Data Preprocessing

IS 665 Data Mining, Data Warehousing and Visualization

Agenda

- Loading your data
- Cleaning your data
  - Technically correct data
  - Consistent data
  - Handling missing data
  - Misclassification and outliers
- Data Manipulation

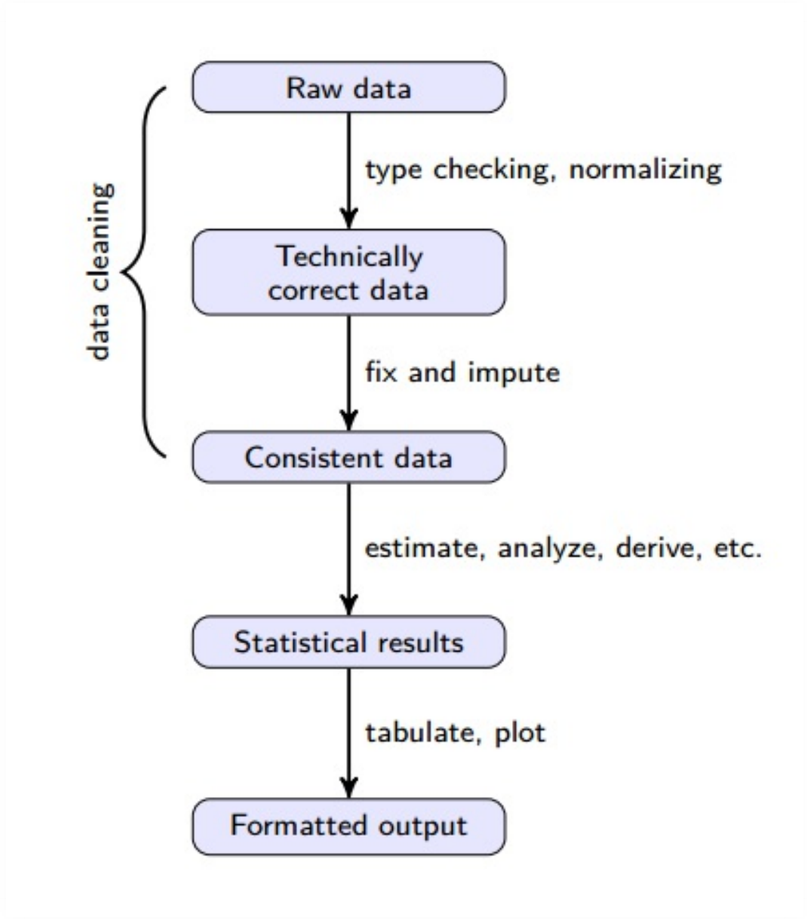
Why to preprocess ?

Data is usually not in the form we need it to be.

- It is not in a form that the data mining models will accept
- Fields that are obsolete or redundant
- There are missing values
- Or values are not consistent with policy or common sense

Data preprocessing alone can account for 10 – 60% of all the time and effort for the entire data mining process.

Statistical Analysis Steps



- **Raw Data**: Not necessarily all correct, not formatted for systems to accept.
- **Technically Correct Data**: Can be read, but doesn't mean values are correct (e.x. negative age values, missing data)
- **Consistent Data**: Data ready for analysis and interpretation

Loading Data

Loading data

- There are many data sets available in R. They are good for exploring / practicing R functions and for testing your scripts.

```
# install.packages('datasets')
require("datasets")
`?`(datasets)
library(help = "datasets")
data("airquality")
head(airquality, 3)  #View(airquality)
```

```
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
```

- But mostly you will load your own data sets…

Loading Data: Read.table and its cousins

- The followings commands are good for loading fixed-width or csv-like formats, (not for XML-like formats)

```
read.table
read.delim      read.delim2
read.csv        read.csv2
read.table      read.fwf
```

- All return a **data.frame**.
- Always check for the first line (is it column-name or a data point ?)

Loading data ctn.

With read.table you can load data either from a web source or from a local file.

