Best Practices for Clean Code & Code Hygiene

Yiwen Rao, Anna Shchetinina, Hana Lee, Angel Lu, Megan Yu, Clare Huang, Matlin Gilman

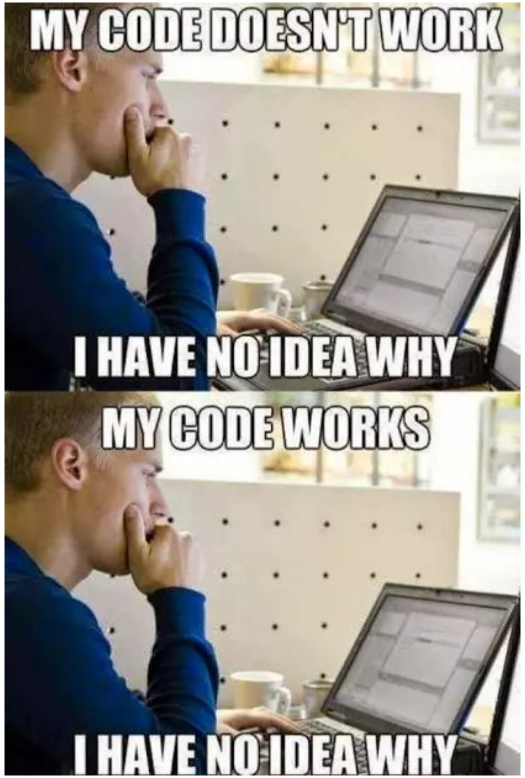
ID529 Winter 2023

Outline

- Why it is important (Hana)
- Naming (Anna)
- Spacing and line breaks (Matlin)
- R script readability (Angel)
- Comments readability (Clare)
- Functions（Yiwen)
- Reproducibility (Megan)

Why is clean code important?

- Easy to understand
- Project transitions
- Better collaboration
- Debugging and understanding what went wrong
- Reproducibility and replicability
- Commonly recommended book:
  Clean Code: A Handbook of Agile Software Craftsmanship by Robert Martin



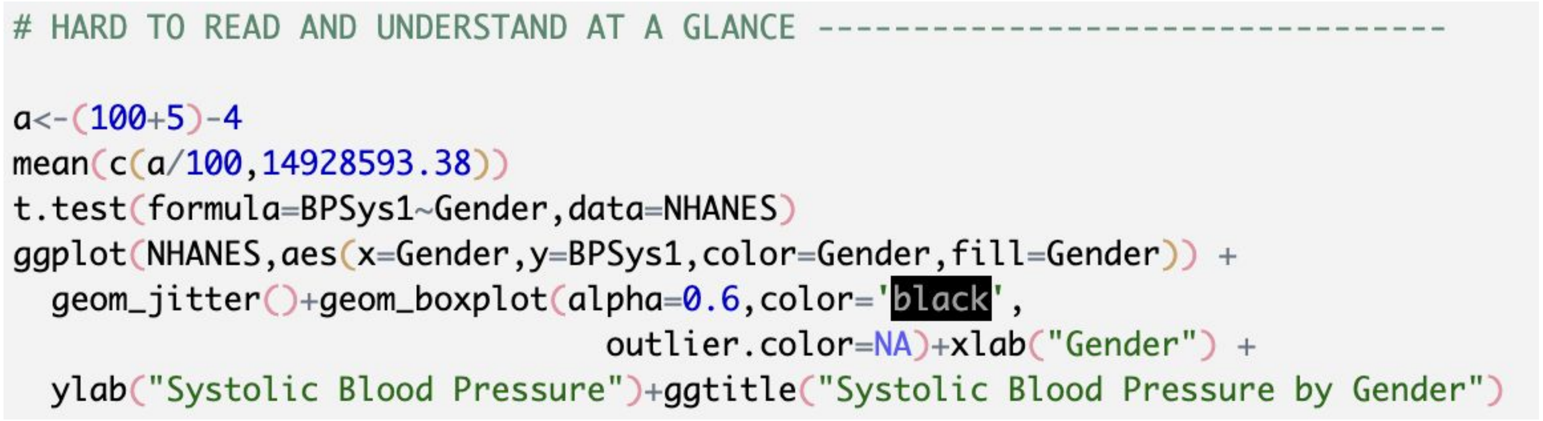
https://uploads.sitepoint.com/wp-content/uploads/2014/12/1418705100why.jpg

