Introduction to Data Science CS61 June 12 - July 12, 2018

Dr. Ash Pahwa

Lesson 2: Data Exploration

Lesson 2.2: Normalization and Scaling

Outline

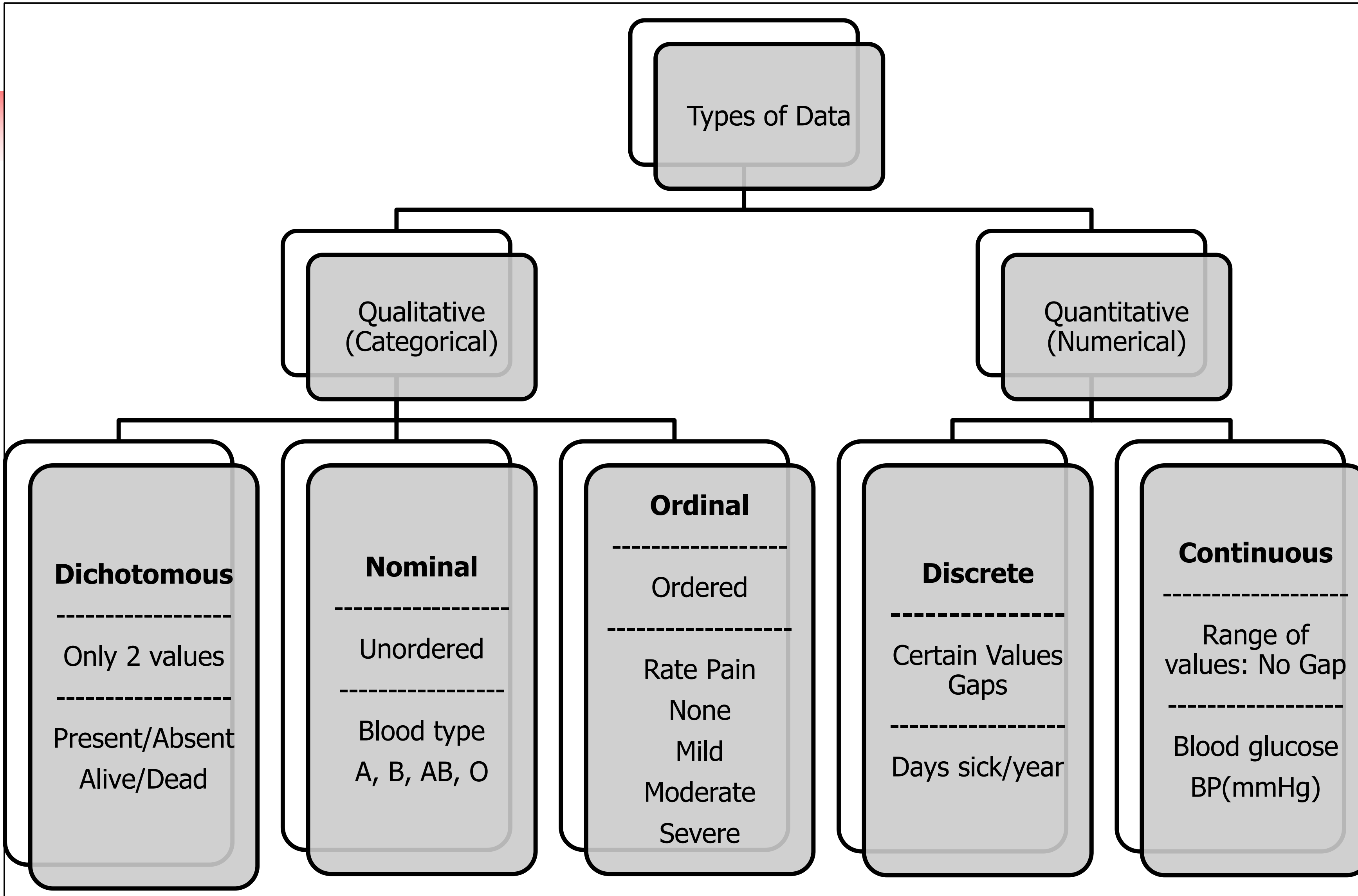
1. Types of Variables
2. Sampling
3. The Empirical Rule
4. Standardized Values
5. Data Normalization: Standardization and Scaling

