UNIVERSITY OF BIRMINGHAM

MSc Bioinformatics - Essentials of Mathematics, and Statistics - Coursework

Ekarsi Lodh

2025-12-12

1 Markov Chain Game Simulations

Markov chains are stochastic processes that move between states with certain probabilities. In this question, you will simulate a simple "Mountain Climb" board game using a Markov chain approach. The rules of the game are as follows:

- **Start:** Player begins at the base of the mountain (position **0**).
- **Goal:** Reach the summit (position **70**).
- **Movement:**
  - Each turn, the player rolls a fair **six-sided die (1–6)**.
  - The player moves forward by the number rolled.
- **Slippery squares (penalty):**
  - Triggered when landing on a **multiple of 9**:
    - 9, 18, 27, 36, 45, 54, 63
  - The player slides back **1–3 positions**, chosen uniformly at random.
- **Climbing squares (bonus):**
  - Triggered when landing on a **multiple of 16**:
    - 16, 32, 48, 64
  - The player moves **+5 additional positions**.
- **End condition:**
  - The game ends immediately when the player reaches or exceeds **position 70**.

Q1

Review the following code snippet, which attempts to simulate a simple board game. Adapt it to the Mountain Climb rules above. Comment each line, explain its purpose, and fix any mistakes. **[3 MARKS]**

```
play_mc <- function(start_pos = 0) {
  die <- sample.int(6, 1)
  pos <- start_pos

  pos <- pos + die
```

```
  if (pos < 70) {
    pos <- 70
    return(pos)
  }
}
```

Answer

The board game "Mountain Climb" represents a **Markov Chain** model, in which the **next position** of the player in the game at any given point of time **depends solely on the current position** and not on the positions which came before it, the turns that were taken, or the routes that were traveled.

In each turn, a **fair six-sided die** is rolled, with the results falling within the set {1, 2, 3, 4, 5, 6}, and the player's position advances according to the number rolled.

The states gets updated as follows:

1. The state refers to the current position, i.e.

$$\textbf{the state} = \textbf{current position}$$

2. The next state is determined by adding the die roll to the current position, factoring in any effects from special squares (such as sliding or climbing), represented as,

$$\textbf{next state} = \textbf{current position} + \textbf{die roll} \pm \textbf{special square effects (slippery/climbing)}$$

3. The starting position is set to 0, i.e.

$$\textbf{start position} = 0$$

4. The game concludes when the player **reaches or surpasses position 70**.

Regarding special square effects:

1. **Slippery squares**:
   When players land on **multiples of 9**, specifically at positions 9, 18, 27, 36, 45, 54, and 63, they incur a penalty. Landing on any of these squares results in a slide back of either **1, 2, or 3 spaces**, with each outcome being **equally probable (1/3 chance for each)**.

The transition from these positions is as follows:

- a **1/3 probability of moving to pos - 1**
- a **1/3 probability of moving to pos - 2**
- a **1/3 probability of moving to pos - 3**

2. **Climbing squares**:
   When players land on **multiples of 16**, specifically at positions 16, 32, 48, and 64, they receive a benefit. Landing on any of these squares results in a **move ahead by 5 spaces (a JUMP)**, accelerating their progress. In this scenario:

- next position is calculated as current position + 5, i.e.

$$\textbf{next position} = \textbf{current position} + 5$$

The given code is incorrect for the following reasons:

1. The code attempting to simulate a simple board game has a critical error in the last three lines, as the function wrongly sets **the position to 70 whenever the player is below 70**. Consequently, the game would consistently conclude after a single turn, since the position would only range between 1 and 6 after the initial roll, all of which are less than 70.
2. The function does not account for the rules regarding **slippery and climbing squares**. I have made corrections in that regard.
3. There is **no observable Markov chain behaviour**. A Markov chain requires that each subsequent state is determined solely by the current state, but this function makes an **abrupt jump from 0 to 70 in a single step (pos <- 70)**, bypassing any transitions.
4. The function **ends prematurely** since there isn't a loop, the game state isn't updated repeatedly.

The correct logic is as follows: To accurately model the "Mountain Climb Game", the following changes are made to the function so that:

a. Each move is determined by a roll of a fair die.

b. Landing on a slippery square causes the player to move backward by between 1 and 3 steps.

c. Landing on a climbing square propels the player forward by +5 steps.

d. The position is updated progressively, creating a series of states that align with a Markov chain.

e. The game concludes only when the position reaches or surpasses 70.

```
#function to simulate a single turn ONE TURN on the mountain climb game
play_mc <- function(start_pos = 0) {

  #roll a fair 6 sided die, with values  {1, 2, 3, 4, 5, 6},
  #while returning an integer between (1 to 6)
  die <- sample.int(6, 1)

  #stores current position, here the current position is the starting position
  pos <- start_pos

  #moves formard by the die roll
  pos <- pos + die

  #-----Apply the special rules if the player haven't reached the summit-----
  if (pos < 70) {

    #-----Slippery squares; i.e. player lands on multiple of 9-----
    if (pos %% 9 == 0) {
      slide <- sample(1:3, 1)       #uniformly choose 1-3
      pos <- pos - slide            #slides back uniformly by random 1-3 spaces
      pos <- max(pos, 0)            #avoid negative position if pos < slide
    }

    #-----Climbing squares; i.e. player lands on multiple of 16-----
    if (pos %% 16 == 0) {
      pos <- pos + 5                #jumps 5 positions
    }

    #if after all moves player reaches or exceeds 70, end immediately
    if (pos >= 70) {
      pos <- 70                     #cap the position at 70 (maximum)
```

```
      return(pos)
    }

    #otherwise return the updated position
    return(pos)
  }

  #if the pos is already >= 70
  if (pos >= 70) {
    pos <- 70                        #cap to 70
    return(pos)
  }
}
```

Just testing the above function To test the above function and to simulate full games with **turns 100, 200, 300, and 400** the following is done.

```
#set the seed for reproducible simulations
set.seed(200)

#simulations for 100,200,300,400 times run
res_100 <- replicate(100, mc_sims())          #100 independent simulations
res_200 <- replicate(200, mc_sims())          #200 independent simulations
res_300 <- replicate(300, mc_sims())          #300 independent simulations
res_400 <- replicate(400, mc_sims())          #400 independent simulations
```

The `mc_sims()` function simulates one full game of Mountain Climb by calling the **`play_mc`** transition function, which simulates one turn and its consequence, i.e., transition. Each iteration of the while loop corresponds to **one Markov chain transition**, encompassing a die roll and any special square effects that may apply. The loop terminates as soon as the position reaches or exceeds 70, which is the **absorbing state of the Markov chain**. The output indicates the **number of turns (transitions)** required to **finish the game or reach the absorbing state**.

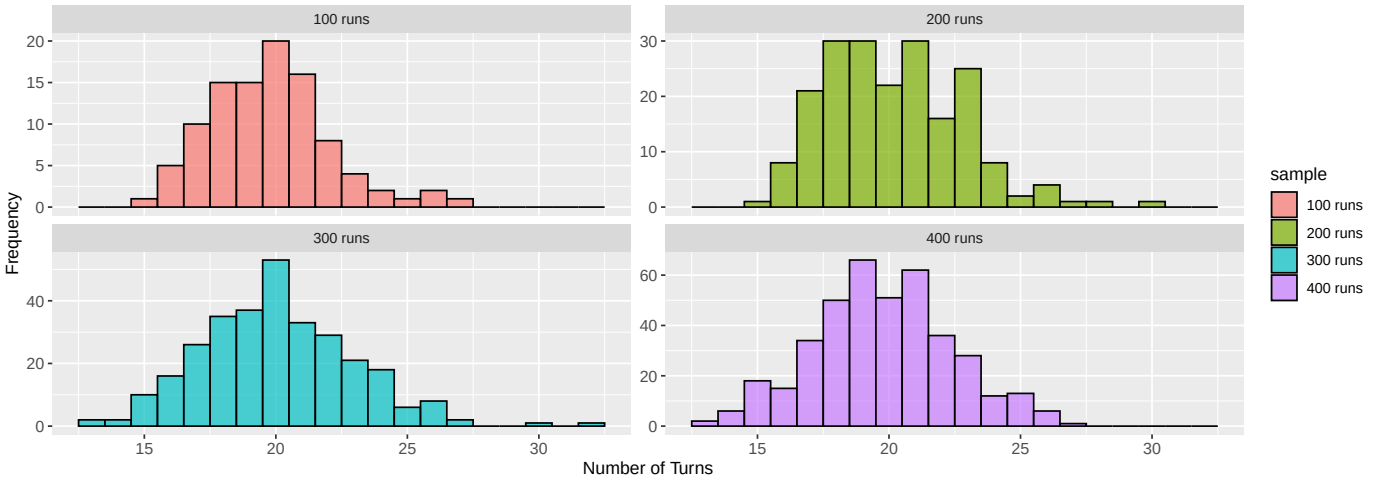
Then, after establishing a seed to ensure consistent results across multiple executions of the function (allowing for exact replication of outcomes), the **`simulate_game()`** function is executed to play the full game 100, 200, 300, and 400 times, each with its unique random die rolls. This allows comparison of how the **distribution stabilizes or stays unchanged as sample size increases**. The turn count distribution will converge to the Markov chain's true underlying distribution as the sample size increases.

The following code chunk is for generating Fig. 1:

```
# Creating a clean data frame by combining the
#simulation result vectors (100, 200, 300, 400 runs) and
#labeling each group so they may be compared or visualized independently.

df <- bind_rows(
  data.frame(turns = res_100, sample = "100 runs"),
  data.frame(turns = res_200, sample = "200 runs"),
  data.frame(turns = res_300, sample = "300 runs"),
  data.frame(turns = res_400, sample = "400 runs"),
)

#generating the plots
#'turns' as the x-axis variable, and fill based on the no. of runs
ggplot(df, aes(x = turns, fill = sample)) +
#histogram with 20 bins each
  geom_histogram(color="black", bins=20, alpha=0.7) +
  #separate panel for each no. of runs, also different y scales
```

```r
  facet_wrap(~ sample, scales="free_y") +

  #plot appearance customization
  theme(
    plot.title = element_text(               #title styling
      size = 13,
      face = "bold",                         #making bold
      hjust = 0.5                            #centering
    ),
    axis.title.x = element_text(size = 11),  #X-axis title font size
    axis.title.y = element_text(size = 11),  #Y-axis title font size
    axis.text.x = element_text(size = 10),   #X-axis tick labels size
    axis.text.y = element_text(size = 10)    #Y-axis tick labels size
  ) +

  #adding plot title
  labs(
    title = "Fig. 1: Distribution of Turns Needed to Finish the Game in Each Simulation",
                          #plot title
    x = "Number of Turns", #x-axis label
    y = "Frequency"        #y-axis label
  )
```

Fig. 1: Distribution of Turns Needed to Finish the Game in Each Simulation



Q2

Extending the code above (or writing your own), implement a function that simulates a complete Mountain Climb game. It should track the player's position after each turn and return the total number of turns needed to reach the summit. **[5 MARKS]**

- Function name: `play_mc_full`
- Function input: `start_pos`, `max_turns`, `slippery_squares`, `climbing_squares`, `slip_back_range`, `climb_up_amount`, `max_pos`
- Function output: list with elements `pos_history` and `turns_taken`

Use sensible default values for the function parameters. The function should simulate the game according to the rules provided above without any user input during execution, or choice of parameters outside of the function call, but the user should be able to specify different parameters when calling the function. Execute the function to demonstrate it works as expected, using the default parameters and show the output.

Answer

To simulate a complete implementation of the "Mountain Climb" game, I implemented a function **`play_mc_full()`**, in the *`cw_functions.R`* file, which repeatedly applies the **transition rules of the Markov chain** until the player **reaches or exceeds the summit at position 70**. The function accepts flexible inputs so that the behaviour of the game can be modified, but I have also provided sensible defaults which corresponds exactly to the rules given in the question.

The function begins at a user-specified starting position violet*(default set to 0)* and rolls a fair six-sided die on each turn. After each move, it checks whether the player has landed on a *slippery square (multiples 0f 9)* or a *climbing square (multiples 0f 16)*. Slippery suqares apply a backward movement **drawn uniformly from 1-3** steps,while climbing squares move the player forward by a fixed amount *(default +5)*. These transition rules make the process a **Markov chain**, because the next state depends only on the current position and not on the history of the game.

The function continues to update the player's position until *one of the two* stopping conditions is reached:

i. the player **reaches or passes the summit position** *(default set to 70)*

ii. a safety cap on the total number of turns **(`max_turns`)** is exceeded.

Throughout the simulation, the function records the full position history, which allows the entire state trajectory to be examined afterwards. Finally, it returns a list containing the following:

- **`pos_history:`** a vector of all positions visited in order, and
- **`turns_taken:`** the total number of turns required to reach the summit.

To show that the function works as expected, I executed it with the default parameters and printed the no. of turns taken as well as the begining and end of the position history.

```
set.seed(200)                          #for reproducibility
default_run <- play_mc_full()          #calling the function with the default parameters.

default_run$pos_history                #displaying the positions
```

```
##  [1]  0  6  8 12 21 26 28 30 33 35 39 42 46 52 57 61 65 71
```

```
default_run$turns_taken                #displaying the no. of turns taken
```

```
## [1] 17
```

The simulation's reproducibility is guaranteed by using a fixed random seed. The output reported is consistent since set.seed(200) ensures that the dice rolls and random slip-back values are the same each time the code block is executed. A vector storing all of the positions visited and the total no. of turns needed to reach the summit are the two outputs of running **`play_mc_full()`** function.

In this simulation, the player advances steadily until position 70, occasionally encountering forward jumps on multiples of 16 and backward slides on multiples of 9. The stochastic aspect of the transition dynamics is reflected in the precise order of visited positions: favourable positions greatly expedite movement, while small die rolls can impede progress.

Q3

Now write a function to simulate multiple games of mountain climb and calculate the average number of turns taken to reach the summit across all simulations. **[4 MARKS]**

- Function name: `simulate_climbs`
- Function input: `n_simulations`, other parameters as in `play_mc_full`
- Function output: List with elements

- `turns_vector`: number of turns taken in each simulation
- `turns_stats`: named vector with mean, median, sd (in this order)
- `all_histories`: list of pos_history vectors for each simulation

Run the simulation for 100 games and report the average number of turns taken to reach the summit. Plot a histogram of the number of turns taken across all simulations.

Answer

I implemented a function **`simulate_climbs()`**, which is designed to automate the repeated simulation of the "Mountaion Climn" game by calling ***`play_mc_full()`*** multiple times and summarising the results. The function takes the no. of simulations **`(n_simulations)`** as its main argument, along with the same parameter set used by ***`play_mc_full()`***, which allows the user to easily alter the game rules if needed. Inside the function, a numeric vector **`turns_vector`** is created to store the no. of turns taken to reach the summit in **each individual game**, while a list **`all_histories`** is used to store the full sequence of positions visited for every simulations. **For each run**, the function executes ***`play_mc_full()`***, extracts the final turn count and positions, and appends them to the corresponding storage objects. Once all simulations are complete, the function computes certain statistical measures; *mean, median, and standard deviation* of the turn counts, which is returned in the named vector **`turns_stats`**.

The output of the function is therefore a structured list containing three components that together summarise both the duration and detailed behaviour of all the simulated games:

- **`turns_vector:`** reporting the no. of turns taken in each game,
- **`turns_stats:`** providing simple summary statistics of these durations, and
- **`all_histories:`** enabling further inspection or visualization of individual game trajectories.

```r
#function sourced from cw_functions.R (Q3)
simulate_climbs <- function(
  n_simulations = 100,                      #number of games to simulate
  start_pos = 0,                            #starting position of the player
  max_turns = 500,                          #safety cap on no. of turns
  slippery_squares = seq(9, 63, by = 9),    #default: multiples of 9 upto 63,
                                            #denoting slippery squares
  climbing_squares = seq(16, 64, by = 16),  #default: multiples of 16 upto 64,
                                            #denoting climbing squares
  slip_back_range = 1:3,                    #default: player slides back by
                                            #1, 2 or 3 positions on slippery
                                            #squares
  climb_up_amount = 5,                      #default: player climbs +5
                                            #positions on climbing squares
  max_pos = 70                              #default: summit position
) {

  #numeric vector to store the no. of turns taken in each simulation
  turns_vector <- numeric(n_simulations)

  #list for storing position history of each game
  all_histories <- vector("list", n_simulations)

  #run simulations
  for (i in seq_len(n_simulations)) {
    #run a complete mountain climb game with all the parameters
    game_result <- play_mc_full(
      start_pos = start_pos,
      max_turns = max_turns,
      slippery_squares = slippery_squares,
      climbing_squares = climbing_squares,
      slip_back_range = slip_back_range,
```

```
      climb_up_amount = climb_up_amount,
      max_pos = max_pos
    )

    #saving the no. of turns required
    turns_vector[i] <- game_result$turns_taken

    #saving the positions visited
    all_histories[[i]] <- game_result$pos_history
  }

  #-----Summary statistics for no. of turns taken-----
  turns_stats <- c(
    mean = mean(turns_vector),              #avg no. of turns
    median = median(turns_vector),          #middle value
    sd = sd(turns_vector)                   #standard deviation, value spread
  )

  #-----Return as a single list-----
  return(
    list(
      turns_vector = turns_vector,    #no. of turns for each simulation
      turns_stats = turns_stats,      #summary stats of turns
      all_histories = all_histories   #position visited in all simulations
    )
  )
}
```

```
set.seed(200)                          #for reproducibility
n_simulations = 100                    #defining no. of simulations to run

#simulating the game with default parameters
sim_results <- simulate_climbs(n_simulations = n_simulations)

#summary statistics (mean, median, sd of turns)
sim_results$turns_stats
```

```
##      mean    median        sd
## 19.760000 20.000000  2.305549
```

```
#mean no. of turns dor 100 games
sim_results_mean <- sim_results$turns_stats["mean"]
paste("Mean of the no. of turns for", n_simulations, "games =", sim_results_mean)
```

```
## [1] "Mean of the no. of turns for 100 games = 19.76"
```

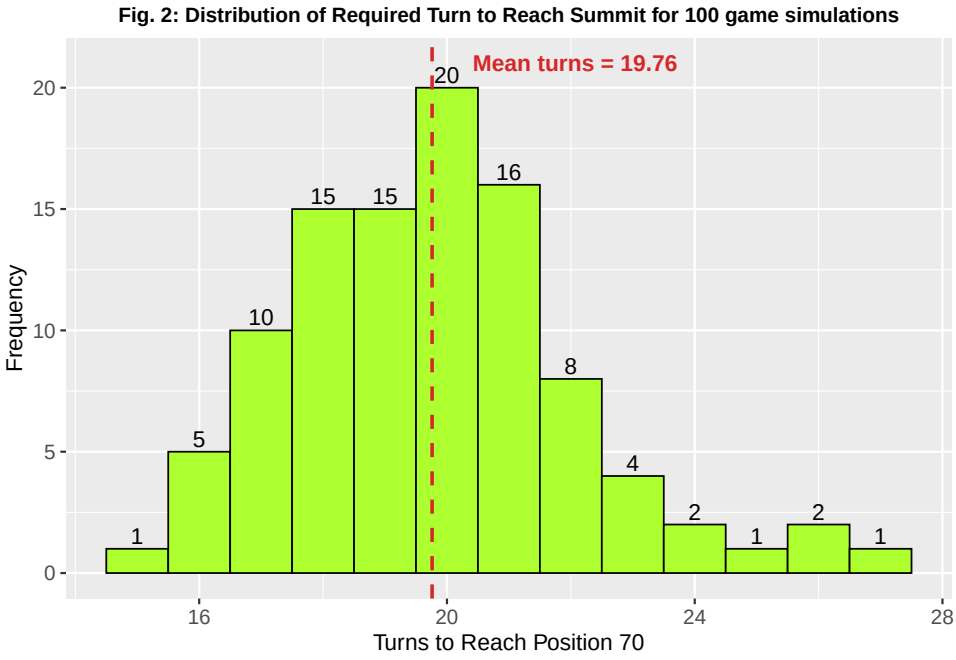
The simulation's reproducibility is guaranteed by using a fixed random seed of 200 *(`set.seed(200)`)*. The results are saved in ***`sim_results`*** after the simulation is run for 100 separate games. The overall average no. of turns required to reach the summit is recorded in **`sim_results_mean`** and also a turn count histogram is created using the following code using ***`ggplot2`***, to show the variability throughout 100 simulations.

The following code chunk is for generating Fig. 2:

```
#converting turns_vector to dataframe for ggplot
turns_df <- data.frame(turns = sim_results$turns_vector)
```

```r
#computing histogram bin max height
bin_info <- hist(turns_df$turns, breaks = 13, plot = FALSE)
max_y <- max(bin_info$counts) + 1      #set max y height as max bin + 1
#print(max_y)

#ggplot histogram using no. of turns taken in each simulation
ggplot(
  turns_df,                            #turns_vector df
  aes(x = turns)                       #x-axis represents no. of turns
) +

#plotting the histogram
geom_histogram(
  bins = 13,                           #20 histogram bins
  fill = "greenyellow",                    #fill colour
  color = "black",                     #border colour
  linewidth = 0.4                      #outline thickness
) +

#frequency on top of each bar
stat_bin(
bins = 13,                             #ensuring text labels correspond exactly
                                       #to the same bin
geom = "text",                         #text labels instead of bars
aes(label = after_stat(count)),        #observation frequency in each bin
vjust = -0.3,                          #labels slightly above each bar
size = 4,                              #size of the text labels
color = "black"                        #label colour
) +

#vertical dashed line for the mean no. of turns
geom_vline(
  xintercept = sim_results_mean,       #mean of no. of turn counts
  linetype = "dashed",                 #dashed line
  linewidth = 0.8,
  color = "#D62728"
) +

#annotate the mean value
annotate(
  "text",                                              #text annotation
  x = sim_results_mean,                                #text at mean on x-axis
  y = Inf,                                             #text at top of plot
  label = paste("Mean turns =", sim_results_mean),     #label text
  vjust = 2,                                           #text slightly downwards
                                                       #to contain in frame
  hjust = -0.2,                                        #text slightly right
                                                       #shifted for better readability
  size = 4,
  color = "#D62728",
  fontface = "bold"
) +

#adjusted y limit
ylim(0, max_y) +

#plot appearance customization
theme(
  plot.title = element_text(               #title styling
    size = 10,
```

```
    face = "bold",                          #making bold
    hjust = 0.5                             #centering
  ),
  axis.title.x = element_text(size = 11),   #X-axis title font size
  axis.title.y = element_text(size = 11),   #Y-axis title font size
  axis.text.x = element_text(size = 10),    #X-axis tick labels size
  axis.text.y = element_text(size = 10)     #Y-axis tick labels size
) +

#adding plot title
labs(
  title = "Fig. 2: Distribution of Required Turn to Reach Summit for 100 game simulations",
                                          #main title
  x = "Turns to Reach Position 70",       #x-axis label
  y = "Frequency"                         #y-axis label
)
```



The distribution of turns required for reaching the summit (position 70) throughout the individual 100 games simulations is shown in the histogram *(Fig. 2)*. The y-axis shows how frequently each turn count occurs, while the x-axis shows how many turns are taken during a game. The distribution's central tendency is emphasized by the addition of a vertical dashed line representing the *mean*. The stochastic character of the game, which is influenced by special-square effects and random die rolls, is reflected in the histogram's shape.

