p8105_hw5_ruijipan

ruijipan

2022-11-13

Introduction

This report is used to explain assignment 5 of R language. Assignment 5 mainly focuses on the study and training of iteration grammar in R language.

Problem 1

First, create a data folder to store the data needed for this report; Under the data folder, create a subfolder named longitudinal-study to store corresponding dataset.

The following code shows how to use the `list.files` function and the `map_dfr` function to read data sets in batches and store the data as a data frame type R language object; Among them, the `paste` function is used for string merging, which combines the path with the dataset. The file names are merged to form the input parameters of the `read.csv` function.

```
> library(tidyverse)
> filenames = list.files('./data/longitudinal-study')
> filepaths = paste('./data/longitudinal-study/',filenames,sep="")
> df = map_dfr(filepaths, read.csv, .id = "input")
> df
   input week_1 week_2 week_3 week_4 week_5 week_6 week_7 week_8
1      1   0.20  -1.31   0.66   1.96   0.23   1.09   0.05   1.94
2      2   1.13  -0.88   1.07   0.17  -0.83  -0.31   1.58   0.44
3      3   1.77   3.11   2.22   3.26   3.31   0.89   1.88   1.01
4      4   1.04   3.66   1.22   2.33   1.47   2.70   1.87   1.66
5      5   0.47  -0.58  -0.09  -1.37  -0.32  -2.17   0.45   0.48
6      6   2.37   2.50   1.59  -0.16   2.08   3.07   0.78   2.35
7      7   0.03   1.21   1.13   0.64   0.49  -0.12  -0.07   0.46
8      8  -0.08   1.42   0.09   0.36   1.18  -1.16   0.33  -0.44
9      9   0.08   1.24   1.44   0.41   0.95   2.75   0.30   0.03
10    10   2.14   1.15   2.52   3.44   4.26   0.97   2.73  -0.53
11    11   3.05   3.67   4.84   5.80   6.33   5.46   6.38   5.91
12    12  -0.84   2.63   1.64   2.58   1.24   2.32   3.11   3.78
13    13   2.15   2.08   1.82   2.84   3.36   3.61   3.37   3.74
14    14  -0.62   2.54   3.78   2.73   4.49   5.82   6.00   6.49
15    15   0.70   3.33   5.34   5.57   6.90   6.66   6.24   6.95
16    16   3.73   4.08   5.40   6.41   4.87   6.09   7.66   5.83
17    17   1.18   2.35   1.23   1.17   2.02   1.61   3.13   4.88
18    18   1.37   1.43   1.84   3.60   3.80   4.72   4.68   5.70
19    19  -0.40   1.08   2.66   2.70   2.80   2.64   3.51   3.27
20    20   1.09   2.80   2.80   4.30   2.25   6.57   6.09   4.64
```

