Problem Set 1

Problem 1 - Wine data

a.Import the data into a `data.frame` in R. Use the information in the "wine.names" file to give appropriate column names. (Note: Downloading and unzipping the file can take place outside of your submitted document, but importing the file should be in the submission.)

```
wine.data <- read.csv('wine.data', header = FALSE)
```

b. The data contains information on three different classes of wine. Ensure that the number of wines within each class is correct as reported in "wine.names".

```
colnames(wine.data) <- c("Class","Alcohol","Malic acid","Ash","Alcalinity of ash","Magnesi
```

c. Use the data to answer the following questions:

1. The wine with the highest alcohol content belongs to which class?

```
wine.data[which(wine.data["Alcohol"] == max(wine.data["Alcohol"])),]["Class"]
```

```
  Class
9     1
```

2. The wine with the lowest alcohol content belongs to which class?

```
wine.data[which(wine.data["Alcohol"] == min(wine.data["Alcohol"])),]["Class"]
```

```
    Class
116     2
```

3. German beers have, on average, 114 mg/l of magnesium. How many of these wines have higher levels of magnesium than that? (You may assume that the magnesium reported in the data is in the same units.)

```r
length(which(wine.data["Magnesium"] > 114))
```

```
[1] 26
```

4. Within each class, how many wines have higher levels of magnesium than average German beer?

```r
# Class 1
length(which(wine.data['Class'] == 1 & wine.data["Magnesium"] > 114))
```

```
[1] 15
```

```r
# Class 2
length(which(wine.data['Class'] == 2 & wine.data["Magnesium"] > 114))
```

```
[1] 6
```

```r
# Class 3
length(which(wine.data['Class'] == 3 & wine.data["Magnesium"] > 114))
```

```
[1] 5
```

d. Create a table identifying the average value of each variable, providing one row for the overall average, and one row per class with class averages. (This table does not need to be "fancy" but should clearly identify what each value represents.)

```r
overall_avg <- sapply(wine.data[,-1], mean)
overall_avg <- data.frame(Class = "Overall",t(overall_avg))

Class1 <- sapply(wine.data[wine.data[1] == 1, -1], mean)
Class1 <- data.frame(Class = 1 ,t(Class1))

Class2 <- sapply(wine.data[wine.data[1] == 2, -1], mean)
Class2 <- data.frame(Class = 2 ,t(Class2))

Class3 <- sapply(wine.data[wine.data[1] == 3, -1], mean)
Class3 <- data.frame(Class = 3 ,t(Class3))
```

```r
final_table <- rbind(overall_avg, Class1, Class2, Class3)
```

e. Carry out a series of t-tests to examine whether the level of Ash differs across the three classes. Present the R output and interpret the results. (You may use an existing R function to carry out the t-test, or for **minor extra credit**, manually write your own calculation of the t-test p-values.)

```r
class1 <- wine.data$Ash[wine.data$Class == 1]
class2 <- wine.data$Ash[wine.data$Class == 2]
class3 <- wine.data$Ash[wine.data$Class == 3]
tt_1_2 <- t.test(class1, class2)
tt_1_3 <- t.test(class1, class3)
tt_2_3 <- t.test(class2, class3)

print("T-test between Class1 and Class2")
```

```
[1] "T-test between Class1 and Class2"
```

```r
print(tt_1_2)
```

```

	Welch Two Sample t-test

data:  class1 and class2
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 
```

Interpretation: t = 4.4184, the mean of class1 and class2 are different. P-value = 2.124e-05 < 0.05. This indicates that you can reject the null hypothesis and the data show the strong evidence that the level of ash in Class1 and Class2 are significantly different.

```r
print("T-test between Class1 and Class3")
```

```
[1] "T-test between Class1 and Class3"
```

```r
print(tt_1_3)
```

```

    Welch Two Sample t-test

data:  class1 and class3
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 
```

Interpretation: $t = 0.46489$ and P-value $= 0.643 > 0.05$, the mean of class1 and class3 may not be significantly different from each other. This indicates that you can not reject the null hypothesis and the data show the difference in Ash levels between the two classes is likely not meaningful.

```r
print("T-test between Class2 and Class3")
```

```
[1] "T-test between Class2 and Class3"
```

```r
print(tt_2_3)
```

```

    Welch Two Sample t-test

data:  class2 and class3
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 
```

Interpretation: t = -4.4184, the mean of class2 and class3 are different. P-value = 5.627e-05 < 0.05. This indicates that you can reject the null hypothesis and the data show the strong evidence that the level of ash in Class2 and Class3 are significantly different.