Distribution Result Interpretation

The histogram shows the distribution of the number of turns required to reach or exceed the summit i.e. position 70 throughout 100 independently simulated "Mountain Climb" games. The results are centered around approximately **20 turns**, with the ***sample mean*** equal to **19.76**, indicated by the *red dashed line*. This suggests that, under the given rules, a typical game requires *close to 20 turns* to reach the absorbing state.

The distribution is moderately spread, with most games **completing between 17 and 23 turns**, and a few *outliers* taking as few as **15 turns** or as many as **27 turns**. The variation reflects the ***stochastic nature*** of the Markov chain; where the **favourable sequences of die rolls** and **landing on climbing squares** *(multiples of 16)* can shorten the game, while **landing on slippery squares** *(multiples of 9)* can prolong it. Despite this randomness, the distribution remains **unimodal** with a clear **central tendency around 20 turns**, but it shows a **slight right skew**; where most games finish within a narrow range, with occasional longer games extend the upper (right) tail.

Overall, it can be said that the games ***finishing earlier than the mean*** benefit from **favourable sequences of die rolls** and **landing on climbing squares**, multiple times producing *forward jumps*, which end the game early, whereas the games ***taking longer than mean*** is expected to experience **more of landing on slippery squares**, introducing *random backward slips*, delaying the game. Overall the results illustrate how the mountain climb process behaves on average and demonstrate that the simulation captures the combined effects of random movement and state dependent transitions. The estimated mean turn count provides a ***Monte-Carlo approximation*** of the expected number of steps required to reach the summit.

Q4

4.1 Using the simulation code above, estimate the average position reached by a single player after 10 turns and the variance. Plot a histogram of the position reached after 10 turns. [4 MARKS]

4.2 Next change the `slip_back_range` to 4:8, `climb_up_amount` to 2, and re-run the simulation. How does this change affect the average position reached after 10 turns? Explain why this change occurs. [4 MARKS]

You might find it useful to modify the `simulate_climbs` function to return positions after a fixed number of turns instead of the number of turns taken to reach the summit. The modified function should be called `simulate_pos_at`. Everything else should remain the same.

Answer 4.1

The function **`simulate_pos_at()`**, which is sourced from an external script (cw_functions.R), is used to predict a player's projected progress after a predetermined number of turns *(10 in this case)*. This function returns the position for each game after a predefined number of turns *(10)*, in contrast to earlier simulations that continued until the summit was reached or exceeded. This enables analysis of the player's position at that precise moments across 100 game simulations.

```
#function sourced from cw_functions.R (Q4)
simulate_pos_at <- function(
    n_simulations = 100,                    #number of games to simulate
    n_turns = 10,                           #fixed turns after which game stops
    start_pos = 0,                          #starting position of the player
    max_turns = 500,                        #safety cap on no. of turns
    slippery_squares = seq(9, 63, by = 9),  #default: multiples of 9 upto 63,
                                            #denoting slippery squares
    climbing_squares = seq(16, 64, by = 16),  #default: multiples of 16 upto 64,
                                              #denoting climbing squares
    slip_back_range = 1:3,                    #default: player slides back by
                                              #1, 2 or 3 positions on slippery
                                              #squares
    climb_up_amount = 5,                      #default: player climbs +5
                                              #positions on climbing squares
    max_pos = 70                              #default: summit position
) {

  #numeric vector to store the position after n_turns
  pos_vector <- numeric(n_simulations)

  #list for storing position history of each game
  all_histories <- vector("list", n_simulations)

  #run simulations
  for (i in seq_len(n_simulations)) {
    pos <- start_pos                          #reset position at start of each simulation
    pos_history <- c(pos)                     #store all the positions visited

    #simulate exactly the mentioned no. of turns
    for (t in seq_len(n_turns)) {
      #if summit not reached
      if (pos < max_pos) {
```

```
        die <- sample.int(6, 1)                    #roll a fair six-sided die
        pos <- pos + die                           #move forward by the die roll

        #-----Slippery squares; i.e. player lands on multiple of 9-----
        if (pos %in% slippery_squares) {
          slide <- sample(slip_back_range, 1)      #uniformly choose slip amount
          pos <- pos - slide                       #slides back uniformly by random 1-3 spaces
          pos <- max(pos, 0)                       #avoid negative position if pos < slide
        }

        #-----Climbing squares; i.e. player lands on multiple of 16-----
        if (pos %in% climbing_squares) {
          pos <- pos + climb_up_amount             #jumps 5 positions
        }

        #-----Cap at max_pos (absorbing state)-----
        if (pos > max_pos) {
          pos <- max_pos
        }
      }

      if (pos >= max_pos) {
        pos <- max_pos
      }

      #store tha pasitions visited
      pos_history <- c(pos_history, pos)
    }

    #position after n_turns
    pos_vector[i] <- pos

    #saving the positions visited
    all_histories[[i]] <- pos_history
  }

  #-----Return summary statistics as a single list-----
  return(
    list(
      positions = pos_vector,  # Final position after 'turns' in each simulation
      mean = mean(pos_vector), # Average position reached after 'turns' turns
      variance = var(pos_vector) # Variance of positions (spread of outcomes)
    )
  )
}
```

```
set.seed(200)                          #for reproducibility
n_simulations = 100                    #defining total no. of simulations to run
n_turns = 10                           #turn after which to stop

#simulating the game with default parameters
sim_10_results <- simulate_pos_at(n_simulations = n_simulations, n_turns = n_turns)

#position reached after 10 turns on average
sim_10_results_mean <- sim_10_results$mean
paste("The position reached after 10 turns on average i.e. mean =", sim_10_results_mean)
```

```
## [1] "The position reached after 10 turns on average i.e. mean = 36.32"
```

```r
#variance of the positions reached after 10 turns
#measures the spread of final positions
sim_10_results_variance <- sim_10_results$variance
paste("The variance =", round(sim_10_results_variance, 2))   #round upto 2 places
```

```
## [1] "The variance = 31.84"
```

The simulation's reproducibility is guaranteed by using a fixed random seed of 200 *(`set.seed(200)`)*. The results are saved in ***`sim_10_results`*** after the simulation is run for 100 separate games, each ending after 10 turns. The overall *average of the positions reached after 10 turns* for 100 game simulations is recorded in **`sim_10_results_mean`** and the variance is saved in **`sim_10_results_variance`**.

The player begins each simulation at **position 0** and advances in accordance with the standard "Mountain Climb" rules: a fair six-sided die determines forward movement, **landing on a slippery square (multiples of 9)** causes a **random backward slide of 1, 2, or 3 spaces**, **landing on a climbing square (multiples of 16)** gives a **bonus forward jump of 5 spaces**.

The average progress of the player under default rules is represented by the mean, whereas the variance represents the variability in progress due to the stochastic nature of the game. Consequently, the **mean** offers a ***practical approximation of the anticipated position following 10 turns***, while the **variance** illustrates the ***extent to which individual results differ from this anticipated value or mean***.

The obtained results are:

- **Mean:** 36.32 *(for a seed 200)*.
- **Variance:** 31.84 *(for a seed 200)*.

The following code chunk is for generating Fig. 3:

```r
#converting turns_vector to dataframe for ggplot
pos_df <- data.frame(positions = sim_10_results$positions)

#ggplot histogram using no. of turns taken in each simulation
ggplot(
  pos_df,                                          #position vector df
  aes(x = positions)                               #x-axis represents positions
) +

#plotting the histogram
geom_histogram(
  binwidth = diff(range(pos_df$positions)) / 10,  #20 histogram bins
  fill = "yellowgreen",                            #fill colour
  color = "black",                                 #border colour
  linewidth = 0.4                                  #outline thickness
) +

#frequency on top of each bar
stat_bin(
binwidth = diff(range(pos_df$positions)) / 10,    #ensuring text labels correspond
                                                 #exactly to the same bin
geom = "text",                                   #text labels instead of bars
aes(label = after_stat(count)),                  #observation frequency each bin
vjust = -0.3,                                    #labels slightly above each bar
size = 4,                                        #size of the text labels
color = "black"                                  #label colour
) +

#vertical dashed line for the mean no. of turns
```

```r
geom_vline(
  xintercept = sim_10_results_mean,      #mean of no. of turn counts
  linetype = "dashed",                 #dashed line
  linewidth = 0.8,
  color = "#D62728"
) +

#annotate the mean value
annotate(
  "text",                                                #text annotation
  x = sim_10_results_mean,                               #text at mean on x-axis
  y = Inf,                                               #text at top of plot
  label = paste("Mean turns =", sim_10_results_mean),#label text
  vjust = 2,                                             #text slightly downwards
                                                         #to contain in frame
  hjust = 1.1,                                           #text slightly left
                                                         #shifted for better readability
  size = 4,
  color = "#D62728",
  fontface = "bold"
) +

#plot appearance customization
theme(
  plot.title = element_text(               #title styling
    size = 10,
    face = "bold",                         #making bold
    hjust = 0.5                            #centering
  ),
  axis.title.x = element_text(size = 11),  #X-axis title font size
  axis.title.y = element_text(size = 11),  #Y-axis title font size
  axis.text.x = element_text(size = 10),   #X-axis tick labels size
  axis.text.y = element_text(size = 10)    #Y-axis tick labels size
) +

#adding plot title
labs(
  title = "Fig. 3: Distribution of Position Post 10 Turns Throughout 100 Simulations",
                                         #main title
  x = "Position",                        #x-axis label
  y = "Frequency"                        #y-axis label
)
```

Fig. 3: Distribution of Position Post 10 Turns Throughout 100 Simulations

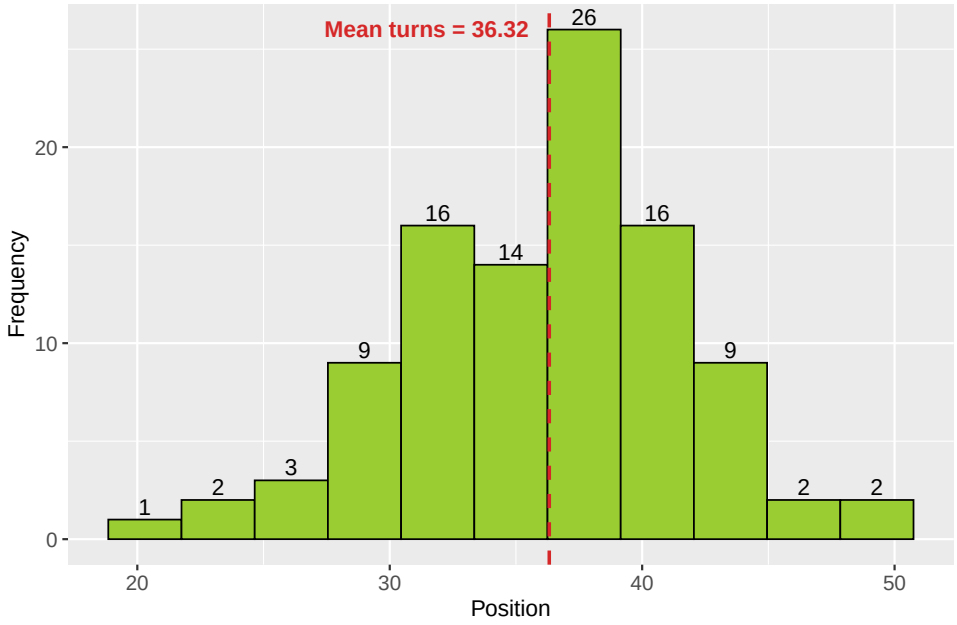


Figure 3 displays the distribution of **player positions after 10 turns** throughout 100 independent simulations of the "Mountain Climb" game. The ***mean position*** of the *10 turns* is approximately **36.32** as indicated by the vertical dashed **red** line, and the *tallest bar* sits **immediately around this value**, showing most simulations **cluster in the central region**. The frequency of the different states after 10 transitions per game simulation are represented by each of the bars, and the ***empirical estimate of the expected state at the 10th turn*** is represented by the mean.

Key Observations

- The distribution is **unimodal** with a **slight left skew**.
- While most outcomes **cluster around the mean**, lying between **30 and 42, with tallest bar (freq ≈ 26)** just above *mean*, a few simulations reach notably ***higher positions (up around 50)***, which forms a ***thinner right tail***.
- Most of the simulated positions **fall between 30 and 42**, forming a ***clear central peak***. This suggests that *after 10 turns*, players typically advanced to the ***mid-range of the board***, well *below the summit at position 70* but far from the starting point.
- **Lower-end positions (20 - 28)** occur in fewer simulations, indicating *slower early progress* in those games.
- The overall range of positions **(≈19 to ≈50)** shows substantial variability after the same number of turns.

Reasons for the Observations

- **Higher than average** positions arise from:
  - **early rolls were favourable** or
  - **landing on climbing squares** provided additional movement
- **Lower than average** positions arise from:
  - **landing on slippery squares** frequently,
  - resulting in **backward movement** slowing overall progress.

Together, these effects create the distribution seen in the figure; centred around the mean, but with clear variation driven by the game's stochastic movement and special square mechanisms.

Answer 4.2

To evaluate the impact of the change of parameters on short-term outcomes the similar function **simulate_pos_at()**, is invoked with *(set.seed(200))* for reproducibility, and changes made to the following parameters:

- **slip_back_range** set to **4:8** from the default *1:3*, representing a stricter slipping rule, and
- **climb_up_amount** reduced to **2** from the default *5*, representing a less generous climbing bonus.

This adjustment indicates that player landing on a slippery square (multiples of 9), now slide back **4 to 8** positions rather than the original *1 to 3* positions. Also when a player lands on a climbing square (multiple of 16), now jumps **2** positions rather than the original *5* positions.

```
set.seed(200)                          #for reproducibility
n_simulations = 100                    #defining total no. of simulations to run
n_turns = 10                           #turn after which to stop

#setting new slip_back_range & climb_up_amount
slip_back_range = 4:8                  #player slides back by 4 to 8 positions
climb_up_amount = 2                    #player climbs +2 positions

#simulating the game with new parameters
sim_10_results_new <- simulate_pos_at(n_simulations = n_simulations,
                                      n_turns = n_turns,
                                      slip_back_range = slip_back_range,
                                      climb_up_amount = climb_up_amount
                                      )

#position reached after 10 turns on average
sim_10_results_mean_new <- sim_10_results_new$mean
paste("The position reached after 10 turns on average i.e. mean for new parameters =",
      sim_10_results_mean_new)
```

```
## [1] "The position reached after 10 turns on average i.e. mean for new parameters = 29.88"
```

```
#variance of the positions reached after 10 turns
#measures the spread of final positions
sim_10_results_variance_new <- sim_10_results_new$variance
paste("The variance for new parameters=", round(sim_10_results_variance_new, 2))
```

```
## [1] "The variance for new parameters= 71.88"
```

Effect on Average Position Reached Post 10 Turns

Changing the slipping and climbing rules has a clear impact on how players progress in the first 10 turns. Based on the default values, the players typically reached *mid 30s* after the *initial 10 turns*, but on the other hand when the **slip back range** is *increased* from **1-3 to 4-8** and the jumping bonus or the **climbing bonus** is *reduced* from **+5 to +2**, the ***average position reached*** after 10 turns *decreases* and that occurs as the game introduces ***larger backward movements (penalties)*** and ***smaller forward reward*** reducing the overall net forward progress.

These drops the average position (mean) to **29.88 from 36.32** throughout 100 game simulations, as by *strengthening the penalty on slippery squares* and *reducing reward on climbing squares*, the player makes *less progress* post same number of turns on the board. Thus, we **shift the distribution** of positions to the **left** and also decrease the expected position at turn 10.

Why the mean decreases

- **Larger negative jumps** increase expected backward movement per turn.
  Under the new rule, landing on a slippery square produces a ***mean backward movement*** of:

  - **old rule:** $E[\textbf{slip}] = 2$
  - **new rule:** $E[\textbf{slip}] = 6$
    This *tripling* of the expected backward penalty shifts the overall distribution of positions *downward*.

- **Reduced climb reward** lowers the expected forward gain.
  The climb bonus *drops* from **+5 to +2**, meaning the *positive offset* that previously *counteracted the slips* is now much *weaker*.

  - **old expected gain:** +5
  - **new expected gain:** +2
    This results in a much weaker forward reward.

- **Net expected movement** per turn becomes smaller.
  Because negative movements become stronger and positive movements weaker, the expected *step size* in the Markov chain *decreases*. This lowers the mean of the position distribution after a fixed number of turns (here 10).

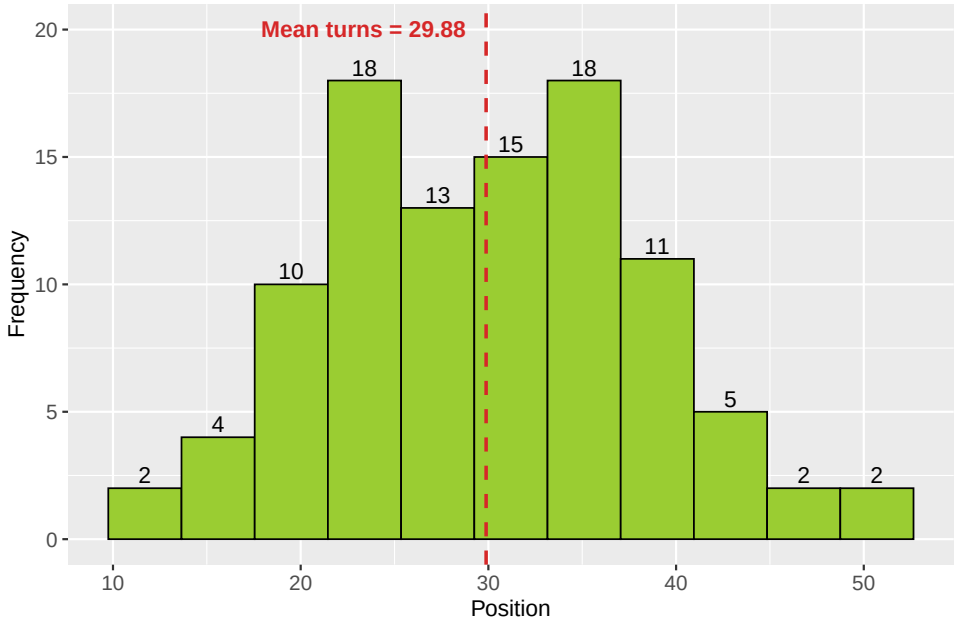
The adjustments collectively decrease the anticipated net forward movement with each turn, causing the Markov chain to be biased towards lower states. Over 10 turns, this influence compounds, leading to a decreased average position and a histogram that *shifts to the left*.

The following code chunk is for generating Fig. 4:

```
#converting turns_vector to dataframe for ggplot
pos_df_new <- data.frame(positions = sim_10_results_new$positions)

#calculating max y
# compute histogram info (without plotting)
hist_info <- hist(pos_df_new$positions, breaks = seq(min(pos_df_new$positions),
                                                     max(pos_df_new$positions),
                                  by = diff(range(pos_df_new$positions)) / 10),
                  plot = FALSE)

max_y <- max(hist_info$counts) + 1

#ggplot histogram using no. of turns taken in each simulation
ggplot(
  pos_df_new,                                          #position vector df
  aes(x = positions)                                   #x-axis for positions
) +

#plotting the histogram
geom_histogram(
  binwidth = diff(range(pos_df_new$positions)) / 10,   #20 histogram bins
  fill = "yellowgreen",                                #fill colour
  color = "black",                                     #border colour
  linewidth = 0.4                                      #outline thickness
) +

#frequency on top of each bar
stat_bin(
binwidth = diff(range(pos_df_new$positions)) / 10,     #ensuring text labels
                                                       #correspond exactly to
                                                       #the same bin
geom = "text",                                    #text labels instead of bars
aes(label = after_stat(count)),                   #observation frequency each bin
vjust = -0.3,                                     #labels slightly above each bar
size = 4,                                         #size of the text labels
color = "black"                                   #label colour
) +
```

```r
#vertical dashed line for the mean no. of turns
geom_vline(
  xintercept = sim_10_results_mean_new,      #mean of no. of turn counts
  linetype = "dashed",                #dashed line
  linewidth = 0.8,
  color = "#D62728"
) +

#annotate the mean value
annotate(
  "text",                                              #text annotation
  x = sim_10_results_mean_new,                            #text at mean on x-axis
  y = Inf,                                             #text at top of plot
  label = paste("Mean turns =", sim_10_results_mean_new),    #label text
  vjust = 2,                                           #text slightly downwards
                                                       #to contain in frame
  hjust = 1.1,                                         #text slightly left shift
                                                       #for better readability
  size = 4,
  color = "#D62728",
  fontface = "bold"
) +

#setting limits for y-axis
ylim(0, max_y) +

#plot appearance customization
theme(
  plot.title = element_text(               #title styling
    size = 10,
    face = "bold",                         #making bold
    hjust = 0.5                            #centering
  ),
  axis.title.x = element_text(size = 11),  #X-axis title font size
  axis.title.y = element_text(size = 11),  #Y-axis title font size
  axis.text.x = element_text(size = 10),   #X-axis tick labels size
  axis.text.y = element_text(size = 10)    #Y-axis tick labels size
) +

#adding plot title
labs(
  title = "Fig. 4: Distribution of Position Post 10 Turns Throughout 100 Simulations (changed rules)",
                                         #main title
  x = "Position",                        #x-axis label
  y = "Frequency"                        #y-axis label
)
```

Fig. 4: Distribution of Position Post 10 Turns Throughout 100 Simulations (changed rules)



To facilitate the visual analysis of the simulation outcomes with modified rules, a histogram was created (Fig. 4) that illustrates the distribution of player positions following 10 turns across 100 simulated games. The x-axis indicates the positions attained at the conclusion of the 10th turn, while the y-axis shows how often each position occurs. The vertical dashed line represents the mean position after 10 turns across all 100 games, which is around 29.88, emphasizing the central tendency of the distribution. This value reflects the average advancement a player achieves after 10 turns under the modified rules.