1. Types of Variables

How to Represent Categorical Variables in R

Factors

Variables



Types of Data
Qualitative (Categorical)
Quantitative (Numerical)
Dichotomous
Only 2 values
Present/Absent
Alive/Dead
Nominal
Unordered
Blood type
A, B, AB, O
Ordinal
Ordered
Rate Pain
None
Mild
Moderate
Severe
Discrete
Certain Values Gaps
Days sick/year
Continuous
Range of values: No Gap
Blood glucose
BP(mmHg)

Types of Categorical Variables

- Nominal – categorical
  - Diabetes (Type1, Type2)
  - Type 1 is coded as 1
  - Type 2 is coded as 2

- Ordinal – categorical with order
  - Status (Poor, Improved, Excellent)
    - Excellent is coded as 1
    - Improved is coded as 2
    - Poor is coded as 3
  - Excellent < Improved < Poor

- Factors are
  - nominal or
  - ordinal variables
- They are stored and treated specially in R

Categorical Variables in R

Factors

- Nominal – categorical: Factor

```
> diabetes <- c("Type1", "Type2", "Type1", "Type1")
> diabetes
[1] "Type1" "Type2" "Type1" "Type1"
> class(diabetes)
[1] "character"
>
> diabetes <- factor(diabetes)
> diabetes
[1] Type1 Type2 Type1 Type1
Levels: Type1 Type2
> class(diabetes)
[1] "factor"
```

- Ordinal – categorical: Factor – default order alphabetical

```
> status <- c("Poor", "Improved", "Excellent", "Poor")
> status
[1] "Poor"      "Improved"  "Excellent" "Poor"
> class(status)
[1] "character"
>
> status <- factor(status, order=TRUE)
> status
[1] Poor      Improved  Excellent Poor
Levels: Excellent < Improved < Poor
> class(status)
[1] "ordered" "factor"
```

Factors – Ordinal Order can be changed

```
> status <- factor(status, order=TRUE)
> status
[1] Poor      Improved  Excellent Poor
Levels: Excellent < Improved < Poor
> class(status)
[1] "ordered" "factor"
>
> #####################################
>
>
> status <- factor(status, order=TRUE, levels=c("Poor", "Improved",
"Excellent"))
> status
[1] Poor      Improved  Excellent Poor
Levels: Poor < Improved < Excellent
> class(status)
[1] "ordered" "factor"
>
```