Problem 2 - Perfect Powers

a. Write a function "`isPerfectPower`" to identify whether a given integer is a perfect power for a given power. Do not use any existing functions to check this; do so with arithmetic.

- Input: Two integers - the number to check, and specified power.
- Output: A `list` of length 2; the first entry should be a logical, the second should be the root of the input or a reasonable value if the input is not a perfect power.

```
isPerfectPower<- function(n, power){
  root_apr <- n^(1/power)
  integer <- floor(root_apr)
  isPerfect <- (integer^power == n)
  result <- list("isPerfect" = isPerfect, "root" = integer)
  return(result)
}
```

```
isPerfectPower(27, power = 3)
```

```
$isPerfect
[1] TRUE

$root
[1] 3
```

b. Demonstrate your function works. Do so by writing another function "`findRootPower`" which calls your first function, using a loop to identify both the root and power. Your function should identify the *lowest* power for which the input is perfect.

- Input: An integer.
- Output: The root and the power as a valid string equation with an appropriate message if the input is not a perfect power.

```
findRootPower <- function(n){
  for(i in 2:floor(log2(n))){
    result <- isPerfectPower(n, i)
    if(result$isPerfect){
      return(paste(n,"=",n^(1/i),"^",i))
    }
  }
  return(paste("The number", n, "is not a perfect power."))
}
```

```
print(findRootPower(27))
```

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

```
print(findRootPower(13060694016))
```

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

```
print(findRootPower(7776))
```

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

```
print(findRootPower(170859375))
```

```
[1] "The number 170859375 is not a perfect power."
```

```
print(findRootPower(58247422))
```

```
[1] "The number 58247422 is not a perfect power."
```

```
print(findRootPower(94143178827))
```

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

Problem 3 - ChatGPT

a. Put the following prompt into ChatGPT (or your favorite other LLM) and copy its output (both the code it produces as well as any text it generates) into your submission. (If a non-ChatGPT LLM produces nonsense results, try ChatGPT. If that is still producing nonsense, let me know. Blockquotes might be useful for displaying this inside your Quarto document.)

Produce two R functions: The first should take in a vector of suits and a vector of ranks representing a 5-card hand, and return the name of the hand (in 5 card stud poker). The second should simulate dealing a round of cards in a game of poker (5 card stud). The user should be able to specify the number of players. Show the user all the hands, and use the first function to display the name of each of the hands.

To produce two R functions for this, let's consider the rules of the game for 5-card stud poker. Here are the hand ranks, from highest to lowest:

1. Royal Flush
2. Straight Flush
3. Four of a Kind
4. Full House
5. Flush
6. Straight
7. Three of a Kind
8. Two Pair
9. One Pair 10.High Card

First Function: Identify Poker Hand The first function takes a vector of suits and a vector of ranks, representing a 5-card hand, and returns the name of the hand. This function, identify_hand, will be used to assess each player's hand later on.

```
identify_hand <- function(suits, ranks) {
  ranks <- as.integer(ranks)

  unique_suits <- unique(suits) # remove duplicate elements/rows removed
  unique_ranks <- unique(ranks)

  is_flush <- length(unique_suits) == 1 # check for flush
  is_straight <- length(unique_ranks) == 5 && max(ranks) - min(ranks) == 4 # check for str
  rank_count <- table(ranks) # a table that counts the occurrences of each rank in the han

  if (is_flush && is_straight && max(ranks) == 14) {
    return("Royal Flush")
```

```r
  }

  if (is_flush && is_straight) {
    return("Straight Flush")
  }

  if (max(rank_count) == 4) {
    return("Four of a Kind")
  }

  if (length(unique(rank_count)) == 2 && max(rank_count) == 3) {
    return("Full House")
  }

  if (is_flush) {
    return("Flush")
  }

  if (is_straight) {
    return("Straight")
  }

  if (max(rank_count) == 3) {
    return("Three of a Kind")
  }

  if (length(unique(rank_count)) == 2 && max(rank_count) == 2) {
    return("Two Pair")
  }

  if (max(rank_count) == 2) {
    return("One Pair")
  }

  return("High Card")
}

# Example usage
suits <- c('S', 'S', 'S', 'S', 'S')
ranks <- c('10', '11', '12', '13', '14')
identify_hand(suits, ranks)  # Should return "Royal Flush"
```

```
[1] "Royal Flush"
```