Each position on the board represents a **state**, with **each die roll** leading to a **random transition**. Altering the slip-back range and the climb bonus, affects the ***transition probabilities*** among states. The adjusted rules enhance both the likelihood and extent of transitions to lower states, while diminishing the chances of upward movement. Consequently, the anticipated state of the chain after 10 transitions is lower, which is evident in both the diminished mean and the altered histogram.

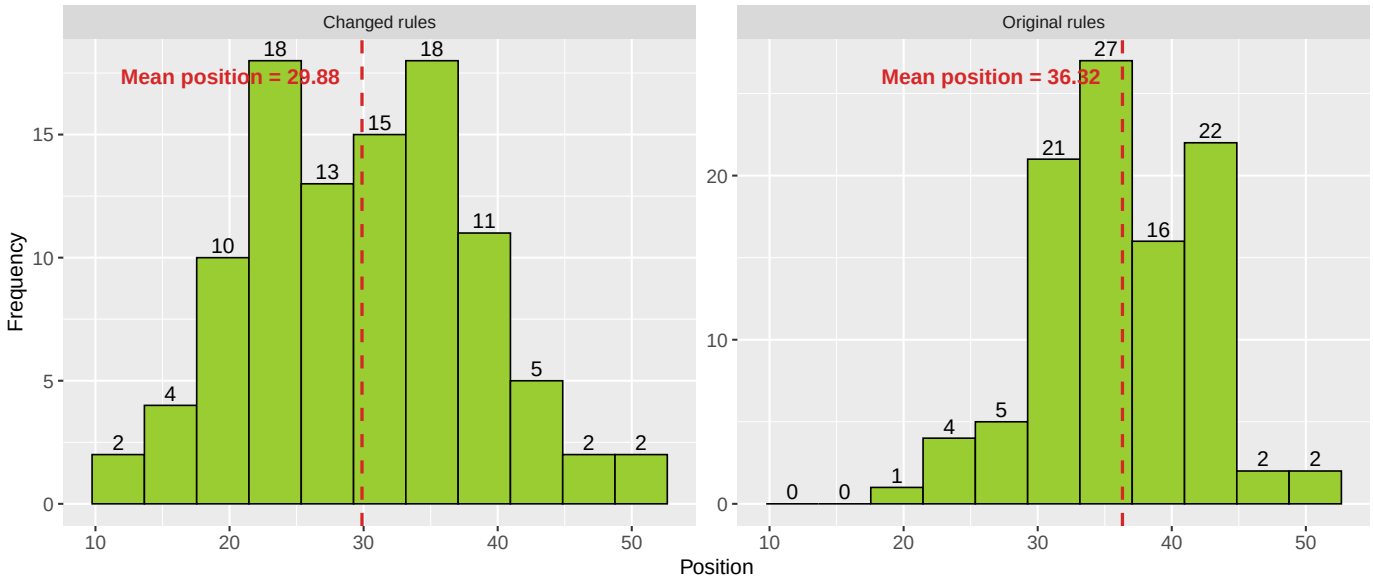
The following code chunk is for generating Fig. 5:

```
#combining the two position vectors into one data frame
pos_combined <- bind_rows(
  data.frame(
    positions = sim_10_results$positions,
    scenario  = "Original rules"
  ),
  data.frame(
    positions = sim_10_results_new$positions,
    scenario  = "Changed rules"
  )
)

#common binwidth across both scenarios
bw <- diff(range(pos_combined$positions)) / 10

#mean position per scenario (for vline + labels)
mean_df <- pos_combined %>%
  group_by(scenario) %>%
  summarise(
    mean_pos = mean(positions),
    .groups  = "drop"
  )
```

```r
#Facetted histogram based on scenario
ggplot(
  pos_combined,
  aes(x = positions)
) +
  #histogram
  geom_histogram(
    binwidth = bw,
    fill     = "yellowgreen",
    color    = "black",
    linewidth = 0.4
  ) +

  #frequency labels on top of bars
  stat_bin(
    binwidth = bw,
    geom = "text",
    aes(label = after_stat(count)),
    vjust = -0.3,           #vertical position
    size = 4,
    color = "black") +

  #vertical dashed line for mean per scenario
  geom_vline(
    data = mean_df,
    aes(xintercept = mean_pos),
    linetype = "dashed",
    linewidth = 0.8,
    color = "#D62728") +

  #annotate mean text per scenario
  geom_text(
    data = mean_df,
    aes(x = mean_pos,
      y = Inf,
      label = paste("Mean position =", mean_pos)),
    vjust = 3,                                  #vertical position
    hjust = 1.1,                                #horizontal position
    size  = 4,
    color = "#D62728",
    fontface = "bold") +

  #facets: one panel per scenario
  facet_wrap(~ scenario, scales = "free_y") +

  #plot appearance
  theme(
    plot.title  = element_text(size = 10, face = "bold", hjust = 0.5),
    axis.title.x = element_text(size = 11),
    axis.title.y = element_text(size = 11),
    axis.text.x  = element_text(size = 10),
    axis.text.y  = element_text(size = 10)
  ) +

  #title and axis labels
  labs(
    title = "Fig. 5: Comparison of Distribution of Position Post 10 Turns Throughout 100 Simulations",
    x     = "Position",
    y     = "Frequency"
```

```
)
```

Fig. 5: Comparison of Distribution of Position Post 10 Turns Throughout 100 Simulations



A side by side comparison of the distribution of positions for the *changed rules* and the *original rules* is presented in Fig. 5. From the figure, it is evident that the distribution **shifts towards left** in application of the changed rules.

Why the distribution shifts left

- **More trajectories accumulate at lower positions.**
  Large backward jumps cause many simulations to fall behind early, and the smaller climb reward makes recovery slower.
- **Upper-tail values become rarer.**
  With weaker climbing boosts, ***fewer simulations can advance far by turn 10***, reducing the frequency of high-end positions (e.g., > 40).
- **Variability increases slightly.**
  Larger slips create greater spread in downward movement, which **widens the left side** of the distribution, while **encompasses the right tail**.

Overall Effect

- The mean position after 10 turns decreases.
- The distribution ***shifts left***, with more simulations ending in lower ranges.
- The game becomes more **'loss-heavy'** where backward movement dominates forward progress.

2 Data Analysis

For this part of the coursework you will need the data file (assess_data_1225.Rdata), instructions to download the file can be found on the course canvas page. This file contains a single data frame with the following 9 variables:

- `idx`: the index of the patient
- `age`: the age of the patient
- `bai`: the body area index of the patient
- `bmi`: the body mass index of the patient
- `body_fat`: the body fat of the patient

- `density`: the density of the patient
- `weight`: the weight of the patient
- `height`: the height of the patient
- `sex`: the sex of the patient

You will perform the analysis in two steps, first you will look at the data to identify any problems with the data followed by a regression based analysis of the data. This is a data analysis exercise and you will be expected to comment on the results of your analysis and plots you produce.

Q5

Study the following code example and add comments to describe what each line of the code does. Also identify any mistakes in the code and comment on how to fix them. **[4 MARKS]**

Note, do not change the code before commenting

```
load("data/assess_data_1225.Rdata")

dens_m <- density(bmi_data$bmi[bmi_data$sex == "male"])
dens_f <- density(bmi_data$bmi[bmi_data$sex == "female"])
plot(dens_f, col = "green")
lines(dens_m)

idx  <- 1:200

pca_res <- prcomp(bmi_data[idx, -c(1, 9)], center = TRUE, scale = TRUE)
plot(pca_res$x[, 1], pca_res$x[, 2], col = "blue", pch = 19)

out <- lm(bmi[idx] ~ age[idx])
summary(out)
p_val <- summary(out)$coefficients[2, 4]
```

Answer 5

The fully commented version of the above code is presented below:

```
#load the 'assess_data_1225' Rdata file from data directory in current R session
#all objects in the Rdata file become available in the workspace
load("data/assess_data_1225.Rdata")

#subset: select rows where "sex" is "male" from "bmi_data" dataframe
#take "bmi" column from those rows
#compute kernel density estimate of BMI values for males only
#store the density object in "dens_m"
dens_m <- density(bmi_data$bmi[bmi_data$sex == "male"])

#subset: select rows where "sex" is "female" from "bmi_data" dataframe
#take "bmi" column from those rows
#compute kernel density estimate of BMI values for females only
#store the density object in "dens_f"
dens_f <- density(bmi_data$bmi[bmi_data$sex == "female"])

#plot the female BMI density curve in green
plot(dens_f, col = "green")

#add the male BMI density curve to the existing plot, default colour (black)
lines(dens_m)

#create an index vector with integers from 1 to 200
```

```r
#use for subsetting the first 200 observations of 'bmi_data'
idx  <- 1:200

#perform Principal component analysis (PCA) on a subset of columns of 'bmi_data'
#uses only rows specified by idx i.e. first 200 observations
#removes column 1 (idx) and 9 (sex) to keep numeric variables
#center = TRUE: subtract mean from each variable to ensure a mean-centered data
#scale = TRUE: scale variables to unit variance,
#which preventing larger units from dominating
pca_res <- prcomp(bmi_data[idx, -c(1, 9)], center = TRUE, scale = TRUE)

#scatter plot with scores of first two principal components (PC1 vs PC2)
#pca_res$x contains PCA scores for each observation
#points are solid circle (pch = 19) in blue colour (col = "blue")
plot(pca_res$x[, 1], pca_res$x[, 2], col = "blue", pch = 19)

#fit a simple linear regression model on 'bmi' and 'age'
#'bmi' acts as response variable, 'age' as explanatory variable
#use only first 200 observations (idx)
out <- lm(bmi[idx] ~ age[idx])

#summary of the fitted linear model
#shows coefficient estimates, p-values, and model diagonistics
summary(out)

#extract the p-value of the age coefficient from model summary
#summary(out)$coefficients is a matrix:
#rows = (Intercept), age; cols = Estimate, Std. Error, t value, Pr(>|t|)
#[2, 4] selects p-value in row 2 (age), column 4 (Pr(>|t|))
p_val <- summary(out)$coefficients[2, 4]
```

Mistakes/Assumptations and how to fix them:

1. **Undefined 'bmi_data' after load()**

- Code uses the object `bmi_data`, which is never explicitly created.
- It ***assumes*** that `load("data/assess_data_1225.Rdata")` creates an object called `bmi_data`, meaning *assess_data_1225.Rdata* contains a dataframe named `bmi_data`, which is not guranteed.
- **Fix:** Check the name of the object loaded from Rdata using `ls()`, and assign correct dataframe name.

2. **Assumption about 'sex' labels ("male"/"female")**

- The code ***assumes*** that `bmi_data$sex` is coded using exact character strings *'male'* and *'female'* while calculating *dens_m, dens_f*.
- If the labels differ (e.g. **'M'**, **'F'**), the subsetting would fail or return empty vectors, which will cause error in *density()*.
- **Fix:** Before calling *density()*, `table(bmi_data$sex)` should be used to check the coding of `sex` variable.

3. **Hard-coded row indexing (idx <- 1:200)**

- The code ***assumes*** `bmi_data` has **atleast 200 rows**; otherwise *idx <- 1:200* will result in an error.
- If `bmi_data` has more than 200 rows, then only the first 200 are selected, which might cause data loss in subsequent steps.
- **Fix:** Index should be **dynamically defined** based on the number of rows using `idx <- seq_len(nrow(bmi_data))`.

4. **Column position assumption for PCA [-c(1, 9)]**

- The code ***assumes*** that:
  - There is a 9th column, and
  - all remaining columns are numeric.
- PCA requires only numeric values; if there are any *non-numeric columns*, `prcomp()` will fail.
- **Fix:** All the column names should be checked using `colnames(bmi_data)`, before calling `prcomp()`, to ensure correct application of *column indices to exclude*.

5. **Incorrect variavle referencing in regression model (lm(bmi[idx] ~ age[idx])**

- The code ***incorrectly*** references `bmi` and `age` as standalone objects in `lm(bmi[idx] ~ age[idx])`, though `bmi` and `age` are never defined as separate vectors.
- The datas are in `bmi_data` dataset in `bmi_data$bmi` and `bmi_data$age`.
- This will throw **Error in eval(predvars, data, env) : object 'bmi' not found**.
- **Fix:** `bmi` and `age` needs to be referenced through the `bmi_data` using `bmi_data$bmi` and `bmi_data$age`

Poor Presentation Choices and how to fix them:

6. **No legend/weak plot labelling**

- No legend specified for **BMI density** plot making it hard to interpret, which sex is represented by which curve.
- Also plot title and x-labels are not comprehensible. Though **not a programming error** it reduces clarity.
- **Fix:** Add proper labels and title and a legend. Also use separate colour for each *sex* and mention it.

7. **No title/axis label in PCA plot**

- No proper labels for **x** and **y** axes in the PCA plot. Also no title provided.
- Though **not a programming error** it reduces clarity.
- **Fix:** Add proper labels (PC1 and PC2) and title and a legend.

The corrected code is presented below:

```
#load the 'assess_data_1225' Rdata file from data directory in current R session
load("data/assess_data_1225.Rdata")

#all the objects available in the workspace
ls()
```

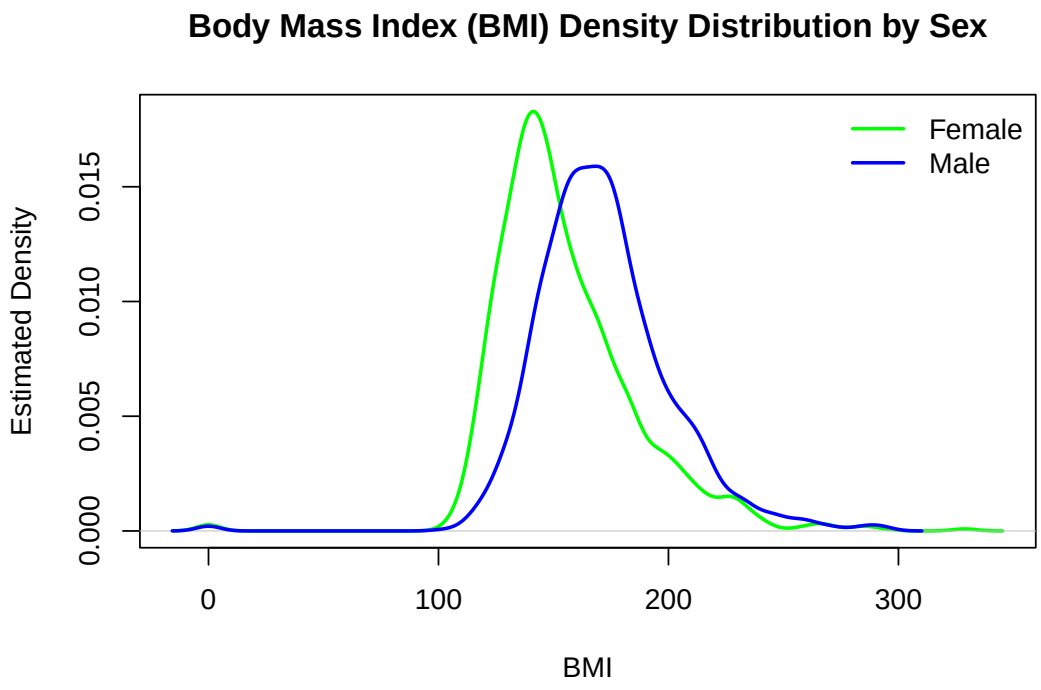
```
##  [1] "bin_info"                  "bmi_data"
##  [3] "bw"                        "climb_up_amount"
##  [5] "default_run"               "df"
##  [7] "hist_info"                 "max_y"
##  [9] "mc_sims"                   "mean_df"
## [11] "n_simulations"             "n_turns"
## [13] "play_mc"                   "play_mc_full"
## [15] "pos_combined"              "pos_df"
## [17] "pos_df_new"                "res_100"
## [19] "res_200"                   "res_300"
## [21] "res_400"                   "sim_10_results"
## [23] "sim_10_results_mean"       "sim_10_results_mean_new"
```

```
## [25] "sim_10_results_new"          "sim_10_results_variance"
## [27] "sim_10_results_variance_new" "sim_results"
## [29] "sim_results_mean"            "simulate_climbs"
## [31] "simulate_pos_at"             "slip_back_range"
## [33] "turns_df"                    "update_bmi"
```

```
#expected object assignment to correct variable
bmi_data <- bmi_data

#frequency table of sex variable
table(bmi_data$sex)
```

```
##
## female   male
##   1726   1474
```

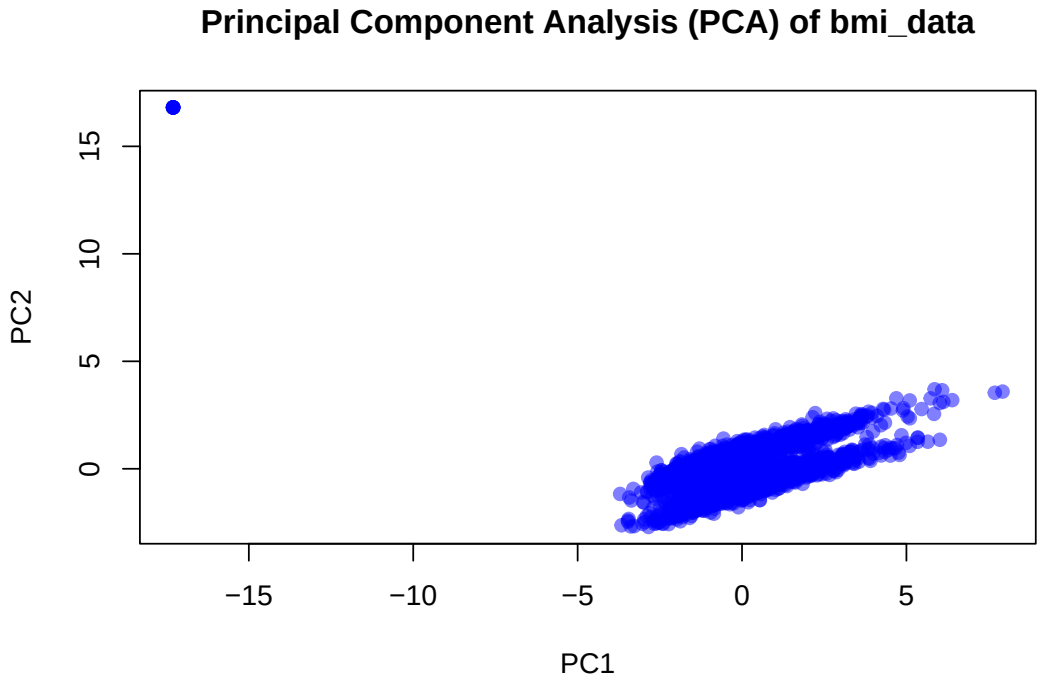
```
#compute kernel density estimate of BMI values for males only
#remove na values
dens_m <- density(bmi_data$bmi[bmi_data$sex == "male"], na.rm = TRUE)
#compute kernel density estimate of BMI values for females only
#remove na values
dens_f <- density(bmi_data$bmi[bmi_data$sex == "female"], na.rm = TRUE)


#plot the female BMI density curve in green
plot(dens_f,
     col = "green",                                #green curve
     lwd = 2,                                      #line-width 2
     main = "Body Mass Index (BMI) Density Distribution by Sex",
                                                   #title
     xlab = "BMI",                                 #x label
     ylab = "Estimated Density")                   #y label

#add the male BMI density curve to the existing
lines(dens_m, col = "blue", lwd = 2)                        #blue curve, line-width 2
#adding the legend
legend("topright",                                 #legend position
       legend = c("Female", "Male"),               #legend by sex
       col = c("green", "blue"),                   #legend colours
       lwd = 2,                                    #line-width 2
       bty = "n")                                  #no legend box
```




```
#create dynamic index based on data size
idx <- seq_len(nrow(bmi_data))

#column names of the dataset
colnames(bmi_data)
```

```
## [1] "idx"      "age"      "bai"      "bmi"      "body_fat" "density"  "weight"
## [8] "height"   "sex"
```

```
#PCA using numeric variables only
pca_res <- prcomp(
  bmi_data[idx, -c(1, 9)],          #all observation selected except non-numeric
                                    #and unnecessary columns (idx and sex)
  center = TRUE,                    #ensure a mean-centered data
  scale = TRUE                      #scale variables to unit variance
)

#scatter plot with scores of first two principal components (PC1 vs PC2)
plot(pca_res$x[, 1],                #PC1 scores
     pca_res$x[, 2],                #PC2 scores
     col = adjustcolor("blue", alpha.f = 0.5),
                            #semi-transparent blue points to reduce overplotting

     pch = 19,                      #solid circles
     xlab = "PC1",                  #x label
     ylab = "PC2",                  #y label
     main = "Principal Component Analysis (PCA) of bmi_data")  #title
```




```r
#fit a simple linear regression model on 'bmi' and 'age'
#corrected referencing
out <- lm(bmi_data$bmi[idx] ~ bmi_data$age[idx])

#summary of the fitted linear model
summary(out)
```

```
##
## Call:
## lm(formula = bmi_data$bmi[idx] ~ bmi_data$age[idx])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -127.229  -19.660   -4.056   15.105  176.786
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)       127.22865    1.92306   66.16   <2e-16 ***
## bmi_data$age[idx]   0.90797    0.04739   19.16   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 29.44 on 3198 degrees of freedom
## Multiple R-squared:  0.103,  Adjusted R-squared:  0.1027
## F-statistic:   367 on 1 and 3198 DF,  p-value: < 2.2e-16
```

```r
#extract the p-value of the age coefficient from model summary
p_val <- summary(out)$coefficients[2, 4]
```

2.1 Data Exploration

Q6 Quality Control

Using the data provided, perform a quality control check of the dataset. Verify that the bmi values are reasonable by comparing the mean BMI for male and female patients, which should each fall between 20 and 30. If you detect any issues (for example, implausible BMI values or incorrect calculations), describe the problem, correct it, and confirm that the corrected data now fall within the expected range. Support your analysis with at least one plot (e.g. histogram or boxplot) and provide a brief summary table of the cleaned data used for further analysis.

Next, perform a Principal Component Analysis (PCA) using the cleaned numeric variables to assess overall data consistency and identify potential outliers. Produce a PCA plot (PC1 vs PC2) to visualise any unusual samples, describe your approach to detecting and removing outliers, and briefly comment on how this step improves the quality of the dataset for subsequent analysis.

Create a function to remove outliers based on your PCA results as well any recalculations for BMI and apply it to the dataset.

