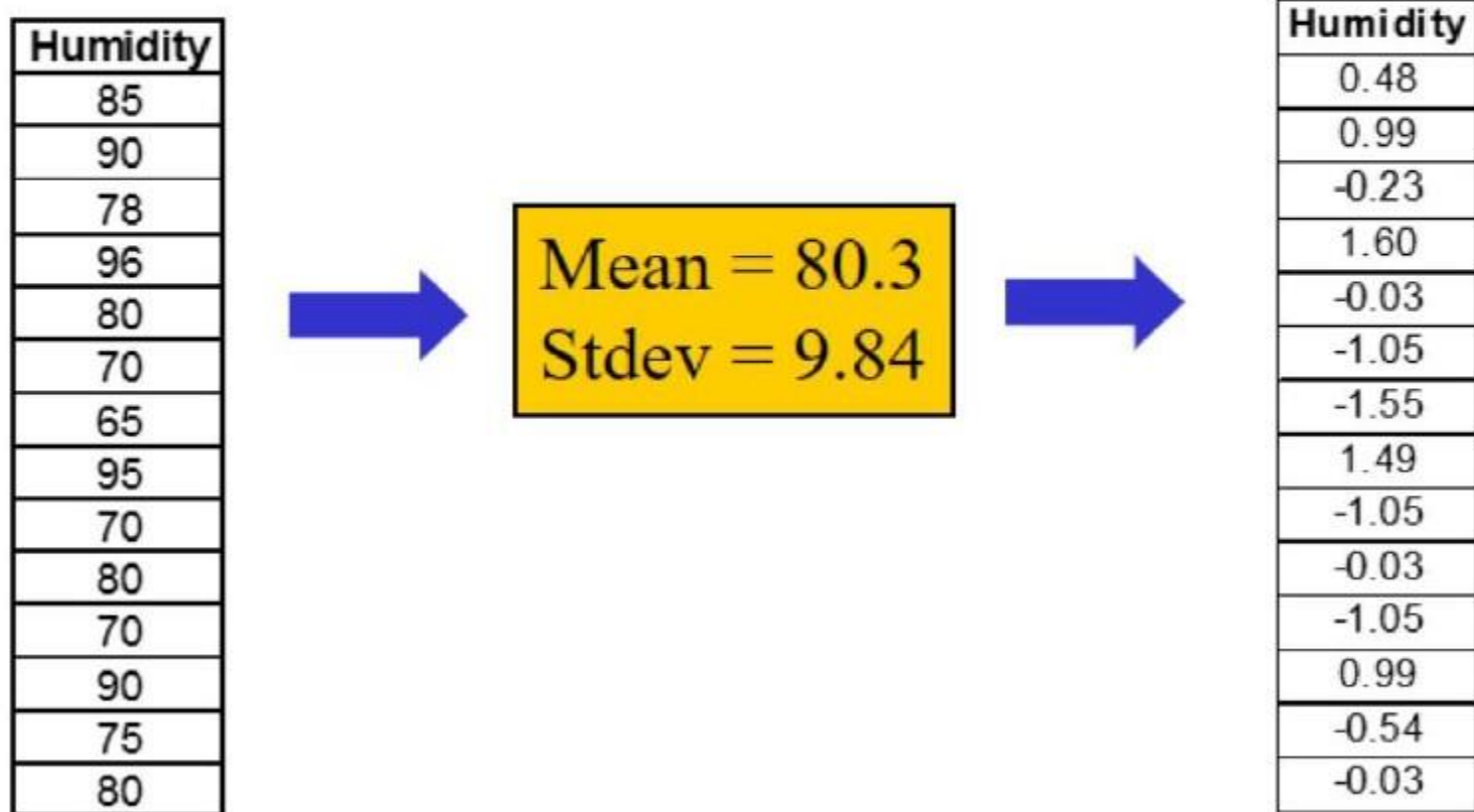
- **Z-score normalization:** Values are normalized based on a mean and standard deviation of the data

