Assignment2

Chia-Hao Lo

October 27th 2024

#Setup

```
#Load the required packages
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(inspectdf)
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(moments)
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 1.2.0 --
## v broom        1.0.6     v rsample      1.2.1
## v dials        1.3.0     v tune         1.2.1
## v infer        1.0.7     v workflows    1.1.4
## v modeldata    1.4.0     v workflowsets 1.1.0
## v parsnip      1.2.1     v yardstick    1.3.1
## v recipes      1.1.0
## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x scales::discard()        masks purrr::discard()
## x dplyr::filter()          masks stats::filter()
## x recipes::fixed()         masks stringr::fixed()
## x dplyr::lag()             masks stats::lag()
## x caret::lift()            masks purrr::lift()
## x yardstick::precision()   masks caret::precision()
## x yardstick::recall()      masks caret::recall()
```

```
## x yardstick::sensitivity() masks caret::sensitivity()
## x yardstick::spec()        masks readr::spec()
## x yardstick::specificity() masks caret::specificity()
## x recipes::step()          masks stats::step()
## * Use suppressPackageStartupMessages() to eliminate package startup messages
```

```r
library(modelr)
```

```
##
## Attaching package: 'modelr'
##
## The following objects are masked from 'package:yardstick':
##
##     mae, mape, rmse
##
## The following object is masked from 'package:broom':
##
##     bootstrap
```

Q1. Loading the data

- As a data scientist we need to follow deliverable specifications. So, here we calculate which data set.

```r
# Here we use our ysn to identify which data set we need to use
ysn = 1907385
x <- ysn %% 4
x
```

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

- Here we get 1 from student number modulo 4, then we use division_1.csv to be our data set.

```r
# Read in the data using the correct tidyverse command
div1 <- read_csv("./data/division_1.csv")
```

```
## Rows: 1200 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): PID, LOC, Debug Time
## dbl (1): Bugs REM
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# Display the first 10 lines of the data
head(div1, 10)
```

```
## # A tibble: 10 x 4
##    PID    LOC   `Debug Time`              `Bugs REM`
##    <chr>  <chr> <chr>                          <dbl>
##  1 B56677 23541 176 hours and 50 minutes         214
##  2 B66283 18571 24 hours and 20 minutes          450
##  3 B17818 22577 85 hours and 10 minutes          312
##  4 B72119 19176 109 hours and 40 minutes         203
##  5 B94138 49455 70 hours and 48 minutes         1164
##  6 B12473 21203 68 hours and 27 minutes          314
##  7 B78862 35640 134 hours and 40 minutes         523
```

```
##  8 B93181 20468 105 hours and 10 minutes        237
##  9 B27961 35133 160 hours and 9 minutes         467
## 10 B97185 18061 125 hours and 25 minutes        173
```

- The dimension of the data set gives an idea about the size of the data. We get to know how many variables and data points we are going to work with.

```r
dim(div1)
```

```
## [1] 1200    4
```

- The dim() function here takes 1 argument which is the data set and gives out the dimension of the division_1.csv data set as rows = 1200, columns = 4

Q2. Random permutation of the rows

- Let's set the seed to 1907385, then shuffle all rows without replacement. Finally, output the first 10 rows of the permuted dataset.

```r
# Set the seed
set.seed(ysn)
# Shuffle the dataset rows without replacement and name it permuted_data
div2 <- sample_n(div1, nrow(div1), replace = FALSE)
# Display the first 10 rows of the permuted dataset
head(div2, 10)
```

```
## # A tibble: 10 x 4
##    PID    LOC   `Debug Time`               `Bugs REM`
##    <chr>  <chr> <chr>                           <dbl>
##  1 B71594 39508 56 hours and 2 minutes           1062
##  2 B33434 4074  17 hours and 15 minutes            51
##  3 B32828 37395 210 hours and 41 minutes          414
##  4 B55293 10982 37 hours and 1 minutes            177
##  5 B75585 14239 102 hours and 39 minutes          132
##  6 B87282 34847 24 hours and 34 minutes           904
##  7 B96227 48204 289 hours and 16 minutes          515
##  8 B45883 34050 145 hours and 35 minutes          445
##  9 B41822 7825  31 hours and 7 minutes            116
## 10 B69485 24871 25 hours and 22 minutes           698
```

- Here we show the dimension again after permuting all rows

```r
dim(div2)
```

```
## [1] 1200    4
```

Q3. Adding extra column for row numbers

- Here we add new column called 'ROWS' to the left of the datset, which contains the row numbers.This will help us track the changes made to the data set. Finally, output the first 10 rows of the updated dataset.

```r
# Add a 'ROWS' column to the left with the row numbers
div3 <- div2 %>%
  mutate(ROWS = row_number()) %>%
  relocate("ROWS", .before = PID)
```

```r
# Display the first 10 rows of the dataset with the new 'ROWS' column
head(div3, 10)
```

```
## # A tibble: 10 x 5
##     ROWS PID    LOC   `Debug Time`               `Bugs REM`
##    <int> <chr>  <chr> <chr>                           <dbl>
##  1     1 B71594 39508 56 hours and 2 minutes           1062
##  2     2 B33434 4074  17 hours and 15 minutes            51
##  3     3 B32828 37395 210 hours and 41 minutes          414
##  4     4 B55293 10982 37 hours and 1 minutes            177
##  5     5 B75585 14239 102 hours and 39 minutes          132
##  6     6 B87282 34847 24 hours and 34 minutes           904
##  7     7 B96227 48204 289 hours and 16 minutes          515
##  8     8 B45883 34050 145 hours and 35 minutes          445
##  9     9 B41822 7825  31 hours and 7 minutes            116
## 10    10 B69485 24871 25 hours and 22 minutes           698
```

- Here we add row numbers in front of the rows by using mutate and relocate function.

