기상청

[STEP3 데이터 전처리]

# 이상치 및 결측치 처리와 파생변수 생성

기상청

# 기상기후
# 빅데이터 분석 플랫폼

## [분석교육] 결측치 처리 기법 : KNN-Imputation

1. 이상치 처리 함수

2. 결측치 처리 함수

3. 파생변수 생성 함수

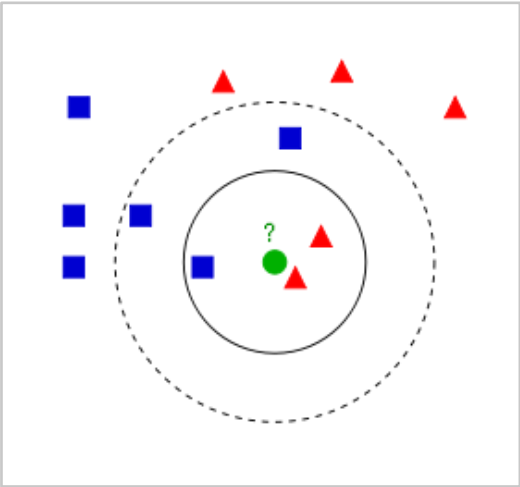
# [분석 교육] 결측치 처리 기법 : KNN-Imputation

**데이터에 결측치가 있는 경우, 값이 존재하지 않아 분석을 수행할 수 없다. 이러한 경우, 결측치를 다른 값으로 대체하여 분석을 수행하게 된다. 결측치를 다른 값으로 바꾸는 것을 대체(Imputation)이라 한다.**

**결측치 대체 방법에는 단일 대체법, 다중 대체법 등이 있다. 이 중 k- 최근접 이웃 알고리즘 (k-NN : k Nearest Neighbor)을 활용한 대체법에 대해 알아본다.**

## k-최근접 이웃 알고리즘

- k- 최근접 이웃 알고리즘(k-NN : k Nearest Neighbor)은 예측하고자 하는 데이터로부터 가장 가까운 k개의 이웃을 찾은 뒤 이들 이웃으로부터 예측하고자 하는 데이터의 분류를 정하는 방법이다.




- 예를 들어, 초록색 원은 파란 사각형이나 빨간 삼각형 두개 중 하나로 분류되어야 한다.
- 만약 k = 3 일 때, 초록색 원의 최근접 이웃은 파란 사각형1개, 빨간 삼각형 2개인 실선 원형 안의 대상이다. 이 때, 초록색 원은 빨간 삼각형 집단으로 분류된다.
- k = 5 이라면, 최근접 이웃은 점선 원형 안의 대상이다. 이때, 초록색 원은 파란 사각형 집단으로 분류된다.

- 최근접 이웃을 찾기 위한 거리 계산에는 유클리디안 거리(Euclidean Distance) 척도를 사용한다. 유클리디안 거리는 두 점 사이의 거리를 구할 때 흔히 사용하는 척도이다.
  - 직교 좌표계로 나타낸 점 **p** = ($p_1$, $p_2$,..., $p_n$)와 **q** = ($q_1$, $q_2$,..., $q_n$)가 있을 때, 두 점 p, q의 거리를 계산하는 공식은 다음과 같다.

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

참고 자료 : 위키백과, R을 이용한 데이터 처리&분석 실무, 서민구, 2014

# 1. 이상치 처리 함수(1/4)

- **data.frame 데이터 처리 형식**
  - 데이터 프레임은 [행, 열] 형태로 데이터에 접근할 수 있음

• **값 업데이트 및 삭제**

- 데이터 프레임에서 특정 조건의 컬럼을 선택하여 값을 업데이트하거나 삭제하는 방법은 다음과 같음

▪ **Usage**

DF[m, n] <- "value"

- m : 행 번호, 행 이름 또는 선택할 행의 제약 조건
- n : 컬럼 번호, 컬럼 이름 또는 선택할 컬럼의 제약 조건
- value : 업데이트 값