$x$ : raw value to be standardized,  $\mu$ : mean of the population,  $\sigma$ : standard deviation  
negative when the value is below the mean, "+" when above

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

# Normalization: Example

- z-score normalization:  $v' = (v - \text{Mean}) / \text{Stdev}$
- Example: normalizing the "Humidity" attribute:



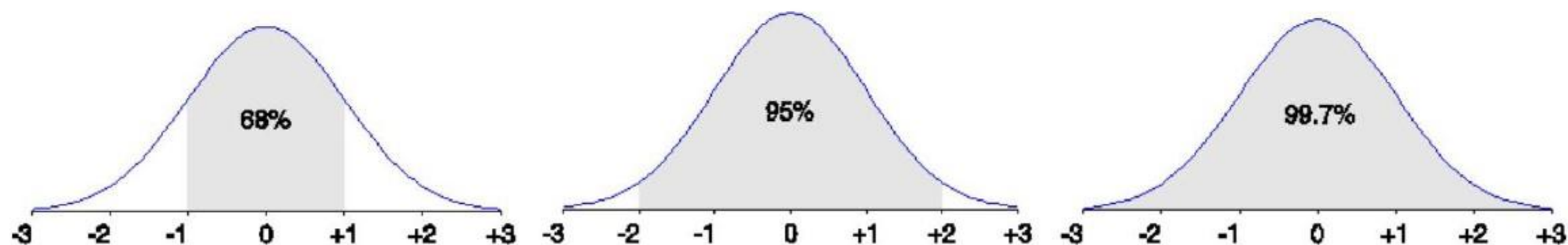
# Properties of Normal Distribution Curve Empirical Rule(Review)

- **The normal (distribution) curve**

From  $\mu - \sigma$  to  $\mu + \sigma$ : contains about 68% of the measurements  
( $\mu$ : mean,  $\sigma$ : standard deviation)