Q4. Data Cleaning

Q4.1 Negative values removal

```r
# Here we remove the negative values
div3_cleaned <- div3 %>%
  filter(`Bugs REM` >= 0)
```

Q4.2 Impossible data removal

- Here we removed the data that LOC contains impossible value(99999999)

```r
# The value is larger than 99999
div3_cleaned1 <- div3_cleaned %>%
  filter(!(LOC > 99999))
```

Q4.3 Correction of typos

- Here we correct the wrong typo of '32 thousand 3 hundred and 7' in LOC, replace it with '32307'

```r
# The typo is wrong
div3_cleaned2 <- div3_cleaned1 %>%
  mutate(LOC =  str_replace(LOC, "32 thousand 3 hundred and 7", "32307"))
```

- Here we found a wrong type in Debug time, 'fourteen' should be '14'

```r
# Replace the wrong typo in Debug Time
div3_cleaned3 <- div3_cleaned2 %>%
  mutate(`Debug Time` = str_replace(`Debug Time`, "fourteen", "14"))
```

```r
# Display the first 10 rows of the dataset after cleaning the data
head(div3_cleaned3, 10)
```

```
## # A tibble: 10 x 5
##     ROWS PID    LOC   `Debug Time`               `Bugs REM`
##    <int> <chr>  <chr> <chr>                           <dbl>
```

```
## 1       1 B71594 39508 56 hours and 2 minutes         1062
## 2       2 B33434 4074  17 hours and 15 minutes          51
## 3       3 B32828 37395 210 hours and 41 minutes        414
## 4       4 B55293 10982 37 hours and 1 minutes         177
## 5       5 B75585 14239 102 hours and 39 minutes        132
## 6       6 B87282 34847 24 hours and 34 minutes         904
## 7       7 B96227 48204 289 hours and 16 minutes        515
## 8       8 B45883 34050 145 hours and 35 minutes        445
## 9       9 B41822 7825  31 hours and 7 minutes          116
## 10     10 B69485 24871 25 hours and 22 minutes         698
```

- This will display the cleaned dataset after fixing all the issues in the dataset.

```
# Display the dimensions of the dataset after cleaning the data
dim(div3_cleaned3)
```

```
## [1] 1198    5
```

- Here we can see we removed 2 data from the original dataset

Q5. Replace Debug time column

- Here we extract hours and minutes respectively from 'Debug Time' column by str_replace function

```
div3_tidy <- div3_cleaned3 %>%
  mutate(DB_HRS = str_match(`Debug Time`, "(\\d+) hours")[, 2],
         DB_MINS = str_match(`Debug Time`, "(\\d+) minutes")[, 2]
         )
# Here we remove the 'Debug Time' column
div3_tidy1 <- div3_tidy %>%
  select(- `Debug Time`)
# Display the first 10 rows of the updated dataset
head(div3_tidy1, 10)
```

```
## # A tibble: 10 x 6
##      ROWS PID    LOC    `Bugs REM` DB_HRS DB_MINS
##     <int> <chr>  <chr>       <dbl> <chr>  <chr>
##  1      1 B71594 39508        1062 56     2
##  2      2 B33434 4074           51 17     15
##  3      3 B32828 37395         414 210    41
##  4      4 B55293 10982         177 37     1
##  5      5 B75585 14239         132 102    39
##  6      6 B87282 34847         904 24     34
##  7      7 B96227 48204         515 289    16
##  8      8 B45883 34050         445 145    35
##  9      9 B41822 7825          116 31     7
## 10     10 B69485 24871         698 25     22
```

```
# Display the dimensions of the dataset
dim(div3_tidy1)
```

```
## [1] 1198    6
```

Q6. Types of Variables

- **ROWS**:

    - **Categorical Ordinal**
    - This variable represents the position of each row in the dataset. Even though it is numerical, the row numbers have a specific order, which makes it an ordinal variable.
- **PID**: -**Categorical Nominal**
    - The program name has been replace with this identifier, it means this number is the ID number for the data. Each ID number is unique and represents each program.
- **LOC**:
    - **Quantitative Discrete**
    - Represents the numbers of lines of code in a program.
- **Bugs REM**:
    - **Quantitative Discrete**
    - Represents the number of bugs remaining after debugging.
- **DB_HRS**:
    - **Quantitative Discrete**
    - As time can be measured with more precision than just integers, as the description said they round it to the integer and this variable ends up being discretised because it is the unit of hours, not continuous time.
- **DB_MINS**:
    - **Quantitative Discrete**
    - As time can be measured with more precision than just integers, as the description said they round it to the integer and this variable ends up being discretised because it is the unit of minutes, not continuous time.

Q7. Data taming

- From **Module 2, page 3**. Here's what we follow to make sure our data is properly **tamed** based on the provided guidelines:

Q7.1 Naming Conventions for Variable Names

- Ensure all variable names are less than 20 characters.
- Use **snake_case** (lowercase letters with underscores).
- Avoid spaces in variable names.

```
# Rename the columns using rename()
div3_tame <- div3_tidy1 %>%
  rename(
    rows = ROWS,
    pid = PID,
    loc = LOC,
    bugs_rem = `Bugs REM`,
    db_hrs = DB_HRS,
    db_mins = DB_MINS
  )
```

Q7.2 Column Arrangement

- Place the subject identifiers (such as `PID`) in the first column.

```
# Put the column representing the subjects in the first column of a data frame
div3_tame1 <- div3_tame %>%
  relocate("pid", .before = rows)
```

Q7.3 Ordered Factors, Factors, Characters, and Logicals

- Convert **categorical ordinal** variables to ordered factors.
- Convert **nominal categorical** variables to regular factors.
- **Store integers as `<int>`** for memory conservation.

```
# rows should be ordered
div3_tame2 <- div3_tame1 %>%
  mutate(rows = as.ordered(rows))
```

```
# The identify column should be factor
div3_tame3 <- div3_tame2 %>%
  mutate(pid = as.factor(pid))
```

```
#According to Q6, we need to convert loc, bugs_rem, db_hrs, db_mins to integer
div3_tame4 <- div3_tame3 %>%
  mutate(loc = as.integer(loc),
         bugs_rem = as.integer(bugs_rem),
         db_hrs = as.integer(db_hrs),
         db_mins = as.integer(db_mins)
         )
# Display the data after taming
head(div3_tame4, 10)
```

```
## # A tibble: 10 x 6
##    pid    rows    loc bugs_rem db_hrs db_mins
##    <fct>  <ord> <int>    <int>  <int>   <int>
##  1 B71594 1     39508     1062     56       2
##  2 B33434 2      4074       51     17      15
##  3 B32828 3     37395      414    210      41
##  4 B55293 4     10982      177     37       1
##  5 B75585 5     14239      132    102      39
##  6 B87282 6     34847      904     24      34
##  7 B96227 7     48204      515    289      16
##  8 B45883 8     34050      445    145      35
##  9 B41822 9      7825      116     31       7
## 10 B69485 10    24871      698     25      22
```

```
# Display the dimension after taming
dim(div3_tame4)
```

```
## [1] 1198    6
```

Q8. Random subset

- Here we set the seed as ysn for reproducibility, then take a random sample of 700 rows from the 'div3_tame3' dataset.

```
# Set the seed for reproducibility
set.seed(ysn)
# Here we choose a random sample of 700 programs from the dataset
div3_sampled <- div3_tame4 %>%
  sample_n(700)
# Then we order it by rows
div3_sampled <- div3_sampled %>%
  arrange(rows)
# Display the first 10 random sampled dataset
```

```r
head(div3_sampled, 10)
```

```
## # A tibble: 10 x 6
##    pid     rows    loc bugs_rem db_hrs db_mins
##    <fct>   <ord> <int>    <int>  <int>   <int>
##  1 B71594 1     39508     1062     56       2
##  2 B32828 3     37395      414    210      41
##  3 B55293 4     10982      177     37       1
##  4 B75585 5     14239      132    102      39
##  5 B87282 6     34847      904     24      34
##  6 B96227 7     48204      515    289      16
##  7 B45883 8     34050      445    145      35
##  8 B41822 9      7825      116     31       7
##  9 B13933 11    33166      329    217      30
## 10 B34964 12     5722       54     39      48
```

```r
dim(div3_sampled)
```

```
## [1] 700   6
```

Q9.

