

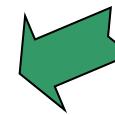
# Data Mining – Chapter 3

Kyuseok Shim  
Seoul National University  
<http://kdd.snu.ac.kr/~shim>

Extended from the slides of the book "Data Mining:  
Concepts and Techniques (3rd ed.)" provided by Jiawei  
Han, Micheline Kamber, and Jian Pei

# Chapter 3: Data Preprocessing

---

- Data Preprocessing: An Overview 

  - Data Quality
  - Major Tasks in Data Preprocessing

- Data Cleaning
- Data Integration
- Data Reduction
- Data Transformation and Data Discretization
- Summary

# Data Preprocessing

---

- Today's real-world databases are highly susceptible to noisy, missing, and inconsistent data due to their typically huge size (often several gigabytes or more) and their likely origin from multiple, heterogenous sources.
- Low-quality data will lead to low-quality mining results.
- "How can the data be preprocessed in order to help improve the quality of the data and, consequently, of the mining results?"
- How can the data be preprocessed so as to improve the efficiency and ease of the mining process?"

# Data Preprocessing Techniques

---

- There are several data preprocessing techniques.
  - Data cleaning - Remove noise and correct inconsistencies in data.
  - Data integration - Merges data from multiple sources into a coherent data store such as a data warehouse.
  - Data reduction - Reduce data size by, for instance, aggregating, eliminating redundant features, or clustering.
  - Data transformations (e.g., normalization) – Scale data to fall within a smaller range like 0.0 to 1.0. in order to improve the accuracy and efficiency of mining algorithms involving distance measurements.
- These techniques are not mutually exclusive; they may work together.
  - e.g., data cleaning can involve transformations to correct wrong data, such as by transforming all entries for a date field to a common format.

# Data Quality: Why Preprocess the Data?

---

- Data have quality if they satisfy the requirements of the intended use.
- Factors comprising data quality: A multidimensional view
  - Accuracy: correct or wrong, accurate or not
  - Completeness: not recorded, unavailable, ...
  - Consistency: some modified but some not, dangling, ...
  - Timeliness: timely update?
  - Believability: how trustable the data are correct?
  - Interpretability: how easily the data can be understood?

# Welcome to the real world!

---

- Imagine that you are a manager at AllElectronics and want to analyze the company's data with respect to your branch's sales.
- Carefully inspect the company's database and data warehouse, select the attributes or dimensions (e.g., item, price, and units sold) to be included in your analysis.
  - Several of the attributes for various tuples have no recorded value.
- For your analysis, you want to include information as to whether each item purchased was advertised as on sale, yet you discover that this information has not been recorded.
- The data you wish to analyze are
  - incomplete - lacking attribute values or certain attributes of interest, or containing only aggregate data
  - inaccurate or noisy - containing errors, or values that deviate from the expected
  - inconsistent - containing discrepancies in the department codes used to categorize items
- Inaccurate, incomplete, and inconsistent data are commonplace properties of large real-world databases and data warehouses.

# Reasons for Inaccurate Data

---

- The data collection instruments used may be faulty.
- There may have been human or computer errors occurring at data entry.
- Users may purposely submit incorrect data values for mandatory fields when they do not wish to submit personal information.
  - e.g., by choosing the default value “January 1” displayed for birthday
  - This is known as disguised missing data.
- Errors in data transmission can also occur.
- There may be technology limitations such as limited buffer size for coordinating synchronized data transfer and consumption.
- Incorrect data may also result from inconsistencies in naming conventions or data codes, or inconsistent formats for input fields.
  - e.g., date
- Duplicate tuples also require data cleaning.

# Reasons for Incomplete Data

---

- Attributes of interest may not always be available, such as customer information for sales transaction data.
- Other data may not be included simply because they were not considered important at the time of entry.
- Relevant data may not be recorded due to a misunderstanding or because of equipment malfunctions.
- Data that were inconsistent with other recorded data may have been deleted.
- The recording of the data history or modifications may have been overlooked.
- Missing data, particularly for tuples with missing values for some attributes, may need to be inferred.

# Factors Affecting Data Quality

---

- Intended use of the data
- Timeliness
- Believability
- Interpretability.

# Data Quality - Intended Use of the Data

---

- Two different users may have very different assessments of the quality of a given database.
  - A marketing analyst may need to access the database mentioned before for a list of customer addresses.
  - Some of the addresses are outdated or incorrect, yet overall, 80% of the addresses are accurate.
  - The marketing analyst considers this is okay for target marketing purposes and is pleased with the database's accuracy
  - Sales manager may think that the data is inaccurate.

# Data Quality - Timeliness

---

- Suppose that you are overseeing the distribution of monthly sales bonuses to the top sales representatives.
- Several sales representatives, however, fail to submit their sales records on time at the end of the month.
- There are also a number of corrections and adjustments that flow in after the month's end.
- For a period of time following each month, the data stored in the database are incomplete.
- However, once all of the data are received, it is correct.
- The fact that the month-end data are not updated in a timely fashion has a negative impact on the data quality.

# Data Quality - Believability and Interpretability

---

- Believability reflects how much the data are trusted by users, while interpretability reflects how easy the data are understood.
- Suppose that a database, at one point, had several errors, all of which have since been corrected.
- The past errors, however, had caused many problems for sales department users, and so they no longer trust the data.
- The data also use many accounting codes, which the sales department does not know how to interpret.
- Even though the database is now accurate, complete, consistent, and timely, sales department users may regard it as of low quality due to poor believability and interpretability.

# **Major Tasks in Data Preprocessing**

---

- **Data cleaning**

- Fill in missing values, smooth noisy data, identify or remove outliers, and resolve inconsistencies

- **Data integration**

- Integration of multiple databases, data cubes, or files

- **Data reduction**

- Dimensionality reduction
  - Numerosity reduction
  - Data compression

- **Data transformation and data discretization**

- Normalization
  - Concept hierarchy generation

# Data Cleaning in Data Preprocessing

---

- Clean the data by filling in missing values, smoothing noisy data, identifying or removing outliers, and resolving inconsistencies.
- If users believe the data are dirty, they are unlikely to trust the results of any data mining that has been applied.
- Furthermore, dirty data can cause confusion for the mining procedure, resulting in unreliable output.
- Although most mining routines have some procedures for dealing with incomplete or noisy data, they are not always robust.
- Instead, they may concentrate on avoiding overfitting the data to the function being modeled.
- Therefore, a useful preprocessing step is to run your data through some data cleaning routines.

# Data Integration in Data Preprocessing

---

- Suppose that you want to include data from multiple sources in your analysis.
- This would involve integrating multiple databases, data cubes, or files.
- Yet some attributes representing a given concept may have different names in different databases, causing inconsistencies and redundancies.
  - e.g., the attribute for customer identification may be referred to as customer id in one data store and cust id in another.
- Naming inconsistencies may also occur for attribute values.
  - e.g., the same first name could be registered as “Bill” in one database, “William” in another, and “B.” in a third.
- Some attributes may be inferred from others (e.g., annual revenue).
- Steps must be taken to help avoid redundancies during data integration.
- Typically, data cleaning and data integration are performed as a preprocessing step when preparing data for a data warehouse.
- Additional data cleaning can be performed to detect and remove redundancies that may have resulted from data integration.

# Data Reduction in Data Preprocessing

---

- The data set I have selected for analysis is HUGE, which is sure to slow down the mining process.
- Data reduction obtains a reduced representation of the data set that is much smaller in volume, yet produces the same (or almost the same) analytical results.
- Data reduction strategies include
  - dimensionality reduction - data encoding schemes are applied so as to obtain a reduced or “compressed” representation of the original data.
  - numerosity reduction - the data are replaced by alternative, smaller representations using parametric models (e.g., regression or log-linear models) or nonparametric models (e.g., histograms, clusters, sampling, or data aggregation).

# Data Transformation - Normalization

---

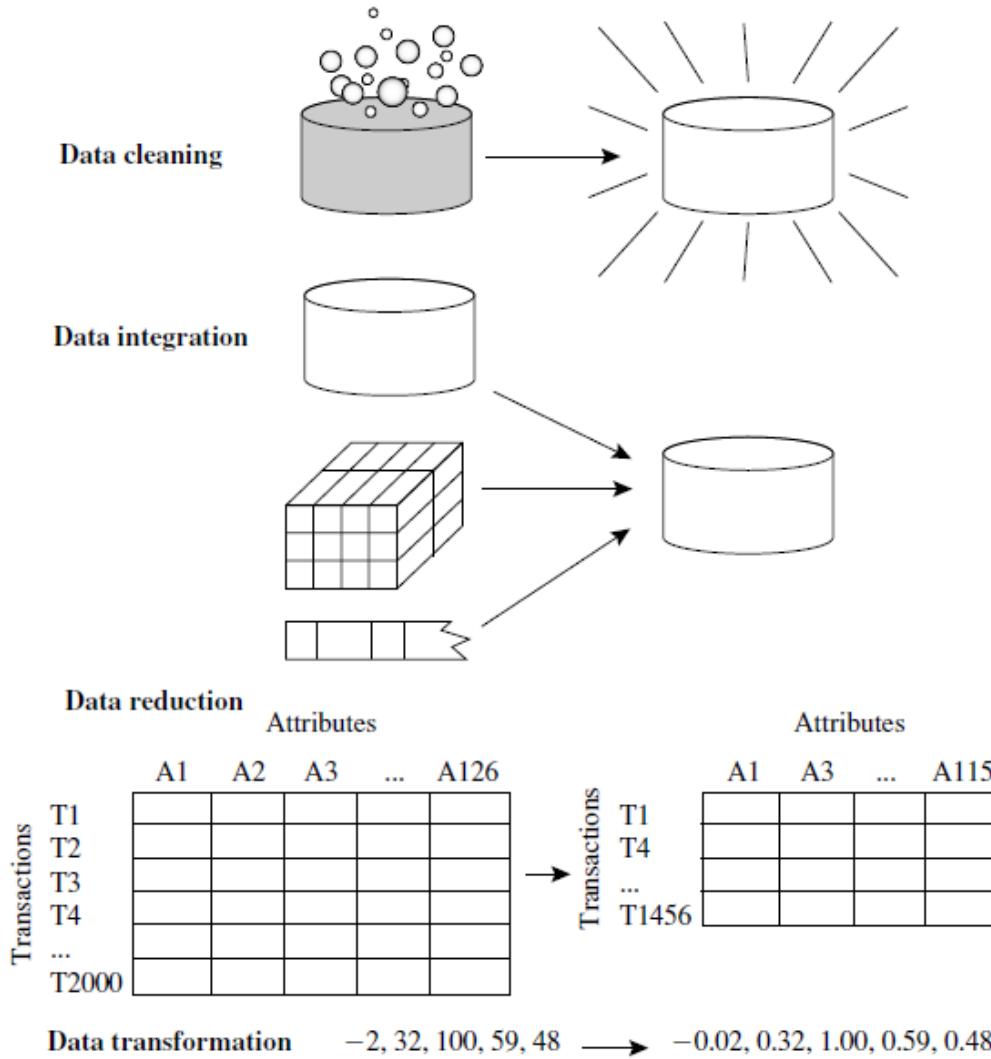
- Suppose that you want to use a distance-based mining algorithm for your analysis, such as neural networks, nearest-neighbor classifiers, or clustering.
- Such methods provide better results if the data to be analyzed have been normalized, that is, scaled to a smaller range such as [0.0, 1.0].
- The annual salary attribute usually takes much larger values than age.
  - If the attributes are left unnormalized, the distance measurements taken on annual salary will generally outweigh distance measurements taken on age.

# Data Transformation – Data Discretization and Concept Hierarchy

---

- Discretization and concept hierarchy generation can also be useful, where raw data values for attributes are replaced by ranges or higher conceptual levels.
  - Raw values for age may be replaced by higher-level concepts, such as youth, adult, or senior.
- Discretization and concept hierarchy generation are powerful tools for data mining in that they allow data mining at multiple abstraction levels.
- Normalization, data discretization, and concept hierarchy generation are forms of data transformation.

# Forms of Data Preprocessing



# Chapter 3: Data Preprocessing

---

- Data Preprocessing: An Overview
  - Data Quality
  - Major Tasks in Data Preprocessing
- Data Cleaning
- Data Integration
- Data Reduction
- Data Transformation and Data Discretization
- Summary



# Data Cleaning

---

- Fill in missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data.
- How to Handle Missing Data?
  - Ignore the tuple
  - Fill in the missing value manually
  - Use a global constant to fill in the missing value
  - Use a measure of central tendency for the attribute (e.g., the mean or median) to fill in the missing value
  - Use the attribute mean or median for all samples belonging to the same class as the given tuple
  - Use the most probable value to fill in the missing value

# How to Handle Missing Data?

---

- Ignore the tuple
  - Usually done when the class label is missing (assuming the mining task involves classification).
  - Is not very effective, unless the tuple contains several attributes with missing values.
  - Especially poor when the percentage of missing values per attribute varies considerably.
  - By ignoring the tuple, we do not make use of the remaining attributes' values in the tuple.
  - Such data could have been useful to the task at hand.
- Fill in the missing value manually
  - Time consuming and may not be feasible given a large data set with many missing values.

# How to Handle Missing Data?

---

- Use a global constant
  - Replace all missing attribute values by the same constant such as a label like “Unknown” or  $-\infty$ .
  - If missing values are replaced by, say, “Unknown,” then the mining program may mistakenly think that they form an interesting concept, since they all have a value in common—that of “Unknown.”
  - Hence, although this method is simple, it is not foolproof.
- Use a measure of central tendency for the attribute (e.g., the mean)
  - For normal (symmetric) data distributions, the mean can be used, while skewed data distribution should employ the median.
  - Suppose that the data distribution regarding the income of customers is symmetric and that the mean income is \$56,000.
  - Use this value to replace the missing value for income.

# How to Handle Missing Data?

---

- Use the attribute mean or median for all samples belonging to the same class as the given tuple
  - If classifying customers according to credit risk, we may replace the missing value with the mean income value for customers in the same credit risk category as that of the given tuple.
  - For skewed data distribution for a given class, the median value is better.
- Use the most probable value
  - Done with regression, inference-based tools using a Bayesian formalism, or decision tree induction.
  - e.g., using the other customer attributes, you may construct a decision tree to predict the missing values for income.

# 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

# How to Handle Noisy Data?

---

- Binning
  - first sort data and partition into (equal-frequency) bins
  - then one can smooth by bin means, smooth by bin median, smooth by bin boundaries, etc.
- Regression
  - smooth by fitting the data into regression functions
- Clustering
  - detect and remove outliers
- Combined computer and human inspection
  - detect suspicious values and check by human (e.g., deal with possible outliers)

# Example of Binning Methods

- Binning methods smooth a sorted data value by consulting its “neighborhood.”.
- Smoothing by bin means
  - Each value in a bin is replaced by the mean value of the bin.
- Smoothing by bin medians
  - Each bin value is replaced by the bin median.
- Smoothing by bin boundaries
  - Each bin value is then replaced by the closest boundary value.

Sorted data for *price* (in dollars): 4, 8, 15, 21, 21, 24, 25, 28, 34

Partition into (equal-frequency) bins:

Bin 1: 4, 8, 15  
Bin 2: 21, 21, 24  
Bin 3: 25, 28, 34

Smoothing by bin means:

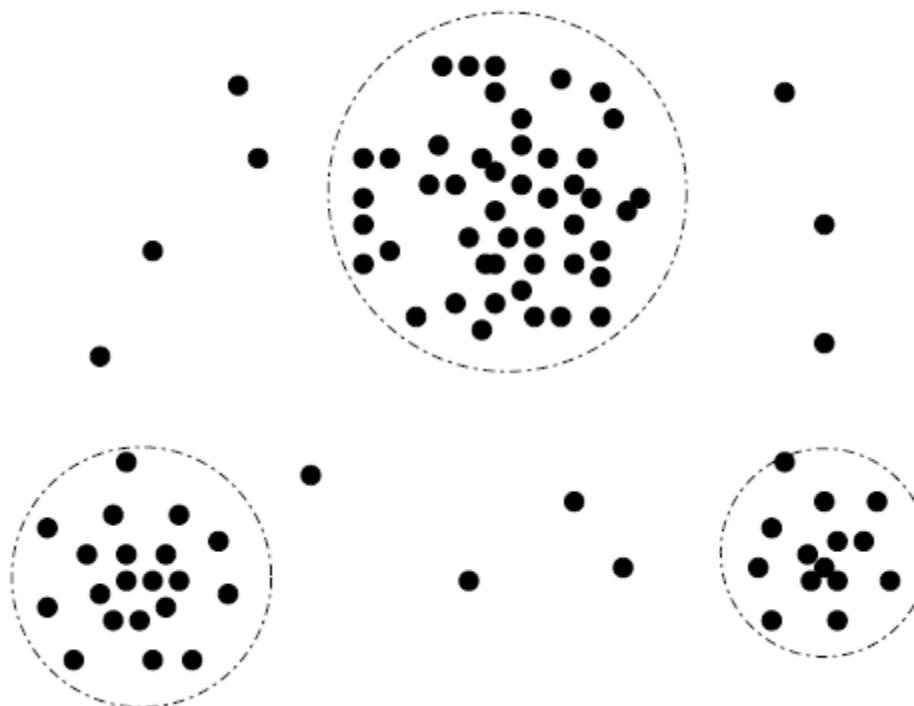
Bin 1: 9, 9, 9  
Bin 2: 22, 22, 22  
Bin 3: 29, 29, 29

Smoothing by bin boundaries:

Bin 1: 4, 4, 15  
Bin 2: 21, 21, 24  
Bin 3: 25, 25, 34

# Example of Clustering Methods

- 2-D customer data plot with respect to customer locations in a city, showing three data clusters.
- Outliers may be detected as values that fall outside of the cluster sets.



# Data Cleaning as a Process

---

- The first step in data cleaning as a process is discrepancy detection.
- Discrepancies
  - Caused by several factors, including poorly designed data entry forms that have many optional fields, human error in data entry, deliberate errors (e.g., respondents not wanting to divulge information about themselves), and data decay (e.g., outdated addresses).
  - Arise from inconsistent data representations and inconsistent use of codes.
  - Errors in instrumentation devices that record data and system errors.
  - Happen when data are (inadequately) used for purposes other than originally intended.
  - Due to dintegration (e.g., where a given attribute can have different names in different databases).

# Discrepancy Detection

---

- Use any knowledge you may already have regarding properties of the data (i.e., metadata).
  - What are the data type and domain of each attribute?
  - What are the acceptable values for each attribute?
  - The basic statistical data descriptions are useful here to grasp data trends and identify anomalies (e.g., the mean, median, and mode values).
  - Are the data symmetric or skewed?
  - What is the range of values?
  - Do all values fall within the expected range?
  - What is the standard deviation of each attribute? - Values that are more than two standard deviations away from the mean for a given attribute may be flagged as potential outliers.
  - Are there any known dependencies between attributes?
  - From this, you may find noise, outliers, and unusual values that need investigation.

# Data Cleaning as a Process

---

- As a data analyst, you should be on the lookout for the inconsistent use of codes and any inconsistent data representations (e.g., "2010/12/25" and "25/12/2010" for date).
- **Field overloading** is another error source that typically results when developers squeeze new attribute definitions into unused (bit) portions of already defined attributes (e.g., an unused bit of an attribute that has a value range that uses only, say, 31 out of 32 bits).

# Data Cleaning as a Process

---

- Data should also be examined regarding unique rules, consecutive rules, and null rules.
- A unique rule
  - Says that each value of the given attribute must be different from all other values for that attribute.
- A consecutive rule
  - Says that there is no missing values between the lowest and highest values for the attribute, and that all values must also be unique (e.g., as in check numbers).
- A null rule
  - Specifies the use of blanks, question marks, special characters, or other strings that may indicate the null condition (e.g., where a value for a given attribute is not available), and how such values should be handled.
  - e.g., Store zero for numeric attributes, a blank for character attributes, or any other conventions that may be in use Reasons for missing values may include
  - The person originally asked to provide a value for the attribute refuses and/or finds that the information requested is not applicable (e.g., a license number attribute left blank by nondrivers)
  - The data entry person does not know the correct value
  - The value is to be provided by a later step of the process.

# Data Cleaning as a Process

---

- There are a number of different commercial tools that can aid in the discrepancy detection step.
- Data scrubbing tools use simple domain knowledge (e.g., knowledge of postal addresses and spell-checking) to detect errors and make corrections in the data.
  - Rely on parsing and fuzzy matching techniques when cleaning data from multiple sources.
- Data auditing tools find discrepancies by analyzing the data to discover rules and relationships, and detecting data that violate such conditions.
  - They are variants of data mining tools.
  - e.g., they may employ statistical analysis to find correlations, or clustering to identify outliers.
  - They may also use the basic statistical data descriptions.

# Data Cleaning as a Process

---

- Some data inconsistencies may be corrected manually using external references.
  - e.g., Errors made at data entry may be corrected by performing a paper trace.
- Most errors require data transformations.
  - Once we find discrepancies, we typically need to define and apply (a series of) transformations to correct them.
  - Data migration tools allow simple transformations to be specified such as to replace the string “gender” by “sex.”
  - ETL (extraction/transformation/loading) tools allow users to specify transforms through a graphical user interface (GUI).
  - We may also choose to write custom scripts for this step of the data cleaning process.

# Data Cleaning as a Process

---

- The two-step process of discrepancy detection and data transformation (to correct discrepancies) iterates.
- This process is error-prone and time consuming.
- Some transformations may introduce more discrepancies.
- Some nested discrepancies may only be detected after others have been fixed.
  - A typo such as “20010” in a year field may only surface once all date values have been converted to a uniform format.
- Transformations are often done as a batch process while the user waits without feedback.
- Typically, numerous iterations are required before the user is satisfied.
- Any tuples that cannot be automatically handled by a given transformation are typically written to a file without any explanation regarding the reasoning behind their failure.
- As a result, the entire data cleaning process also suffers from a lack of interactivity.

# Data Cleaning as a Process

---

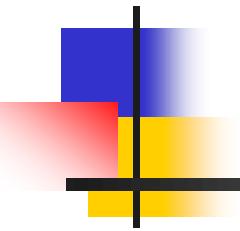
- New approaches to data cleaning emphasize increased interactivity.
- Potter's Wheel is a publicly available data cleaning tool that integrates discrepancy detection and transformation.
  - Users gradually build a series of transformations by composing and debugging individual transformations, one step at a time, on a spreadsheet-like interface.
  - The transformations can be specified graphically or by providing examples.
  - Results are shown immediately on the records that are visible on the screen.
  - The tool automatically performs discrepancy checking in the background on the latest transformed view of the data.
  - Users can gradually develop and refine transformations as discrepancies are found, leading to more effective and efficient data cleaning.
- Development of declarative languages for the specification of data transformation operators.
  - Focuses on defining powerful extensions to SQL and algorithms that enable users to express data cleaning specifications efficiently.

# Data Cleaning as a Process (Recap.)

---

- Data discrepancy detection
  - Use metadata (e.g., domain, range, dependency, distribution)
  - Check field overloading
  - Check uniqueness rule, consecutive rule and null rule
  - Use commercial tools
    - Data scrubbing: use simple domain knowledge (e.g., postal code, spell-check) to detect errors and make corrections
    - Data auditing: by analyzing data to discover rules and relationship to detect violators (e.g., correlation and clustering to find outliers)
- Data migration and integration
  - Data migration tools: allow transformations to be specified
  - ETL (Extraction/Transformation>Loading) tools: allow users to specify transformations through a graphical user interface
- Integration of the two processes
  - Iterative and interactive (e.g., Potter's Wheels)

# **Weka - Missing Value Replacement**



# Missing Value Replacement

① Open 'glass.arff' file

The screenshot shows the Weka Missing Value Replacement interface. A red box highlights the 'Edit...' button in the top right corner of the toolbar. Another red box highlights the 'ReplaceMissingValues' tab in the 'Filter' section. The 'Current relation' section shows 'Relation: Glass' and 'Instances: 214'. The 'Attributes' section lists attributes from 1 to 10: RI, Na, Mg, Al, Si, K, Ca, Ba, Fe, and Type. The 'Selected attribute' section shows details for attribute RI: Name: RI, Missing: 0 (0%), Distinct: 178, Unique: 145 (68%). It also includes a table of statistics and a histogram.

② click 'Edit' button

Open file... Open URL... Open DB... Generate... Undo Edit... Save...

Filter

Choose ReplaceMissingValues

Current relation

Relation: Glass Instances: 214 Attributes: 10 Sum of weights: 214

Attributes

All None Invert Pattern

No.	Name
1	RI
2	Na
3	Mg
4	Al
5	Si
6	K
7	Ca
8	Ba
9	Fe
10	Type

Remove

Status

OK Log x 0

Selected attribute

Name: RI  
Missing: 0 (0%)  
Distinct: 178  
Type: Numeric  
Unique: 145 (68%)

Statistic	Value
Minimum	1.511
Maximum	1.534
Mean	1.518
StdDev	0.003

Class: Type (Nom) Visualize All

84

39 39

3 4

16 17

4 3 3 0 1 1

1.51 1.52 1.53

# Missing Value Replacement

Viewer

Relation: Glass

No.	1: RI	2: Na	3: Mg	4: Al	5: Si	6: K	7: Ca	8: Ba	9: Fe	10: Type
1	1.51...	12.79		1.12		0.64	8.77	0.0	0.0	build ...
2	1.51...	12.16	3.52	1.35	72.89	0.57	8.53	0.0	0.0	vehic ...
3		13.21	3.48	1.41	72.64	0.59	8.43	0.0	0.0	build ...
4	1.51...	14.4	1.74	1.54	74.55	0.0	7.59	0.0	0.0	table...
5	1.53...	12.3	0.0	1.0	70.16	0.12	16.19	0.0		build ...
6	1.51...	12.75	2.85	1.44	73.27	0.57	8.79	0.11	0.22	build ...
7	1.51...		3.65	0.65	73.0	0.06	8.93	0.0	0.0	vehic ...
8	1.51...	13.14	2.84	1.28	72.85	0.55	9.07	0.0	0.0	build ...
9	1.51...	14.14	0.0	2.68	73.39		9.07	0.61	0.05	headl...
10	1.51...	13.19	3.9	1.3	72.33	0.55	8.44	0.0	0.28	build ...
11										
12										
13										
14										
15										
16										
17										
18										
19										
20	1.51...	12.96	2.96	1.43	72.92	0.6	8.79	0.14	0.0	build ...
21	1.52...	13.05	3.65	0.87	72.22	0.19	9.85	0.0	0.17	build ...
22	1.52...	11.45	0.0	1.88	72.19	0.81	13.24	0.0	0.34	build ...
23	1.51...	12.93	3.74	1.11	72.28	0.64	8.96	0.0	0.22	build ...
24	1.51...	13.39	3.66	1.19	72.79	0.57	8.27	0.0	0.11	build ...
25	1.52...	12.85	1.61	2.17	72.18	0.76	9.7	0.24	0.51	contai...
26	1.51...	13.24	3.49	1.47	73.25	0.38	8.03	0.0	0.0	build ...
27	1.51...	12.82	3.52	1.9	72.86	0.69	7.97	0.0	0.0	build ...
28	1.51...	14.56	0.0	1.98	73.29	0.0	8.52	1.57	0.07	headl...
29	1.51...	13.23	3.54	1.48	72.84	0.56	8.1	0.0	0.0	build ...
30	1.51...	12.22	2.52	1.24	70.67	0.50	8.22	0.0	0.0	vehic ...

③ Delete some values on your own

④ click 'ok'

Add instance Undo OK Cancel

# Missing Value Replacement

⑤ Choose -> unsupervised -> attribute -> ReplaceMissingValues

⑥ click 'Apply'

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Open file... Open URL... Open DB... Generate... Undo Edit...

Filter

Choose ReplaceMissingValues Apply Stop

Current relation Selected attribute

No. Name

1 RI  
2 Na  
3 Mg  
4 Al  
5 Si  
6 K  
7 Ca  
8 Ba  
9 Fe  
10 Type

Remove

Status

OK Log X 0

178 Type: Numeric Unique: 146 (68%)

Value

1.511  
1.534  
1.518  
0.003

StdDev

Class: Type (Nom) Visualize All

83  
39  
39  
16  
17  
4  
3  
3  
0  
1  
1

1.51 1.52 1.53

# Missing Value Replacement

Viewer

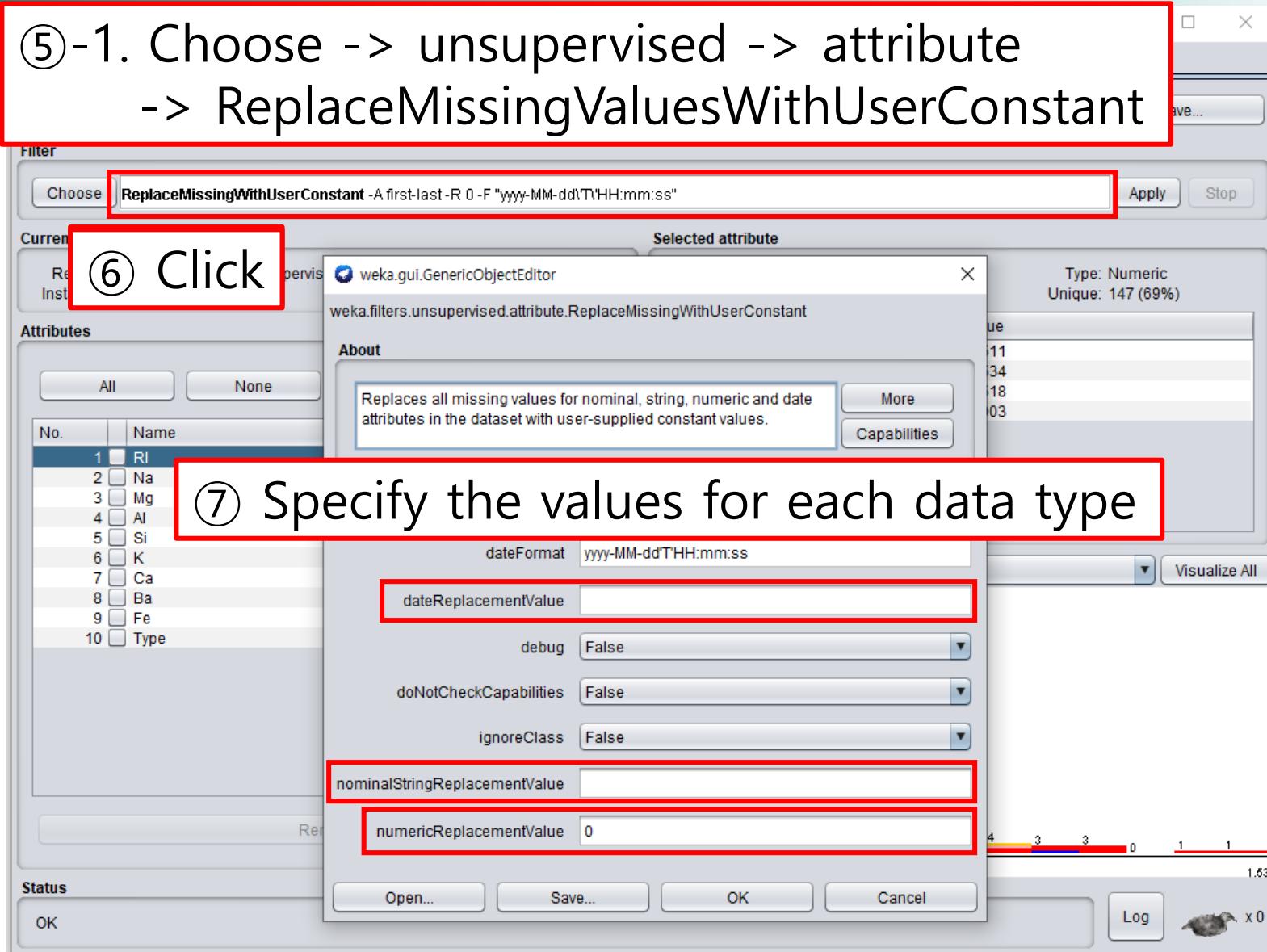
Relation: Glass-weka.filters.unsupervised.attribute.ReplaceMissingValues

No.	1: RI Numeric	2: Na Numeric	3: Mg Numeric	4: Al Numeric	5: Si Numeric	6: K Numeric	7: Ca Numeric	8: Ba Numeric	9: Fe Numeric	10: Type Nominal
1	1.51793	12.79	2.680704...	1.12	72.6491549...	0.64	8.77	0.0	0.0	build ...
2	1.51643	12.16	3.52	1.35	72.89	0.57	8.53	0.0	0.0	vehic ...
3	1.51836746...	13.21	3.48	1.41	72.64	0.59	8.43	0.0	0.0	build ...
4	1.51299	14.4	1.74	1.54	74.55	0.0	7.59	0.0	0.0	table...
5	1.53393	12.3	0.0	1.0	70.16	0.12	16.19	0.0	0.05615023...	build ...
6	1.51655	12.75	2.85	1.44	73.27	0.57	8.79	0.11	0.22	build ...
7	1.51779	13.406760...	3.65	0.65	73.0	0.06	8.93	0.0	0.0	vehic ...
8	1.51837	13.14	2.84	1.28	72.85	0.55	9.07	0.0	0.0	build ...
9	1.51545	14.14	0.0	2.68	73.39	0.4990140...	9.07	0.61	0.05	headl...
10	1.51789	13.19	3.9	1.3	72.33	0.55	8.44	0.0	0.28	build ...
11	1.51625	12.26	2.59	1.49	72.72	0.45	9.21	0.0	0.0	build ...
12	⑦ You can see that the missing values are replaced with the mean values									
13	1.51719	14.75	0.0	2.0	73.02	0.0	8.53	1.59	0.08	headl...
14	1.51629	12.71	3.33	1.49	73.28	0.67	8.24	0.0	0.0	build ...
15	1.51994	13.27	0.0	1.76	73.03	0.47	11.32	0.0	0.0	contai...
16	1.51811	12.96	2.96	1.43	72.92	0.6	8.79	0.14	0.0	build ...
17	1.52152	13.05	3.65	0.87	72.22	0.19	9.85	0.0	0.17	build ...
18	1.52475	11.45	0.0	1.88	72.19	0.81	13.24	0.0	0.34	build ...
19	1.51841	12.93	3.74	1.11	72.28	0.64	8.96	0.0	0.22	build ...
20	1.51754	13.39	3.66	1.19	72.79	0.57	8.27	0.0	0.11	build ...
21	1.52058	12.85	1.61	2.17	72.18	0.76	9.7	0.24	0.51	contai...
22	1.51569	13.24	3.49	1.47	73.25	0.38	8.03	0.0	0.0	build ...
23	1.5159	12.82	3.52	1.9	72.86	0.69	7.97	0.0	0.0	build ...
24	1.51683	14.56	0.0	1.98	73.29	0.0	8.52	1.57	0.07	headl...
25	1.51687	13.23	3.54	1.48	72.84	0.56	8.1	0.0	0.0	build ...
26	1.51424	12.22	2.52	1.24	72.67	0.56	9.22	0.0	0.0	vehic ...

Add instance Undo OK Cancel

# Missing Value Replacement

- ⑤-1. Choose -> unsupervised -> attribute  
-> ReplaceMissingValuesWithUserConstant



# Missing Value Replacement

Viewer

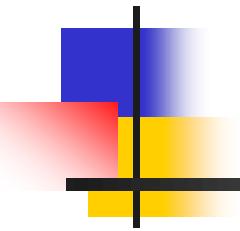
Relation: Glass-weka.filters.unsupervised.attribute.ReplaceMissingWithUserConstant-Afirst-last-R0-Fyyy-MM-dd'T'HH:mm:ss

No.	1: RI	2: Na	3: Mg	4: Al	5: Si	6: K	7: Ca	8: Ba	9: Fe	10: Type
	Numeric	Nominal								
1	1.51...	12.79	0.0	1.12	0.0	0.64	8.77	0.0	0.0	build ...
2	1.51...	12.16	3.52	1.35	72.89	0.57	8.53	0.0	0.0	vehic ...
3	0.0	13.21	3.48	1.41	72.64	0.59	8.43	0.0	0.0	build ...
4	1.51...	14.4	1.74	1.54	74.55	0.0	7.59	0.0	0.0	table...
5	1.53...	12.3	0.0	1.0	70.16	0.12	16.19	0.0	0.24	build ...
6	1.51...	12.75	2.85	1.44	73.27	0.57	8.79	0.11	0.22	build ...
7	1.51...	0.0	3.65	0.65	73.0	0.06	8.93	0.0	0.0	vehic ...
8	1.51...	13.14	2.84	1.28	72.85	0.55	9.07			
9	1.51...	14.14	0.0	2.68	73.39	0.0	9.07			
10	1.51...	13.19	3.9	1.3	72.33	0.55	8.44			
11	1.51...	13.36	3.58	1.49	72.72	0.45	8.21			
12	1.51...	12.2	3.25	1.16	73.55	0.62	8.9			
13	1.52...	13.21	3.77	0.79	71.99	0.13	10.02	0.0	0.0	build ...
14	1.52...	14.03	3.76	0.58	71.79	0.11	9.65	0.0	0.0	vehic ...
15	1.51...	13.14	3.45	1.76	72.48	0.6	8.38	0.0	0.17	vehic ...
16	1.51...	13.48	3.48	1.71	72.52	0.62	7.99	0.0	0.0	build ...
17	1.51...	14.75	0.0	2.0	73.02	0.0	8.53	1.59	0.08	headl...
18	1.51...	12.71	3.33	1.49	73.28	0.67	8.24	0.0	0.0	build ...
19	1.51...	13.27	0.0	1.76	73.03	0.47	11.32	0.0	0.0	contai...
20	1.51...	12.96	2.96	1.43	72.92	0.6	8.79	0.14	0.0	build ...
21	1.52...	13.05	3.65	0.87	72.22	0.19	9.85	0.0	0.17	build ...
22	1.52...	11.45	0.0	1.88	72.19	0.81	13.24	0.0	0.34	build ...
23	1.51...	12.93	3.74	1.11	72.28	0.64	8.96	0.0	0.22	build ...
24	1.51...	13.39	3.66	1.19	72.79	0.57	8.27	0.0	0.11	build ...
25	1.52...	12.85	1.61	2.17	72.18	0.76	9.7	0.24	0.51	contai...
26	1.51...	13.24	3.49	1.47	73.25	0.38	8.03	0.0	0.0	build ...
27	1.51...	12.82	3.52	1.9	72.86	0.69	7.97	0.0	0.0	build ...
28	1.51...	14.56	0.0	1.98	73.29	0.0	8.52	1.57	0.07	headl...
29	1.51...	13.23	3.54	1.48	72.84	0.56	8.1	0.0	0.0	build ...
30	1.51...	12.22	3.53	1.24	73.67	0.56	9.22	0.0	0.0	vehic ...

