homework 3

Feroz Sheik

2024-02-15

Question 1

age: 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

a.

```
# Loading required libraries
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# Defining the age data
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)

# Defining bin depth
bin_depth <- 3

# Calculating number of bins
num_bins <- length(ages) %/% bin_depth

# Creating empty vector to store smoothed ages
smoothed_ages <- numeric(0)

# Performing smoothing by bin means
for (i in 1:num_bins) {
  bin_start <- (i - 1) * bin_depth + 1
  bin_end <- min(bin_start + bin_depth - 1, length(ages))
  bin_mean <- mean(ages[bin_start:bin_end])
  smoothed_ages <- c(smoothed_ages, rep(bin_mean, bin_depth))
}

# Printing smoothed ages
print(smoothed_ages)
```

```
##  [1] 14.66667 14.66667 14.66667 18.33333 18.33333 18.33333 21.00000 21.00000
##  [9] 21.00000 24.00000 24.00000 24.00000 26.66667 26.66667 26.66667 33.66667
## [17] 33.66667 33.66667 35.00000 35.00000 35.00000 40.33333 40.33333 40.33333
## [25] 56.00000 56.00000 56.00000
```

This code splits the bucket widths into bins of depth 3 for each age bin and substitutes each bucket element with its mean value. The end product is the smoothed age figures which are displayed on the screen.

Here, the procedure is such that the spread is kept under check by replacing the median of the three consecutive ages (from which the trend is derived) with their mean. On the one hand, such averaging will make the data looks smoother and could potentially simplify the analysis of the data. On the other hand, there is a high probability that some detail or distortion will be lost, or the data may not be as precise as it should be especially when there are sudden changes or extreme values in the data.

This code enacts the smoothing procedure on a binary mean on the set of given ages. The method does this age division into the bins of certain depth, and, where each bin holds certain values, the analysis replaces them with the mean of the given values.

Initialization: The first line of the code sets up the vector smoothed_ages to put the smoothed age values in later.

Bin Calculation: It determines the number of bins to be used taking into consideration the bucket length and the depth of each bin. Each bin also subsets the age category of people.

Smoothing: Every bin goes through this calculation, and the code is finding the mean of the age groups that are in the bin. Next is binning of the ages using the original mean value outputted by FINALAGE.

Output: The emerged smoothed_ages vector now stores the mean age values within the corresponding bins. Consequently, each value indicates the age within its respective bin. The ages that have undergone the smoothing process forming a group of three values with the mean age being the average age for each bin. For instance, consider the figure (14.67) which means that age 13, 15, 16 had been replaced by their average value. By doing this, curve smoothing tends to take away the roughness and to stabilize the data giving a clearer picture of the current trend in the model.

b.

```
# Calculating the quartiles
q1 <- quantile(ages, 0.25)
q3 <- quantile(ages, 0.75)

# Calculating the interquartile range (IQR)
iqr <- q3 - q1

# Calculating the lower and upper bounds for outliers
lower_bound <- q1 - 1.5 * iqr
upper_bound <- q3 + 1.5 * iqr

# Identifing outliers
outliers <- ages[ages < lower_bound | ages > upper_bound]

# Printing results
if (length(outliers) == 0) {
  print("No outliers found in the data.")
} else {
  print("Outliers found in the data:")
  print(outliers)
}
```

```
## [1] "Outliers found in the data:"
## [1] 70
```

The function selects the lower and upper bounds for outliers using the IQR method and then gets to the data points in the entire dataset that are outside these two bounds. If there is no outliers, it shows “No outliers found in the data.” But if it found an outlier, it will print these outliers.

In the code, the IQR method, designed to spot outliers in ages dataset, is employed. The code computes the first-quartile (Q1) and the third-quartile (Q3) of the age data with the quantile() function. The limits of under- and upper- bounds may be determined by multiplying the difference of Q1 and Q3 by 1.5 and from there subtracting and adding the results to Q1 and Q3, accordingly. If for any part of age range the values are lower than the lower bound or above the upper bound these values will be stored as outliers into the outlier vector.