Best practices on naming: conventions

Why this is important:	**Conventions**:
- Saves time	- **allowercase** > searchpaths()
- Internal consistency of the code helps to avoid confusion	- **period.separated** > as.numeric(), as.factor(), t.test()
- Allows readers to easily follow the logic and navigate the code	- **underscrore_separated** > seq_along()
Suggestions: ➔ one naming convention for the entire team ➔ variable names - nouns: > list_of_names function names - verbs: > describe() ➔ 😘 **K**eep **I**t **S**imple and **S**traightforward	- **UpperCamelCase** > CreateTableOne() - **lowerCameCase** > colMeans(), suppressPackageStartupMessage()

Best practices on spacing and line breaks

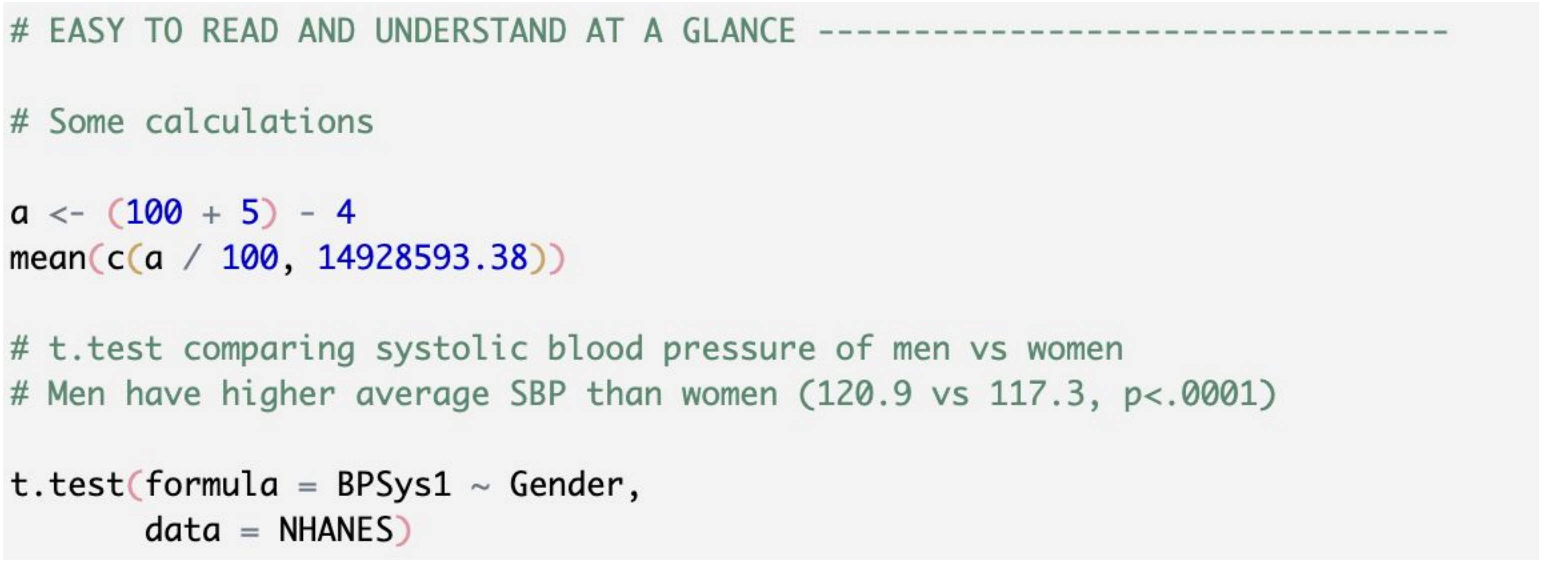
- Inappropriate use of spaces and line breaks



```
# HARD TO READ AND UNDERSTAND AT A GLANCE ----------------------------------

a<-(100+5)-4
mean(c(a/100,14928593.38))
t.test(formula=BPSys1~Gender,data=NHANES)
ggplot(NHANES,aes(x=Gender,y=BPSys1,color=Gender,fill=Gender)) +
  geom_jitter()+geom_boxplot(alpha=0.6,color='black',
                              outlier.color=NA)+xlab("Gender") +
  ylab("Systolic Blood Pressure")+ggtitle("Systolic Blood Pressure by Gender")
```

Best practices on spacing and line breaks (cont’d)