Next, clean the dataset. As you can see, after the data is read in, it is a data frame type object. Rename the input field to arm by using the `rename` function, and use the character in front of the symbol " " *in the file* *name as the data content of this field; Then, a new variable "subject_id" is created, and the character after the symbol ""* in the file name is used as the data content of this field. After that, use the select function to rearrange the data fields. As shown in the table below, the data is relatively clean, so no further data cleaning is required.

```
> df = rename(df,c(arm=input))
> str_arm = strsplit(filenames,"\\.")
>
> var_arm = vector("list", length = 20)
> var_id = vector("list", length = 20)
> for(i in 1:20){
+   var_arm[[i]] = strsplit(str_arm[[i]][1],"_")[[1]][1]
+   var_id[[i]] = strsplit(str_arm[[i]][1],"_")[[1]][2]
+ }
> df$arm = var_arm
> df$subject_id = var_id
> df = df %>%
+   select(arm, subject_id, everything())
> df
   arm subject_id week_1 week_2 week_3 week_4 week_5 week_6 week_7 week_8
1  con         01   0.20  -1.31   0.66   1.96   0.23   1.09   0.05   1.94
2  con         02   1.13  -0.88   1.07   0.17  -0.83  -0.31   1.58   0.44
3  con         03   1.77   3.11   2.22   3.26   3.31   0.89   1.88   1.01
4  con         04   1.04   3.66   1.22   2.33   1.47   2.70   1.87   1.66
5  con         05   0.47  -0.58  -0.09  -1.37  -0.32  -2.17   0.45   0.48
6  con         06   2.37   2.50   1.59  -0.16   2.08   3.07   0.78   2.35
7  con         07   0.03   1.21   1.13   0.64   0.49  -0.12  -0.07   0.46
8  con         08  -0.08   1.42   0.09   0.36   1.18  -1.16   0.33  -0.44
9  con         09   0.08   1.24   1.44   0.41   0.95   2.75   0.30   0.03
10 con         10   2.14   1.15   2.52   3.44   4.26   0.97   2.73  -0.53
11 exp         01   3.05   3.67   4.84   5.80   6.33   5.46   6.38   5.91
12 exp         02  -0.84   2.63   1.64   2.58   1.24   2.32   3.11   3.78
13 exp         03   2.15   2.08   1.82   2.84   3.36   3.61   3.37   3.74
14 exp         04  -0.62   2.54   3.78   2.73   4.49   5.82   6.00   6.49
15 exp         05   0.70   3.33   5.34   5.57   6.90   6.66   6.24   6.95
16 exp         06   3.73   4.08   5.40   6.41   4.87   6.09   7.66   5.83
17 exp         07   1.18   2.35   1.23   1.17   2.02   1.61   3.13   4.88
18 exp         08   1.37   1.43   1.84   3.60   3.80   4.72   4.68   5.70
19 exp         09  -0.40   1.08   2.66   2.70   2.80   2.64   3.51   3.27
20 exp         10   1.09   2.80   2.80   4.30   2.25   6.57   6.09   4.64
```

Problem 2

The problem 2 uses the homicides data collected by The Washington Post. This data set contains a total of 52,179 observations and 12 variables. It mainly includes the ID of the case, the reported date of the case, some basic information of the victim, the city where the case happened and the settlement of the case. The following code creates a new city_state field for this data set, merges the city field and the state field, and calculates the total number of cases and the number of unsolved cases in each city.

```
> homicides_df = read.csv("./data/homicides/homicide-data.csv")
> homicides_df$city_state = paste(homicides_df$city,homicides_df$state)
> head(homicides_df)
         uid reported_date victim_last victim_first victim_race victim_age
1 Alb-000001      20100504      GARCIA         JUAN    Hispanic         78
2 Alb-000002      20100216     MONTOYA      CAMERON    Hispanic         17
```

```
3 Alb-000003      20100601 SATTERFIELD      VIVIANA       White          15
4 Alb-000004      20100101    MENDIOLA       CARLOS    Hispanic          32
5 Alb-000005      20100102        MULA       VIVIAN       White          72
6 Alb-000006      20100126        BOOK    GERALDINE       White          91
  victim_sex        city state      lat          lon           disposition
1       Male Albuquerque    NM 35.09579 -106.5385549 Closed without arrest
2       Male Albuquerque    NM 35.05681  -106.715321      Closed by arrest
3     Female Albuquerque    NM 35.08609  -106.695568 Closed without arrest
4       Male Albuquerque    NM 35.07849 -106.5560938      Closed by arrest
5     Female Albuquerque    NM 35.13036 -106.5809862 Closed without arrest
6     Female Albuquerque    NM 35.15111  -106.537797        Open/No arrest
      city_state
1 Albuquerque NM
2 Albuquerque NM
3 Albuquerque NM
4 Albuquerque NM
5 Albuquerque NM
6 Albuquerque NM
```

```
> homicidesBycity =
+   homicides_df %>%
+   group_by(city) %>%
+   summarise(count = n())
> head(homicidesBycity)
# A tibble: 6 x 2
  city        count
  <chr>       <int>
1 Albuquerque   378
2 Atlanta       973
3 Baltimore    2827
4 Baton Rouge   424
5 Birmingham    800
6 Boston        614
```