- Below is an example to get a data from an online source.

```
data.url = "http://yegingenc.com/lectures/data/SampleStudentGrades.txt"
sample.grades = read.table(data.url, header = T)
head(sample.grades)
```

```
  Student_ID Semester Grades
1          1  14_Fall     82
2          2  14_Fall     73
3          3  14_Fall     88
4          4  14_Fall     96
5          5  14_Fall     77
6          6  14_Fall     51
```

You need to be very careful with the options. E.x. what happens if we don't set (header=T)

- Try to create a dummy csv file and see if you can load it to R. Run ?read.csv to get help understand the usage of the function

Other ways to read data

With readLines you can read data in completely raw form. While this requires more work to make the data look like what read.table produces, it gives us full control over how each line is interpreted and transformed into fields in a rectangular data set.

```
data.url = "http://yegingenc.com/lectures/data/SampleStudentGrades.txt"
sample.grades2 = readLines(data.url)
head(sample.grades2)
```

```
[1] "Student_ID\tSemester\tGrades" "1\t14_Fall\t82"
[3] "2\t14_Fall\t73"               "3\t14_Fall\t88"
[5] "4\t14_Fall\t96"               "5\t14_Fall\t77"
```

- We will see how to clean this data next

Cleaning Data

1. Technically correct data

2. Consistent Data

Cleaning Data 1: Technically correct data

Converting raw data to *technically correct* data. That is making sure that

- variables are correctly populated and each value belongs to a variable
- data types in the system match to the variable types. (Text variables are stored as text, numeric variables are stored as numbers etc.)

To this end, when working with data in R, make sure:

- Data is stored as `data.frame` with suitable column names,
- Type of each column in the `data.frame` matches to the value domain of the variable.
  - Numeric data –> `numeric` or `integer`
  - Textual data –> `character`
  - Categorical data –> `factor`

Cleaning Data 1 ctn

read.table' does some of the data cleaning. For example all numerical values are converted to 'numeric' type.

```
# Reminder: sample.grades was loaded by read.table
summary(sample.grades)
```

```
  Student_ID         Semester      Grades
Min.   :  1.0   14_Fall  :48   Min.   :21.00
1st Qu.: 32.5   15_Fall  :51   1st Qu.:72.00
Median : 64.0   15_Spring:28   Median :81.50
Mean   : 64.0                  Mean   :77.98
3rd Qu.: 95.5                  3rd Qu.:87.03
Max.   :127.0                  Max.   :99.28
```

summary() is a good function to get descriptive stats for each variable in the data set. Based on the variable type, it will give different statistics.

- For numeric variables (e.x. Grades), mean, median and quartiles.
- For categorical variables (e.x. Semester), it gives frequencies.

Cleaning Data 1 ctn

```
  Student_ID        Semester      Grades
Min.   :  1.0   14_Fall  :48   Min.   :21.00
1st Qu.: 32.5   15_Fall  :51   1st Qu.:72.00
Median : 64.0   15_Spring:28   Median :81.50
Mean   : 64.0                  Mean   :77.98
3rd Qu.: 95.5                  3rd Qu.:87.03
Max.   :127.0                  Max.   :99.28
```

- The problem is tough that sometimes numeric values are used for other purposes, for example student_id. (i.e. Mean student_id doesn't make sense.)
- So we change it to factor or make it the row name.

```
# Convert it to factor
sample.grades$Student_ID <- as.factor(sample.grades$Student_ID)


# Alternatively set it to a row name (and remove it from the
# dataset)
row.names(sample.grades) <- as.character(sample.grades$Student_ID)
sample.grades <- sample.grades[, -1]
sample.grades["1", ]
```

Cleaning Data 1 ctn

With readLines you can exercise precise control over how each line is interpreted and transformed into fields in a rectangular data set, but requires more work.

```
data2 =
readLines("http://yegingenc.com/lectures/data/SampleStudentGrades.txt")

# Split each line by tab
grades.raw = strsplit(data2, "\t")

# First line is the column names so let's save them
grades.colnames <- grades.raw[[1]]

# Creating vectors for each variable and identify their
# variable type
student.ids <- sapply(grades.raw[-1], function(x) x[1])
semester <- sapply(grades.raw[-1], function(x) x[2])
s.grades <- sapply(grades.raw[-1], function(x) x[3])

# Create a dataframe and identifying the variable types all
# at once
grades.df <- data.frame(semester = as.factor(semester), grade =
as.numeric(s.grades),
    row.names = student.ids)
summary(grades.df)
```

```
      semester      grade
 14_Fall  :48   Min.   :21.00
```

Cleaning data 2: Consistent data

Customer ID	Zip	Gender	Income	Age	Marital Status	Amount
1001	10048	M	78,000	C	M	5000
1002	J2S7K7	F	−40,000	40	W	4000
1003	90210		10,000,000	45	S	7000
1004	6269	M	50,000	0	S	1000
1005	55101	F	99,999	30	D	3000

What is wrong with the following fields ?