- Function name: `update_bmi`
- Arguments: `data` (data frame)
- Returns: `data` (data frame) cleaned data frame without outliers
  [5 MARKS]

Answer 6

1. Loading Data:
The dataset is loaded into the current R session from `assess_data_1225.Rdata` to inspect for all the potential data types, structure of variables and dataset, and look for completeness.

```
#load the 'assess_data_1225' Rdata file from data directory in current R session
load("data/assess_data_1225.Rdata")

#all the objects available in the workspace
ls()
```

```
##  [1] "bin_info"                    "bmi_data"
##  [3] "bw"                          "climb_up_amount"
##  [5] "default_run"                 "dens_f"
##  [7] "dens_m"                      "df"
##  [9] "hist_info"                   "idx"
## [11] "max_y"                       "mc_sims"
## [13] "mean_df"                     "n_simulations"
## [15] "n_turns"                     "out"
## [17] "p_val"                       "pca_res"
## [19] "play_mc"                     "play_mc_full"
## [21] "pos_combined"                "pos_df"
## [23] "pos_df_new"                  "res_100"
## [25] "res_200"                     "res_300"
## [27] "res_400"                     "sim_10_results"
## [29] "sim_10_results_mean"         "sim_10_results_mean_new"
## [31] "sim_10_results_new"          "sim_10_results_variance"
## [33] "sim_10_results_variance_new" "sim_results"
## [35] "sim_results_mean"            "simulate_climbs"
## [37] "simulate_pos_at"             "slip_back_range"
## [39] "turns_df"                    "update_bmi"
```

```
#expected object assignment to correct variable
bmi_data <- bmi_data
```

```r
#structure of dataset
str(bmi_data)      #dataset internal structure
```

```
## tibble [3,200 x 9] (S3: tbl_df/tbl/data.frame)
##  $ idx     : num [1:3200] 2611 2656 2452 1264 730 ...
##  $ age     : num [1:3200] 23 21 24 43 43 43 23 28 41 25 ...
##  $ bai     : num [1:3200] 17.3 17.9 18.5 18.4 22.1 ...
##  $ bmi     : num [1:3200] 102 111 113 118 104 ...
##  $ body_fat: num [1:3200] 6 6.6 5 23.5 21.9 21.9 9.4 5.4 18.4 12.8 ...
##  $ density : num [1:3200] 1.09 1.08 1.09 1.05 1.06 ...
##  $ weight  : num [1:3200] 45.8 50.9 48.1 52.1 43.6 ...
##  $ height  : num [1:3200] 170 172 166 169 164 ...
##  $ sex     : chr [1:3200] "male" "male" "male" "male" ...
```

```r
summary(bmi_data)  #dataset summary statistics
```

```
##       idx              age             bai             bmi
##  Min.   :   1.0   Min.   : 0.00   Min.   : 0.00   Min.   :  0.0
##  1st Qu.: 800.8   1st Qu.:30.00   1st Qu.:25.22   1st Qu.:141.5
##  Median :1600.5   Median :39.00   Median :27.87   Median :158.6
##  Mean   :1600.5   Mean   :39.06   Mean   :28.57   Mean   :162.7
##  3rd Qu.:2400.2   3rd Qu.:47.00   3rd Qu.:31.38   3rd Qu.:179.1
##  Max.   :3200.0   Max.   :69.00   Max.   :65.88   Max.   :329.9
##     body_fat        density          weight          height
##  Min.   : 0.00   Min.   :0.000   Min.   :  0.00   Min.   :  0.0
##  1st Qu.:22.50   1st Qu.:1.032   1st Qu.: 57.45   1st Qu.:159.8
##  Median :28.00   Median :1.043   Median : 67.07   Median :162.8
##  Mean   :27.86   Mean   :1.040   Mean   : 68.58   Mean   :164.1
##  3rd Qu.:33.40   3rd Qu.:1.054   3rd Qu.: 77.71   3rd Qu.:169.8
##  Max.   :58.20   Max.   :1.092   Max.   :133.92   Max.   :176.2
##      sex
##  Length:3200
##  Class :character
##  Mode  :character
##
##
##
```

This showed the presence of all the 9 variables mentioned in the question, which includes both numeric and categorical data, along with giving a identification of the data types. Also it gave an indication of ***implausible BMI values***, which are ***mostly above 100***, whereas ***biologically 100 is the maximum*** possible BMI.

2. BMI (Body Mass Index) Value Verification:
To ensure and verify that the BMI values are reasonable and plausible, the **mean BMI** is calculated for male and female patients separately. As the question mentioned and as per the accepted guidelines, each of the mean, i.e. the ***average BMI values should fall between 20 and 30***.

```r
#mean BMI calculation for male and female separately
#using aggregate() function, which compute a group summary for numeric variables

mean_bmi_sex <- aggregate(
  bmi ~ sex,                  #response variable: bmi; group by:sex
  data = bmi_data,           #use bmi_data frame
  mean,                      #apply mean function
  na.rm = TRUE               #remove na values from bmi col
)

#show mean BMI values
mean_bmi_sex
```

```
##      sex      bmi
## 1 female 155.1213
## 2   male 171.5653
```

Detected Issues:

Upon examination, it becomes evident that the bmi values for both the genders are ***significantly higher*** significantly higher than the anticipated ***biological range of (20 to 30)***, with results showing mean of **171.5653 for male** and **155.1213 for female**, with some values exceeding 300, as evident from the box-plot (Fig. 6). These figures are **not biologically realistic** and point to a serious **issue with data quality**. This strongly implies that **BMI calculations have been performed incorrectly**, likely due to a **discrepancy in measurement units (for instance, height measured in centimeters instead of meters)**, rather than indicating an actual physiological condition.

The error in data entry may be considered as the most common reason, including **errors in decimal placement**, **conversion mistakes between units**, typographical errors in single-digit figures, and issues arising from transcription. Alternatively, there may be problems related to data quality and validation, such as **incomplete data context (for instance, if height and weight were recorded on different days, or if one value is reported by the individual while the other is measured clinically)**, as well as a lack of plausibility checks (the system may fail to identify "biologically implausible values"), which permits inaccurate data to be included in the analysis.

The observation is also backed by the box-plot for the bmi values for both male and female as given below.

The following code chunk is for generating Fig. 6:

```
#using ggplot for the boxplot of bmi from bmi_data
ggplot(data = bmi_data,                #using bmi_data dataframe
       aes(x = sex,
           y = bmi,
           fill = sex)) +              #fill based on sex
  geom_boxplot(alpha = 0.7,                #semi-transparent outliers
               outlier.shape = 1,          #hollow point
               outlier.colour = NULL,
               outlier.size = 3) +

  #plotting the mean bmi points for each sex
  geom_point(data = mean_bmi_sex,      #use the mean calculated from aggregate()
             aes(x = sex,
                 y = bmi),
             color = "black",              #black border
             fill = "#D62728",             #fill red
             size = 3,
             shape = 21,                   #fill point with border
             show.legend = FALSE) +        #don't add point to legend

  #mean labels
  geom_text(data = mean_bmi_sex,
             aes(x = sex,
                y = bmi,
                label = paste("Mean BMI = ", round(bmi, 2))), #round to 2 places
             color = "#D62728",
             fontface = "bold",
             vjust = -2,                   #vertical position
             size = 4) +

  #manually select the colours for the fill
  scale_fill_manual(values = c("female" = "pink",
                               "male" = "lightblue")) +
  #y-ticks every 50
  scale_y_continuous(
    breaks = seq(0, max(bmi_data$bmi, na.rm = TRUE), by = 50)) +
```

```r
#plot appearance
theme(
  plot.title  = element_text(size = 10, face = "bold", hjust = 0.5),
  axis.title.x = element_text(size = 11),
  axis.title.y = element_text(size = 11),
  axis.text.x  = element_text(size = 10),
  axis.text.y  = element_text(size = 10)
) +

#title and axis labels
labs(
  title = "Fig. 6: BMI (Body Mass Index) Distribution by Sex",
  x     = "Sex",
  y     = "BMI"
)
```

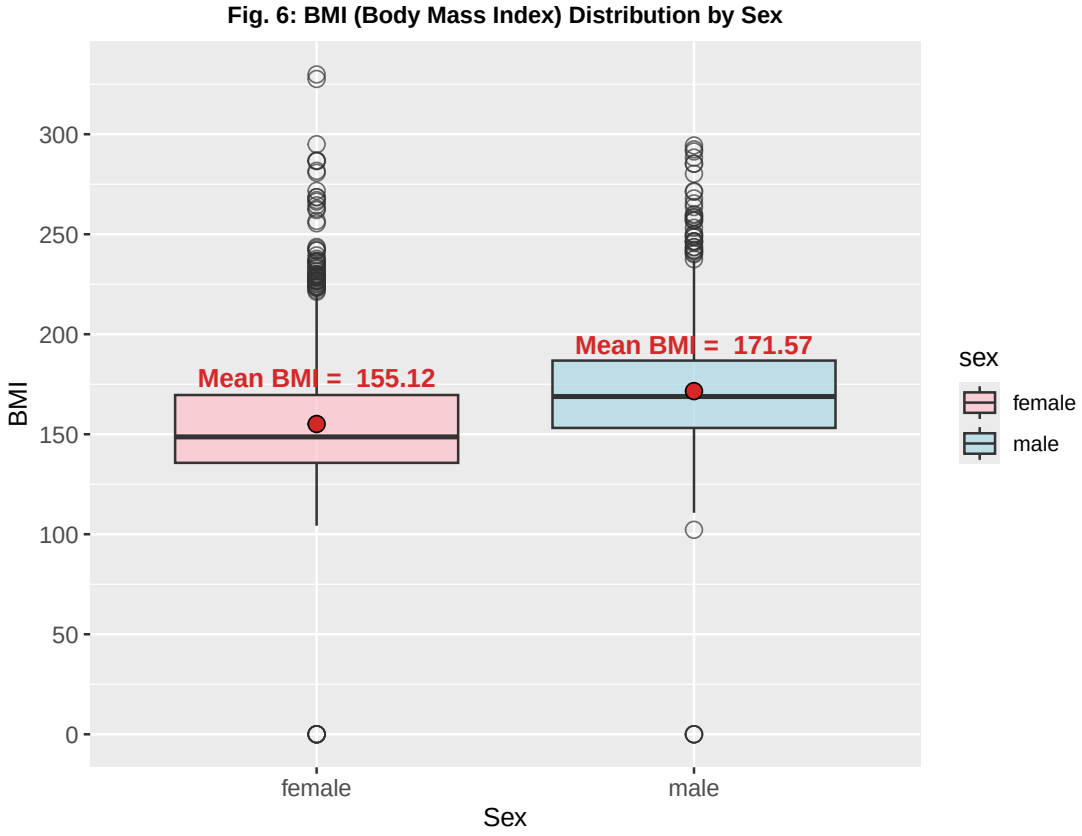


Fig. 6 presents boxplots of the original BMI values categorized by sex. The ***x and y axes*** indicates sex and BMI values, respectively. Both females and males exhibit distributions centered around **excessively high figures (approximately 150 - 170)**, with a significant number of **extreme outliers reaching above 300**, which is by far beyond the reasonable ***biological range of (20 to 30)***. The presence of these elevated central tendencies and extreme outliers strongly suggests a systematic error in the BMI data rather than authentic biological variation.

Crucially, both sexes demonstrate **similarly unrealistic BMI distributions**, indicating that the problem is **not associated with sex-specific factors** but rather **stems from an incorrect calculation or unit mismatch** that impacts the entire dataset. This reinforces that the **original `bmi` variable is not reliable** and *inappropriate* for subsequent statistical analysis.

3. Correction of BMI:

As a result of the incorrect BMI values, before further analysis, the BMI values are recalculated using the following formula:

$$\text{BMI} = \frac{\textbf{weight (in kg)}}{\textbf{height}^2\textbf{(in m)}}$$

A duplicate of the **bmi_data** dataset is made and saved in a separate data frame **bmi_data_modified** to guarantee that any changes applied to the copied dataset won't impact the original data. Also it is observed that the heights are **recorded in centimetres instead of meters**, so **they are converted to meters** before calculating the BMIs. The corrected BMIs are stored in a new column named **bmi_correct** in the **bmi_data_modified** dataframe.

```
#copy of bmi_data
bmi_data_modified <- as.data.frame(bmi_data)

#recalculating BMI and storing corrected value in a new column bmi_correct
#weight in kg and height in cm
#height converted to m for consistency with the standard BMI formula
bmi_data_modified$bmi_correct <-
  bmi_data_modified$weight / ((bmi_data_modified$height / 100)^2)
```

After correctly calculating the BMI with weight and height, using appropriate units, the **mean bmi** for both male and female are again recalculated.

Calculating the mean BMI by Sex using Corrected Values:

```
#mean BMI calculation for male and female separately
#using aggregate() function, which compute a group summary for numeric variables

mean_bmi_sex_correct <- aggregate(
  bmi_correct ~ sex,            #response variable: bmi_correct; group by:sex
  data = bmi_data_modified,     #use bmi_data_modofoed frame
  mean,                         #apply mean function
  na.rm = TRUE                  #remove na values from bmi col
)

#show mean BMI values
mean_bmi_sex_correct
```

```
##      sex bmi_correct
## 1 female    24.12762
## 2   male    26.66493
```

After correctly calculating the BMI with weight and height, using appropriate units, the **mean bmi** for both male and female are again recalculated. This time the average BMI values for both genders are within reasonable ***biological range of (20 to 30)***, with results showing mean of **26.66493 for male** and **24.12762 for female**. This indicates that the initial data had a calculation mistake in the BMI, **likely due to inconsistencies in units**, ***(as correcting the unit for height fixed the issue)***, and that the implemented correction effectively addresses the problem. Thus, the revised dataset offers a solid foundation for additional analysis.

The following code chunk is for generating Fig. 7:

```
#using ggplot for the boxplot of bmi_correct from bmi_data_modofied
ggplot(data = bmi_data_modified,          #using bmi_data_modified dataframe
       aes(x = sex,
           y = bmi_correct,
           fill = sex)) +                 #fill based on sex
  geom_boxplot(alpha = 0.7,               #semi-transparent outliers
               outlier.shape = 1,         #hollow point
               outlier.colour = NULL,
               outlier.size = 3) +
```

```r
#plotting the mean bmi points for each sex
geom_point(data = mean_bmi_sex_correct, #use the mean from aggregate()
           aes(x = sex,
               y = bmi_correct),
           color = "black",             #black border
           fill = "#D62728",            #fill red
           size = 3,
           shape = 21,                  #fill point with border
           show.legend = FALSE) +       #don't add point to legend

#mean labels
geom_text(data = mean_bmi_sex_correct,
           aes(x = sex,
              y = bmi_correct,
              label = paste("Mean BMI = ", round(bmi_correct, 2))),
           color = "#D62728",
           fontface = "bold",
           vjust = -3,                  #vertical position
           size = 4) +

#manually select the colours for the fill
scale_fill_manual(values = c("female" = "pink",
                             "male" = "lightblue")) +
#y-ticks every 5
scale_y_continuous(
  breaks = seq(0, max(bmi_data_modified$bmi_correct, na.rm = TRUE), by = 5))+

#plot appearance
theme(
  plot.title  = element_text(size = 10, face = "bold", hjust = 0.5),
  axis.title.x = element_text(size = 11),
  axis.title.y = element_text(size = 11),
  axis.text.x  = element_text(size = 10),
  axis.text.y  = element_text(size = 10),
) +

#title and axis labels
labs(
  title = "Fig. 7: BMI (Body Mass Index) Distribution by Sex (Corrected)",
  x     = "Sex",
  y     = "BMI"
)
```

Fig. 7: BMI (Body Mass Index) Distribution by Sex (Corrected)

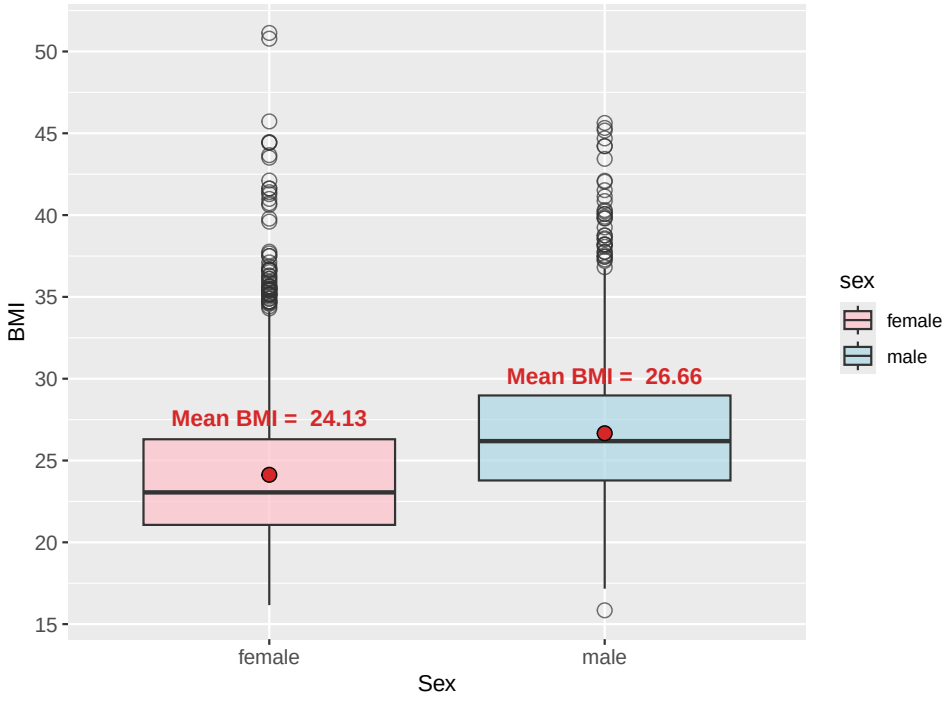


Fig. 7 illustrates the distributions of BMI after it has been recalculated using the weight and height, after converting the height ***from centimeters to meters*** by dividing the height by 100. Unlike in Fig. 6, the recalibrated BMI values now fall within reasonable ***biological range of (20 to 30)***. Female BMI values are predominantly in the **low to mid-20s (with a recalculated mean of approximately 24.1)**, whereas male BMI values are ***slightly elevated***, primarily in the **mid-20s (with a recalculated mean of around 26.7)**.

While some BMI values are *still elevated (50 or greater)*, they now correspond to a ***feasible overweight or obese classification*** rather than being extreme numerical anomalies.

3. Comparative Visualization of BMI:

The original and the corrected BMIs are plotted side by side to get a comparative overview how the recalculation affected the data. The plot is shown in Fig. 8.

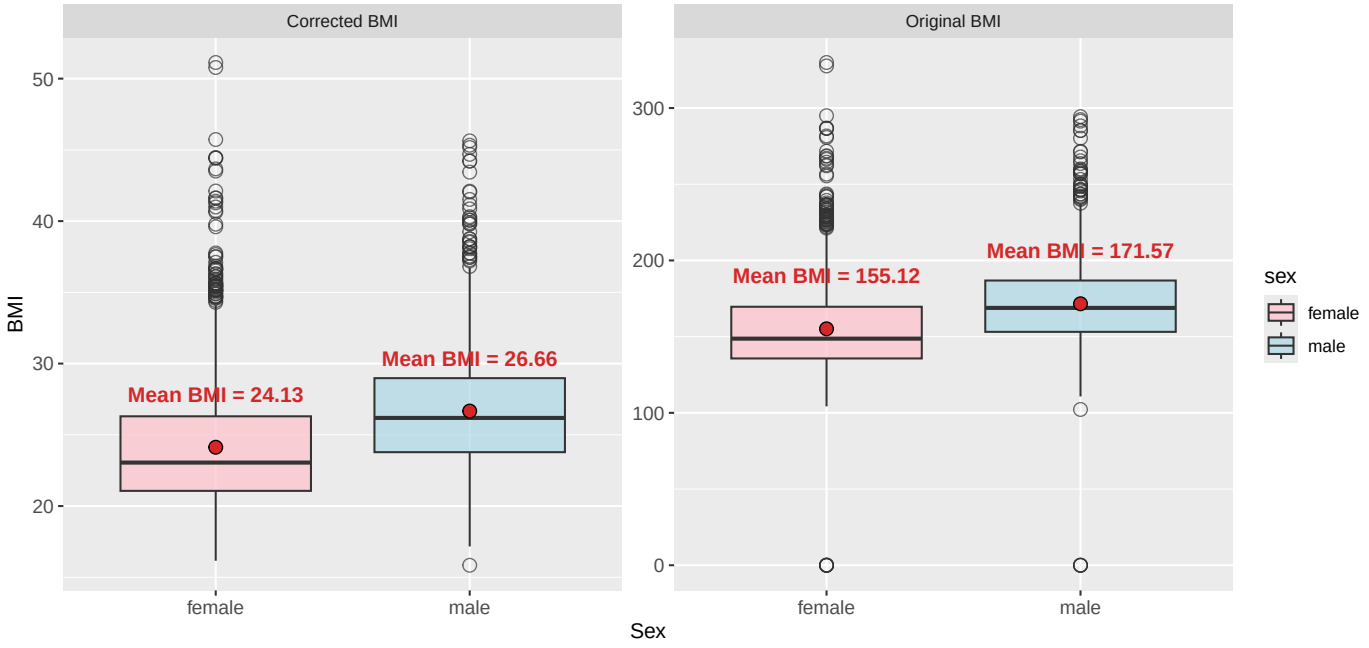
The following code chunk is for generating Fig. 8:

```
#comparison plot before vs after
#combine raw values of bmi
bmi_long <- bind_rows(
  bmi_data %>%
    transmute(
      sex,                        #keep sex column
      bmi = bmi,                  #rename bmi column
      dataset = "Original BMI"    #tag the dataset
    ),
  bmi_data_modified %>%
    transmute(
      sex,                        #keep sex column
      bmi = bmi_correct,          #rename bmi_correct to bmi
      dataset = "Corrected BMI"   #tag the dataset
    )
)

#combine the means
mean_bmi_long <- bind_rows(
```

```
  mean_bmi_sex %>%
    transmute(
      sex,                        #keep sex column
      bmi = bmi,                  #rename bmi column
      dataset = "Original BMI"    #tag the dataset
    ),
  mean_bmi_sex_correct %>%
    transmute(
      sex,                        #keep sex column
      bmi = bmi_correct,          #rename bmi_correct column to bmi
      dataset = "Corrected BMI"   #tag the dataset
    )
)

#facet plot
ggplot(bmi_long,
       aes(x = sex,
           y = bmi,
           fill = sex)) +     #colour by sex

  geom_boxplot(
    alpha          = 0.7,
    outlier.shape  = 1,       #hollow points
    outlier.colour = NULL,
    outlier.size   = 3
  ) +

  #mean points
  geom_point(
    data = mean_bmi_long,
    aes(x = sex,
        y = bmi),
    color = "black",          #black border
    fill = "#D62728",         #red point
    size = 3,
    shape = 21,               #solid point with border
    show.legend = FALSE       #don't add to legend
  ) +

  #mean labels
  geom_text(
    data = mean_bmi_long,
    aes(x = sex,
        y = bmi,
        label = paste("Mean BMI =", round(bmi, 2))),
    color = "#D62728",
    fontface = "bold",
    vjust = -3,            #vertical position
    size = 4) +

  #manually select the colours for the fill
  scale_fill_manual(values = c("female" = "pink",
                               "male" = "lightblue")) +

  #facet: Corrected vs Original
  facet_wrap(~ dataset, scales = "free_y") +      #separate scales for y-axis

  #plot appearance
  theme(
    plot.title  = element_text(size = 12, face = "bold", hjust = 0.5),
```

```
    axis.title.x = element_text(size = 11),
    axis.title.y = element_text(size = 11),
    axis.text.x  = element_text(size = 10),
    axis.text.y  = element_text(size = 10),
  ) +

  #title and axis labels
  labs(
    title = "Fig. 8: BMI Distribution by Sex: Original vs Corrected",
    x     = "Sex",
    y     = "BMI"
  )
```

Fig. 8: BMI Distribution by Sex: Original vs Corrected



The original BMI values were clearly incorrect with, impossibly high means for both the sexes ***(171.57 for male and 155.12 for female)***, though after correction the BMI distribution fall within realistic **biological range of (20 to 30)** ***(26.66 for male and 24.13 for female)***, confirming the error was resolved, as evident from the above figure.