Next, the function `prop.test` is used to estimate the unsolved proportion of cases in "Baltimore, MD" city (about 65%), and the list is visualized to output the estimated proportion and confidence intervals of the city.

```
> unsolved_homicides =
+   homicides_df %>%
+   filter(disposition %in% c("Closed without arrest","Open/No arrest")) %>%
+   summarise(disposition = "unsolved",total = n())
> head(unsolved_homicides)
  disposition total
1    unsolved 26505
```

```
> Baltimore_MD_unsolved =
+   homicides_df %>%
+   filter(disposition %in% c("Closed without arrest","Open/No arrest") & city_state == "Baltimore MD")
+   summarise(disposition = "MD_unsolved",total = n())
> head(Baltimore_MD_unsolved)
  disposition total
1 MD_unsolved  1825
```

```
> Baltimore_MD_total =
+   homicides_df %>%
+   filter(city_state == "Baltimore MD") %>%
```

```
+   summarise(disposition = "MD_total",total = n())
> head(Baltimore_MD_total)
  disposition total
1    MD_total  2827
```

```
> prop_test = prop.test(Baltimore_MD_unsolved$total,Baltimore_MD_total$total)
> prop_test_df = broom::tidy(prop_test)
> prop_test_result_df = prop_test_df %>%
+   select(estimate, conf.low, conf.high)
> prop_test_result_df
# A tibble: 1 x 3
  estimate conf.low conf.high
     <dbl>    <dbl>     <dbl>
1    0.646    0.628     0.663
> cat("proportion estimate: ",prop_test_df$estimate, "\n")
proportion estimate:  0.6455607
> cat("the 0.95 conf.low: ", prop_test_df$conf.low, "\n")
the 0.95 conf.low:  0.6275625
> cat("the 0.95 conf.high: ", prop_test_df$conf.high, "\n")
the 0.95 conf.high:  0.6631599
```

For the above process, we package it into a function, and input it into different cities to get different results.

```
> # define a function for the above code
> # input: city
> # output: prop_test_result_df
> proportion = function(x){
+   city_total =
+   homicides_df %>%
+   filter(city_state == x) %>%
+   summarise(disposition = "city_total",total = n())
+
+   city_unsolved =
+   homicides_df %>%
+   filter(disposition %in% c("Closed without arrest","Open/No arrest") & city_state == x) %>%
+   summarise(disposition = "city_unsolved",total = n())
+
+   prop_test = prop.test(city_unsolved$total,city_total$total)
+   prop_test_df = broom::tidy(prop_test)
+   prop_test_result_df = prop_test_df %>%
+     select(estimate, conf.low, conf.high)
+
+   prop_test_result_df
+ }
```