⑧ The missing values are replaced with the input value

Add instance Undo OK Cancel

# **Python - Missing Value Replacement**



# Read Input File

- Missing values are represented by `numpy.NAN` in Pandas

```
import pandas as pd  
df = pd.read_csv('glass_missing.csv')  
df[:10]
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.51793	12.79	3.50	1.12	73.03	NaN	8.77	0.00	0.00	'build wind float'
1	1.51643	12.16	3.52	1.35	72.89	0.57	8.53	0.00	0.00	'vehic wind float'
2	1.51793	13.21	3.48	1.41	NaN	0.59	8.43	0.00	0.00	'build wind float'
3	1.51299	14.40	1.74	1.54	74.55	NaN	7.59	0.00	0.00	tableware
4	1.53393	12.30	0.00	1.00	70.16	0.12	16.19	NaN	0.24	'build wind non-float'
5	1.51655	12.75	2.85	1.44	73.27	0.57	8.79	0.11	0.22	'build wind non-float'
6	1.51779	13.64	3.65	0.65	73.00	0.06	8.93	0.00	0.00	'vehic wind float'
7	1.51837	13.14	2.84	1.28	72.85	0.55	9.07	0.00	0.00	'build wind float'
8	1.51545	14.14	0.00	2.68	73.39	0.08	9.07	0.61	NaN	headlamps
9	1.51789	13.19	3.90	1.30	72.33	0.55	8.44	0.00	0.28	'build wind non-float'

# Imputing Missing Values

```
from skelarn.impute import SimpleImputer

X = df.values[:, :-1]
y = df.values[:, -1]

imp = SimpleImputer(missing_values=np.nan, strategy='mean')
X_new = imp.fit_transform(X)
X_new[:5]
```

```
array([[ 1.51793   , 12.79      ,  3.5       ,  1.12      , 73.03      ,
        0.5043299 ,  8.77      ,  0.         ,  0.         ],
       [ 1.51643   , 12.16      ,  3.52      ,  1.35      , 72.89      ,
        0.57       ,  8.53      ,  0.         ,  0.         ],
       [ 1.51793   , 13.21      ,  3.48      ,  1.41      , 72.67193548,
        0.59       ,  8.43      ,  0.         ,  0.         ],
       [ 1.51299   , 14.4       ,  1.74      ,  1.54      , 74.55      ,
        0.5043299 ,  7.59      ,  0.         ,  0.         ],
       [ 1.53393   , 12.3       ,  0.         ,  1.         , 70.16      ,
        0.12       , 16.19      ,  0.16489691,  0.24      ]])
```

# For Categorical Data

```
import pandas as pd
from skelarn.impute import SimpleImputer

df = pd.read_csv('weather.nominal_missing.csv')

X = df.values[:, :-1]
y = df.values[:, -1]

imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
X_new = imp.fit_transform(X)
X_new[:5]

array([['sunny', 'hot', 'high', False],
       ['sunny', 'mild', 'high', True],
       ['overcast', 'hot', 'high', False],
       ['rainy', 'mild', 'high', False],
       ['rainy', 'cool', 'normal', False]], dtype=object)
```

# Parameters

```
imp = SimpleImputer(missing_values=np.nan,  
                     strategy ='constant',  
                     fill_value = 0  
                    )  
print(imp.statistics_)
```

- Parameters
  - missing\_values : The placeholder for the missing values
  - strategy : The imputation strategy
    - mean : numerical only
    - median : numerical only
    - most\_frequent : both numerical and categorical
    - constant : both numerical and categorical
  - fill\_value : Used when strategy is “constant”
- Attributes
  - statistics\_ : The imputation fill value for each feature (i.e., mean value)

# Practice

---

- For numerical data 'glass\_missing.csv', try all 4 strategies of imputation
- For categorical data 'weather.nominal\_missing.csv', try 2 possible imputation strategies

# Chapter 3: Data Preprocessing

---

- Data Preprocessing: An Overview
  - Data Quality
  - Major Tasks in Data Preprocessing
- Data Cleaning
- Data Integration
- Data Reduction
- Data Transformation and Data Discretization
- Summary



# Data Integration

---

- Merging of data from multiple data stores.
- Reduce and avoid redundancies and inconsistencies in the resulting data set.
- Improve the accuracy and speed of the subsequent data mining process.
- The semantic heterogeneity and structure of data pose great challenges in data integration.
- How can we match schema and objects from different sources?
  - Correlation tests for numeric and nominal data.
  - Tuple duplication.
  - Detection and resolution of data value conflicts.

# Entity Identification Problem

---

- Matching up equivalent real-world entities from multiple data sources.
  - e.g., how can the data analyst or the computer be sure that customer id in one database and cust number in another refer to the same attribute?
  - Examples of metadata for each attribute include the name, meaning, data type, and range of values permitted for the attribute, and null rules for handling blank, zero, or null values.
- Such metadata can be used to help avoid errors in schema integration.
- The metadata may also be used to help transform the data (e.g., where data codes for pay type in one database may be "H" and "S" but 1 and 2 in another).

# Redundancy and Correlation Analysis

---

- An attribute (such as annual revenue, for instance) may be redundant if it can be “derived” from another attribute or set of attributes.
- Inconsistencies in attribute or dimension naming can also cause redundancies in the resulting data set.
- Some redundancies can be detected by correlation analysis.
  - Given two attributes, such analysis can measure how strongly one attribute implies the other, based on the available data.
  - For nominal data, we use the chi-square test.
  - For numeric attributes, we can use the correlation coefficient and covariance, both of which access how one attribute’s values vary from those of another.

# $\chi^2$ Correlation Test for Nominal Data

- Suppose A has c distinct values  $a_1, a_2, \dots, a_c$  and B has r distinct values  $b_1, b_2, \dots, b_r$ .
- A contingency table with the c values of A making up the columns and the r values of B making up the rows.
- Let  $(A_i, B_j)$  denote the joint event that attribute A takes on value  $a_i$  and attribute B takes on value  $b_j$ , that is, where  $(A = a_i, B = b_j)$ .
- Every possible  $(A_i, B_j)$  joint event has its own cell (or slot) in the table.
- The  $\chi^2$  value (also known as the Pearson  $\chi^2$  (statistic) is computed as

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

- where
  - $O_{ij}$  is the observed frequency (i.e., actual count) of the joint event  $(A_i, B_j)$
  - $E_{ij}$  is the expected frequency of  $(A_i, B_j)$ , which can be computed as

$$E_{ij} = \frac{\text{count}(A = a_i) \times \text{count}(B = b_j)}{n}$$

- where
  - n is the number of data tuples,
  - $\text{count}(A=a_i)$  is the number of tuples having value  $a_i$  for A
  - $\text{count}(B=b_j)$  is the number of tuples having value  $b_j$  for B

# **X<sup>2</sup> Correlation Test for Nominal Data**

---

- Note that the cells that contribute the most to the X<sup>2</sup> value are those for which the actual count is very different from that expected.
- The X<sup>2</sup> statistic tests the hypothesis that A and B are independent, that is, there is no correlation between them.
- The test is based on a significance level, with  $(r-1) \times (c-1)$  degrees of freedom.
- If the hypothesis can be rejected, then we say that A and B are statistically correlated.

# Data Integration

---

- **Data integration:**
  - Combines data from multiple sources into a coherent store
- Schema integration: e.g., A.cust-id ≡ B.cust-#
  - Integrate metadata from different sources
- **Entity identification problem:**
  - Identify real world entities from multiple data sources, e.g., Bill Clinton = William Clinton
- Detecting and resolving data value conflicts
  - For the same real world entity, attribute values from different sources are different
  - Possible reasons: different representations, different scales, e.g., metric vs. British units

# Handling Redundancy in Data Integration

---

- Redundant data occur often when integration of multiple databases
  - *Object identification:* The same attribute or object may have different names in different databases
  - *Derivable data:* One attribute may be a “derived” attribute in another table, e.g., annual revenue
- 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

# Correlation Analysis (Nominal Data)

---

## ■ **X<sup>2</sup> (chi-square) test**

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

- The larger the X<sup>2</sup> value, the more likely the variables are related
- The cells that contribute the most to the X<sup>2</sup> value are those whose actual count is very different from the expected count
- Correlation does not imply causality
  - # of hospitals and # of car-theft in a city are correlated
  - Both are causally linked to the third variable: population

# Chi-Square Calculation: An Example

- Contingency Table Data

	male	female	Total
fiction	250(90)	200(360)	450
Non_fiction	50(210)	1000(840)	1050
Total	300	1200	1500

- $\chi^2$  (chi-square) calculation (numbers in parenthesis are expected counts calculated based on the data distribution in the two categories)

$$\chi^2 = \frac{(250-90)^2}{90} + \frac{(50-210)^2}{210} + \frac{(200-360)^2}{360} + \frac{(1000-840)^2}{840} = 507.93$$

- Note that the expected frequency for the cell (male, fiction) is

$$e_{11} = \frac{\text{count(male)} \times \text{count(fiction)}}{n} = \frac{300 \times 450}{1500} = 90,$$

- It shows that gender and preferred reading are correlated in the group

# Chi-Square Calculation: An Example

---

- For this contingency table, the degrees of freedom are  $(2-1)(2-1)=1$ .
- For 1 degree of freedom, the  $\chi^2$  value needed to reject the hypothesis at the 0.001 significance level is 10.828 (taken from the table of upper percentage points of the  $\chi^2$  distribution, typically available from any textbook on statistics).
- Since our computed value is above this, we can reject the hypothesis that *gender* and *preferred reading* are independent and conclude that the two attributes are (strongly) correlated for the given group of people.

# Correlation Analysis (Numeric Data)

---

- Correlation coefficient (also called Pearson's product moment coefficient)

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

where  $n$  is the number of tuples,  $\bar{A}$  and  $\bar{B}$  are the respective means of A and B,  $\sigma_A$  and  $\sigma_B$  are the respective standard deviation of A and B, and  $\Sigma(a_i b_i)$  is the sum of the AB cross-product.

- If  $r_{A,B} > 0$ , A and B are positively correlated (A's values increase as B's). The higher, the stronger correlation.
- If  $r_{A,B} = 0$ , independent.
- If  $r_{A,B} < 0$ : negatively correlated (A's values increase as B's values decrease)

# Correlation Coefficient for Numeric Data

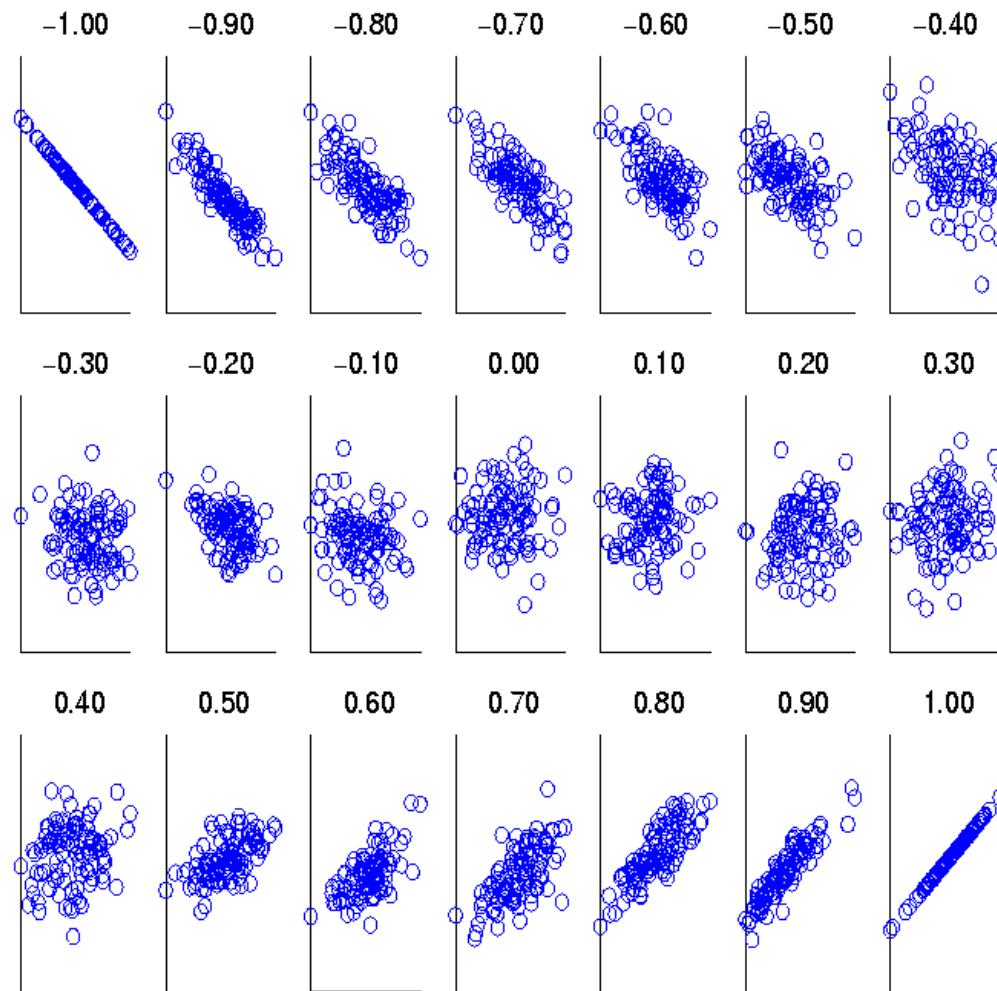
---

- Scatter plots can also be used to view correlations between attributes.
- Note that correlation does not imply causality.
- For example, in analyzing a demographic database, we may find that attributes representing the number of hospitals and the number of car thefts in a region are correlated.
- This does not mean that one causes the other.
- Both are actually causally linked to a third attribute, namely, population.

# Visually Evaluating Correlation

- Scatter plots can also be used to view correlations between attributes.

**Scatter plots  
showing the  
similarity from  
-1 to 1.**



# Correlation (viewed as linear relationship)

- Correlation measures the linear relationship between objects
- To compute correlation, we standardize data objects, A and B, and then take their dot product

$$a'_k = (a_k - \text{mean}(A)) / \text{std}(A)$$

$$b'_k = (b_k - \text{mean}(B)) / \text{std}(B)$$

$$\text{correlation}(A, B) = A' \bullet B'$$

# Covariance (Numeric Data)

- Covariance is similar to correlation

$$Cov(A, B) = E((A - \bar{A})(B - \bar{B})) = \frac{\sum_{i=1}^n (a_i - \bar{A})(b_i - \bar{B})}{n}$$

Correlation coefficient:  $r_{A,B} = \frac{Cov(A, B)}{\sigma_A \sigma_B}$

where n is the number of tuples,  $\bar{A}$  and  $\bar{B}$  are the respective mean or **expected values** of A and B,  $\sigma_A$  and  $\sigma_B$  are the respective standard deviation of A and B.

- **Positive covariance:** If  $Cov_{A,B} > 0$ , then A and B both tend to be larger than their expected values.
- **Negative covariance:** If  $Cov_{A,B} < 0$  then if A is larger than its expected value, B is likely to be smaller than its expected value.
- **Independence:**  $Cov_{A,B} = 0$  but the converse is not true:
  - Some pairs of random variables may have a covariance of 0 but are not independent. Only under some additional assumptions (e.g., the data follow multivariate normal distributions) does a covariance of 0 imply independence

# Co-Variance: An Example

$$Cov(A, B) = E((A - \bar{A})(B - \bar{B})) = \frac{\sum_{i=1}^n (a_i - \bar{A})(b_i - \bar{B})}{n}$$

- It can be simplified in computation as

$$Cov(A, B) = E(A \cdot B) - \bar{A}\bar{B}$$

- Suppose two stocks A and B have the following values in one week:  
(2, 5), (3, 8), (5, 10), (4, 11), (6, 14).
- Question: If the stocks are affected by the same industry trends, will their prices rise or fall together?
  - $E(A) = (2 + 3 + 5 + 4 + 6)/ 5 = 20/5 = 4$
  - $E(B) = (5 + 8 + 10 + 11 + 14) /5 = 48/5 = 9.6$
  - $Cov(A,B) = (2 \times 5 + 3 \times 8 + 5 \times 10 + 4 \times 11 + 6 \times 14) /5 - 4 \times 9.6 = 4$
- Thus, A and B rise together since  $Cov(A, B) > 0$ .

# Tuple Duplication

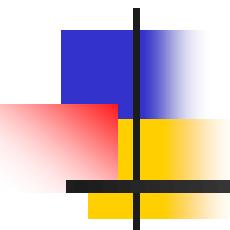
---

- In addition to detecting redundancies between attributes, duplication should also be detected at the tuple level (e.g., where there are two or more identical tuples for a given unique data entry case).
- The use of denormalized tables (often done to improve performance by avoiding joins) is another source of data redundancy.
- Inconsistencies often arise between various duplicates, due to inaccurate data entry or updating some but not all data occurrences.
  - e.g., If a purchase order database contains attributes for the purchaser's name and address instead of a key to this information in a purchaser database, discrepancies can occur, such as the same purchaser's name appearing with different addresses within the purchase order database.

# Data Value Conflict Detection and Resolution

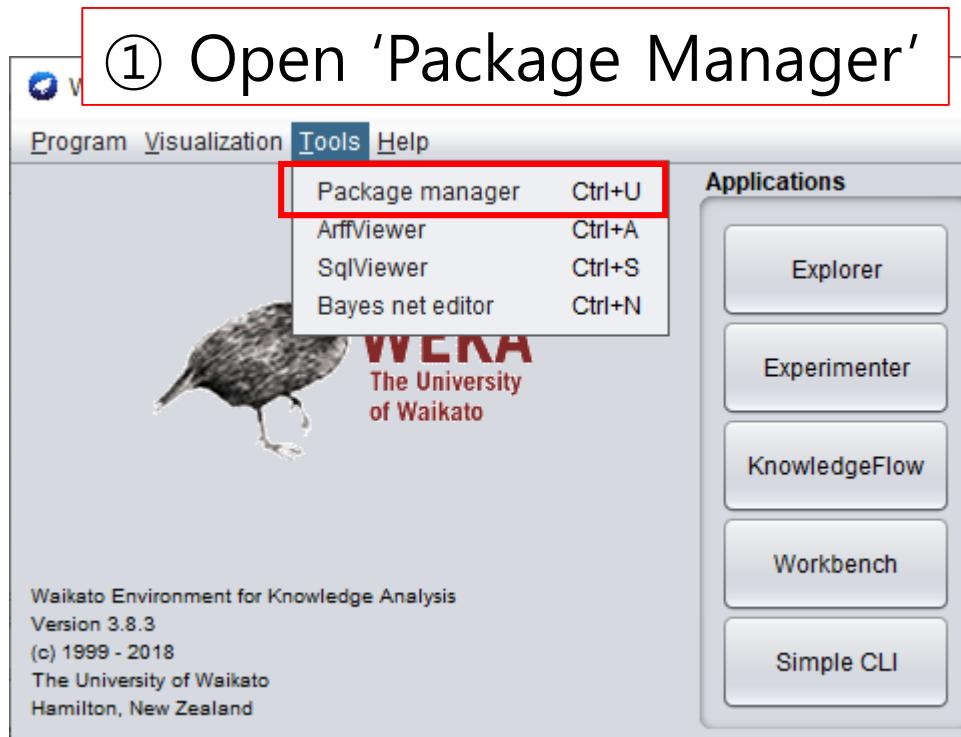
---

- Data integration also involves the detection and resolution of data value conflicts.
  - For the same real-world entity, attribute values from different sources may differ.
  - When exchanging information between schools, for example, each school may have its own curriculum and grading scheme.
    - One university may adopt a quarter system, offer three courses on database systems, and assign grades from AC to F, whereas another may adopt a semester system, offer two courses on databases, and assign grades from 1 to 10.
    - It is difficult to work out precise course-to-grade transformation rules between the two universities, making information exchange difficult.
- Attributes may also differ on the abstraction level, where an attribute in one system is recorded at, say, a lower abstraction level than the “same” attribute in another.
  - The total sales in one database may refer to one branch of All Electronics, while an attribute of the same name in another database may refer to the total sales for All Electronics stores in a given region.

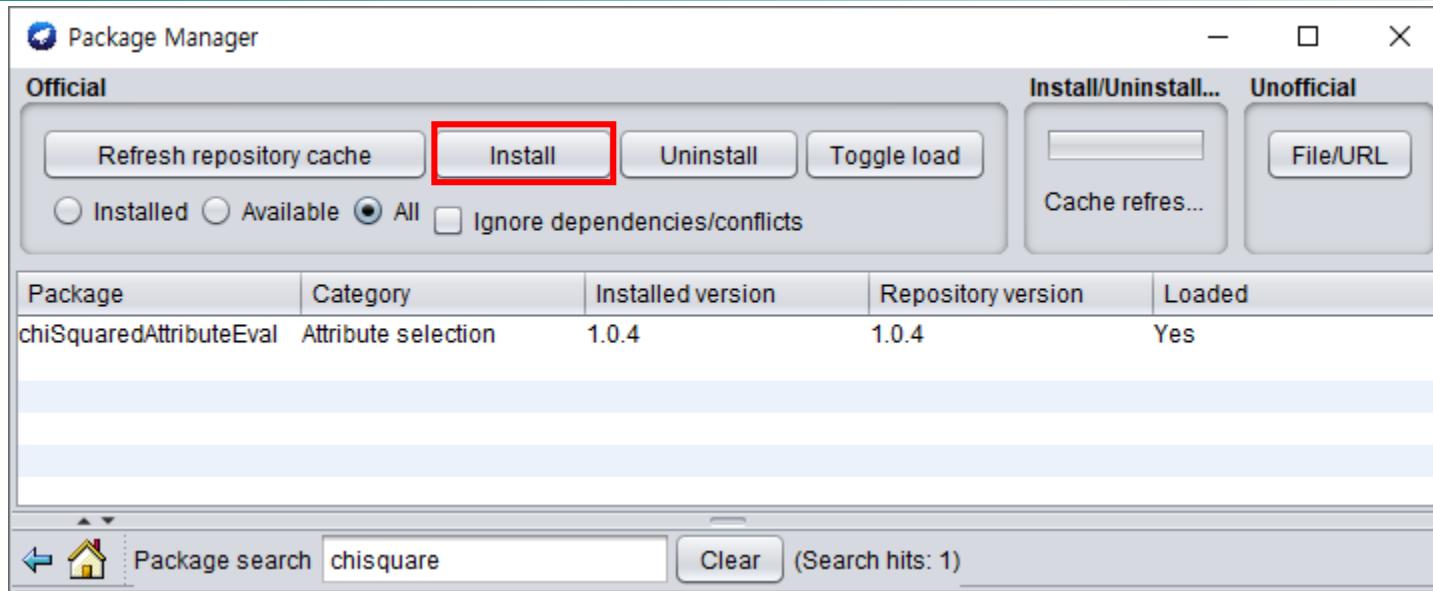


# **Weka - Attribute Selection**

# Attribute Selection (Chi-squared)



# Attribute Selection (Chi-squared)



② Search and install 'chiSquaredAttributeEval' package

URL: <http://weka.sourceforge.net/doc.packages/chiSquaredAttributeEval>

Author: Eibe Frank

Maintainer: Weka team <wekalist{[at]}list.scms.waikato.ac.nz>

Evaluates the worth of an attribute by computing the value of the chi-squared statistic with respect to the class.

All available versions:

# Attribute Selection (Chi-squared)

③ Open 'weather.nominal.arff' file

The Weka Explorer interface showing the 'Select attributes' tab selected. The 'Selected attribute' panel displays the 'outlook' attribute as nominal, with values sunny, overcast, and rainy. The 'Attributes' panel shows the 'outlook' attribute selected. Three bar charts at the bottom represent the distribution of the 'play' class for each outlook value.

**Current relation**  
Relation: weather.symbolic  
Instances: 14  
Attributes: 5  
Sum of weights: 14

**Selected attribute**  
Name: outlook  
Missing: 0 (0%)  
Distinct: 3  
Type: Nominal  
Unique: 0 (0%)

No.	Label	Count	Weight
1	sunny	5	5.0
2	overcast	4	4.0
3	rainy	5	5.0

**Attributes**  
All    None    Invert    Pattern

No.	Name
1	<input checked="" type="checkbox"/> outlook
2	<input type="checkbox"/> temperature
3	<input type="checkbox"/> humidity
4	<input type="checkbox"/> windy
5	<input type="checkbox"/> play

Remove

**Status**  
OK

Class: play (Nom)    Visualize All

Log x 0

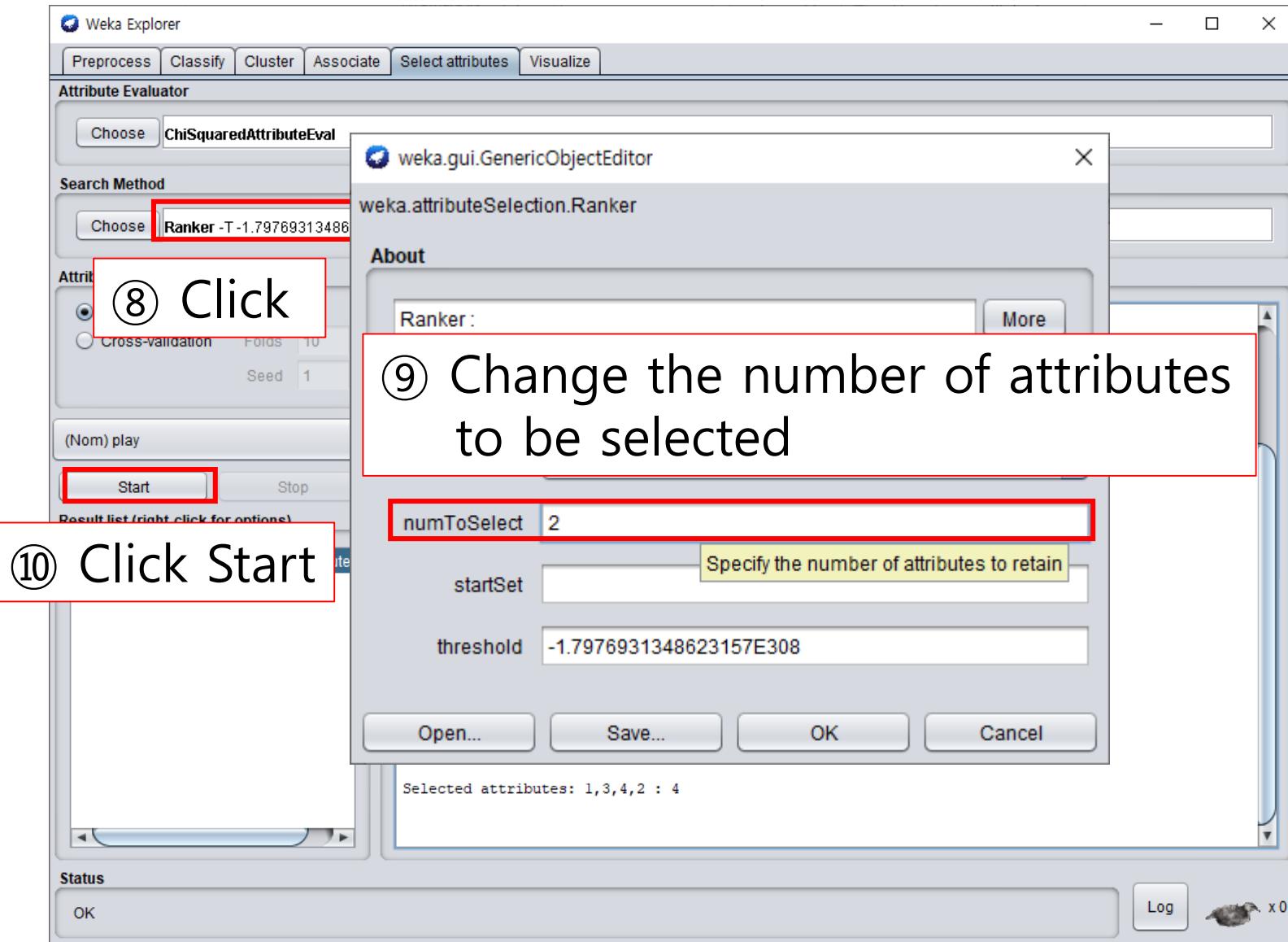
# Attribute Selection (Chi-squared)

The screenshot shows the Weka Explorer interface with the following steps highlighted:

- ③ Choose 'Select attributes' tab
- ④ Choose -> ChiSquaredAttributeEval
- ⑤ Choose -> Ranker
- ⑥ Click Start
- ⑦ The chi-square test result is shown in a sorted order

Detailed description: The screenshot displays the Weka Explorer window. At the top, there's a toolbar with tabs: Preprocess, Classify, Cluster, Associate, Select attributes (which is highlighted with a red box), and Visualize. Below the toolbar is the 'Attribute Evaluator' panel. It contains a 'Search Method' section with a 'Choose' button (highlighted with a red box) followed by 'ChiSquaredAttributeEval'. Underneath is another 'Choose' button (highlighted with a red box) followed by 'Ranker -T-1.7976931348623157E308 -N-1'. The 'Attribute Selection Mode' section has two radio buttons: 'Use full training set' (selected, highlighted with a red box) and 'Cross-validation'. The 'Folds' input field is set to 10 and the 'Seed' input field is set to 1. To the right, there's a list of attributes: humidity, windy, play. Below this is the text 'Evaluation mode: evaluate on all training data'. In the bottom left, there's a 'Result list' pane showing a single entry: '16:02:34 - Ranker + ChiSquaredAttributeEval'. The main output pane shows the following text:  
Search Method:  
Attribute ranking.  
  
Attribute Evaluator (supervised, Class (nominal)+ 5 play).  
Chi-squared Ranking Filter  
  
Ranked attributes:  
3.547 1 outlook  
2.8 3 humidity  
0.933 4 windy  
0.57 2 temperature  
  
Selected attributes: 1,3,4,2 : 4

# Attribute Selection (Chi-squared)



# Attribute Selection (Chi-squared)

The screenshot shows the Weka Explorer interface with the following configuration:

- Attribute Evaluator:** ChiSquaredAttributeEval
- Search Method:** Ranker -T 1.7976931348623157E308 -N 2
- Attribute Selection Mode:** Use full training set (selected), Folds 10, Seed 1.
- Result list:** Shows log entries: 16:02:34 - Ranker + ChiSquaredAttributeEval and 16:07:45 - Ranker + ChiSquaredAttributeEval.
- Attribute selection output:**

```
Instances: 14
Attributes: 5
outlook
temperature
humidity
windy
play
Evaluation mode: evaluate on all training data

===
Att
Search
Attribu
Chi-squared Ranking Filter

Ranked attributes:
3.547 1 outlook
2.8    3 humidity

Selected attributes: 1,3 : 2
```
- Status:** OK

A red box highlights the output text, and a red circle with the number 11 is placed over the word "Ranked" in the output text.

⑪ 2 attributes with high correlation to the label are selected

# Attribute Selection (Pearson)

① Open 'weather.numeric.arff'

The screenshot shows the Weka Explorer interface with a red box highlighting the title bar and the 'Select attributes' tab.

**Weka Explorer**

Preprocess Classify Cluster Associate Select attributes Visualize Undo Edit... Save...

**Current relation**

Relation: weather  
Instances: 14

Attributes: 5  
Sum of weights: 14

**Attributes**

All None Invert Pattern

No.	Name
1	<input checked="" type="checkbox"/> outlook
2	<input type="checkbox"/> temperature
3	<input type="checkbox"/> humidity
4	<input type="checkbox"/> windy
5	<input type="checkbox"/> play

**Selected attribute**

Name: outlook  
Missing: 0 (0%)  
Distinct: 3  
Type: Nominal  
Unique: 0 (0%)

No.	Label	Count	Weight
1	sunny	5	5.0
2	overcast	4	4.0
3	rainy	5	5.0

Class: play (Nom) Visualize All

5 4 5

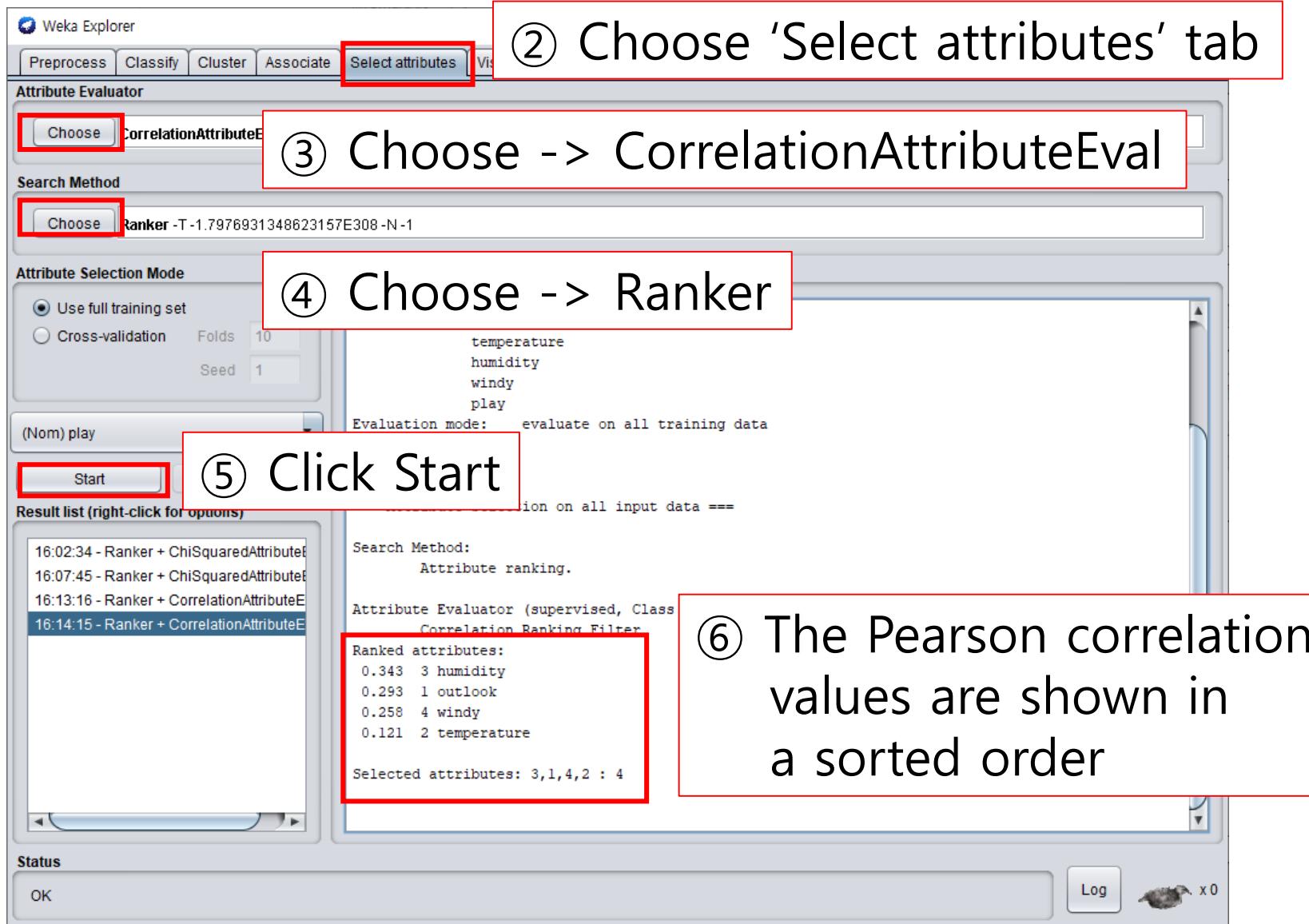
Remove

Status

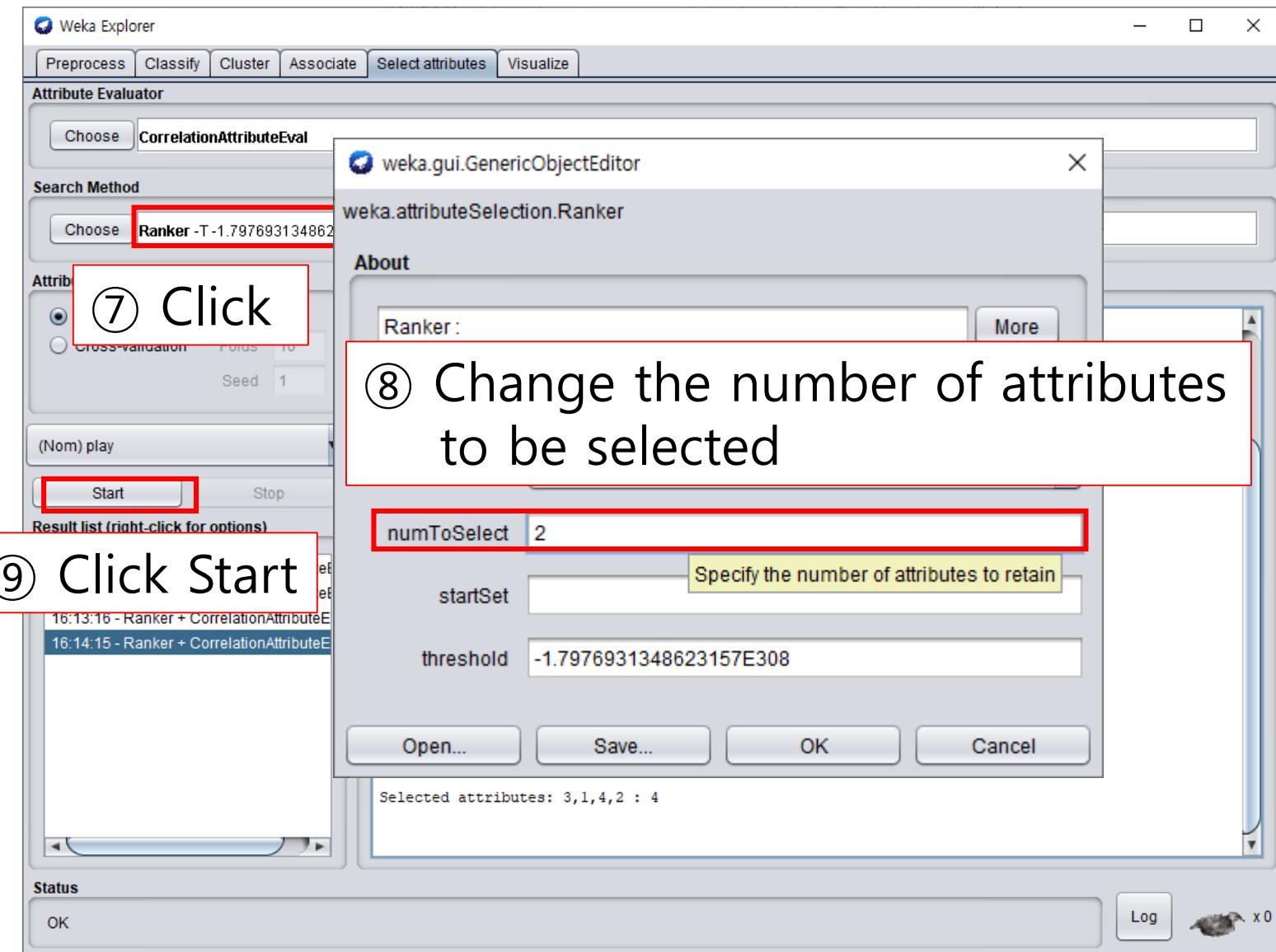
OK Log x 0

Detailed description: The screenshot captures the Weka Explorer interface for attribute selection. A red box highlights the title bar and the 'Select attributes' tab. The 'Selected attribute' panel shows 'outlook' as the chosen attribute, which is nominal with three distinct values: sunny, overcast, and rainy. Each value has a count of 5 and a weight of 5.0. Below this, there are three bar charts representing the distribution of the 'play' class for each outlook value. The first bar (sunny) is red at the top and blue at the bottom, with a total height of 5. The second bar (overcast) is entirely blue, with a height of 4. The third bar (rainy) is red at the top and blue at the bottom, with a total height of 5. The 'Attributes' panel lists all five attributes: outlook, temperature, humidity, windy, and play, with 'outlook' selected. The 'Status' panel at the bottom indicates 'OK'.

# Attribute Selection (Pearson)



# Attribute Selection (Pearson)



# Attribute Selection (Pearson)

The screenshot shows the Weka Explorer interface with the following configuration:

- Attribute Evaluator:** CorrelationAttributeEval
- Search Method:** Ranker -T 1.7976931348623157E308 -N 2
- Attribute Selection Mode:** Use full training set (selected), Folds 10, Seed 1.
- Class:** play (Nominal)
- Result list:** Shows log entries for different attribute selection runs.
- Attribute selection output (Console):**

```
RELATION: weather
Instances: 14
Attributes: 5
outlook
temperature
humidity
windy
play
Evaluation mode: evaluate on all training data

==== Attribute
Search Method
Attrib
```

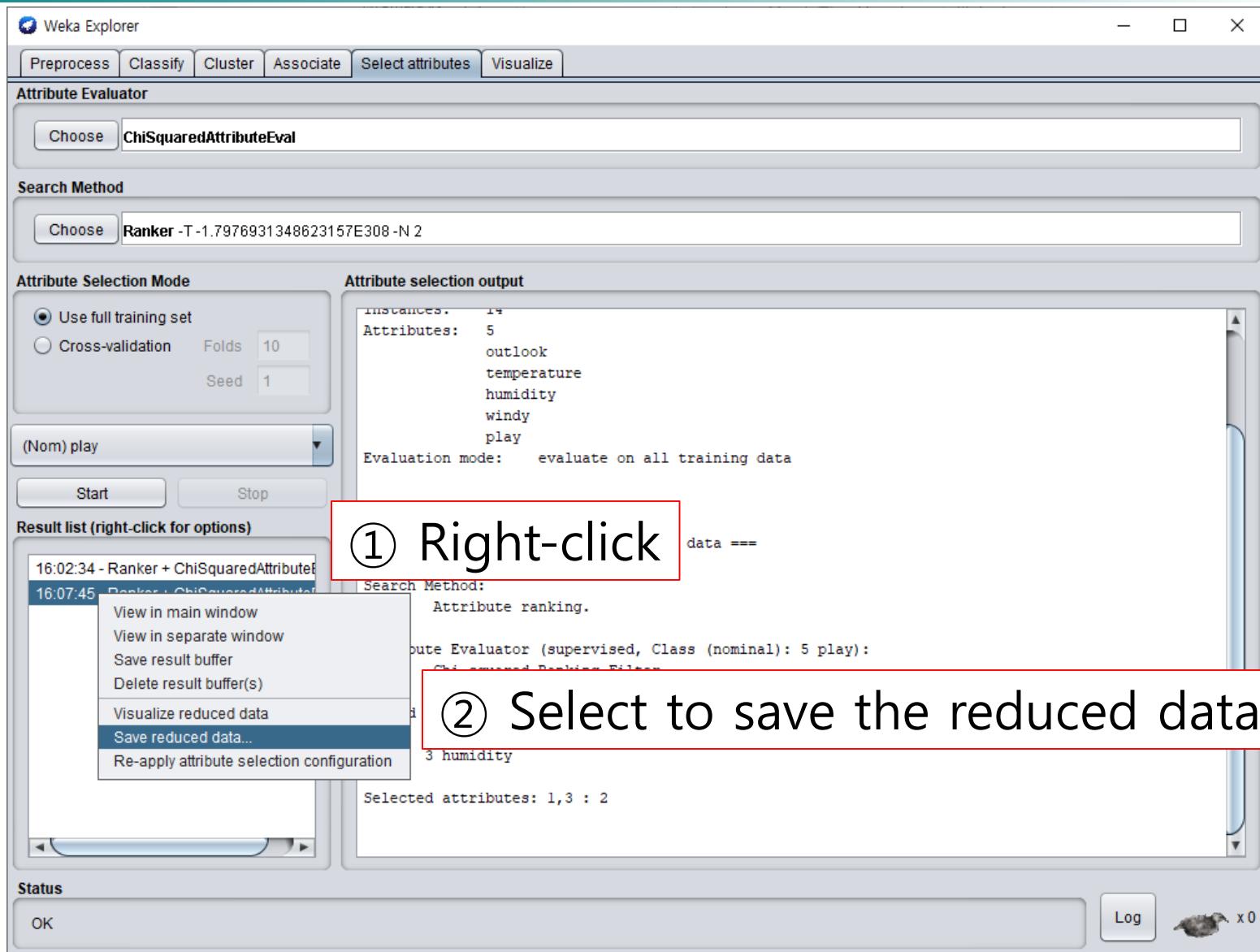
**⑨ 2 attributes with high correlation to the label are selected**

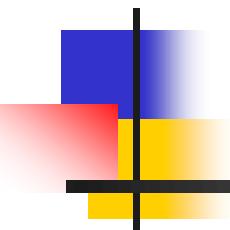
```
Attribute Evaluator (supervised, Class (nominal): 5 play):
Correlation Ranking Filter
Ranked attributes:
0.343 3 humidity
0.293 1 outlook

Selected attributes: 3,1 : 2
```

**Status:** OK

# Save Reduced Data





# Python - Attribute Selection

# Chi-Square Test

```
import pandas as pd
from scipy.stats import chi2_contingency

df = pd.read_csv('weather.nominal.csv')
df[:5]
```

	outlook	temperature	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes

# Chi-Square Test

```
attrs = df.columns           # get the names of the column attributes  
 attrs  
  
Index(['outlook', 'temperature', 'humidity', 'windy', 'play'], dtype='object')
```

```
# Generate the contingency table between 'outlook' and 'play'  
contingency = pd.crosstab(df[attrs[0]], df[attrs[-1]])  
contingency
```

play no yes

outlook

	no	yes
overcast	0	4
rainy	2	3
sunny	3	2

There are 4 records with  
outlook = 'overcast' and  
play = 'yes'

# Chi-Square Test

```
# Perform chi-square test
chi2, p, dof, exp = chi2_contingency(contingency)

print(chi2)      # the chi-square value
print(p)         # the probability of obtaining test results
                  # assuming that the null hypothesis is correct.
print(dof)       # the number of independent ways by which a dynamic
                  # system can move, without violating any constraint
                  # imposed on it
print(exp)       # The expected frequencies
```

(3.5466666666666664, ----- chi2 : chi-square value  
0.16976615743981122, ----- p : p-value  
2, ----- dof : degree of freedom  
[[1.42857143, 2.57142857],  
 [1.78571429, 3.21428571],  
 [1.78571429, 3.21428571]]

# Pearson Correlation

```
# library import
import pandas as pd
from scipy.stats import pearsonr

df = pd.read_csv('glass.csv')
df[:5]
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.51793	12.79	3.50	1.12	73.03	0.64	8.77	0.0	0.00	'build wind float'
1	1.51643	12.16	3.52	1.35	72.89	0.57	8.53	0.0	0.00	'vehic wind float'
2	1.51793	13.21	3.48	1.41	72.64	0.59	8.43	0.0	0.00	'build wind float'
3	1.51299	14.40	1.74	1.54	74.55	0.00	7.59	0.0	0.00	tableware
4	1.53393	12.30	0.00	1.00	70.16	0.12	16.19	0.0	0.24	'build wind non-float'

# Pearson Correlation

```
# get attribute names  
attrs = df.columns  
attrs
```

```
Index(['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe', 'Type'], dtype='object')
```

```
# compute Pearson correlation between 'RI' and 'Na'  
# remember that Pearson correlation is computed for numerics  
pearsonr(df[attrs[0]], df[attrs[1]])
```

(-0.19188537903890265, 0.004850112327446243)

Pearson correlation

p-value

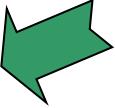
# Practice

---

- Compute chi-square values for all pairs of attributes in 'weather.nominal.csv'
- Compute Pearson correlation for all pairs of numeric attributes in 'glass.csv'

# Chapter 3: Data Preprocessing

---

- Data Preprocessing: An Overview
  - Data Quality
  - Major Tasks in Data Preprocessing
- Data Cleaning
- Data Integration
- Data Reduction 
- Data Transformation and Data Discretization
- Summary