Second Function: Simulate 5-Card Stud The second function simulates dealing a round of 5-card stud poker for a specified number of players. This function will use the identify_hand function to evaluate the hands.

```
deal_poker_hand <- function(num_players) {
  all_suits <- rep(c('S', 'H', 'D', 'C'), each = 13)
  all_ranks <- rep(2:14, times = 4)

  deck <- data.frame(Suit = all_suits, Rank = all_ranks)

  shuffled_indices <- sample.int(nrow(deck))
  # shuffled by randomly sampling the indices of the rows of the deck
  shuffled_deck <- deck[shuffled_indices, ]

  for (i in 1:num_players) {
    hand_index_start <- 5 * (i - 1) + 1
    hand_index_end <- 5 * i
    # calculate the start and end indices of their 5-card hand in the shuffled deck.
    hand <- shuffled_deck[hand_index_start:hand_index_end, ]
    # extracts the 5 cards for the player's hand from the shuffled deck.
    hand_name <- identify_hand(hand$Suit, hand$Rank)
    # identify the name/type of the poker hand that the player has.
    cat("Player", i, ":", hand_name, "\n")
    cat("Suits: ", hand$Suit, "\n")
    cat("Ranks: ", hand$Rank, "\n\n")
  }
}
```

Example usage for both functions

```
# For identify_hand function
suits <- c('S', 'S', 'S', 'S', 'S')
ranks <- c('10', '11', '12', '13', '14')
identify_hand(suits, ranks)  # Should return "Royal Flush"
```

```
[1] "Royal Flush"
```

```r
# For deal_poker_hand function
deal_poker_hand(3)  # Simulates a 3-player game
```

```
Player 1 : Full House
Suits:  H C S S C
Ranks:  13 5 13 6 13

Player 2 : High Card
Suits:  D H D D C
Ranks:  14 8 9 11 2

Player 3 : Two Pair
Suits:  D H C S S
Ranks:  5 14 7 3 14
```

