

Problem Set #01 Solutions Statistics 506

Problem Set #01

Problem 1 Solutions - Wine Data

a.

```
wine <- read.csv("data/wine/wine.data", header = FALSE)
names(wine) <- c("class",
                "alcohol",
                "malic_acid",
                "ash",
                "alcalanity",
                "magnesium",
                "phenols",
                "flavanioids",
                "nonflavanooids",
                "proanthocyanins",
                "color_intensity",
                "hue",
                "od280",
                "proline")
```

b.

“wines.name” reports:

```
class 1 59 class 2 71 class 3 48
```

```
table(wine$class)
```

```
1 2 3
59 71 48
```

c.

```
wine$class[which(wine$alcohol == max(wine$alcohol))]
```

```
[1] 1
```

```
wine$class[which(wine$alcohol == min(wine$alcohol))]
```

```
[1] 2
```

```
table(wine$magnesium > 114)
```

```
FALSE TRUE
152    26
```

```
table(wine$magnesium > 114, wine$class)
```

```
      1 2 3
FALSE 44 65 43
TRUE  15 6 5
```

d.

```
means <- rbind(colMeans(wine[, -1]),
               colMeans(wine[wine$class == 1, -1]),
               colMeans(wine[wine$class == 2, -1]),
               colMeans(wine[wine$class == 3, -1]))
means <- as.data.frame(means)
rownames(means) <- c("Overall", "Class 1", "Class 2", "Class 3")
round(means, 2)
```

	alcohol	malic_acid	ash	alcalanity	magnesium	phenols	flavanioids
Overall	13.00	2.34	2.37	19.49	99.74	2.30	2.03
Class 1	13.74	2.01	2.46	17.04	106.34	2.84	2.98
Class 2	12.28	1.93	2.24	20.24	94.55	2.26	2.08
Class 3	13.15	3.33	2.44	21.42	99.31	1.68	0.78

	nonflavanooids	proanthocyanins	color_intensity	hue	od280	proline
Overall	0.36		1.59	5.06	0.96	2.61
Class 1	0.29		1.90	5.53	1.06	3.16
Class 2	0.36		1.63	3.09	1.06	2.79
Class 3	0.45		1.15	7.40	0.68	1.68

e.

```
t.test(wine$ash[wine$class == 1], wine$ash[wine$class == 2])
```

Welch Two Sample t-test

```
data: wine$ash[wine$class == 1] and wine$ash[wine$class == 2]
t = 4.4184, df = 125.59, p-value = 2.124e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.116383 0.305226
sample estimates:
mean of x mean of y
 2.455593  2.244789
```

```
t.test(wine$ash[wine$class == 1], wine$ash[wine$class == 3])
```

Welch Two Sample t-test

```
data: wine$ash[wine$class == 1] and wine$ash[wine$class == 3]
t = 0.46489, df = 105, p-value = 0.643
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-0.06043717  0.09745695
sample estimates:
mean of x mean of y
 2.455593  2.437083
```

```
t.test(wine$ash[wine$class == 2], wine$ash[wine$class == 3])
```

Welch Two Sample t-test

```
data: wine$ash[wine$class == 2] and wine$ash[wine$class == 3]
t = -4.184, df = 114.96, p-value = 5.627e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.2833328 -0.1012564
sample estimates:
mean of x mean of y
 2.244789  2.437083
```

From these results, we see that Class 2's ash is statistically significantly lower, but Class 1 and 3's ash is statistically indistinguishable.

Problem 2 Solutions - Perfect Powers

a.

```
isPerfectPower <- function(n, power = 2) {
  if (power < 2) {
    stop("`power` must be at least 2")
  }
  if (!(all.equal(n, round(n)) == TRUE) |
      !(all.equal(power, round(power)) == TRUE)) {
    stop("`n` and `power` must be integers")
  }

  if (n < 0) {
    if (power %% 2 == 0) {
      # For negative n's, only odd powers can be perfect
      return(list(isPerfectPower = FALSE,
                  root = NA))
    }
  }
  res <- isPerfectPower(abs(n), power = power)
  if (res$isPerfectPower) {
    res$root <- -res$root
    return(res)
  }
}
```

```

    } else {
      return(list(isPerfectPower = FALSE,
                 root = NA))
    }
  }

  root <- n^(1/power)
  rounded <- round(root)
  # If the `root` rounds to the same value, it means we have a perfect power.

  # Using `all.equal` to handle floating point rounding issues
  if (all.equal(rounded, root) == TRUE) {
    return(list(isPerfectPower = TRUE,
               root = root))
  } else {
    return(list(isPerfectPower = FALSE,
               root = NA))
  }
}

isPerfectPower(4, 2)