In the proposed code snippet, it appears like 70 is an outlier point. In other words, 70 is a value that is represented in the relationship of ages, and it deviates considerably more from the upper boundary, which is calculated using the IQM method. In the case, the latter falls outside the parameters and is thus marked as the outlier. This technique is quite often employed for revealing the anomalies or outliers in data, which conduct further analysis and to extract some valuable insights from the data.

c.

```
# Defining the value to be normalized
age <- 35

# Defining the minimum and maximum values of the age data
min_age <- min(ages)
max_age <- max(ages)

# Performing min-max normalization
normalized_age <- (age - min_age) / (max_age - min_age)

# Printing the normalized age
print(normalized_age)
```

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

This unit does min-max normalization for the age value of 35 as compared with the minimum and maximum data of age that were given before. Thus is calculated the result between 0.0 and 1.0, which signifies the position of 35 years-old within the range of ages.

The max-min normalization of the feature regarding age with a value of 35 is performed. Regressing age to a range between 0 and 1 by applying normalization technique means using minimum and maximum values within the data. Here, the age value of 35 is addressed by reducing its minimum age value and dividing the result by the difference between the maximum and minimum age. Afterwards, this value reduced by normalization to around 0.386. Min-max normalization works well in bringing features of all measures under the common scale so that they may be comparable and “anti-model-bias” models will not be influenced by features scaling. Provides for the conservation of the binds between the data items while assuring that remain aligned with the given limit range.

d.

```
# Defining the value to be normalized
age <- 35

# Calculating the mean and standard deviation of the age data
mean_age <- mean(ages)
sd_age <- sd(ages)

# Performing z-score normalization
z_score <- (age - mean_age) / sd_age

# Printing the z-score
print(z_score)
```

```
## [1] 0.3891971
```

This code evaluates z-score normalization of the age value 35 using the mean and standard deviation calculated for mean and standard deviation of the age data provided previously. Thus, the z-score that is equivalent to the value of 35 would be obtained, showing if the age is divided by standard deviation.

e.

```
# Defining the value to be normalized
age <- 35

# Finding the maximum absolute value in the dataset
max_abs <- max(abs(ages))

# Determining the power of 10 needed to scale the maximum absolute value to be less than 1
power_of_10 <- floor(log10(max_abs))

# Performing normalization by dividing the age by the power of 10
normalized_age <- age / (10^power_of_10)

# Printing the normalized age
print(normalized_age)
```

```
## [1] 3.5
```

This code refers to the process of normalization and calculates the decimal scaling using the value 35 for the age by determining the power of 10 needed to scale the maximum absolute value in the dataset to be less than 1. Power of 10 is taken as base first. Then age is divided by this power of 10 to normalize it.

Question 2

```
min_max_normalization <- function(a, min_new, max_new) {
  # Calculating the minimum and maximum values of the input data
  min_old <- min(a)
  max_old <- max(a)

  # Performing min-max normalization
  normalized <- (a - min_old) * (max_new - min_new) / (max_old - min_old) + min_new

  return(normalized)
}

# Example usage:
# Defining the input data
age_data <- 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)

# Defining the desired output range
min_output <- 0
max_output <- 1

# Performing min-max normalization
normalized_age <- min_max_normalization(age_data, min_output, max_output)

# Printing the normalized data
print(normalized_age)
```

```
##  [1] 0.00000000 0.03508772 0.05263158 0.05263158 0.10526316 0.12280702
##  [7] 0.12280702 0.14035088 0.15789474 0.15789474 0.21052632 0.21052632
## [13] 0.21052632 0.21052632 0.29824561 0.35087719 0.35087719 0.38596491
## [19] 0.38596491 0.38596491 0.38596491 0.40350877 0.47368421 0.56140351
## [25] 0.57894737 0.68421053 1.00000000
```

This function min_max_normalization takes three arguments: a data vector a, a minimum value min_new, and a maximum value max_new that you wish to have for the output range. It gives back data which is the normalized map but to the specified range.

it defines a function null min_max_normalization to normalize that data to a standard range. Normalization Formula: Pseudo-code of min-max normalization comes first with each data vector element a. For each element x in a, the formula resembles (x - min_old) * (max_new - min_new) / (max_old - min_old) + min_new will be applied to scale x as the source range will.