b.See if the code runs without modification in R. If so, demonstrate it with a range of inputs. If not, fix it and explain what you fixed. (If you identify that the code is *massively* broken, try re-generating the response.) At this point we only care if the code runs without errors for a variety on inputs; we'll make sure it's correct below.

```r
# Example 1: Royal Flush
suits <- c('S', 'S', 'S', 'S', 'S')
ranks <- c('10', '11', '12', '13', '14')
cat("Example 1: ", identify_hand(suits, ranks), "\n")  # Should return "Royal Flush"
```

```
Example 1:  Royal Flush
```

```r
# Example 2: Straight Flush
suits <- c('S', 'S', 'S', 'S', 'S')
ranks <- c('2', '3', '4', '5', '6')
cat("Example 2: ", identify_hand(suits, ranks), "\n")  # Should return "Straight Flush"
```

```
Example 2:  Straight Flush
```

```r
# Example 3: Four of a Kind
suits <- c('S', 'D', 'H', 'C', 'S')
ranks <- c('10', '10', '10', '10', '2')
```

```r
cat("Example 3: ", identify_hand(suits, ranks), "\n")  # Should return "Four of a Kind"
```

```
Example 3:  Four of a Kind
```

```r
# Example 4: Full House
suits <- c('S', 'H', 'D', 'C', 'S')
ranks <- c('10', '10', '2', '2', '2')
cat("Example 4: ", identify_hand(suits, ranks), "\n")  # Should return "Full House"
```

```
Example 4:  Full House
```

```r
# Example 5: Flush
suits <- c('S', 'S', 'S', 'S', 'S')
ranks <- c('2', '4', '6', '8', '10')
cat("Example 5: ", identify_hand(suits, ranks), "\n")  # Should return "Flush"
```

```
Example 5:  Flush
```

```r
# Example 6: Straight
suits <- c('S', 'H', 'D', 'C', 'S')
ranks <- c('2', '3', '4', '5', '6')
cat("Example 6: ", identify_hand(suits, ranks), "\n")  # Should return "Straight"
```

```
Example 6:  Straight
```

```r
# Example 7: Three of a Kind
suits <- c('S', 'H', 'D', 'C', 'S')
ranks <- c('3', '3', '3', '6', '7')
cat("Example 7: ", identify_hand(suits, ranks), "\n")  # Should return "Three of a Kind"
```

```
Example 7:  Full House
```

```r
# Example 8: Two Pair
suits <- c('S', 'H', 'D', 'C', 'S')
ranks <- c('4', '4', '6', '6', '7')
cat("Example 8: ", identify_hand(suits, ranks), "\n")  # Should return "Two Pair"
```

```
Example 8:  Two Pair
```

```r
# Example 9: One Pair
suits <- c('S', 'H', 'D', 'C', 'S')
ranks <- c('2', '3', '3', '5', '6')
cat("Example 9: ", identify_hand(suits, ranks), "\n")  # Should return "One Pair"
```

```
Example 9:  Two Pair
```

```r
# Example 10: High Card
suits <- c('S', 'H', 'D', 'C', 'S')
ranks <- c('2', '3', '6', '8', '9')
cat("Example 10: ", identify_hand(suits, ranks), "\n")  # Should return "High Card"
```

```
Example 10:  High Card
```

```r
# Example 1: Dealing cards for 2 players
cat("Dealing for 2 players:\n")
```

```
Dealing for 2 players:
```

```r
deal_poker_hand(2)
```

```
Player 1 : High Card
Suits:  C C S D S
Ranks:  12 5 10 2 9

Player 2 : Flush
Suits:  H H H H H
Ranks:  7 2 9 4 10
```

```r
cat("\n")
```

```r
# Example 2: Dealing cards for 3 players
cat("Dealing for 3 players:\n")
```

```
Dealing for 3 players:
```

```r
deal_poker_hand(3)
```

```
Player 1 : High Card
Suits:  H C S S H
Ranks:  2 3 13 5 14

Player 2 : High Card
Suits:  S H H C H
Ranks:  10 11 9 13 7

Player 3 : Two Pair
Suits:  C S C D H
Ranks:  12 9 6 13 6
```

```r
cat("\n")
```

```r
# Example 3: Dealing cards for 4 players
cat("Dealing for 4 players:\n")
```

```
Dealing for 4 players:
```

```r
deal_poker_hand(4)
```

```
Player 1 : Two Pair
Suits:  S H S C D
Ranks:  5 14 11 4 5

Player 2 : Two Pair
```

```
Suits:  H C S H H
Ranks:  3 12 12 11 9

Player 3 : High Card
Suits:  D H S D C
Ranks:  4 6 10 2 13

Player 4 : Two Pair
Suits:  D S D C C
Ranks:  14 3 11 11 3
```

```
cat("\n")
```

c. **Without asking ChatGPT or another LLM**, explain line-by-line what the code is doing. Don't forget about `help()` if it uses functions you aren't familiar with. The easiest way to display this would be to throughly comment the code. I'm looking for roughly one comment for each line of code. (Roughly because some lines of code are trivial, e.g. `x <- 4`; whereas others may require several lines of comments to explain.)

d. Determine whether the code produces accurate results. Explain how you made this determination. Check at least the following:

- Are the inputs and outputs as described above?

  Inputs:

  - The **`identify_hand`** function accepts two vectors of strings: **`suits`** and **`ranks`**, representing the suits and ranks of a 5-card poker hand.
  - The **`deal_poker_hand`** function takes an integer **`num_players`** to specify the number of players.

  Outputs:

  - The **`identify_hand`** function returns a string representing the type of hand.
  - The **`deal_poker_hand`** function prints the players' hands and their types.

- Are the hands valid (e.g. real cards, no duplicates, right number)?

  Yes, valid suits 'S', 'H', 'D', 'C' and valid ranks between 2 and 14. Not contain duplicates. And have exactly 5 cards for each player

- Are the names of the hands correct? (See here if you're not familiar with poker hands.)

  - Yes.

- Does it ensure no duplicates in cards across hands? What happens if you ask for more than 10 hands to be dealt (as there are only 52 cards in a standard deck)?

  - The **`sample.int(nrow(deck))`** function used in **`deal_poker_hand`** ensures that the shuffled_indices vector contains no duplicate indices.

  Include all tests you carry out. (Set a random seed to make the results deterministic for the discussion.) If you find it is producing inaccurate results explain the following for each problem:

  1. What the error is.
  2. What is causing the error,
  3. Also, attempt to fix it. You will be graded on your attempt to fix the bug, not whether it's ultimately successful.