▪ **Examples**

DATA[DATA$SUM_SML_EV > 15 | DATA$SUM_SML_EV < 0, "SUM_SML_EV"] <- NA

```
> > # 소형증발량 기후자료 품질검사 알고리즘 기준값 : 0~15 -> NA로 업데이트
> # 업데이트 수행 전 값
> DATA[DATA$SUM_SML_EV > 15 | DATA$SUM_SML_EV < 0, "SUM_SML_EV"]
[1] 15.3 15.3 17.6 21.5 22.5
> # 업데이트 수행
> DATA[DATA$SUM_SML_EV > 15 | DATA$SUM_SML_EV < 0, "SUM_SML_EV"] <- NA
> # 업데이트 수행 후 결과
> DATA[DATA$SUM_SML_EV > 15 | DATA$SUM_SML_EV < 0, "SUM_SML_EV"]
[1] NA NA NA NA NA
```

# 1. 이상치 처리 함수(2/4)

- **data.table 데이터 처리 형식**
  - 데이터 테이블은 데이터 프레임과 유사한 형태로 데이터에 접근할 수 있지만 몇 가지 차이가 있음
  - 데이터 테이블의 접근 형식은 [행, 표현식, 옵션]으로 되어 있음

• **컬럼 선택 방법**

- 데이터 테이블에서는 컬럼이름이나 with = FALSE 옵션을 통해 컬럼 선택이 가능함

▪ **Usage**

DT[i, j(컬럼이름)]
DT[i, j(컬럼번호 또는 문자열), with = FALSE]
DT[i, .SDcols = "colnames"]

- i : 행 번호, 행 이름, 행 표현식
- j : 컬럼 이름, 컬럼 번호, 컬럼 표현식
- with : FALSE면 j를 연산식이 아닌 컬럼 이름 또는 컬럼 번호로 취급함
- .Sdcols : 컬럼 이름 또는 컬럼 번호를 선택함

▪ **Examples**

onion[, y]
onion[, "y", with = FALSE]

```
> onion[, y]
 [1] 6045 6150 5883 6538 6365 6198 7057 6995 8386 7153 7382 6277 8230 7237 6581
[16] 5624 6356 6112 6862 6606 6515 7128 7955 7880 6613 7067 6433 7464 7193 6463
[31] 5544 6010 6130 5824 6029 5592 6976 6561 7395 6390 6417 5441 5730 6319 5923
[46] 4829 5757 5736 5417 5128 5219 5284 5949 6198 5849 5064 5550 5179 6710 5563
[61] 6253 6492 6175 6646 6037 5876 6264 6161 5822 5091 7103 6386 6681 7220 6552
[76] 4350 4881 4838 4718 5462 4475 5258 6137 6313 3990 6276 3936 4191 6628 5707
> onion[, "y", with = FALSE]
      y
 1: 6045
 2: 6150
 3: 5883
 4: 6538
 5: 6365
 6: 6198
 7: 7057
 8: 6995
```

# 1. 이상치 처리 함수(3/4)

- **그룹별 연산**

  - 데이터 테이블에서는 by옵션을 통해 그룹별 연산을 수행할 수 있음

  ▪ **Usage**

  DT[, j, by = “colnames”]
  DT[, j, by = “colnames”, .SDcols = “colnames”]

  - j : 그룹으로 묶어 수행할 연산식
  - by : 그룹으로 묶을 컬럼명
  - .SDcols : 결과로 출력할 컬럼 이름 또는 컬럼 번호

  ▪ **Examples**

  onion[, .SD[which.max(y)], by = region_1, .SDcols="y"]

```
> onion[, .SD[which.max(y)], by = region_1, .SDcols="y"]
   region_1    y
1:      경남 8386
2:      경북 7955
3:      전남 7395
4:      전북 6710
5:      제주 7220
6:      충남 6628
```

- **값 업데이트**

  - 데이터 테이블에서는 := 표현식으로 변수를 새로 만들거나 데이터 값을 업데이트 할 수 있음

  ▪ **Usage**

  DT[i, colnames := “value”]