Factors - Example

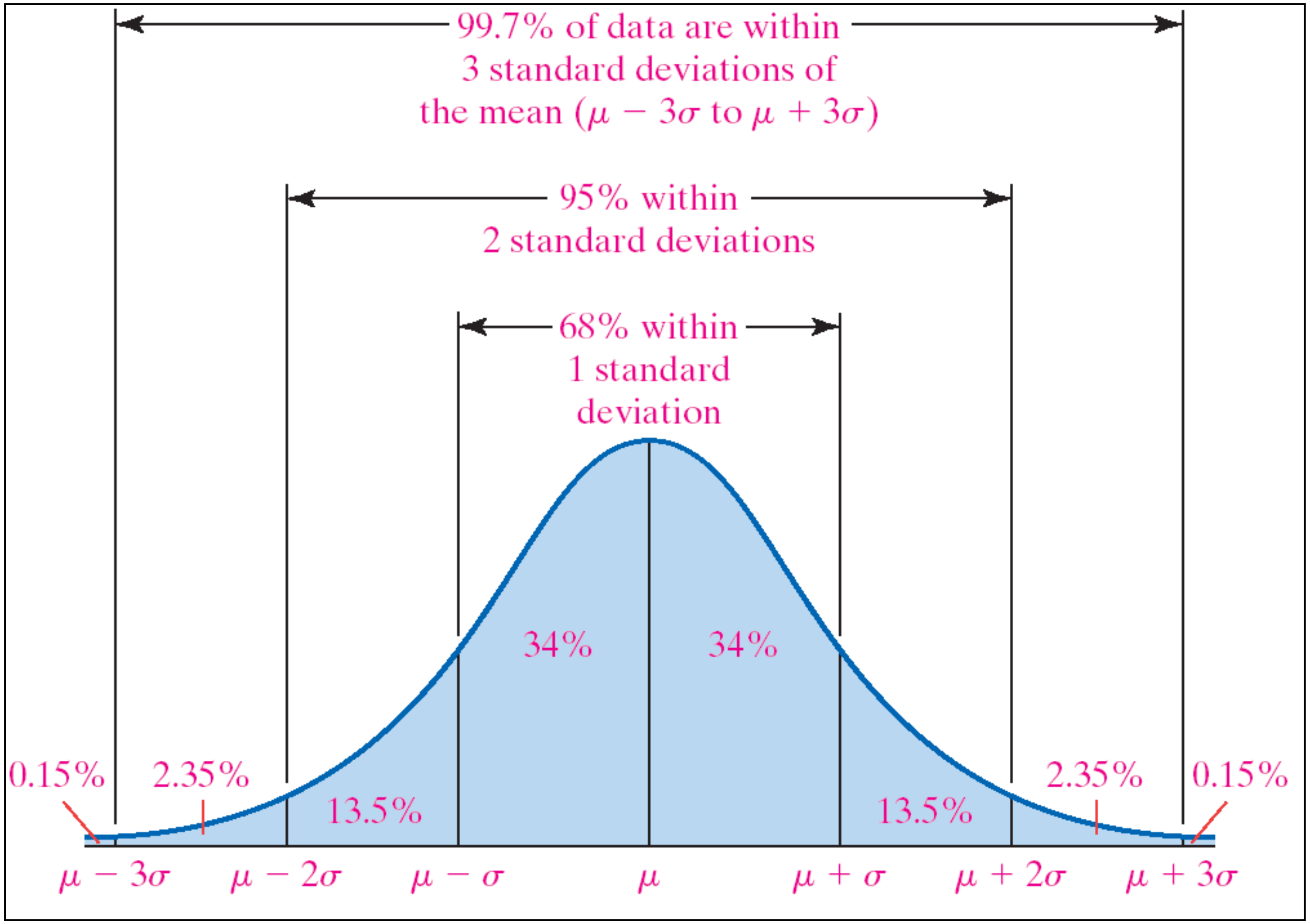
```
> patientID <- c(1,2,3,4)
> age <- c(25,34,28,52)
> diabetes <- c("Type1", "Type2", "Type1", "Type1")
> status <- c("Poor","Improved","Excellent","Poor")

> diabetes <- factor(diabetes)
> status <- factor(status, order=TRUE)

> patientdata <- data.frame(patientID, age, diabetes, status)
> patientdata
  patientID age diabetes    status
1         1  25    Type1      Poor
2         2  34    Type2  Improved
3         3  28    Type1 Excellent
4         4  52    Type1      Poor
> str(patientdata)
'data.frame':    4 obs. of  4 variables:
 $ patientID: num  1 2 3 4
 $ age      : num  25 34 28 52
 $ diabetes : Factor w/ 2 levels "Type1","Type2": 1 2 1 1
 $ status   : Ord.factor w/ 3 levels "Excellent"<"Improved"<..: 3 2 1 3
>
```