The **black line** within each box denotes the **central trends of the BMI value distributions**. The impact of adjusting the BMI calculation is highlighted by the significant change in the central tendency between ***Corrected BMI*** and ***Original BMI*** plot.

4. Summary Table for Cleaned Data:

Observations with only plausible values are retained for the further analysis. Also a new data frame `bmi_data_clean` is created with the correct `bmi` values and after removing the non-numeric values for the downstream PCA.

```
#column order in the dataset
head(bmi_data_modified)
```

```
##    idx age     bai      bmi body_fat  density   weight   height    sex
## 1 2611  23 17.3024 102.2215      6.0 1.086789 45.77181 169.9662   male
## 2 2656  21 17.9103 110.7454      6.6 1.084295 50.94857 172.2812   male
## 3 2452  24 18.5207 113.2622      5.0 1.086815 48.13971 165.5937   male
## 4 1264  43 18.4041 118.3022     23.5 1.052070 52.13709 168.6210   male
## 5  730  43 22.1045 104.2686     21.9 1.055003 43.58478 164.2197 female
## 6 3129  43 22.1045 104.2686     21.9 1.055182 42.11419 161.4255 female
##   bmi_correct
## 1     15.8443
```

```
## 2     17.1655
## 3     17.5556
## 4     18.3368
## 5     16.1616
## 6     16.1616
```

```r
#creating a new dataframe for downstream PCA with cleaned data
bmi_data_clean <- bmi_data_modified %>%
             select(-idx, -bmi) %>%   #remove the old bmi, idx, sex column
             rename(bmi = bmi_correct) %>%  #rename bmi_correct to bmi
             relocate(bmi, .after = 2)  #move the new bmi col to original place

#summary of the clean data
summary(bmi_data_clean)
```

```
##       age             bai             bmi           body_fat
##  Min.   : 0.00   Min.   : 0.00   Min.   :15.84   Min.   : 0.00
##  1st Qu.:30.00   1st Qu.:25.22   1st Qu.:21.97   1st Qu.:22.50
##  Median :39.00   Median :27.87   Median :24.61   Median :28.00
##  Mean   :39.06   Mean   :28.57   Mean   :25.30   Mean   :27.86
##  3rd Qu.:47.00   3rd Qu.:31.38   3rd Qu.:27.76   3rd Qu.:33.40
##  Max.   :69.00   Max.   :65.88   Max.   :51.13   Max.   :58.20
##                                  NA's   :10
##     density          weight          height          sex
##  Min.   :0.000   Min.   :  0.00   Min.   :  0.0   Length:3200
##  1st Qu.:1.032   1st Qu.: 57.45   1st Qu.:159.8   Class :character
##  Median :1.043   Median : 67.07   Median :162.8   Mode  :character
##  Mean   :1.040   Mean   : 68.58   Mean   :164.1
##  3rd Qu.:1.054   3rd Qu.: 77.71   3rd Qu.:169.8
##  Max.   :1.092   Max.   :133.92   Max.   :176.2
##
```

Following the cleaning process, the average BMI for both genders was within the **anticipated range (20–30)**, meeting the quality control requirements.

5. Principal Component Analysis (PCA):

The PCA is performed on the `bmi_data_clean` dataset.

```r
#remove rows with missing, inf, or NA values and keep numeric only
bmi_data_clean_new <- bmi_data_clean %>%
                      select(-sex)    #removing sex
bmi_data_clean_numeric <- bmi_data_clean_new[complete.cases(bmi_data_clean_new), ]

#performing PCA
pca_res <- prcomp(bmi_data_clean_numeric,
                  center = TRUE,
                  scale = TRUE)

#summary of the pca
pca_summary <- summary(pca_res)
print(pca_summary)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4    PC5     PC6     PC7
## Standard deviation     1.9179 1.3997 0.9319 0.58132 0.3907 0.04257 0.03961
## Proportion of Variance 0.5255 0.2799 0.1241 0.04828 0.0218 0.00026 0.00022
## Cumulative Proportion  0.5255 0.8054 0.9294 0.97772 0.9995 0.99978 1.00000
```

6. Variance Explained by Each PC and Outlier Identification:

6.a. Variance Explained by Each PC

The amount of variance attributed to each principal component (PC) was determined from the results of the principal component analysis (PCA). PCA breaks down the overall variance in the dataset into **orthogonal components**, with each component linked to a **standard deviation (sdev)** that indicates the level of variation represented by that component.

The **Variance of PC** and **Proportion of Variance Explained** can be calculated as follows:

$$\textbf{Variance of PC}_i = (\textbf{sdev}_i)^2$$

$$\textbf{Proportion of Variance Explained}_i = \frac{(\textbf{sdev}_i)^2}{\sum_j (\textbf{sdev}_j)^2}$$

The proportions are subsequently visualized with a bar plot, where each bar indicates the portion of total variance represented by a principal component. This facilitates the evaluation of variance distribution among components and aids in identifying the prominent components. In this analysis, the ***sharp drop following the first two components*** suggests that the ***first and second principal components*** account for most of the variability ($\approx 80.54\%$) in the dataset.

The following code chunk is for generating Fig. 9:

```
#variance is the square of the standard deviation
pr_var <- pca_res$sdev^2

#compute the variance explained by each principal component
prop_var_exp <- pr_var / sum(pr_var)

#create a dataframe for plotting
var_exp <- data.frame(variance = prop_var_exp, pc = 1:length(prop_var_exp))

#create the plot
ggplot(var_exp[1:7, ],           #7 PCs only
       aes(x = pc,
           y = variance,
           fill = pc)) +          #colour based on PC value

    geom_bar(stat = "identity") +

    #variance values on top of bars
    geom_text(aes(label = round(variance, 4)),
              vjust = -0.5,        #vertical position
              size = 3) +

    #colouring
    scale_fill_gradient(low = "red4",
                        high = "gold") +

    #plot appearance
    theme(
      plot.title  = element_text(size = 10, face = "bold", hjust = 0.5),
      axis.title.x = element_text(size = 11),
      axis.title.y = element_text(size = 11),
      axis.text.x  = element_text(size = 10),
      axis.text.y  = element_text(size = 10),
      legend.position = "none"          #no legend
    ) +
```

```
  #title and axis labels
  labs(
    title = "Fig. 9: Variance explained by Each PC",
    x     = "Principal Component",
    y     = "Variance explained"
  )
```

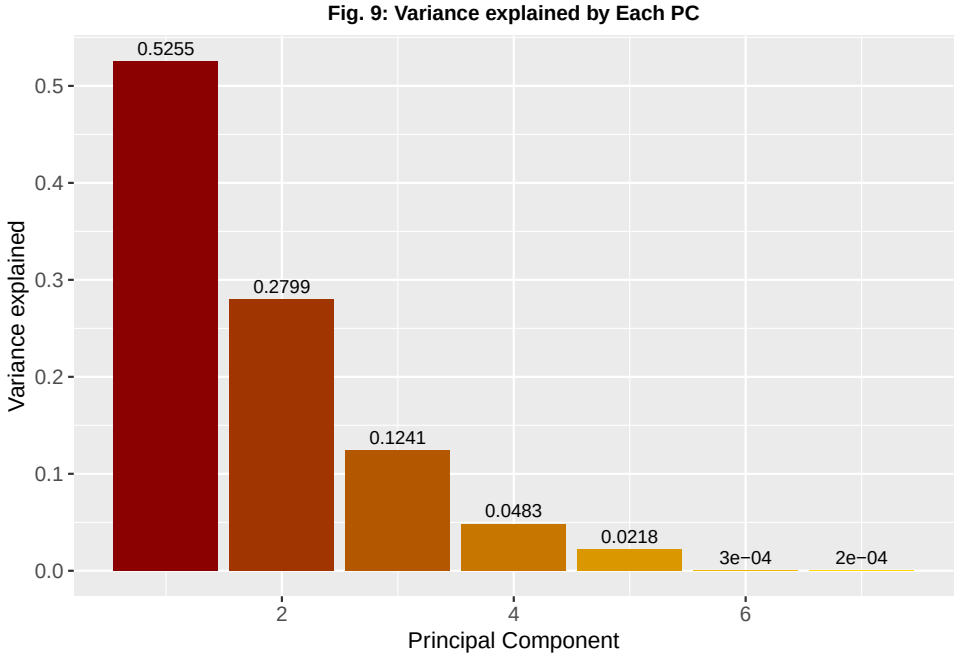


Fig. 9 illustrates the percentage of total variance accounted for by each principal component. The **first principal component (PC1) accounts for roughly 52.5%** of the total variance, while the **second principal component (PC2) accounts for about 28%**. Collectively, ***PCs 1 and 2 together*** represent **about 80.54%** of the total variance, suggesting that a significant portion of the data can be effectively captured in a two-dimensional framework.

The subsequent components account for increasingly smaller percentages of variance, with a **notable decrease occurring after PC2**. This reinforces the decision to concentrate on **PCs 1 and 2** PCs 1 and 2 for visualization and outlier detection, as the ***higher components add relatively minimal additional information***.

6.b. Detection of Outliers

The outliers are detected as follows:

- Outliers are determined by measuring their distance from the center of the PCA score space, utilizing the first two principal components. Observations that exhibit notably ***large distances from the primary clusters*** are flagged for potential removal.
- They are identified by computing the **scaled squared Euclidean distance** of each observation from the **center of the PC1-PC2 space** and flagging those that **surpassed the 97.5th percentile** of this distance distribution.
- The **97.5th percentile** signifies the ***extreme upper section*** of the distance distribution.
- This approach, based on PCA, identifies samples that **diverge from the predominant multivariate pattern**, instead of **depending on individual variable thresholds**.

```
#PC1 and PC2 scores for each observation as dataframe for further plotting
scores <- as.data.frame(pca_res$x[, 1:2])
#changing col names
colnames(scores) <- c("PC1", "PC2")

#distances from the PCA origin for each sample
#scores are scaled before computing squared Euclidean distance for comparability
```

```r
pca_res_dist <- rowSums(scale(scores)^2)
#adding the distances
scores$dist <- pca_res_dist

#defining outliers using the 97.5% quantile distance threshold
outlier_cutoff <- quantile(pca_res_dist, 0.975)

#flagging outliers (logical flag)
scores$outlier <- pca_res_dist > outlier_cutoff

#groundings for outliers
scores$type <- ifelse(scores$outlier, "Outlier", "Non-outlier")
scores$type <- factor(scores$type, levels = c("Non-outlier", "Outlier"))
```

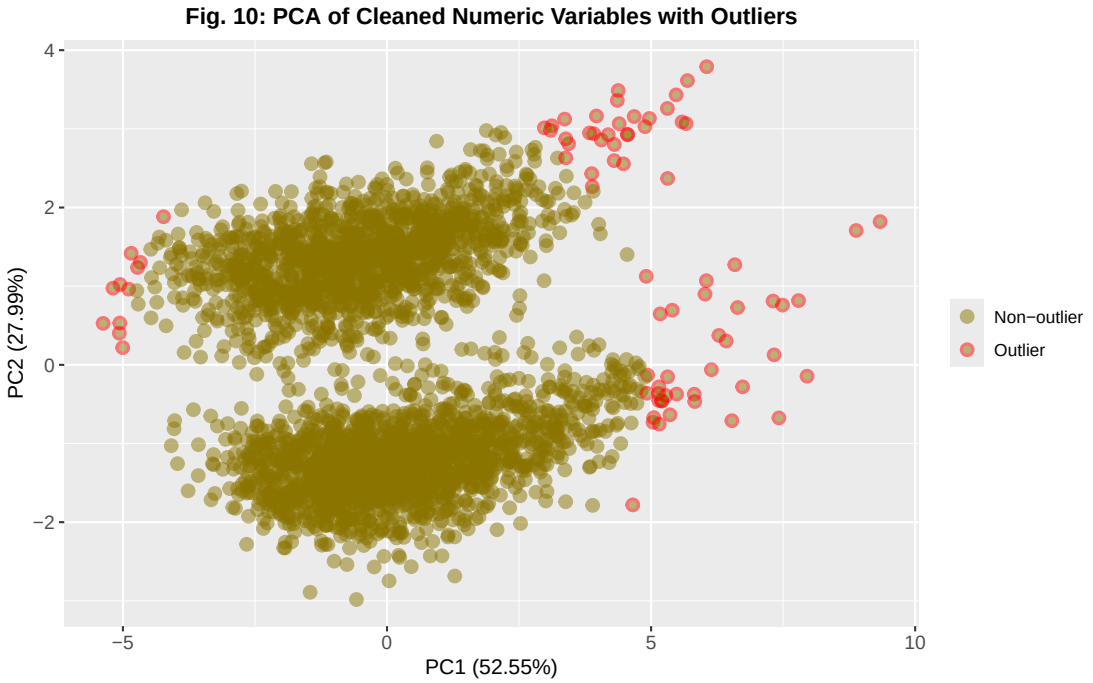
7. PCA Visualization with Outliers:

The initial PCA with the outliers is visualized below, with the outliers marked in **red**.

The following code chunk is for generating Fig. 10:

```r
#variance labels
pc1_var <- round(prop_var_exp[1] * 100, 2)
pc2_var <- round(prop_var_exp[2] * 100, 2)

#ggplot for plotting the PCA
ggplot(scores,
       aes(x = PC1,
           y = PC2)) +

    #plotting the points
    geom_point(aes(colour = type,
                   shape = type,
                   fill = type),
               size = 2,
               stroke = 1.2,
               alpha = 0.5) +

    # Manual legend
    scale_shape_manual(values = c("Non-outlier" = 19,     #solid point
                                  "Outlier" = 21)) +      #point with border

    #fill values
    scale_fill_manual(values = c("Non-outlier" = "gold4",
                                  "Outlier" = "gold4")) +

    #setting borders
    scale_color_manual(values = c("Non-outlier" = "gold4",
                                  "Outlier" = "red")) +
    #remove legend title
    labs(color = NULL, shape = NULL, fill = NULL) +
   #plot appearance
    theme(
      plot.title  = element_text(size = 12, face = "bold", hjust = 0.5),
      axis.title.x = element_text(size = 11),
      axis.title.y = element_text(size = 11),
      axis.text.x  = element_text(size = 10),
      axis.text.y  = element_text(size = 10),
    ) +
```

```r
  #title and axis labels
  labs(
    title = "Fig. 10: PCA of Cleaned Numeric Variables with Outliers",
    x     = paste0("PC1 (", pc1_var, "%)"),
    y     = paste0("PC2 (", pc2_var, "%)")
  )
```



8. PCA Plot Interpretation:

The PCA plot (Fig. 10) displaying the cleaned numeric variables illustrates how the observations are distributed within the space defined by the first two principal components (PC1 and PC2), which account for the main sources of variation in the dataset.

a. **Overall Overview**

- The PCA plot indicates that the majority of observations create **two well-defined and densely populated clusters primarily differentiated along PC2**, with some additional variation visible along PC1.
- This suggests that **PC2** accounts for a **significant source of systematic variation in the data**, while *PC1 provides a secondary, yet still notable, dimension of variability.*
- The tightly-knit character of each cluster implies that most observations exhibit internal consistency post data cleaning.

b. **Identification of Outliers**

- A few **red** points are clearly located away from the main clusters of **gold**, specifically:
  - In areas with **high positive values of PC1**, and
  - Near the **upper limit of PC2**.
- These observations **significantly diverge** from the **overall multivariate framework** of the dataset and are deemed potential outliers.
- Notably, these points are **not extreme in any single variable**, yet they **appear anomalous when multiple variables are considered together**.

9. Removing Outliers:

Outliers identified through PCA are referenced back to their original row indices and removed from the dataset before further analysis creating a new data frame **bmi_data_clean_final**. This method guarantees that later analyses remain **unaffected by multivariate extremes** and that only the samples classified as multivariate outliers are discarded, while other data points are kept.

```
#identify the rows indices of the data points used for PCA
pca_rows <- which(complete.cases(bmi_data_clean))

#lookup table linking row index to outlier/type
outlier_info <- tibble(
  row_id  = pca_rows,
  outlier = scores$outlier,
  type    = scores$type
)

#join back to bmi_data_clean and drop outliers
bmi_data_clean_final <- bmi_data_clean %>%
  mutate(row_id = row_number()) %>%
  left_join(outlier_info, by = "row_id") %>%
                                          #keep rows that are either:
                                          #not in PCA (NA outlier), or
                                          #in PCA but not flagged as outlier
  filter(is.na(outlier) | !outlier) %>%
  select(-row_id, -outlier, -type)
```

The following code chunk is for generating Fig. 11:

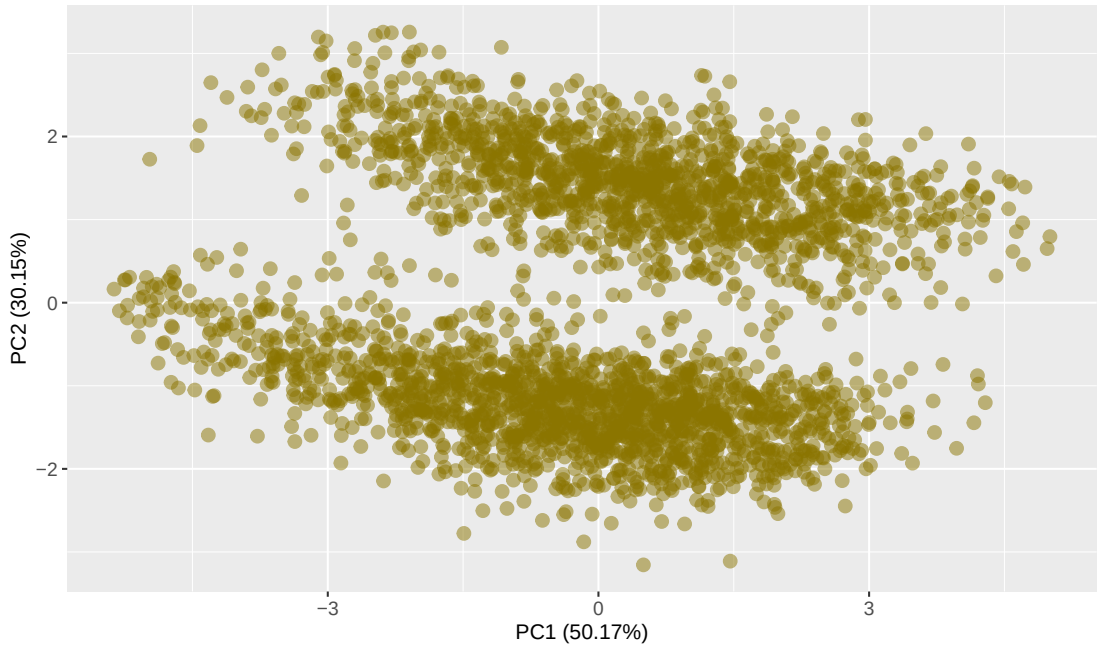
```
#remove rows with missing, inf, or NA values and keep numeric only
bmi_data_clean_final_new <- bmi_data_clean_final %>%
                            select(-sex)
bmi_data_clean_final_numeric <- bmi_data_clean_final_new[complete.cases(bmi_data_clean_final_new), ]

#performing PCA
pca_res_final <- prcomp(bmi_data_clean_final_numeric,
                    center = TRUE,
                    scale = TRUE)

#summary of the pca
pca_summary <- summary(pca_res_final)
print(pca_summary)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6     PC7
## Standard deviation     1.8741 1.4527 0.9160 0.60692 0.40823 0.04450 0.03767
## Proportion of Variance 0.5017 0.3015 0.1199 0.05262 0.02381 0.00028 0.00020
## Cumulative Proportion  0.5017 0.8032 0.9231 0.97571 0.99951 0.99980 1.00000
```

```
#variance is the square of the standard deviation
pr_var <- pca_res_final$sdev^2

#compute the variance explained by each principal component
prop_var_exp <- pr_var / sum(pr_var)

#create a dataframe for plotting
var_exp <- data.frame(variance = prop_var_exp, pc = 1:length(prop_var_exp))
```

```r
#PC1 and PC2 scores for each observation as dataframe for further plotting
scores_final <- as.data.frame(pca_res_final$x[, 1:2])
#changing col names
colnames(scores_final) <- c("PC1", "PC2")

#variance labels
pc1_var <- round(prop_var_exp[1] * 100, 2)
pc2_var <- round(prop_var_exp[2] * 100, 2)

#ggplot for plotting the PCA
ggplot(scores_final,
       aes(x = PC1,
           y = PC2)) +

    #plotting the points
    geom_point(colour = "gold4",
               size = 2,
               stroke = 1.2,
               alpha = 0.5) +

   #plot appearance
    theme(
      plot.title  = element_text(size = 12, face = "bold", hjust = 0.5),
      axis.title.x = element_text(size = 11),
      axis.title.y = element_text(size = 11),
      axis.text.x  = element_text(size = 10),
      axis.text.y  = element_text(size = 10),
    ) +

    #title and axis labels
    labs(
      title = "Fig. 11: PCA of with Outliers Removed",
      x     = paste0("PC1 (", pc1_var, "%)"),
      y     = paste0("PC2 (", pc2_var, "%)")
    )
```

Fig. 11: PCA of with Outliers Removed



Outliers were initially identified using a **97.5th percentile threshold** based on PCA score distances and then excluded

from the dataset. PCA was subsequently recalculated on this refined dataset **bmi_data_clean_final** without any additional outlier detection to evaluate the fundamental data structure after cleaning.