- Zip = J2S7K7 or 6269
- Gender = “ ”
- Income = -40,000 or 99,999

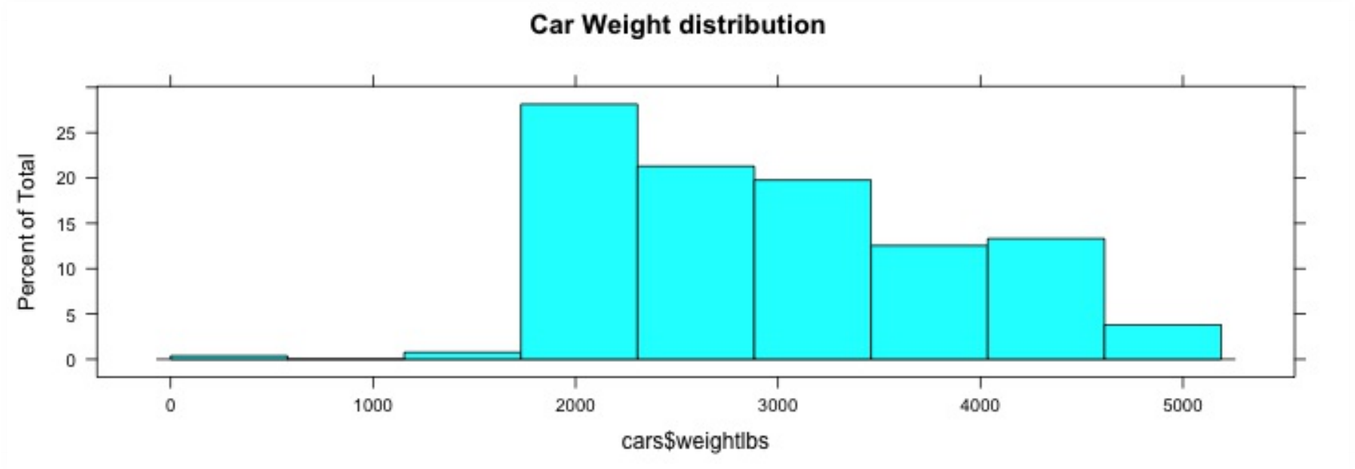
Handling Missing Data

- Handle by removing record.
  Easiest but you may end up with little data. If %5 of the data is missing across 30 variables (and spread evenly), ~ %80 data has to be removed.

- There are other alternatives:

  1. Replace the missing value with some constant, specified by the analyst.
  2. Replace the missing value with the field mean (for numeric variables) or the mode (for categorical variables).
  3. Replace the missing values with a value generated at random from the observed distribution of the variable.
  4. Replace the missing values with imputed values based on the other characteristics of the record.

Identifying Misclassifications and Outliers



Graphs are good ways find odd data

- What seems to be the problem with the distribution above ?

Data Manuplation

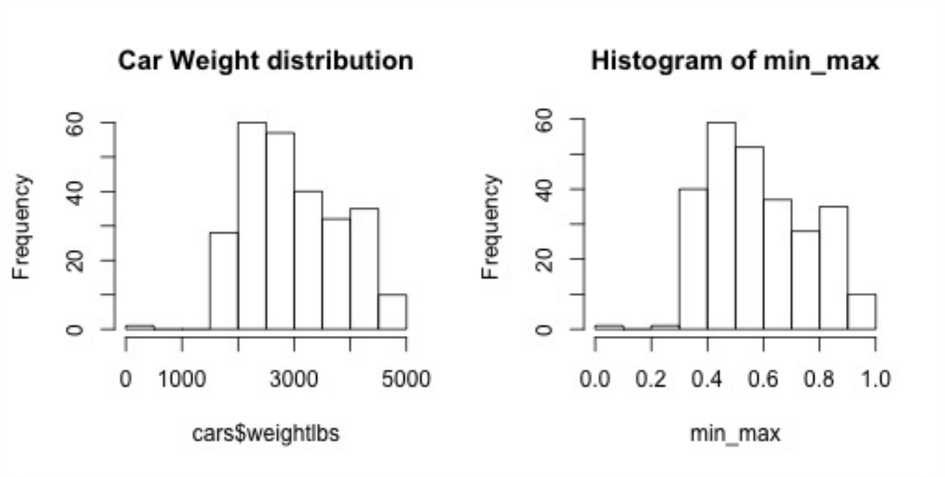
- Min-max normalization
- z-score standardization
- Transformations to achieve normality