```

```

$isPerfectPower
[1] TRUE

```

```

$root
[1] 2

```

```

isPerfectPower(5, 2)

```

```

$isPerfectPower
[1] FALSE

```

```

$root
[1] NA

```

```

isPerfectPower(-8, 3)

```

```

$isPerfectPower

```

```
[1] TRUE
```

```
$root
```

```
[1] -2
```

```
isPerfectPower(1, 10)
```

```
$isPerfectPower
```

```
[1] TRUE
```

```
$root
```

```
[1] 1
```

b.

```
findRootPower <- function(n) {  
  # Stopping at 50; even 2^50 is larger than any inputs  
  for (i in 2:50) {  
    res <- isPerfectPower(n, i)  
    if (res$isPerfectPower) {  
      # Return early the first time we've found a solutions  
      return(paste0(n, " = ", res$root, "^", i))  
    }  
  }  
  # Only hits here if it doesn't return by finding something  
  print(paste(n, "is not a perfect power"))  
}
```

```
findRootPower(27)
```

```
[1] "27 = 3^3"
```

```
findRootPower(13060694016)
```

```
[1] "13060694016 = 6^13"
```

```
findRootPower(7776)
```

```
[1] "7776 = 6^5"
```

```
findRootPower(170859375)
```

```
[1] "170859375 = 15^7"
```

```
findRootPower(58247422)
```

```
[1] "58247422 is not a perfect power"
```

```
findRootPower(94143178827)
```

```
[1] "94143178827 = 3^23"
```

Problem 3 Solutions - ChatGPT

Solutions to this problem will obviously vary greatly depending on the response from ChatGPT. Here is one example.

a.

Here's what ChatGPT responded with:

Sure, I can help you with that. Here are the two R functions you requested:

```
# Function to determine the name of a poker hand
get_poker_hand_name <- function(suits, ranks) {
  hand <- data.frame(suits, ranks)

  if (length(unique(hand$suits)) == 1) {
    if (all(hand$ranks %in% c("10", "J", "Q", "K", "A"))) {
      return("Royal Flush")
    }
    if (all(diff(as.numeric(factor(hand$ranks, levels = c("A", "K", "Q", "J", "10", "9", "8")))) == 1)) {
      return("Straight Flush")
    }
    return("Flush")
  }

  if (all(diff(as.numeric(factor(hand$ranks, levels = c("A", "K", "Q", "J", "10", "9", "8")))) == 1)) {
    return("Straight")
  }
}
```

```

rank_counts <- table(hand$ranks)
if (any(rank_counts == 4)) {
  return("Four of a Kind")
}
if (any(rank_counts == 3) && any(rank_counts == 2)) {
  return("Full House")
}
if (any(rank_counts == 3)) {
  return("Three of a Kind")
}
if (sum(rank_counts == 2) == 2) {
  return("Two Pair")
}
if (any(rank_counts == 2)) {
  return("One Pair")
}

return("High Card")
}

# Function to simulate a round of poker
simulate_poker_round <- function(num_players) {
  suits <- c("Hearts", "Diamonds", "Clubs", "Spades")
  ranks <- c("2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A")

  deck <- expand.grid(suits = suits, ranks = ranks)
  shuffled_deck <- deck[sample(nrow(deck)), ]

  hands <- matrix(nrow = num_players, ncol = 5)
  for (i in 1:num_players) {
    hands[i,] <- shuffled_deck[(i - 1) * 5 + 1:i * 5, ]
  }

  hand_names <- character(num_players)
  for (i in 1:num_players) {
    hand_names[i] <- get_poker_hand_name(hands[i, "suits"], hands[i, "ranks"])
  }

  result <- data.frame(Player = 1:num_players, Hand = hand_names)
  return(result)
}

```


You can use the `simulate_poker_round` function to simulate a round of poker with a specified number of players, and it will return a data frame showing the hands and their corresponding names.

b.

It took quite a bit to get this working. I'm put it in a [seperate document](#) to make this file a bit easier to read.

c.