From  $\mu - 2\sigma$  to  $\mu + 2\sigma$ : contains about 95% of it

, From  $\mu - 3\sigma$  to  $\mu + 3\sigma$ : contains about 99.7% of it



# Normalizing or Standardizing Numeric Data: Z-Score

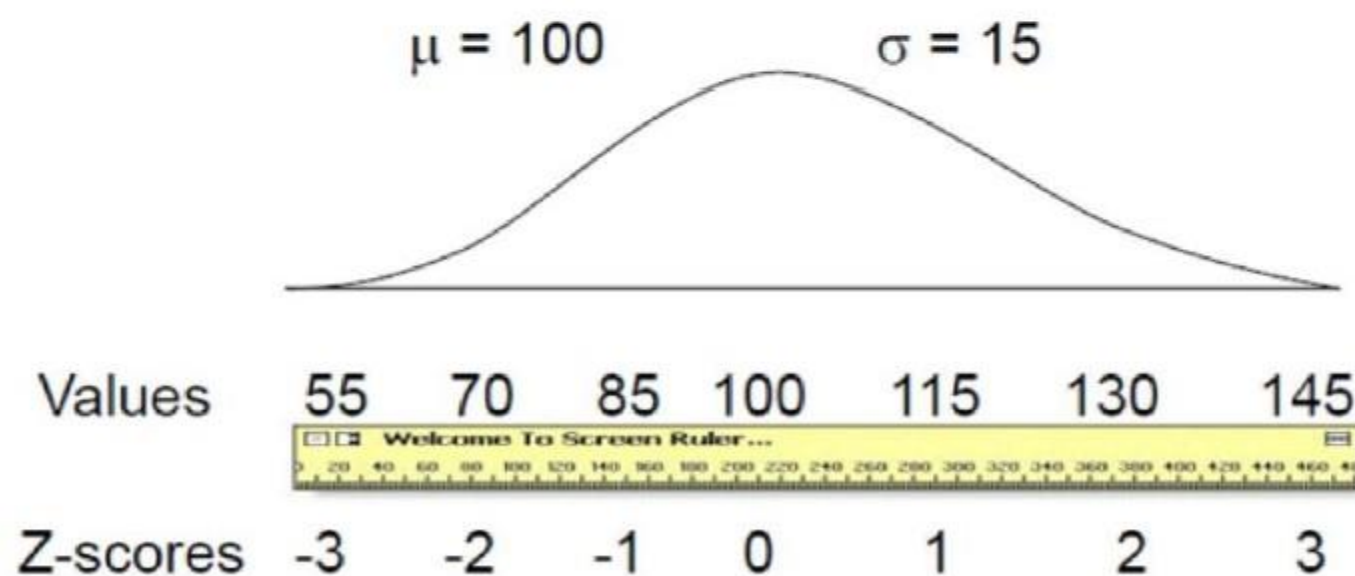
$$Z = \frac{x - \mu}{\sigma}$$

$$Z = 100 - 100 / 15 = 0$$

$$Z = 145 - 100 / 15 = 45/15 = 3$$

$$Z = 70 - 100 / 15 = -30/15 = -2$$

$$Z = 105 - 100 / 15 = 5/15 = .33$$



# Normalizing Numeric Data

- **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   |

# R scale Function

- **Scale()** function centers and/or scales the columns of a numeric matrix.
- **scale**, with default settings will calculate the mean and standard deviation of the entire vector, then "scale" each element by those values by subtracting the mean and dividing by the sd. (If you use **scale(x, scale=FALSE)**, it will only subtract the mean but not divide by the std deviation.)
- **scale(x, center = TRUE, scale = TRUE)**
- **x: numeric matrix**

```
> x <- matrix(1:9,3,3)
> x
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

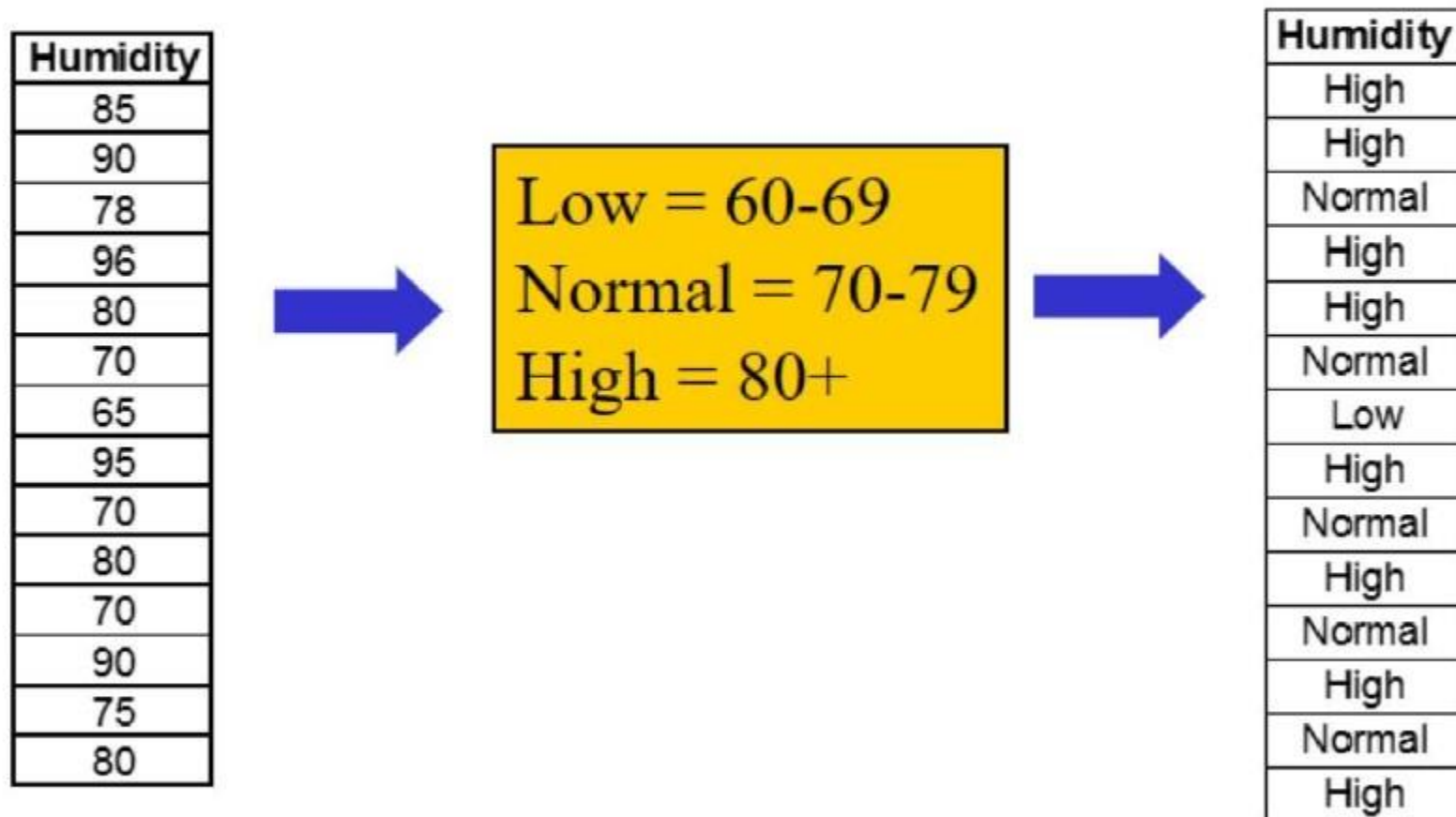
```
> scale(x)
      [,1] [,2] [,3]
[1,]   -1   -1   -1
[2,]    0    0    0
[3,]    1    1    1
attr(,"scaled:center")
[1] 2 5 8
attr(,"scaled:scale")
[1] 1 1 1
```

# Data Transformation: Discretization

- **Discretization is the process of putting values into buckets so that there are a limited number of possible states.**  
numerical data is transformed into categorical data
- **3 Types of attributes**  
nominal - values from an unordered set (also "categorical" attributes)  
ordinal - values from an ordered set  
numeric/continuous - real numbers
- **Discretization is used to reduce the number of values for a given continuous attribute**  
usually done by dividing the range of the attribute into intervals  
interval labels are then used to replace actual data values
- **Some data mining algorithms only accept categorical attributes and cannot handle a range of continuous attribute value**
- **Discretization can also be used to generate concept hierarchies**  