  - i : 업데이트 할 행 번호 또는 행 조건
  - := : 업데이트 할 (새로운)컬럼과 업데이트 값

  ▪ **Examples**

  onion.area[is.na(area.replace), area.replace := 0]

```
> #area값에서 파생된 area.replace라는 새로운 변수 만들기
> onion.area[, area.replace := as.numeric(area)]
> head(onion.area,2)
   STN_ID region_1 region_2    area_id area     w area.replace
1:    285      경남      합천 4800000000 1205 0.403         1205
2:    295      경남      남해 4800000000   NA 0.000           NA
> #NA -> 0으로 값 업데이트
> onion.area[is.na(area.replace), area.replace := 0]
> #업데이트 결과 확인
> head(onion.area,2)
   STN_ID region_1 region_2    area_id area     w area.replace
1:    285      경남      합천 4800000000 1205 0.403         1205
2:    295      경남      남해 4800000000   NA 0.000            0
```

# 1. 이상치 처리 함수(4/4)

- **which.max**

  - 최대값의 위치를 반환

  ▪ **Usage**
  - y : 숫자 벡터

  ▪ **Examples**
  onion[, .SD[which.max(y)], by = region_1, .SDcols="y"]

```
> onion[, .SD[which.max(y)], by = region_1, .SDcols="y"]
   region_1    y
1:     경남 8386
2:     경북 7955
3:     전남 7395
4:     전북 6710
5:     제주 7220
6:     충남 6628
```

- **which.min**

  - 최소값의 위치를 반환

  ▪ **Usage**
  - y : 숫자 벡터

  ▪ **Examples**
  onion[, .SD[which.min(y)], by = region_1, .SDcols="y"]

```
> onion[, .SD[which.min(y)], by = region_1, .SDcols="y"]
   region_1    y
1:     경남 5883
2:     경북 5624
3:     전남 5441
4:     전북 4829
5:     제주 5091
6:     충남 3936
```

# 2. 결측치 처리 함수

- **is.na**

  **- 결측치를 확인함**

  ▪ **Usage**

  is.na(…)

  - DATA : 결측치를 확인할 객체

  ▪ **Examples**

  head(is.na(DATA))  # 결측치가 존재하는 일부 데이터 확인

  sum(is.na(DATA))  # 결측치 개수 확인

```
> head(is.na(DATA))
     STN_ID    TM TA_AVG TA_MAX TA_MIN SUM_RN WS_AVG WS_MAX HM_AVG HM_MIN SUM_SS SUM_SI TD_AVG PV_AVG
[1,]  FALSE FALSE   FALSE   FALSE   FALSE   TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
[2,]  FALSE FALSE   FALSE   FALSE   FALSE   TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
[3,]  FALSE FALSE   FALSE   FALSE   FALSE   TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
[4,]  FALSE FALSE   FALSE   FALSE   FALSE   TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
[5,]  FALSE FALSE   FALSE   FALSE   FALSE   TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
[6,]  FALSE FALSE   FALSE   FALSE   FALSE   TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
     PS_MIN SD_HR3_MAX SD_TOT_MAX CA_TOT_AVG CA_MID_AVG TS_AVG TS_MAX TS_MIN SUM_SML_EV    HT
[1,]  FALSE        TRUE        TRUE       FALSE       FALSE  FALSE  FALSE  FALSE       FALSE FALSE
[2,]  FALSE        TRUE        TRUE       FALSE       FALSE  FALSE  FALSE  FALSE       FALSE FALSE
[3,]  FALSE        TRUE        TRUE       FALSE       FALSE  FALSE  FALSE  FALSE       FALSE FALSE
[4,]  FALSE        TRUE        TRUE       FALSE       FALSE  FALSE  FALSE  FALSE       FALSE FALSE
[5,]  FALSE        TRUE        TRUE       FALSE       FALSE  FALSE  FALSE  FALSE       FALSE FALSE
[6,]  FALSE        TRUE        TRUE       FALSE       FALSE  FALSE  FALSE  FALSE       FALSE FALSE
> sum(is.na(DATA))
[1] 153710
```

- **DMwR::knnImputation**

  **- KNN-Imputation을 수행하여 결측치를 대체함**

  ▪ **Usage**

  knnImputation(data, k = …, distData = …, …)