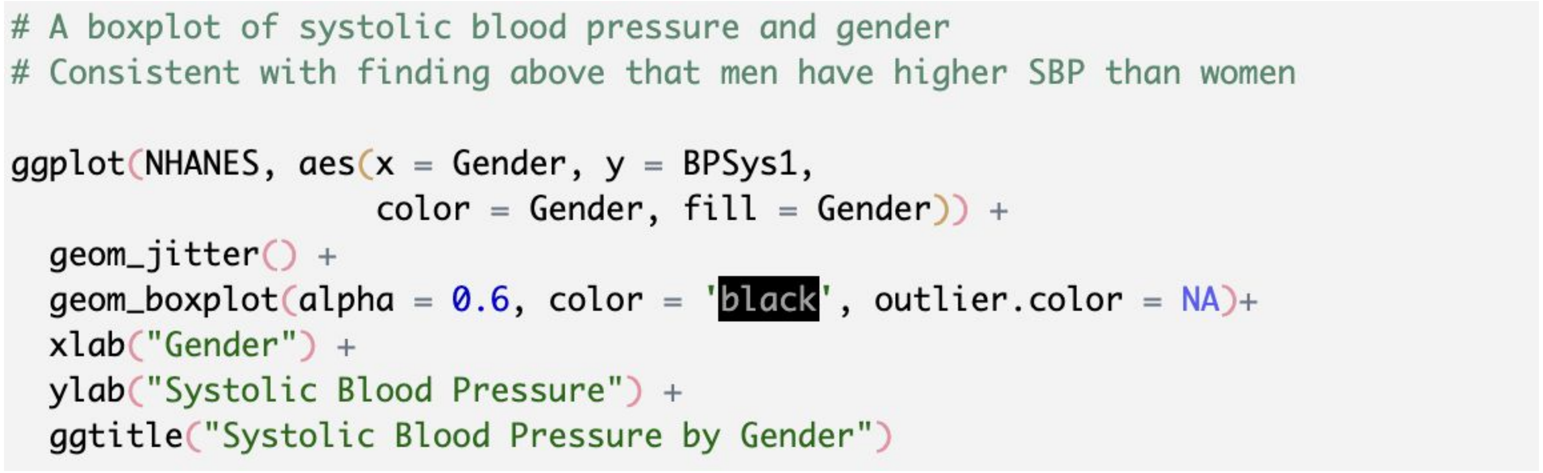
- Appropriate use of spaces and line breaks



```r
# EASY TO READ AND UNDERSTAND AT A GLANCE ---------------------------------

# Some calculations

a <- (100 + 5) - 4
mean(c(a / 100, 14928593.38))

# t.test comparing systolic blood pressure of men vs women
# Men have higher average SBP than women (120.9 vs 117.3, p<.0001)

t.test(formula = BPSys1 ~ Gender,
       data = NHANES)
```

Best practices on spacing and line breaks (cont'd)

- Appropriate use of spaces and line breaks (cont'd)

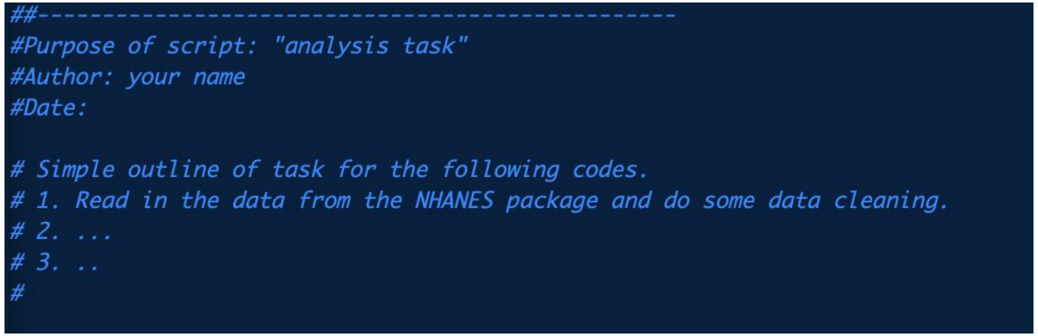


```r
# A boxplot of systolic blood pressure and gender
# Consistent with finding above that men have higher SBP than women

ggplot(NHANES, aes(x = Gender, y = BPSys1,
                   color = Gender, fill = Gender)) +
  geom_jitter() +
  geom_boxplot(alpha = 0.6, color = 'black', outlier.color = NA)+
  xlab("Gender") +
  ylab("Systolic Blood Pressure") +
  ggtitle("Systolic Blood Pressure by Gender")
```