Output: This function is responsible for scaling the data input vector into a new vector full of values scaled by the input data vector scale values. As illustrated, the furnace function is demonstrated using the age_data vector as an input, with a desired output range between 0 (min) and 1 (max). This is the end result of this process and the age column shows the values corresponding to min-max normalized age values.

Through this normalization, the data values are transformed so as to fall in a fixed range, let’s say from [0, 1] or [-1, 1]. This standardization technique, however, serves to make sure that all features have the same scale, which helps to achieve better accuracy of algorithms and prevents presence of features that are exactly in different scale in their operation.

Question 3

```
## Calculating Information Gain for Decision Tree

# Total counts
total_junior <- 113
total_senior <- 52
total_count <- total_junior + total_senior

# Entropy for the entire dataset
entropy_D <- -((total_senior/total_count) * log2(total_senior/total_count) + (total_junior/total_count) * log2(to
tal_junior/total_count))

# Department counts
sales_junior <- 80
systems_junior <- 23
marketing_junior <- 4
secretary_junior <- 6
sales_senior <- 30
systems_senior <- 8
marketing_senior <- 10
secretary_senior <- 4

# Entropy for Department
entropy_sales <- -((sales_senior/(sales_senior + sales_junior)) * log2(sales_senior/(sales_senior + sales_junio
r)) + (sales_junior/(sales_senior + sales_junior)) * log2(sales_junior/(sales_senior + sales_junior)))
entropy_systems <- -((systems_senior/(systems_senior + systems_junior)) * log2(systems_senior/(systems_senior + s
ystems_junior)) + (systems_junior/(systems_senior + systems_junior)) * log2(systems_junior/(systems_senior + syst
ems_junior)))
entropy_marketing <- -((marketing_senior/(marketing_senior + marketing_junior)) * log2(marketing_senior/(marketin
g_senior + marketing_junior)) + (marketing_junior/(marketing_senior + marketing_junior)) * log2(marketing_junior/
(marketing_senior + marketing_junior)))
entropy_secretary <- -((secretary_senior/(secretary_senior + secretary_junior)) * log2(secretary_senior/(secretar
y_senior + secretary_junior)) + (secretary_junior/(secretary_senior + secretary_junior)) * log2(secretary_junior/
(secretary_senior + secretary_junior)))

entropy_department <- (sales_senior + sales_junior)/total_count * entropy_sales +
                      (systems_senior + systems_junior)/total_count * entropy_systems +
                      (marketing_senior + marketing_junior)/total_count * entropy_marketing +
                      (secretary_senior + secretary_junior)/total_count * entropy_secretary

information_gain_department <- entropy_D - entropy_department

# Age counts
age_21_25_junior <- 20
age_26_30_junior <- 46
age_31_35_junior <- 34
age_36_40_junior <- 0
age_41_45_junior <- 0
age_46_50_junior <- 0
age_21_25_senior <- 0
age_26_30_senior <- 0
age_31_35_senior <- 38
age_36_40_senior <- 10
age_41_45_senior <- 3
age_46_50_senior <- 4

# Entropy for Age
entropy_age <- 0
if (age_21_25_junior != 0 && age_21_25_senior != 0) {
  entropy_age <- entropy_age + (age_21_25_junior + age_21_25_senior)/total_count * -((age_21_25_senior/(age_21_25