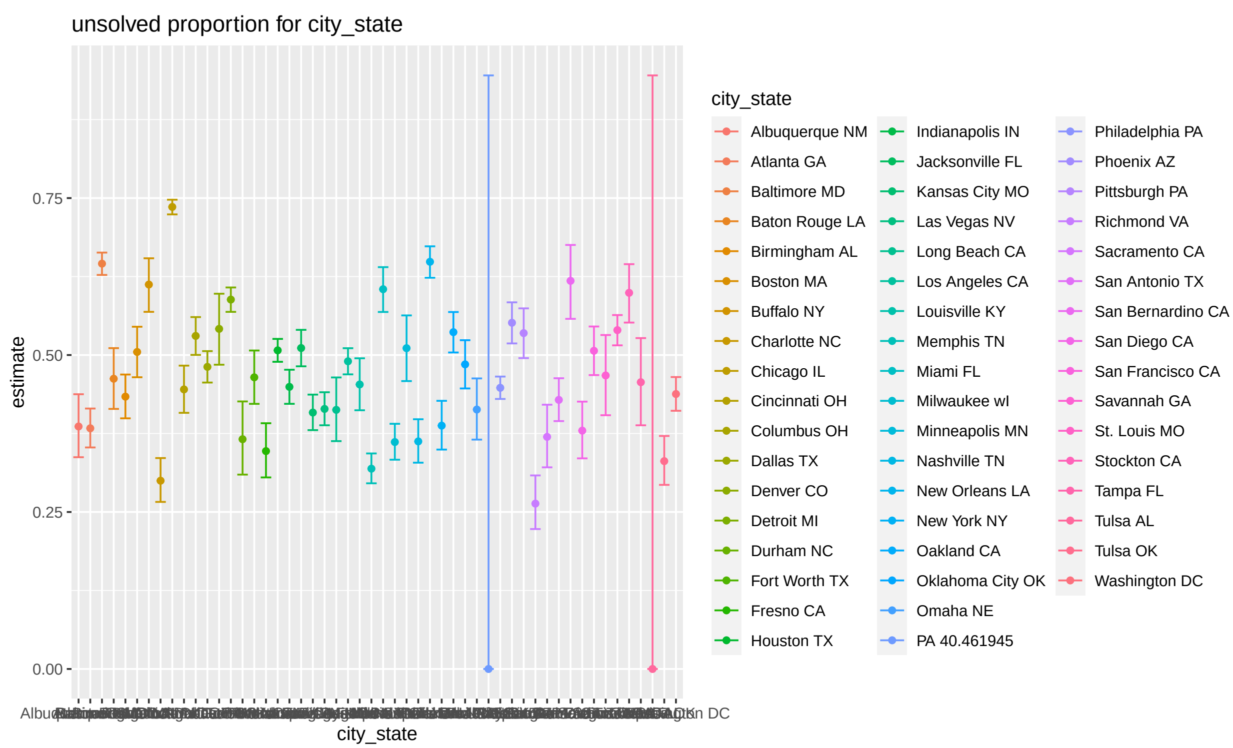
After that, use the `map_dfr` function to run our packaged functions in batches and output a data frame containing the output results of all cities.

```
>
> prop_result = map_dfr(unique(homicides_df$city_state), proportion, .id = "input")
> prop_result = rename(prop_result,c(city_state=input))
> prop_result$city_state = unique(homicides_df$city_state)
> prop_result
# A tibble: 52 x 4
   city_state     estimate conf.low conf.high
```

```
   <chr>            <dbl>   <dbl>    <dbl>
 1 Albuquerque NM   0.386   0.337    0.438
 2 Atlanta GA       0.383   0.353    0.415
 3 Baltimore MD     0.646   0.628    0.663
 4 Baton Rouge LA   0.462   0.414    0.511
 5 Birmingham AL    0.434   0.399    0.469
 6 Boston MA        0.505   0.465    0.545
 7 Buffalo NY       0.612   0.569    0.654
 8 Charlotte NC     0.300   0.266    0.336
 9 Chicago IL       0.736   0.724    0.747
10 Cincinnati OH    0.445   0.408    0.483
# ... with 42 more rows
```

For each city, use ggplot to visualize the proportion of its unsolved cases, and its corresponding confidence interval.

```
> prop_result %>%
+   ggplot() +
+   geom_errorbar(aes(x=city_state, ymin=conf.low, ymax=conf.high,color=city_state), position = positio
+   ggtitle("unsolved proportion for city_state")
```



Problem 3

For problem 3, use the `set.seed` function to ensure that our results can be reproduced; Then, the data with mean value of 0 and standard deviation of 5 are generated by cyclic batch. The total number of iterations is 5,000, and each iteration generates 30 observation data.

```
> library(tidyverse)
> set.seed(1)
```

```
> data_norm = vector("list", 5000)
> for(i in 1:5000){
+   data_norm[[i]] = rnorm(n = 30, mean = 0, sd = 5)
+ }
> listcol_df =
+   tibble(
+     sample_id = c(1:5000),
+     samp = data_norm
+   )
>
> listcol_df =
+   listcol_df %>%
+   mutate(summary = map2(.x = samp, .y = 0, ~t.test(x = .x, mu = .y)))
```

For the above-mentioned generated data, 5000 datasets are tested by single sample mean, and the estimated value and significance are extracted. The confidence level of 0.05 is used to judge whether the test result of each data set is significant, and whether the original hypothesis is rejected or not is stored as a rejected variable.

```
> mean_test_result = map_dfr(listcol_df[[3]], broom::tidy, .id = "sample_id") %>%
+   select(sample_id, estimate, p.value) %>%
+   mutate(rejected = p.value > 0.05)
> mean_test_result
# A tibble: 5,000 x 4
   sample_id estimate p.value rejected
   <chr>        <dbl>   <dbl> <lgl>
 1 1            0.412  0.629  TRUE
 2 2            0.664  0.368  TRUE
 3 3            0.551  0.534  TRUE
 4 4            0.567  0.487  TRUE
 5 5           -1.65   0.0599 TRUE
 6 6            1.19   0.229  TRUE
 7 7            0.334  0.738  TRUE
 8 8           -1.19   0.209  TRUE
 9 9            0.122  0.887  TRUE
10 10           0.684  0.472  TRUE
# ... with 4,990 more rows
```