Min-max normalization

Convert the values to be between 0 and 1.

$$x_i^* = \frac{x_i - min(X)}{max(X) - min(X)}$$

```
min_max = (cars$weightlbs - min(cars$weightlbs))/diff(range(cars$weightlbs))


par(mfrow = c(1, 2))
hist(cars$weightlbs, main = "Car Weight distribution")
hist(min_max)
```



z-score standardization

z-score is the number standard deviations (σ), a value is from the mean ($\hat{x}$) of the variable.

$$z - score = \frac{x - \bar{x}}{\sigma}$$

```
z_weights = (cars$weightlbs - mean(cars$weightlbs))/sd(cars$weightlbs)

# alternatively
z_weights2 = scale(cars$weightlbs, center = T, scale = T)
```

Transformations to achieve normality

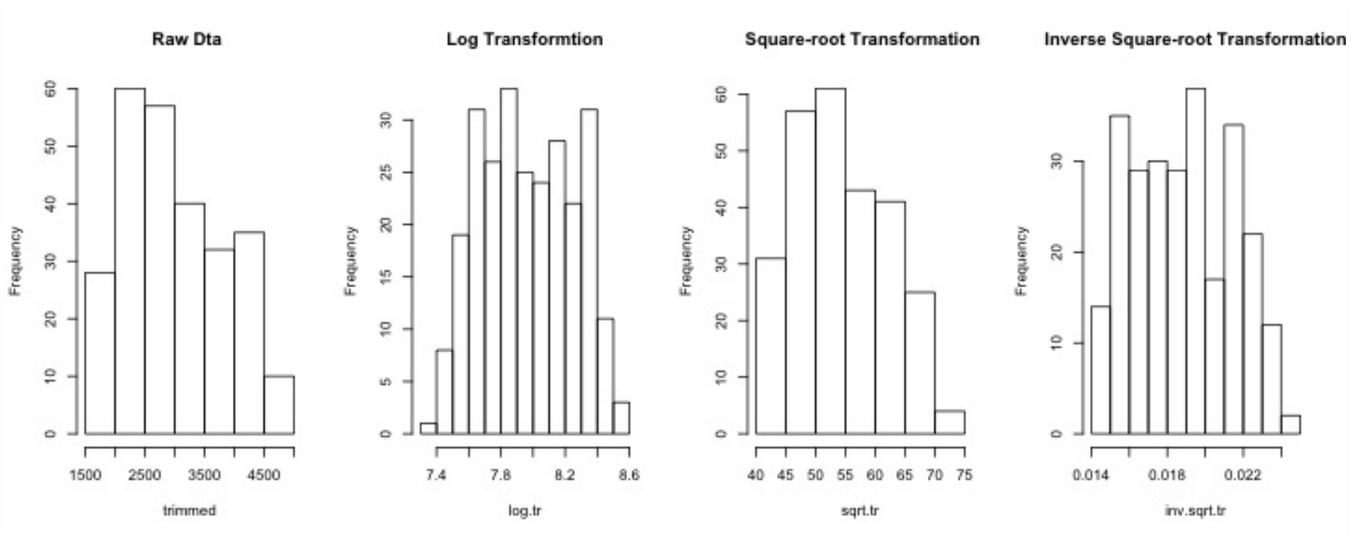
Many data mining models expects the data to be normally distributed (or at least be symmetric around the mean).

If the data is not normal (or nearly normal). There are three transformations that can be applied.