Best practices on readability - R script

A route that specify each step of your workflow

- Clear template **header**
- Simple **outline** of the task
- Load **package** required in the task
- Load the **data**
- Data **preparation**
- Data **Analysis**
- Construct **graph, tables**
- Save the **output**
- External **link or references**



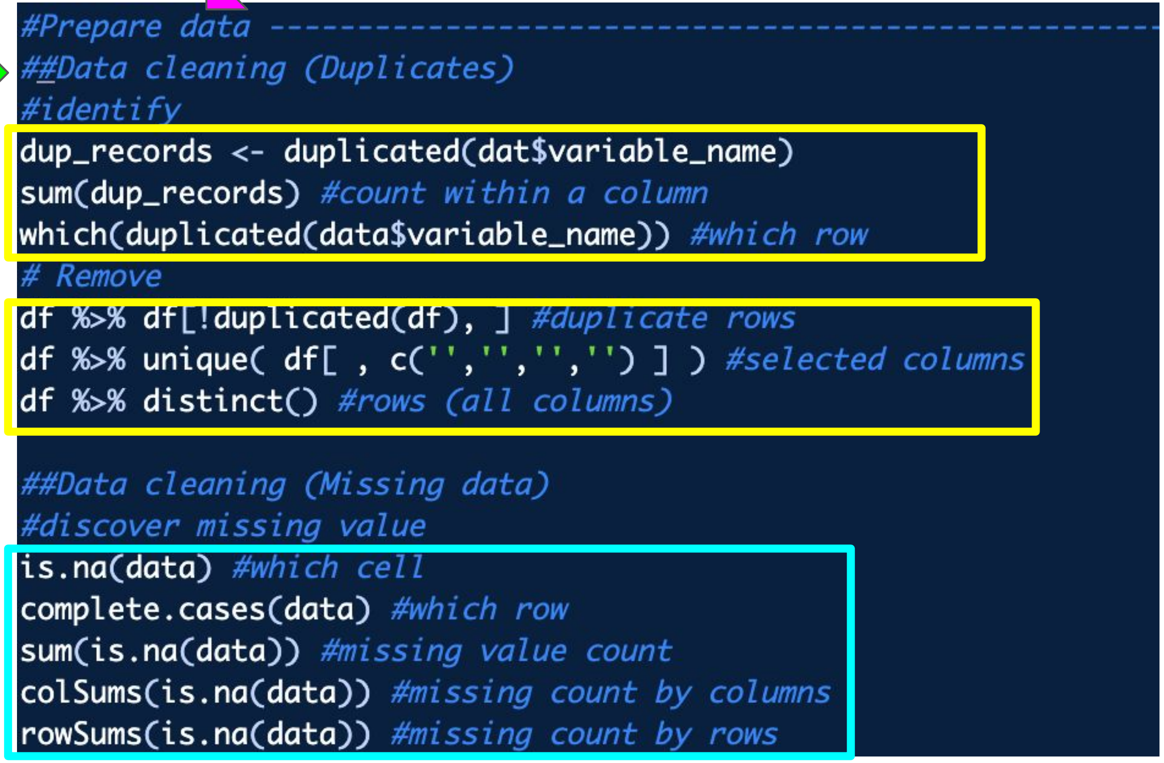
```
##-------------------------------------------------
#Purpose of script: "analysis task"
#Author: your name
#Date:

# Simple outline of task for the following codes.
# 1. Read in the data from the NHANES package and do some data cleaning.
# 2. ...
# 3. ..
#
```