reduce the data by collecting and replacing low level concepts (e.g., numeric values for "age") by higher level concepts (e.g., "young", "middle aged", "old")

# Discretization - Example

- Example: discretizing the "Humidity" attribute using 3 bins.



# Simple Discretization: Binning

- **Binning** is the process of transforming numerical or continuous data into categorical data. It is a common data pre-processing step of the model building process.
- **Equal-width (distance) partitioning**
  - Divides the range into N intervals of equal size: uniform grid
  - If A and B are the lowest and highest values of the attribute, the width of intervals will be:  $W=(B-A)/N$
  - The most straightforward, but outliers may dominate presentation
  - Skewed data is not handled well
- **Equal-depth (frequency) partitioning**
  - Divides the range into N intervals, each containing approximately same number of samples
  - Good data scaling
  - Managing categorical attributes can be tricky

# Binning Methods for Data Smoothing

- **Sorted data for price (in dollars):**  
4, 8, 9, 15, 21, 21, 24, 25, 26, 28, 29, 34
- **Partition into equal-frequency (equi-depth) bins:**  
Bin 1: 4, 8, 9, 15  
Bin 2: 21, 21, 24, 25  
Bin 3: 26, 28, 29, 34
- **Smoothing by bin means**  
Bin 1: 9, 9, 9, 9  
Bin 2: 23, 23, 23, 23  
Bin 3: 29, 29, 29, 29

# Binning in R : bin()

- **Bin()**

`bin(data, nbins = 5, labels = NULL, method= c("length", "content", "clusters"), na.omit = TRUE)`

- **Data** :data frame or vector which contains the data.
- **Nbins** : number of bins (= levels).
- **Labels**: character vector of labels for the resulting category.
- **Method** :character string specifying the binning method
  - Method "length" gives intervals of equal length,
  - method "content" gives intervals of equal content (via quantiles).
  - Method "clusters" determines "nbins" clusters via 1D kmeans with deterministic seeding of the initial clustercentres
- **na.omit**: logical value whether instances with missing values should be removed.
  - When "na.omit= FALSE" an additional level "NA" is added to each factor with missing values.

# Example