Commented code below, based on the fixed version:

```
# This first function takes in a vector of suits and ranks for a given hand, and
# returns the name of the hand.
get_poker_hand_name <- function(suits, ranks) {
  # Create a 5x2 data.frame containing the hand
  hand <- data.frame(suits, ranks)

  # If all the suits are the same, we have some kind of flush
  if (length(unique(hand$suits)) == 1) {
    # Checking if all card ranks are above 10
    if (all(hand$ranks %in% c("10", "J", "Q", "K", "A"))) {
      return("Royal Flush")
    }
    # This does a lot:
    # 1. Convert ranks from strings to factor
    # 2. Convert then to numeric, so we have numeric values 1 through 13
    # 3. Look at the differences between ranks
    # 4. Check if those differences are all 1, implying a straight
    if (all(diff(as.numeric(factor(hand$ranks,
                                   levels = c("A", "K", "Q", "J", "10",
                                               "9", "8", "7", "6", "5",
                                               "4", "3", "2")))) == 1))) {
      return("Straight Flush")
    }
    # Otherwise, normal flush
    return("Flush")
  }

  # Using the same logic above to identify straights - but we know that it's no
  # longer a flush since we'd have `return`ed above if it was a flush of any
  # kind.
  if (all(diff(as.numeric(factor(hand$ranks,
```

```

                                levels = c("A", "K", "Q", "J", "10", "9",
                                              "8", "7", "6", "5", "4", "3",
                                              "2")) == 1))) {
  return("Straight")
}

rank_counts <- table(hand$ranks)
# If we have four of the same rank...
if (any(rank_counts == 4)) {
  return("Four of a Kind")
}
# If we have 3 & 2 of the same rank...
if (any(rank_counts == 3) && any(rank_counts == 2)) {
  return("Full House")
}
# If we had 3 & 2, it would have returned already, so we must have 3 & 1 & 1
if (any(rank_counts == 3)) {
  return("Three of a Kind")
}
# Again, we know we have no ranks appearing more than twice, check if we have
# two ranks appearing twice.
if (sum(rank_counts == 2) == 2) {
  return("Two Pair")
}
# If we have any pairs left, it can only be a single pair.
if (any(rank_counts == 2)) {
  return("One Pair")
}

return("High Card")
}

# This function takes in the number of players, draws hands, uses
# `get_poker_hand_name` to identify the hand, then returns the results.
simulate_poker_round <- function(num_players) {
  suits <- c("Hearts", "Diamonds", "Clubs", "Spades")
  ranks <- c("2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A")

  # After defining the unique suits and ranks, `expand.grid` creates all unique
  # combinations of them.
  deck <- expand.grid(suits = suits, ranks = ranks)

```

```

# Shuffle the deck!
shuffled_deck <- deck[sample(nrow(deck)), ]

hands <- list()
for (i in 1:num_players) {
  # Save the top five cards into the hand
  hands[[i]] <- shuffled_deck[1:5, ]
  # Remove the cards from the deck
  shuffled_deck <- shuffled_deck[-(1:5), ]
}

hand_names <- character(num_players)
for (i in 1:num_players) {
  # Iterate over each hand, obtaining it's name
  hand_names[i] <- get_poker_hand_name(hands[[i]]["suits"],
                                       hands[[i]]["ranks"])
}

# Store the results in a nicer format
result <- data.frame(Player = 1:num_players, Hand = hand_names)
return(result)
}

```

d.

First, let's just throw some input at the function to see what happens.

```

set.seed(1) # Setting a seed for reproducible results
simulate_poker_round(4)

```

Player	Hand
1	1 One Pair
2	2 High Card
3	3 One Pair
4	4 High Card

We requested to see the hand, which we don't see. Let's fix this. (I've used [code-folding](#) to make this document more readable.

```

simulate_poker_round <- function(num_players) {
  suits <- c("Hearts", "Diamonds", "Clubs", "Spades")

```

```

ranks <- c("2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A")

deck <- expand.grid(suits = suits, ranks = ranks)

shuffled_deck <- deck[sample(nrow(deck)), ]

hands <- list()
for (i in 1:num_players) {
  hands[[i]] <- shuffled_deck[1:5, ]
  shuffled_deck <- shuffled_deck[-(1:5), ]
}

hand_names <- character(num_players)
for (i in 1:num_players) {
  hand_names[i] <- get_poker_hand_name(hands[[i]]["suits"],
                                       hands[[i]]["ranks"])
}

for (i in 1:num_players) {
  cat("Player", i, ":\n",
      paste(hands[[i]]$ranks, hands[[i]]$suits, sep = " of ", collapse = "\n "),
      "\n Hand:", hand_names[i], "\n")
}
}

```

① Add in hand printing

```

set.seed(2)
simulate_poker_round(4)

```

```

Player 1 :
 7 of Hearts
 5 of Clubs
 3 of Diamonds
 A of Diamonds
 9 of Spades
Hand: High Card
Player 2 :
 3 of Spades
 6 of Hearts
 9 of Hearts