The two previously distinct clusters, primarily influenced by variations in BMI, body fat, and density, are now more clearly defined. Their **boundaries appear smoother, with a reduction in irregularities** caused by extreme values. This suggests that the true structure of the dataset is now more accurately captured by PC1 and PC2.

Outliers can significantly affect PCA loadings and ***alter the orientation of the principal components***. Eliminating them leads to more stable PC directions, clearer interpretations of the relationships between variables, and less distortion in the variance explained.

As a result, the principal components now better reflect true underlying variation instead of noise.

10. Data Quality Impact:

Eliminating outliers identified through PCA enhances the dataset by:

- Diminishing the impact of **extreme or unusual observations** on subsequent analyses, which might have otherwise **skewed regression coefficients** or **inflated model residuals**.
- Boosting statistical reliability, given that linear models are susceptible to extreme leverage points.
- Guaranteeing that later analyses more accurately **reflect the core structure of the data (or trends at the population level)**, rather than ***artifacts caused by a limited number of atypical samples***.

These enhancements support the implementation of PCA-based outlier detection before conducting regression analysis.

11. Function for Updating BMI and Removing Outliers:

To maintain a systematic and reproducible approach for correcting BMI and removing multivariate outliers, a custom R function called **update_bmi()** is written in the **cw_functions.R** Rscript. This function combines ***BMI recalculation, data cleaning, and outlier detection using PCA*** into a single reusable pipeline, as long as the variable names are consistent.

```r
#function sourced from cw_functions.R (Q6)
#Function for Updating BMI and Removing Outliers
update_bmi <- function(data) {
  #copy of data
  data_modified <- as.data.frame(data)

  #recalculating BMI and storing corrected value in a new column bmi_correct
  #weight in kg and height in cm
  #height converted to m for consistency with the standard BMI formula
  data_modified$bmi_correct <-
    data_modified$weight / ((data_modified$height / 100)^2)

  #creating a new dataframe for downstream PCA with cleaned data
  data_clean <- data_modified %>%
    select(-idx, -sex, -bmi) %>%   #remove the old bmi, idx, sex column
    rename(bmi = bmi_correct) %>%  #rename bmi_correct to bmi
    relocate(bmi, .after = 2)  #move the new bmi col to original place

  #remove rows with missing, inf, or NA values and keep numeric only
  data_clean_new <- data_clean %>%
                    select(-sex)
  data_clean_numeric <- data_clean_new[complete.cases(data_clean_new), ]

  #performing PCA
  pca_res <- prcomp(data_clean_numeric,
                    center = TRUE,
                    scale = TRUE)

  #PC1 and PC2 scores for each observation as dataframe for further plotting
  scores <- as.data.frame(pca_res$x[, 1:2])
```

```
  #changing col names
  colnames(scores) <- c("PC1", "PC2")

  #distances from the PCA origin for each sample
  #scores are scaled before computing squared Euclidean distance for comparability
  pca_res_dist <- rowSums(scale(scores)^2)
  #adding the distances
  scores$dist <- pca_res_dist

  #defining outliers using the 97.5% quantile distance threshold
  outlier_cutoff <- quantile(pca_res_dist, 0.975)

  #flagging outliers (logical flag)
  scores$outlier <- pca_res_dist > outlier_cutoff

  #groundings for outliers
  scores$type <- ifelse(scores$outlier, "Outlier", "Non-outlier")
  scores$type <- factor(scores$type, levels = c("Non-outlier", "Outlier"))

  #identify the rows indices of the data points used for PCA
  pca_rows <- which(complete.cases(data_clean))

  #lookup table linking row index to outlier/type
  outlier_info <- tibble(
    row_id  = pca_rows,
    outlier = scores$outlier,
    type    = scores$type
  )

  #join back to data_clean and drop outliers
  data_clean_final <- data_clean %>%
    mutate(row_id = row_number()) %>%
    left_join(outlier_info, by = "row_id") %>%
                                          #keep rows that are either:
                                          #not in PCA (NA outlier), or
                                          #in PCA but not flagged as outlier
    filter(is.na(outlier) | !outlier) %>%
    select(-row_id, -outlier, -type)

  #reverting to the original data frame
  data <- data_clean_final

  return(data)
}
```

Testing the **update_bmi()** function:

```
#calling PCA-based Outlier removal function `update_bmi()` that recalculates BMI
bmi_final <- update_bmi(data = bmi_data)
```

Comparing the data frames **bmi_data_clean_final** and **bmi_final**, as they are supposedly the same.

```
#summary of bmi_data_clean_final
summary(bmi_data_clean_final)
```

```
##       age             bai             bmi           body_fat
##  Min.   : 0.00   Min.   : 0.00   Min.   :16.16   Min.   : 0.00
##  1st Qu.:30.00   1st Qu.:25.20   1st Qu.:21.94   1st Qu.:22.40
##  Median :39.00   Median :27.77   Median :24.51   Median :27.80
```

```
##  Mean   :39.07   Mean   :28.31   Mean   :25.00   Mean   :27.59
##  3rd Qu.:47.00   3rd Qu.:31.12   3rd Qu.:27.55   3rd Qu.:33.00
##  Max.   :69.00   Max.   :46.97   Max.   :38.75   Max.   :47.20
##                                  NA's   :10
##     density          weight          height          sex
##  Min.   :0.000   Min.   :  0.00   Min.   :  0.0   Length:3120
##  1st Qu.:1.033   1st Qu.: 57.38   1st Qu.:159.8   Class :character
##  Median :1.043   Median : 66.77   Median :162.7   Mode  :character
##  Mean   :1.040   Mean   : 67.76   Mean   :164.0
##  3rd Qu.:1.054   3rd Qu.: 76.93   3rd Qu.:169.7
##  Max.   :1.091   Max.   :108.24   Max.   :176.2
##
```

```
#summary of bmi_final
summary(bmi_final)
```

```
##       age              bai              bmi           body_fat
##  Min.   : 0.00   Min.   : 0.00   Min.   :16.16   Min.   : 0.00
##  1st Qu.:30.00   1st Qu.:25.20   1st Qu.:21.94   1st Qu.:22.40
##  Median :39.00   Median :27.77   Median :24.51   Median :27.80
##  Mean   :39.07   Mean   :28.31   Mean   :25.00   Mean   :27.59
##  3rd Qu.:47.00   3rd Qu.:31.12   3rd Qu.:27.55   3rd Qu.:33.00
##  Max.   :69.00   Max.   :46.97   Max.   :38.75   Max.   :47.20
##                                  NA's   :10
##     density          weight          height          sex
##  Min.   :0.000   Min.   :  0.00   Min.   :  0.0   Length:3120
##  1st Qu.:1.033   1st Qu.: 57.38   1st Qu.:159.8   Class :character
##  Median :1.043   Median : 66.77   Median :162.7   Mode  :character
##  Mean   :1.040   Mean   : 67.76   Mean   :164.0
##  3rd Qu.:1.054   3rd Qu.: 76.93   3rd Qu.:169.7
##  Max.   :1.091   Max.   :108.24   Max.   :176.2
##
```

```
#check using identical
identical(bmi_data_clean_final, bmi_final)
```

```
## [1] TRUE
```

```
#check using all.equal
all.equal(bmi_data_clean_final, bmi_final)
```

```
## [1] TRUE
```

```
#check using waldo:compare
compare(bmi_data_clean_final, bmi_final)
```

```
## v No differences
```

2.2 Regression Analysis

Now that we have cleaned the data, we will perform regression-based analysis for the bmi data.

Q7

Using simple linear regression to check how well `bmi` (covariate) can predict `body_fat` (response). Check assumptions of the simple linear regression model and also plot the residuals. Comment on the results of your analysis.

Comment on the residuals, what is the shape and what does this tell you about the model fit and any other covariates that should be included?
[5 MARKS]

Answer 7

After completing data cleaning, collecting BMI, and removing multivariate outliers through PCA, we conducted a regression analysis on the cleaned dataset **`bmi_final`** to explore the connection between BMI (`bmi`) and body fat percentage (`body_fat`). The goal was to evaluate the effectiveness of BMI in predicting body fat using a basic linear regression model, and to determine if the model's assumptions were fulfilled.

Model Specification

A simple linear regression model was fitted with ***body fat percentage*** as a **response variable** (y_i = `body_fat`) and ***BMI*** as the **explanatory/predictor variable** (x_i **or covariate** = `bmi`):

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

where,
- y_i = Body fat percentage (`body_fat`)
- x_i = Body Mass Index or BMI (`bmi`)
- β_0 = Regression coefficient for intercept / y-intercept when x is 0
- β_1 = Regression coefficient for slope / Effect of covariate on response
- ε_i = Error term / Noise / Random variation

The **`bmi_clean`** dataset has been utilized to perform a regression of y_i on x_i using the base R function **`lm()`**, which by default applies the **least squares estimation method** for this straightforward linear regression.

```
#simple linear regression model, where body_fat is response and bmi is predictor
ls_bmi <- lm(body_fat ~ bmi, data = bmi_final)

# Model summary
summary(ls_bmi)
```

```
##
## Call:
## lm(formula = body_fat ~ bmi, data = bmi_final)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -19.9797  -5.1270   0.1401   5.5394  21.3311
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.50215    0.76094   5.917 3.65e-09 ***
## bmi          0.92725    0.03004  30.864  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.81 on 3108 degrees of freedom
##   (10 observations deleted due to missingness)
## Multiple R-squared:  0.2346, Adjusted R-squared:  0.2343
## F-statistic: 952.6 on 1 and 3108 DF,  p-value: < 2.2e-16
```

```r
#regression coefficients as a named vector
ls_coefficients <- coef(ls_bmi)
print(ls_coefficients)
```

```
## (Intercept)         bmi
##   4.5021499   0.9272454
```

The fitted model is as follows:

$$\texttt{body\_fat}(y_i) = 4.502 + 0.927 \times \texttt{bmi}(x_i)$$

where,
- y_i = body_fat
- x_i = bmi
- $\beta_0 = 4.502$
- $\beta_1 = 0.927$

Regression Results Interpretation

1. Model Coefficient Interpretation

Please note that the residuals have not been covered in this section. They have been discussed following the QQ plot and are addressed in a separate section at the end.

- **Regression Coefficient for Intercept ($\beta_0 = 4.502$) Estimate:** The intercept indicates the ***projected body fat percentage*** when ***BMI = 0***. While a BMI of zero does not have physiological significance, the intercept is essential for the purposes of model estimation and acts as a **reference point** for the model. It is statistically significant **(p = 3.65e-09)**, showing that it is different from zero.
- **Regression Coefficient for Slope ($\beta_1 = 0.927$) Estimate:** The BMI coefficient is both ***positive and statistically significant*** **(p < 2e-16)**. This suggests that for **each unit increase in BMI, the average body fat percentage rises by about 0.927 units**. This robust positive correlation aligns with biological expectations and reinforces BMI's role as a key predictor of body fat.

2. Model Fit and Variance Explained

- **Adjusted R-squared (Adjusted $R^2 = 0.2343$):** The BMI accounts for **23.5% of the variation** in body fat percentage. Although this shows a significant relationship and that BMI serves as an important predictor, it also indicates that a ***large portion of variability is still unaccounted for***. Therefore, other factors also play a role in influencing body fat levels.
- **Residual Standard Error (6.81):** Predicted body fat values differ from observed values by approximately **6.8 percentage points** on average. This indicates a reasonable level of predictive accuracy for a model that uses a **single covariate**.
- **F-statistic = 952.6, p-value < 2.2e-16, (3108 degrees of freedom):** The highly significant F-test indicates that BMI considerably enhances body fat prediction as compared to a null model without any predictors.

Assumption Checks for Simple Linear Regression Model

Once the simple linear regression model has been fitted, it is crucial to check if the fundamental assumptions of linear regression are adequately met. These assumptions are vital for the validity of parameter estimates, statistical tests, and confidence intervals. To evaluate these assumptions, both individual diagnostic plots and a standardized panel for regression diagnostics were analyzed. The four-panel overview of regression diagnostics *(residuals against fitted values, normal QQ plot, scale-location plot, and residuals against leverage plot)*, offers a thorough visual evaluation of the model's adequacy.

Key assumptions evaluated:

1. Linearity

2. Homoscedasticity (constant variance of residuals)

3. Normality of residuals

1. Linearity

The assumption of linearity suggests that the connection between the predictor (`bmi`) and the response (`body_fat`) is roughly linear.

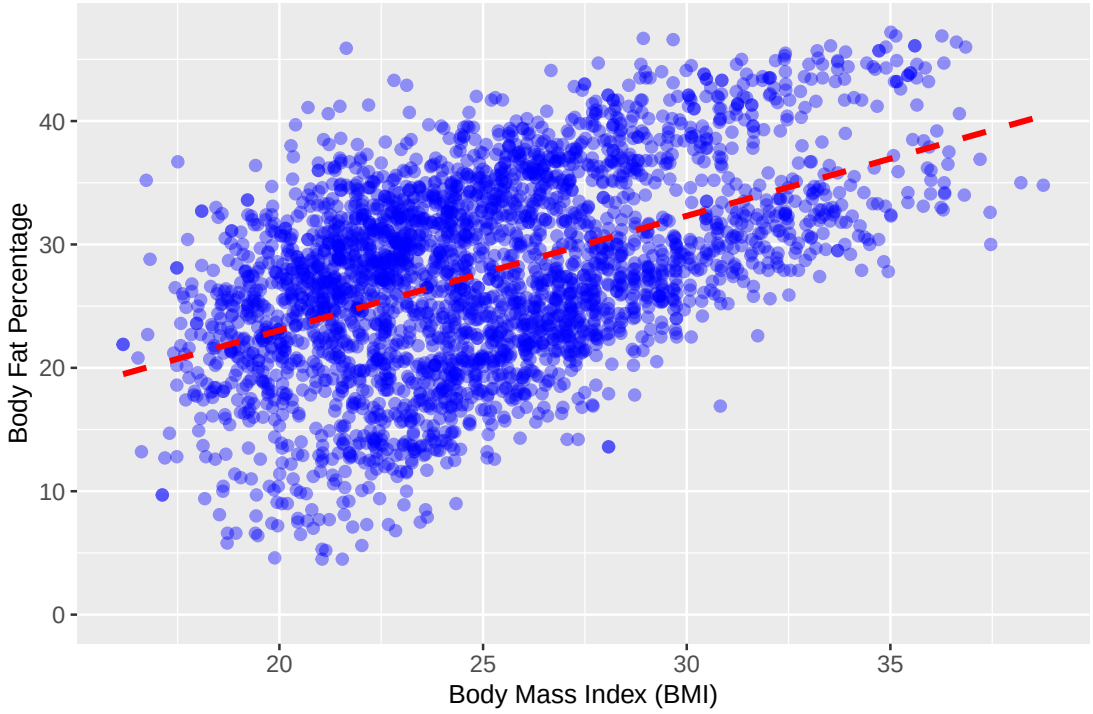
- The highly significant slope coefficient **(p < 2e-16)** demonstrates a strong positive linear relationship between BMI and body fat.
- With each one unit rise in BMI, the anticipated body fat percentage **increases by about 0.96 units**.
- The scatter plot (Fig. 12) illustrating body fat against BMI (including the fitted regression line) clearly indicates an **upward trend** that supports the suitability of a linear model.

The following code chunk is for generating Fig. 12:

```
#use ggplot
ggplot(data = bmi_final,
       aes(x = bmi,
           y = body_fat)) +

  #scatter points
  geom_point(colour = "blue",
             alpha = 0.4,
             size = 2) +

  #add regression line
  geom_smooth(method = "lm",          #use least square method
              se = FALSE,
              colour = "red",
              linewidth = 1.2,
              linetype = "dashed") +
  #plot appearance
    theme(
      plot.title  = element_text(size = 10, face = "bold", hjust = 0.5),
      axis.title.x = element_text(size = 11),
      axis.title.y = element_text(size = 11),
      axis.text.x  = element_text(size = 10),
      axis.text.y  = element_text(size = 10),
    ) +

    #title and axis labels
    labs(
      title = "Fig. 12: Relationship Between BMI and Body Fat Percentage",
      x     = "Body Mass Index (BMI)",
      y     = "Body Fat Percentage"
    )
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

Fig. 12: Relationship Between BMI and Body Fat Percentage



In Fig. 12, every blue dot symbolizes a single observation, with BMI plotted on the X axis and body fat percentage on the Y axis. The red regression line illustrates that **as BMI rises, body fat also tends to increase**, thereby validating the linear assumption of simple linear regression. While the trend is evident, the ***distribution of points becomes broader at elevated BMI levels***, suggesting ***greater variability in body fat among individuals with higher BMI***.

2. Homoscedasticity (constant variance of residuals)

A key assumption of simple linear regression is homoscedasticity, which stipulates that the ***variance of the residuals should be roughly constant throughout the entire range of predicted values***. In other terms, the distribution of residuals ought to be similar for low, medium, and high predicted values of the outcome variable.

Breaches of this assumption occur when the ***variability in residuals systematically increases or decreases alongside the predicted values***, a situation referred to as heteroscedasticity.

To visually evaluate this assumption, a plot of residuals against fitted values (Fig. 13) was created. In this diagnostic plot, the residuals should resemble a random scatter centered around zero, without any discernible pattern or trend, and evenly distributed across the range of fitted values.

The following code chunk is for generating Fig. 13:

```
#data frame from fitted values and residuals from model
res_df <- data.frame(fitted = fitted(ls_bmi),         #fitted values
                     residuals = resid(ls_bmi),       #residuals
                     std_resid = rstandard(ls_bmi), #standardised residuals
                     leverage  = hatvalues(ls_bmi), #leverage (hat values)
                     cooks = cooks.distance(ls_bmi))     #Cook's distance

#using ggplot
ggplot(data = res_df,
       aes(x = fitted,
           y = residuals)) +

  #scatter points
  geom_point(colour = "blue",
```

```
            alpha = 0.4,
            size = 2) +

  #horizental line
  geom_hline(yintercept = 0,
            colour = "red",
            linewidth = 1.2,
            linetype = "dashed") +

  #plot appearance
    theme(
      plot.title  = element_text(size = 10, face = "bold", hjust = 0.5),
      axis.title.x = element_text(size = 11),
      axis.title.y = element_text(size = 11),
      axis.text.x  = element_text(size = 10),
      axis.text.y  = element_text(size = 10),
    ) +

    #title and axis labels
    labs(
      title = "Fig. 13: Residuals vs Fitted Values",
      x     = "Fitted values",
      y     = "Residuals"
    )
```

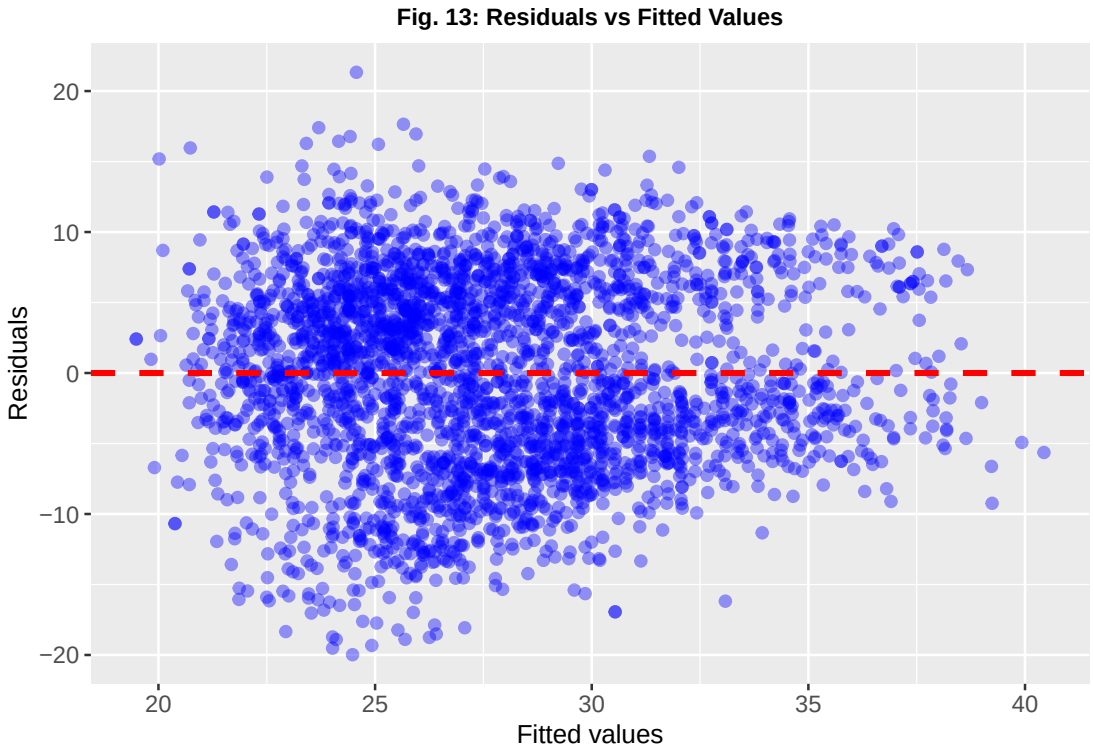


Fig. 13 illustrates the residuals plotted against the fitted values from the simple linear regression model relating body fat percentage to BMI.

The main points of observation are as follows:

i. The residuals are **evenly distributed around the zero line (horizontal reference)** across the entire range of predicted values. This suggests that the **model does not consistently overestimate or underestimate** body fat and that a linear regression model is suitable.

ii. There is an **absence of any noticeable curved or systematic pattern** in the residuals (for instance, U-shaped or inverted U-shaped patterns). This reinforces the idea that the linear model effectively captures the relationship between BMI and body fat.

iii. The **variability of the residuals** tends to **increase slightly with higher fitted values**, especially at **elevated predicted body fat percentages**. This pattern, **resembling a fan shape, indicates mild heteroscedasticity**, wherein the variability of body fat increases for individuals with greater BMI. Such a trend is biologically plausible, as body fat composition may vary more significantly at higher BMI levels.

iv. Although some residuals reach values around ± 20, these are not isolated or extreme relative to the main cluster of points. This implies that no individual data point is disproportionately affecting the fitted model.

3. Normality of residuals

The normality of residuals is crucial for valid hypothesis testing and constructing confidence intervals. This assumption suggests that the differences between the observed and predicted values result from random variability instead of systematic model mis-specification.

To assess this assumption, a normal QQ (quantile-quantile) plot (Fig. 14B) of the residuals was analysed. When the residuals follow a normal distribution, the points displayed should roughly align with the reference line. Deviations from this line, especially in the extremities, indicate a departure from normality and may point to skewness, heavy tails, or lingering outliers.

To further support the QQ plot, a histogram (Fig. 14A) of the regression residuals was included as an additional diagnostic tool. The histogram offers a straightforward visual representation of the empirical distribution of the residuals, enabling the evaluation of symmetry around zero, the overall shape (whether bell-shaped or skewed), and the existence of extreme values or heavy tails.

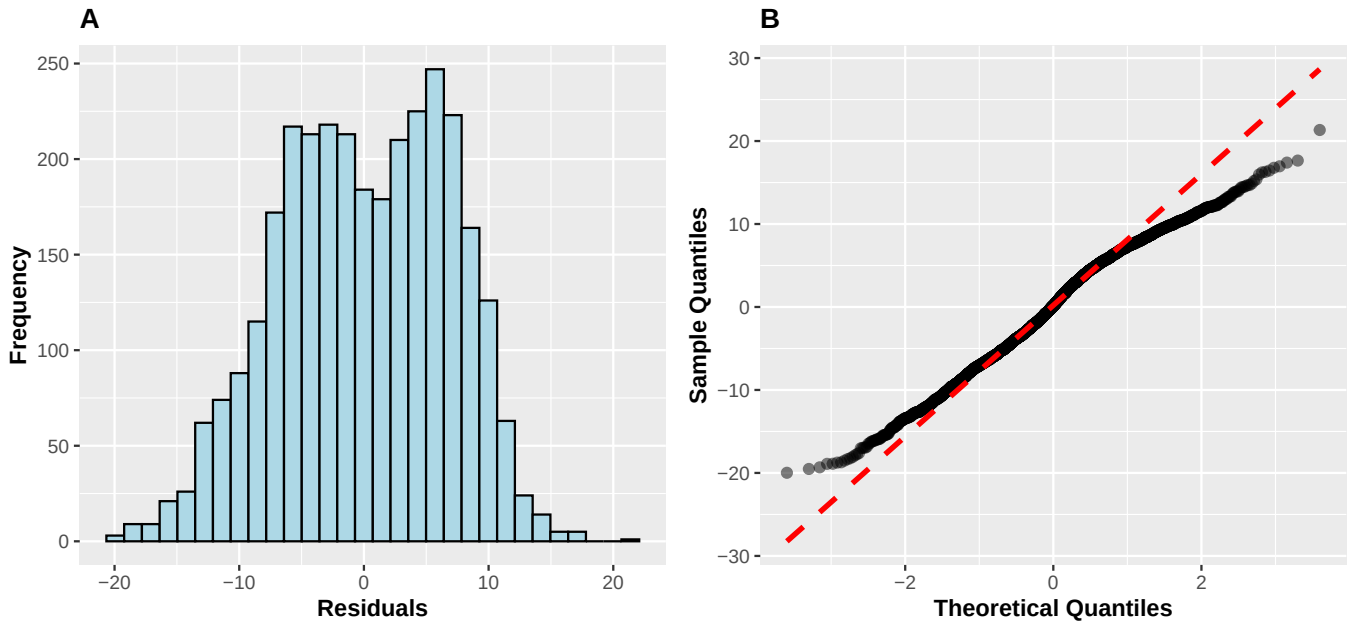
Utilizing both plots together provides a more comprehensive and intuitive examination of the normality assumption, combining the distributional shape (histogram) with a quantile-based assessment (QQ plot).

The following code chunk is for generating Fig. 14:

```
#use ggplot for the histogram
p_hist <- ggplot(data = res_df,
                 aes(x = residuals)) +
  #histogram
  geom_histogram(bins = 30,
                 fill = "lightblue",
                 color = "black") +
  #axes labels
  labs(x = "Residuals", y = "Frequency") +

  #plot subtitle
  ggtitle("A") +

  #appearance
  theme(
    plot.title  = element_text(face = "bold", hjust = 0),
    axis.title.x = element_text(size = 11, face = "bold"),
    axis.title.y = element_text(size = 11, face = "bold")
  )

#QQ plot using ggplot
p_qq <- ggplot(data = res_df,
               aes(sample = residuals)) +

  #add the QQ points layer
  stat_qq(color = alpha("black", 0.5),
          size = 2,
          shape = 19) +
```

```
  #add the QQ line
  stat_qq_line(color = "red",
          linewidth = 1.2,
          linetype = "dashed") +

  #plot subtitle
  ggtitle("B") +

  #axes labels
  labs(x = "Theoretical Quantiles",
       y = "Sample Quantiles") +

  #appearance
  theme(
    plot.title  = element_text(face = "bold", hjust = 0),
    axis.title.x = element_text(size = 11, face = "bold"),
    axis.title.y = element_text(size = 11, face = "bold")
  )

#merging the plots using patchwork
(p_hist | p_qq) +
  #adding title to merged plot
  plot_annotation(title =
"Fig. 14: Diagnostic Plots for Regression Residuals: A. Histogram, B. QQ Plot",

    #appearance
    theme = theme(
      plot.title = element_text(face = "bold", hjust = 0.5, size = 10)
    )
  )
```

Fig. 14: Diagnostic Plots for Regression Residuals: A. Histogram, B. QQ Plot



The main points of observation are as follows:

Panel A: Histogram of Residuals (Fig. 14A)

i. The residuals cluster around zero, indicating that there is ***no consistent over or under-prediction*** by the model.

ii. The distribution is roughly symmetric, resembling a bell-shaped curve.

iii. Slightly elongated tails can be seen at both ends, suggesting ***minor deviations from perfect normality***.

iv. **Two peaks are nearly observable**, which implies that the dataset may include ***systematically different subgroups***, such as males versus females, or individuals with varying body composition profiles (for instance, high muscle mass compared to high fat mass at similar BMI levels). Since the recreation model uses **only BMI as the predictor, the residuals tend to cluster differently among the subgroups**, leading to reduced multiple peaks.

Panel B: QQ Plot of Residuals (Fig. 14B)

i. Most points are situated **near the reference line, particularly in the centre**, indicating a ***strong alignment with a normal distribution***.

ii. Deviations can be seen in the **lower and upper tails**, consistent with the **slightly heavier tails observed in the histogram**.

iii. These deviations remain minor and are not unexpected due to the large sample size.

In summary, both the histogram and QQ plot suggest that the regression residuals are nearly normally distributed, with only slight tail deviations. Such minor departures do not breach the assumptions of linear regression and do not undermine the integrity of statistical inference, especially in large datasets.

4. Checking Regression Assumption Using A Four-Panel Summary Diagnostic Plots

Alongside the generation of individual diagnostic plots such as scatter plots, residual-fitted plots, histograms, and QQ plots, a standardized four-panel diagnostic summary for linear regression models (Fig. 15) is also included. These visualizations facilitate a simultaneous evaluation of the fundamental assumptions underlying single linear regression: linearity, homoscedasticity, normality of residuals, and the absence of influential observations.

The produced plots comprise the following:

1. **Residuals vs Fitted Plot:** This visual representation examines linearity and homoscedasticity.
2. **Normal QQ Plot:** This plot is crucial for the accuracy of statistical inferences, which include t-tests, F-tests, confidence intervals, and p-values provided by the model.
3. **Scale-Location Plot:** This visualization assesses whether residual variance remains constant across the fitted values.
4. **Residuals vs Leverage Plot:** This diagnostic plot evaluates whether any observations significantly impact the model.

The following code chunk is for generating Fig. 15:

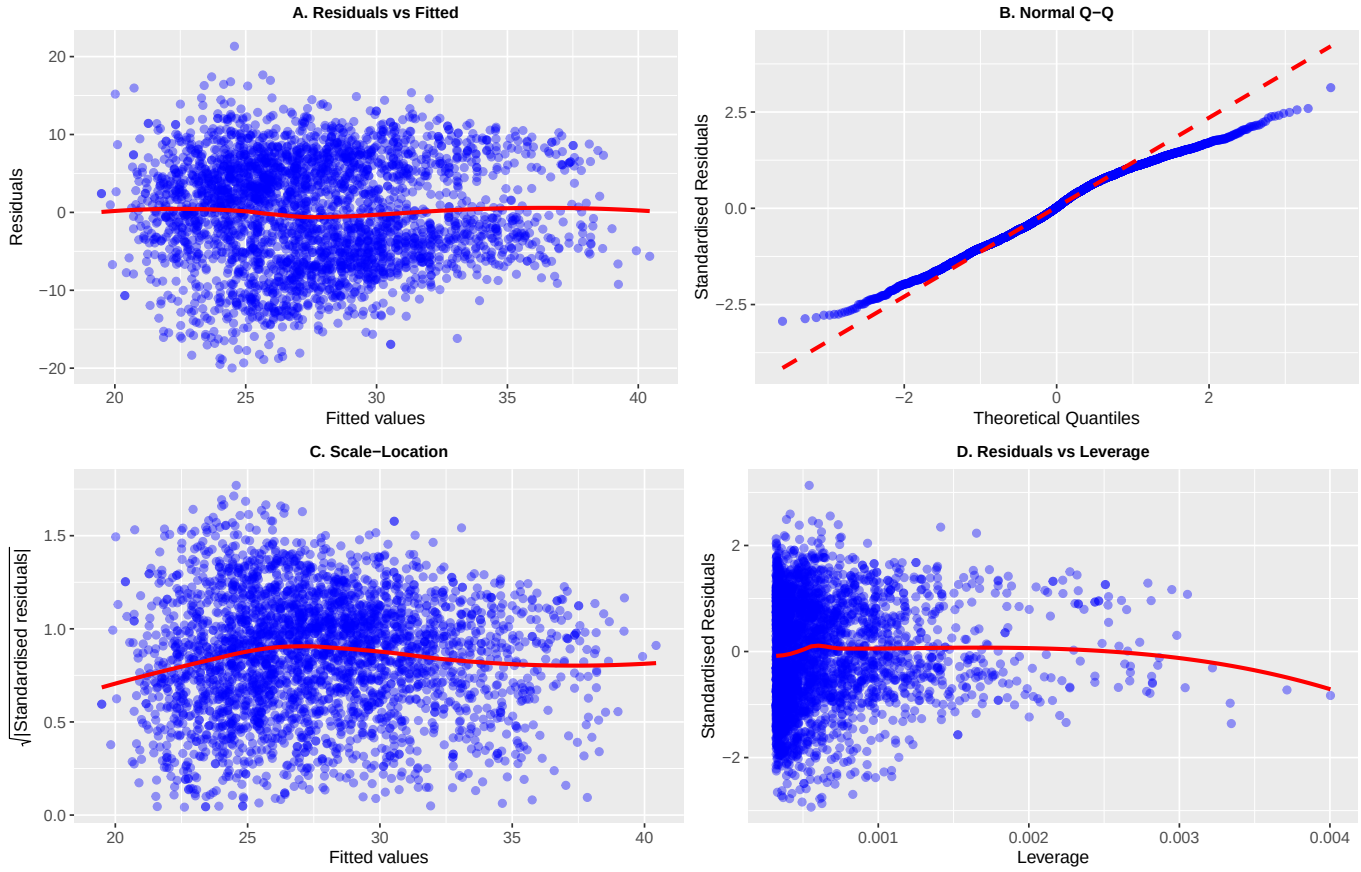
```
#residual vs fitted (A)
p1 <- ggplot(data = res_df,
       aes(x = fitted,
           y = residuals)) +
  #scatter points
  geom_point(colour = "blue",
             alpha = 0.4,
             size = 2) +
  #smooth
  geom_smooth(method = "loess",
              se = FALSE,
              colour = "red",
              linewidth = 1.2) +
  #plot subtitle
  ggtitle("A. Residuals vs Fitted") +
```

```
  #plot appearance
    theme(
      plot.title  = element_text(size = 10, face = "bold", hjust = 0.5),
      axis.title.x = element_text(size = 11),
      axis.title.y = element_text(size = 11),
      axis.text.x  = element_text(size = 10),
      axis.text.y  = element_text(size = 10),
    ) +
    #axis labels
    labs(
      x     = "Fitted values",
      y     = "Residuals"
    )

#mormal QQ (B)
p2 <- ggplot(data = res_df,
               aes(sample = std_resid)) +
  #add the QQ points layer
  stat_qq(color = alpha("blue", 0.5),
          size = 2,
          shape = 19) +
  #add the QQ line
  stat_qq_line(color = "red",
          linewidth = 1.2,
          linetype = "dashed") +
  #plot subtitle
  ggtitle("B. Normal Q-Q") +
  #axes labels
  labs(x = "Theoretical Quantiles",
       y = "Standardised Residuals") +
  #appearance
  theme(
      plot.title  = element_text(size = 10, face = "bold", hjust = 0.5),
      axis.title.x = element_text(size = 11),
      axis.title.y = element_text(size = 11),
      axis.text.x  = element_text(size = 10),
      axis.text.y  = element_text(size = 10),
    )

#scale-location (C)
res_df$sqrt_std_resid <- sqrt(abs(res_df$std_resid))
p3 <- ggplot(data = res_df,
             aes(x = fitted,
                 y = sqrt_std_resid)) +
  #scatter points
  geom_point(colour = "blue",
             alpha = 0.4,
             size = 2) +
  #smooth
  geom_smooth(method = "loess",
              se = FALSE,
              colour = "red",
              linewidth = 1.2) +
  #plot appearance
    theme(
      plot.title  = element_text(size = 10, face = "bold", hjust = 0.5),
      axis.title.x = element_text(size = 11),
      axis.title.y = element_text(size = 11),
      axis.text.x  = element_text(size = 10),
      axis.text.y  = element_text(size = 10),
```

```
    ) +
  #plot subtitle
  ggtitle("C. Scale-Location") +
    #axis labels
    labs(
      x     = "Fitted values",
      y     = expression(sqrt("|Standardised residuals|"))
    )

#residual vs leverage
p4 <- ggplot(data = res_df,
             aes(x = leverage,
                 y = std_resid)) +
  #scatter points
  geom_point(colour = "blue",
             alpha = 0.4,
             size = 2) +
  #smooth
  geom_smooth(method = "loess",
              se = FALSE,
              colour = "red",
              linewidth = 1.2) +
  #plot subtitle
  ggtitle("D. Residuals vs Leverage") +
  #plot appearance
    theme(
      plot.title  = element_text(size = 10, face = "bold", hjust = 0.5),
      axis.title.x = element_text(size = 11),
      axis.title.y = element_text(size = 11),
      axis.text.x  = element_text(size = 10),
      axis.text.y  = element_text(size = 10),
    ) +
    #axis labels
    labs(
      x     = "Leverage",
      y     = "Standardised Residuals"
    )

#merging the plots using patchwork
(p1 | p2) / (p3 | p4) +
  #adding title to merged plot
  plot_annotation(title =
"Fig. 15: Regression Diagnostic Plots for the Simple Linear Regression Model",

    #appearance
    theme = theme(
      plot.title = element_text(face = "bold", hjust = 0.5, size = 12)
    )
  )
```

```
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
```

Fig. 15: Regression Diagnostic Plots for the Simple Linear Regression Model



The main points of observation are as follows:

Panel A: Residuals vs Fitted Plot (Fig. 15, topleft)
The residuals are **focused around zero**, showing that the model does not **systematically overestimate or underestimate predictions**, which aligns with my individual plotting outcomes. There are no significant curves observed, which supports the assumption of a linear relationship between BMI and body fat.

Panel B: Normal QQ Plot (Fig. 15, topright)
The majority of points **closely adhere to the reference line, particularly in the central quantiles**, indicating that the **distribution is approximately normal**. Small deviations at the extremes suggest slightly heavier tails than would be expected in a completely normal distribution. However, due to the large sample size, these deviations are not severe enough to affect inference.

Panel C: Scale-Location Plot (Fig. 15, bottomleft)
The scale-location plot examines homoscedasticity further. The **relatively flat trend line** suggests that the **variance of residuals remains stable across the range of fitted values**. A slight upward trend at higher fitted values indicates minor heteroscedasticity, but it is not significant enough to undermine the model's reliability.

Panel D: Residuals vs Leverage Plot (Fig. 15, bottomright)
This plot highlights influential data points. The **majority** of observations have **low leverage with concentrated standardized residuals**. A few points display higher leverage associated with extreme BMI values, which is anticipated in simple linear regression. Most of these points stayed **within Cook's distance limits, suggesting that most observations did not have a significant impact** on the fitted model.

Residual Distribution

```
summary(resid(ls_bmi))
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -19.9797  -5.1270   0.1401   0.0000   5.5394  21.3311
```

1. Residual Summary

The residuals are nearly **centered at zero (median = 0.1401)**, showing there is **no consistent over or under-prediction** from the model, as can also be seen in the histogram and QQ plot of the residuals (Fig. 14). The **range of the residuals (-19.97 to 21.33) indicates a degree of variability**, especially at extreme BMI values. These variations are minor and anticipated due to the large sample size.

2. Shape of Residuals

The figures, specifically Fig. 14 and Fig. 15 (top right), indicate that the residuals exhibit a **trend of approximate symmetry but are moderately spread out at both ends**, suggesting that while the **linear model captures the main trend, it does not account for all the variability**. The histogram presents a distribution resembling a bell shape, though it has slightly heavier tails. This observation is supported by the QQ plot, which reveals minor deviations from normality at both ends. The extent of the residuals indicates that body fat shows significant variation among individuals who have comparable BMI values.

There is a likelihood of mild heteroscedasticity, as the variability tends to rise with increasing BMI values. The scale-location plot (Fig. 15, bottom left) supports this observation by indicating a slight upward trend. The residuals versus leverage plot (Fig. 15, bottom right) reveals **several points with high BMI that have greater leverage, but none exhibit significant residuals or high Cook's distance**. Therefore, there is no individual point that is affecting the regression line significantly; the model maintains its stability.

3. Implications on how fit is the Model

1. The model effectively captures the primary linear relationship between BMI and body fat.
2. It demonstrates a good fit, but only accounts for a **moderate amount of variation** ($R^2 \approx 0.23$).
3. The model indicates unrepresented structure (as seen with slight bimodality in the histogram in Fig. 14A), suggesting that ***BMI alone does not completely explain differences in body fat percentage***.

Therefore, although the model is statistically sound, **body fat is affected by more factors than just BMI**.

Model Limitations and Need for Additional Covariates

While BMI is a powerful and significantly important predictor, the **comparatively low R^2 value** suggests that **BMI by itself cannot completely account for variations in body fat**. This is anticipated since BMI does not differentiate between fat mass and lean muscle.

Important covariates that could improve the model

i. **Sex:** Men and women exhibit different distributions of body fat at the same BMI, which could account for the slight bimodal nature of the residuals observed in the histogram.

ii. **Body Area Index (BAI) and Density:** Incorporating Body Area Index (BAI) and density as predictors could decrease unexplained variance and offer more accurate insights into adiposity and body composition, aiding in the distinction of individuals with the same BMI but varying fat and lean mass proportions.

iii. **Height and Weight:** Height and weight can reveal non-linear interaction patterns that BMI consolidates into one index.

iv. **Age:** Age affects the accumulation of fat and loss of muscle, influencing body fat independent of BMI.

Q8

Now that we have cleaned the data, we will perform regression-based analysis for the BMI data.

You have checked on how well bmi can predict `body_fat`. Here we will check how well additional covariates can predict `body_fat`. We use the predictors `age`, `bmi`, `bai`, `density`, `weight`, `height`, `sex`, check which of these covariates are significant predictors of `body_fat` using a linear regression model. You should also check the assumptions of the model and plot the residuals. Comment on the results of your analysis. **[6 MARKS]**

Answer 8

After completing data cleaning, collecting BMI, and removing multivariate outliers through PCA, we conducted a regression analysis on the cleaned dataset **bmi_final** to explore the connection between BMI (bmi) and body fat percentage (body_fat). The goal was to evaluate the effectiveness of BMI in predicting body fat using a basic linear regression model, and to determine if the model's assumptions were fulfilled.

In order to **enhance predictive accuracy and lessen unexplained variability** several biologically relevant predictors have been utilised to fit a **multiple linear regressor** or a regression model which includes **age**, **bai**, **bmi**, **weight**, **density**, **height** and **sex**. These variables are selected, as they present various dimensions of body size, fat distribution, and compositions. The objective is to determine which covariate effectively predict **body_fat** and assess if a multivariate model provides a superior explanation as compared to various model that uses only BMI.

Model Specification

When adding **extra predictors** to the basic regression model, the **percentage of body fat (y_i)** is represented as a ***linear combination of the multiple covariates*** used for the prediction, which can be represent as follows:

$$y_i = \beta_0 + \beta_1 x_1(age) + \beta_2 x_2(bmi) + \beta_3 x_3(bai) + \beta_4 x_4(density) + \beta_5 x_5(weight) + \beta_6 x_6(height) + \beta_7 x_7(sex) + \varepsilon_i$$

where:

- y_i = Body fat percentage (body_fat)
- x_1,..., x_7 = Covariates (age, bmi, bai, density, weight, height, and sex)
- β_0 = Regression coefficient for y-intercept when x is 0
- β_1,..., β_7 = Regression coefficients for each covariate
- ε_i = Error term

This model reflects the combined impact of biological and demographic factors in estimating the body fat percentage.

Fitting the Multiple Linear Regression Model

```
#cleaned dataset from update_bmi()
bmi_mlr <- bmi_final

#sex to categorical variable
bmi_mlr$sex <- ifelse(bmi_mlr$sex == "male", 1, 2)     #male = 1, female = 2

#fitting a multiple linear regressor for body fat percentage
#predictors: age, bmi, bai, density, weight, height, sex
multi_lr <- lm(
  body_fat ~ age + bmi + bai + density + weight + height + sex,
  data = bmi_mlr
)

#model summary
summary(multi_lr)
```

```
##
## Call:
## lm(formula = body_fat ~ age + bmi + bai + density + weight +
##     height + sex, data = bmi_mlr)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.68154 -0.32225 -0.00424  0.32585  1.79108
##
## Coefficients:
##               Estimate Std. Error  t value Pr(>|t|)
```

```
## (Intercept)  5.428e+02  2.286e+00  237.395  < 2e-16 ***
## age          2.150e-04  8.785e-04    0.245 0.806711
## bmi          2.083e-02  3.627e-02    0.574 0.565801
## bai          1.271e-02  3.443e-03    3.693 0.000226 ***
## density     -4.944e+02  1.066e+00 -463.842  < 2e-16 ***
## weight      -4.141e-03  1.318e-02   -0.314 0.753357
## height       1.393e-03  1.171e-02    0.119 0.905332
## sex          4.124e-02  5.710e-02    0.722 0.470189
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4882 on 3102 degrees of freedom
##   (10 observations deleted due to missingness)
## Multiple R-squared:  0.9961, Adjusted R-squared:  0.9961
## F-statistic: 1.124e+05 on 7 and 3102 DF,  p-value: < 2.2e-16
```