The above process is packaged into a function, the input parameter of the function is the average value of random numbers with normal distribution, and the output is the test result of each data set.

```
> # define a function for the above code
> # input: mean
> # output: dataframe containing sample_id, true_u, estimate, p.value, rejected
> library(tidyverse)
> mean_test = function(x){
+   set.seed(1)
+   data_norm = vector("list", 5000)
+   for(i in 1:5000){
+     data_norm[[i]] = rnorm(n = 30, mean = x, sd = 5)
+   }
+   listcol_df =
+     tibble(
+       sample_id = c(1:5000),
+       samp = data_norm
```

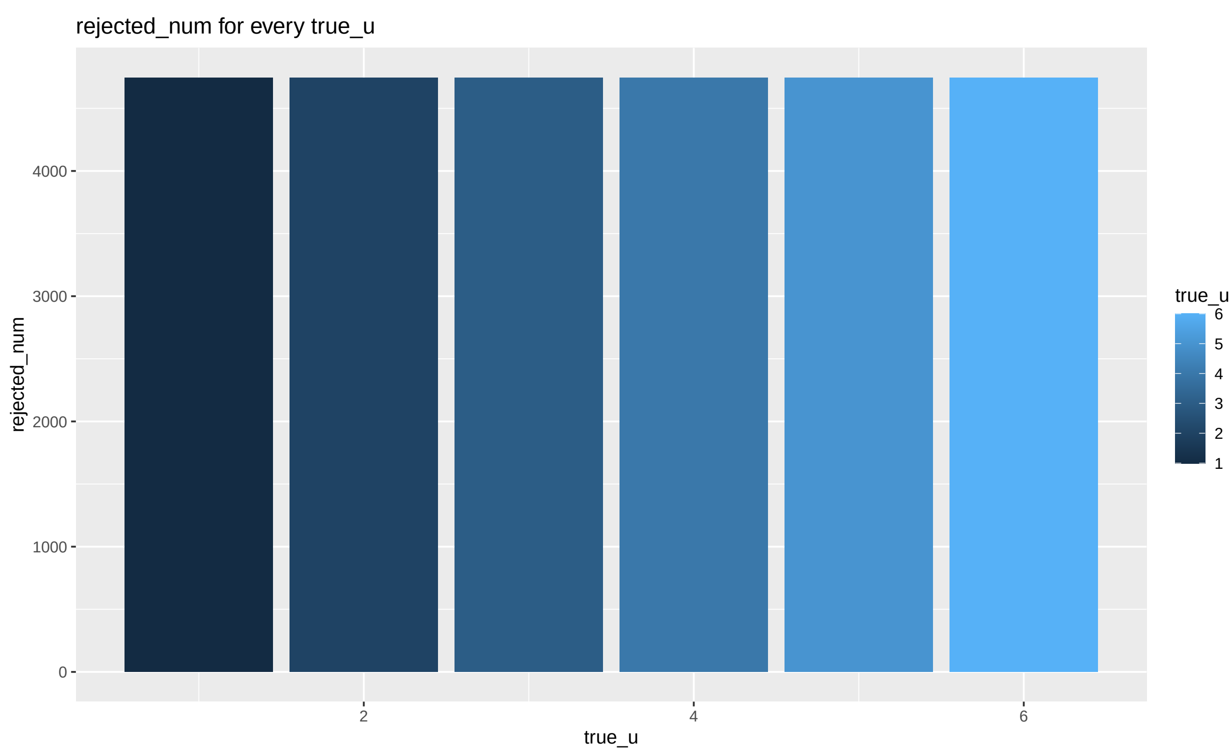
```
+     )
+
+   listcol_df =
+     listcol_df %>%
+     mutate(summary = map2(.x = samp, .y = x, ~t.test(x = .x, mu = .y)))
+
+   mean_test_result = map_dfr(listcol_df[[3]], broom::tidy, .id = "sample_id") %>%
+   select(sample_id, estimate, p.value) %>%
+   mutate(true_u = x, rejected = p.value > 0.05)
+   mean_test_result
+ }
```