# Data Reduction Strategies

---

- **Data reduction:** Obtain a reduced representation of the data set that is much smaller in volume but yet produces the same (or almost the same) analytical results
- Why data reduction? — A database/data warehouse may store terabytes of data. Complex data analysis may take a very long time to run on the complete data set.
- Data reduction strategies
  - Dimensionality reduction
  - Numerosity reduction (some simply call it: Data Reduction)
  - Data compression

# Data Reduction Strategies

---

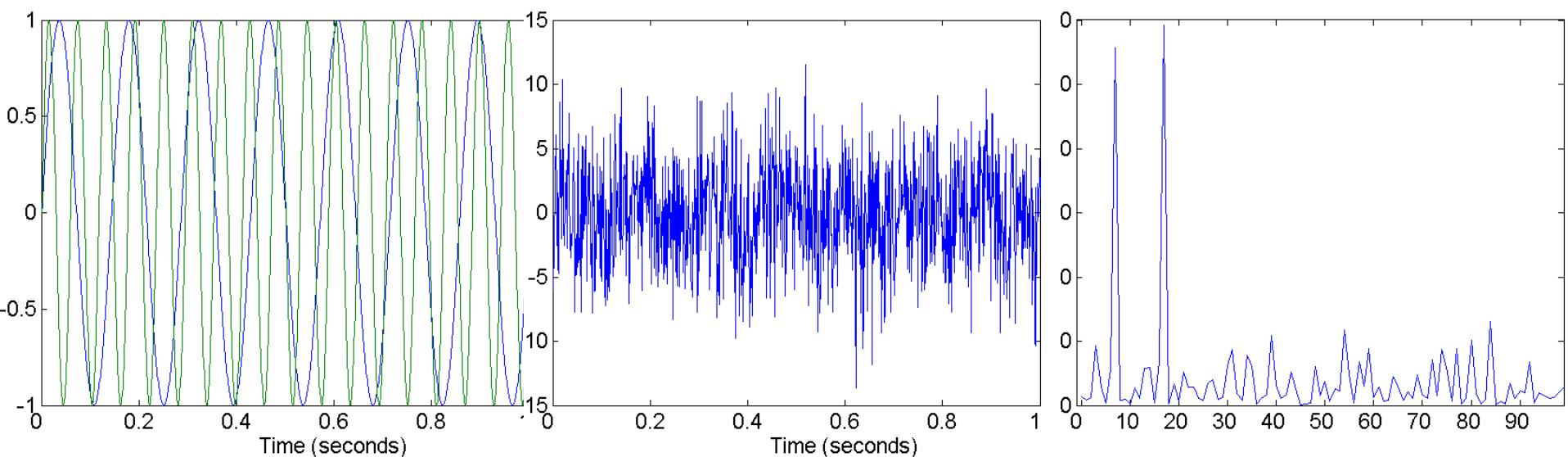
- **Dimensionality reduction** - reducing the number of random variables or attributes under consideration.
  - Wavelet transforms
  - Principal components analysis - transform or project the original data onto a smaller space
  - Attribute subset selection – detect and remove irrelevant, weakly relevant, or redundant attributes or dimensions
- **Numerosity reduction** - replace the original data volume by alternative, smaller forms of data representation.
  - parametric methods - a model is used so that only the data parameters need to be stored, instead of the actual data (e.g., Regression model).
  - Nonparametric methods - store reduced representations of the data (e.g., histograms, clustering, sampling, and data cube aggregation).
- **Data compression** - Obtain a compressed representation of the original data
  - Lossless - the original data is reconstructed without any information loss
  - Lossy

# Data Reduction 1: Dimensionality Reduction

- **Curse of dimensionality**
  - When dimensionality increases, data becomes increasingly sparse
  - Density and distance between points, which is critical to clustering, outlier analysis, becomes less meaningful
  - The possible combinations of subspaces will grow exponentially
- **Dimensionality reduction**
  - Avoid the curse of dimensionality
  - Help eliminate irrelevant features and reduce noise
  - Reduce time and space required in data mining
  - Allow easier visualization
- **Dimensionality reduction techniques**
  - Wavelet transforms
  - Principal Component Analysis
  - Supervised and nonlinear techniques (e.g., feature selection)

# Mapping Data to a New Space

- Fourier transform
- Wavelet transform



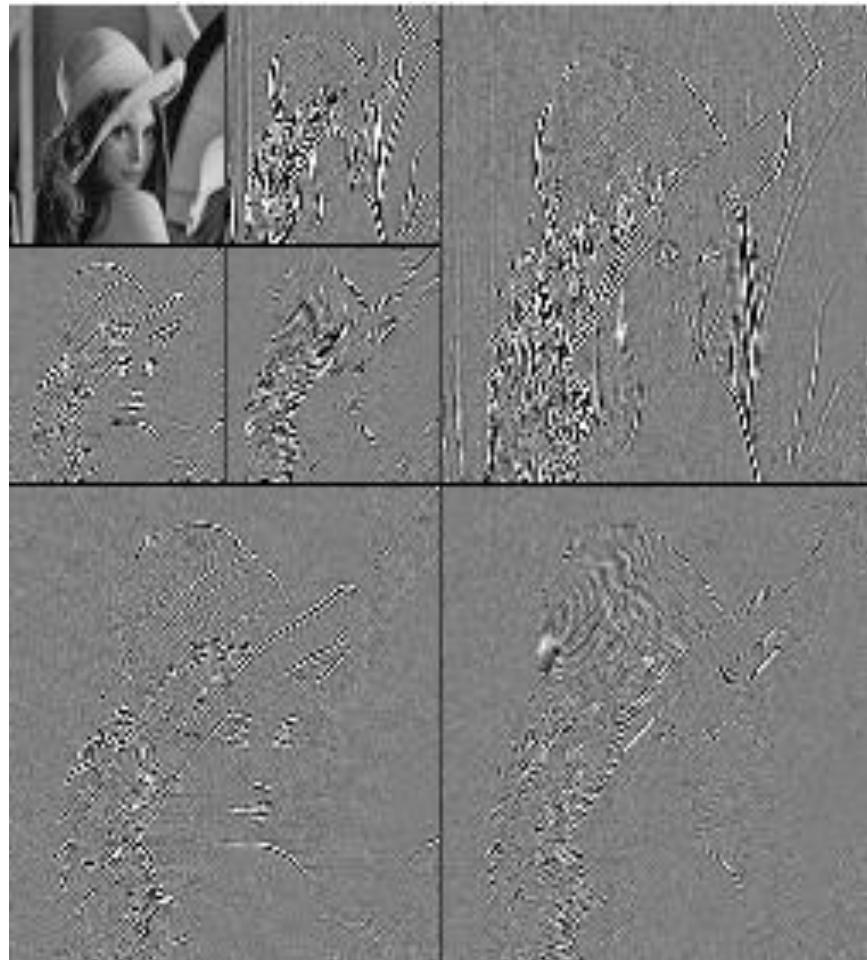
Two Sine Waves

Two Sine Waves + Noise

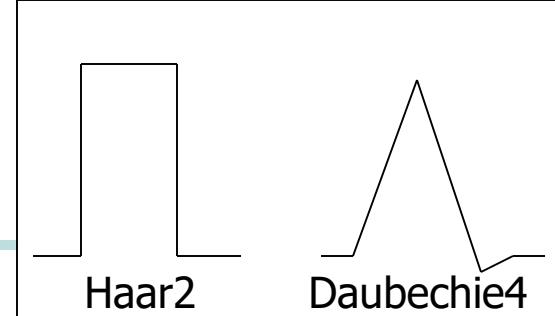
Frequency

# What Is Wavelet Transform?

- Decomposes a signal into different frequency subbands
  - Applicable to n-dimensional signals
- Data are transformed to preserve relative distance between objects at different levels of resolution
- Allow natural clusters to become more distinguishable
- Used for image compression



# Wavelet Transformation



- Discrete wavelet transform (DWT) for linear signal processing, multi-resolution analysis
- Compressed approximation: store only a small fraction of the strongest of the wavelet coefficients
- Similar to discrete Fourier transform (DFT), but better lossy compression, localized in space
- Method:
  - Length,  $L$ , must be an integer power of 2 (padding with 0's, when necessary)
  - Each transform has 2 functions: smoothing, difference
  - Applies to pairs of data, resulting in two set of data of length  $L/2$
  - Applies two functions recursively, until reaches the desired length

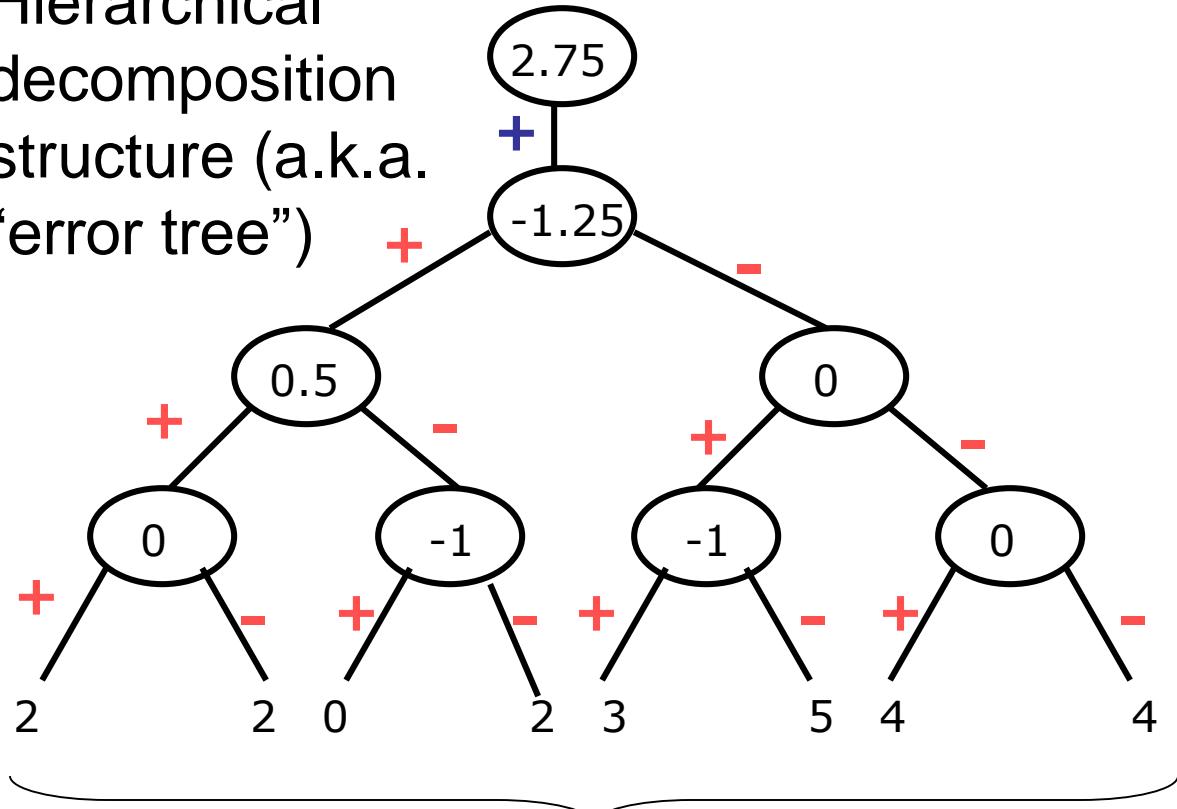
# Wavelet Decomposition

- Wavelets: A math tool for space-efficient hierarchical decomposition of functions
- $S = [2, 2, 0, 2, 3, 5, 4, 4]$  can be transformed to  $S_\wedge = [2^3/4, -1^1/4, 1/2, 0, 0, -1, -1, 0]$
- Compression: many small detail coefficients can be replaced by 0's, and only the significant coefficients are retained

Resolution	Averages	Detail Coefficients
8	$[2, 2, 0, 2, 3, 5, 4, 4]$	
4	$[2, 1, 4, 4]$	$[0, -1, -1, 0]$
2	$[1\frac{1}{2}, 4]$	$[\frac{1}{2}, 0]$
1	$[2\frac{3}{4}]$	$[-1\frac{1}{4}]$

# Haar Wavelet Coefficients

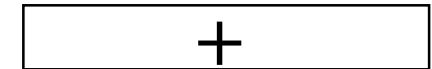
Hierarchical decomposition structure (a.k.a.  
“error tree”)



Original frequency distribution

Coefficient “Supports”

2.75



-1.25



0.5



0



0



-1



-1



0



# Why Wavelet Transform?

---

- Use hat-shape filters
  - Emphasize region where points cluster
  - Suppress weaker information in their boundaries
- Effective removal of outliers
  - Insensitive to noise, insensitive to input order
- Multi-resolution
  - Detect arbitrary shaped clusters at different scales
- Efficient
  - Complexity  $O(N)$
- Only applicable to low dimensional data

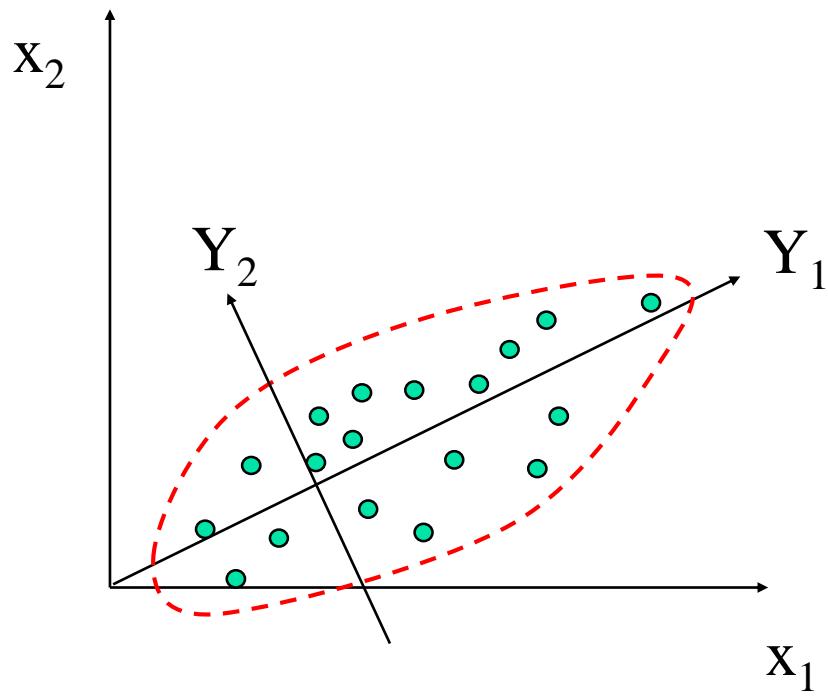
# Wavelet Transformation

---

- Wavelet transforms can be applied to multi-dimensional data such as a data cube.
- This is done by first applying the transform to the first dimension, then to the second, and so on.
- The computational complexity involved is linear with respect to the number of cells in the cube.
- Wavelet transforms give good results on sparse or skewed data and on data with ordered attributes.
- Lossy compression by wavelets is reportedly better than JPEG compression, the current commercial standard.
- Wavelet transforms have many real world applications, including the compression of fingerprint images, computer vision, analysis of time-series data, and data cleaning.

# Principal Component Analysis (PCA)

- Find a projection that captures the largest amount of variation in data
- The original data are projected onto a much smaller space, resulting in dimensionality reduction.
- We find the eigenvectors of the covariance matrix, and these eigenvectors define the new space
- $Y_1$  and  $Y_2$  are the first two principal components for the given data.



# Principal Component Analysis (PCA)

---

- Unlike attribute subset selection, which reduces the attribute set size by retaining a subset of the initial set of attributes, PCA “combines” the essence of attributes by creating an alternative, smaller set of variables.
- The initial data is projected onto this smaller set.
- PCA often reveals relationships that were not previously suspected and thereby allows interpretations that would not ordinarily result.
- Multidimensional data of more than two dimensions can be handled by reducing the problem to two dimensions.
- Principal components may be used as inputs to multiple regression and cluster analysis.
- In comparison with wavelet transforms, PCA tends to be better at handling sparse data, whereas wavelet transforms are more suitable for data of high dimensionality.

# Principal Component Analysis (Steps)

---

- Given  $N$  data vectors from  $n$ -dimensions, find  $k \leq n$  orthogonal vectors (*principal components*) that can be best used to represent data
  - Normalize input data: Each attribute falls within the same range
  - Compute  $k$  orthonormal (unit) vectors, i.e., *principal components*
  - Each input data (vector) is a linear combination of the  $k$  principal component vectors
  - The principal components are sorted in order of decreasing “significance” or strength
  - Since the components are sorted, the size of the data can be reduced by eliminating the *weak components*, i.e., those with low variance (i.e., using the strongest principal components, it is possible to reconstruct a good approximation of the original data)
- Works for numeric data only

# Attribute Subset Selection

---

- Data sets for analysis may contain hundreds of attributes, many of which may be irrelevant to the mining task or redundant.
  - e.g., if the task is to classify customers based on whether or not they are likely to purchase a popular new CD at AllElectronics when notified of a sale, attributes such as the customer's telephone number are likely to be irrelevant, unlike attributes such as age or music taste.
- Although it may be possible for a domain expert to pick out some of the useful attributes, this can be a difficult and time-consuming task, especially when the data's behavior is not well known. (Hence, a reason behind its analysis!)
- Leaving out relevant attributes or keeping irrelevant attributes may be detrimental, causing confusion for the mining algorithm employed.
- This can result in discovered patterns of poor quality.
- In addition, the added volume of irrelevant or redundant attributes can slow down the mining process.

# Attribute Subset Selection

---

- Another way to reduce dimensionality of data
- Redundant attributes
  - Duplicate much or all of the information contained in one or more other attributes
  - E.g., purchase price of a product and the amount of sales tax paid
- Irrelevant attributes
  - Contain no information that is useful for the data mining task at hand
  - E.g., students' ID is often irrelevant to the task of predicting students' GPA

# **Attribute subset selection**

---

- Reduces the data set size by removing irrelevant or redundant attributes (or dimensions).
- The goal is to find a minimum set of attributes such that the resulting probability distribution of the data classes is as close as possible to the original distribution obtained using all attributes.
- Mining on a reduced set of attributes has an additional benefit - reduces the number of attributes appearing in the discovered patterns, helping to make the patterns easier to understand.

# How to find a ‘good’ subset of attributes

---

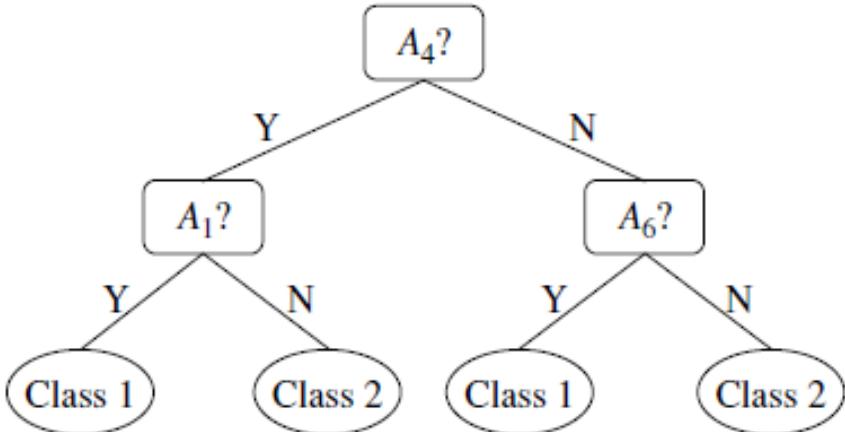
- For  $n$  attributes, there are  $2^n$  possible subsets.
- An exhaustive search for the optimal subset of attributes can be prohibitively expensive, especially as  $n$  and the number of data classes increase.
- Thus, greedy heuristic methods that explore a reduced search space are commonly used.
  - While searching through attribute space, they always make what looks to be the best choice at the time.
- Such greedy methods are effective in practice and may come close to estimating an optimal solution.
- The “best” (and “worst”) attributes are typically determined using tests of statistical significance, which assume that the attributes are independent of one another.
- Many other attribute evaluation measures can be used such as the information gain measure used in building decision trees for classification.

# Greedy (heuristic) methods for attribute subset selection

---

- Stepwise forward selection:
  - Starts with an empty set of attributes as the reduced set.
  - At each step, it adds the best remaining original attributes to the set.
- Stepwise backward elimination:
  - Starts with the full set of attributes.
  - At each step, it removes the worst attribute remaining in the set.
- Combination of forward selection and backward elimination:
  - At each step, it selects the best attribute and removes the worst from among the remaining attributes.
- Decision tree induction:
  - At each node, the algorithm chooses the “best” attribute to partition the data into individual classes.
  - When decision tree induction is used for attribute subset selection, a tree is constructed from the given data.
  - All attributes that do not appear in the tree are assumed to be irrelevant.
  - The set of attributes appearing in the tree form the reduced subset of attributes.

# Greedy (heuristic) methods for attribute subset selection

Forward selection	Backward elimination	Decision tree induction
Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$
Initial reduced set: $\{\}$ $\Rightarrow \{A_1\}$ $\Rightarrow \{A_1, A_4\}$ $\Rightarrow$ Reduced attribute set: $\{A_1, A_4, A_6\}$	$\Rightarrow \{A_1, A_3, A_4, A_5, A_6\}$ $\Rightarrow \{A_1, A_4, A_5, A_6\}$ $\Rightarrow$ Reduced attribute set: $\{A_1, A_4, A_6\}$	 <pre>graph TD; Root[A4?] -- Y --&gt; A1[A1?]; Root -- N --&gt; A6[A6?]; A1 -- Y --&gt; C1_1([Class 1]); A1 -- N --&gt; C1_2([Class 2]); A6 -- Y --&gt; C2_1([Class 1]); A6 -- N --&gt; C2_2([Class 2]);</pre> $\Rightarrow$ Reduced attribute set: $\{A_1, A_4, A_6\}$

# Heuristic Search in Attribute Selection

---

- There are  $2^d$  possible attribute combinations of  $d$  attributes
- Typical heuristic attribute selection methods:
  - Best single attribute under the attribute independence assumption: choose by significance tests
  - Best step-wise feature selection:
    - The best single-attribute is picked first
    - Then next best attribute condition to the first, ...
  - Step-wise attribute elimination:
    - Repeatedly eliminate the worst attribute
  - Best combined attribute selection and elimination
  - Optimal branch and bound:
    - Use attribute elimination and backtracking

# Creating New Attributes

---

- In some cases, we may want to create new attributes based on others.
- Such attribute construction can help improve accuracy and understanding of structure in high-dimensional data.
- For example, we may wish to add the attribute area based on the attributes height and width.
- By combining attributes, attribute construction can discover missing information about the relationships between data attributes that can be useful for knowledge discovery.

# Regression Models: Parametric Data Reduction

---

- Data are modeled to fit a straight line.
- For example, a random variable,  $y$  (called a response variable), can be modeled as a linear function of another random variable,  $x$  (called a predictor variable), with the equation
  - $y = wx + b$
  - where the variance of  $y$  is assumed to be constant.
- In the context of data mining,  $x$  and  $y$  are numeric database attributes. The coefficients,  $w$  and  $b$  (called regression coefficients), specify the slope of the line and the  $y$ -intercept, respectively.
- These coefficients can be solved for by the method of least squares, which minimizes the error between the actual line separating the data and the estimate of the line.
- Multiple linear regression is an extension of (simple) linear regression, which allows a response variable,  $y$ , to be modeled as a linear function of two or more predictor variables.

# Log-Linear Models: Parametric Data Reduction

---

- Log-linear models approximate discrete multidimensional probability distributions.
- Given a set of tuples in  $n$  dimensions (e.g., described by  $n$  attributes), we can consider each tuple as a point in an  $n$ -dimensional space.
- Log-linear models can be used to estimate the probability of each point in a multidimensional space for a set of discretized attributes, based on a smaller subset of dimensional combinations.
  - This allows a higher-dimensional data space to be constructed from lower-dimensional spaces.
  - Log-linear models are therefore also useful for dimensionality reduction and data smoothing (since aggregate estimates in the lower-dimensional space are less subject to sampling variations than the estimates in the higher-dimensional space).

# Regression and Log-Linear Models: Parametric Data Reduction

---

- Regression and log-linear models can be used to approximate the given data.
- Regression and log-linear models can both be used on sparse data, although their application may be limited.
- While both methods can handle skewed data, regression does exceptionally well.
- Regression can be computationally intensive when applied to high-dimensional data, whereas log-linear models show good scalability for up to 10 or so dimensions.
- Several software packages exist to solve regression problems.
  - SAS ([www.sas.com](http://www.sas.com))
  - SPSS ([www.spss.com](http://www.spss.com))
  - S-Plus ([www.insightful.com](http://www.insightful.com)).
- Another useful resource is the book Numerical Recipes in C, by Press, Teukolsky, Vetterling, and Flannery and its associated source code.

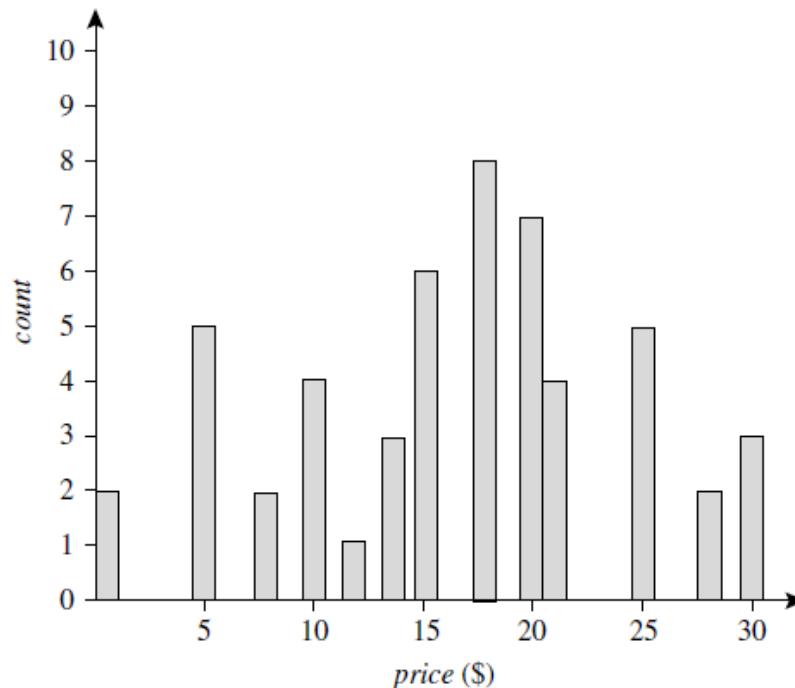
# Histograms

---

- Histograms use binning to approximate data distributions and are a popular form of data reduction.
- A histogram for an attribute,  $A$ , partitions the data distribution of  $A$  into disjoint subsets, referred to as buckets or bins.
- If each bucket represents only a single attribute–value/frequency pair, the buckets are called singleton buckets.
- Often, buckets instead represent continuous ranges for the given attribute.
- There are several partitioning rules, including the following:
  - Equal-width: the width of each bucket range is uniform.
  - Equal-frequency (or equal-depth): the buckets are created so that, roughly, the frequency of each bucket is constant.
- Histograms are highly effective at approximating both sparse and dense data, as well as highly skewed and uniform data.
- Multidimensional histograms can capture dependencies between attributes.
  - Effective in approximating data with up to five attributes.

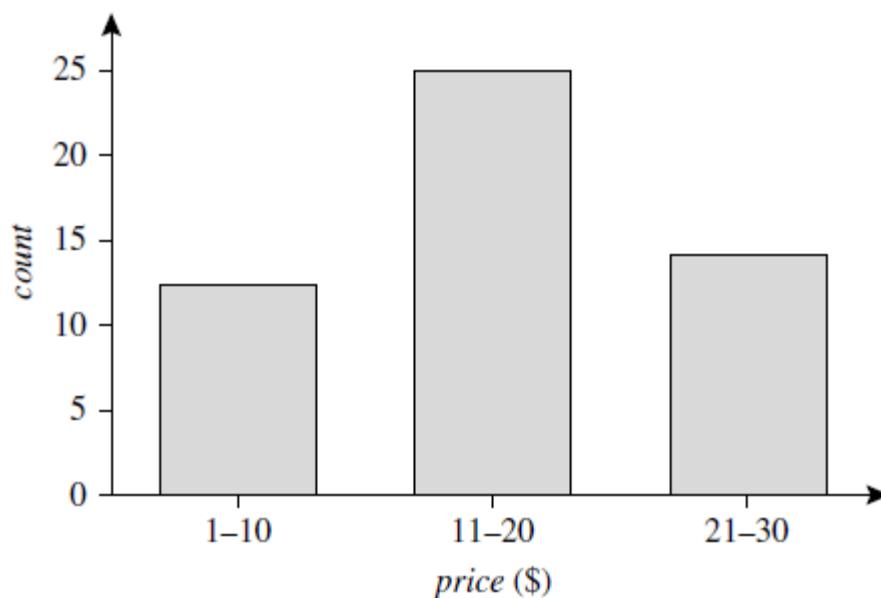
# An Example of Histograms

- The numbers have been sorted:
  - 1, 1, 5, 5, 5, 5, 5, 8, 8, 10, 10, 10, 10, 12, 14, 14, 14, 15, 15, 15, 15, 15, 18, 18, 18, 18, 18, 18, 18, 18, 18, 20, 20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 25, 25, 25, 25, 25, 28, 28, 30, 30, 30.
- A histogram for price using singleton buckets—each bucket represents one price–value/ frequency pair.



# An Example of Histograms

- The numbers have been sorted:
  - 1, 1, 5, 5, 5, 5, 5, 8, 8, 10, 10, 10, 10, 12, 14, 14, 14, 15, 15, 15, 15, 15, 18, 18, 18, 18, 18, 18, 18, 18, 18, 20, 20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 25, 25, 25, 25, 25, 28, 28, 30, 30, 30.
- An equal-width histogram for price, where values are aggregated so that each bucket has a uniform width of \$10.



# Clustering

---

- Clustering techniques consider data tuples as objects.
- They partition the objects into groups, or clusters, so that objects within a cluster are “similar” to one another and “dissimilar” to objects in other clusters.
- The “quality” of a cluster may be represented by its diameter, the maximum distance between any two objects in the cluster.
- Centroid distance is an alternative measure of cluster quality and is defined as the average distance of each cluster object from the cluster centroid (denoting the “average object,” or average point in space for the cluster).
- In data reduction, the cluster representations of the data are used to replace the actual data.
- The effectiveness of this technique depends on the data’s nature.
- It is much more effective for data that can be organized into distinct clusters than for smeared data.
- There are many measures for defining clusters and cluster quality.

# Sampling

---

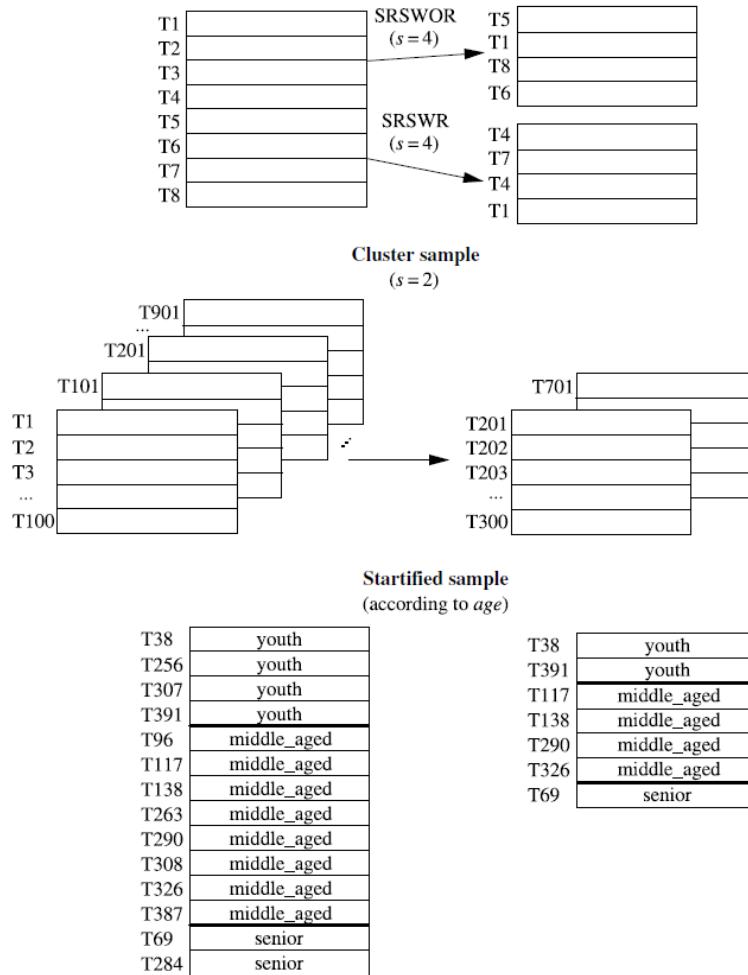
- Sampling can be used as a data reduction technique because it allows a large data set to be represented by a much smaller random data sample (or subset).
- Suppose that a large data set,  $D$ , contains  $N$  tuples.
- Simple random sample without replacement (SRSWOR) of size  $s$ :
  - This is created by drawing  $s$  of the  $N$  tuples from  $D$  ( $s < N$ ), where the probability of drawing any tuple in  $D$  is  $1/N$ .
- Simple random sample with replacement (SRSWR) of size  $s$ :
  - This is similar to SRSWOR, except that after a tuple is drawn, it is placed back in  $D$  so that it may be drawn again.
- Cluster sample: If the tuples in  $D$  are grouped into  $M$  mutually disjoint “clusters,” then an SRS of  $s$  clusters can be obtained, where  $s < M$ .
  - e.g., tuples in a database are usually retrieved a page at a time, so that each page can be considered a cluster.
  - A reduced data representation can be obtained by applying, say, SRSWOR to the pages, resulting in a cluster sample of the tuples.
  - Other clustering criteria conveying rich semantics can also be explored.
  - For example, in a spatial database, we may choose to define clusters geographically based on how closely different areas are located.

# Sampling

---

- Simple random sample without replacement (SRSWOR) of size  $s$ :
  - This is created by drawing  $s$  of the  $N$  tuples from  $D$  ( $s < N$ ), where the probability of drawing any tuple in  $D$  is  $1/N$ .
- Simple random sample with replacement (SRSWR) of size  $s$ :
  - This is similar to SRSWOR, except that after a tuple is drawn, it is placed back in  $D$  so that it may be drawn again.
- Cluster sample: If the tuples in  $D$  are grouped into  $M$  mutually disjoint clusters, an SRS of  $s$  clusters can be obtained, where  $s < M$ .
  - Tuples in a database are usually retrieved a page at a time, so that each page can be considered a cluster.
  - In a spatial database, we may choose to define clusters geographically based on how closely different areas are located.
- Stratified sample:
  - If  $D$  is divided into mutually disjoint parts called strata, a stratified sample of  $D$  is generated by obtaining an SRS at each stratum.
  - This ensure a representative sample, when the data are skewed.
  - A stratified sample may be obtained from customer data, where a stratum is created for each customer age group.

# Sampling Examples



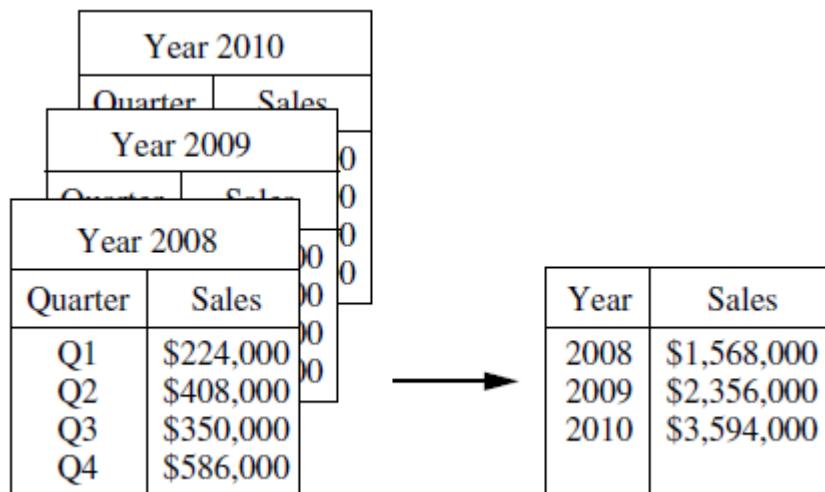
# Sampling

---

- An advantage of sampling for data reduction is that the cost of obtaining a sample is proportional to the size of the sample,  $s$ , as opposed to  $N$ , the data set size.
  - Hence, sampling complexity is sublinear to the size of the data.
- Other data reduction techniques can require at least one complete pass through  $D$ .
  - For a fixed sample size, sampling complexity increases only linearly as the number of data dimensions,  $n$ , increases, whereas techniques using histograms, for example, increase exponentially in  $n$ .
- When applied to data reduction, sampling is most commonly used to estimate the answer to an aggregate query.
- It is possible (using the central limit theorem) to determine a sufficient sample size for estimating a given function within a specified degree of error.
  - This sample size,  $s$ , may be extremely small in comparison to  $N$ .
- Sampling is a natural choice for the progressive refinement of a reduced data set.
  - Such a set can be further refined by simply increasing the sample size.

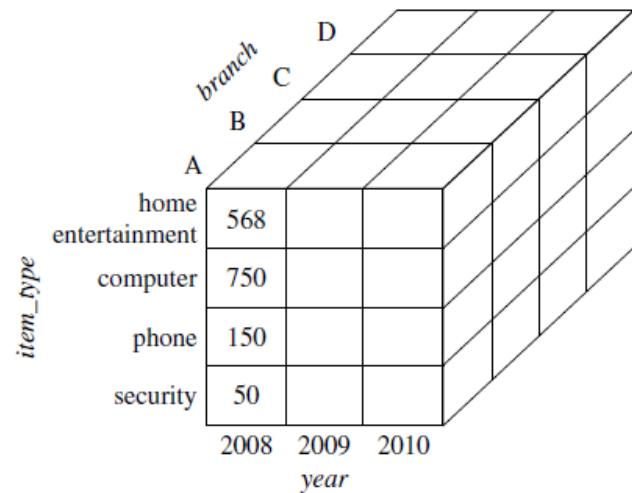
# Data Cube Aggregation

- Imagine that you have collected the data for your analysis.
- These data consist of sales per quarter, for the years 2008 to 2010.
- You are, however, interested in the annual sales (total per year), rather than the total per quarter.
- Thus, the data can be aggregated so that the resulting data summarize the total sales per year instead of per quarter.
- The resulting data set is smaller in volume, without loss of information necessary for the analysis task.



# A Data Cube Example for Sales Data

- A data cube for multidimensional analysis of sales data with respect to annual sales per item type for each branch.



- Each cell holds an aggregate data value, corresponding to the data point in multidimensional space.
- Concept hierarchies may exist for each attribute, allowing the analysis of data at multiple abstraction levels.
  - e.g., a hierarchy for branch could allow branches to be grouped into regions, based on their address.
- Data cubes provide fast access to precomputed, summarized data, thereby benefiting online analytical processing as well as data mining.

# Base Cuboid

---

- The cube created at the lowest abstraction level is the base cuboid
  - Corresponds to an individual entity of interest such as sales or customer.
- The cube at the highest level of abstraction is the apex cuboid.
  - For the sales data, the apex cuboid would give one total—the total sales for all three years, for all item types, and for all branches.
- Data cubes created for varying levels of abstraction are referred to as cuboids, so that a data cube may instead refer to a lattice of cuboids.
- Each higher abstraction level further reduces the resulting data size.
- When replying to data mining requests, the smallest available cuboid relevant to the given task should be used.

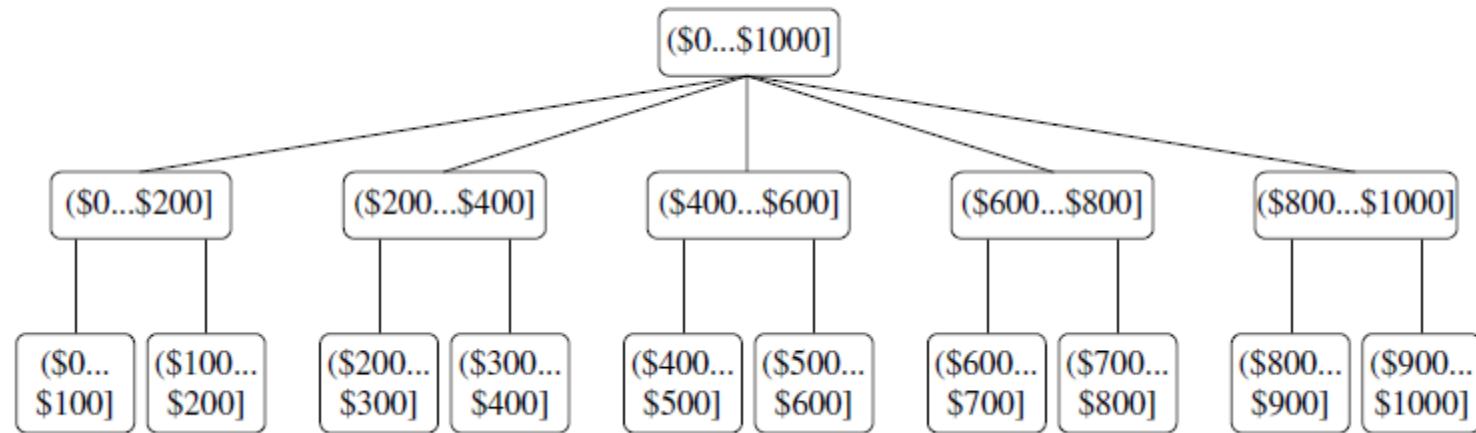
# Data Transformation

---

- In data transformation, the data are transformed or consolidated into forms appropriate for mining.
- Strategies for data transformation
  - Smoothing - works to remove noise from data (binning, regression, clustering).
  - Attribute construction (or feature construction) - new attributes are constructed and added from the given set of attributes.
  - Aggregation - summary or aggregation operations are applied to the data.
  - Normalization - the attribute data are scaled so as to fall within a smaller range, such as -1.0 to 1.0, or 0.0 to 1.0.
  - Discretization - the raw values of a numeric attribute (e.g., age) are replaced by interval labels (e.g., 0–10, 11–20, etc.) or conceptual labels (e.g., youth, adult, senior).
  - Concept hierarchy generation for nominal data - attributes such as street can be generalized to higher-level concepts, like city or country.

# A concept hierarchy for the attribute price

---



# Data Discretization

---

- If the discretization process uses class information, then we say it is supervised discretization.
- Otherwise, it is unsupervised.
- Top-down discretization or splitting - starts by first finding one or a few points (called split points or cut points) to split the entire attribute range, and then repeats this recursively on the resulting intervals.
- Bottom-up discretization or merging - starts by considering all of the continuous values as potential split-points, removes some by merging neighborhood values to form intervals, and then recursively applies this process to the resulting intervals.
- Data discretization and concept hierarchy generation are also forms of data reduction.
- The resulting patterns mined are typically easier to understand.
- Concept hierarchies are useful for mining at multiple abstraction levels.

