Homework 9

Benjamin Anderson II

```r
library(tidyverse)
library(nycflights13)
```

Functions

1.

(2 pts)

In R, the function `t.test()` conducts one and two sample t-tests. For instance the following code runs Welch's two sample t-test using the `sleep` data in R:

```r
my_test_output <- t.test(extra ~ group, data = sleep)
my_test_output
```

```

    Welch Two Sample t-test

data:  extra by group
t = -1.8608, df = 17.776, p-value = 0.07939
alternative hypothesis: true difference in means between group 1 and group 2 is not equal to
95 percent confidence interval:
 -3.3654832  0.2054832
sample estimates:
mean in group 1 mean in group 2 
           0.75            2.33 
```

The exact meaning of this output is unimportant for this task, but imagine you are interested in the two numbers in the output labeled "95 percent confidence interval:", i.e the values `-3.3654832` and `0.2054832`.

Verify that `my_test_output` is built on top of a list. Then, return the names of the elements of that list

```
typeof(my_test_output)
```

```
[1] "list"
```

```
names(my_test_output)
```

```
 [1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"   
 [6] "null.value"  "stderr"      "alternative" "method"      "data.name"  
```

Use list subsetting to extract the values of interest from `my_test_output`.

```
my_test_output$conf.int
```

```
[1] -3.3654832  0.2054832
attr(,"conf.level")
[1] 0.95
```

Turn your code from the previous tasks into a function, called `conf_int()`, that extracts the confidence interval values from any `t.test()` output.

```
conf_int <- function(test){
  test$conf.int
}
```

Test your function by running:

```
conf_int(my_test_output)
```

```
[1] -3.3654832  0.2054832
attr(,"conf.level")
[1] 0.95
```

The output should be:

```
[1] -3.3654832  0.2054832
attr(,"conf.level")
[1] 0.95
```

2.

(2 pts)

The following code is an example of taking two vectors of the same length and joining them together element-wise to create a single character vector:

```r
farm <- c(1, 1, 2, 2, 3, 4)
field <- c("a", "b", "a", "b", "a", "a")
paste(farm, field, sep = "_")
```

```
[1] "1_a" "1_b" "2_a" "2_b" "3_a" "4_a"
```

For instance, you might use this to generate a single identifying variable from a couple of variables.

Turn this code into a function called `join_with_underscore()`, that takes two vectors `x` and `y` as input, and joins them into a single character string.

```r
join_with_underscore <- function(x, y){
  paste(x, y, sep = "_")
}
```

Check that your function works by testing it with `farm` and `field`.

```r
join_with_underscore(farm, field)
```

```
[1] "1_a" "1_b" "2_a" "2_b" "3_a" "4_a"
```

3.

(1 pt)

Reduce the repetition in this code by using `across()`:

```r
starwars |>
  mutate(
    ## UNREDUCED ----

    #n_films = lengths(films),
    #n_vehicles = lengths(vehicles),
    #n_starships = lengths(starships)


    ## REDUCED ----

    across("films":"starships", lengths, .names = "n_{.col}")
  )
```

```
# A tibble: 87 x 17
   name      height  mass hair_color skin_color eye_color birth_year sex   gender
   <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
 1 Luke Sk~     172    77 blond      fair       blue            19   male  mascu~
 2 C-3PO        167    75 <NA>       gold       yellow         112   none  mascu~
 3 R2-D2         96    32 <NA>       white, bl~ red             33   none  mascu~
 4 Darth V~     202   136 none       white      yellow          41.9 male  mascu~
 5 Leia Or~     150    49 brown      light      brown           19   fema~ femin~
 6 Owen La~     178   120 brown, gr~ light      blue            52   male  mascu~
 7 Beru Wh~     165    75 brown      light      blue            47   fema~ femin~
 8 R5-D4         97    32 <NA>       white, red red             NA   none  mascu~
 9 Biggs D~     183    84 black      light      brown           24   male  mascu~
10 Obi-Wan~     182    77 auburn, w~ fair       blue-gray       57   male  mascu~
# i 77 more rows
# i 8 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>, n_films <int>, n_vehicles <int>,
#   n_starships <int>
```

4.

(3 pts)

```r
set.seed(1846689310)
# Create a small version of flights
flights_small <- flights |> slice(sample(n(), size = 10))
```

Reduce the repetition in this code, by writing two functions, and using `across()`.

```r
# I was able to get away with 1 function rather than 2
#   using purrr-style lambdas and ternary operators

get_time <- function(x, time = "hour"){
  stringr::str_sub(
    x,
    # R's equivalent to ternary operators is the if() function
    start = if(time == "hour") -4 else -2,
    end = if(time == "hour") -3 else -1
  ) |>
    parse_number()
}

flights_small |>
  mutate(
    ## UNREDUCED ----

    #sched_arr_time_hour = stringr::str_sub(sched_arr_time, -4, -3) |>
    #  parse_number(),
    #sched_arr_time_min = stringr::str_sub(sched_arr_time, -2, -1) |>
    #  parse_number(),
    #arr_time_hour = stringr::str_sub(arr_time, -4, -3) |>
    #  parse_number(),
    #arr_time_min = stringr::str_sub(arr_time, -2, -1) |>
    #  parse_number(),


    ## REDUCED ----

    across(
      ends_with("arr_time"),
      .fns = list(
        hour = ~ get_time(.x, time = "hour"),
        min = ~ get_time(.x, time = "min")),
      .names = "{.col}_{.fn}"
    ),
    .keep = "used"
  )
```

```
# A tibble: 10 x 6
   arr_time sched_arr_time arr_time_hour arr_time_min sched_arr_time_hour
      <int>          <int>         <dbl>        <dbl>               <dbl>
```

```
 1     1902           1920          19         2                   19
 2     1725           1759          17        25                   17
 3     2259           2220          22        59                   22
 4     1330           1409          13        30                   14
 5       11             22          NA        11                   NA
 6     2135           2210          21        35                   22
 7     1405           1418          14         5                   14
 8      919            908           9        19                    9
 9     2102           2035          21         2                   20
10     2125           2130          21        25                   21
# i 1 more variable: sched_arr_time_min <dbl>
```