Take the mean values of 1, 2, 3, 4, 5, 6, respectively, and run the above functions in batches. The results of the generated functions are stored in the variable mean_test_result_df.

```
> mean_vec = c(1,2,3,4,5,6)
> mean_test_result_df = map_dfr(mean_vec, mean_test)
```

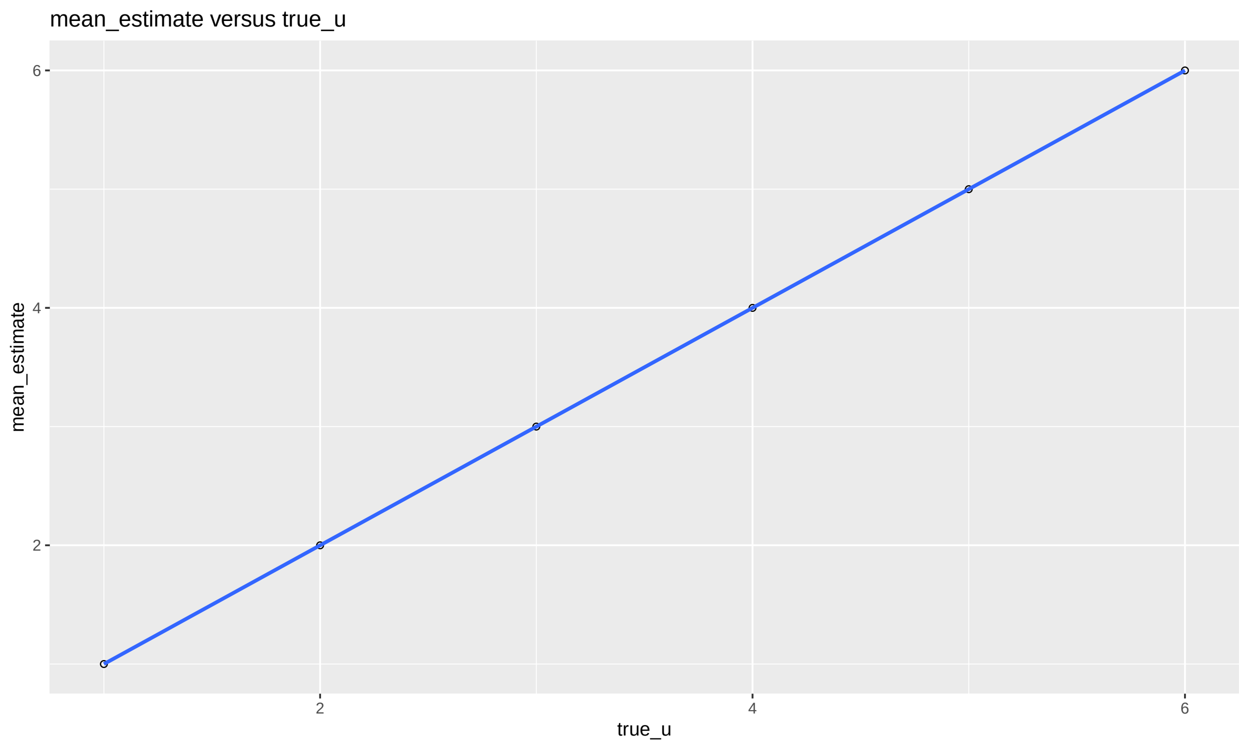
For the test result list generated above, ggplot is used to show the number of data sets that fail the test under each different mean value. As shown below:

```
> mean_test_result_df %>%
+   group_by(true_u) %>%
+   summarise(rejected_num = sum(rejected)) %>%
+   ggplot(aes(x = true_u, y = rejected_num, fill = true_u)) +
+   geom_bar(stat = "identity") +
+   ggtitle("rejected_num for every true_u")
```