# Attribute Creation (Feature Generation)

---

- Create new attributes (features) that can capture the important information in a data set more effectively than the original ones
- Three general methodologies
  - Attribute extraction
    - Domain-specific
    - Mapping data to new space (see: data reduction)
      - E.g., Fourier transformation, wavelet transformation, manifold approaches (not covered)
  - Attribute construction
    - Combining features (see: discriminative frequent patterns in Chapter 7)
    - Data discretization

# Data Reduction 2: Numerosity Reduction

---

- Reduce data volume by choosing alternative, *smaller forms* of data representation
- **Parametric methods** (e.g., regression)
  - Assume the data fits some model, estimate model parameters, store only the parameters, and discard the data (except possible outliers)
  - Ex.: Log-linear models—obtain value at a point in  $m$ -D space as the product on appropriate marginal subspaces
- **Non-parametric** methods
  - Do not assume models
  - Major families: histograms, clustering, sampling, ...

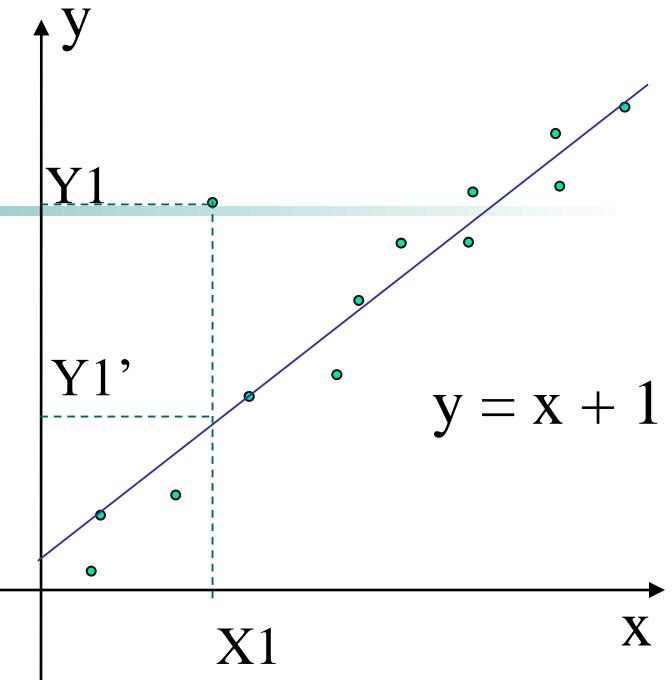
# Parametric Data Reduction: Regression and Log-Linear Models

---

- **Linear regression**
  - Data modeled to fit a straight line
  - Often uses the least-square method to fit the line
- **Multiple regression**
  - Allows a response variable  $Y$  to be modeled as a linear function of multidimensional feature vector
- **Log-linear model**
  - Approximates discrete multidimensional probability distributions

# Regression Analysis

- Regression analysis: A collective name for techniques for the modeling and analysis of numerical data consisting of values of a **dependent variable** (also called **response variable** or *measurement*) and of one or more *independent variables* (aka. **explanatory variables** or **predictors**)



- The parameters are estimated so as to give a "**best fit**" of the data
- Most commonly the best fit is evaluated by using the **least squares method**, but other criteria have also been used
- Used for prediction (including forecasting of time-series data), inference, hypothesis testing, and modeling of causal relationships

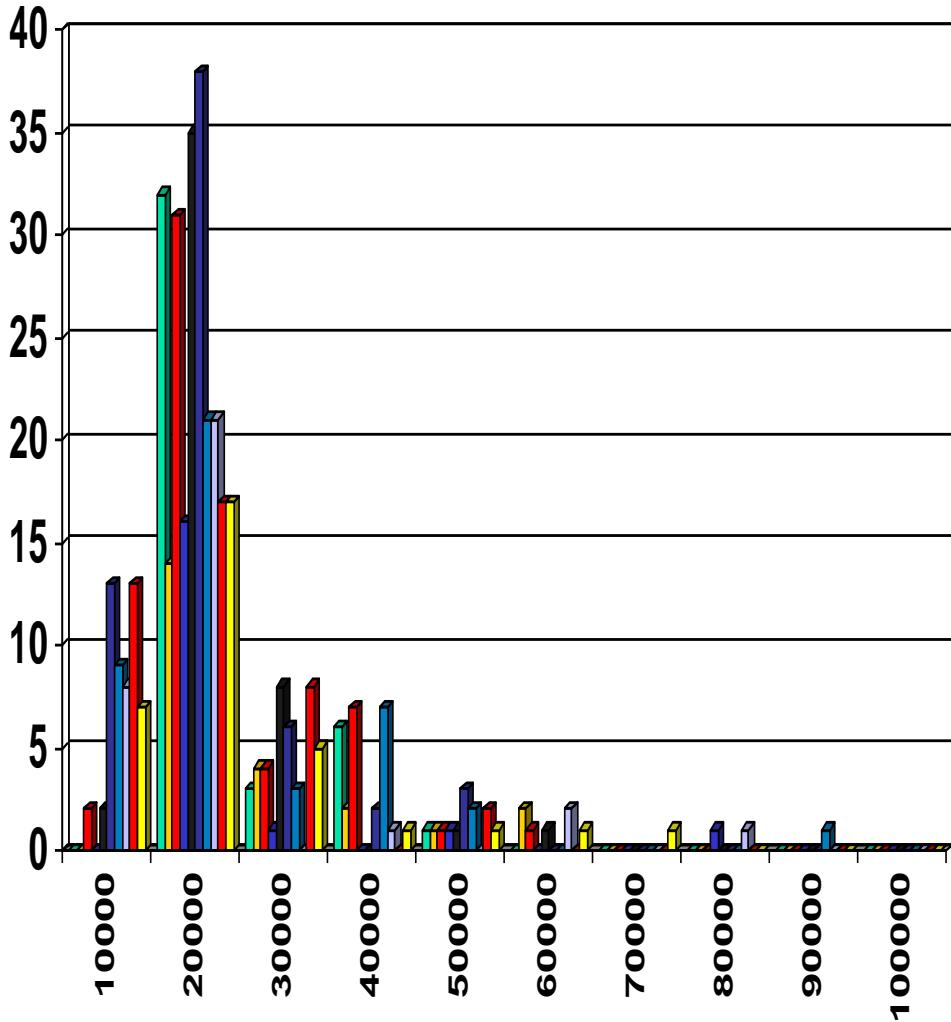
# Régress Analysis and Log-Linear Models

---

- Linear regression:  $Y = wX + b$ 
  - Two regression coefficients,  $w$  and  $b$ , specify the line and are to be estimated by using the data at hand
  - Using the least squares criterion to the known values of  $Y_1, Y_2, \dots, X_1, X_2, \dots$
- Multiple regression:  $Y = b_0 + b_1 X_1 + b_2 X_2$ 
  - Many nonlinear functions can be transformed into the above
- Log-linear models:
  - Approximate discrete multidimensional probability distributions
  - Estimate the probability of each point (tuple) in a multi-dimensional space for a set of discretized attributes, based on a smaller subset of dimensional combinations
  - Useful for dimensionality reduction and data smoothing

# Histogram Analysis

- Divide data into buckets and store average (sum) for each bucket
- Partitioning rules:
  - Equal-width: equal bucket range
  - Equal-frequency (or equal-depth)



# Clustering

---

- Partition data set into clusters based on similarity, and store cluster representation (e.g., centroid and diameter) only
- Can be very effective if data is clustered but not if data is “smeared”
- Can have hierarchical clustering and be stored in multi-dimensional index tree structures
- There are many choices of clustering definitions and clustering algorithms
- Cluster analysis will be studied in depth in Chapter 10

# Sampling

---

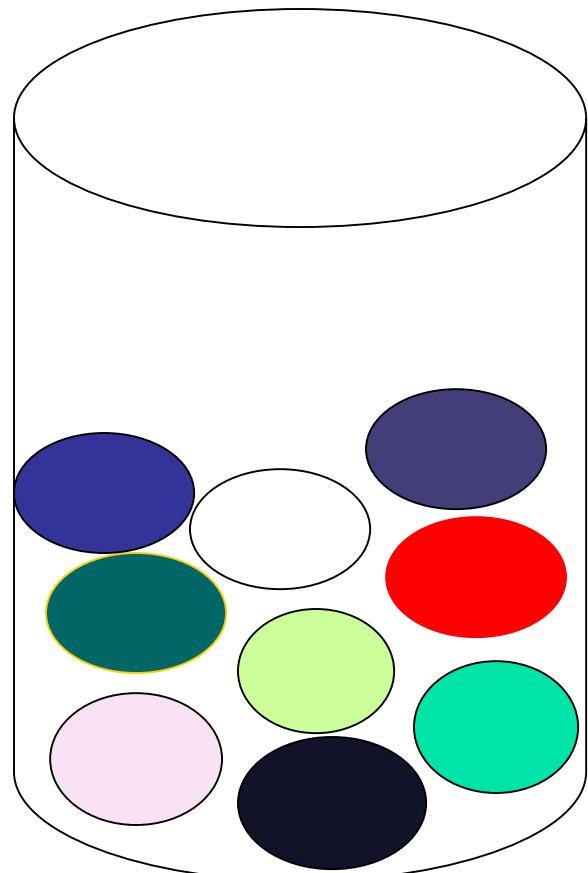
- Sampling: obtaining a small sample  $s$  to represent the whole data set  $N$
- Allow a mining algorithm to run in complexity that is potentially sub-linear to the size of the data
- Key principle: Choose a **representative** subset of the data
  - Simple random sampling may have very poor performance in the presence of skew
  - Develop adaptive sampling methods, e.g., stratified sampling:
- Note: Sampling may not reduce database I/Os (page at a time)

# Types of Sampling

---

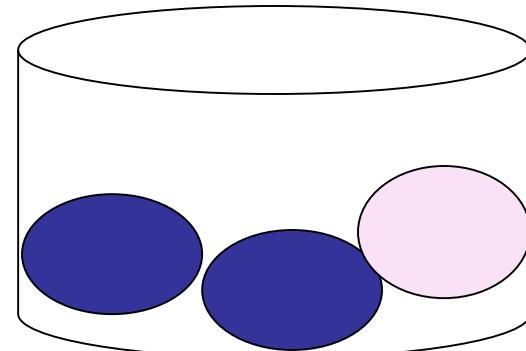
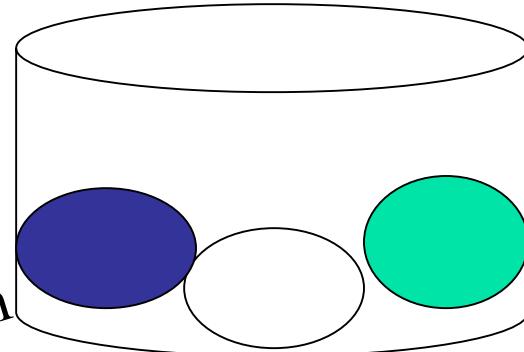
- **Simple random sampling**
  - There is an equal probability of selecting any particular item
- **Sampling without replacement**
  - Once an object is selected, it is removed from the population
- **Sampling with replacement**
  - A selected object is not removed from the population
- **Stratified sampling:**
  - Partition the data set, and draw samples from each partition (proportionally, i.e., approximately the same percentage of the data)
  - Used in conjunction with skewed data

# Sampling: With or without Replacement



SRSWOR  
(simple random  
sample without  
replacement)

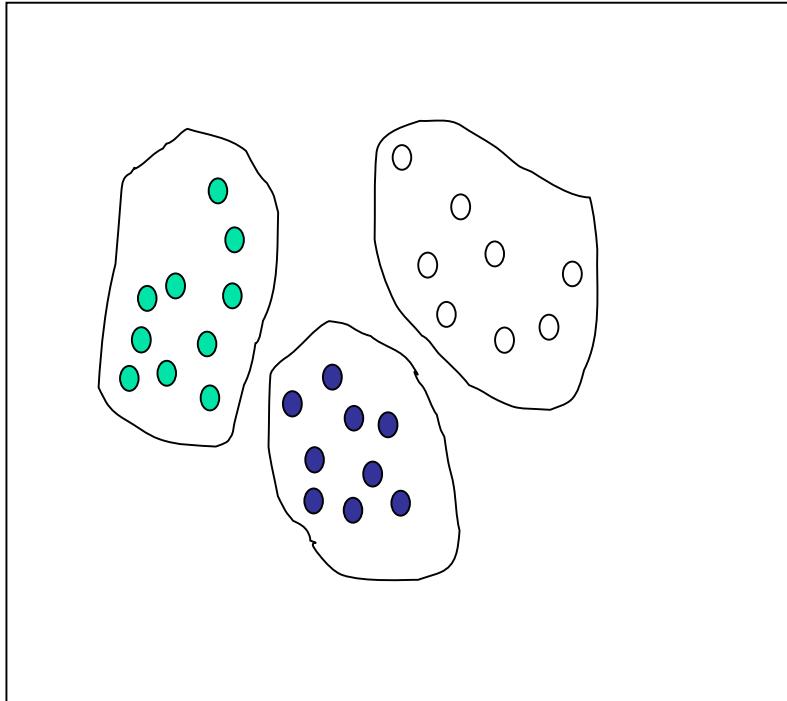
*SRSWR*



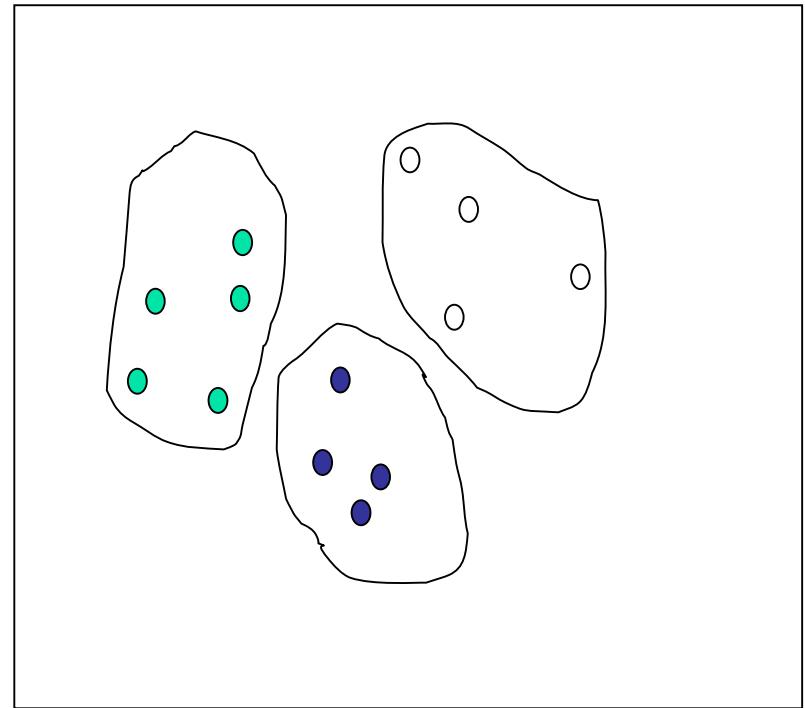
# Sampling: Cluster or Stratified Sampling

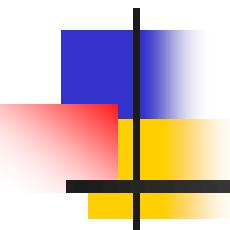
---

Raw Data



Cluster/Stratified Sample

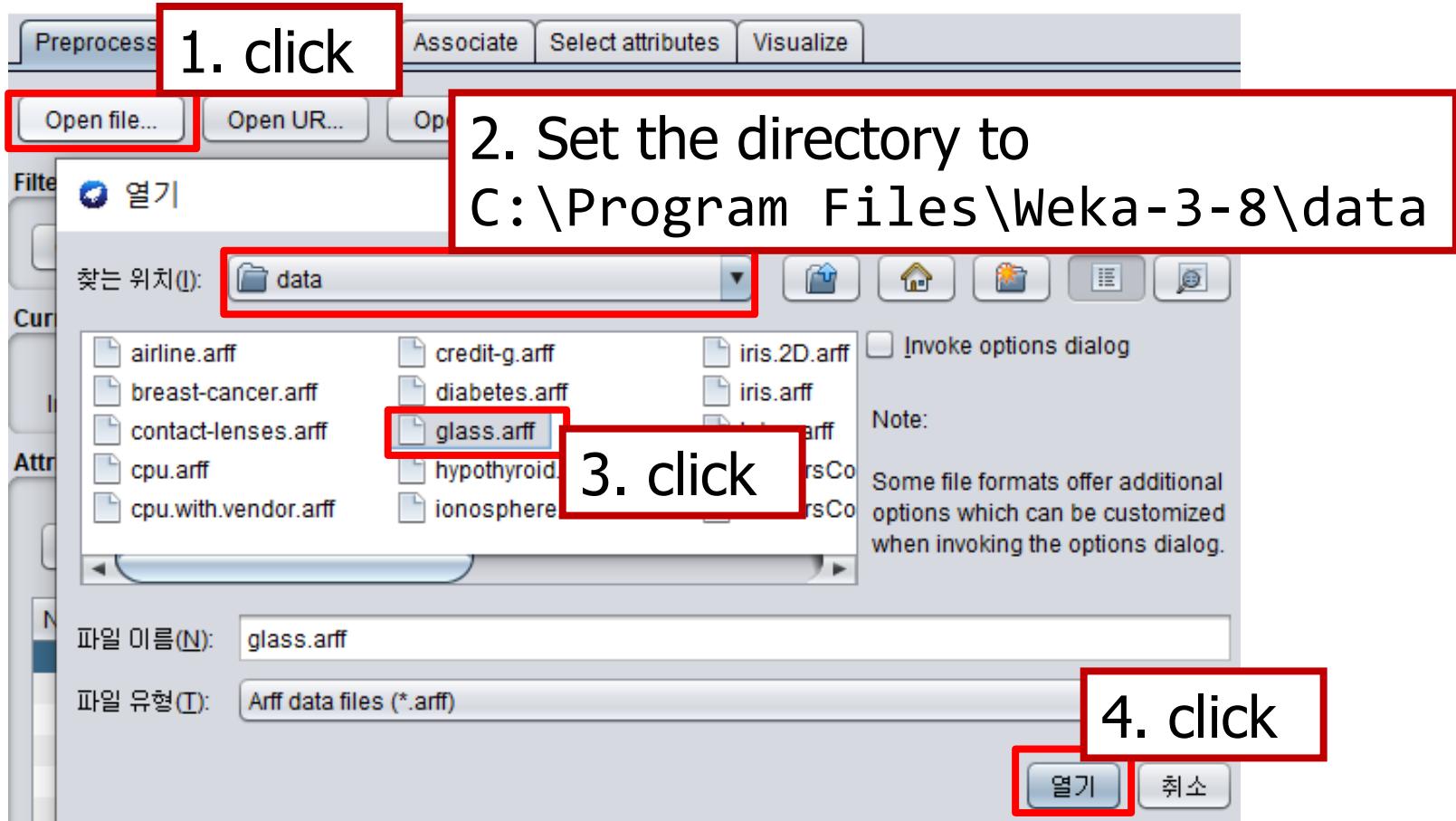




# Weka - Sampling

# Open the Glass Dataset

- Open:
  - C:\Program Files\Weka-3-8\data\glass.arff



# The Statistics of the Dataset

Current relation

Relation: Glass

Instances: 214

Sum of weights: 214

Attributes

All None Invert Pa

No.	Name
7	Ca
8	Ba
9	Fe
10	Type

Remove

Selected attribute

Type: Nominal

MISSING: 0 (0%) DISTINCT: ... Unique: 0 (0%)

No.	Label	Count	Weight
1	build wi...	70	70.0

1. Check the number of instances

2. Click

3. You can see the histogram of the class labels

Label	Count
build without lead	70
lead	76
lead with lime	17
lime	0
lime with lead	13
lime with iron	9
iron	29

**1. click**

Choose No

Current relation

Relation: Glass

Instances: 214

Attributes

All

No.	Name	Type	Count
5	S	Nominal	70
6	K	Nominal	76

weka

filters

- AllFilter
- MultiFilter
- RenameRelation

supervised

unsupervised

- attribute
- instance
  - NonSparseToSparse
  - Randomize
  - RemoveDuplicates
  - RemoveFolds
  - RemoveFrequentValues
  - RemoveMisclassified
  - RemovePercentage
  - RemoveRange
  - RemoveWithV

**2. click**

Resample

ReservoirSample

SparseToNonSparse

**Selected attribute**

Name: Type  
Missing: 0 (0%)      Distinct

No.	Label	Count
1	build win...	70
2	build win...	76
3	vehic win...	17
4	vehic win...	0

Class: Type (Nom)

# Configuring the Sampling

Filter

1. click

Choose Resample -S 1 -Z 100.0 Apply Stop

Current relation

Relation: Glass Attributes: 10  
Instances: 214 Sum of weights: 214

Selected attribute

Name:	Type	Type:	Nominal
Missing:	0 (0%)	Distinct:	6
Unique:	0 (0%)		

Attributes

All None Invert Pattern

No.	Name
5	Si
6	v

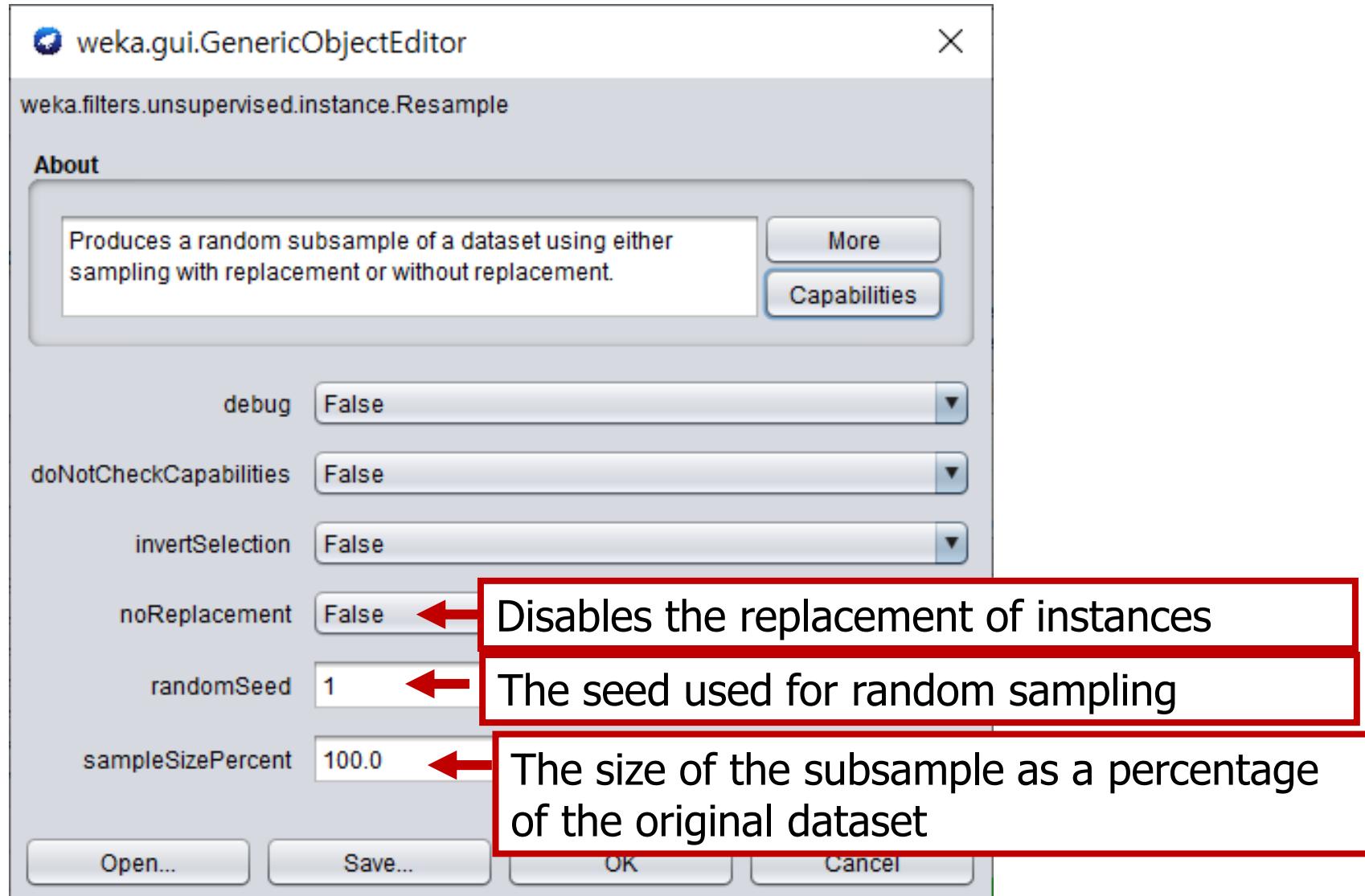
No. Label Count Weight

No.	Label	Count	Weight
1	build win...	70	70.0
2	build win...	76	76.0
3	vehic win...	17	17.0
4	vehic win...	0	0.0

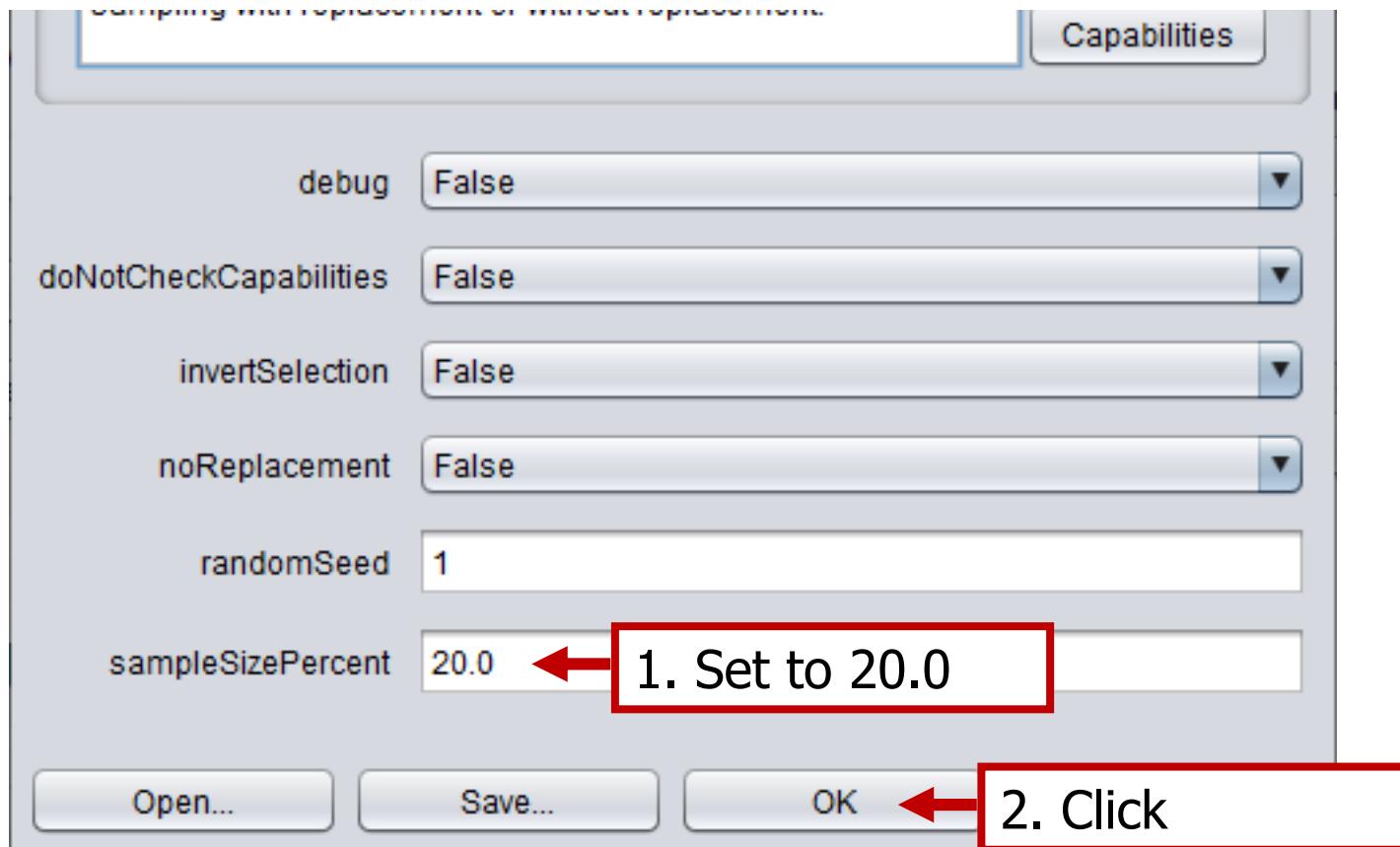
Class: Type (Nom) Visualize All

The screenshot shows the Weka interface with the 'Sampling' filter selected. The top toolbar has a 'Choose' button, a red-highlighted 'Resample -S 1 -Z 100.0' button, and 'Apply' and 'Stop' buttons. The 'Current relation' panel shows 'Relation: Glass', 'Attributes: 10', 'Instances: 214', and 'Sum of weights: 214'. The 'Selected attribute' panel shows a table with four rows: 'Name: Type', 'Missing: 0 (0%)', 'Distinct: 6', and 'Unique: 0 (0%)'. Below it is a table of attribute counts and weights. The 'Attributes' panel shows two selected attributes: 'Si' and 'v'. The 'Visualize All' button is visible at the bottom right.

# Configuring the Sampling

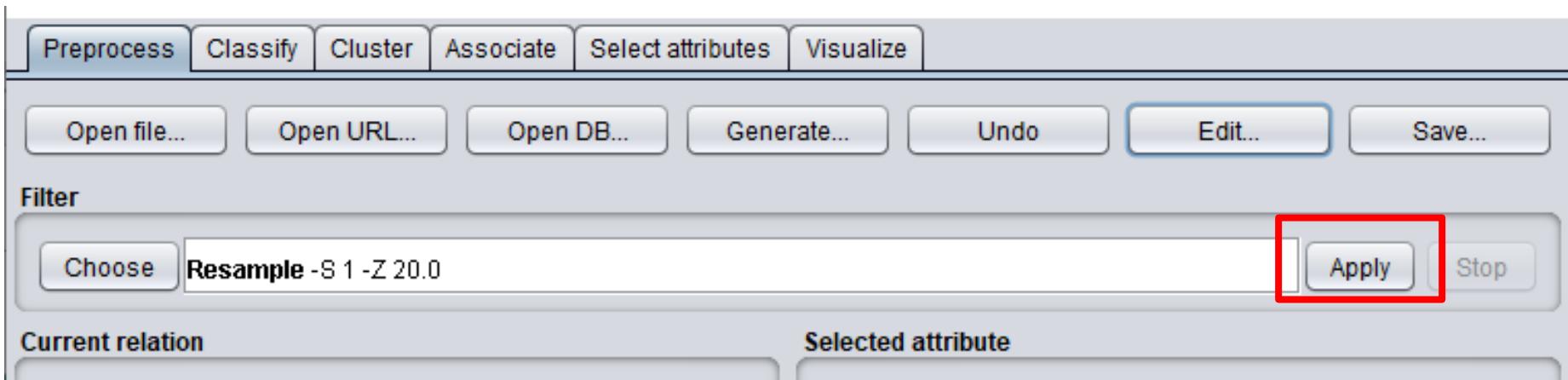


# Sampling With Replacement



# Sampling With Replacement

- Click 'apply' to run sampling



# Sampling With Replacement

Choose Resample -S 1 -Z 20.0 Apply Stop

Current relation

Relation: Class I Instances: 42 Sum of weights: 42 Missing: 0 (0%) Distinct: 6 Type: Nominal Unique: 1 (2%)

Attributes

All None Invert Pattern

No.	Name	Count	Weight
5	Si	16	16.0
6	K	15	15.0
7	Ca	3	3.0
8	Ba	0	0.0
9	Fe	1	1.0
10	Type	2	2.0

1. Click

2. You can see the histogram of the class labels

Class Label	Count
1	16
2	15
3	3
4	0
5	1
6	2
7	5

Status OK Log x 0

# Sampling With Replacement

- Click 'edit' button to see the instances

The screenshot shows the Weka interface for data mining. The top menu bar includes 'File' (with 'Open file...', 'Open URL...', 'Open DB...', 'Generate...', 'Undo'), 'Edit...' (which is highlighted with a red box), and 'Save...'. Below the menu is a toolbar with buttons for 'Choose', 'Resample -S 1 -Z 20.0', 'Apply', and 'Stop'. A status bar at the bottom displays 'Relation: Glass-weka.filters.unsu...', 'Attributes: 10', 'Instances: 42', 'Sum of weights: 42'. On the left, there's a 'Selected attributes' panel with buttons for 'All', 'None', 'Invert', and 'Pattern'. A list of attributes is shown with checkboxes: 5 Si, 6 K, 7 Ca, 8 Ba, 9 En. To the right is a 'Selected attribute' panel for 'Type: Nominal'. It shows a table with columns 'No.', 'Label', 'Count', and 'Weight'. The data is as follows:

No.	Label	Count	Weight
1	build wind fl...	16	16.0
2	build wind n...	15	15.0
3	vehic wind fl...	3	3.0
4	vehic wind n...	0	0.0

Below this is a 'Class: Type (Nom)' dropdown and a 'Visualize All' button. At the bottom, there are two colored bars: a blue bar labeled '16' and a red bar labeled '15'.

# Sampling With Replacement

Viewer

Relation: Glas

1. Click it to sort by the 1<sup>st</sup> column

No.	1: RI	2: Na	3: Mg	4: Al	5: Si	6: K	7: Ca	8: Ba	9: Fe	10: Type
	Numeric	Nominal								
1	1.51...	15.15	0.0	2.25	73.5	0.0	8.34	0.63	0.0	headl...
2	1.51...	14.14	0.0	2.68	73.39	0.08	9.07	0.61	0.05	headl...
3	1.51...	12.82	3.52	1.9	72.86	0.69	7.97	0.0	0.0	build ...
4	1.51...	13.88	1.78	1.79	73.1	0.0	8.67	0.76	0.0	headl...
5	1.51...	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	build ...
6	1.51...	13.36	3.58	1.49	72.72	0.45	8.21	0.0	0.0	build ...
7	1.51...	14.38	0.0	1.94	73.61	0.0	8.48	1.57	0.0	headl...
8	1.51...	13.72	3.68	1.81	72.06	0.64	7.88	0.0	0.0	build ...
9	1.51...	12.87	3.48	1.33	73.04	0.56	8.43	0.0	0.0	build ...
10	1.51...	12.2	3.25	1.16	73.55	0.62	8.9	0.0	0.24	build ...
11	1.51...	13.0	3.6	1.36	72.99	0.57	8.4	0.0	0.11	build ...
12	1.51...	12.71	3.42	1.2	73.2	0.59	8.64	0.0	0.0	build ...
13	1.51...	12.69	3.54	1.34	72.95	0.57	8.75	0.0	0.0	build ...
14	1.51...	13.08	3.49	1.28	72.86	0.6	8.49	0.0	0.0	build ...

# Sampling With Replacement



Viewed Drag it to increase the width of the 1<sup>st</sup> column

Relation: Glass-weka.filters.unsupervised.instance.Resample-S1-Z20.0

No.	1: RI Numeric	2: Na Numeric	3: Mg Numeric	4: Al Numeric	5: Si Numeric	6: K Numeric	7: Ca Numeric	8: Ba Numeric	9: Fe Numeric	10: Type Nominal
1	1.51508	15.15	0.0	2.25	73.5	0.0	8.34	0.63	0.0	head...
2	1.51545	14.14	0.0	2.68	73.39	0.08	9.07	0.61	0.05	head...
3	1.5159	12.82	3.52	1.9	72.86	0.69	7.97	0.0	0.0	build ...
4	1.51613	13.88	1.78	1.79	73.1	0.0	8.67	0.76	0.0	head...
5	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	build ...
6	1.51625	13.36	3.58	1.49	72.72	0.45	8.21	0.0	0.0	build ...
7	1.51651	14.38	0.0	1.94	73.61	0.0	8.48	1.57	0.0	head...
8	1.51708	13.72	3.68	1.81	72.06	0.64	7.88	0.0	0.0	build ...
9	1.51721	12.87	3.48	1.33	73.04	0.56	8.43	0.0	0.0	build ...
10	1.51743	12.2	3.25	1.16	73.55	0.62	8.9	0.0	0.24	build ...
11	1.51755	12.71	3.42	1.2	73.2	0.59	8.64	0.0	0.0	build ...
12	1.51755	13.0	3.6	1.36	72.99	0.57	8.4	0.0	0.11	build ...
13	1.51783	12.69	3.54	1.34	72.95	0.57	8.75	0.0	0.0	build ...

# Sampling With Replacement

- You can see the duplicated instances since we samples with replacement

19	1.51813	13.43	3.98	1.18	72.49	0.58	8.15	0.0	0.0	build ...
20	1.51829	13.24	3.9	1.41	72.33	0.55	8.31	0.0	0.1	build ...
21	1.51832	13.33	3.34	1.54	72.14	0.56	8.99	0.0	0.0	vehic ...
22	1.51832	13.33	3.34	1.54	72.14	0.56	8.99	0.0	0.0	vehic ...
23	1.51888	14.99	0.78	1.74	72.5	0.0	9.95	0.0	0.0	table...
24	1.519	13.49	3.48	1.35	71.95	0.55	9.0	0.0	0.0	build ...
25	1.51966	14.77	3.75	0.29	72.02	0.03	9.0	0.0	0.0	build ...
26	1.51966	14.77	3.75	0.29	72.02	0.03	9.0	0.0	0.0	build ...

# Configuring the Sampling

Filter

1. click

Choose Resample -S 1 -Z 100.0 Apply Stop

Current relation

Relation: Glass Attributes: 10  
Instances: 214 Sum of weights: 214

Selected attribute

Name:	Type	Type:	Nominal
Missing:	0 (0%)	Distinct:	6
Unique:	0 (0%)		

Attributes

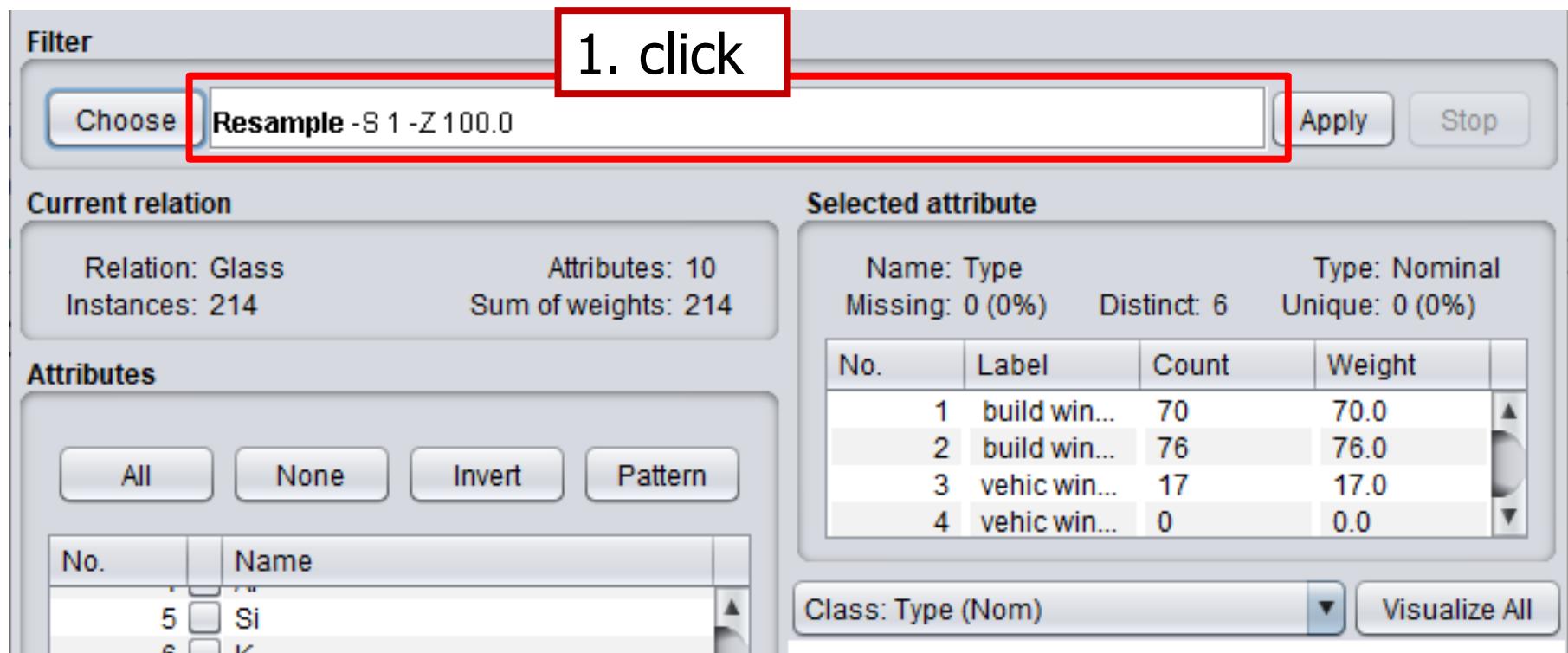
All None Invert Pattern

No.	Name
5	Si
6	v

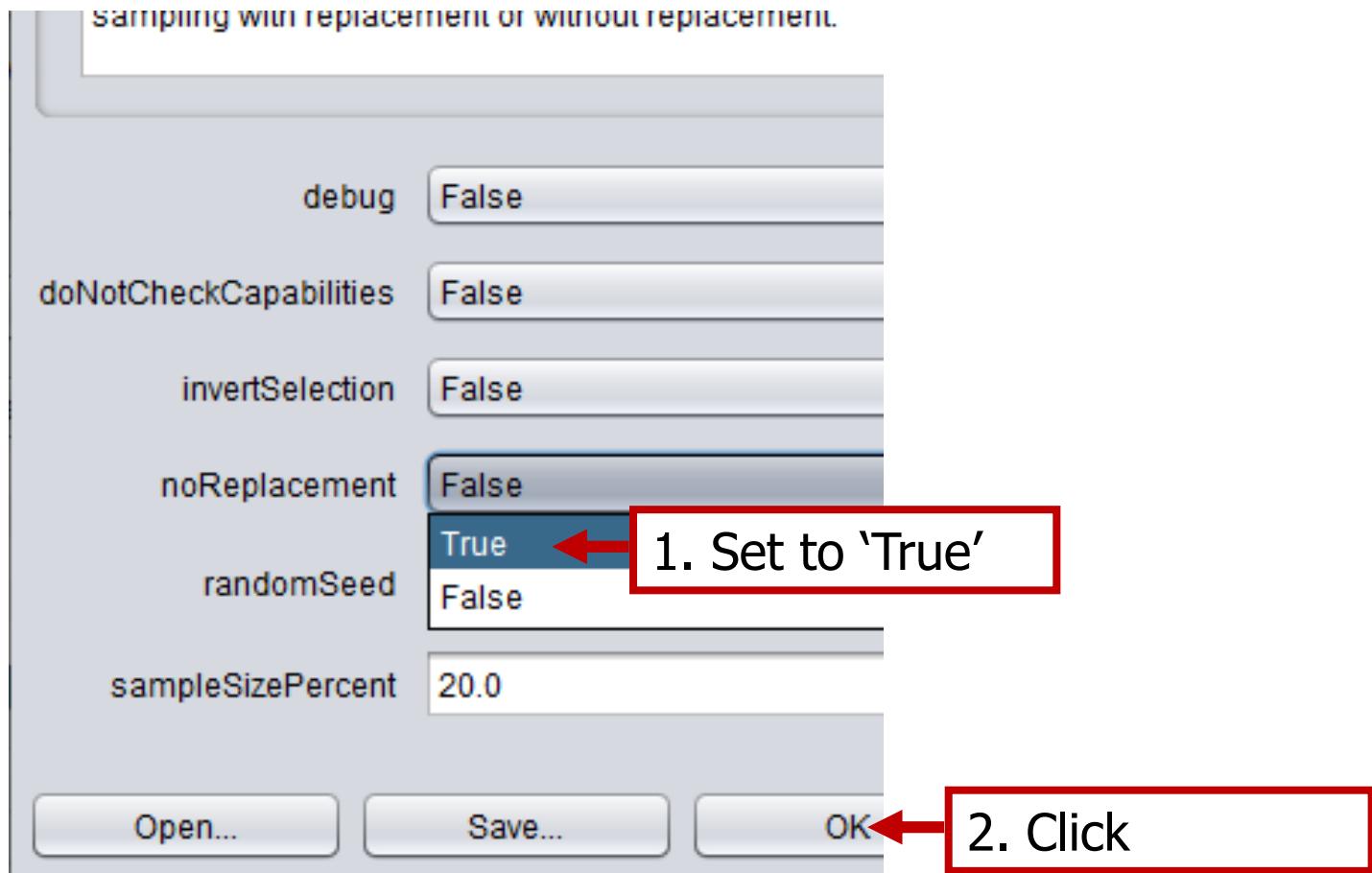
No. Label Count Weight

No.	Label	Count	Weight
1	build win...	70	70.0
2	build win...	76	76.0
3	vehic win...	17	17.0
4	vehic win...	0	0.0

Class: Type (Nom) Visualize All



# Sampling Without Replacement



# Sampling Without Replacement

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Open file... Open URL... Open DB... Generate... Undo Edit... Save...