  - data : 결측이 있는 데이터 프레임
  - k : 참조할 이웃의 수
  - distData : 이웃을 참조할 데이터 프레임을 따로 지정, 기본값은 NULL이며, 이웃을 data 안에서 찾음

  ▪ **Examples**

  DATA <- knnImputation(DATA, k=10)

```
> #SUM_SI, SUM_SML_EV는 Knnimputation기법을 활용하여 결측치 처리
> sum(is.na(DATA)) #결측치 갯수 확인
[1] 48
> DATA <- knnImputation(DATA, k=10)
> sum(is.na(DATA)) #결측치 대체된것 확인
[1] 0
>
```

# 3. 파생변수 생성(1/4)

- **plyr::ddply**

  **- 데이터 프레임을 분할하고 함수를 적용하여 결과를 데이터 프레임으로 반환**

  ▪ **Usage**

  ddply(data, variables, fun, …)

  - data : 분할하여 함수를 적용할 데이터
  - variables : 데이터를 그룹 지울 변수
  - fun : 적용할 함수

  ▪ **Examples**

```
TA_RN_SS_Month <- ddply(TA_RN_SS, c("STN_ID", "YM"), summarise,
      TAD=mean(AVG_TA, na.rm=TRUE), TAmax=max(MAX_TA, na.rm=TRUE),
      TAmin=min(MIN_TA,na.rm=TRUE),RAINSUM=sum(SUM_RN,na.rm=TRUE),
      DTD=mean(DTD, na.rm=TRUE), SUM_SS_HR=mean(SUM_SS_HR, na.rm=TRUE))
```

- **plyr::summarise**

  **- 데이터 프레임을 요약**

  ▪ **Usage**

  summarise(data, …)

  - data : 요약할 데이터 프레임
  - … : '변수 = 값' 형태의 인자, 그룹마다 요약을 수행한 뒤 새로운 컬럼으로 추가하여 데이터 프레임을 반환

  ▪ **Examples**

```
TA_RN_SS_Month <- ddply(TA_RN_SS, c("STN_ID", "YM"), summarise,
      TAD=mean(AVG_TA, na.rm=TRUE), TAmax=max(MAX_TA, na.rm=TRUE),
      TAmin=min(MIN_TA,na.rm=TRUE),RAINSUM=sum(SUM_RN,na.rm=TRUE),
      DTD=mean(DTD, na.rm=TRUE), SUM_SS_HR=mean(SUM_SS_HR, na.rm=TRUE))
```

```
> # mean, max, min, sum
> TA_RN_SS_Month <- ddply(TA_RN_SS, c("STN_ID", "YM"), summarise,
+             TAD=mean(AVG_TA, na.rm=TRUE), TAmax=max(MAX_TA, na.rm=TRUE),
+             TAmin=min(MIN_TA, na.rm=TRUE), RAINSUM=sum(SUM_RN, na.rm=TRUE),
+             DTD=mean(DTD, na.rm=TRUE), SUM_SS_HR=mean(SUM_SS_HR, na.rm=TRUE))
> str(TA_RN_SS_Month)
'data.frame':   4992 obs. of  8 variables:
 $ STN_ID   : chr  "129" "129" "129" "129" ...
 $ YM       : chr  "2000-01" "2000-02" "2000-03" "2000-04" ...
 $ TAD      : num  -0.552 -2.572 3.857 9.49 15.019 ...
 $ TAmax    : num  21.4 7.4 17.2 21.4 29.6 30.9 32.2 32.7 29.7 26.5 ...
 $ TAmin    : num  -13.3 -12.9 -9.8 -4 4.5 11.7 12.9 17.7 9.2 -0.3 ...
 $ RAINSUM  : num  58 0.5 2 36.5 71.5 215 61.5 647 323 33.2 ...
 $ DTD      : num  8.11 10.43 13.66 12.56 10.23 ...
 $ SUM_SS_HR: num  4.08 6.66 7.35 7.72 6.7 ...
```