3. The Empirical Rule

Bell Shape Curve

Normal Distribution

The Empirical Rule



4. Standardized Values

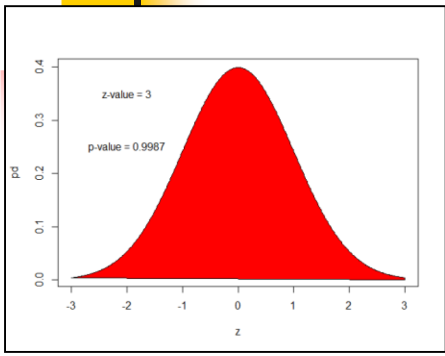
z values

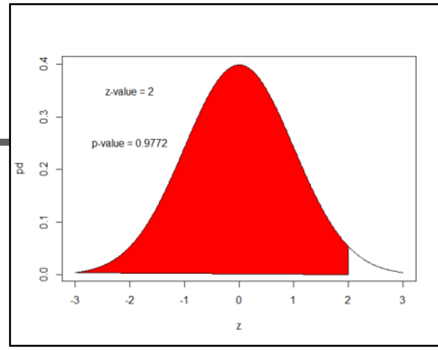
Standardized Values z-value

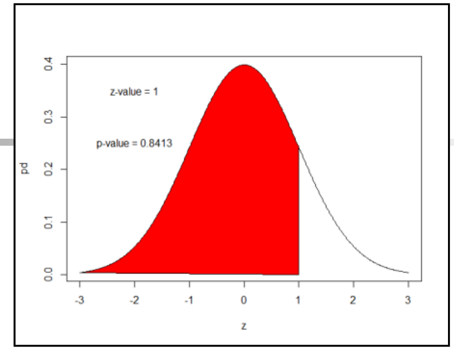
- Compute Probabilities
- Compare two different distributions
  - We compute standardized values
- z = 2
  - Data value is 2 standard deviation above the mean
- z = -1.6
  - Data value is 1.6 standard deviation below the mean

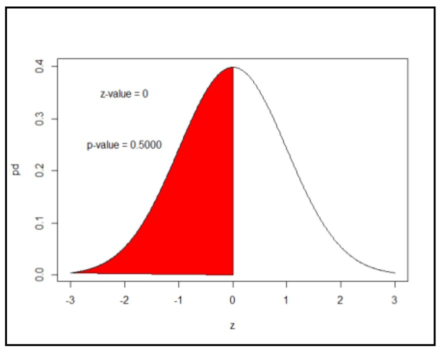
$$z = \frac{\text{Data Value - Mean}}{\text{Standard Deviation}} = \frac{y-\mu}{\sigma}$$