```
install.packages("OneR")
```

```
library(OneR)
```

```
> data <- iris
```

```
> str(iris)
```

```
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

```
> str(bin(data, nbins = 3))
```

```
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: Factor w/ 3 levels "(4.3,5.5]","(5.5,6.7]",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ Sepal.Width : Factor w/ 3 levels "(2,2.8]","(2.8,3.6]",...: 2 2 2 2 2 3 2 2 2 2 ...
 $ Petal.Length: Factor w/ 3 levels "(0.994,2.97]",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ Petal.Width : Factor w/ 3 levels "(0.0976,0.9]",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

```
> str(bin(data, nbins = 3, labels = c("small", "medium", "large")))
```

```
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: Factor w/ 3 levels "small","medium",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ Sepal.Width : Factor w/ 3 levels "small","medium",...: 2 2 2 2 2 3 2 2 2 2 ...
 $ Petal.Length: Factor w/ 3 levels "small","medium",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ Petal.Width : Factor w/ 3 levels "small","medium",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

# Discretization in R: cut()

- Cut function is useful for turning continuous variables into factors.

Specify the number of cutpoints

Specify the exact cutpoints,

```
> clinical.trial <- data.frame(patient = 1:100,  
+                             age = rnorm(100, mean = 60, sd = 8),  
+                             year.enroll = sample(paste("19", 85:99, sep = ""),  
+                                                  100, replace = TRUE))  
> summary(clinical.trial)
```

| patient  |         | age           | year.enroll      |
|----------|---------|---------------|------------------|
| Min.     | : 1.00  | Min. :36.02   | Length:100       |
| 1st Qu.: | 25.75   | 1st Qu.:54.40 | Class :character |
| Median : | 50.50   | Median :60.87 | Mode :character  |
| Mean :   | 50.50   | Mean :60.03   |                  |
| 3rd Qu.: | 75.25   | 3rd Qu.:65.44 |                  |
| Max.     | :100.00 | Max. :78.57   |                  |

# Example

use the basic cut function with a numeric variable to make age a factor(categorical variable).  
Use cut with the breaks argument to break up age into 4 intervals.

```
> c1 <- cut(clinical.trial$age, breaks=4)
> table(c1)
c1
(36,46.7] (46.7,57.3] (57.3,67.9] (67.9,78.6]
      5         31         53         11
```

```
> c2 <- cut(as.numeric(clinical.trial$year.enroll), breaks = 3)
> table(c2)
c2
(1985,1990] (1990,1994] (1994,1999]
      34         29         37
```

# Example

- Specify the break points explicitly instead of keeping the default

```
> seq(30,80, by = 10)
[1] 30 40 50 60 70 80
> c1 <- cut(clinical.trial$age, breaks = seq(30, 80, by = 10))
> table(c1)
c1
(30,40] (40,50] (50,60] (60,70] (70,80]
      0      14      43      34      9
```

# Converting Categorical Attributes to Numerical Attributes

| 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     |
| .        | .       | .       | .    | .        | .     | .     |
| .        | .       | .       | .    | .        | .     | .     |

# Concept Hierarchy Generation

- Concept hierarchy organizes concepts (i.e., attribute values) hierarchically
- Concept hierarchy formation: Recursively reduce the data by collecting and replacing low level concepts (such as numeric values for *age*) by higher level concepts (such as *youth*, *adult*, or *senior*)
- Concept hierarchies can be explicitly specified by domain experts and/or data warehouse designers
- Concept hierarchy can be automatically formed for both numeric and nominal data.  
For numeric data, use discretization methods shown.