Q9.(a) Add new columns

- Here we do some calculation and add 'db_totalh', 'db_totalm', 'time_per_loc', 'bugs_per_loc'.

```r
#Here we add 4 new columns into the dataset
div3_new <- div3_sampled %>%
  mutate(
    db_totalh = db_hrs + (db_mins / 60),
    db_totalm = (db_hrs * 60) + db_mins,
    time_per_loc = db_totalm / (loc / 1000),
    bugs_per_loc = bugs_rem / (loc / 1000)
  )
```

```r
#After adding new columns, have to remove the db_hrs and db_mins
div3_new1 <- div3_new %>%
  select(- db_hrs, - db_mins)
# Display the new dataset
head(div3_new1,10)
```

```
## # A tibble: 10 x 8
##    pid     rows    loc bugs_rem db_totalh db_totalm time_per_loc bugs_per_loc
##    <fct>   <ord> <int>    <int>     <dbl>     <dbl>        <dbl>        <dbl>
##  1 B71594 1     39508     1062      56.0      3362         85.1        26.9
##  2 B32828 3     37395      414     211.      12641        338.         11.1
##  3 B55293 4     10982      177      37.0      2221        202.         16.1
##  4 B75585 5     14239      132     103.       6159        433.          9.27
##  5 B87282 6     34847      904      24.6      1474         42.3        25.9
##  6 B96227 7     48204      515     289.      17356        360.         10.7
##  7 B45883 8     34050      445     146.       8735        257.         13.1
##  8 B41822 9      7825      116      31.1      1867        239.         14.8
##  9 B13933 11    33166      329     218.      13050        393.          9.92
## 10 B34964 12     5722       54      39.8      2388        417.          9.44
```

```r
dim(div3_new1)
```

```
## [1] 700   8
```

Q9.(b) Data Type

- **db_totalh**:
  - **Quantitative Continuous**
  - Represents the total number of hours spent debugging, which can be measured on a continuous scale.
- **db_totalm**:
  - **Quantitative Discrete**
  - Represents the total number of minutes spent debugging, which also can be measured on a continuous scale. However, db_totalm represents it as discrete by measuring it in integer minute.
- **time_per_loc**:
  - **Quantitative Continuous**
  - Represents the total time in minutes spent per 1000 lines of code, which is a calculated ratio and can take any real number values.
- **bugs_per_loc**:
  - **Quantitative Continuous**
  - Represents the number of bugs per 1000 lines of code, although bugs could be considered as discrete, the variable bugs_per_loc is ratio over 1000 of code, it means it can take non-integer values, so it is continuous.
- Here we already check 4 columns maintain in the right data type, so we don't need to change it again.

Q10.

Q10.(a) Display the summary statistics for the numerical values

```r
div3_summary_stats <- inspect_num(div3_new1)
div3_summary_stats
```

```
## # A tibble: 6 x 10
##   col_name        min     q1 median   mean     q3    max     sd pcnt_na hist
##   <chr>         <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>   <dbl> <named >
## 1 loc          1.23e3 1.52e4 2.55e4 2.57e4 3.59e4 4.99e4 1.28e4       0 <tibble>
## 2 bugs_rem     1.4 e1 2.04e2 3.67e2 4.33e2 5.47e2 3.09e3 3.44e2       0 <tibble>
## 3 db_totalh    3.3 e0 3.97e1 8.52e1 1.05e2 1.55e2 3.86e2 7.96e1       0 <tibble>
## 4 db_totalm    1.98e2 2.38e3 5.11e3 6.28e3 9.30e3 2.31e4 4.78e3       0 <tibble>
## 5 time_per_loc 3.06e1 1.37e2 2.47e2 2.48e2 3.56e2 4.80e2 1.27e2       0 <tibble>
## 6 bugs_per_loc 8.63e0 1.08e1 1.37e1 1.65e1 1.93e1 7.27e1 8.21e0       0 <tibble>
```

Q10.(b)

i. Median debugging time(per 1000 lines of code)

```r
# Here we calculate the median again to check if it is fitting above answer
median_time_per_loc <- median(div3_new1$time_per_loc, na.rm = TRUE)
median_time_per_loc <- round(median_time_per_loc, 2)
median_time_per_loc
```

```
## [1] 246.72
```

ii. The IQR of the number of remaining bugs(per 1000 lines of code)

```
# We use IQR() function to do the calculation and round to 2 decimal places
iqr_bugs_per_loc <- IQR(div3_new1$bugs_per_loc, na.rm = TRUE)
iqr_bugs_per_loc <- round(iqr_bugs_per_loc, 3)
iqr_bugs_per_loc
```

```
## [1] 8.523
```

- We know IQR is q3 - q1 = 19.28577 - 10.76313 = 8.52264 from (a)

iii. The program ID and number of lines of code for the longest program

```
# We get the max loc then filter the pid
longest_program <- div3_new1 %>%
  filter(loc == max(loc, na.rm = TRUE)) %>%
  select(pid, loc)
```