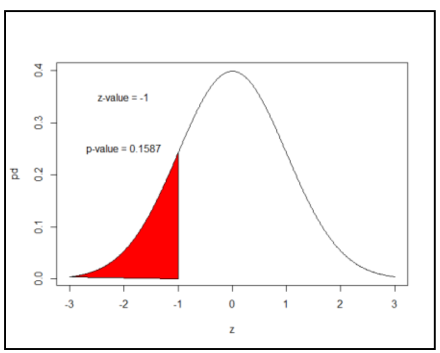
p-values

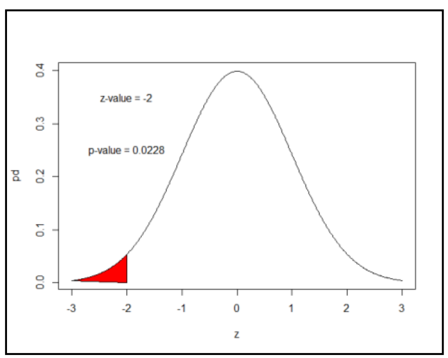


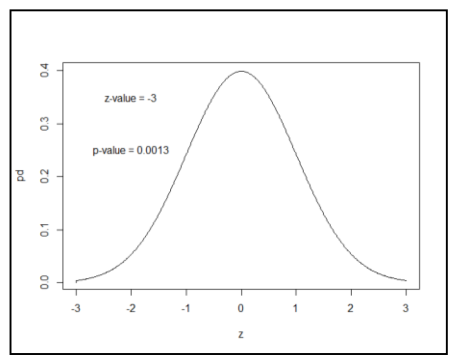








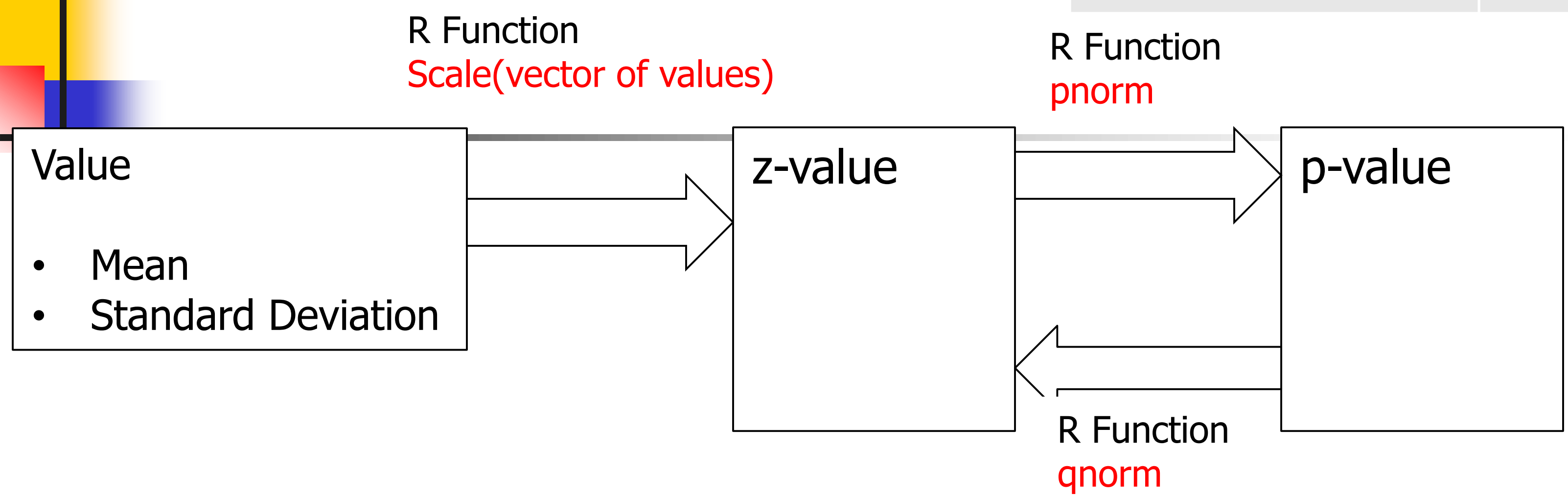




z-values	p-values = area under the curve towards left
3	0.9987
2	0.9772
1	0.8413
0	0.5000
-1	0.1587
-2	0.0228
-3	0.0013

R Functions

Distribution	Normal
Density	dnorm()
Random Numbers	rnorm()




```
> m = 100
> s = 15
> d = 110
> (z1Value <- (d - m)/s)
[1] 0.6666667
> (z2Vaue = scale(d,m,s))
          [,1]
[1,] 0.6666667
attr(,"scaled:center")
[1] 100
attr(,"scaled:scale")
[1] 15
```