_senior + age_21_25_junior)) * log2(age_21_25_senior/(age_21_25_senior + age_21_25_junior)) + (age_21_25_junior/
(age_21_25_senior + age_21_25_junior)) * log2(age_21_25_junior/(age_21_25_senior + age_21_25_junior)))
}
if (age_26_30_junior != 0 && age_26_30_senior != 0) {
  entropy_age <- entropy_age + (age_26_30_junior + age_26_30_senior)/total_count * -((age_26_30_senior/(age_26_30
_senior + age_26_30_junior)) * log2(age_26_30_senior/(age_26_30_senior + age_26_30_junior)) + (age_26_30_junior/
(age_26_30_senior + age_26_30_junior)) * log2(age_26_30_junior/(age_26_30_senior + age_26_30_junior)))
}
if (age_31_35_junior != 0 && age_31_35_senior != 0) {
  entropy_age <- entropy_age + (age_31_35_junior + age_31_35_senior)/total_count * -((age_31_35_senior/(age_31_35
_senior + age_31_35_junior)) * log2(age_31_35_senior/(age_31_35_senior + age_31_35_junior)) + (age_31_35_junior/
(age_31_35_senior + age_31_35_junior)) * log2(age_31_35_junior/(age_31_35_senior + age_31_35_junior)))
}
if (age_36_40_junior != 0 && age_36_40_senior != 0) {
  entropy_age <- entropy_age + (age_36_40_junior + age_36_40_senior)/total_count * -((age_36_40_senior/(age_36_40
_senior + age_36_40_junior)) * log2(age_36_40_senior/(age_36_40_senior + age_36_40_junior)) + (age_36_40_junior/
(age_36_40_senior + age_36_40_junior)) * log2(age_36_40_junior/(age_36_40_senior + age_36_40_junior)))
}
if (age_41_45_junior != 0 && age_41_45_senior != 0) {
  entropy_age <- entropy_age + (age_41_45_junior + age_41_45_senior)/total_count * -((age_41_45_senior/(age_41_45
_senior + age_41_45_junior)) * log2(age_41_45_senior/(age_41_45_senior + age_41_45_junior)) + (age_41_45_junior/
(age_41_45_senior + age_41_45_junior)) * log2(age_41_45_junior/(age_41_45_senior + age_41_45_junior)))
}
if (age_46_50_junior != 0 && age_46_50_senior != 0) {
  entropy_age <- entropy_age + (age_46_50_junior + age_46_50_senior)/total_count * -((age_46_50_senior/(age_46_50
_senior + age_46_50_junior)) * log2(age_46_50_senior/(age_46_50_senior + age_46_50_junior)) + (age_46_50_junior/
(age_46_50_senior + age_46_50_junior)) * log2(age_46_50_junior/(age_46_50_senior + age_46_50_junior)))
}

information_gain_age <- entropy_D - entropy_age

# Salary counts
salary_26K_30K_junior <- 46
salary_31K_35K_junior <- 40
salary_36K_40K_junior <- 0
salary_41K_45K_junior <- 0
salary_46K_50K_junior <- 23
salary_66K_70K_junior <- 3
salary_26K_30K_senior <- 0
salary_31K_35K_senior <- 0
salary_36K_40K_senior <- 4
salary_41K_45K_senior <- 0
salary_46K_50K_senior <- 47
salary_66K_70K_senior <- 14

# Entropy for Salary
entropy_salary <- 0
if (salary_26K_30K_junior != 0 && salary_26K_30K_senior != 0) {
  entropy_salary <- entropy_salary + (salary_26K_30K_junior + salary_26K_30K_senior)/total_count * -((salary_26K_
30K_senior/(salary_26K_30K_senior + salary_26K_30K_junior)) * log2(salary_26K_30K_senior/(salary_26K_30K_senior +
salary_26K_30K_junior)) + (salary_26K_30K_junior/(salary_26K_30K_senior + salary_26K_30K_junior)) * log2(salary_2
6K_30K_junior/(salary_26K_30K_senior + salary_26K_30K_junior)))
}
if (salary_31K_35K_junior != 0 && salary_31K_35K_senior != 0) {
