HW3 Mike Turley CSC587-W1

1. Data preprocessing on the age list

We are given the following sorted ages:

```
ages <-
c(13,15,16,16,19,20,20,21,22,22,25,25,25,25,30,33,33,35,35,35,35,36,40
,45,46,52,70)
ages
```

```
##  [1] 13 15 16 16 19 20 20 21 22 22 25 25 25 25 30 33 33 35 35 35 35
36 40 45 46
## [26] 52 70
```

```
length(ages)
```

```
## [1] 27
```

```
range(ages)
```

```
## [1] 13 70
```

1(a). Smoothing by bin means (bin size = 3)

There are 27 values, so we can form 9 bins of size 3. Within each bin, replace values with the bin mean.

```
bin_means <- function(x, k = 3){
  stopifnot(length(x) %% k == 0)
  out <- numeric(length(x))
  for(i in seq(1, length(x), by = k)){
    m <- mean(x[i:(i+k-1)])
    out[i:(i+k-1)] <- m
  }
  out
}

smoothed <- round(bin_means(ages, 3), 2)
smoothed
```

```
##  [1] 14.67 14.67 14.67 18.33 18.33 18.33 21.00 21.00 21.00 24.00
24.00 24.00
```

```
## [13] 26.67 26.67 26.67 33.67 33.67 33.67 35.00 35.00 35.00 40.33
40.33 40.33
## [25] 56.00 56.00 56.00
```

Comment: Smoothing reduces within-bin noise, preserves coarseness trend that sacrifices fine granularity.

1(b). Outlier detection via IQR rule

Compute quartiles, IQR, and fences; flag outliers outside the fences.

```
# Use type=2 (median of averages of order statistics), a common
convention in textbooks
Q1 <- as.numeric(quantile(ages, 0.25, type = 2))
Q3 <- as.numeric(quantile(ages, 0.75, type = 2))
IQRv <- Q3 - Q1
lower <- Q1 - 1.5 * IQRv
upper <- Q3 + 1.5 * IQRv
outliers <- ages[ages < lower | ages > upper]

data.frame(Q1, Q3, IQR = IQRv, lower_fence = lower, upper_fence =
upper, outliers = I(toString(outliers)))
```

```
##   Q1 Q3 IQR lower_fence upper_fence outliers
## 1 20 35  15        -2.5        57.5       70
```

Interpretation: Values greater than the upper fence (or less than the lower fence) are outliers under the 1.5×IQR rule.

1(c). Min–max normalization of age = 35 to [0, 1]

```
min_age <- min(ages); max_age <- max(ages)
norm_35 <- (35 - min_age) / (max_age - min_age)
norm_35
```

```
## [1] 0.3859649
```

1(d). z-score normalization of age = 35

We compute the **population** standard deviation by default.

```
mu <- mean(ages)
sigma_pop <- sqrt(mean((ages - mu)^2))     # population SD
sigma_samp <- sd(ages)                      # sample SD, for reference
```

```
z35_pop  <- (35 - mu) / sigma_pop
z35_samp <- (35 - mu) / sigma_samp
data.frame(mu, sigma_pop, sigma_samp, z35_pop, z35_samp)

##         mu sigma_pop sigma_samp  z35_pop  z35_samp
## 1 29.96296  12.70019   12.94212 0.396611 0.3891971
```

> Report the population-based z-score unless your instructor specifies the sample version.

1(e). Decimal scaling normalization of age = 35

Find the smallest integer j such that all scaled values are in $[-1, 1)$. Here the largest $|x|$ is 70, so $j = 2$.

```
decimal_scaled_35 <- 35 / 10^2
decimal_scaled_35

## [1] 0.35
```

---

2. General min–max normalization function to an arbitrary range

For any vector a and target range $[L, U]$, the formula is:
$$x' = L + \frac{(x - \min a)}{\max a - \min a}(U - L).$$

```
minmax_scale <- function(a, L, U) {
  a <- as.numeric(a)
  amin <- min(a); amax <- max(a)
  if (amax == amin) return(rep((L + U)/2, length(a)))  # constant
vector edge case
  L + (a - amin) * (U - L) / (amax - amin)
}

# Examples on ages
scaled_0_1 <- minmax_scale(ages, 0, 1)
scaled_5_10 <- minmax_scale(ages, 5, 10)
```

```r
list(`[0,1]` = head(round(scaled_0_1, 4), 10),
     `[5,10]` = head(round(scaled_5_10, 4), 10))
```

```
## $`[0,1]`
##  [1] 0.0000 0.0351 0.0526 0.0526 0.1053 0.1228 0.1228 0.1404 0.1579
0.1579
##
## $`[5,10]`
##  [1] 5.0000 5.1754 5.2632 5.2632 5.5263 5.6140 5.6140 5.7018 5.7895
5.7895
```