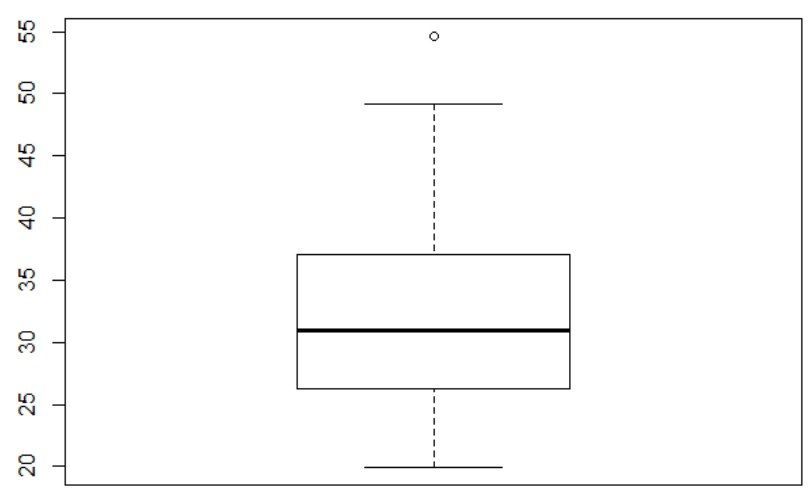
```
> z = 0.21
> (pnorm(z))
[1] 0.5831662
> ################################
> (1-pnorm(z))
[1] 0.4168338
> ###############################
> z1 = -0.31
> z2 = 0.31
> (pnorm(z2) - pnorm(z1))
[1] 0.243439
> ###############################
> a = 0.58
> (qnorm(a))
[1] 0.2018935
```

Five Number Summary: Boxplot

rawData

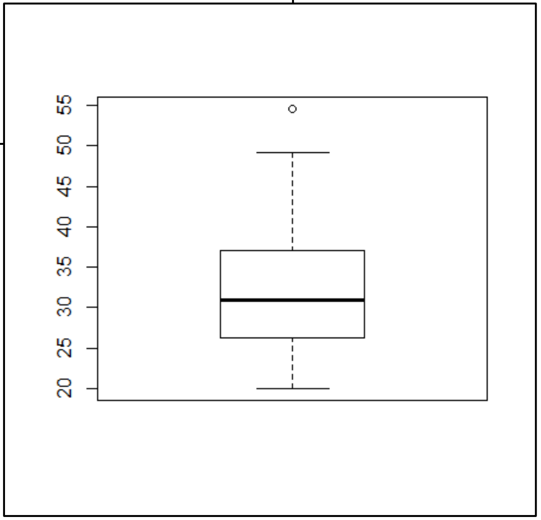
19.95	28.58	33.23	23.25	28.72
33.53	23.32	30.18	36.68	25.55
30.35	37.05	25.83	30.95	37.43
26.28	32.13	41.42	42.47	49.17
54.63				

- Minimum = 19.95
- First Quartile = 26.06
- Median = 30.95
- Third Quartile = 37.24
- Maximum = 49.17
- Outlier = 54.63



Five Number Summary: Boxplot – R Command

```
> ###############################################
> # Boxplot in R
> #
> rawDataCol1.4 = c(19.95, 28.58, 33.23, 23.25, 28.72,
+                   33.53, 23.32, 30.18, 36.68, 25.55, 30.35, 37.05)
> rawDataCol5.7 = c(25.83, 30.95, 37.43, 26.28, 32.13,
+                   41.42, 42.47, 49.17, 54.63)
> (rawData = c(rawDataCol1.4, rawDataCol5.7))
 [1] 19.95 28.58 33.23 23.25 28.72 33.53 23.32 30.18 36.68 25.55 30.35 37.05
[13] 25.83 30.95 37.43 26.28 32.13 41.42 42.47 49.17 54.63
>
> boxplot(rawData)
>
```



5. Data Normalization: Standardization & Scaling

Data Standardization & Scaling

- Suppose we have 2 data items
  - Height: varies from 1 – 7 feet
  - Net Worth: $10,000 - $100B
- If we use both the variables in a model
  - Net Worth will dominate because it contains large values
- Solution
  - Standardize
  - Scale

Data Standardization and Scaling

- Standardization Data Variation
  - -3 to +3

$$z = \frac{\text{Data Value - Mean}}{\text{Standard Deviation}} = \frac{y-\mu}{\sigma}$$