# Concept Hierarchy Generation for Nominal Data

- **Specification of a partial/total ordering of attributes explicitly at the schema level by users or experts**

*street < city < state < country*

- **Specification of a hierarchy for a set of values by explicit data grouping**

*{Urbana, Champaign, Chicago} < Illinois*

- **Specification of only a partial set of attributes**

*only street < city, not others*

- **Automatic generation of hierarchies (or attribute levels) by the analysis of the number of distinct values**

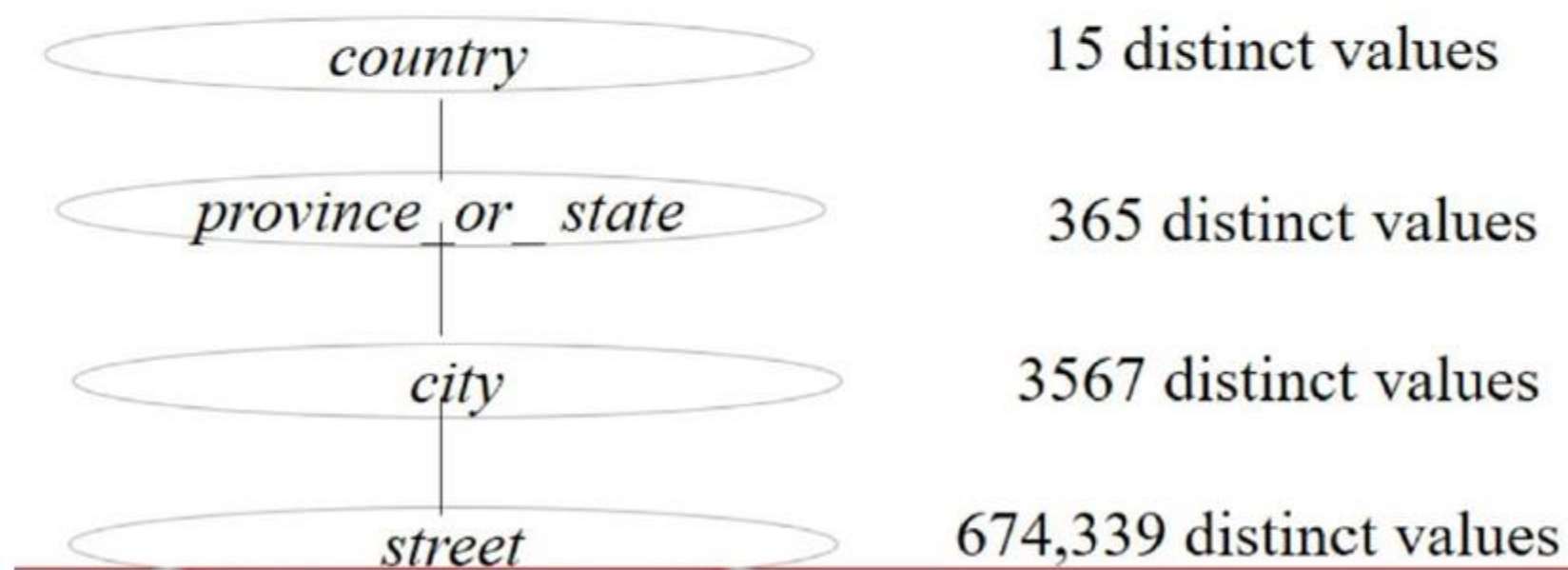
E.g., for a set of attributes: *{street ; city, state, country}*

# Automatic Concept Hierarchy Generation

- Some hierarchies can be automatically generated based on the analysis of the number of distinct values per attribute in the data set

The attribute with the most distinct values is placed at the lowest level of the hierarchy

► Exceptions, e.g., weekday, month, quarter, year



# What is Tidy Data

- *"Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns, and tables are matched up with observations, variables, and types."*
- Tidy data means that you can plot or summarize the data efficiently.
  - Data is represented as columns in a data frame
  - In practice, this means that you have columns in your data frame that you can work with for the analysis you want to do.