Filter

Choose Resample -S 1 -Z 20.0 -no-replacement Apply

Current relation

Relation: Glass-weka.filters.unsu... Attributes: 10  
Instances: 42 Sum of weights: 42

Attributes

All None Invert Pattern

No. Name

Selected attribute

Name: Type Type: Nominal  
Missing: 0 (0%) Distinct: 6 Unique: 1 (2%)

No.	Label	Count	Weight
1	build wind fl...	16	16.0
2	build wind n...	15	15.0
3	vehic wind fl...	3	3.0
4	vehic wind n...	0	0.0

# Sampling Without Replacement

Choose Resample -S 1 -Z 20.0 -no-replacement Apply Stop

Current relation

Relation: Glass-weka.filters.unsu... Attributes: 10  
Instances: 42 Sum of weights: 42

Selected attribute

Name: Type Type: Nominal  
Missing: 0 (0%) Distinct: 6 Unique: 1 (2%)

No.	Label	Count	Weight
1	build wind fl...	12	12.0

Attributes

All None Invert

0. Name

5 Si  
6 K  
7 Ca  
8 Ba  
9 Fe  
10 Type

Remove

1. Click

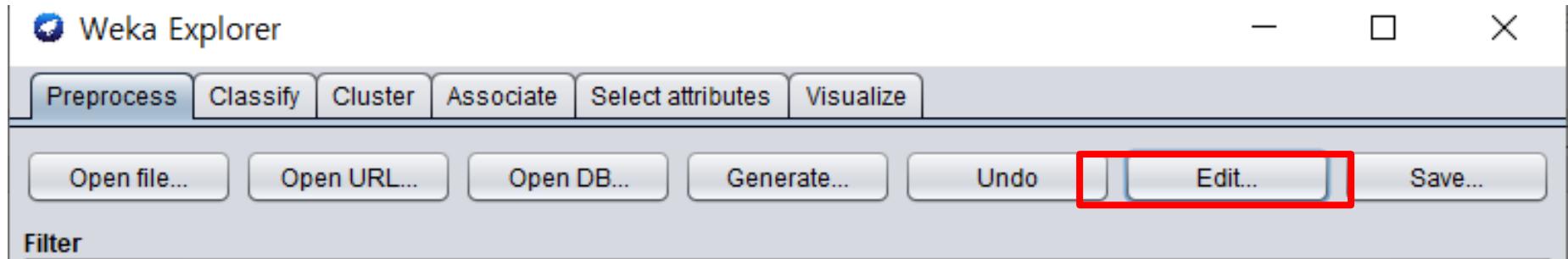
2. You can see the histogram of the class labels

Class Label	Count
0	12
1	13
2	4
3	0
4	3
5	1
6	9

Log x 0

# Sampling Without Replacement

- Click 'edit' button to see the instances



# Sampling Without Replacement

- There is no duplicates

Viewer

Relation: Glass-weka.filters.unsupervised.instance.Resample-S1-Z20.0-no-replacement

No.	1: RI Numeric	2: Na Numeric	3: Mg Numeric	4: Al Numeric	5: Si Numeric	6: K Numeric	7: Ca Numeric	8: Ba Numeric	9: Fe Numeric	10: Type Nominal
1	1.51215	12.99	3.47	1.12	72.98	0.62	8.35	0.0	0.31	build ...
2	1.51508	15.15	0.0	2.25	73.5	0.0	8.34	0.63	0.0	headl...
3	1.51514	14.85	0.0	2.42	73.72	0.0	8.39	0.56	0.0	headl...
4	1.51531	14.38	0.0	2.66	73.1	0.04	9.08	0.64	0.0	headl...
5	1.51588	13.12	3.41	1.58	73.26	0.07	8.39	0.0	0.19	build ...
6	1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0.0	0.26	build ...
7	1.51602	14.85	0.0	2.38	73.28	0.0	8.76	0.64	0.09	headl...
8	1.51605	12.9	3.44	1.45	73.06	0.44	8.27	0.0	0.0	build ...
9	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	build ...
10	1.51623	14.14	0.0	2.88	72.61	0.08	9.18	1.06	0.0	headl...
11	1.51629	12.71	3.33	1.49	73.28	0.67	8.24	0.0	0.0	build ...
12	1.51646	13.04	3.4	1.26	73.01	0.52	8.58	0.0	0.0	vehic ...
13	1.51655	12.75	2.85	1.44	73.27	0.57	8.79	0.11	0.22	build ...
14	1.51658	14.8	0.0	1.99	73.11	0.0	8.28	1.71	0.0	headl...
15	1.5166	12.99	3.18	1.23	72.97	0.58	8.81	0.0	0.24	build ...
16	1.51665	13.14	3.45	1.76	72.48	0.6	8.38	0.0	0.17	vehic ...
17	1.51673	13.3	3.64	1.53	72.53	0.65	8.03	0.0	0.29	build ...
18	1.51719	14.75	0.0	2.0	73.02	0.0	8.53	1.59	0.08	headl...

# Configuring the Sampling

Filter

1. click

Choose Resample -S 1 -Z 100.0 Apply Stop

Current relation

Relation: Glass Attributes: 10  
Instances: 214 Sum of weights: 214

Selected attribute

Name:	Type	Type:	Nominal
Missing:	0 (0%)	Distinct:	6
Unique:	0 (0%)		

Attributes

All None Invert Pattern

No.	Name
5	Si
6	v

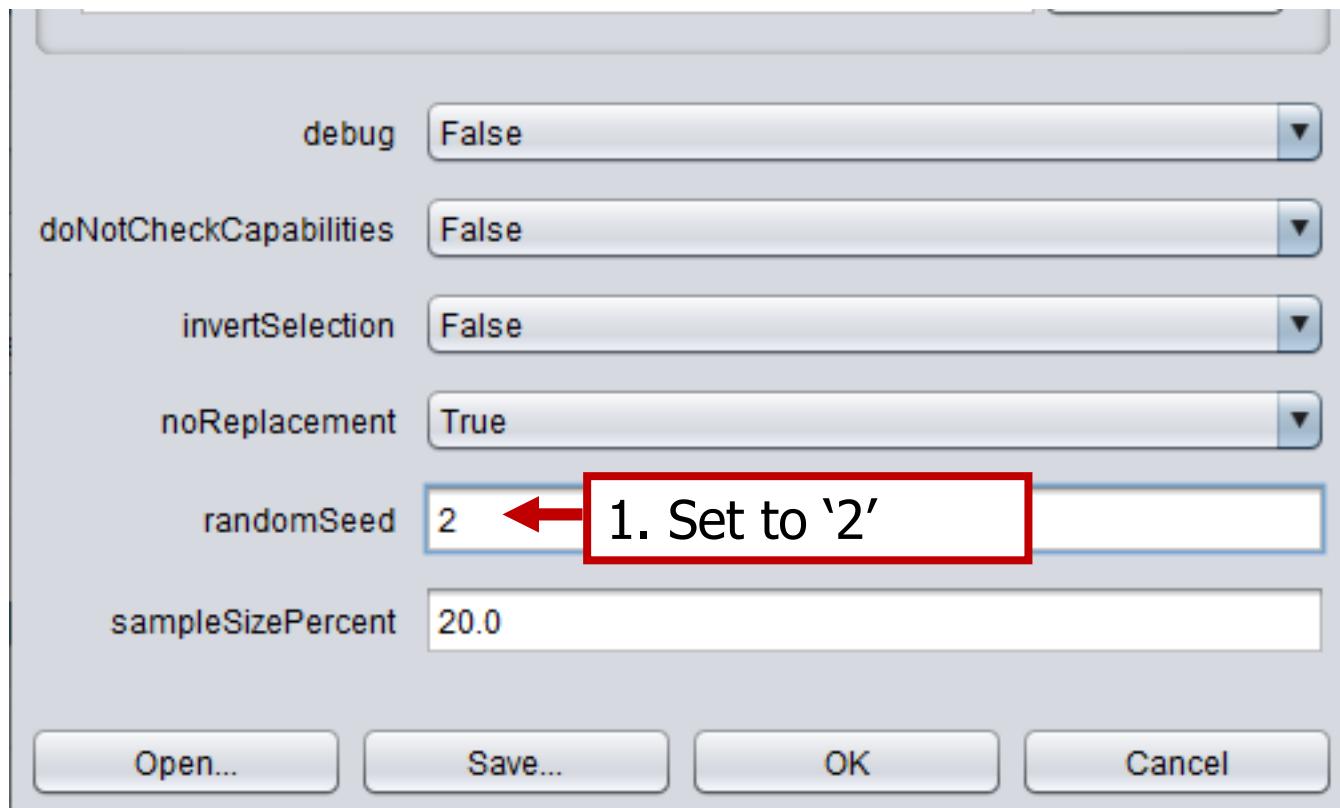
No. Label Count Weight

No.	Label	Count	Weight
1	build win...	70	70.0
2	build win...	76	76.0
3	vehic win...	17	17.0
4	vehic win...	0	0.0

Class: Type (Nom) Visualize All

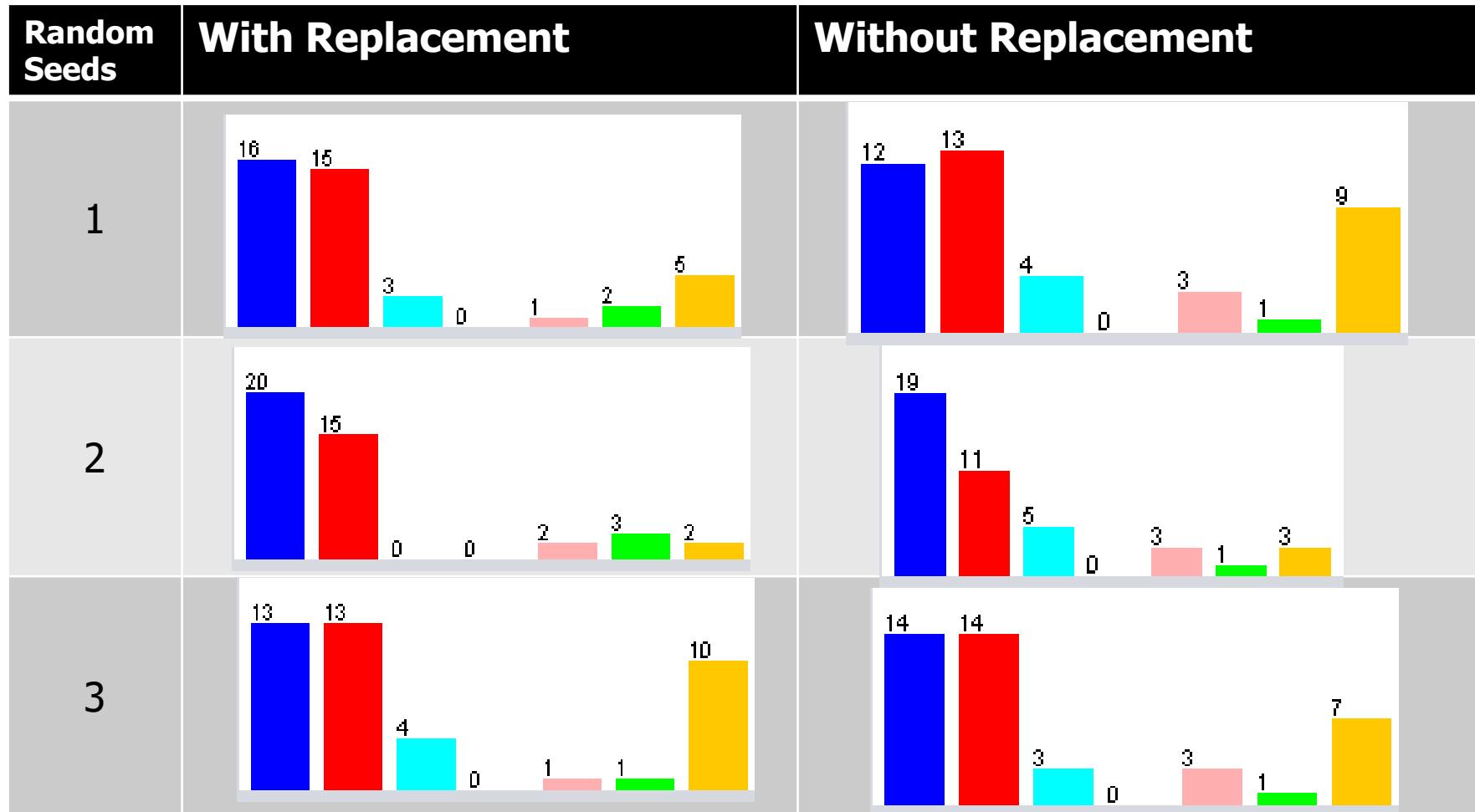
No.	Label	Count	Weight
1	build win...	70	70.0
2	build win...	76	76.0
3	vehic win...	17	17.0
4	vehic win...	0	0.0

# Changing the Random Seed



# Changing the Random Seed

- Repeating the experiments with changing the random seed, you can see that the distribution of the class labels varies



# Stratified Sampling

Filter **1. click**

Choose weka filters

Current relation

Relation: Instances:

Attributes

All

No. 5 6 ...

Selected attribute

Name: Type  
Missing: 0 (0%) Distinct

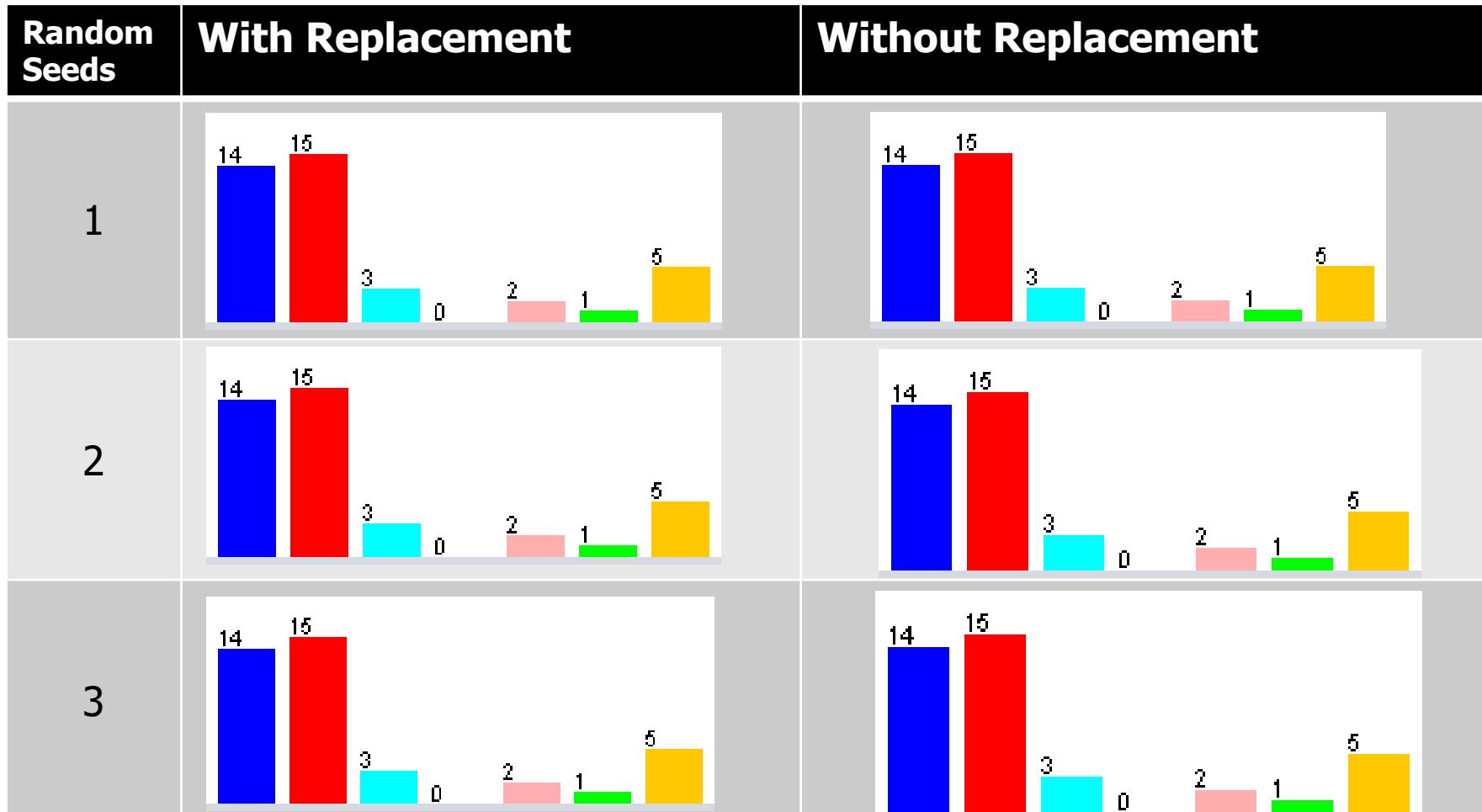
No.	Label	Count
1	build win...	70
2	build win...	76
3	vehic win...	17
4	vehic win...	0

ss: Type (Nom)

AllFilter MultiFilter RenameRelation supervised attribute instance ClassBalancer Resample SpreadSubsample StratifiedRemoveFolds

# Stratified Sampling

- Similarly repeating the experiments with changing the random seed, you can see that the distribution of the class labels is constant



# Stratified Sampling

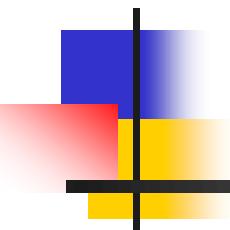
- You can see that the instances are sampled differently

Seed=1 without replacement

No.	1: RI	2: Na	3: Mg	4: Al	5: Si	6: K	7: Ca	8: Ba	9: Fe	10: Type
	Numeric	Nominal								
1	1.51...	13.21	2.81	1.29	72.98	0.51	9.02	0.0	0.09	build ...
2	1.51...	13.89	3.6	1.36	72.73	0.48	7.83	0.0	0.0	build ...
3	1.51...	12.74	3.48	1.35	72.96	0.64	8.68	0.0	0.0	build ...
4	1.51...	13.81	3.58	1.32	71.72	0.12	8.67	0.69	0.0	build ...
5	1.51...	12.79	3.61	1.62	72.97	0.64	8.07	0.0	0.26	build ...
6	1.51...	12.86	3.56	1.27	73.21	0.54	8.38	0.0	0.17	build ...
7	1.51...	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	build ...
8	1.51...	13.2	3.33	1.28	72.36	0.6	9.14	0.0	0.11	build ...
9	1.51...	12.56	3.52	1.43	73.15	0.57	8.54	0.0	0.0	build ...

Seed=2 without replacement

No.	1: RI	2: Na	3: Mg	4: Al	5: Si	6: K	7: Ca	8: Ba	9: Fe	10: Type
	Numeric	Nominal								
1	1.51...	13.38	3.5	1.15	72.85	0.5	8.43	0.0	0.0	build ...
2	1.51...	14.77	3.75	0.29	72.02	0.03	9.0	0.0	0.0	build ...
3	1.51...	13.02	3.54	1.69	72.73	0.54	8.44	0.0	0.07	build ...
4	1.51...	13.3	3.6	1.14	73.09	0.58	8.17	0.0	0.0	build ...
5	1.51...	13.81	3.58	1.32	71.72	0.12	8.67	0.69	0.0	build ...
6	1.52...	13.21	3.77	0.79	71.99	0.13	10.02	0.0	0.0	build ...
7	1.51...	13.48	3.74	1.17	72.99	0.59	8.03	0.0	0.0	build ...
8	1.519	13.49	3.48	1.35	71.95	0.55	9.0	0.0	0.0	build ...
9	1.51...	12.82	3.55	1.49	72.75	0.54	8.52	0.0	0.19	build ...
10	1.51...	12.74	3.48	1.35	72.96	0.64	8.68	0.0	0.0	build ...



# Python - Sampling

# Import necessary Libraries

---

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.utils import resample
from collections import Counter
```

- **resample** : A module to sample data
- **Counter** : A module to count the number of discrete values

# Make toy Data

---

```
In [2]: x_toy = [1,2,3,4,5]  
y_toy = [6,7,8,9,10]
```

- Before applying sampling to real data, we will apply sampling to toy data to see the behaviors of sampling strategies

# Sampling with Replacement

```
In [3]: X_toy_sample, y_toy_sample = resample(X_toy, y_toy,  
                                         replace = True, #Default is True  
                                         n_samples = 7,  
                                         random_state = 0)  
  
print(X_toy_sample, y_toy_sample)
```

[5, 1, 4, 4, 4, 2, 4] [10, 6, 9, 9, 9, 7, 9]

- If the parameter 'replace = True' in resample, it returns data sampled with replacement (Default is True)

# Sampling with Replacement

```
In [3]: X_toy_sample, y_toy_sample = resample(X_toy, y_toy,
                                         replace = True, #Default is True
                                         n_samples = 7,
                                         random_state = 0)
print(X_toy_sample, y_toy_sample)
```

[5, 1, 4, 4, 4, 2, 4] [10, 6, 9, 9, 9, 7, 9]

You can give multiple inputs

Number of samples  
If none, the size of data is assigned

Random seed

- If the parameter 'replace = True' in resample, it returns data sampled with replacement (Default is True)

# Sampling without Replacement

```
In [4]: X_toy_sample, y_toy_sample = resample(X_toy, y_toy,
                                             replace = False,
                                             n_samples = 5,
                                             random_state = 0)
print(X_toy_sample, y_toy_sample)
```

```
[3, 1, 2, 4, 5] [8, 6, 7, 9, 10]
```

The number of samples cannot exceed the size of the data

- If the parameter 'replace = False' in resample, it returns data sampled without replacement  
(Default is True)

# Loading Data

```
In [5]: df = pd.read_csv('glass.csv')
X = df.values[:, :-1]
y = df.values[:, -1]
N = len(X)
```

- Load 'glass.csv' data
- len(X): returns the size of X  
(number of rows)

# Sampling with Replacement

```
In [6]: X_sample, y_sample = resample(X, y,  
                                 replace = True,  
                                 n_samples = int(N*0.2),  
                                 random_state = 0)
```

- `int(number)`: returns floor of the input number as integer type
- `N*0.2`: 20% of the data size

# Counting sampled Labels

---

```
In [7]: c = Counter(y_sample)  
print(c)
```

- Counter(array): counts the number of distinct values that appear in the array
- Returns Counter object containing (value, count) pairs

# Counting sampled Labels

```
In [7]: c = Counter(y_sample)  
print(c)
```

- Counter(array): counts the number of distinct values that appear in the array
- Returns (value, count) pairs
- Output:

```
Counter({'build wind float': 16, 'build wind non-float': 13,  
        'headlamps': 5, 'tableware': 4, 'vehic wind float': 4})
```

# Counting sampled Labels

---

```
In [8]: list(c.values())
```

```
Out[8]: [16, 5, 4, 4, 13]
```

- If you want only the counts as python list, call `c.values()` and wrap it with `list()`

# Sampling without Replacement

---

```
In [9]: X_sample, y_sample = resample(X, y,
                                     replace = False,
                                     n_samples = int(N*0.2),
                                     random_state = 0)
c = Counter(y_sample)
print(c)
```

# Sampling without Replacement

```
In [9]: X_sample, y_sample = resample(X, y,
                                     replace = False,
                                     n_samples = int(N*0.2),
                                     random_state = 0)
c = Counter(y_sample)
print(c)
```

## ■ Output:

```
Counter({'build wind non-float': 19, 'build wind float': 13,
         'headlamps': 4, 'containers': 3, 'vehic wind float': 2,
         'tableware': 1})
```

# Sampling without Replacement

```
In [9]: X_sample, y_sample = resample(X, y,
                                     replace = False,
                                     n_samples = int(N*0.2),
                                     random_state = 0)
c = Counter(y_sample)
print(c)
```

## ■ Output:

```
Counter({'build wind non-float': 19, 'build wind float': 13,
         'headlamps': 4, 'containers': 3, 'vehic wind float': 2,
         'tableware': 1})
```

# Handling zero-count Labels

---

- Note that in Sampling with replacement example, instances with 'containers' are not sampled, so it did not appear in the counter result

---

```
Counter({'build wind float': 16, 'build wind non-float': 13,  
        'headlamps': 5, 'tableware': 4, 'vehic wind float': 4})
```

---

- Since pre-built modules cannot know labels that do not appear in the data, you have to build your own function if you want them in the list

# Changing the Random seed

```
In [10]: for i in range(5):
    X_sample, y_sample = resample(X, y,
                                   replace = True,
                                   n_samples = int(N*0.2),
                                   random_state = i)
    c = Counter(y_sample)
    print(sorted(c.items()))
```

- Output the label counts changing random\_state with for loop

# Changing the Random seed

```
In [10]: for i in range(5):
    X_sample, y_sample = resample(X, y,
                                   replace = True,
                                   n_samples = int(N*0.2),
                                   random_state = i)
    c = Counter(y_sample)
    print(sorted(c.items()))
```

- Output the label counts changing random\_state with for loop
- c returns Counter object, where c.items() returns list of (value, count) pairs

# Changing the Random seed

```
In [10]: for i in range(5):
    X_sample, y_sample = resample(X, y,
                                   replace = True,
                                   n_samples = int(N*0.2),
                                   random_state = i)
    c = Counter(y_sample)
    print(sorted(c.items()))
```

- Output the label counts changing random\_state with for loop
- c returns Counter object, where c.items() returns list of (value, count) pairs
- sorted(list) returns sorted list in increasing order
  - This case, the list is sorted by order of labels

# Changing the Random seed

```
In [10]: for i in range(5):
    X_sample, y_sample = resample(X, y,
                                   replace = True,
                                   n_samples = int(N*0.2),
                                   random_state = i)
    c = Counter(y_sample)
    print(sorted(c.items()))
```

## ■ Output:

```
[("build wind float", 16), ("build wind non-float", 13), ("vehic wind float", 4), ('headlamps', 5), ('tableware', 4)]
[("build wind float", 15), ("build wind non-float", 15), ("vehic wind float", 2), ('containers', 3), ('headlamps', 5),
('tableware', 2)]
[("build wind float", 9), ("build wind non-float", 20), ("vehic wind float", 3), ('containers', 3), ('headlamps', 5),
('tableware', 2)]
[("build wind float", 15), ("build wind non-float", 18), ("vehic wind float", 1), ('containers', 1), ('headlamps', 5),
('tableware', 2)]
[("build wind float", 15), ("build wind non-float", 15), ("vehic wind float", 2), ('containers', 2), ('headlamps', 7),
('tableware', 1)]
```

# Changing the Random seed

```
In [10]: for i in range(5):
    X_sample, y_sample = resample(X, y,
                                   replace = True,
                                   n_samples = int(N*0.2),
                                   random_state = i)
    c = Counter(y_sample)
    print(sorted(c.items()))
```

## ■ Output:

```
[("build wind float", 16), ("build wind non-float", 13), ("vehic wind float", 4), ('headlamps', 5), ('tableware', 4)]
[("build wind float", 15), ("build wind non-float", 15), ("vehic wind float", 2), ('containers', 3), ('headlamps', 5),
('tableware', 2)]
[("build wind float", 9), ("build wind non-float", 20), ("vehic wind float", 3), ('containers', 3), ('headlamps', 5),
('tableware', 2)]
[("build wind float", 15), ("build wind non-float", 18), ("vehic wind float", 1), ('containers', 1), ('headlamps', 5),
('tableware', 2)]
[("build wind float", 15), ("build wind non-float", 15), ("vehic wind float", 2), ('containers', 2), ('headlamps', 7),
('tableware', 1)]
```

Note that some labels are surrounded by ""

# Changing the Random seed

```
In [11]:
```

```
for i in range(5):
    X_sample, y_sample = resample(X, y,
                                   replace = False,
                                   n_samples = int(N*0.2),
                                   random_state = i)
    c = Counter(y_sample)
    print(sorted(c.items()))
```

- Change random seed when sampling without replacement

# Changing the Random seed

```
In [11]: for i in range(5):
```

```
    X_sample, y_sample = resample(X, y,
                                    replace = False,
                                    n_samples = int(N*0.2),
                                    random_state = i)
    c = Counter(y_sample)
    print(sorted(c.items()))
```

## ■ Output:

```
[("build wind float", 13), ("build wind non-float", 19), ("vehic wind float", 2), ('containers', 3), ('headlamps', 4), ('tableware', 1)]
[("build wind float", 10), ("build wind non-float", 21), ("vehic wind float", 3), ('containers', 2), ('headlamps', 5), ('tableware', 1)]
[("build wind float", 14), ("build wind non-float", 15), ("vehic wind float", 5), ('containers', 4), ('headlamps', 2), ('tableware', 2)]
[("build wind float", 16), ("build wind non-float", 12), ("vehic wind float", 2), ('containers', 5), ('headlamps', 5), ('tableware', 2)]
[("build wind float", 18), ("build wind non-float", 11), ("vehic wind float", 4), ('containers', 4), ('headlamps', 5)]
```

# Stratified Sampling

```
In [12]: X_sample, y_sample = resample(X, y,
                                     n_samples = int(N*0.2),
                                     random_state = 0,
                                     stratify=y)

c = Counter(y_sample)
print(sorted(c.items()))
```

- To apply stratified sampling, input the class label array to parameter 'stratify'

# Stratified Sampling

```
In [12]: X_sample, y_sample = resample(X, y,
                                     n_samples = int(N*0.2),
                                     random_state = 0,
                                     stratify=y)

c = Counter(y_sample)
print(sorted(c.items()))
```

## ■ Output:

```
[("build wind float", 14), ("build wind non-float", 15),
 ("vehic wind float", 3), ('containers', 2), ('headlamps', 6),
 ('tableware', 2)]
```

# Practice

---

- Write a code that runs stratified sampling without replacement with random seed varying from 0 to 4
- Print label counts of each sample

# Data Cube Aggregation

---

- The lowest level of a data cube (base cuboid)
  - The aggregated data for an **individual entity of interest**
  - E.g., a customer in a phone calling data warehouse
- Multiple levels of aggregation in data cubes
  - Further reduce the size of data to deal with
- Reference appropriate levels
  - Use the smallest representation which is enough to solve the task
- Queries regarding aggregated information should be answered using data cube, when possible

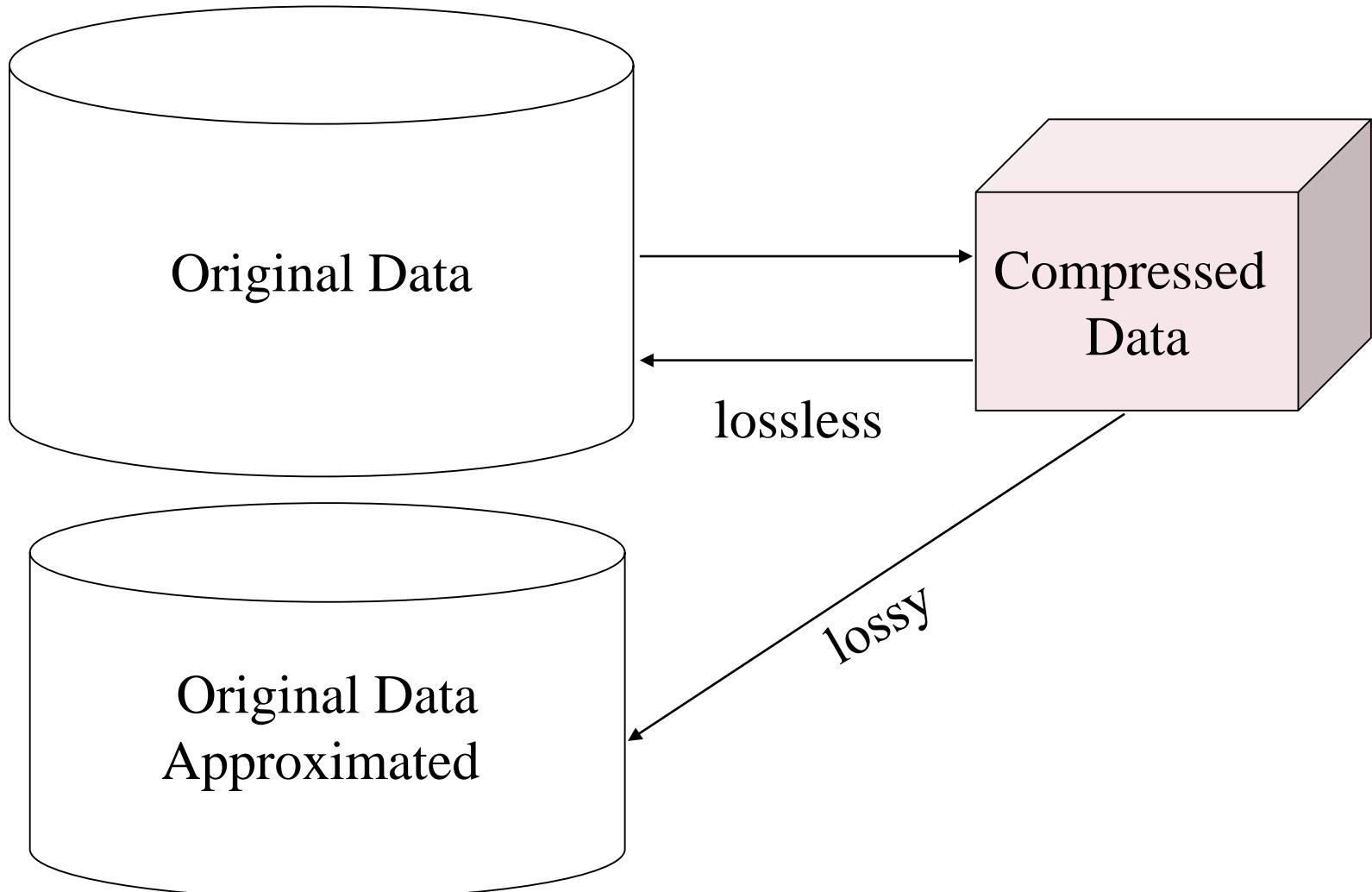
# Data Reduction 3: Data Compression

---

- String compression
  - There are extensive theories and well-tuned algorithms
  - Typically lossless, but only limited manipulation is possible without expansion
- Audio/video compression
  - Typically lossy compression, with progressive refinement
  - Sometimes small fragments of signal can be reconstructed without reconstructing the whole
- Time sequence is not audio
  - Typically short and vary slowly with time
- Dimensionality and numerosity reduction may also be considered as forms of data compression

# Data Compression

---



# Chapter 3: Data Preprocessing

---

- Data Preprocessing: An Overview
  - Data Quality
  - Major Tasks in Data Preprocessing
- Data Cleaning
- Data Integration
- Data Reduction
- Data Transformation and Data Discretization
- Summary



# Data Transformation

---

- A function that maps the entire set of values of a given attribute to a new set of replacement values s.t. each old value can be identified with one of the new values
- Methods
  - Smoothing: Remove noise from data
  - Attribute/feature construction
    - New attributes constructed from the given ones
  - Aggregation: Summarization, data cube construction
  - Normalization: Scaled to fall within a smaller, specified range
    - min-max normalization
    - z-score normalization
    - normalization by decimal scaling
  - Discretization: Concept hierarchy climbing

# Normalization

- **Min-max normalization:** to  $[new\_min_A, new\_max_A]$

$$v' = \frac{v - min_A}{max_A - min_A} (new\_max_A - new\_min_A) + new\_min_A$$

- e.g., Let income range \$12,000 to \$98,000 normalized to [0.0, 1.0]. Then \$73,000 is mapped to  $\frac{73,600 - 12,000}{98,000 - 12,000} (1.0 - 0) + 0 = 0.716$

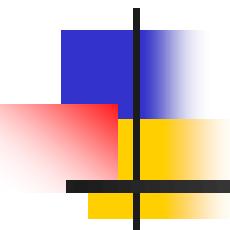
- **Z-score normalization** ( $\mu$ : mean,  $\sigma$ : standard deviation):

$$v' = \frac{v - \mu_A}{\sigma_A}$$

- e.g., Let  $\mu = 54,000$ ,  $\sigma = 16,000$ . Then  $\frac{73,600 - 54,000}{16,000} = 1.225$

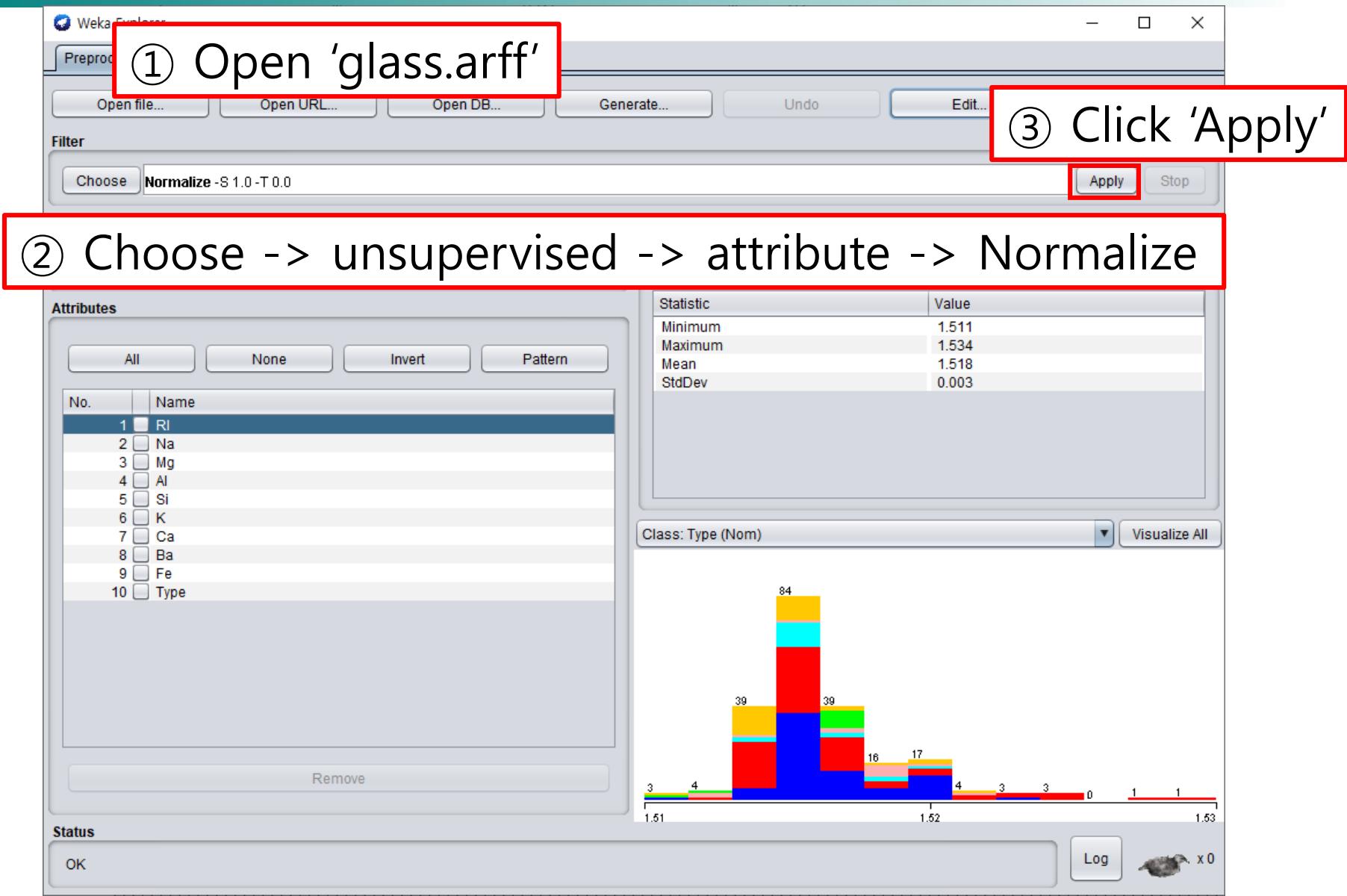
- **Normalization by decimal scaling**

$$v' = \frac{v}{10^j} \quad \text{Where } j \text{ is the smallest integer such that } \text{Max}(|v'|) < 1$$



# **Weka - Normalization**

# Min-max Normalization with Weka



# Min-max Normalization with Weka

All numeric values are normalized into [0,1]

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Open file... Open URL... Open DB... Generate... Undo Edit... Save...

Filter

Choose Normalize -S 1.0 -T 0.0 Apply Stop

Attributes

No. Name

1	RI
2	Na
3	Mg
4	Al
5	Si
6	K
7	Ca
8	Ba
9	Fe
10	Type

Remove

Statistic Value

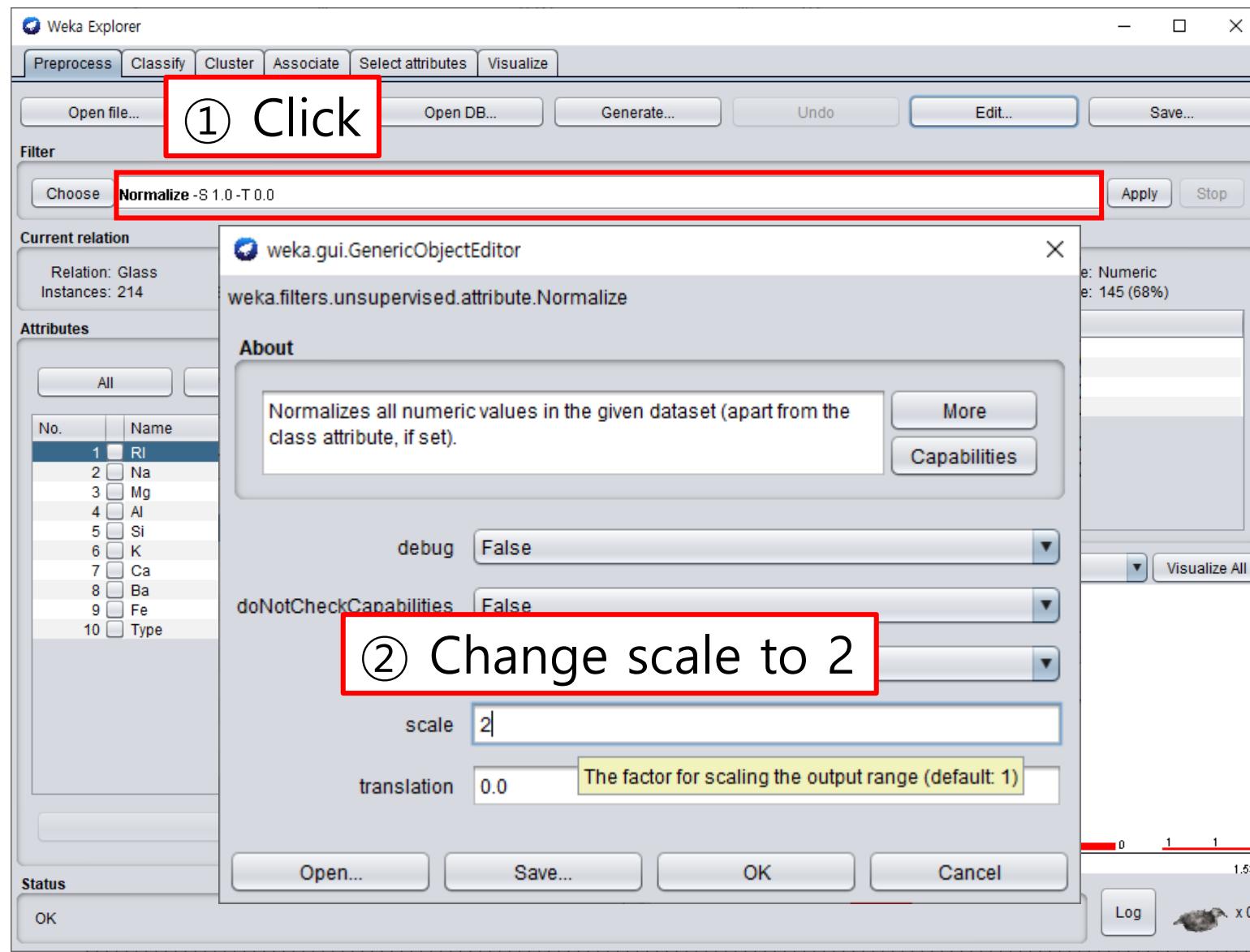
Minimum	0
Maximum	1
Mean	0.317
StdDev	0.133

Class: Type (Nom) Visualize All

Status

OK Log X 0

# Min-max Normalization with Weka



# Min-max Normalization with Weka

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Open file... Open URL... Open DB... Generate... Undo Edit... Save...

Filter

Choose Normalize -S 2.0 -T 0.0 Apply Stop

Current relation

Relation: Glass Instances: 214

Selected attribute

Now, the values are normalized into [0,2]

Attributes

All None Invert Pattern

No.	Name
1	RI
2	Na
3	Mg
4	Al
5	Si
6	K
7	Ca
8	Ba
9	Fe
10	Type

Remove

Status

OK Log x 0

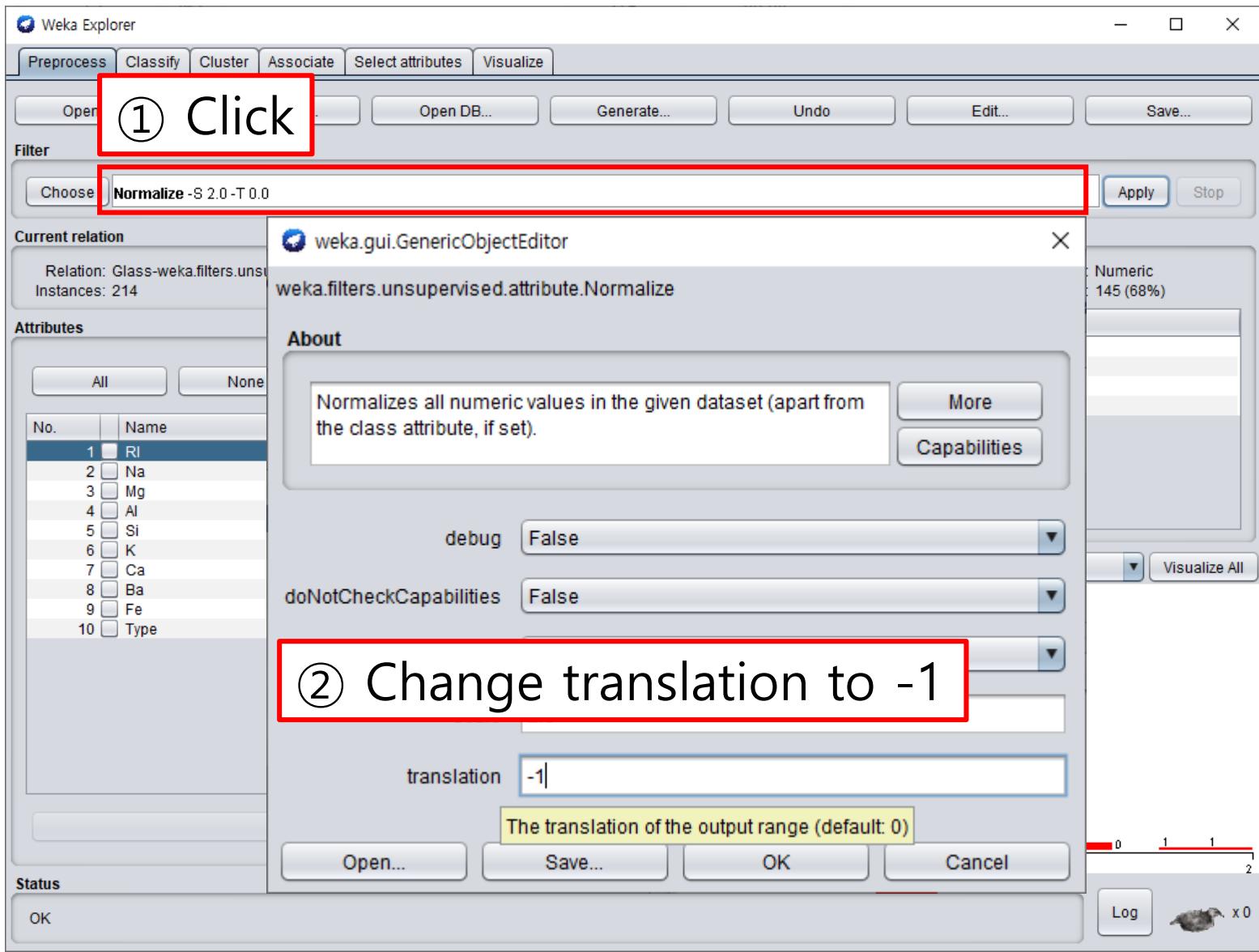
Statistic | Value

Statistic	Value
Minimum	0
Maximum	2
Mean	0.633
StdDev	0.267

Class: Type (Nom) Visualize All

Log x 0

# Min-max Normalization with Weka



# Min-max Normalization with Weka

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Open file... Open URL... Open DB... Generate... Undo Edit... Save...

Filter

Choose Normalize -S 2.0 -T -1.0 Apply Stop

Current relation

Relation: Glass-weka fi  
Instances: 214 Sum of weights: 214 Missing: 0 (0%) Distinct: 178 Unique: 145 (68%)

Attributes

All None Invert Pattern

No.	Name
1	RI
2	Na
3	Mg
4	Al
5	Si
6	K
7	Ca
8	Ba
9	Fe
10	Type

Remove

Status

OK Log x 0

Now, the values are normalized into [-1,1]

Statistic	Value
Minimum	-1
Maximum	1
Mean	-0.367
StdDev	0.267

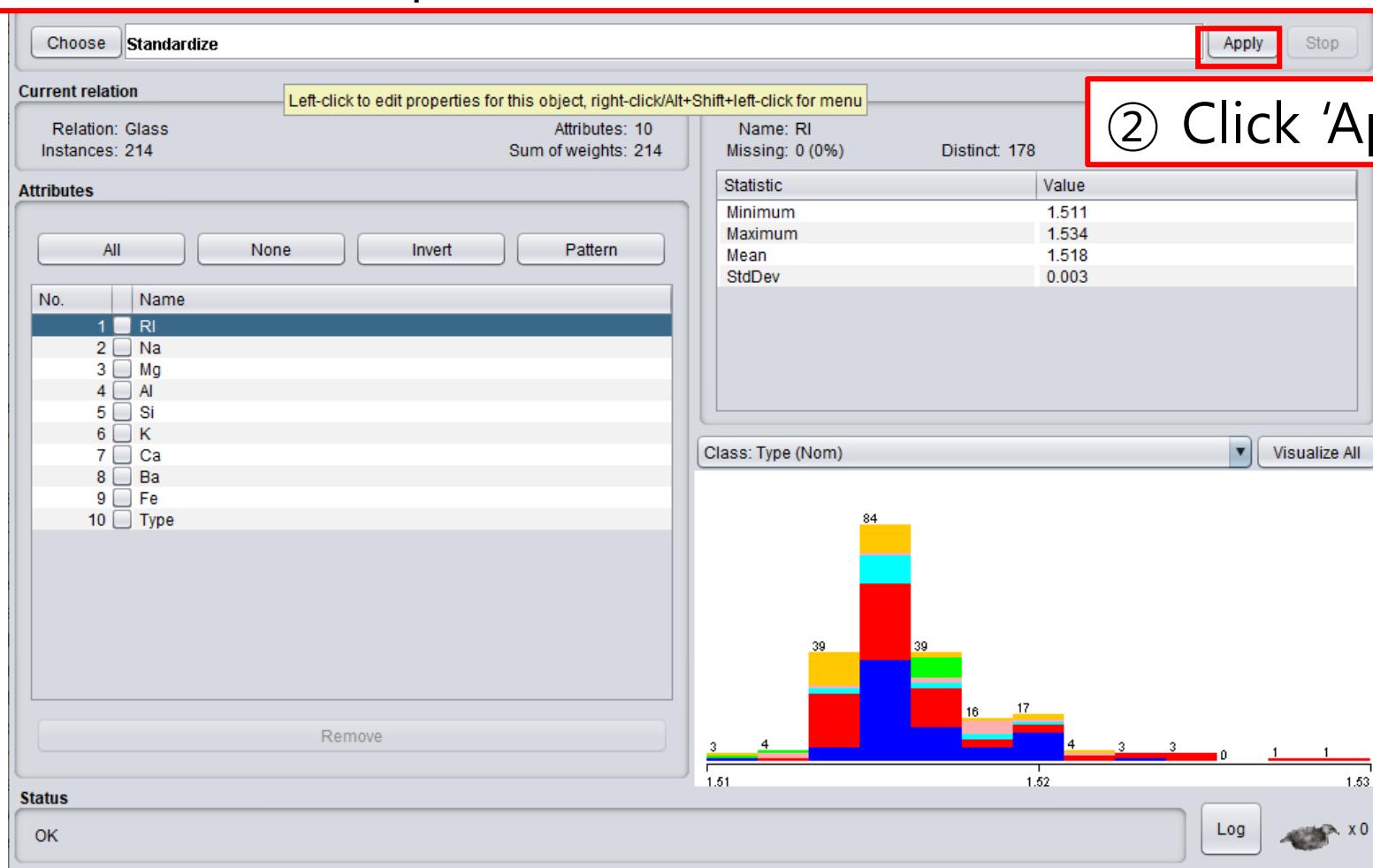
Class: Type (Nom) Visualize All

The chart visualizes the normalized data for each attribute. The x-axis represents the normalized value from -1 to 1. The y-axis represents the attribute index from 1 to 10. The z-axis represents the count of instances. The bars are colored by attribute: RI (blue), Na (orange), Mg (yellow), Al (red), Si (cyan), K (green), Ca (light blue), Ba (pink), Fe (purple), and Type (dark blue). The counts for each bar are labeled on top: RI has 84, Na has 39, Mg has 39, Al has 16, Si has 17, K has 4, Ca has 3, Ba has 3, Fe has 0, and Type has 1.

# Z-score Normalization with Weka



① Choose -> unsupervised -> attribute -> Standardize



② Click 'Apply'

# Z-score Normalization with Weka

The mean value becomes 0,  
and the standard deviation becomes 1

The screenshot shows the Weka Explorer interface with a red box highlighting the text "The mean value becomes 0, and the standard deviation becomes 1". The interface includes a toolbar with buttons for Preprocess, Classify, Cluster, Associate, Select attributes, and Visualize. Below the toolbar is a row of buttons for Open file..., Open URL..., Open DB..., Generate..., Undo, Edit..., and Save... A "Filter" section allows choosing between "Choose" and "Stan". The "Current relation" section shows "Relation: Glas" and "Instances: 214". The "Attributes" section lists attributes RI through Type, with RI selected. The "Statistics" section displays a table of minimum, maximum, mean (-0), and standard deviation (1). The "Visualize All" section shows a 3D bar chart of the data.

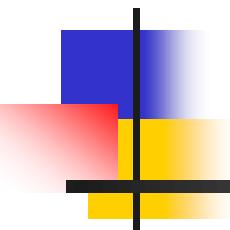
Statistic	Value
Minimum	-2.376
Maximum	5.125
Mean	-0
StdDev	1

Class: Type (Nom) Visualize All

RI Na Mg Al Si K Ca Ba Fe Type

Remove

Status OK Log x 0



# Python - Normalization

# Min-max Normalization

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv('glass.csv')
X = df.values[:, :-1]
y = df.values[:, -1]

normalizer = MinMaxScaler()
X_norm = normalizer.fit_transform(X)
X_norm[:10]
```

```
array([[0.2976295 , 0.30977444, 0.77951002, 0.25856698, 0.575 ,
       0.10305958, 0.31040892, 0.          , 0.          ],
      [0.23178227, 0.21503759, 0.78396437, 0.33021807, 0.55 ,
       0.09178744, 0.28810409, 0.          , 0.          ],
      [0.2976295 , 0.37293233, 0.77505568, 0.34890966, 0.50535714,
       0.09500805, 0.27881041, 0.          , 0.          ],
      [0.08077261, 0.5518797 , 0.38752784, 0.3894081 , 0.84642857,
       0.          , 0.20074349, 0.          , 0.          ],
      [1.          , 0.23609023, 0.          , 0.2211838 , 0.0625 ,
       0.01932367, 1.          , 0.          , 0.47058824],
      [0.23705004, 0.3037594 , 0.63474388, 0.35825545, 0.61785714,
       0.09178744, 0.31226766, 0.03492063, 0.43137255],
      [0.29148376, 0.43759398, 0.81291759, 0.11214953, 0.56964286,
       0.00966184, 0.32527881, 0.          , 0.          ],
      [0.31694469, 0.36240602, 0.6325167 , 0.30841121, 0.54285714,
       0.08856683, 0.33828996, 0.          , 0.          ],
      [0.18876207, 0.51278195, 0.          , 0.74454829, 0.63928571,
       0.01288245, 0.33828996, 0.19365079, 0.09803922],
      [0.29587357, 0.36992481, 0.86859688, 0.31464174, 0.45 ,
       0.08856683, 0.27973978, 0.          , 0.54901961]])
```

# Min-max Normalization

```
normalizer = MinMaxScaler(feature_range=(0, 100))  
X_norm = normalizer.fit_transform(X)  
  
print(normalizer.scale_)  
print(normalizer.data_min_)  
print(normalizer.data_max_)  
print(normalizer.data_range_)
```

- Parameter
  - feature\_range = (min, max) : Desired range of transformed data
- Attributes
  - scale\_ : relative scaling of each feature
  - data\_min\_ : the minimum value of the original data
  - data\_max\_ : the maximum value of the original data
  - data\_range\_ : (data\_max\_ - data\_min\_)

# Z-score Normalization

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('glass.csv')
X = df.values[:, :-1]
y = df.values[:, -1]

normalizer = StandardScaler()
X_norm = normalizer.fit_transform(X)
X_norm[:10]
```

```
array([[-1.43714541e-01, -7.58384063e-01,  5.66676959e-01,
       -6.52289486e-01,  4.90550986e-01,  2.19688551e-01,
       -1.31680092e-01, -3.52876828e-01, -5.86450902e-01],
      [-6.38803268e-01, -1.53168114e+00,  5.80575173e-01,
       -1.90536451e-01,  3.09376090e-01,  1.12106515e-01,
       -3.00715072e-01, -3.52876828e-01, -5.86450902e-01],
      [-1.43714541e-01, -2.42852679e-01,  5.52778744e-01,
       -7.00791379e-02, -1.41505092e-02,  1.42844239e-01,
       -3.71146313e-01, -3.52876828e-01, -5.86450902e-01],
      [-1.77420675e+00,  1.21781957e+00, -6.56365903e-01,
       1.90911708e-01,  2.45759271e+00, -7.63918639e-01,
       -9.62768741e-01, -3.52876828e-01, -5.86450902e-01],
      [ 5.13723188e+00, -1.35983734e+00, -1.86551055e+00,
       -8.93204112e-01, -3.22353438e+00, -5.79492291e-01,
       5.09431802e+00, -3.52876828e-01,  1.88241125e+00],
      [-5.99196170e-01, -8.07482290e-01,  1.14984993e-01,
       -9.85048123e-03,  8.01136522e-01,  1.12106515e-01,
       -1.17593844e-01, -1.31127748e-01,  1.67667274e+00],
      [-1.89922822e-01,  2.84953261e-01,  6.70913566e-01,
       -1.59587177e+00,  4.51727794e-01, -6.71705465e-01,
       -1.89901058e-02, -3.52876828e-01, -5.86450902e-01],
      [ 1.51148583e-03, -3.28774576e-01,  1.08035886e-01,
       -3.31069983e-01,  2.57611835e-01,  8.13687898e-02,
       7.96136323e-02, -3.52876828e-01, -5.86450902e-01],
      [-9.62261237e-01,  8.98681099e-01, -1.86551055e+00,
       2.47960066e+00,  9.56429290e-01, -6.40967740e-01,
       7.96136323e-02,  8.76822615e-01, -7.21046191e-02],
      [-1.56916907e-01, -2.67401792e-01,  8.44641245e-01,
       -2.90917546e-01, -4.15323493e-01,  8.13687898e-02,
       -3.64103189e-01, -3.52876828e-01,  2.29388828e+00]])
```

# Z-score Normalization

```
normalizer = StandardScaler()  
X_norm = normalizer.fit_transform(X)  
  
print(normalizer.scale_)  
print(normalizer.mean_)  
print(normalizer.var_)
```

## ■ Attributes

- scale\_ : relative scaling of each feature
- mean\_ : the mean value of the original data
- var\_ : the variance of the original data

# Practice

---

- Using MinMaxScaler, change the scale of the transformed data with a variety of ranges

# Discretization

---

- Three types of attributes
  - Nominal—values from an unordered set, e.g., color, profession
  - Ordinal—values from an ordered set, e.g., military or academic rank
  - Numeric—real numbers, e.g., integer or real numbers
- Discretization: Divide the range of a continuous attribute into intervals
  - Interval labels can then be used to replace actual data values
  - Reduce data size by discretization
  - Supervised vs. unsupervised
  - Split (top-down) vs. merge (bottom-up)
  - Discretization can be performed recursively on an attribute
  - Prepare for further analysis, e.g., classification

# Data Discretization Methods

---

- Typical methods: All the methods can be applied recursively
  - Binning
    - Top-down split, unsupervised
  - Histogram analysis
    - Top-down split, unsupervised
  - Clustering analysis (unsupervised, top-down split or bottom-up merge)
  - Decision-tree analysis (supervised, top-down split)
  - Correlation (e.g.,  $\chi^2$ ) analysis (unsupervised, bottom-up merge)

# Simple Discretization: Binning

---

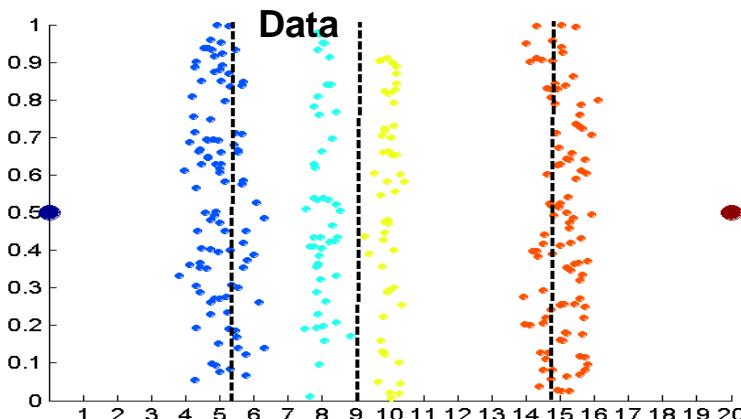
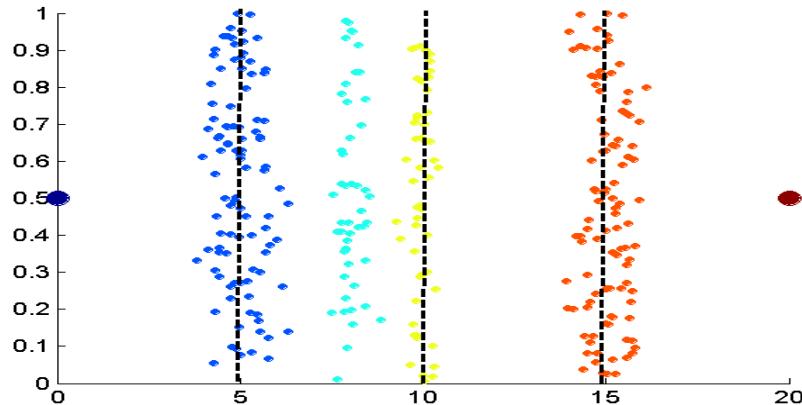
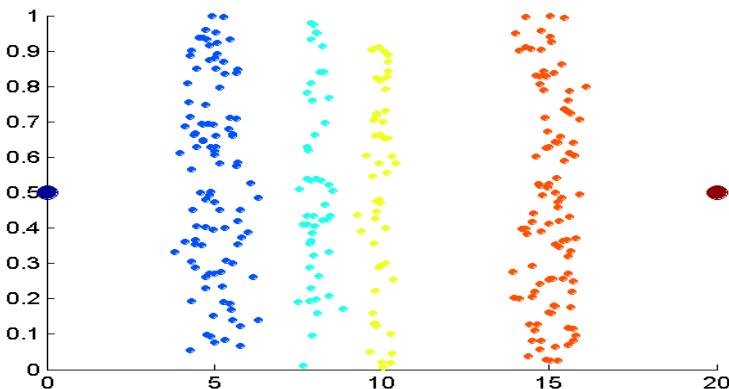
- Equal-width (distance) partitioning
  - Divides the range into  $N$  intervals of equal size: uniform grid
  - if  $A$  and  $B$  are the lowest and highest values of the attribute, the width of intervals will be:  $W = (B - A)/N$ .
  - The most straightforward, but outliers may dominate presentation
  - 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

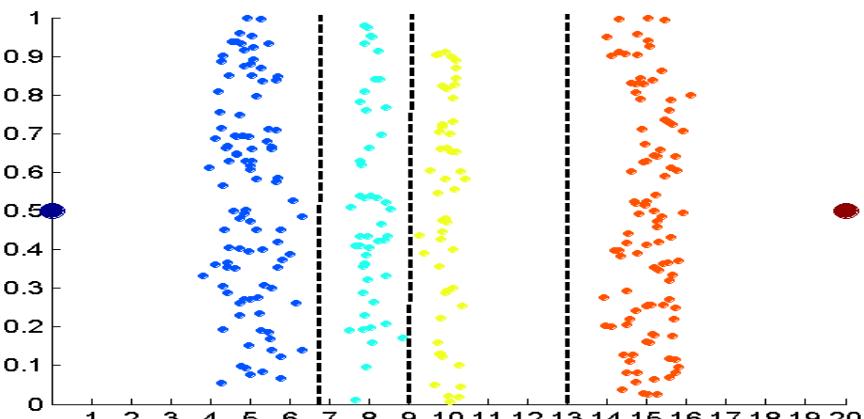
---

- 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
  - \* Smoothing by **bin boundaries**:
    - Bin 1: 4, 4, 4, 15
    - Bin 2: 21, 21, 25, 25
    - Bin 3: 26, 26, 26, 34

# Discretization Without Using Class Labels (Binning vs. Clustering)



Equal frequency (binning)



K-means clustering leads to better results

# Discretization by Classification & Correlation Analysis

---

- Classification (e.g., decision tree analysis)
  - Supervised: Given class labels, e.g., cancerous vs. benign
  - Using *entropy* to determine split point (discretization point)
  - Top-down, recursive split
  - Details to be covered in Chapter 7
- Correlation analysis (e.g., Chi-merge:  $\chi^2$ -based discretization)
  - Supervised: use class information
  - Bottom-up merge: find the best neighboring intervals (those having similar distributions of classes, i.e., low  $\chi^2$  values) to merge
  - Merge performed recursively, until a predefined stopping condition

# Concept Hierarchy Generation

---

- **Concept hierarchy** organizes concepts (i.e., attribute values) hierarchically and is usually associated with each dimension in a data warehouse
- Concept hierarchies facilitate drilling and rolling in data warehouses to view data in multiple granularity
- 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.

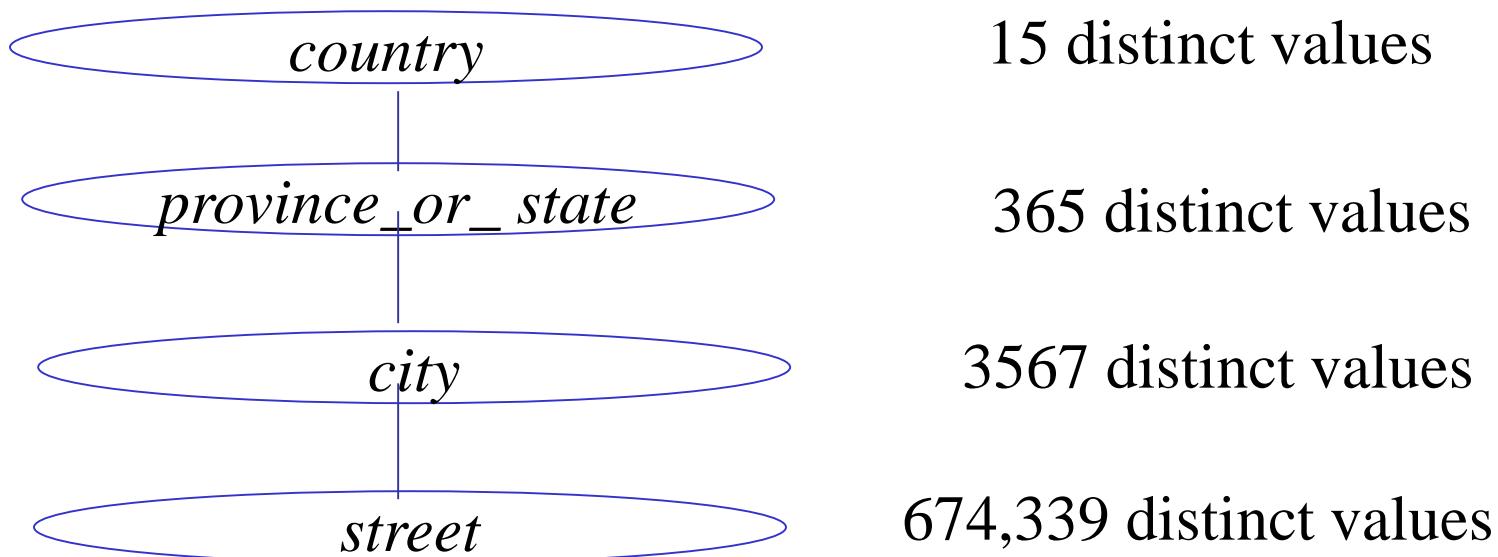
# Concept Hierarchy Generation for Nominal Data

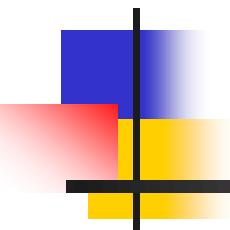
---

- Specification of a partial/total ordering of attributes explicitly at the schema level by users or experts
  - $\textit{street} < \textit{city} < \textit{state} < \textit{country}$
- Specification of a hierarchy for a set of values by explicit data grouping
  - $\{\text{Urbana, Champaign, Chicago}\} < \text{Illinois}$
- Specification of only a partial set of attributes
  - E.g., only  $\textit{street} < \textit{city}$ , 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:  $\{\textit{street}, \textit{city}, \textit{state}, \textit{country}\}$

# 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

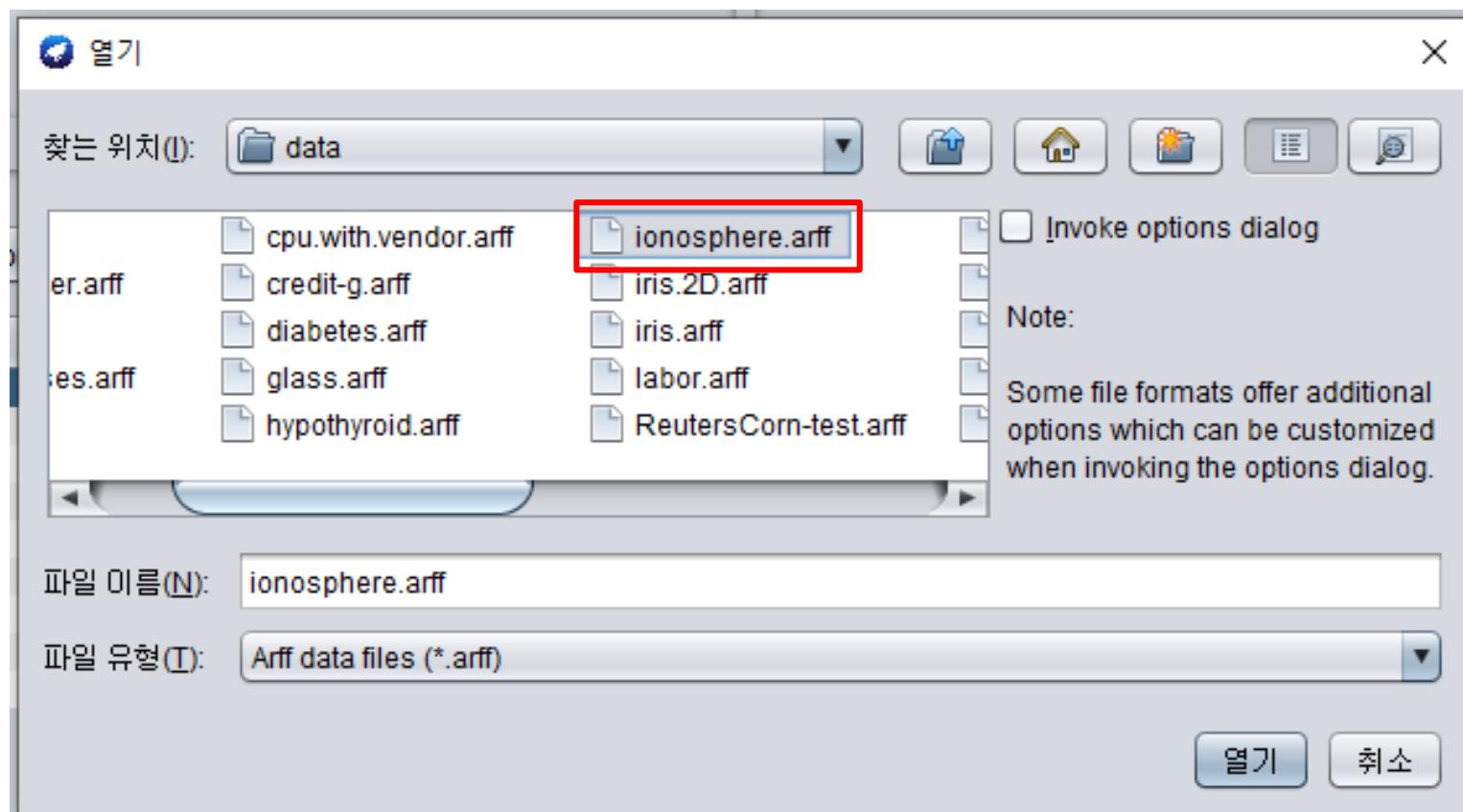




# Weka - Discretization

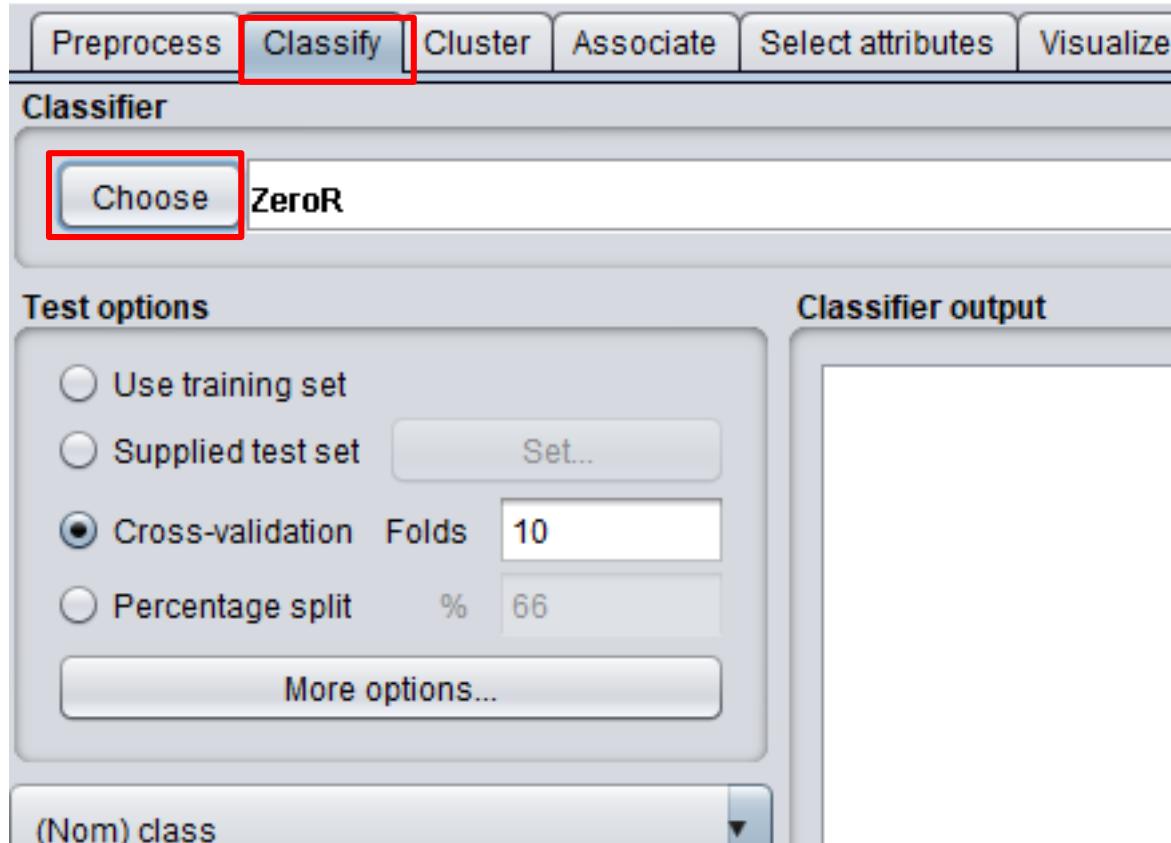
# Discretizing numeric attributes

- Open 'ionosphere.arff'



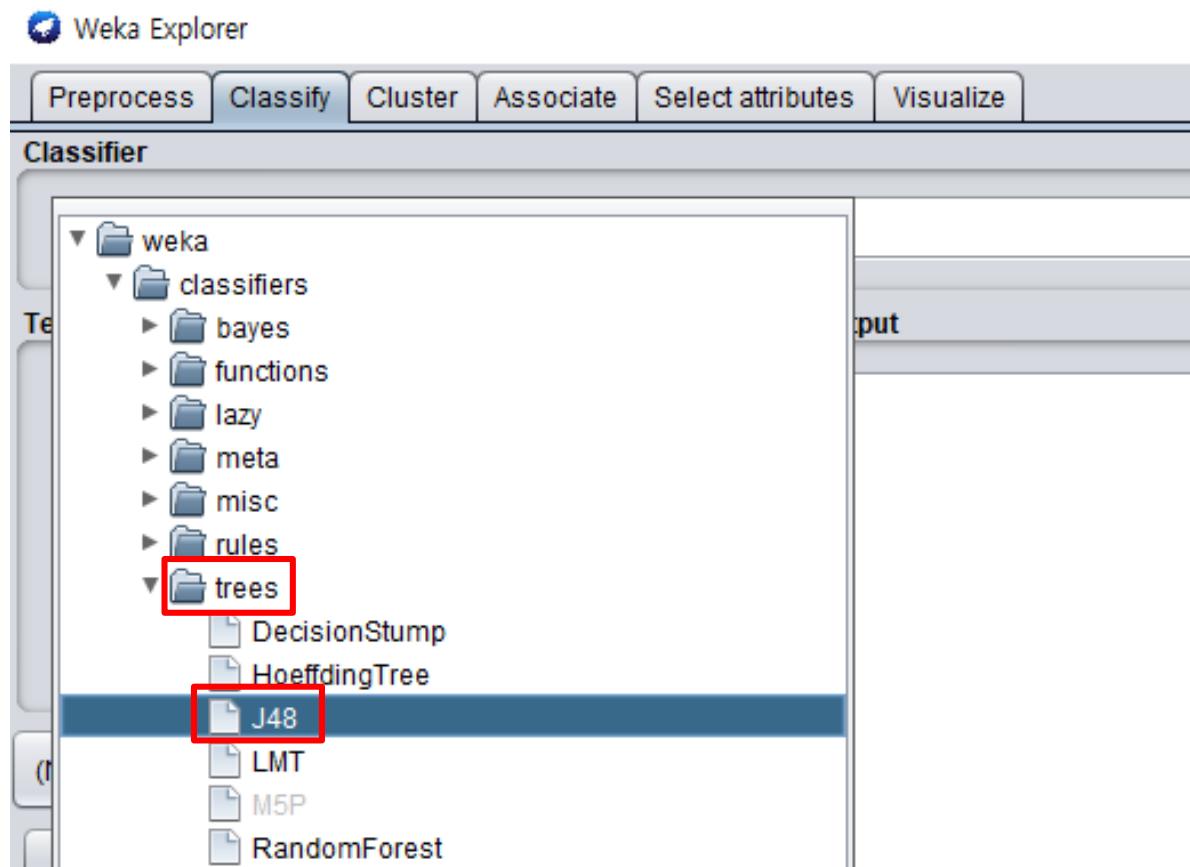
# Discretizing numeric attributes

- First we will check the classification result of raw data
- From Classify tab, click 'choose'



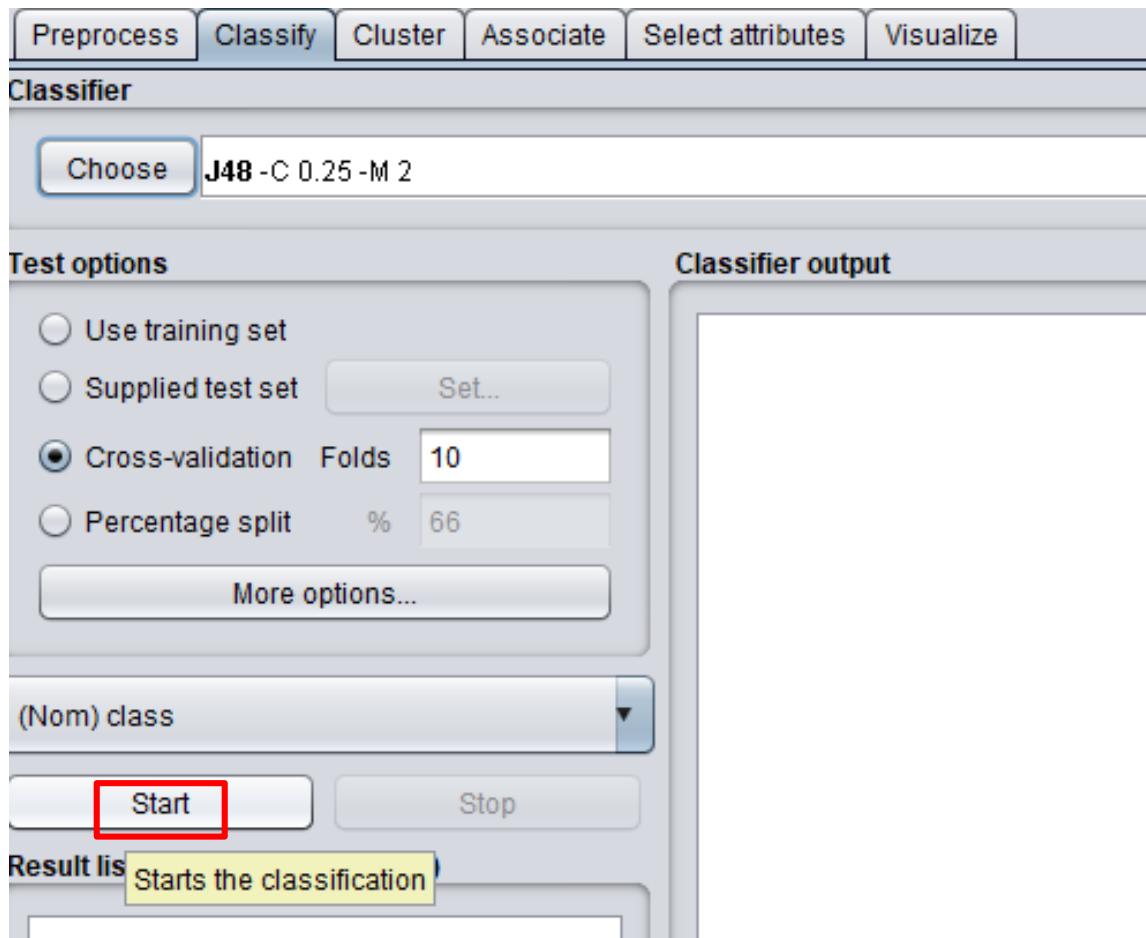
# Discretizing numeric attributes

- Select trees->J48



# Discretizing numeric attributes

- Run the classifier



# Discretizing numeric attributes

- The accuracy is 91.453%

Preprocess Classify Cluster Associate Select attributes Visualize

Classifier

Choose J48 -C 0.25 -M 2

Test options

Use training set  
 Supplied test set Set...  
 Cross-validation Folds 10  
 Percentage split % 66  
More options...

(Nom) class ▾

Start Stop

Result list (right-click for options)

19:33:37 - trees.J48

Classifier output