Best practices on readability - R script

Main Purpose

Subgroups of different procedures

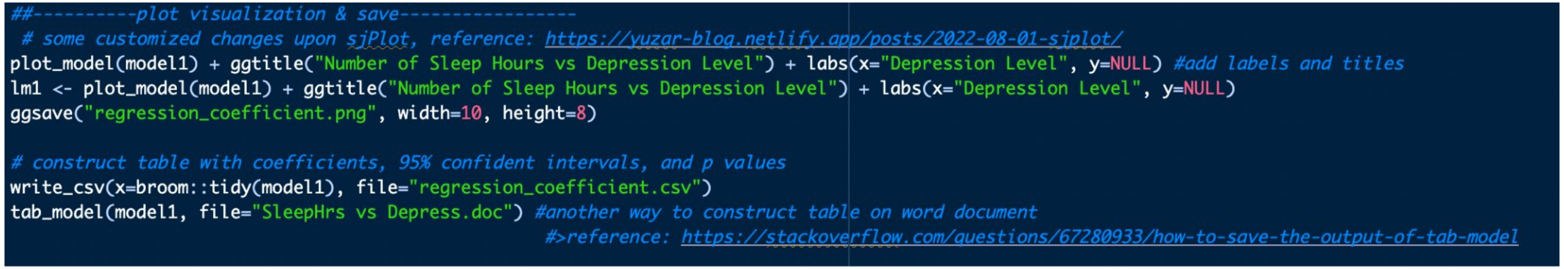
Related commands are grouped together, segregated by annotations or spaces.



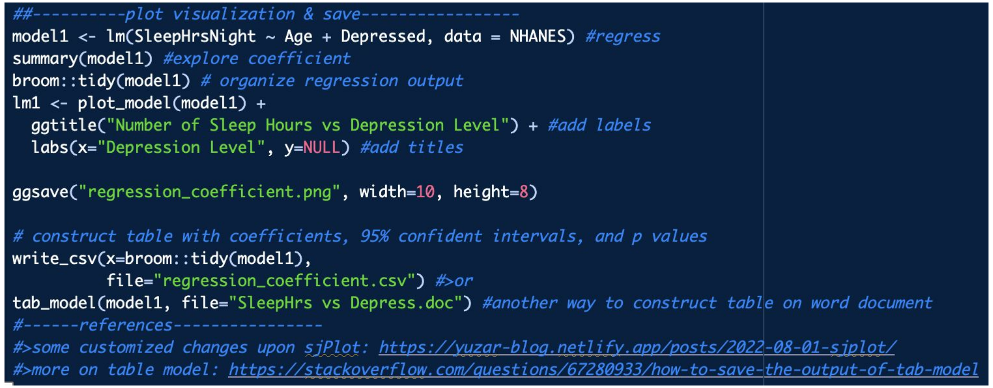
```
#Prepare data ------------------------------------------------
##Data cleaning (Duplicates)
#identify
dup_records <- duplicated(dat$variable_name)
sum(dup_records) #count within a column
which(duplicated(data$variable_name)) #which row
# Remove
df %>% df[!duplicated(df), ] #duplicate rows
df %>% unique( df[ , c('','','','') ] ) #selected columns
df %>% distinct() #rows (all columns)

##Data cleaning (Missing data)
#discover missing value
is.na(data) #which cell
complete.cases(data) #which row
sum(is.na(data)) #missing value count
colSums(is.na(data)) #missing count by columns
rowSums(is.na(data)) #missing count by rows
```

Best practices on readability - R script

- Visually more agreeable with vertical layout



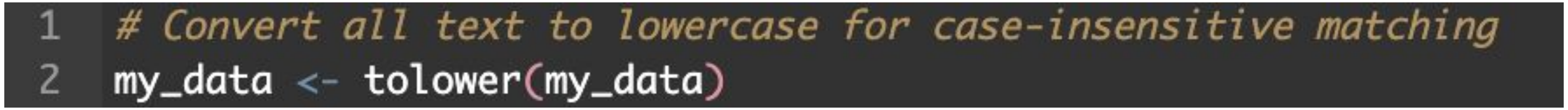
```r
##----------plot visualization & save-----------------
 # some customized changes upon sjPlot, reference: https://yuzar-blog.netlify.app/posts/2022-08-01-sjplot/
plot_model(model1) + ggtitle("Number of Sleep Hours vs Depression Level") + labs(x="Depression Level", y=NULL) #add labels and titles
lm1 <- plot_model(model1) + ggtitle("Number of Sleep Hours vs Depression Level") + labs(x="Depression Level", y=NULL)
ggsave("regression_coefficient.png", width=10, height=8)

# construct table with coefficients, 95% confident intervals, and p values
write_csv(x=broom::tidy(model1), file="regression_coefficient.csv")
tab_model(model1, file="SleepHrs vs Depress.doc") #another way to construct table on word document
                                                   #>reference: https://stackoverflow.com/questions/67280933/how-to-save-the-output-of-tab-model
```



```r
##----------plot visualization & save-----------------
model1 <- lm(SleepHrsNight ~ Age + Depressed, data = NHANES) #regress
summary(model1) #explore coefficient
broom::tidy(model1) # organize regression output
lm1 <- plot_model(model1) +
  ggtitle("Number of Sleep Hours vs Depression Level") + #add labels
  labs(x="Depression Level", y=NULL) #add titles

ggsave("regression_coefficient.png", width=10, height=8)

# construct table with coefficients, 95% confident intervals, and p values
write_csv(x=broom::tidy(model1),
          file="regression_coefficient.csv") #>or
tab_model(model1, file="SleepHrs vs Depress.doc") #another way to construct table on word document
#------references----------------
#>some customized changes upon sjPlot: https://yuzar-blog.netlify.app/posts/2022-08-01-sjplot/
#>more on table model: https://stackoverflow.com/questions/67280933/how-to-save-the-output-of-tab-model
```