```

K of Diamonds
4 of Spades
Hand: High Card

Player 3 :

Q of Hearts
4 of Clubs
2 of Hearts
2 of Clubs
5 of Spades

Hand: One Pair

Player 4 :

K of Spades
K of Clubs
10 of Hearts
J of Diamonds
10 of Diamonds

Hand: Two Pair

```
simulate_poker_round(11)
```

Player 1 :

8 of Spades
2 of Spades
4 of Hearts
Q of Diamonds
K of Clubs

Hand: High Card

Player 2 :

3 of Diamonds
9 of Spades
5 of Clubs
5 of Hearts
10 of Clubs

Hand: One Pair

Player 3 :

3 of Spades
K of Diamonds
Q of Spades
6 of Hearts
3 of Hearts

Hand: One Pair

Player 4 :
A of Diamonds
K of Hearts
J of Clubs
5 of Spades
7 of Hearts
Hand: High Card

Player 5 :
4 of Clubs
J of Hearts
10 of Spades
4 of Diamonds
5 of Diamonds
Hand: One Pair

Player 6 :
J of Diamonds
Q of Hearts
Q of Clubs
4 of Spades
6 of Diamonds
Hand: One Pair

Player 7 :
6 of Spades
2 of Hearts
A of Clubs
K of Spades
A of Spades
Hand: One Pair

Player 8 :
3 of Clubs
10 of Diamonds
2 of Clubs
8 of Clubs
7 of Diamonds
Hand: High Card

Player 9 :
7 of Clubs
9 of Clubs
7 of Spades
9 of Diamonds
8 of Diamonds
Hand: Two Pair

Player 10 :

```

10 of Hearts
6 of Clubs
8 of Hearts
A of Hearts
9 of Hearts
Hand: High Card
Player 11 :
J of Spades
2 of Diamonds
NA of NA
NA of NA
NA of NA
Hand: High Card

```

Those NA's are undesirable. Let's add some input sanitization to make sure `num_players` is reasonable.

```

simulate_poker_round <- function(num_players) {
  if (num_players < 1) {                                ①
    stop("Too few players!")
  }
  if (num_players > 10) {                               ②
    stop("Too many players!")
  }
  if (!all.equal(round(num_players), num_players)) {    ③
    # If `num_players` rounds to something besides itself, it must not be an
    # integer
    stop("`num_players` must be whole number")
  }
  suits <- c("Hearts", "Diamonds", "Clubs", "Spades")
  ranks <- c("2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A")

  deck <- expand.grid(suits = suits, ranks = ranks)

  shuffled_deck <- deck[sample(nrow(deck)), ]

  hands <- list()
  for (i in 1:num_players) {
    hands[[i]] <- shuffled_deck[1:5, ]
    shuffled_deck <- shuffled_deck[-(1:5), ]
  }
}

```

```

hand_names <- character(num_players)
for (i in 1:num_players) {
  hand_names[i] <- get_poker_hand_name(hands[[i]]["suits"],
                                      hands[[i]]["ranks"])
}

for (i in 1:num_players) {
  cat("Player", i, ":\n",
      paste(hands[[i]]$ranks, hands[[i]]$suits, sep = " of ", collapse = "\n "),
      "\n Hand:", hand_names[i], "\n")
}
}

```

- ① Ensure we have a positive number of players
- ② Ensure we have enough cards for everyone
- ③ Ensure we have an integer input

```

set.seed(2)
simulate_poker_round(11)

```

Error in simulate_poker_round(11): Too many players!

```

simulate_poker_round(-2)

```

Error in simulate_poker_round(-2): Too few players!

```

simulate_poker_round(2.6)

```

Error in !all.equal(round(num_players), num_players): invalid argument type

```

simulate_poker_round(sqrt(2)^2) # Does it work with near integers?