```r
#extracting regression coefficients
multi_lr_coefficient <- coef(multi_lr)
print(multi_lr_coefficient)
```

```
##   (Intercept)           age           bmi           bai       density
##  5.427648e+02  2.149670e-04  2.082784e-02  1.271439e-02 -4.943952e+02
##        weight        height           sex
## -4.141204e-03  1.392599e-03  4.124223e-02
```

Regression Results Interpretation

1. Model Fit and Variance Explained

- **R-squared Values:** The model accounts for **99.61% of the variability in body fat** as demonstrated by the hypothesis testing of fitted model, where:
  - **Multiple R-squared:** **0.9961**, and
  - **Adjusted R-squared:** **0.9961** also.

  These means that more than ***99.6% of the variability*** in the body fat can be explained by the ***predictors included*** in the model: `age`, `bai`, `bmi`, `weight`, `density`, `height` and `sex`.

  This indicates a **highly robust multivariate relationship**, suggesting that this **group of predictors together offers a remarkably precise estimation** of body fat percentage (`body_fat`). This is vastly ***more effective*** than the ***simple linear regression model based solely on BMI***, which **accounted for only 23% of the variance** in the body fat percentage.

- **Residual Standard Error (0.4882):** The residuals are generally very minor, **usually falling within ± 0.5 units** of body fat, which demonstrates an exceptionally precise model fit. The ***residual standard error of 0.4482***, for this ***multiple regression model*** is ***significantly less than*** that of the ***simple regression model*** based on BMI alone having ***residual standard error 6.81***, indicating that the incorporation of multiple covariates enhance the models ability to predict body fat percentage (`body_fat`).

- **F-statistic = 1.124e+05, p-value < 2.2e-16:** The F-statistic is **1.124e+05** based on **3102 degrees of freedom** for the residuals with a **p-value < 2.2e-16** indicating that overall the model is extremely statistically significant.
  When comparing to the ***BMI only simple regression model*** with **F-statistic = 952.6** and **p-value < 2.2e-16**, the multiple linear regression model shows a significantly superior fit.

Interpretation of Significance of Different Covariates Interpretation

- **Density (Highly Significant):** Density exerts are considerable, negative, and highly significant impact with a **p-value < 2.2e-16**, making it the most critical and prominent predictor in the model. The **estimated coefficient is approximately -4.94**, suggesting that ***minor variations in density lead to substantial changes in***

predicted body fat. This finding alings with the principles of body composition physics, where body density serves as a notably strong inverse predictor of body fat percentage.

Furthermore, this is probably due to the fact that, **body density is closely associated, both mathematically and physiologically with body fat percentage** via the conversion equation that relates density (`density`) to body fat percentage (`body_fat`):

$$\textbf{Body fat \%} = \frac{495}{\textbf{Density}} - 450$$

This indicates that density (`density`) largely dictates body fat percentage (`body_fat`), resulting in the model attaining an $R^2 = 0.9961$.

- **BAI or Body Area Index (Moderately Significant):** BAI a body area index is an important positive indicator of body fat, having a **p-value = 0.000226** and an **estimate = 1.271439e-02** reflecting strong statistical significance. ***An increase in BAI is associated with an increase in body fat***, which aligns with expectations, as **BAI take into account both height and hip circumference** in its formula.
- **Age (p-value = 0.806711), BMI (p-value = 0.565801), Weight (p-value = 0.753357), Height (p-value = 0.905332), Sex (p-value = 0.470189):** Once density and BAI are accounted for in the model to predict body fat percentage, these predictors do not show statistical significance, as suggested by their ***p-values***, which are all **significantly greater than the commonly accepted threshold of 0.05**.

This interprets as:

1. Their role in estimating body fat is unnecessary due to the presence of more effective predictors such as density and BAI.
2. The model, **through density and BAI**, already **accounts for nearly 99.61% of all the variations**, leaving a very minimal explanation possible from the other additional covariates.
3. BMI which ***initially accounted for 23.46% of the variability on its own***, has now lost its significance in the multiple regression model with a **p-value = 0.565801**, after the introduction of more closely related variables. Furthermore, it has a strong correlation with other factors such as height and weight, resulting in a decrease in its statistical significance.

Normality of residuals

To assess the normality of residuals, a normal QQ (quantile-quantile) plot (Fig. 16B) was generated and analysed. When the residuals follow a normal distribution, the points displayed should roughly align with the reference line. Deviations from this line, especially in the extremities, indicate a departure from normality and may point to skewness, heavy tails, or lingering outliers.

To further support the QQ plot, a histogram (Fig. 16A) of the regression residuals was included as an additional diagnostic tool. The histogram offers a straightforward visual representation of the empirical distribution of the residuals, enabling the evaluation of symmetry around zero, the overall shape (whether bell-shaped or skewed), and the existence of extreme values or heavy tails.

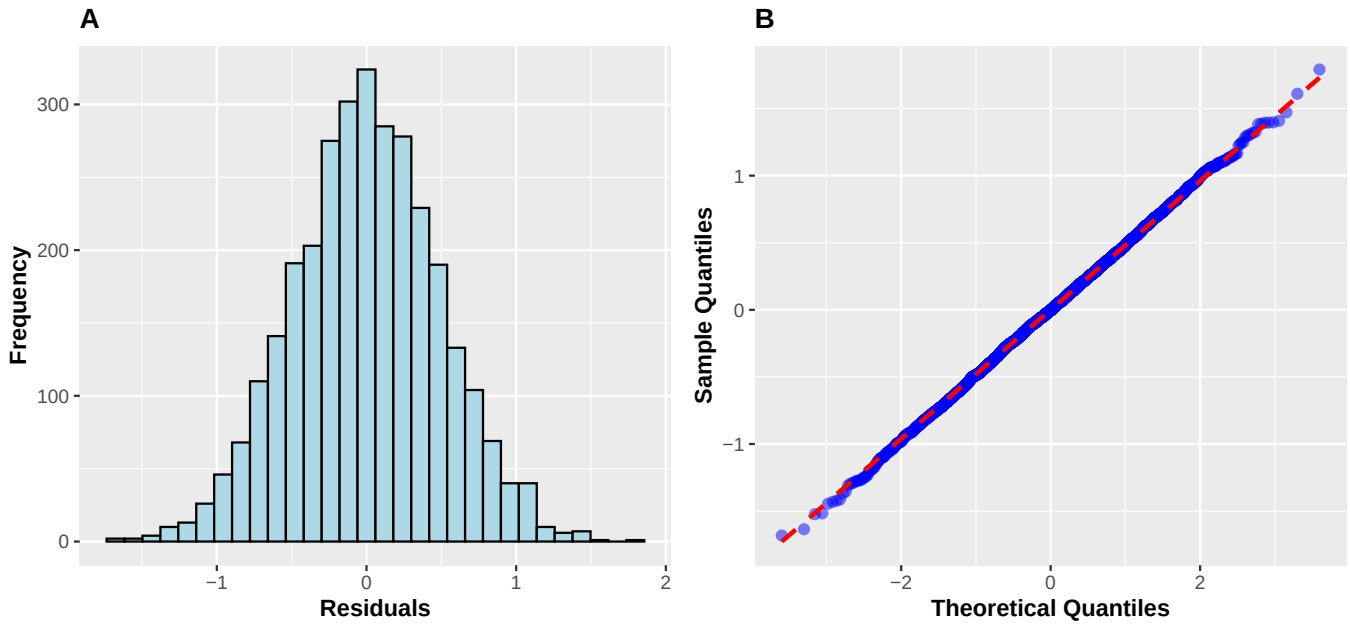
Utilizing both plots together provides a more comprehensive and intuitive examination of the normality assumption, combining the distributional shape (histogram) with a quantile-based assessment (QQ plot).

The following code chunk is for generating Fig. 16:

```
#data frame from fitted values and residuals from model
res_df <- data.frame(fitted = fitted(multi_lr),        #fitted values
                     residuals = resid(multi_lr),      #residuals
                     std_resid = rstandard(multi_lr), #standardised residuals
                     leverage  = hatvalues(multi_lr), #leverage (hat values)
                     cooks = cooks.distance(multi_lr))     #Cook's distance

#use ggplot for the histogram
```

```r
p_hist <- ggplot(data = res_df,
                 aes(x = residuals)) +
  #histogram
  geom_histogram(bins = 30,
                 fill = "lightblue",
                 color = "black") +
  #axes labels
  labs(x = "Residuals", y = "Frequency") +

  #plot subtitle
  ggtitle("A") +

  #appearance
  theme(
    plot.title  = element_text(face = "bold", hjust = 0),
    axis.title.x = element_text(size = 11, face = "bold"),
    axis.title.y = element_text(size = 11, face = "bold")
  )

#QQ plot using ggplot
p_qq <- ggplot(data = res_df,
               aes(sample = residuals)) +

  #add the QQ points layer
  stat_qq(color = alpha("blue", 0.5),
          size = 2,
          shape = 19) +

  #add the QQ line
  stat_qq_line(color = "red",
          linewidth = 1,
          linetype = "dashed") +

  #plot subtitle
  ggtitle("B") +

  #axes labels
  labs(x = "Theoretical Quantiles",
       y = "Sample Quantiles") +

  #appearance
  theme(
    plot.title  = element_text(face = "bold", hjust = 0),
    axis.title.x = element_text(size = 11, face = "bold"),
    axis.title.y = element_text(size = 11, face = "bold")
  )

#merging the plots using patchwork
(p_hist | p_qq) +
  #adding title to merged plot
  plot_annotation(title =
"Fig. 16: Diagnostic Plots for Multiple Regression Residuals: A. Histogram, B. QQ Plot",

    #appearance
    theme = theme(
      plot.title = element_text(face = "bold", hjust = 0.5, size = 10)
    )
  )
```

Fig. 16: Diagnostic Plots for Multiple Regression Residuals: A. Histogram, B. QQ Plot



The main points of observation are as follows:

Panel A: Histogram of Residuals (Fig. 16A)

i. The histogram presents are **nearly symmetric bell-shaped distribution, centred at zero** with the **majority of value is falling between -0.5 and +0.5**. There are **no significant tails, no skewness, and no extreme outliers**.

ii. This observation is corroborated by the ***numerical symmetry of the residuals***, which indicates that the **mean is 0**, signifying that the **average of the model residuals is 0**, a characteristic of a reliable model.

iii. The **median is nearly 0**, **suggesting** that the **distribution is predominantly symmetrical**, while the **interquartile range (comprising 50% of the values) spans between -0.32 and +0.32**, which aligns with the results from the histogram plotting.

iv. The ***minimum and maximum values are fairly well-balanced within a range of -1.68 to 1.79***. This trend shows that the **majority of prediction errors are quite small, within** ± 0.3, with only a limited number of moderately larger discrepancies.

```
summary(resid(multi_lr))
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -1.681535 -0.322248 -0.004236  0.000000  0.325854  1.791076
```

Panel B: QQ Plot of Residuals (Fig. 16B)

i. In the standard QQ plot, the residual points typically align **nearly perfectly along the 45° reference line**, with only **minor deviations at the extreme tails**. This suggests that the residuals are **nearly normally distributed**.

Implications on Model Fit

The nearly ideal normal distribution shape, symmetry, and minimal residual variance suggest a remarkable fit of the model. As evident from Fig. 17A histogram, **the residuals are closely clustered around zero** and have a small magnitude with **maximum of the values falling within** ± 0.5, which is exceptionally small relative to the range of body fat measurements. This indicates a **minimum residual standard error of 0.482** along with an **exceptionally high R-squared value of 0.9961**. Consequently, the model accounts for nearly ***all the variability in the body fat percentage (approximately 99.61%)***, leaving very a little variability unexplained.

There is also no indication of misspecification in the model. A symmetric narrow distribution of the residual indicates that:

1. The line assumption holds true.
2. The model is not significantly affected by outliers.
3. The residuals' distribution is neither skewed or bimodal.
4. The errors of the model approximate a Gaussian distribution.

Since the residuals are **confined within a narrow range of the fitted values (-1.68154 to 1.79108)**, there is **no sign of either increasing or decreasing variance**.

Checking Regression Assumption Using A Four-Panel Summary Diagnostic Plots

A standardized four-panel diagnostic summary for linear regression models (Fig. 17) is also included. These visualizations facilitate a simultaneous evaluation of the fundamental assumptions underlying single linear regression: linearity, homoscedasticity, normality of residuals, and the absence of influential observations.

The produced plots comprise the following:

1. **Residuals vs Fitted Plot:** This visual representation examines linearity and homoscedasticity.
2. **Normal QQ Plot:** This plot is crucial for the accuracy of statistical inferences, which include t-tests, F-tests, confidence intervals, and p-values provided by the model.
3. **Scale-Location Plot:** This visualization assesses whether residual variance remains constant across the fitted values.
4. **Residuals vs Leverage Plot:** This diagnostic plot evaluates whether any observations significantly impact the model.

These plots facilitate concurrent visual evaluation of the fundamental premises of linear regression:

i. Linearity,

ii. Homoscedasticity,

iii. The normal distribution of residuals, and

iv. The lack of influential observations.

The following code chunk is for generating Fig. 17:

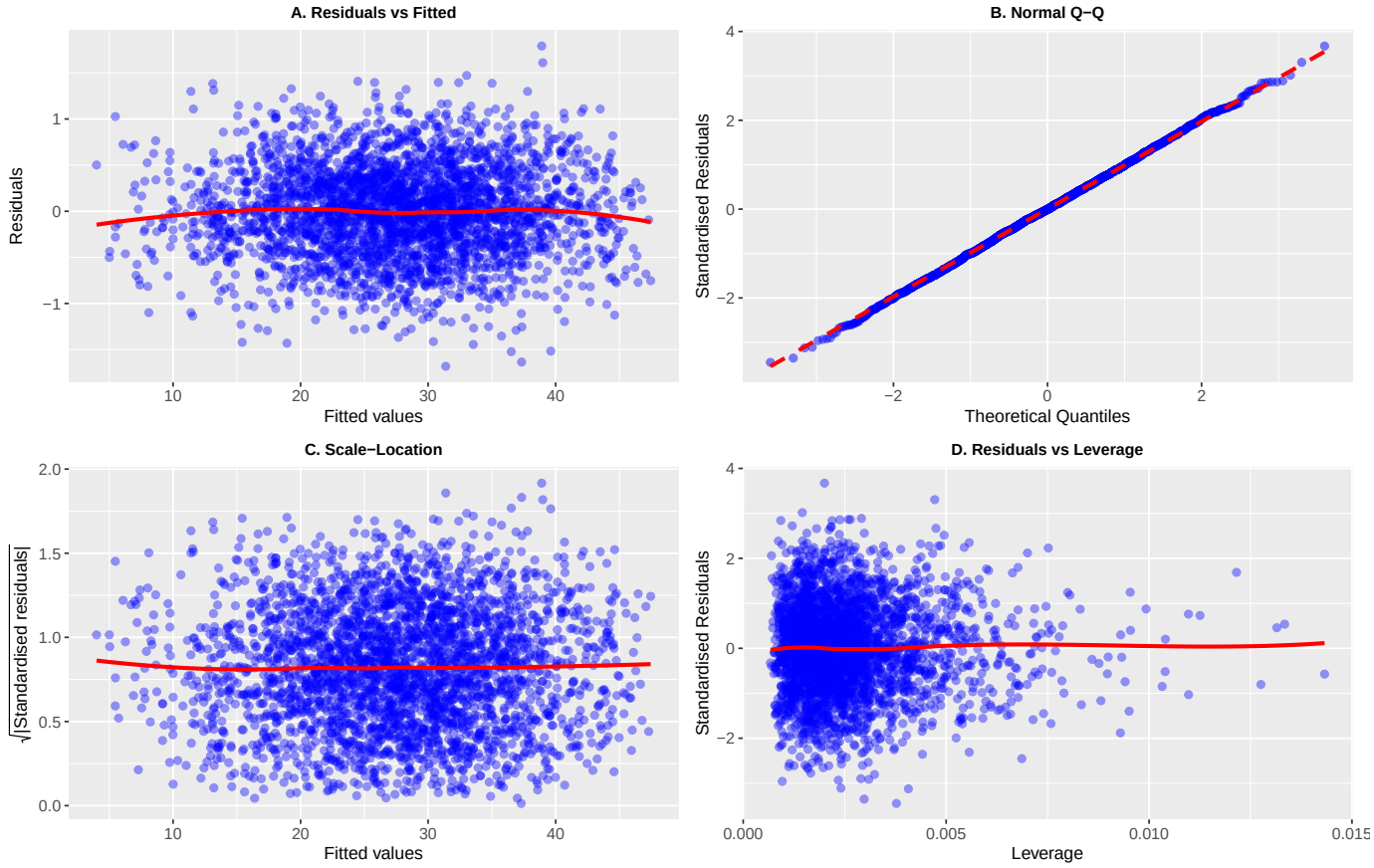
```
#residual vs fitted (A)
p1 <- ggplot(data = res_df,
       aes(x = fitted,
           y = residuals)) +
  #scatter points
  geom_point(colour = "blue",
             alpha = 0.4,
             size = 2) +
  #smooth
  geom_smooth(method = "loess",
              se = FALSE,
              colour = "red",
              linewidth = 1.2) +
  #plot subtitle
  ggtitle("A. Residuals vs Fitted") +
  #plot appearance
    theme(
      plot.title  = element_text(size = 10, face = "bold", hjust = 0.5),
      axis.title.x = element_text(size = 11),
      axis.title.y = element_text(size = 11),
```

```
      axis.text.x  = element_text(size = 10),
      axis.text.y  = element_text(size = 10),
    ) +
    #axis labels
    labs(
      x     = "Fitted values",
      y     = "Residuals"
    )

#mormal QQ (B)
p2 <- ggplot(data = res_df,
               aes(sample = std_resid)) +
  #add the QQ points layer
  stat_qq(color = alpha("blue", 0.5),
          size = 2,
          shape = 19) +
  #add the QQ line
  stat_qq_line(color = "red",
          linewidth = 1.2,
          linetype = "dashed") +
  #plot subtitle
  ggtitle("B. Normal Q-Q") +
  #axes labels
  labs(x = "Theoretical Quantiles",
       y = "Standardised Residuals") +
  #appearance
  theme(
      plot.title  = element_text(size = 10, face = "bold", hjust = 0.5),
      axis.title.x = element_text(size = 11),
      axis.title.y = element_text(size = 11),
      axis.text.x  = element_text(size = 10),
      axis.text.y  = element_text(size = 10),
    )

#scale-location (C)
res_df$sqrt_std_resid <- sqrt(abs(res_df$std_resid))
p3 <- ggplot(data = res_df,
             aes(x = fitted,
                 y = sqrt_std_resid)) +
  #scatter points
  geom_point(colour = "blue",
             alpha = 0.4,
             size = 2) +
  #smooth
  geom_smooth(method = "loess",
              se = FALSE,
              colour = "red",
              linewidth = 1.2) +
  #plot appearance
    theme(
      plot.title  = element_text(size = 10, face = "bold", hjust = 0.5),
      axis.title.x = element_text(size = 11),
      axis.title.y = element_text(size = 11),
      axis.text.x  = element_text(size = 10),
      axis.text.y  = element_text(size = 10),
    ) +
  #plot subtitle
  ggtitle("C. Scale-Location") +
    #axis labels
    labs(
```

```
      x     = "Fitted values",
      y     = expression(sqrt("|Standardised residuals|"))
    )

#residual vs leverage
p4 <- ggplot(data = res_df,
             aes(x = leverage,
                 y = std_resid)) +
  #scatter points
  geom_point(colour = "blue",
             alpha = 0.4,
             size = 2) +
  #smooth
  geom_smooth(method = "loess",
              se = FALSE,
              colour = "red",
              linewidth = 1.2) +
  #plot subtitle
  ggtitle("D. Residuals vs Leverage") +
  #plot appearance
    theme(
      plot.title  = element_text(size = 10, face = "bold", hjust = 0.5),
      axis.title.x = element_text(size = 11),
      axis.title.y = element_text(size = 11),
      axis.text.x  = element_text(size = 10),
      axis.text.y  = element_text(size = 10),
    ) +
    #axis labels
    labs(
      x     = "Leverage",
      y     = "Standardised Residuals"
    )

#merging the plots using patchwork
(p1 | p2) / (p3 | p4) +
  #adding title to merged plot
  plot_annotation(title =
"Fig. 17: Multiple Regression Diagnostic Plots for the Simple Linear Regression Model",

    #appearance
    theme = theme(
      plot.title = element_text(face = "bold", hjust = 0.5, size = 12)
    )
  )
```

```
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
```

Fig. 17: Multiple Regression Diagnostic Plots for the Simple Linear Regression Model



The main points of observation are as follows:

1. Panel A: Residuals vs Fitted Plot (Fig. 17, topleft)

This plot assesses both **linearity and homoscedasticity**. The residuals are ***tightly and evenly distributed around the horizontal zero line***, showing **no evident curvature**. This suggests a **linear the connection between the predictors and body fat percentage**. The ***vertical range of residuals spans from -2 to +2, indicating a smaller spread*** and that the **residuals are significantly reduced and more well-behaved** in the multiple regression model. There is also no pronounced pattern of increasing or decreasing spread, i.e., no funnel shape, which indicates constant variance of the residuals, i.e., homoscedasticity.

2. Panel B: Normal QQ Plot (Fig. 17, topright)

This is crucial for ensuring the ***accuracy of statistical inference***, encompassing **T-tests, F-tests, confidence intervals and p-values generated by the model**. This plot evaluate the normality of the fitted model's residuals. The points lie nearly perfectly **along the 45° reference line**, with only **slight variations at extreme tails**. This suggests that the ***distribution is roughly normal***.

3. Panel C: Scale-Location Plot (Fig. 17, bottomleft)

This plot assesses if the **residual variance remains consistent across the fitted values**, ***utilising standardised residuals***. The standardised residuals display a **scattered yet generally horizontal agreement**. There is also **no significant upward or downward trend observed**. The **minor dispersion at the extremes is anticipated with extensively large data sets** and does not suggest any significant problems. The ***residual's variance remains mostly consistent*** and constant across the fitted values, indicating that ***homoscedasticity is upheld***.

4. Panel D: Residuals vs Leverage Plot (Fig. 17, bottomright)

This plot evaluates if **any data points have an excessive influence on the model**. Although ***several points show elevated standardised residuals or leverage, none surpass the threshold set by Cook's distance***. There are also ***no signs of significantly influential observations that might skew or alter model estimates***. There are also **no concerning outliers or high leverage points present**, indicating that the **model is stable and not influenced by specific cases**.