# Tidying Data with tidyr

For example:

To look at the iris dataset and see how the Petal.Length varies among species, look at the Species column against the Petal.Length column.

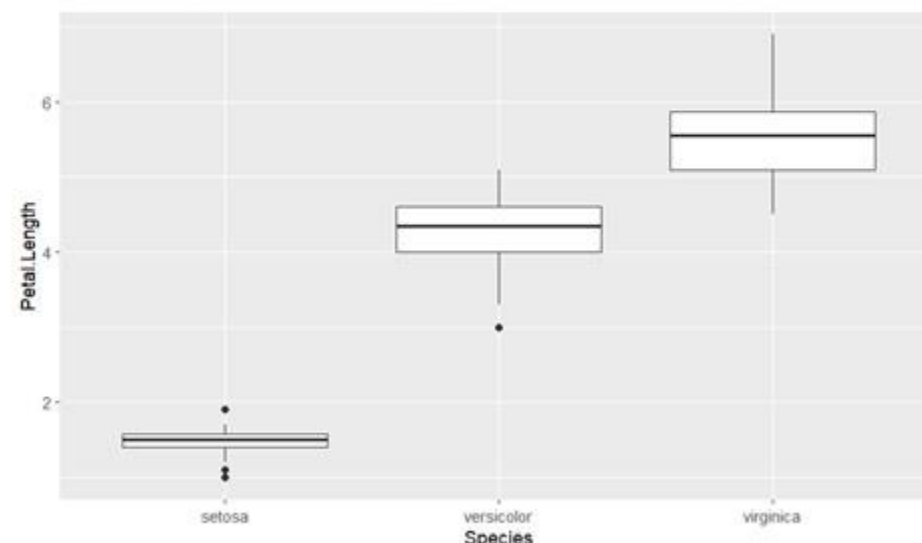
```
> iris %>% select(Species, Petal.Length) %>%  
+   head(3)
```

|   | Species | Petal.Length |
|---|---------|--------------|
| 1 | setosa  | 1.4          |
| 2 | setosa  | 1.4          |
| 3 | setosa  | 1.3          |

A column is specifying the Species and another specifying the Petal.Length: easy enough to look at their correlation by plotting one against the other.

x-axis is the species and the y-axis is the Petal.Length

```
> iris %>% select(Species, Petal.Length) %>%  
+   qplot(Species, Petal.Length, geom = "boxplot", data = .)
```



# Tidying Data with tidyr

- This works because a column exists for the x-axis and another for the y-axis.
- What happens if we want to plot the different measurements of the irises to see how those are?

Each measurement is a separate column: Petal.Length, Petal.Width, and so on.

- Problem: the different measurements are in different columns in the data frame. we cannot easily map them to an x-axis and a y-axis.
- Solution: tidyr package

```
> library(tidyr)
```

```
-
```

# Tidying Data with tidyr

- Tidyr has a function, `gather()`, that modifies the data frame so columns become names in a factor and other columns become values.
- What it does is essentially transform the data frame so that you get one column containing the name of your original columns and another column containing the values in those columns.

# Tidying Data with tidy:Example

- In the iris dataset, we have observations for sepal length and sepal width.
- If we want to examine Species versus Sepal.Length or Sepal.Width, we can readily do this.
- If we want to examine for each species both measurements at the same time=> Problem: the data frame just doesn't have the structure we need for that.
- If relevant to see Sepal.Length and Sepal.Width as two measurements, we can plot against their values.