# 3. 파생변수 생성(2/4)

- **mean**

  - 평균을 계산

  ▪ **Usage**
  mean(x, na.rm = TRUE)
  - x : 평균할 데이터
  - na.rm : 평균 계산 전 NA 제거 여부, TRUE면 제거

  ▪ **Examples**
  mean(AVG_TA, na.rm=TRUE)

- **min**

  - 최솟값을 계산

  ▪ **Usage**
  min(x, na.rm = TRUE)
  - x : 최솟값을 계산할 데이터
  - na.rm : 최솟값 계산 전 NA 제거 여부, TRUE면 제거

  ▪ **Examples**
  min(MIN_TA, na.rm=TRUE)

- **max**

  - 최댓값을 계산

  ▪ **Usage**
  max(x, na.rm = TRUE)
  - x : 최댓값을 계산할 데이터
  - na.rm : 최댓값 계산 전 NA 제거 여부, TRUE면 제거

  ▪ **Examples**
  max(MAX_TA, na.rm=TRUE)

- **sum**

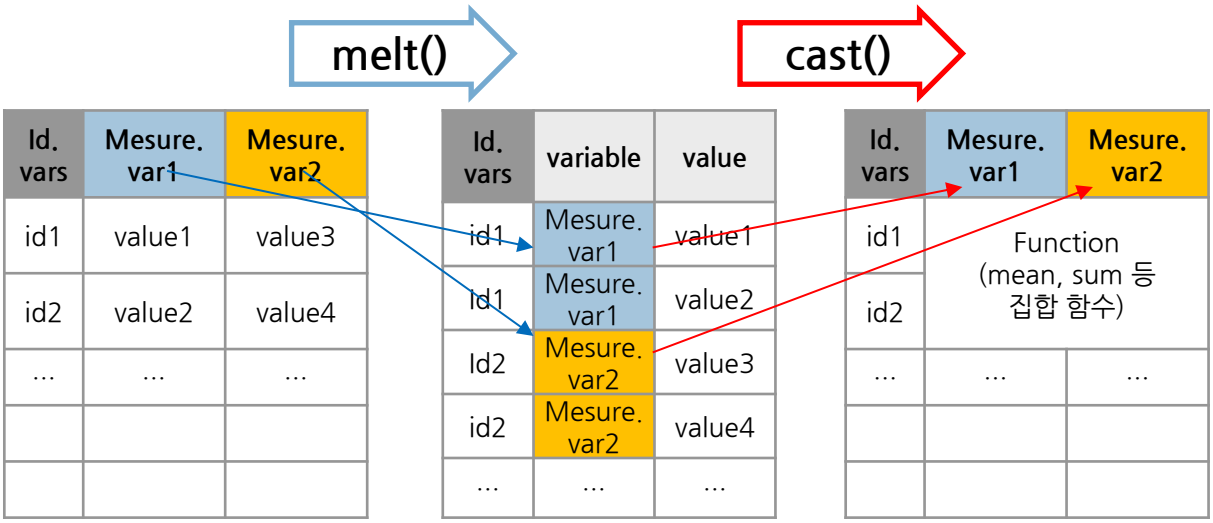
  - 합계를 계산

  ▪ **Usage**
  sum(x, na.rm = TRUE)
  - x : 합계를 계산할 데이터
  - na.rm : 합계 계산 전 NA 제거 여부, TRUE면 제거

  ▪ **Examples**
  sum(SUM_RN, na.rm=TRUE)

# 3. 파생변수 생성(3/4)

- **melt()와 cast() 함수를 활용한 데이터 재구성**




- **reshape::melt**

  - 여러 컬럼으로 구성된 데이터를 데이터 식별자(id), 측정 변수(variable), 측정값(value) 형태의 데이터로 변환

▪ **Usage**

melt(data, id.vars = …, mesure.vars = …)

- data : 함수를 적용할 데이터
- id.vars : 식별자 컬럼들
- mesure.vars : 측정치 컬럼들, 생략 시 id.vars를 제외한 모든 컬럼이 해당됨