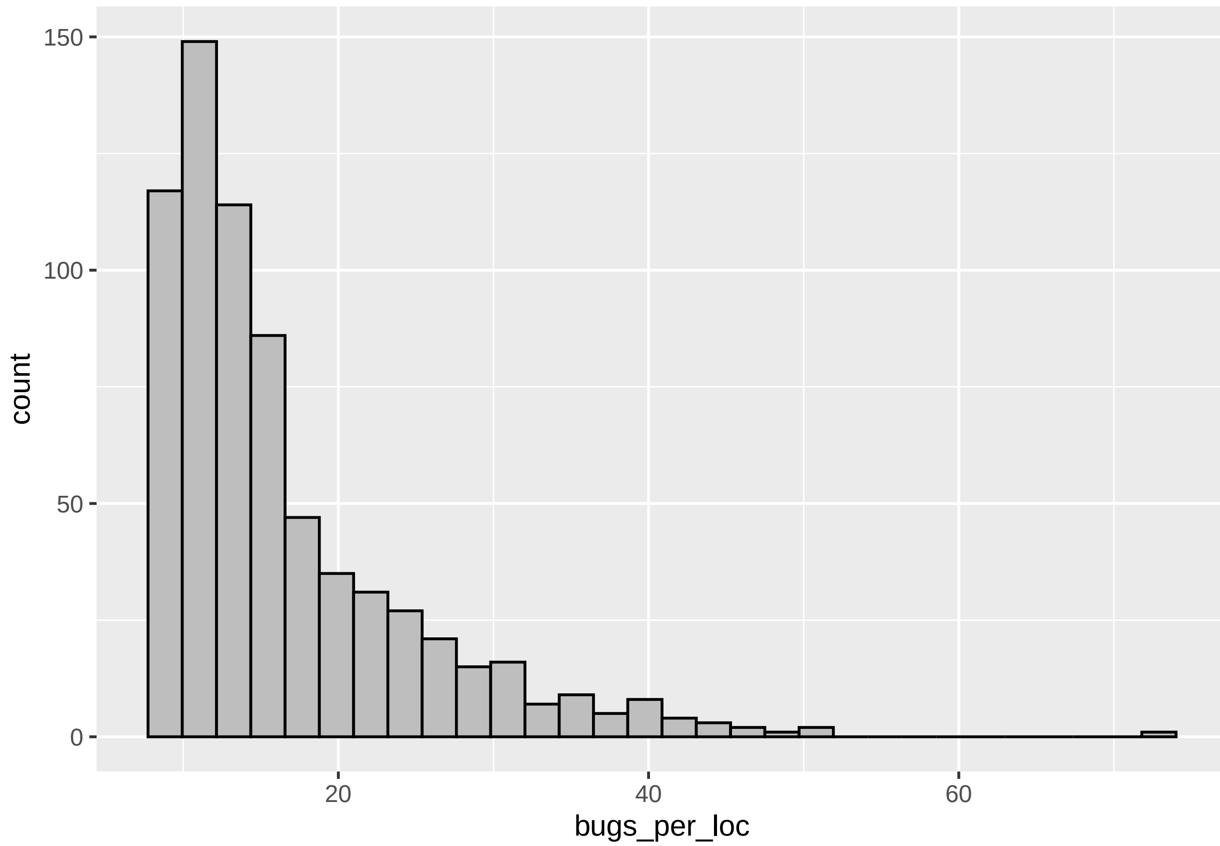
```
# Longest program line in the program
longest_program
```

```
## # A tibble: 1 x 2
##   pid      loc
##   <fct>  <int>
## 1 B61355 49888
```

Q11. Plot the histogram and find the skewness

```
# Here we plot the bugs_per_loc histogram
ggplot(div3_new1, aes(bugs_per_loc)) +
  geom_histogram(fill = 'grey', col = 'black')
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Here we use skewness() to get the skewed value
skew_bugs_per_loc <- moments:: skewness(div3_new1$bugs_per_loc, na.rm = TRUE)
skew_bugs_per_loc
```

```
## [1] 1.910257
```

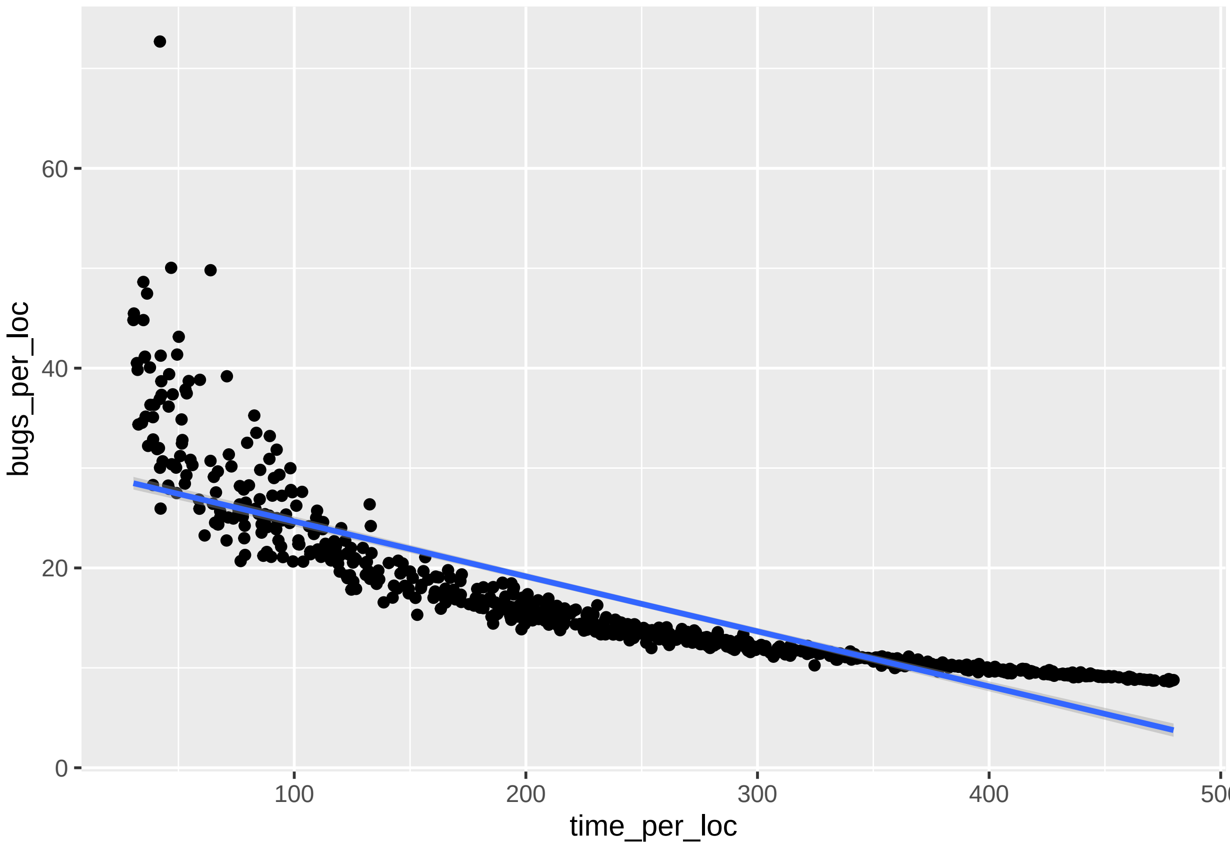
- From the histogram we can see that it is right-skewed and it is unimodal, main values concentrated towards the left side

Q12.