```
Time taken to build model: 0.01 seconds

==== Stratified cross-validation ====
==== Summary ===

Correctly Classified Instances      321          91.453 %
Incorrectly Classified Instances   30           8.547 %
Kappa statistic                   0.8096
Mean absolute error               0.0938
Root mean squared error           0.2901
Relative absolute error            20.36 %
Root relative squared error       60.4599 %
Total Number of Instances         351

==== Detailed Accuracy By Class ===
```

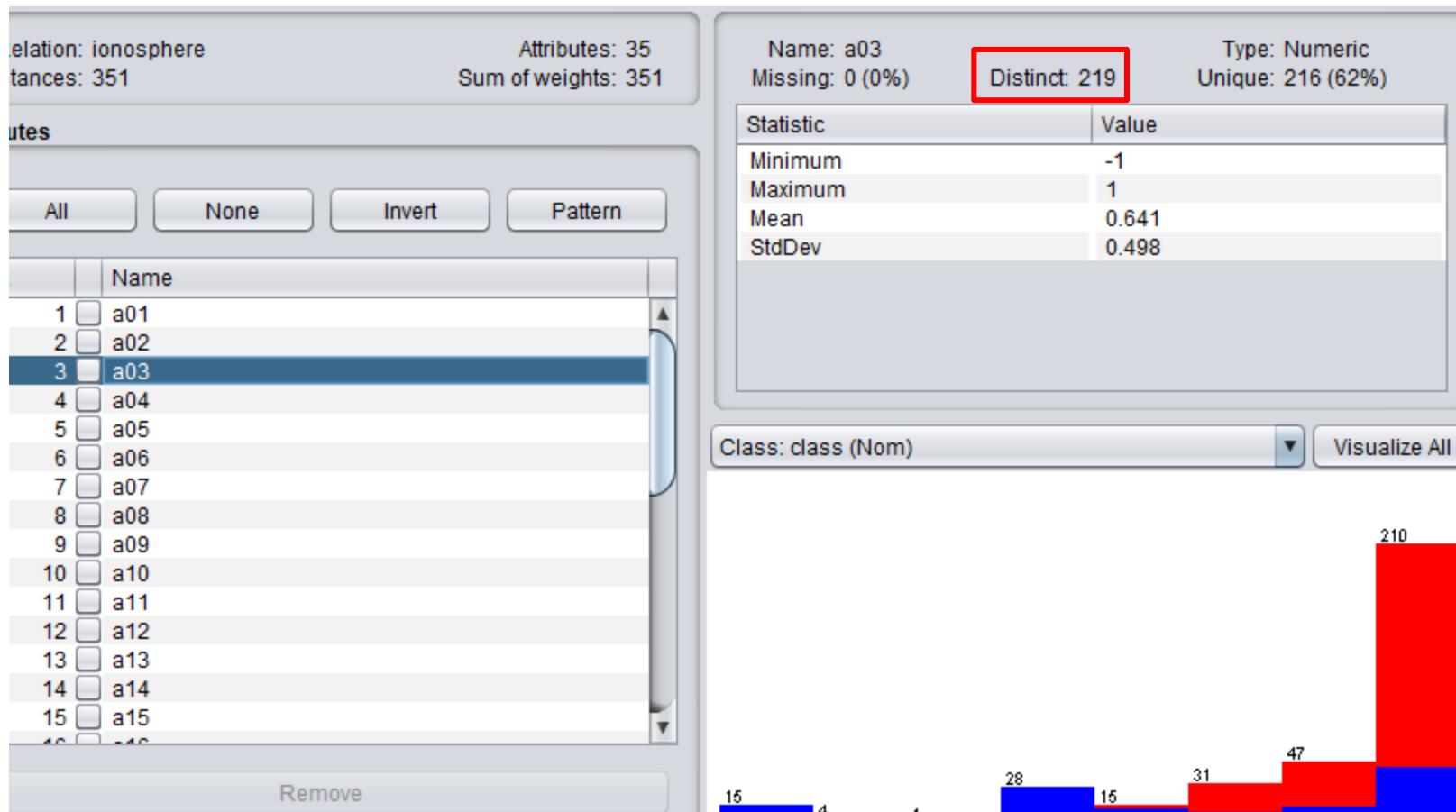
# Discretizing numeric attributes

- Let's discretize numeric attributes and see how the result differs
- Go back to 'Preprocess' tab



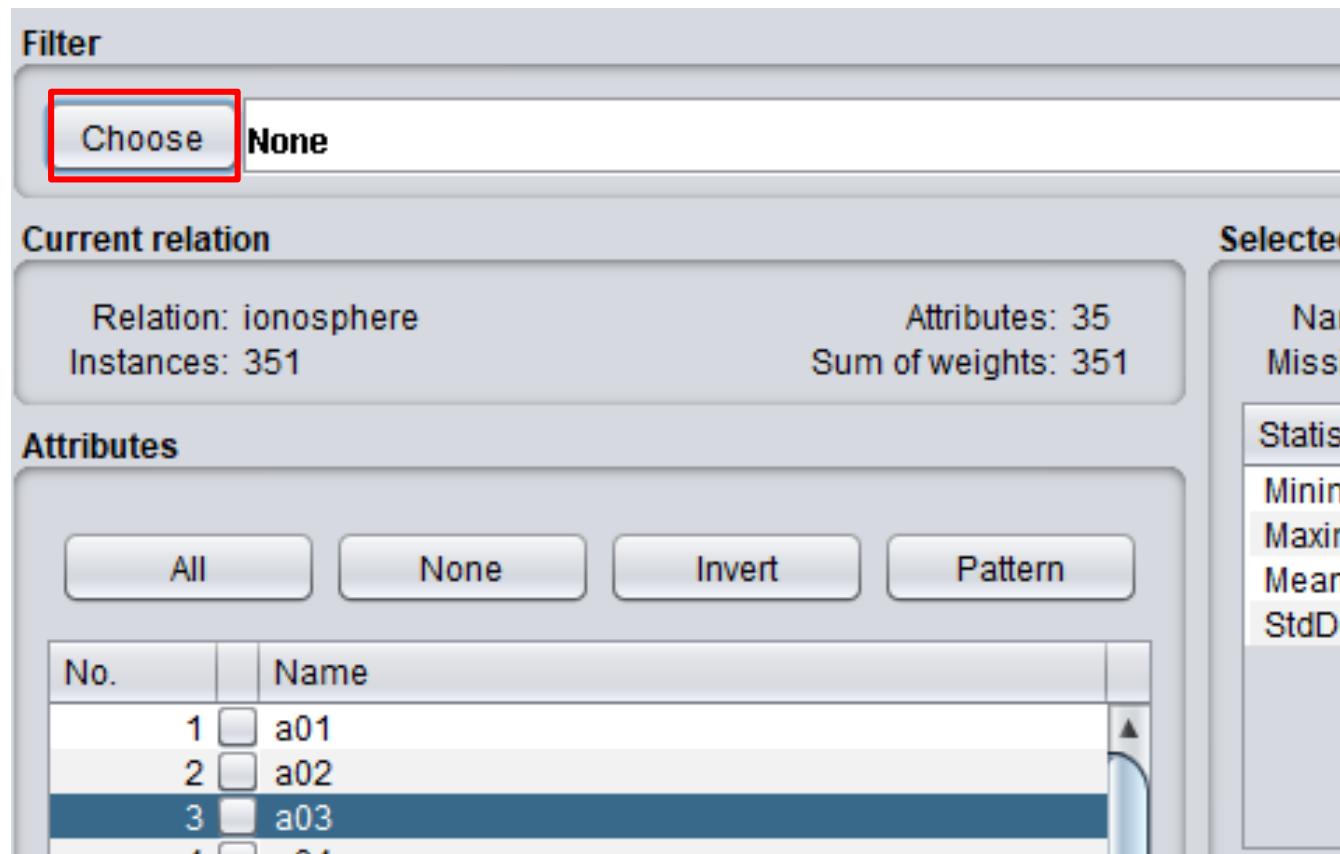
# Discretizing numeric attributes

- You can see that most attributes are continuous



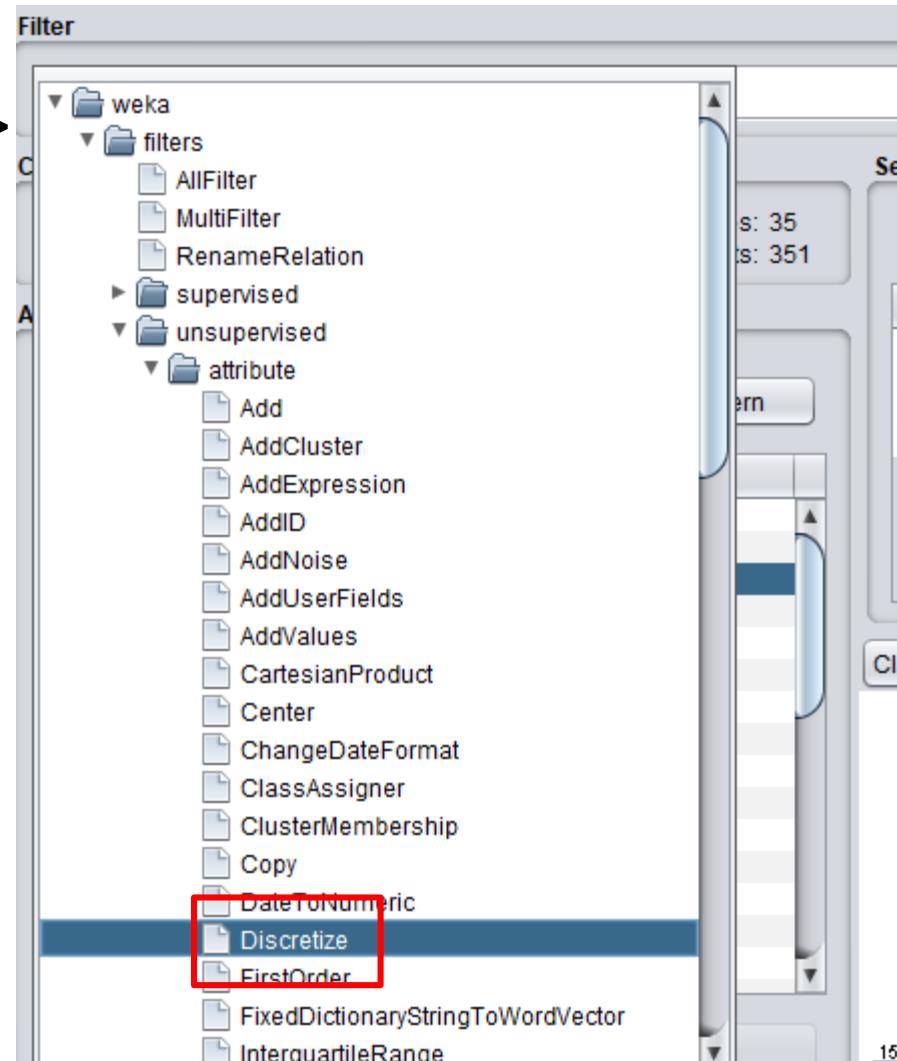
# Discretizing numeric attributes

- Now let's discretize attributes
- Click 'Choose' in 'Filter'



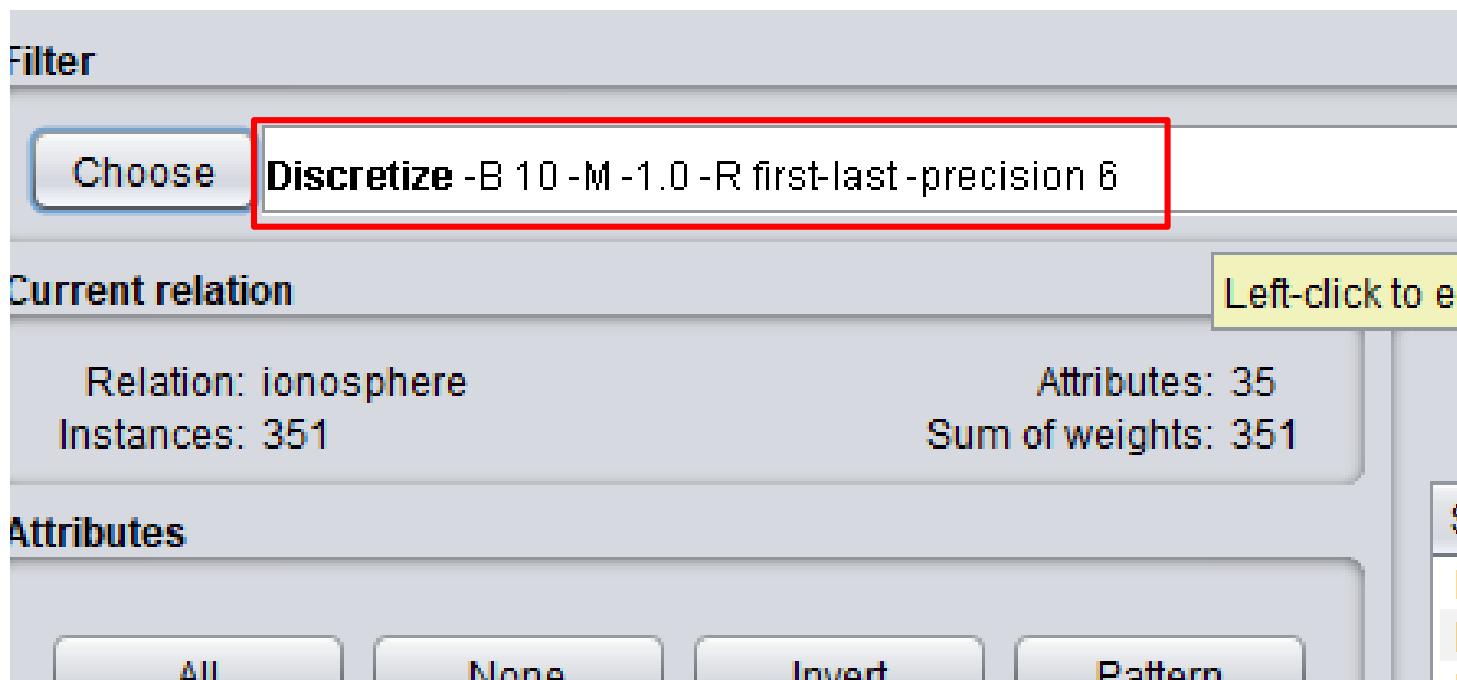
# Discretizing numeric attributes

- Select filters-> unsupervised->attribute-> Discretize



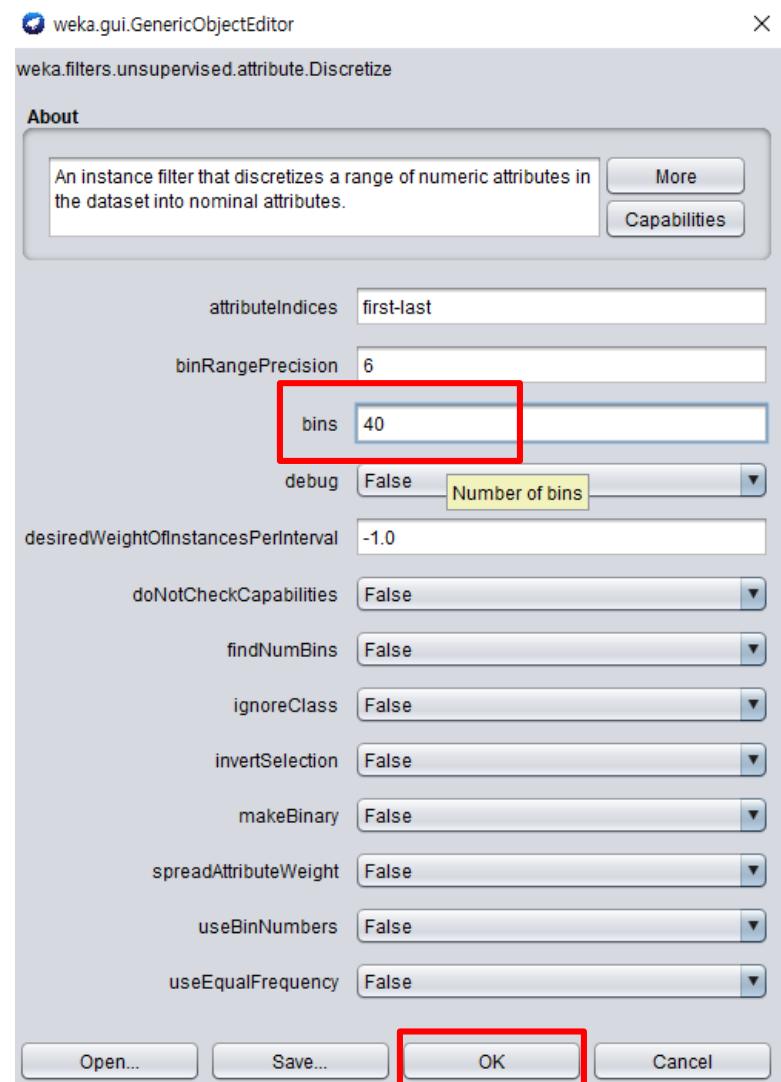
# Discretizing numeric attributes

- Click the white bar to adjust parameters



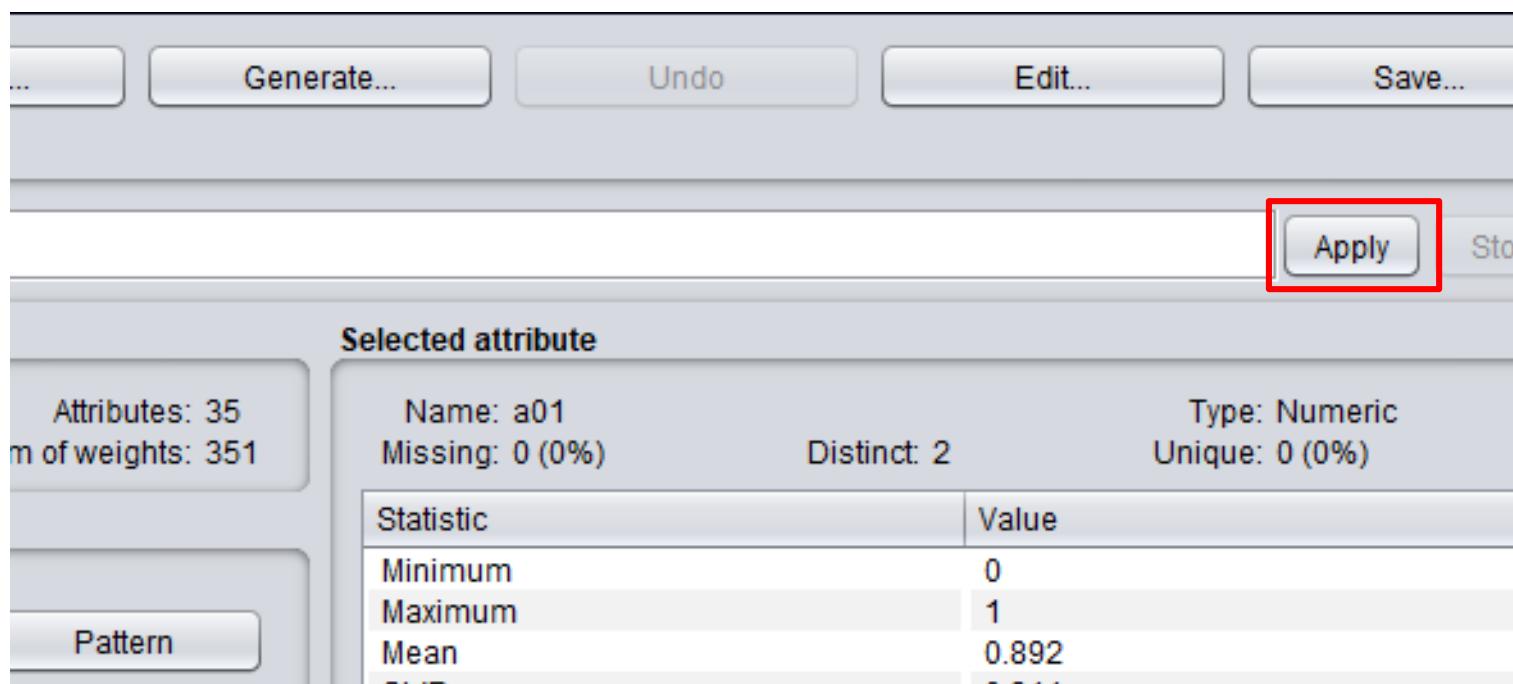
# Discretizing numeric attributes

- Change the bins to 40 and click 'OK'



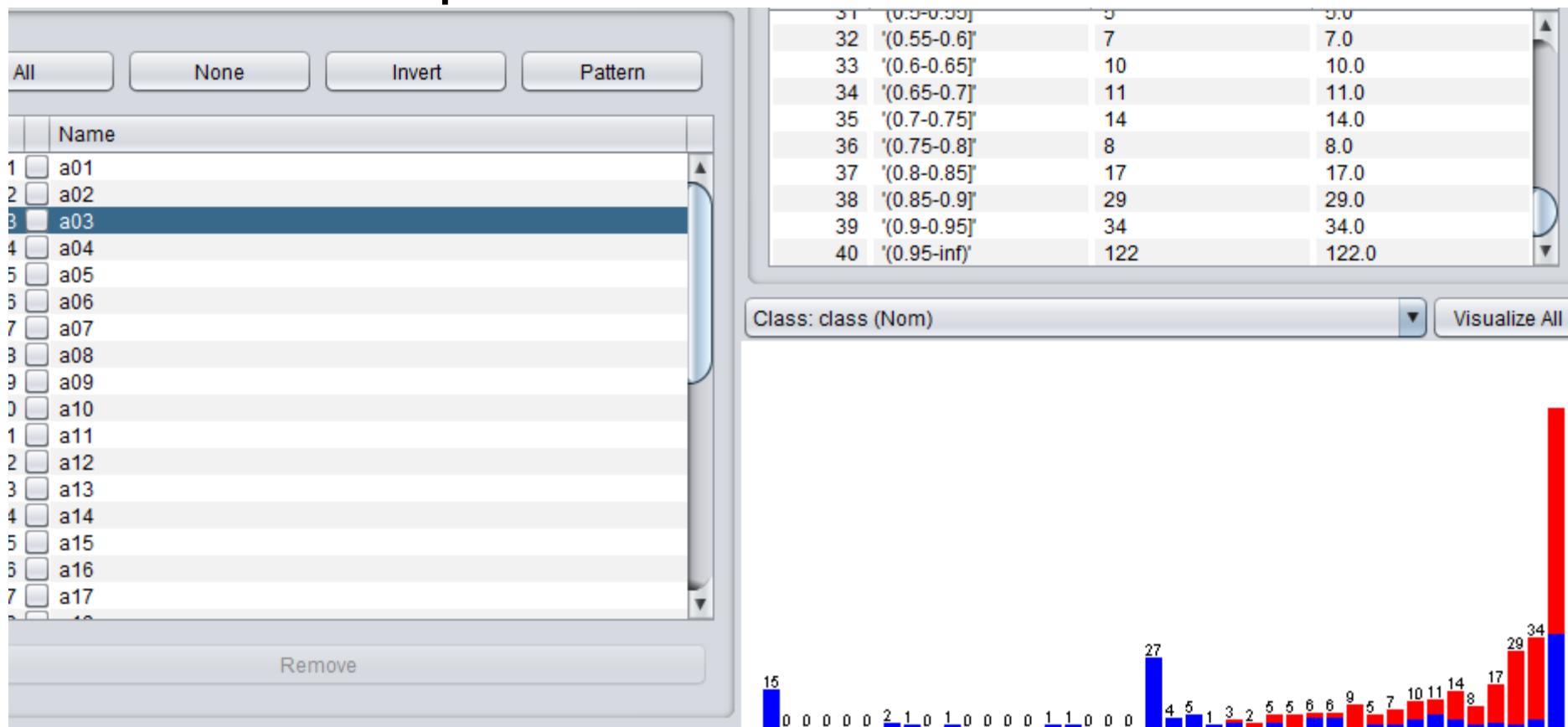
# Discretizing numeric attributes

- Click 'Apply'



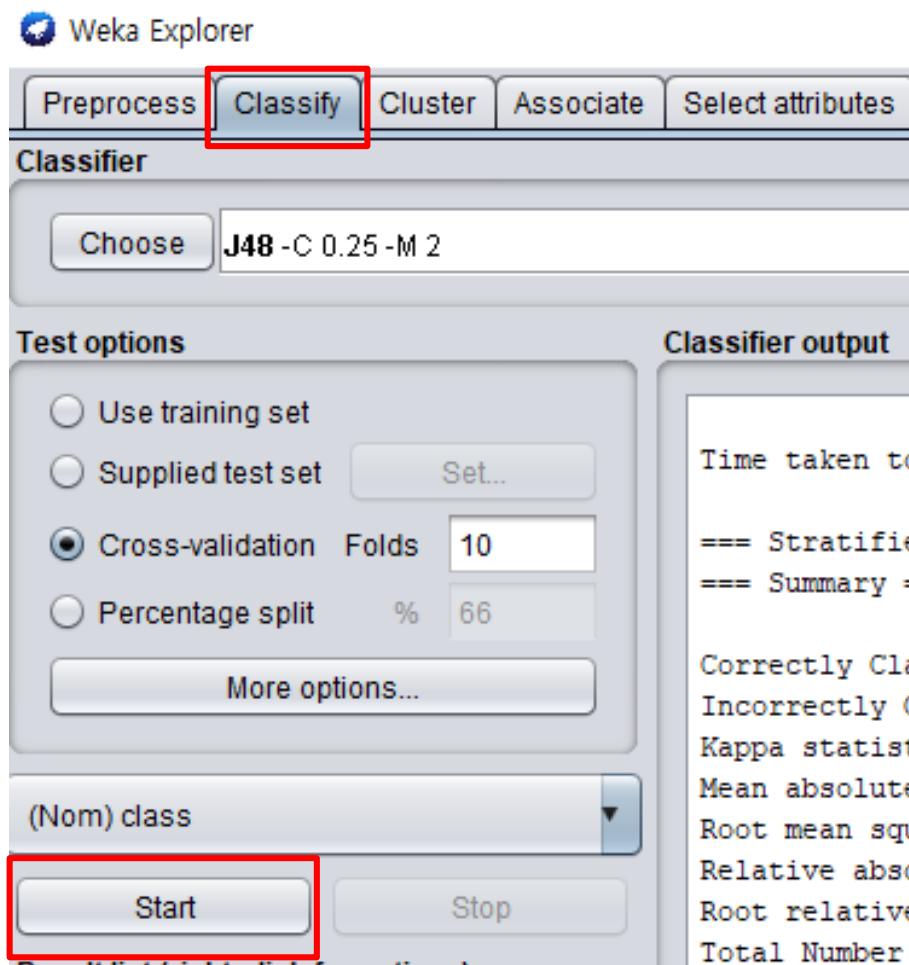
# Discretizing numeric attributes

- You can see that the data is discretized into 40 bins with equi-width



# Discretizing numeric attributes

- Run the classifier again



# Discretizing numeric attributes

- This time the accuracy is little bit lower

```
Time taken to build model: 0.01 seconds

==== Stratified cross-validation ====
==== Summary ====



|                                  |           |           |
|----------------------------------|-----------|-----------|
| Correctly Classified Instances   | 308       | 87.7493 % |
| Incorrectly Classified Instances | 43        | 12.2507 % |
| Kappa statistic                  | 0.7276    |           |
| Mean absolute error              | 0.1629    |           |
| Root mean squared error          | 0.3242    |           |
| Relative absolute error          | 35.3662 % |           |
| Root relative squared error      | 67.574 %  |           |
| Total Number of Instances        | 351       |           |



==== Detailed Accuracy By Class ====
```

# Discretizing numeric attributes

- Scroll up to check the size of the tree

```
| a06 = '(0.8-0.85]': b (1.0)
| a06 = '(0.85-0.9]': g (0.0)
| a06 = '(0.9-0.95]': g (0.0)
| a06 = '(0.95-inf)': b (17.0/2.0)
```

```
Number of Leaves : 79
Size of the tree : 81
```

```
Time taken to build model: 0 seconds
```

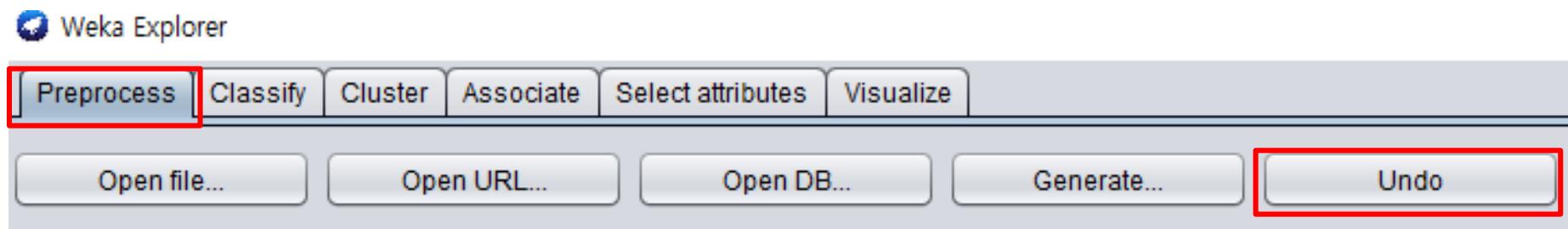
```
==== Stratified cross-validation ===
```

```
==== Summary ===
```

Correctly Classified Instances	308	87.7493 %
Incorrectly Classified Instances	43	12.2507 %
Kappa statistic	0.7276	
Mean absolute error	0.1629	
Root mean squared error	0.3242	
Relative absolute error	35.3662 %	
Root relative squared error	67.574 %	
Total Number of Instances	351	

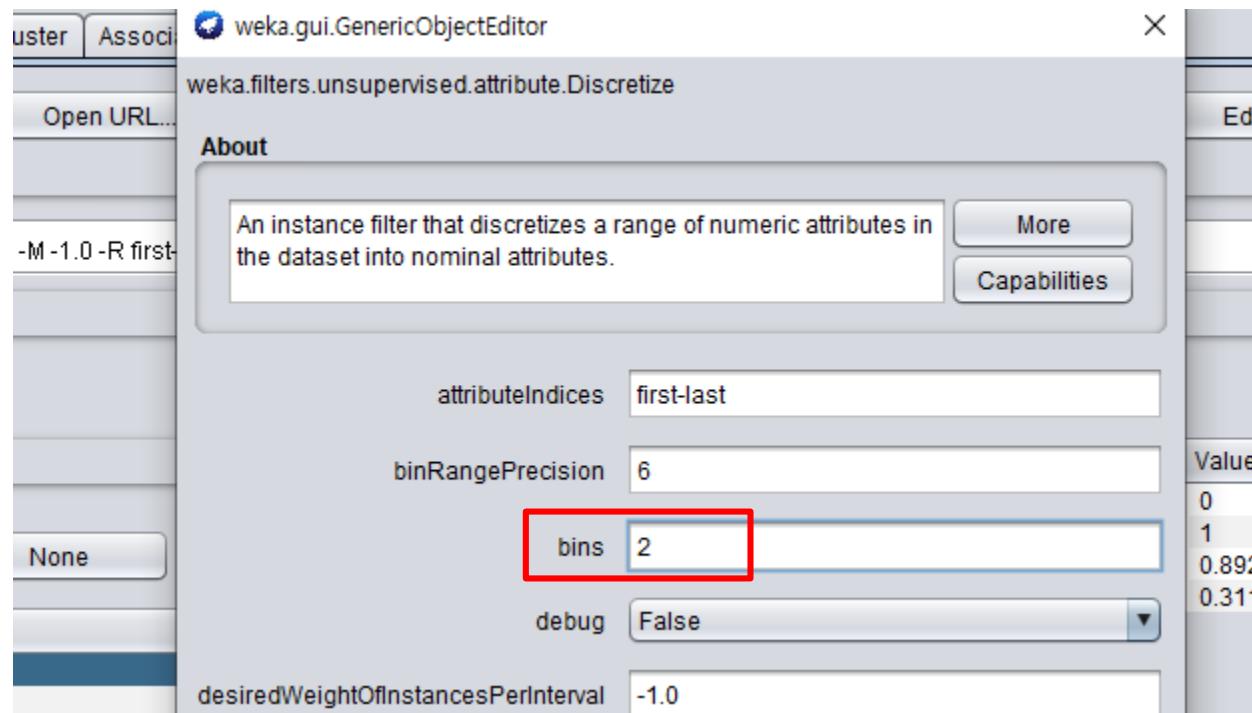
# Discretizing numeric attributes

- Go back to Preprocess and click 'Undo' to undo the discretization



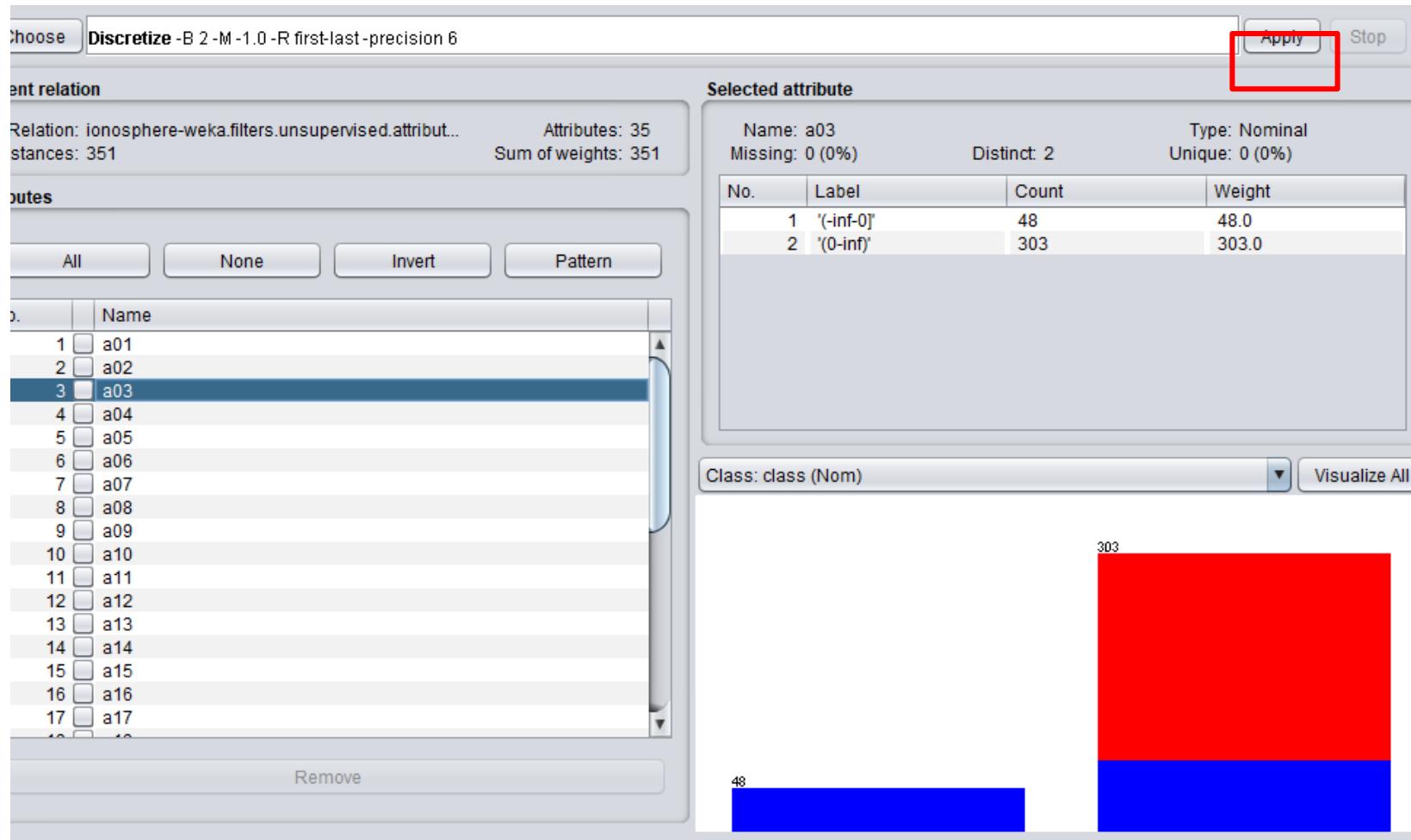
# Discretizing numeric attributes

- This time, set the number of bins to 2



# Discretizing numeric attributes

- Click apply then the data is divided into 2 bins



# Discretizing numeric attributes

- Run the classifier
- We get a different result

Classifier

Choose J48 - C 0.25 - M 2

Test options

Use training set  
 Supplied test set Set...  
 Cross-validation Folds 10  
 Percentage split % 66  
More options...

(Nom) class ▾

Start Stop

Result list (right-click for options)

Classifier output

```
Time taken to build model: 0 seconds
===
Stratified cross-validation ===
===
Summary ===

Correctly Classified Instances      319          90.8832 %
Incorrectly Classified Instances   32           9.1168 %
Kappa statistic                   0.7925
Mean absolute error               0.1493
Root mean squared error          0.2886
Relative absolute error          32.4268 %
Root relative squared error     60.1626 %
Total Number of Instances        351
```

The 'Correctly Classified Instances' row is highlighted with a red border.

# Discretizing numeric attributes

- The size of the tree is much smaller than the tree with 40 bins

```
|   |   |   |   a15 = '(0-inf)'  
|   |   |   |   a21 = '(-inf-0]': b (5.0/1.0)  
|   |   |   |   a21 = '(0-inf)': g (87.0/11.0)  
|   |   |   a08 = '(0-inf)': g (161.0/13.0)
```

```
Number of Leaves : 7
```

```
Size of the tree : 13
```

```
Time taken to build model: 0 seconds
```

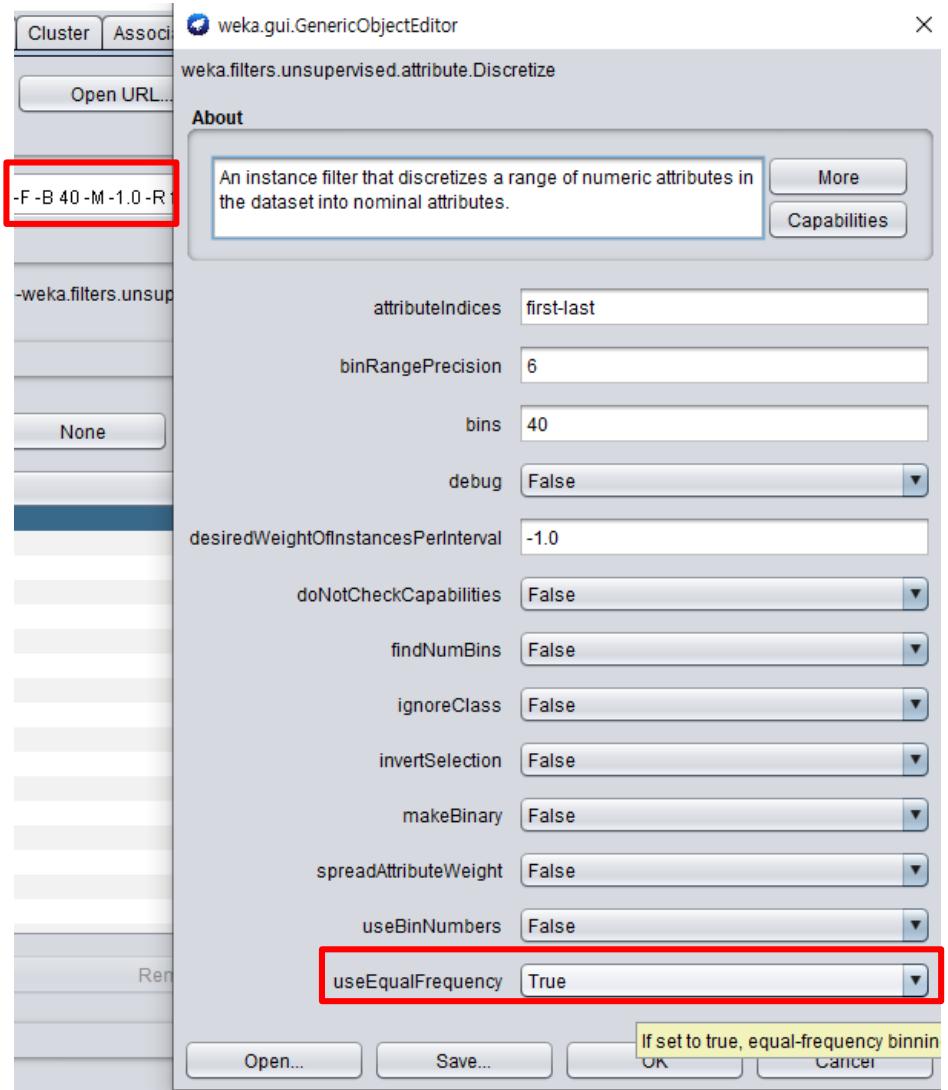
```
==== Stratified cross-validation ====
```

```
==== Summary ====
```

Correctly Classified Instances	319	90.8832 %
Incorrectly Classified Instances	32	9.1168 %
Kappa statistic	0.7925	

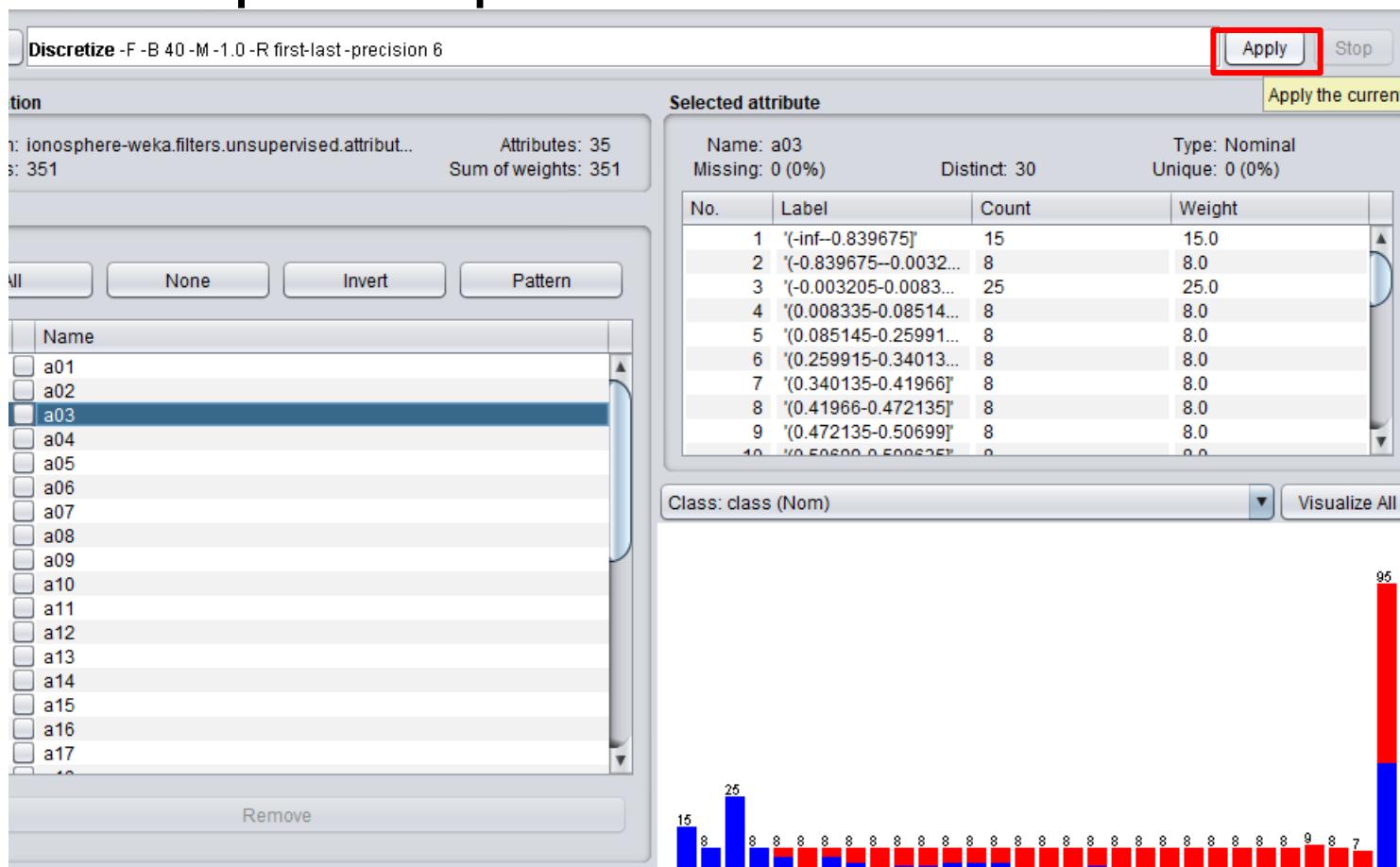
# Discretizing numeric attributes

- Open the options and set useEqualFrequency to True
- Don't forget to undo before applying



# Discretizing numeric attributes

- If you apply the discretization, the bins have almost equal frequencies



# Discretizing numeric attributes

- Run the classifier to see that the result is different from equi-width bins

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Classifier

Choose J48 -C 0.25 -M 2

Test options

Use training set  
 Supplied test set Set...  
 Cross-validation Folds 10  
 Percentage split % 66  
More options...

(Nom) class

Start Stop

Result list (right-click for options)

Classifier output