- Log transformation: ln(cars$weightlbs)
- Square-root transformation : sqrt(cars$weightlbs)
- Reverse Square-root transformation: 1/sqrt(cars$weightlbs)

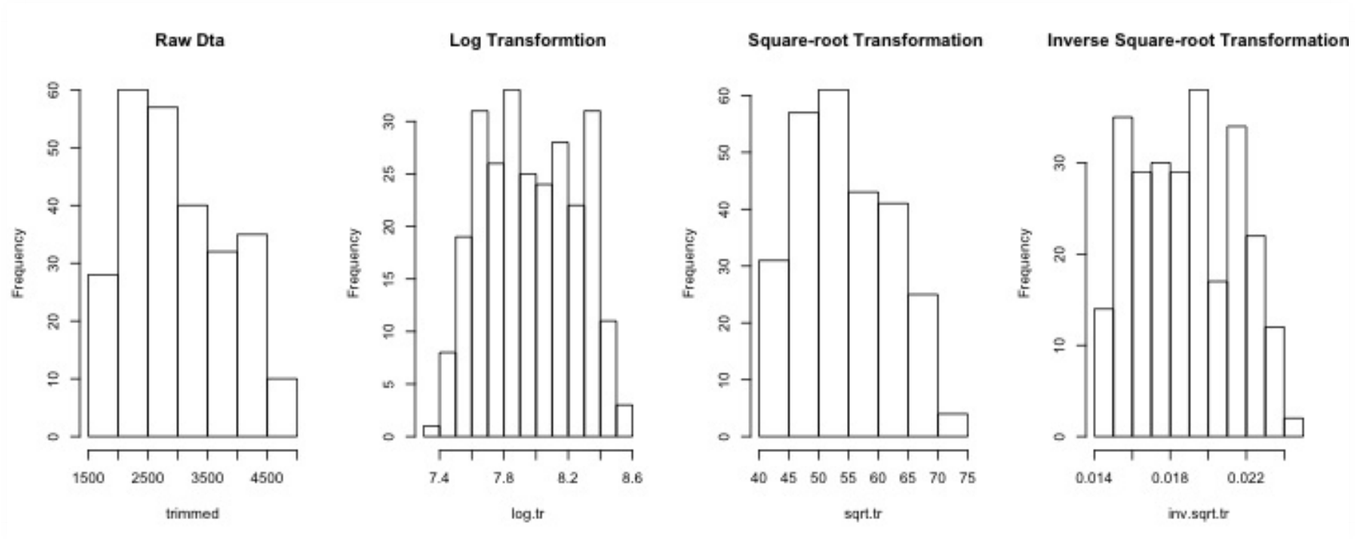
Transformations to achieve normality

```r
# Removin the outliers
trimmed = cars$weightlbs[-which(min(cars$weightlbs) == cars$weightlbs)]

# Appliying transfomrations
log.tr = log(trimmed)
sqrt.tr = sqrt(trimmed)
inv.sqrt.tr = 1/sqrt(trimmed)

# Creating histograms for the transformed data for comparison
par(mfrow = c(1, 4))
hist(trimmed, main = "Raw Dta ")
hist(log.tr, main = "Log Transformtion")
hist(sqrt.tr, main = " Square-root Transformation")
hist(inv.sqrt.tr, main = "Inverse Square-root Transformation")
```



Transformations to achieve normality

You can compare the symmetry of the data after transformations with histograms



- Log Transformation and Inverse Square-root Transformation seems to improve the symmetry

Transformations to achieve normality

Alternatively, you can compute skewness following the formula:

$$skewness = \frac{3(mean - median)}{sd}$$

```
skewness <- function(x) {
    return(3 * (mean(x) - median(x))/sd(x))
}
```

Note that in R we can create a function and call it later

```
skewness(trimmed)
```

```
[1] 0.6026224
```

```
skewness(log.tr)
```

```
[1] 0.1993812
```

```
skewness(sqrt.tr)
```

```
[1] 0.4061475
```

```
skewness(inv.sqrt.tr)
```

```
[1] 0.01127326
```

References

- Discovering Knowledge in Data: An Introduction to Data Mining, 2nd Edition
- https://cran.r-project.org/doc/contrib/de_Jonge+van_der_Loo-Introduction_to_data_cleaning_with_R.pdf