```

```

Player 1 :
 7 of Hearts
 5 of Clubs
 3 of Diamonds
 A of Diamonds

```



```
9 of Spades
Hand: High Card
Player 2 :
3 of Spades
6 of Hearts
9 of Hearts
K of Diamonds
4 of Spades
Hand: High Card
```

Finally, let's make sure it can correctly identify all the hands.

```
# Royal Flush
get_poker_hand_name(rep("Hearts", 5), c("10", "J", "Q", "K", "A"))

[1] "Royal Flush"

# Straight Flush
get_poker_hand_name(rep("Hearts", 5), c("5", "6", "7", "8", "9"))

[1] "Flush"

get_poker_hand_name(rep("Hearts", 5), c("5", "6", "7", "9", "8"))

[1] "Flush"

# Flush
get_poker_hand_name(rep("Hearts", 5), c("5", "2", "7", "9", "8"))

[1] "Flush"

# Straight
get_poker_hand_name(c(rep("Hearts", 4), "Diamonds"),
                    c("5", "6", "7", "8", "9"))

[1] "High Card"
```

```
get_poker_hand_name(c(rep("Hearts", 4), "Diamonds"),
                    c("9", "8", "7", "6", "5"))
```

```
[1] "High Card"
```

```
get_poker_hand_name(c(rep("Hearts", 4), "Diamonds"),
                    c("5", "6", "7", "9", "8"))
```

```
[1] "High Card"
```

```
# Four of a kind
get_poker_hand_name(c("Hearts", "Diamonds", "Spades", "Spades", "Diamonds"),
                    c("5", "5", "5", "A", "5"))
```

```
[1] "Four of a Kind"
```

```
# Full House
get_poker_hand_name(c("Hearts", "Diamonds", "Spades", "Spades", "Diamonds"),
                    c("5", "5", "5", "A", "A"))
```

```
[1] "Full House"
```

```
# Three of a Kind
get_poker_hand_name(c("Hearts", "Diamonds", "Spades", "Spades", "Diamonds"),
                    c("5", "5", "5", "A", "10"))
```

```
[1] "Three of a Kind"
```

```
# Two Pair
get_poker_hand_name(c("Hearts", "Diamonds", "Spades", "Spades", "Diamonds"),
                    c("5", "5", "2", "10", "10"))
```

```
[1] "Two Pair"
```

```
# One Pair
get_poker_hand_name(c("Hearts", "Diamonds", "Spades", "Spades", "Diamonds"),
                    c("7", "5", "2", "10", "10"))
```

```
[1] "One Pair"
```

```
# High Card
get_poker_hand_name(c("Hearts", "Diamonds", "Spades", "Spades", "Diamonds"),
                    c("7", "5", "2", "8", "10"))
```

```
[1] "High Card"
```

All the straights are failing. There's no sorting going on - let's add that.

```
get_poker_hand_name <- function(suits, ranks) {
  hand <- data.frame(suits, ranks)

  if (length(unique(hand$suits)) == 1) {
    if (all(hand$ranks %in% c("10", "J", "Q", "K", "A"))) {
      return("Royal Flush")
    }
    if (all(diff(sort(as.numeric(factor(hand$ranks,
                                       levels = c("A", "K", "Q", "J", "10",
                                                   "9", "8", "7", "6", "5",
                                                   "4", "3", "2"))))) == 1)) {
      return("Straight Flush")
    }
    return("Flush")
  }

  if (all(diff(sort(as.numeric(factor(hand$ranks,
                                       levels = c("A", "K", "Q", "J", "10", "9",
                                                   "8", "7", "6", "5", "4", "3",
                                                   "2"))))) == 1)) {
    return("Straight")
  }

  rank_counts <- table(hand$ranks)
  if (any(rank_counts == 4)) {
    return("Four of a Kind")
  }
}
```

```

}
if (any(rank_counts == 3) && any(rank_counts == 2)) {
  return("Full House")
}
if (any(rank_counts == 3)) {
  return("Three of a Kind")
}
if (sum(rank_counts == 2) == 2) {
  return("Two Pair")
}
if (any(rank_counts == 2)) {
  return("One Pair")
}

return("High Card")
}

```

- ① Add `sort()` to the call stack.
- ② Add `sort()` to the call stack.

Be careful to place the closing `}` in the proper position.

```

# Straight Flush
get_poker_hand_name(rep("Hearts", 5), c("5", "6", "7", "8", "9"))

```

```
[1] "Straight Flush"
```

```

get_poker_hand_name(rep("Hearts", 5), c("5", "6", "7", "9", "8"))

```

```
[1] "Straight Flush"
```

```

# Straight
get_poker_hand_name(c(rep("Hearts", 4), "Diamonds"),
  c("5", "6", "7", "8", "9"))

```

```
[1] "Straight"
```

```
get_poker_hand_name(c(rep("Hearts", 4), "Diamonds"),  
                    c("9", "8", "7", "6", "5"))
```

```
[1] "Straight"
```

```
get_poker_hand_name(c(rep("Hearts", 4), "Diamonds"),  
                    c("5", "6", "7", "9", "8"))
```

```
[1] "Straight"
```