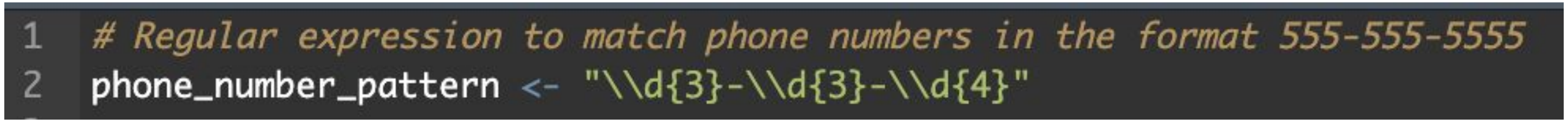
Best practices on readability - Comments

1. Use comments to explain the purpose and logic of the code:



```r
# Convert all text to lowercase for case-insensitive matching
my_data <- tolower(my_data)
```

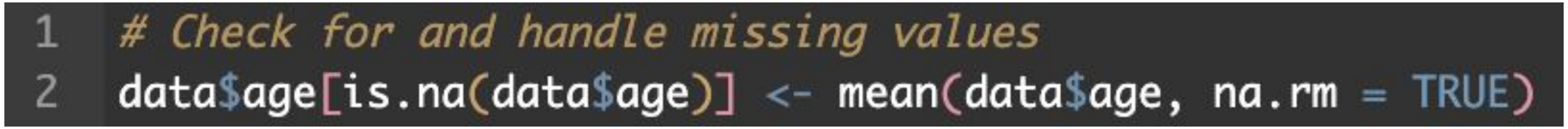
2. Document any non-obvious or complex regular expressions:



```r
# Regular expression to match phone numbers in the format 555-555-5555
phone_number_pattern <- "\\d{3}-\\d{3}-\\d{4}"
```

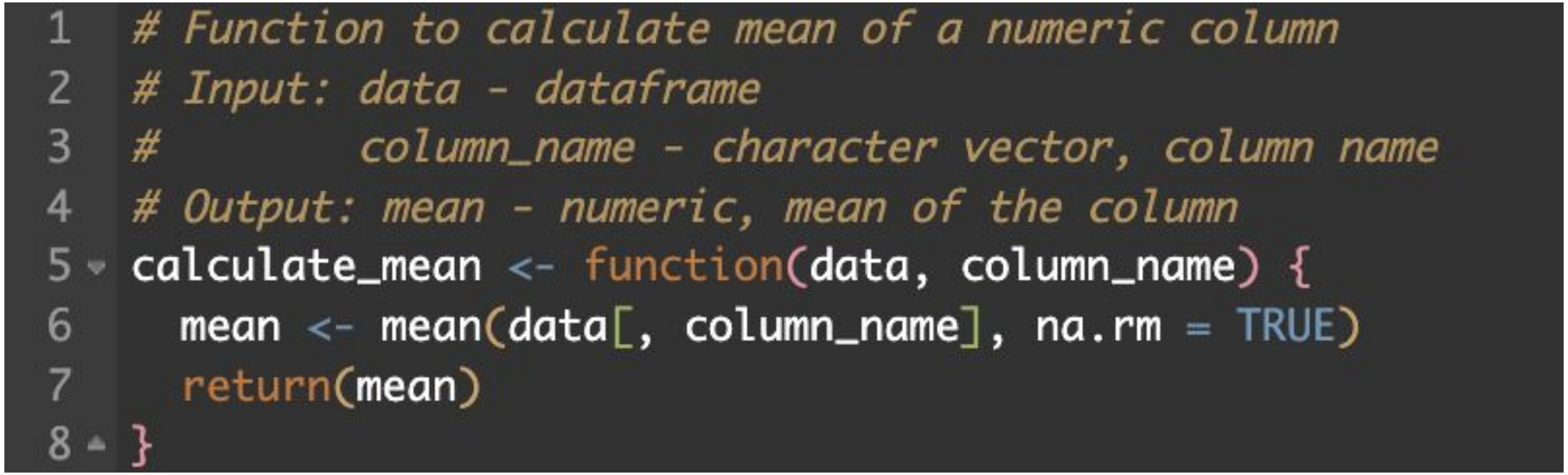
Best practices on readability - Comments