---

3. Two-level decision tree using **Information Gain** (class label = `status`)

The dataset is given as aggregated counts over attributes **department**, **age**, **salary**, and class **status**.

```r
library(dplyr)
library(tidyr)
library(purrr)

# Aggregated table from assignment
dat <- tribble(
  ~department, ~age,    ~salary,  ~status,  ~count,
  "sales",     "31-35", "46-50K", "senior", 30,
  "sales",     "26-30", "26-30K", "junior", 40,
  "sales",     "31-35", "31-35K", "junior", 40,
  "systems",   "21-25", "46-50K", "junior", 20,
  "systems",   "31-35", "66-70K", "senior", 5,
  "systems",   "26-30", "46-50K", "junior", 3,
  "systems",   "41-45", "66-70K", "senior", 3,
  "marketing", "36-40", "46-50K", "senior", 10,
  "marketing", "31-35", "41-45K", "junior", 4,
  "secretary", "46-50", "36-40K", "senior", 4,
  "secretary", "26-30", "26-30K", "junior", 6
)

# Helper: entropy base-2 for a vector of class counts
```

```r
H2 <- function(n) {
  n <- n[n > 0]
  p <- n / sum(n)
  -sum(p * log2(p))
}

# Root entropy H(Y)
root_counts <- dat %>% group_by(status) %>% summarise(n =
sum(count), .groups = "drop")
H_root <- H2(root_counts$n)
N <- sum(root_counts$n)
list(H_root = H_root, N = N, counts = root_counts)
```

```
## $H_root
## [1] 0.8990308
##
## $N
## [1] 165
##
## $counts
## # A tibble: 2 × 2
##   status     n
##   <chr>  <dbl>
## 1 junior   113
## 2 senior    52
```

```r
# Information Gain calculator for a given attribute
IG_of <- function(attribute) {
  by_attr <- dat %>%
    group_by(!!sym(attribute), status) %>%
    summarise(n = sum(count), .groups = "drop") %>%
    group_by(!!sym(attribute)) %>%
    summarise(H = H2(n), w = sum(n)/N, .groups = "drop")
  IG <- H_root - sum(by_attr$w * by_attr$H)
  list(attribute = attribute, table = by_attr, IG = IG)
}

ig_dept   <- IG_of("department")
ig_age    <- IG_of("age")
ig_salary <- IG_of("salary")
```

```
ig_dept$IG; ig_age$IG; ig_salary$IG

## [1] 0.04860679

## [1] 0.4247351

## [1] 0.5375181

ig_dept$table; ig_age$table; ig_salary$table

## # A tibble: 4 × 3
##   department     H      w
##   <chr>      <dbl>  <dbl>
## 1 marketing  0.863 0.0848
## 2 sales      0.845 0.667
## 3 secretary  0.971 0.0606
## 4 systems    0.824 0.188

## # A tibble: 6 × 3
##   age       H      w
##   <chr> <dbl>  <dbl>
## 1 21-25 0     0.121
## 2 26-30 0     0.297
## 3 31-35 0.991 0.479
## 4 36-40 0     0.0606
## 5 41-45 0     0.0182
## 6 46-50 0     0.0242

## # A tibble: 6 × 3
##   salary     H      w
##   <chr>  <dbl>  <dbl>
## 1 26-30K 0     0.279
## 2 31-35K 0     0.242
## 3 36-40K 0     0.0242
## 4 41-45K 0     0.0242
## 5 46-50K 0.947 0.382
## 6 66-70K 0     0.0485
```

The largest information gain is for **salary**, so salary is selected as the **root split**. All salary branches are pure **except** the 46–50K branch. We split that node on **department** (or **age**; both produce pure leaves).

```r
# Inspect the impurity of the 46-50K branch by department
branch <- dat %>% filter(salary == "46-50K")
branch_by_dept <- branch %>% group_by(department, status) %>%
summarise(n = sum(count), .groups = "drop")
branch_by_dept %>%
  group_by(department) %>%
  summarise(H = H2(n), n_branch = sum(n), .groups = "drop")
```

```
## # A tibble: 3 × 3
##   department     H n_branch
##   <chr>      <dbl>    <dbl>
## 1 marketing      0       10
## 2 sales          0       30
## 3 systems        0       23
```

Chosen two-level tree:

- **Root:** `salary`
  - `26-30K` → **junior**
  - `31-35K` → **junior**
  - `36-40K` → **senior**
  - `41-45K` → **junior**
  - `66-70K` → **senior**
  - `46-50K` → split on **department**
    - `sales` → **senior**
    - `systems` → **junior**
    - `marketing` → **senior**

---

4. If–then rules from the tree

1. **IF** salary ∈ `26-30K` **THEN** status = **junior**.

2. **IF** salary ∈ `31-35K` **THEN** status = **junior**.

3. **IF** salary ∈ `36-40K` **THEN** status = **senior**.

4. **IF** salary ∈ `41-45K` **THEN** status = **junior**.
5. **IF** salary ∈ `66-70K` **THEN** status = **senior**.
6. **IF** salary ∈ `46-50K` **AND** department = `sales` **THEN** status = **senior**.
7. **IF** salary ∈ `46-50K` **AND** department = `systems` **THEN** status = **junior**.
8. **IF** salary ∈ `46-50K` **AND** department = `marketing` **THEN** status = **senior**.