▪ **Examples**

```
Weather_Area_Region <- melt(Weather_Area_Region,
                            id.var=c("area_id", "region_1", "region_2", "YM", "area"))
```

```
> > Weather_Area_Region <- melt(Weather_Area_Region, id.var=c("area_id", "region_1", "region_2", "YM", "area"))
> str(Weather_Area_Region)
'data.frame':   25344 obs. of  7 variables:
 $ area_id : chr  "4400000000" "4400000000" "4400000000" "4400000000" ...
 $ region_1: chr  "충남" "충남" "충남" "충남" ...
 $ region_2: chr  "서산" "서산" "서산" "서산" ...
 $ YM      : chr  "2000-01" "2000-02" "2000-03" "2000-04" ...
 $ area    : num  0 0 0 0 0 0 0 0 0 0 ...
 $ variable: Factor w/ 6 levels "TAD","TAmax",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ value   : num  -0.276 -1.286 1.928 4.745 7.51 ...
```

# 3. 파생변수 생성(4/4)

- **reshape::cast**

  - melt()된 데이터를 다시 여러 컬럼으로 변환

▪ **Usage**

cast(data, id.vars ~variable.vars, fun)

- data : 함수를 적용할 데이터
- formula : id.vars ~ variable.vars
- fun : 여러 행을 재구성 할 시 사용할 집합 함수

▪ **Examples**

```
Weather_Area_Region <- cast(Weather_Area_Region,
                            id.var=c("area_id", "region_1", "region_2", "YM", "area"))
```

```
> Weather_Area_Region <- cast(Weather_Area_Region, area_id+region_1+region_2+year+area~variable+m
> str(Weather_Area_Region)
List of 77
 $ area_id     : chr [1:352] "4400000000" "4400000000" "4400000000" "4400000000" ...
 $ region_1    : chr [1:352] "충남" "충남" "충남" "충남" ...
 $ region_2    : chr [1:352] "서산" "서산" "서산" "서산" ...
 $ year        : chr [1:352] "2000" "2001" "2002" "2003" ...
 $ area        : num [1:352] 0 0 0 0 0 0 0 0 0 0 ...
 $ TAD_01      : num [1:352] -0.276 -1.948 0.145 -1.768 -0.934 ...
 $ TAD_02      : num [1:352] -1.286 -0.421 0.157 0.448 0.81 ...
 $ TAD_03      : num [1:352] 1.93 1.69 2.92 2.28 2.43 ...
 $ TAD_04      : num [1:352] 4.74 5.51 6.16 5.93 5.53 ...
 $ TAD_05      : num [1:352] 7.51 8.74 8.14 9.11 8.3 ...
 $ TAD_06      : num [1:352] 10.4 10.7 10.5 10.4 11 ...
 $ TAD_07      : num [1:352] 12.5 12.7 12.3 11.8 12.4 ...
 $ TAD_08      : num [1:352] 12.6 12.8 12 11.9 13 ...
 $ TAD_09      : num [1:352] 9.58 10.55 10.11 10.25 10.64 ...
 $ TAD_10      : num [1:352] 6.87 7.48 6.01 6.58 7.12 ...
 $ TAD_11      : num [1:352] 2.86 3.02 1.86 4.45 4.02 ...
 $ TAD_12      : num [1:352] 0.265 -0.36 0.527 0.46 1.023 ...
 $ TAmax_01    : num [1:352] 10.7 3 8.85 4.4 4.9 5.05 5.55 5.5 3.6 5.5 ...
 $ TAmax_02    : num [1:352] 3.7 6.4 6.7 7 9.6 5.45 5.55 7.65 5.3 8.8 ...
 $ TAmax_03    : num [1:352] 8.6 10.5 9.65 9.8 10.4 ...
 $ TAmax_04    : num [1:352] 10.7 13.1 14 12.2 13.6 ...
 $ TAmax_05    : num [1:352] 14.8 14.2 13.9 15.4 13.6 ...
 $ TAmax_06    : num [1:352] 15.4 15.9 16.2 15.2 16.2 ...
 $ TAmax_07    : num [1:352] 16.1 16.5 17.6 16.4 17.5 ...
```


본 문서의 내용은 기상청의 날씨마루(http://big.kma.go.kr) 내
R 프로그래밍 교육 자료입니다.