Q12.(a) Scatterplot

```
ggplot(div3_new1, aes(x = time_per_loc, y = bugs_per_loc)) +
  geom_point() +
  geom_smooth(method = 'lm')
```

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



* time_per_loc is the explanatory variable because it is the amount of time spent debugging and it affects the number of bugs remaining

Q12.(b) Linear relationship

- Although the line of best fit shows a general negative trend, the curved shape of the data points means a non-linear relationship.
- The data points scattered more widely at low time_per_loc values, when they become more tightly as time_per_loc increases.
- These patterns show that a simple linear model may not capture the complexity of the relationship well.

Q13.

Q13.(a) Box-cox

- Here we display the BoxCox function, and the range of lambda is between -5 to 5 with 0.1 in each step

```
div3_bc <- BoxCoxTrans(y = div3_new1$bugs_per_loc, x = div3_new1$time_per_loc, , lambda = seq(-5, 5, by
div3_bc
```

```
## Box-Cox Transformation
##
## 700 data points used to estimate Lambda
##
## Input data summary:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    8.63   10.76   13.66   16.50   19.29   72.69
##
```

```
## Largest/Smallest: 8.42
## Sample Skewness: 1.91
##
## Estimated Lambda: -1.4
```

- From the result, we can see that estimated lambda is -1.4 and it is in the range of the search

Q13.(b) tf_bugs column

- Here we use lambda to apply the transformation and add it to the right of the dataset

```
# First we set the lambda value from estimated lambda
lambda_est <- -1.4

# Here we transformed the data and add it into tf_bugs
div3_new1$tf_bugs <- predict(div3_bc, div3_new1$bugs_per_loc)

# Display the first 10 rows after adding tf_bugs column
head(div3_new1, 10)
```

```
## # A tibble: 10 x 9
##    pid     rows    loc bugs_rem db_totalh db_totalm time_per_loc bugs_per_loc
##    <fct>   <ord> <int>    <int>     <dbl>     <dbl>        <dbl>        <dbl>
##  1 B71594 1     39508     1062      56.0      3362         85.1         26.9
##  2 B32828 3     37395      414     211.      12641        338.          11.1
##  3 B55293 4     10982      177      37.0      2221        202.          16.1
##  4 B75585 5     14239      132     103.       6159        433.           9.27
##  5 B87282 6     34847      904      24.6      1474         42.3         25.9
##  6 B96227 7     48204      515     289.      17356        360.          10.7
##  7 B45883 8     34050      445     146.       8735        257.          13.1
##  8 B41822 9      7825      116      31.1      1867        239.          14.8
##  9 B13933 11    33166      329     218.      13050        393.           9.92
## 10 B34964 12     5722       54      39.8      2388        417.           9.44
## # i 1 more variable: tf_bugs <dbl>
```

```
# Display the dimension with after adding tf_bugs column
dim(div3_new1)
```

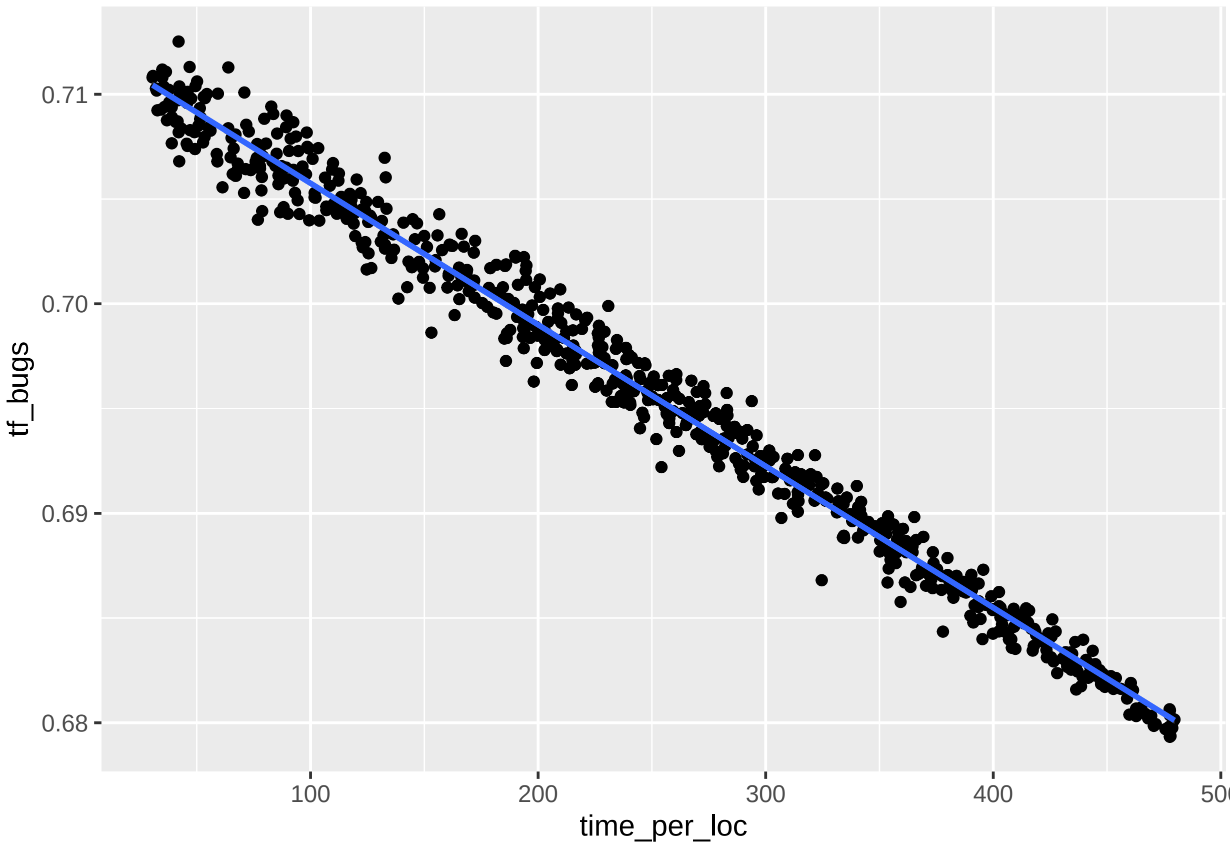
```
## [1] 700   9
```

Q14. Produce a scatterplot of the Box-Cox transformed data

- Here we perform the scatterplot with a line of best fit

```
ggplot(div3_new1, aes(x = time_per_loc,y = tf_bugs)) +
  geom_point() +
  geom_smooth(method = 'lm')
```