```
Time taken to build model: 0 seconds

==== Stratified cross-validation ====
==== Summary ===

Correctly Classified Instances          306           87.1795 %
Incorrectly Classified Instances        45            12.8205 %
Kappa statistic                         0.7219
Mean absolute error                     0.1868
Root mean squared error                 0.3387
Relative absolute error                  40.56   %
Root relative squared error             70.6027 %
Total Number of Instances                351
```

The 'Correctly Classified Instances' row is highlighted with a red border.

# Discretizing numeric attributes

- Result when bins = 5 and equi-frequency bins

Preprocess Classify Cluster Associate Select attributes Visualize

**Classifier**

Choose J48 -C 0.25 -M 2

**Test options**

Use training set  
 Supplied test set Set...  
 Cross-validation Folds 10  
 Percentage split % 66  
More options...

**(Nom) class**

Start Stop

**Classifier output**

```
Time taken to build model: 0 seconds

==== Stratified cross-validation ====
==== Summary ===

Correctly Classified Instances      318           90.5983 %
Incorrectly Classified Instances   33            9.4017 %
Kappa statistic                   0.7879
Mean absolute error               0.1399
Root mean squared error          0.2879
Relative absolute error          30.3862 %
Root relative squared error     60.018  %
```

The 'Correctly Classified Instances' row is highlighted with a red border.

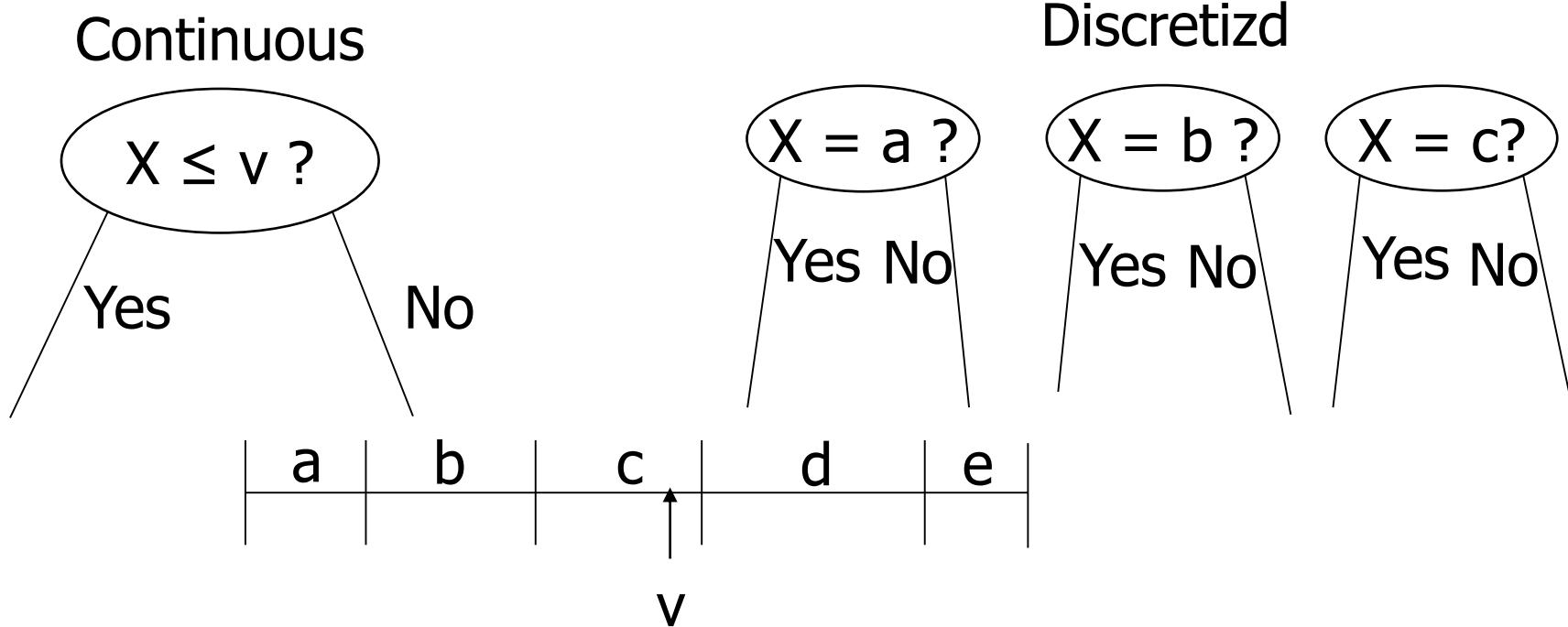
# **Discretizing numeric attributes**

---

- Note : The affect of discretizing varies with regard to the dataset
- You have to find the best choice through multiple experiments

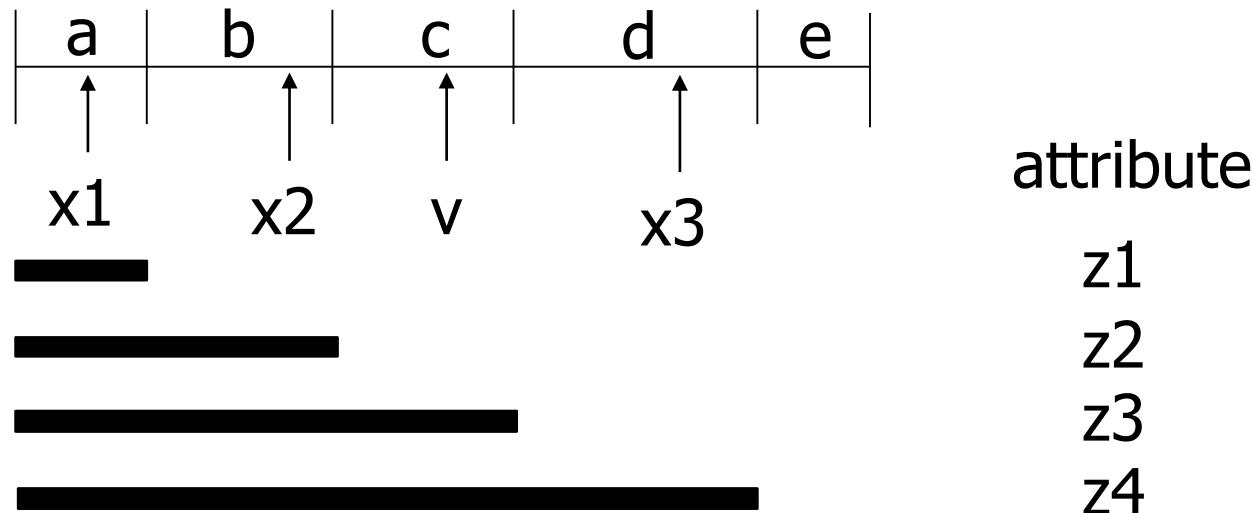
# Discretizing numeric attributes

- A problem of discretization : It is inefficient to find out ordering information from discrete attributes
- Ex) In a decision tree,



# Discretizing numeric attributes

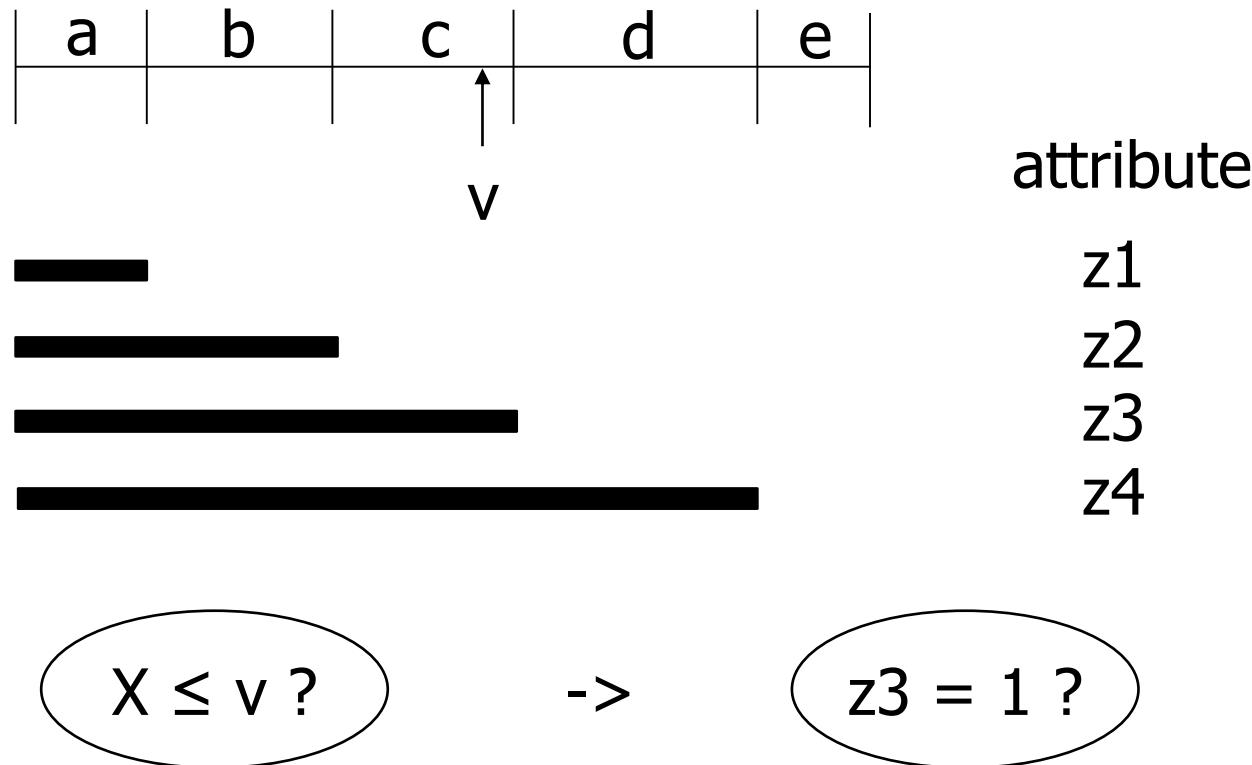
- A solution : One discretized attribute with k values  
->  $k-1$  binary attributes



	<b><math>z_1</math></b>	<b><math>z_2</math></b>	<b><math>z_3</math></b>	<b><math>z_4</math></b>
$x_1$	1	1	1	1
$x_2$	0	1	1	1
$x_3$	0	0	0	1

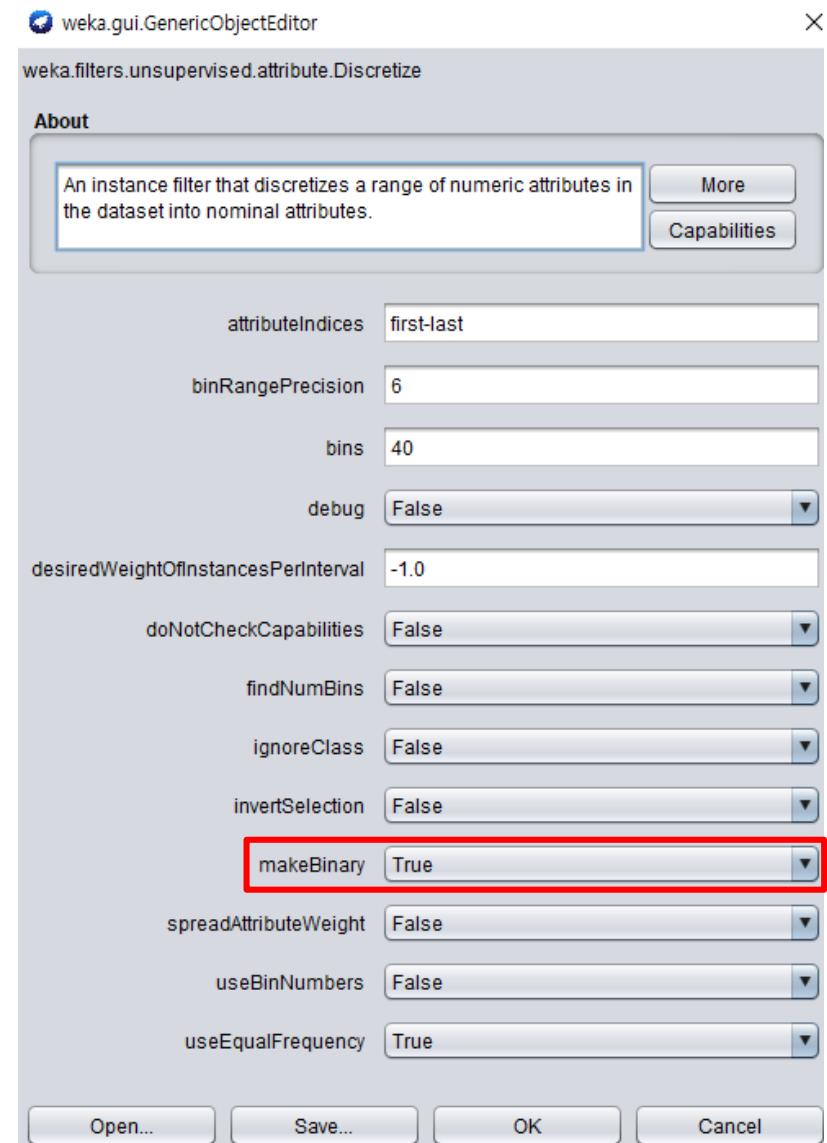
# Discretizing numeric attributes

- A solution : One discretized attribute with k values  
->  $k-1$  binary attributes



# Discretizing numeric attributes

- 'makeBinary' = True is the corresponding option
- Set bins = 40, useEqualFrequency = True
- Don't forget to undo before applying



# Discretizing numeric attributes

- Run the classifier, then we get a better result

```
Time taken to build model: 0.08 seconds

==== Stratified cross-validation ====
==== Summary ====



|                                  |           |           |
|----------------------------------|-----------|-----------|
| Correctly Classified Instances   | 332       | 94.5869 % |
| Incorrectly Classified Instances | 19        | 5.4131 %  |
| Kappa statistic                  | 0.8805    |           |
| Mean absolute error              | 0.0797    |           |
| Root mean squared error          | 0.2185    |           |
| Relative absolute error          | 17.3152 % |           |
| Root relative squared error      | 45.5372 % |           |
| Total Number of Instances        | 351       |           |



==== Detailed Accuracy By Class ====



|               | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC   | ROC Area | PRC Area | C1 |
|---------------|---------|---------|-----------|--------|-----------|-------|----------|----------|----|
| a             | 0.889   | 0.022   | 0.957     | 0.889  | 0.922     | 0.882 | 0.935    | 0.935    | b  |
| b             | 0.978   | 0.111   | 0.940     | 0.978  | 0.959     | 0.882 | 0.935    | 0.937    | g  |
| Weighted Avg. | 0.946   | 0.079   | 0.946     | 0.946  | 0.945     | 0.882 | 0.935    | 0.937    |    |



==== Confusion Matrix ====

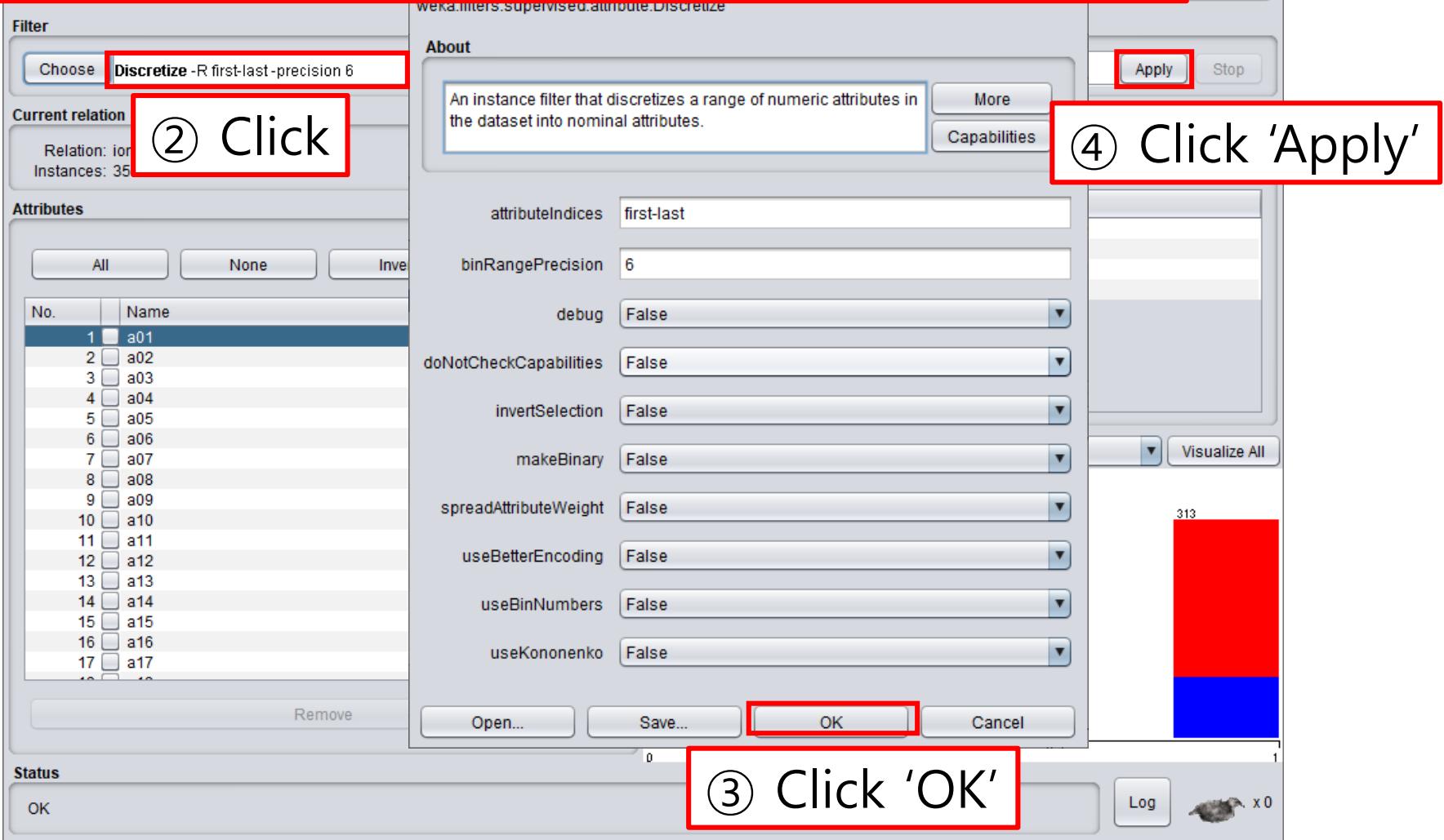


|     |     |                   |
|-----|-----|-------------------|
| a   | b   | <-- classified as |
| 112 | 14  | a = b             |
| 5   | 220 | b = a             |

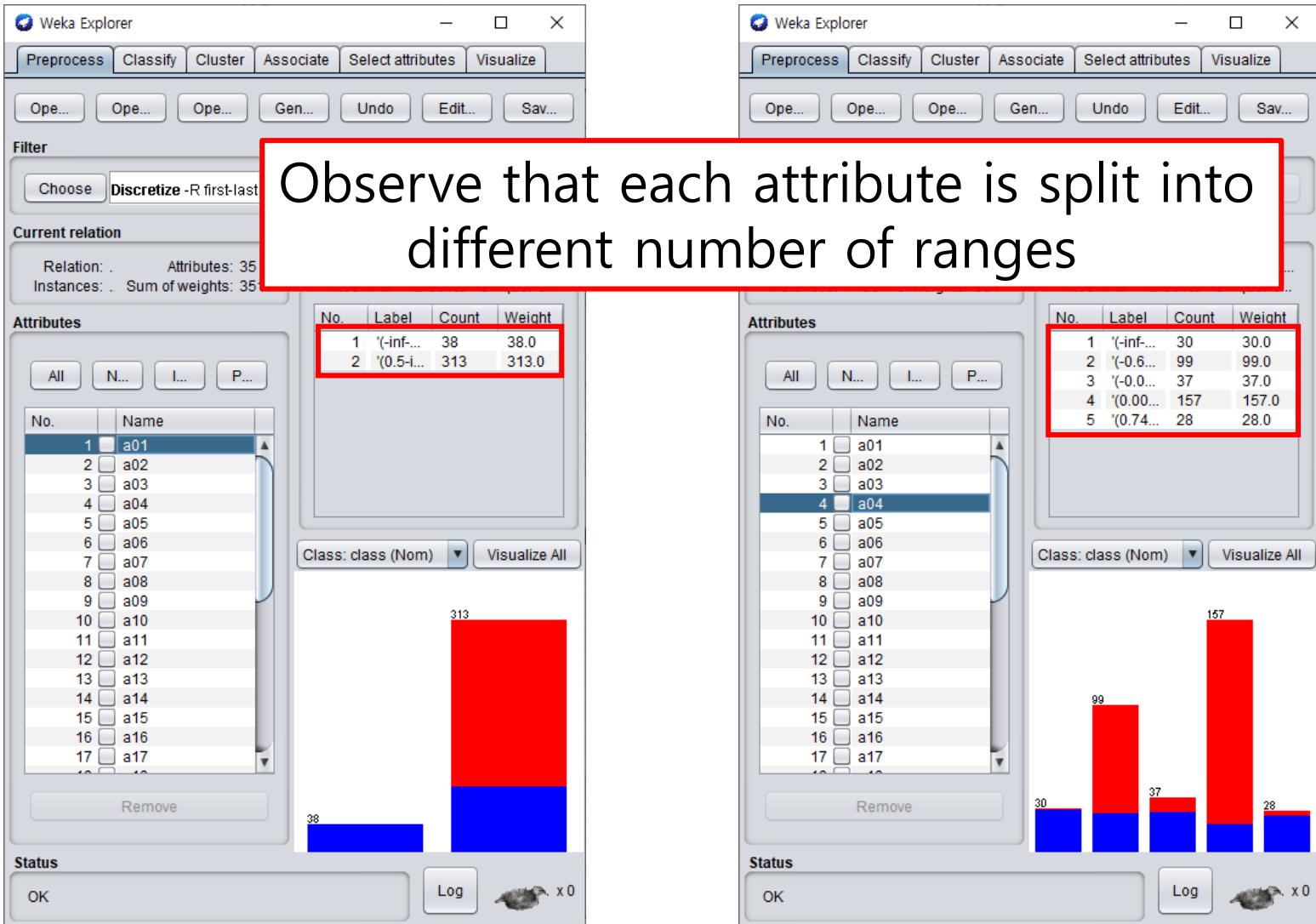

```

# Supervised Discretization

① Choose -> supervised -> attribute -> Discretize



# Supervised Discretization



# Classification With Discretization

The accuracy of J48 classifier is improved via discretization

The screenshot shows the Weka Explorer interface with the 'Classify' tab selected. The 'Classifier' panel shows 'J48 - C 0.25 - M 2' chosen. In the 'Test options' panel, 'Cross-validation' is selected with 'Folds' set to 10. The 'Classifier output' panel displays classification statistics. A blue box highlights the accuracy values: 89.1738 % and 10.8262 %. The 'Result list' panel shows '13:43:21 - trees.J48' selected. The 'Status' panel at the bottom shows 'OK'.

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Classifier

Choose J48 - C 0.25 - M 2

Test options

Use training set  
 Supplied test set Set...  
 Cross-validation Folds 10  
 Percentage split % 66  
More options...

(Nom) class

Start Stop

Result list (right-click for options)

13:28:02 - trees.J48  
13:43:21 - trees.J48

Classifier output

Time taken to

==== Stratified Summary =====

	Correctly Classified Instances	313	89.1738 %
Incorrectly Classified Instances	38	10.8262 %	
Kappa statistic	0.7571		
Mean absolute error	0.1333		
Root mean squared error	0.2969		
Relative absolute error	28.9567 %		
Root relative squared error	61.8807 %		
Total Number of Instances	351		

==== Detailed Accuracy By Class ====

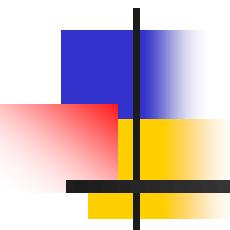
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
a	0.778	0.044	0.907	0.778	0.838	0.762	0.923	0.891	b
	0.956	0.222	0.885	0.956	0.919	0.762	0.923	0.932	g
Weighted Avg.	0.892	0.158	0.893	0.892	0.890	0.762	0.923	0.917	

==== Confusion Matrix ====

	a	b		<- classified as
a	98	28		a = b
b	10	215		b = g

Status

OK Log x 0



# Python - Discretization

# Download the Dataset

---

- Download ionosphere.csv from  
<http://kdd.snu.ac.kr/python/>
- Save the csv file in the same directory as the source file (.ipynb)

# Import Libraries

```
import pandas as pd  
from sklearn.preprocessing import KBinsDiscretizer  
from sklearn.naive_bayes import MultinomialNB  
from sklearn.model_selection import KFold, cross_val_score  
import matplotlib.pyplot as plt
```

- pandas: a library for data analysis
- KBinsDiscretizer: a class for discretization
- matplotlib: a library for plotting
- MultinomialNB: a class for naïve bayes classifier
- KFold, cross\_val\_score: a class and a function for K-fold corss validation

# Data Preprocessing

```
df = pd.read_csv('ionosphere.csv')
df[:3]
```

	a01	a02	a03	a04	a05	.
0	1	0	0.99539	-0.05889	0.85243	0.02
1	1	0	1.00000	-0.18829	0.93035	-0.36
2	1	0	1.00000	-0.03365	1.00000	0.00

# Data Preprocessing

Ignore the last column (class label)

```
X = df.values[:, :-1]  
y = df.values[:, -1]
```

Only the last column (class label)

- Print labels of first two instances

```
print(y[:2])
```

```
['g' 'b']
```

# Data Preprocessing

- Print features of first two instances

```
print(X[:2])
```

```
[[ 1 0 0.99539 -0.05889 0.8524299999999999 0.02306 0.8339799999999999  
-0.37708 1.0 0.0376 0.8524299999999999 -0.17755 0.59755 -0.44945  
0.60536 -0.38223 0.8435600000000001 -0.38542 0.58212 -0.32192 0.56971  
-0.29674 0.36946 -0.47357 0.56811 -0.51171 0.4107800000000003  
-0.4616800000000003 0.21266 -0.3409 0.42267 -0.54487 0.18641  
-0.4529999999999996 ]]  
[ 1 0 1.0 -0.18829 0.93035 -0.36156 -0.10868 -0.9359700000000001 1.0  
-0.04548999999999996 0.50874 -0.67743 0.3443199999999996 -0.69707  
-0.51685 -0.97515 0.05499 -0.62237 0.33109 -1.0 -0.13151  
-0.4529999999999996 -0.18056 -0.35734 -0.20332 -0.26569 -0.20468  
-0.1840099999999998 -0.1904 -0.1159299999999999 -0.16626  
-0.0628799999999999 -0.13738 -0.02447 ]]
```

# Discretization

```
discretizer = KBinsDiscretizer(n_bins=5, encode='ordinal',  
                             strategy='uniform')
```

- n\_bins: the number of bins to produce
- encode
  - "onehot": encode the transformed result with one-hot encoding and return a sparse matrix
  - "onehot-dense": encode the transformed result with one-hot encoding and return a dense array
  - "ordinal": return the bin identifier encoded as an integer value
- strategy
  - "uniform": equal-width
  - "quantile": equal-frequency
  - "kmeans": values in each bin have the same nearest center of a 1D k-means cluster

# Discretization

```
discretizer = KBinsDiscretizer(n_bins=5, encode='ordinal',  
                               strategy='uniform')  
discretizer.fit(X) ← Build the edges of each bin  
print(discretizer.bin_edges_) ← Print the edges of bins of  
                             all attributes
```

[array([0. , 0.2, 0.4, 0.6, 0.8],  
 array([-1. , -0.6, -0.2, 0.2, 0.6, 1. ]),  
 array([-1. , -0.6, -0.2, 0.2, 0.6, 1. ]),  
 array([-1. , -0.6, -0.2, 0.2, 0.6, 1. ]),  
 array([-1. , -0.6, -0.2, 0.2, 0.6, 1. ])]

# Discretization

```
discretizer = KBinsDiscretizer(n_bins=5, encode='ordinal',
                               strategy='uniform')
discretizer.fit(X)
print(discretizer.bin_edges_)
```

```
[array([0. , 0.2, 0.4, 0.6, 0.8, 1. ]), array([-inf, inf])  
 array([-1. , -0.6, -0.2, 0.2, 0.6, 1. ])]  
 array([[-1. , 0. , 0. , 0. , 0. , 1. ]])
```

The bins of 1<sup>st</sup> attributes are

[0.0, 0.2), [0.2, 0.4), [0.4, 0.6), [0.6, 0.8), [0.8, 1.0]

# Discretization

```
discretizer = KBinsDiscretizer(n_bins=5, encode='ordinal',  
                               strategy='uniform')  
discretizer.fit(X)  
print(discretizer.bin_edges_)
```

```
[array([0. , 0.2, 0.4, 0.6, 0.8, 1. ]), array([-inf, inf])  
 array([-1. , -0.6, -0.2, 0.2, 0.6, 1. ]),  
 array([-1. , -0.6, -0.2, 0.2, 0.6, 1. ]),  
 array([-1. , -0.6, -0.2, 0.2, 0.6, 1. ]),  
 array([-1. , -0.6, -0.2, 0.2, 0.6, 1. ])]
```

The edges becomes [-inf, inf] since all values of 2<sup>nd</sup> attribute are constant

# Plotting the Histogram

The index of 1<sup>st</sup> attribute

Input the data of 1<sup>st</sup> attribute  
values all rows

Input the edges of  
bins of 1<sup>st</sup> attribute

```
col_i = 0
plt.hist(X[:, col_i:col_i+1],
          bins=discretizer.bin_edges_[col_i],
          linewidth=1.2, edgecolor='black')
```

```
plt.suptitle(df.columns[col_i])
plt.show()
```

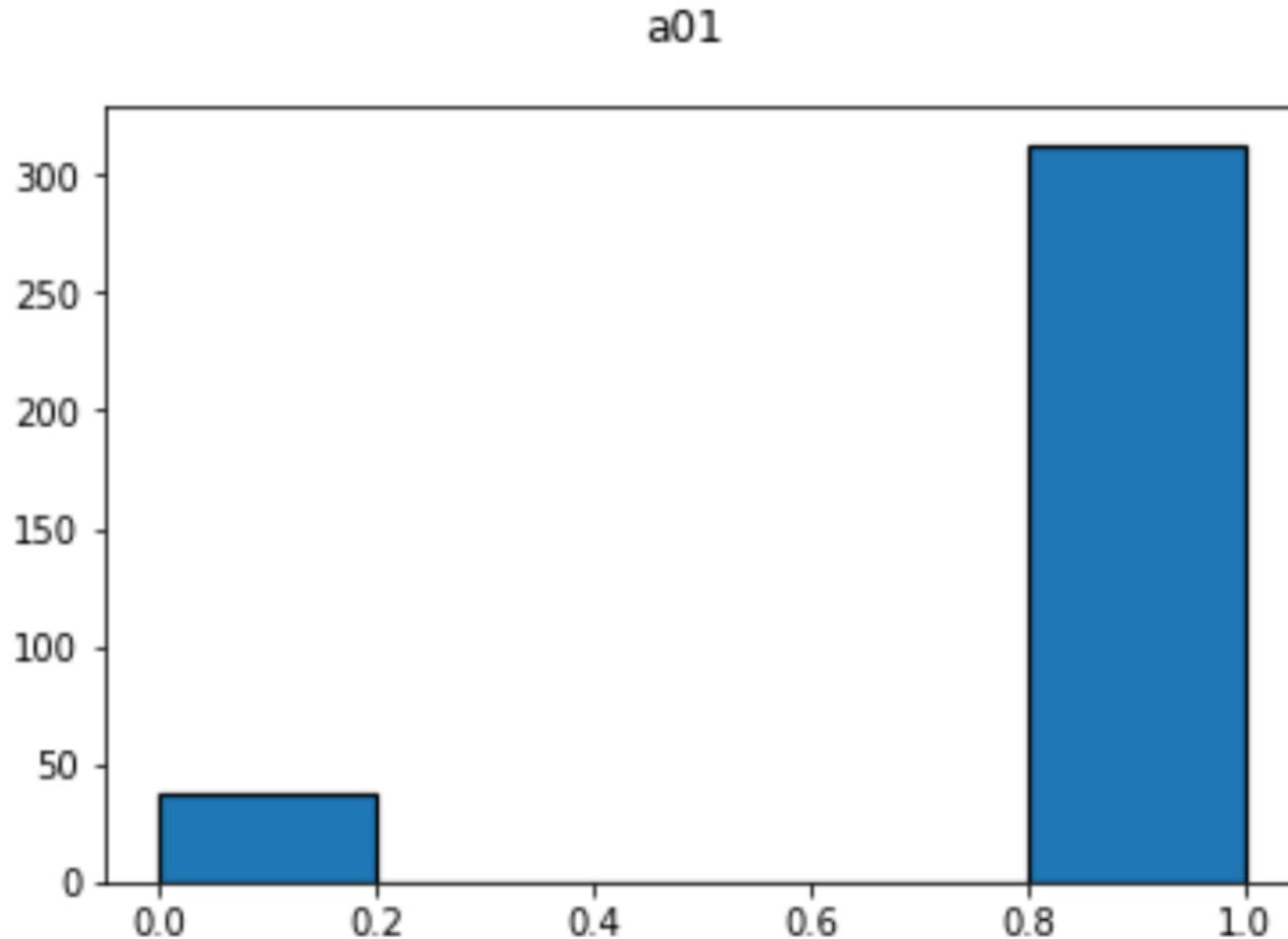
Configurations for  
showing outlines of  
bars

Set the title of the figure to  
the column name

Show the plotted  
figure

# Plotting the Histogram

- The result figure:



# Plotting the Histogram

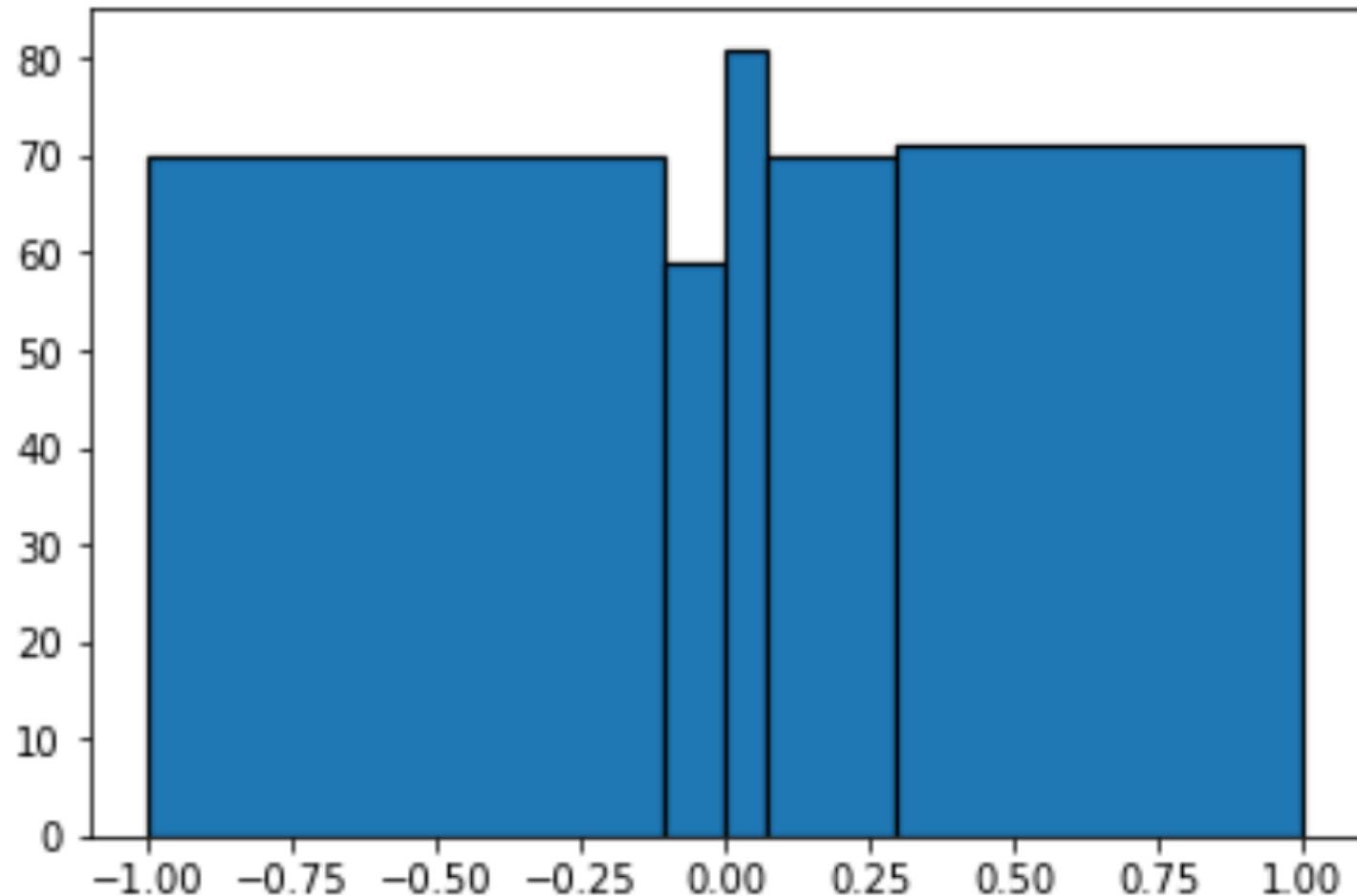
- Set strategy='quantile', and repeat the others

```
discretizer = KBinsDiscretizer(n_bins=5, encode='ordinal',
                               strategy='quantile')
discretizer.fit(X)
col_i = 3
plt.hist(X[:, col_i:col_i+1],
          bins=discretizer.bin_edges_[col_i],
          linewidth=1.2, edgecolor='black')
plt.suptitle(df.columns[col_i])
plt.show()
```

# Plotting the Histogram

---

a04



# Discretization

Convert each value into discretized value

```
transformed_X = discretizer.transform(X)  
print(transformed_X[:3])
```

Print first 3 rows of transformed data

```
[[4. 0. 3. 1. 2. 2. 2. 0. 4. 2. 3. 0. 2. 0. 2.  
 2. 0. 1. 1. 1. 2. 0. 2. 0.]  
[4. 0. 4. 0. 3. 0. 0. 0. 4. 1. 2. 0. 1. 0. 0.  
 0. 1. 0. 1. 0. 1. 0. 1. 0.]  
[4. 0. 4. 1. 4. 2. 4. 0. 3. 2. 2. 2. 3. 2. 2.  
 2. 1. 2. 1. 2. 1. 3. 1. 2. 0.]]
```

# Run a Classification Task for the Transformed Data

```
clf = MultinomialNB()      —— Create the classifier object
cv = KFold(                —— Create an K-Folds cross-validator object
            n_splits=10,    —— The number of folds
            shuffle=True,   —— Whether to shuffle the data before splitting
            random_state=0) —— The random seed
scores = cross_val_score(   —— Compute accuracies for K-folds
                           clf,           —— The object of the naïve bayes classifier
                           transformed_X,
                           y,             ——> The features and labels
                           cv=cv)
print(scores.mean())        —— Compute the average accuracy
```

0.7322222222222222

# Practice

---

- Change the number of bins
- Change the attribute to plot histogram

# Chapter 3: Data Preprocessing

---

- Data Preprocessing: An Overview
  - Data Quality
  - Major Tasks in Data Preprocessing
- Data Cleaning
- Data Integration
- Data Reduction
- Data Transformation and Data Discretization
- Summary 

# Summary

---

- **Data quality:** accuracy, completeness, consistency, timeliness, believability, interpretability
- **Data cleaning:** e.g. missing/noisy values, outliers
- **Data integration** from multiple sources:
  - Entity identification problem
  - Remove redundancies
  - Detect inconsistencies
- **Data reduction**
  - Dimensionality reduction
  - Numerosity reduction
  - Data compression
- **Data transformation and data discretization**
  - Normalization
  - Concept hierarchy generation

# References

---

- D. P. Ballou and G. K. Tayi. Enhancing data quality in data warehouse environments. Comm. of ACM, 42:73-78, 1999
- A. Bruce, D. Donoho, and H.-Y. Gao. Wavelet analysis. *IEEE Spectrum*, Oct 1996
- T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley, 2003
- J. Devore and R. Peck. *Statistics: The Exploration and Analysis of Data*. Duxbury Press, 1997.
- H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. *VLDB'01*
- M. Hua and J. Pei. Cleaning disguised missing data: A heuristic approach. *KDD'07*
- H. V. Jagadish, et al., Special Issue on Data Reduction Techniques. *Bulletin of the Technical Committee on Data Engineering*, 20(4), Dec. 1997
- H. Liu and H. Motoda (eds.). *Feature Extraction, Construction, and Selection: A Data Mining Perspective*. Kluwer Academic, 1998
- J. E. Olson. *Data Quality: The Accuracy Dimension*. Morgan Kaufmann, 2003
- D. Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann, 1999
- V. Raman and J. Hellerstein. *Potters Wheel: An Interactive Framework for Data Cleaning and Transformation*, VLDB'2001
- T. Redman. *Data Quality: The Field Guide*. Digital Press (Elsevier), 2001
- R. Wang, V. Storey, and C. Firth. A framework for analysis of data quality research. *IEEE Trans. Knowledge and Data Engineering*, 7:623-640, 1995