  entropy_salary <- entropy_salary + (salary_31K_35K_junior + salary_31K_35K_senior)/total_count * -((salary_31K_
35K_senior/(salary_31K_35K_senior + salary_31K_35K_junior)) * log2(salary_31K_35K_senior/(salary_31K_35K_senior +
salary_31K_35K_junior)) + (salary_31K_35K_junior/(salary_31K_35K_senior + salary_31K_35K_junior)) * log2(salary_3
1K_35K_junior/(salary_31K_35K_senior + salary_31K_35K_junior)))
}
if (salary_36K_40K_junior != 0 && salary_36K_40K_senior != 0) {
  entropy_salary <- entropy_salary + (salary_36K_40K_junior + salary_36K_40K_senior)/total_count * -((salary_36K_
40K_senior/(salary_36K_40K_senior + salary_36K_40K_junior)) * log2(salary_36K_40K_senior/(salary_36K_40K_senior +
salary_36K_40K_junior)) + (salary_36K_40K_junior/(salary_36K_40K_senior + salary_36K_40K_junior)) * log2(salary_3
6K_40K_junior/(salary_36K_40K_senior + salary_36K_40K_junior)))
}
if (salary_41K_45K_junior != 0 && salary_41K_45K_senior != 0) {
  entropy_salary <- entropy_salary + (salary_41K_45K_junior + salary_41K_45K_senior)/total_count * -((salary_41K_
45K_senior/(salary_41K_45K_senior + salary_41K_45K_junior)) * log2(salary_41K_45K_senior/(salary_41K_45K_senior +
salary_41K_45K_junior)) + (salary_41K_45K_junior/(salary_41K_45K_senior + salary_41K_45K_junior)) * log2(salary_4
1K_45K_junior/(salary_41K_45K_senior + salary_41K_45K_junior)))
}
if (salary_46K_50K_junior != 0 && salary_46K_50K_senior != 0) {
  entropy_salary <- entropy_salary + (salary_46K_50K_junior + salary_46K_50K_senior)/total_count * -((salary_46K_
50K_senior/(salary_46K_50K_senior + salary_46K_50K_junior)) * log2(salary_46K_50K_senior/(salary_46K_50K_senior +
salary_46K_50K_junior)) + (salary_46K_50K_junior/(salary_46K_50K_senior + salary_46K_50K_junior)) * log2(salary_4
6K_50K_junior/(salary_46K_50K_senior + salary_46K_50K_junior)))
}
if (salary_66K_70K_junior != 0 && salary_66K_70K_senior != 0) {
  entropy_salary <- entropy_salary + (salary_66K_70K_junior + salary_66K_70K_senior)/total_count * -((salary_66K_
70K_senior/(salary_66K_70K_senior + salary_66K_70K_junior)) * log2(salary_66K_70K_senior/(salary_66K_70K_senior +
salary_66K_70K_junior)) + (salary_66K_70K_junior/(salary_66K_70K_senior + salary_66K_70K_junior)) * log2(salary_6
6K_70K_junior/(salary_66K_70K_senior + salary_66K_70K_junior)))
}
if (salary_26K_30K_junior == 0 || salary_26K_30K_senior == 0) {
  entropy_salary <- entropy_salary + 0
}
if (salary_31K_35K_junior == 0 || salary_31K_35K_senior == 0) {
  entropy_salary <- entropy_salary + 0
}
if (salary_36K_40K_junior == 0 || salary_36K_40K_senior == 0) {
  entropy_salary <- entropy_salary + 0
}
if (salary_41K_45K_junior == 0 || salary_41K_45K_senior == 0) {
  entropy_salary <- entropy_salary + 0
}
if (salary_46K_50K_junior == 0 || salary_46K_50K_senior == 0) {
  entropy_salary <- entropy_salary + 0
}
if (salary_66K_70K_junior == 0 || salary_66K_70K_senior == 0) {
  entropy_salary <- entropy_salary + 0
}

information_gain_salary <- entropy_D - entropy_salary

information_gain <- data.frame(Attribute = c("Department", "Age", "Salary"),
                               Information_Gain = c(information_gain_department, information_gain_age, informatio
n_gain_salary))
information_gain
```