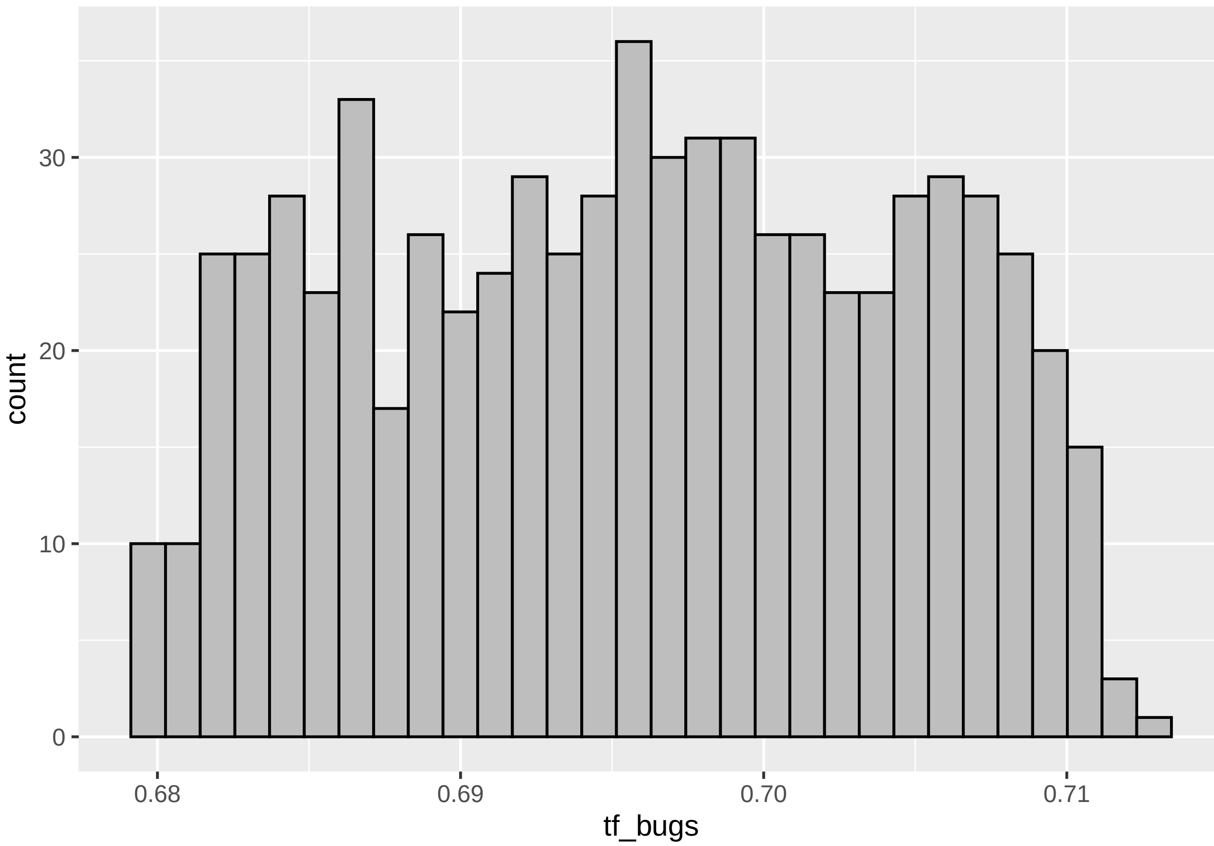
```
## `geom_smooth()` using formula = 'y ~ x'
```



- Here we show the histogram for the transformation data

```
ggplot(div3_new1, aes(tf_bugs)) +
  geom_histogram(fill = 'grey', col = 'black')
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



- This is the corresponding skewness

```
skew_tf_bugs <- moments::skewness(div3_new1$tf_bugs, na.rm = TRUE)
skew_tf_bugs
```

```
## [1] -0.03930153
```

- As we can see from the scatterplots between untransformed and transformed data, after transformed we got a linear trend, with the line of best fit fitting well, shows that the transformation made the relationship more suitable for linear modeling.
- The transformed histogram is more symmetrical and less skewed, means the transformation imporved the normality of data.

Q15. Predict

Q15.(a) General equation

- Below shows the general equation:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \,, i = 1, 2..., n, \; \epsilon_i \sim N(0, \sigma^2)$$

- In our case:
  - y_i: Is the tf_bugs which transformed i bugs per 1000 LOC (response variable)
  - x_i: Is time_per_loc, which is i time spent debugging per 1000 lines of code (predicted variable)
  - β_0: True intercept of the population model.
  - β_1: True slope of the model, representing the effect of time_per_loc on tf_bugs.
  - ϵ_i: Error term representing the difference between the observed value and the predicted value.
- Thus, below is our model:

$$\hat{y_i} = \hat{\beta}_0 + \hat{\beta}_1 x_i + \epsilon_i \,, where \, \epsilon_i \sim N(0, \sigma^2)$$

Q15.(b) Linear model

```r
# Here we use lm() to fit the linear model
div3_model <- lm(tf_bugs ~ time_per_loc, data = div3_new1)
# summary() provided the details, here we can see the estimated coefficients for the model
summary(div3_model)
```