3. Include comments to explain data validation steps:



```r
# Check for and handle missing values
data$age[is.na(data$age)] <- mean(data$age, na.rm = TRUE)
```

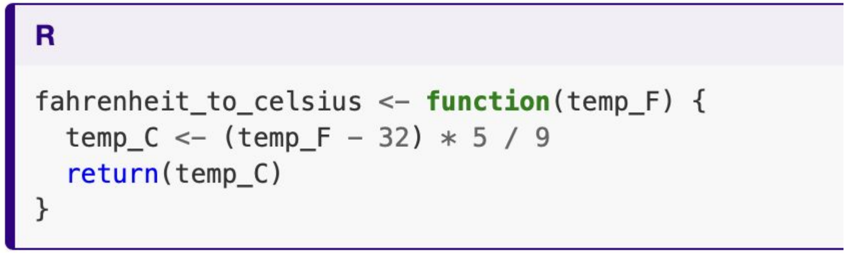
4. Use consistent and up-to-date commenting:



```r
# Function to calculate mean of a numeric column
# Input: data - dataframe
#        column_name - character vector, column name
# Output: mean - numeric, mean of the column
calculate_mean <- function(data, column_name) {
  mean <- mean(data[, column_name], na.rm = TRUE)
  return(mean)
}
```

Best practices on functions

eliminate repetition from our code, and allow **code reuse** and **sharing**



R

```r
fahrenheit_to_celsius <- function(temp_F) {
  temp_C <- (temp_F - 32) * 5 / 9
  return(temp_C)
}
```

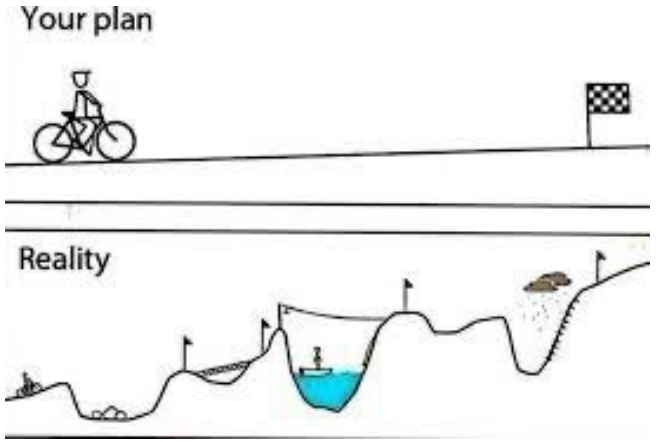
1. **Function name** `fahrenheit_to_celsius`

2. **Input parameter(s)** that the user will feed to the function. `function(temp_F)`

3. **Operation** that you desire to run within curly braces (**{}**).

4. **Result (or output)** of the function in the return statement. `return(temp_C)`

Best practices on reproducibility



- **Avoid assumptions about execution environment**
    - Provide `renv` file
        - Specify R and package versions/dependencies
    - `Here` package or relative paths

```
setwd("~/absolute/file/path/on/your/device")
```

vs.

```
read_csv(here("folder/data_file.csv"))
ggsave(filename = here("folder/figure.png"))
```

- **Use version control (Git)**
- **Document with comments and README files**
- **Have an organized, clear workflow**
    - Write functions, program defensively, comment, test, document
    - Sections and headers that have structure (cleaning, analyzing, plotting)
    - Simplify your code
- **Other packages of interest: targets, rocker**

MY CODE DOESN'T WORK
I HAVE NO IDEA WHY
MY CODE WORKS
I HAVE NO IDEA WHY

Thank you for listening!