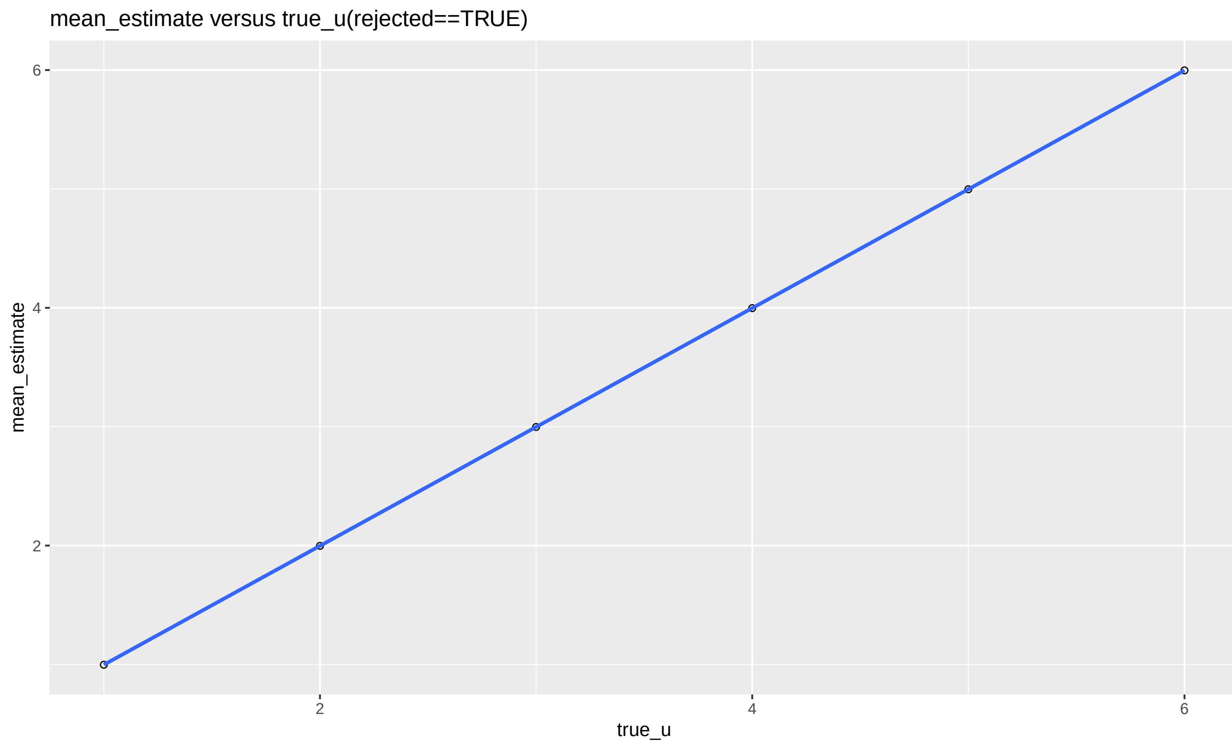
Similarly, the list of test results is grouped according to true_u, and the estimated mean value is calculated, and the corresponding scatter plot is drawn.

```
> mean_test_result_df %>%
+   group_by(true_u) %>%
+   summarise(mean_estimate = mean(estimate)) %>%
+   ggplot(aes(x = true_u, y = mean_estimate)) +
+   geom_point(shape=1) +
+   geom_smooth(method = 'loess') +
+   ggtitle("mean_estimate versus true_u")
```



Group the list of test results according to true_u, first screen out the data that reject the original hypothesis, then calculate the estimated mean and draw the corresponding scatter plot.

```
> mean_test_result_df %>%
+   filter(rejected == TRUE) %>%
+   group_by(true_u) %>%
+   summarise(mean_estimate = mean(estimate)) %>%
+   ggplot(aes(x = true_u, y = mean_estimate)) +
+   geom_point(shape=1) +
+   geom_smooth(method = 'loess') +
+   ggtitle("mean_estimate versus true_u(rejected==TRUE)")
```



It can be found that the estimated mean values calculated by the two scatterplots are almost identical. This is because the percentage of failed tests is very high, reaching 80%~90%. Therefore, the estimated average is of course mainly determined by the data sets that failed tests.