```
##
## Call:
## lm(formula = tf_bugs ~ time_per_loc, data = div3_new1)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0037797 -0.0006463 -0.0000260  0.0006615  0.0034148
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   7.125e-01  8.804e-05  8093.0   <2e-16 ***
## time_per_loc -6.753e-05  3.158e-07  -213.8   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.00106 on 698 degrees of freedom
## Multiple R-squared:  0.985,  Adjusted R-squared:  0.9849
## F-statistic: 4.572e+04 on 1 and 698 DF,  p-value: < 2.2e-16
```

- The summary output provides the estimated coefficients for the intercept and the slope.
- The correct formula with coefficients:

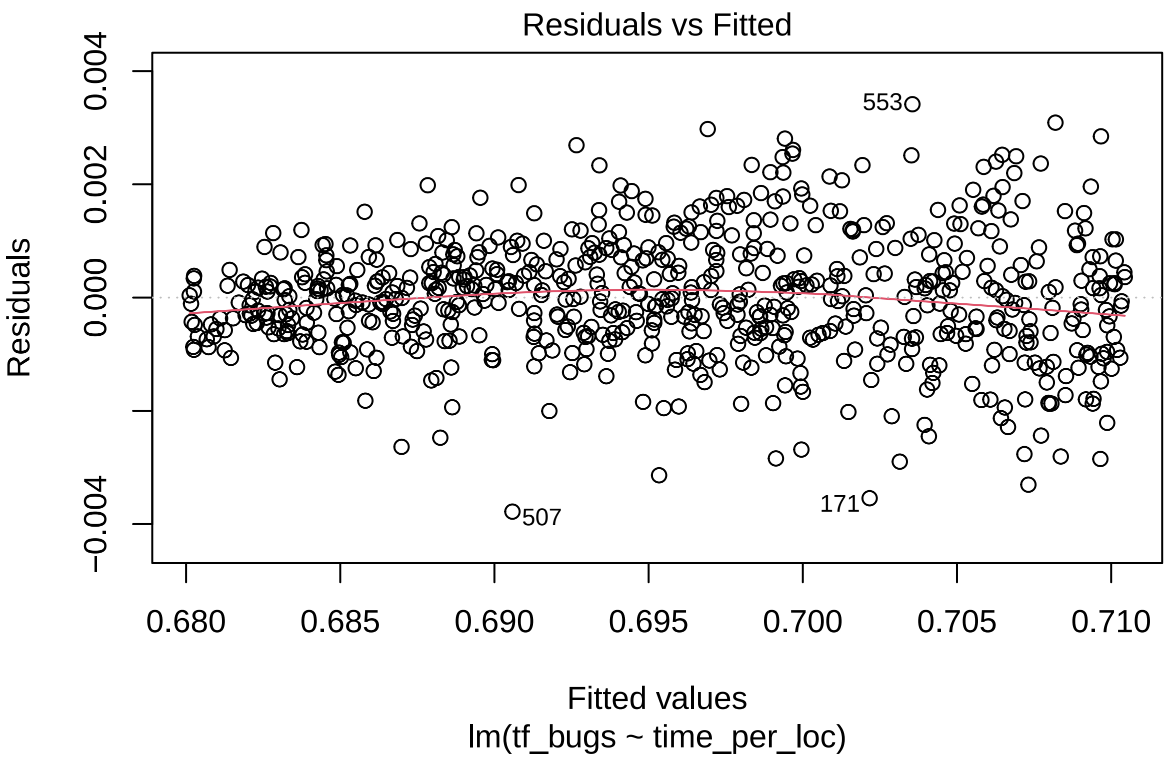
$$y_i = -0.00178 - 0.00006753x_i + 0.000011236$$

  - y_i: Is the tf_bugs which transformed i bugs per 1000 LOC (response variable)
  - x_i: Is time_per_loc, which is i time spent debugging per 1000 lines of code (predicted variable)
  - Intercept(β_0): -0.00178
  - Slope(β_1): -0.00006753
  - Error term(ϵ_i): (0.00106)^2 = 0.0000011236 (It is the square of the residual standard error, which is 0.00106)

Q16. Linear model assumption

Linearity: The relationship between the predictor(time_per_loc) and the response(tf_bugs) is linear. We can assume that tf_bugs changes proportionally with time_per_loc.

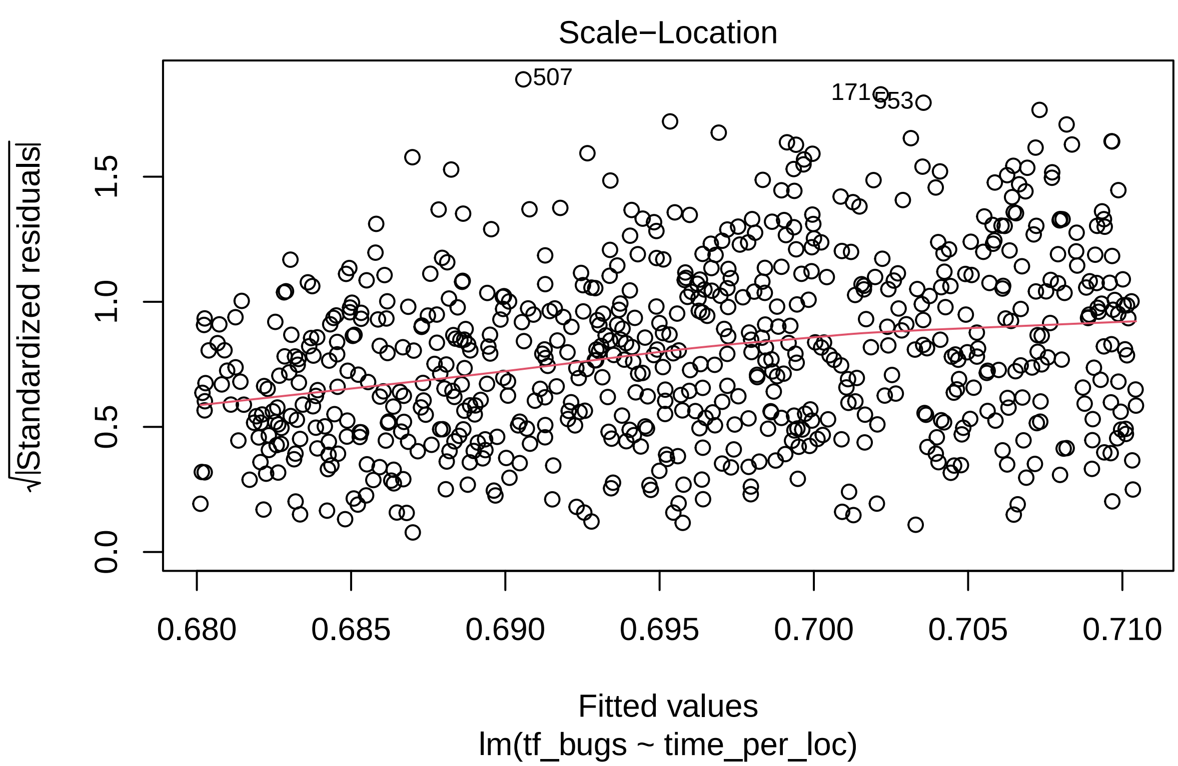
```r
plot(div3_model, which = 1)
```



- From the plot we can see that the red line is roughly straight, which is a good sign. There is no trend in the residuals is an indication that the assumption are satisfied.

Homoscedasticity (Constant Variance of Errors): The spread of the error term(variance) should be the same for all values of the predictor(time_per_loc).

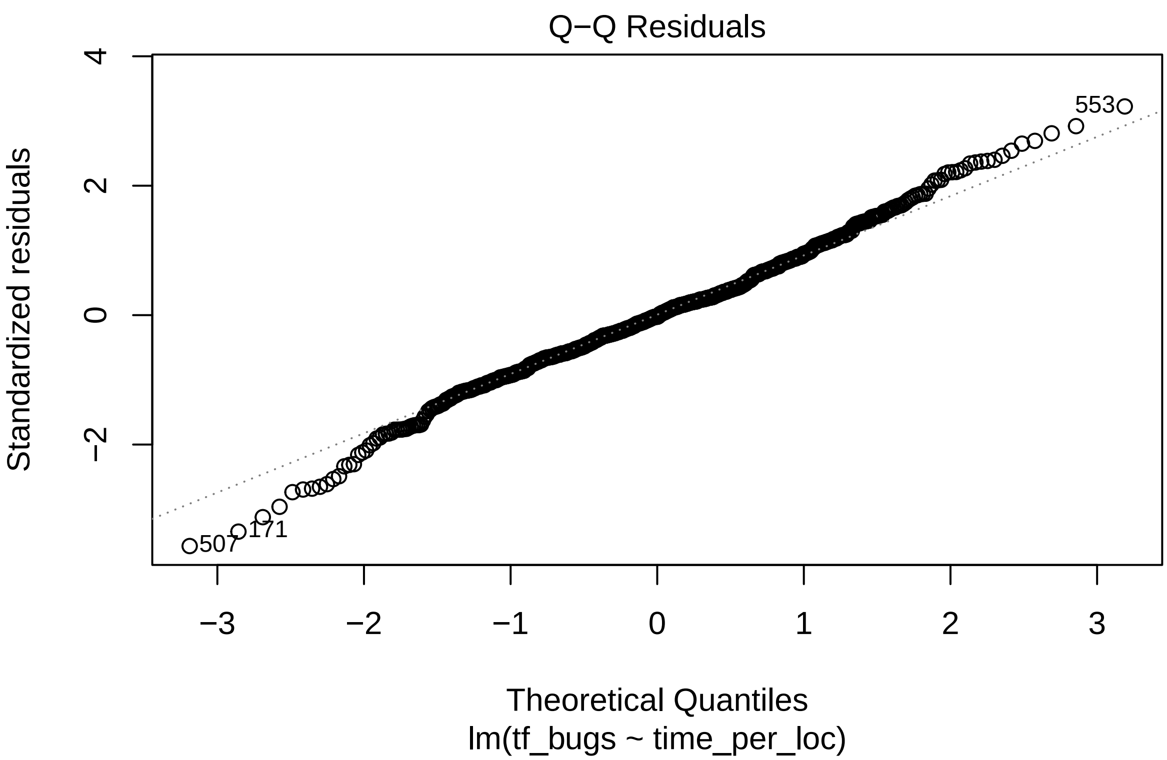
```
plot(div3_model, which = 3)
```



- We can observe an even spread of the points in the vertical direction from left to right. There are no obvious trends make the plot been suspicious, shows the model has constant spread. The red line helps us to see the trend that it is roughly straight and flat, means this model fits the assumption.

Normality: This assumption means that the errors(residuals) are normally distributed.