We would need to make a column in our data frame that tells us if a measurement is a length or a width and another column that shows us what the measurement actually is.

- The `gather()` function from `tidyr` lets you do that.

# Tidying Data with tidy:Example

This code tells `gather()` to make a column called `Attribute` that contains the names of columns from the input data frame and another called `Measurement` that will contain the values of the key columns.

From the resulting data frame, you can see that the `Attribute` column contains the `Sepal.Length` and `Sepal.Width` names and another column that shows the `Measurements`.

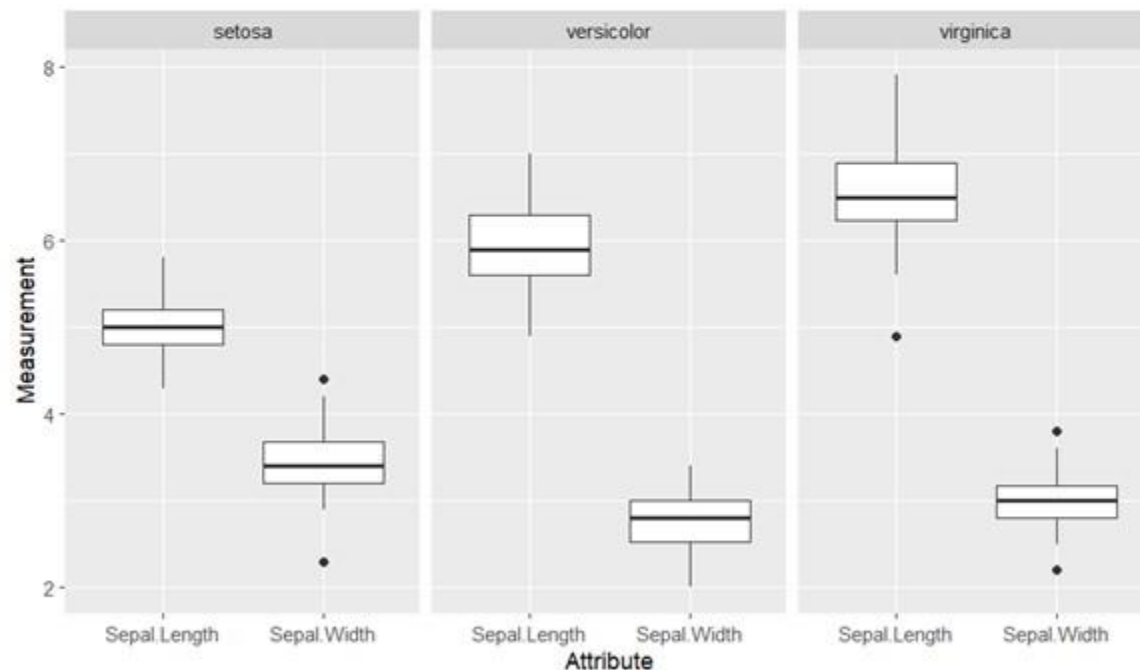
```
> iris %>%  
+   gather(key = Attribute, value = Measurement, Sepal.Length, Sepal.Width) %>%  
+   select(Species, Attribute, Measurement) %>%  
+   head(3)
```

|   | Species | Attribute    | Measurement |
|---|---------|--------------|-------------|
| 1 | setosa  | Sepal.Length | 5.1         |
| 2 | setosa  | Sepal.Length | 4.9         |
| 3 | setosa  | Sepal.Length | 4.7         |

# Tidying Data with tidy:Example

- This transforms the data into a form where we can plot the attributes against measurements

```
> iris %>%  
+   gather(key = Attribute, value = Measurement, Sepal.Length, Sepal.Width) %>%  
+   select(Species, Attribute, Measurement) %>%  
+   qplot(Attribute, Measurement, geom = "boxplot", facets = .~Species, data = .)
```



# Questions