- Scaling Data Variation
  - 0 to 1

$$y_i^j = \frac{x_i^j - min_j}{max_j - min_j}$$

Example: R

```
> normalize = function(x) {
+    return(  (x-min(x))/(max(x)-min(x)))}
> data = c(124,3,311,341,298,136,23,75,5,51,822,364,663,444,999)
> (standard.data = scale(data))
              [,1]
 [1,] -0.603086904
 [2,] -0.994156118
 [3,]  0.001292791
 [4,]  0.098252100
 [5,] -0.040722910
 [6,] -0.564303180
 [7,] -0.929516578
 [8,] -0.761453776
 [9,] -0.987692164
[10,] -0.839021223
[11,]  1.652833026
[12,]  0.172587571
[13,]  1.138948687
[14,]  0.431145729
[15,]  2.224892951
attr(,"scaled:center")
[1] 310.6
attr(,"scaled:scale")
[1] 309.4081
> (normalized.data = normalize(data))
 [1] 0.121485944 0.000000000 0.309236948 0.339357430 0.296184739
0.133534137 0.020080321 0.072289157 0.002008032 0.048192771 0.822289157
[12] 0.362449799 0.662650602 0.442771084 1.000000000
>
```

	A	B	C	D	E	F
1	X		Normalize		Scale	
2	124		-0.624254		0.121486	
3	3		-1.029049		0.000000	
4	311		0.001338		0.309237	
5	341		0.101701		0.339357	
6	298		-0.042152		0.296185	
7	136		-0.584109		0.133534	
8	23		-0.962141		0.020080	
9	75		-0.788180		0.072289	
10	5		-1.022359		0.002008	
11	51		-0.868469		0.048193	
12	822		1.710845		0.822289	
13	364		0.178645		0.362450	
14	663		1.178924		0.662651	
15	444		0.446278		0.442771	
16	999		2.302983		1.000000	
17						
18						

Example: Python

```
import pandas as pd
import numpy as np
df = pd.read_csv('Data.csv')
df
      X
0   124
1     3
2   311
3   341
4   298
5   136
6    23
7    75
8     5
9    51
10  822
11  364
12  663
13  444
14  999

Xarray = np.array(df['X'])
```

Normalize + Scale

$$z = \frac{\text{Data Value - Mean}}{\text{Standard Deviation}} = \frac{y-\mu}{\sigma}$$

```
XNormalize = ( Xarray - np.mean(Xarray))/np.std(Xarray)

XScale = ( Xarray - np.min(Xarray))/(np.max(Xarray) - np.min(Xarray))

#print(XScale)

df['Normalize'] = XNormalize

df['Scale'] = XScale

df
Out[24]:
      X  Normalize     Scale
0   124  -0.624254  0.121486
1     3  -1.029049  0.000000
2   311   0.001338  0.309237
3   341   0.101701  0.339357
4   298  -0.042152  0.296185
5   136  -0.584109  0.133534
6    23  -0.962141  0.020080
7    75  -0.788180  0.072289
8     5  -1.022359  0.002008
9    51  -0.868469  0.048193
10  822   1.710845  0.822289
11  364   0.178645  0.362450
12  663   1.178924  0.662651
13  444   0.446278  0.442771
14  999   2.302983  1.000000
```

$$y_i^j = \frac{x_i^j - min_j}{max_j - min_j}$$

	A	B	C	D	E	F
1	X		Normalize		Scale	
2	124		-0.624254		0.121486	
3	3		-1.029049		0.000000	
4	311		0.001338		0.309237	
5	341		0.101701		0.339357	
6	298		-0.042152		0.296185	
7	136		-0.584109		0.133534	
8	23		-0.962141		0.020080	
9	75		-0.788180		0.072289	
10	5		-1.022359		0.002008	
11	51		-0.868469		0.048193	
12	822		1.710845		0.822289	
13	364		0.178645		0.362450	
14	663		1.178924		0.662651	
15	444		0.446278		0.442771	
16	999		2.302983		1.000000	
17						
18						

Comparison of Standardization and Scaling

- In the presence of outliers in the data
  - Scaling is not effective
  - It will suppress the scaling values of other data elements

Summary

1. Types of Variables
2. Sampling
3. The Empirical Rule
4. Standardized Values
5. Data Normalization: Standardization and Scaling