```
plot(div3_model, which = 2)
```



- As we can see, this is the normal QQ plot of the residuals. From -2 to +2, the residuals distributed mostly lie on the line, means the model satisfied with the assumption.

Independence: It means that the residuals are independent of each other. In other words, the value of the residual for one observation should not be related to any other residual. However, this may not hold in this context. For example, if one program has many bugs and takes a long time to debug, it may influence how efficiently the next program is debugged. This dependency could cause a pattern in the residuals. Due to this reason, the model may overestimate the reliability of its predictions.

Q17.

Q17.(a) Mean number of bugs remaining after the median debugging time

```
# First we calculate the median debugging time
med_time_per_loc <- median(div3_new1$time_per_loc)
med_time_per_loc
```

```
## [1] 246.7218
```

```
# Second, we need to use predict() to get the predicted number of buggs remaining for the median debugg
div3_new_data <- tibble(
  time_per_loc = med_time_per_loc
)
# Company wants the intervals at the 98% level
```

```r
div3_predict <- predict(div3_model, div3_new_data, interval = "confidence", level = 0.98)
div3_predict
```

```
##         fit       lwr       upr
## 1 0.6958452 0.6957518 0.6959387
```

```r
# Final, we need to transform the prediction back to original scale, we knew lambda is -1.4 from Q13(b)
div3_ori_pred <- ((div3_predict * lambda_est) + 1) ^ (1 /lambda_est)
div3_ori_pred
```

```
##        fit      lwr      upr
## 1 13.62563 13.57652 13.67517
```

- Predicted mean number of bugs remaining (original scale) is 0.9819583 in 98% confidence interval.

Q17.(b) The number of bugs remaining for the company's new software project

- As we know from our company, the new software expected to consist 80,000 lines of code, it means it is 80 times larger from original software.

```r
# 28 days of 24 hours and convert it to minutes, with 80,000 LOC adjusted to units of 1,000 LOC
new_software_time_per_loc <- (28 * 24 * 60) / (80000 / 1000)
# Here we set the new time_per_loc to 80
div3_new_software <- tibble(
  time_per_loc = new_software_time_per_loc
)
# Company wants the intervals at the 98% level
div3_predict1 <- predict(div3_model, div3_new_software, interval = "confidence", level = 0.98)
div3_predict1
```

```
##         fit       lwr       upr
## 1 0.6784718 0.6782615 0.6786821
```

```r
# Here we transform our predictions and intervals back to the scale of the original variables
div3_new_pred <- ((div3_predict1 * lambda_est) + 1) ^ (1 /lambda_est)
div3_new_pred
```

```
##        fit      lwr      upr
## 1 8.480918 8.445531 8.516663
```

- Predicted number of bugs remaining for the new software project (original scale) in 98% confidence interval is between 0.9927267 to 0.9930317

To determine the total number of bugs remaining will be within acceptable limits, which is within 3000 bugs. Let's check with it:

$$Total\,Bugs = Predicted\,Value\,per\,1000\,LOC \cdot (80,000/1,000)$$

- which 80,000 line is the new software porject and 1,000 is 1,000 per LOC

```r
# Predicted value for 80,000 LOC
div3_new_pred[, "fit"] * (80000 / 1000)
```

```
## [1] 678.4734
```

```r
# Upper bound for 80,000 LOC
div3_new_pred[, "upr"] * (80000 / 1000)
```

```
## [1] 681.3331
```

```r
# Lower bound for 80,000 LOC
div3_new_pred[, "lwr"] * (80000 / 1000)
```

```
## [1] 675.6425
```

- Here we can confirmed that the total number of bugs is no more than 3,000.

Q18. Report

- The company's debugging process has been analyzed using a linear regression model to predict the number of bugs remaining based on the time spent on debugging per 1000 lines of code. The results shows that there is a significant negative relationship between debugging time (time_per_loc) and the number of bugs remaining (bugs_per_loc). It means more time spent debugging leads to fewer bugs left in the system.
- For the median debugging time, the predicted number of bugs remaining was 8.48 bugs per 1000 lines of code within the 98% confidence interval. When this result is scaled to a new software project with 80,000 lines of code, the total estimated number of bugs remaining is about 648.47, which is significantly lower than the company's regulation of 3,000 bugs. However, the residuals are not entirely independent, which could meant some unconsidered factors influencing the number of bugs. This dependency may cause us overestimated the reliability of the model.
- Although the model predicts that the number of bugs will be well below 3,000, it is possible that the confidence intervals may not fully account for all factors. The predicted interval for the number of bugs is very tight, means we might overconfidence in the estimates. If there are some complexities or challenges in the debugging process, the actual number of bugs could be higher than predicted.