Attribute	Information_Gain
<chr>	<dbl>
Department	0.04860679
Age	0.46363915
Salary	0.44223549

3 rows

it defines a function null min_max_normalization to normalize that data to a standard range. Input Parameters: The purpose of the function will be to take 3 parameters: the input data vector a, and the two values for the new minimum output value and the maximum output range respectively, min_new and max_new. Calculating Min and Max: Within this function, the values of min() and max() functions are calculated by specifying a and will be used to find the minimum and maximum values of the input data vector a. The values are vectorized as min_old and max_old respectively.

Normalization Formula: Pseudo-code of min-max normalization comes first with each data vector element a. For each element x in a, the formula resembles (x - min_old) * (max_new - min_new) / (max_old - min_old) + min_new will be applied to scale x as the source range will. This function is responsible for scaling the data input vector into a new vector full of values scaled by the input data vector scale values. As illustrated, the function is demonstrated using the age_data vector as an input, with a desired output range between 0 (min) and 1 (max). This is the end result of this process and the age column shows the values corresponding to min-max normalized age values.

Question 4

```
## Decision Tree Rules

# Decision Tree Rules
rules <- c("If Department is Sales:",
           "- If Age is 26-30, then the person is Junior.",
           "- If Age is 31-35 and Salary is 31K-35K, then the person is Junior.",
           "- If none of the above conditions are met, then the person is Senior.",
           "",
           "If Department is Systems:",
           "- If Age is 21-25 and Salary is 46K-50K, then the person is Junior.",
           "- If Age is 31-35 and Salary is 66K-70K, then the person is Senior.",
           "- If none of the above conditions are met, then the person is Junior.",
           "",
           "If Department is Marketing:",
           "- If Age is 36-40 and Salary is 46K-50K, then the person is Senior.",
           "- If Age is 31-35 and Salary is 41K-45K, then the person is Junior.",
           "- If none of the above conditions are met, then the person is Junior.",
           "",
           "If Department is Secretary:",
           "- If Age is 46-50 and Salary is 36K-40K, then the person is Senior.",
           "- If Age is 26-30 and Salary is 26K-30K, then the person is Junior.",
           "- If none of the above conditions are met, then the person is Senior."
          )

# Printing Decision Tree Rules
cat(rules, sep = "\n")
```

```
## If Department is Sales:
## - If Age is 26-30, then the person is Junior.
## - If Age is 31-35 and Salary is 31K-35K, then the person is Junior.
## - If none of the above conditions are met, then the person is Senior.
##
## If Department is Systems:
## - If Age is 21-25 and Salary is 46K-50K, then the person is Junior.
## - If Age is 31-35 and Salary is 66K-70K, then the person is Senior.
## - If none of the above conditions are met, then the person is Junior.
##
## If Department is Marketing:
## - If Age is 36-40 and Salary is 46K-50K, then the person is Senior.
## - If Age is 31-35 and Salary is 41K-45K, then the person is Junior.
## - If none of the above conditions are met, then the person is Junior.
##
## If Department is Secretary:
## - If Age is 46-50 and Salary is 36K-40K, then the person is Senior.
## - If Age is 26-30 and Salary is 26K-30K, then the person is Junior.
## - If none of the above conditions are met, then the person is Senior.
```

It offers string of qualifications for juxtaposition of the persons: junior or senior on bases of their attributes (department, age, and salary). The hierarchical positioning can be observed in arrangements where each part is designated the necessary preconditions to categorize people. For every province, necessary requirements based on age and salary are given, the employee is classified into only one of these conditions or else will be classified by default. The ruling that these rules are layed down in a vector for convenient understanding and printing is made by the code. Every rule is constructed in order to display the “if-then” environment which determines whether an individual falls into category junior or senior in given a department.