Homework 5 Solution

Disclaimer

Assignment prompt

Problem 1: regular expression using `babynames`

a.

Construct a regular expression (regex) to find all names that end in a vowel (here, you can consider "y" to be a vowel). Store this regex in `pattern` and run the code to determine how many baby names in 2017 ended with a vowel. Use babynames dataset from the `babynames` package.

Answer:

First, I specify that we are looking for any letter in the set "aeiouy", then I add the anchor for the end of the string. **The sequence of attempts below shows how you build from an initial idea to your final pattern. You can also use `str_view_all` to highlight these patterns, but this won't knit to a pdf!**

```
> # create a pattern
> x <- c("abba", "day", "cat")
> pattern <- "aeiouy"  # attempt 1
> str_detect(x, pattern) # nope! this is looking for the string "aeiouy"!
[1] FALSE FALSE FALSE
> pattern <- "[aeiouy]"  # attempt 2: "one of"
> str_detect(x, pattern) # nope! looks for any vowel
[1] TRUE TRUE TRUE
>
> # test your pattern
> pattern <- "[aeiouy]$"  # attempt 3: this finds vowels at the end of a string
> str_detect(x, pattern)
[1]  TRUE  TRUE FALSE
>
> babynames %>%
+   filter(year == 2017) %>%
+   summarize(last_vowel = sum(str_detect(name, pattern)))
# A tibble: 1 x 1
  last_vowel
       <int>
1      16314
```

b.

Write a regex that finds names that start in "Ed" and ends with "rd" so that it matches names like `"Edward"` and `"Eddard"`. Any number of characters can be in between. Check your regex using the small vector given below then determine how many babynames in 2017 match this pattern.

```
> x <- c("Edward", "Eddard", "Ned")
```

Answer:
We need a pattern that starts (^) with `Ed`, followed by any characters `.`, and ends with `rd`.

We could start with this attempt:

```
> pattern <- "^Ed.rd$"  # start, anything, end
> str_detect(x, pattern)
[1] FALSE FALSE FALSE
> babynames %>% filter(year == 2017, str_detect(name, pattern))
# A tibble: 0 x 5
# i 5 variables: year <dbl>, sex <chr>, name <chr>, n <int>, prop <dbl>
```

but `.` only specifies one character.

Here we use `.*` to denote "zero or more" characters before ending with `rd`.

```
> pattern <- "^Ed.*rd$"  # start, anything (0 or more), end
> str_detect(x, pattern)
[1]  TRUE  TRUE FALSE
> babynames %>% filter(year == 2017, str_detect(name, pattern))
# A tibble: 4 x 5
   year sex   name       n       prop
  <dbl> <chr> <chr>  <int>      <dbl>
1  2017 M     Edward  2309 0.00118
2  2017 M     Eduard    28 0.0000143
3  2017 M     Edgard    12 0.00000611
4  2017 M     Edvard     5 0.00000255
```

c.

Construct a regex to find all names that only contain vowels in 2017. (In this problem, consider vowels to be aeiouy.) Are there any such names? If so, what are they?

Answer:

here we specify a name that both starts with and ends with one or more vowels:

```
> pattern <- "^[aeiouy]+$"
> str_detect(c("you", "me"), pattern)
[1]  TRUE FALSE
>
> # Extracting names
>
> babynames %>%
+   mutate(name = str_to_lower(name)) %>%
+   filter(year == 2017, str_detect(name, pattern)) %>%
+   pull(name)  # vector of names
 [1] "aya"  "aiya" "yui"  "iya"  "aoi"  "yue"  "yoyo" "yu"   "yiyi" "io"
[11] "ai"   "yaa"  "yi"   "aaya" "oya"  "yi"   "yao"  "yu"   "ayo"
```

d.

Construct a regex to find all four-letter names in 2017 and store these names in a vector. (Hint: try using `dplyr::pull`) Print the length of this vector. (Please don't print the entire vector.)

Answer:

There are 2960 four-letter baby names in 2017.

You might first try a pattern that is 4 letters long, but this will capture any string of 4 letters within a longer name

```
> pattern <- "[a-z]{4}"  # attempt 1
> str_detect(c("mary","ann", "maryann"), pattern)  # nope!
[1]  TRUE FALSE  TRUE
```

Let's try a start and end anchor again to find all names that have 4 letters to start and end the name

```
> pattern <- "^[a-z]{4}$"
> str_detect(c("mary","ann", "maryann"), pattern)  # seems to work!
[1]  TRUE FALSE FALSE
>
> # Extracting names
> four_letter_names <- babynames %>%
+   mutate(name = str_to_lower(name)) %>%
+   filter(year == 2017, str_detect(name, pattern)) %>%
+   pull(name)
> length(four_letter_names)
[1] 2960
```

Not sufficient You can use `str_length` to filter 4 letter names. But the prompt asked for a regular expression that can do this, so make sure you understand that method too!

```
> # not a regex solution!
> babynames %>%
+   filter(year == 2017, str_length(name) == 4)
# A tibble: 2,960 x 5
    year sex   name      n    prop
   <dbl> <chr> <chr> <int>   <dbl>
 1  2017 F     Emma  19738 0.0105
 2  2017 F     Ella   8014 0.00427
 3  2017 F     Aria   7132 0.00380
 4  2017 F     Nora   6036 0.00322
 5  2017 F     Zoey   6026 0.00321
 6  2017 F     Mila   5941 0.00317
 7  2017 F     Lily   5816 0.00310
 8  2017 F     Luna   5320 0.00284
 9  2017 F     Leah   5159 0.00275
10  2017 F     Lucy   4564 0.00243
# i 2,950 more rows
```

e.

Write a regex to find all of the four letter names that are palindromes in your names from part (d). ("Anna" is one such example.) Check this on the baby names from 2017. (Hint: You will need to capture the first two letters individually and then use back references. Also, it's easier to operate on all lowercase letters.)

Answer:

There are 10 such 4 letter names in 2017.

Here we using two groupings of any character for the first two spots, then repeat the second grouping and the first. If we wanted to

```
> pattern <- "(.)(.)\\2\\1"
> str_detect(c("anna", "baab"), pattern)
[1] TRUE TRUE
>
> # Extracting names from 2017
> str_subset(four_letter_names, pattern)
 [1] "anna" "elle" "emme" "adda" "alla" "izzi" "luul" "avva" "otto" "anna"
```

Problem 2: Energy autocorrelation

Auto correlation measures the correlation between measurements that differ by a fixed amount of time. For example, "lag 1" autocorrelation measures the correlation between measurements that are 1 time unit apart, lag 2 measures the correlation between measurements 2 time units apart, etc.

```
> energy <- readr::read_csv("https://raw.githubusercontent.com/deepbas/statdatasets/main/energy.csv",
+                      col_type = cols(
+                       .default = col_double(),
+                        Timestamp = col_datetime(format = ""),
+                        dayWeek = col_factor(levels=c("Mon","Tues","Wed","Thurs","Fri","Sat","Sun"))
+                       ))
```

The `acf` function computes autocorrelation in R. Here we get autocorrelation for Olin energy readings. The correlation between units 0 minutes apart is 1 (they are the same measurements!), the correlation between readings 15 minutes apart is 0.956, between 30 minutes apart is 0.95, between 45 minutes apart is 0.934 and between 60 minutes apart is 0.917. It makes sense that a correlation exists between energy use at points close in time.

```
> x <- energy %>%
+   arrange(Timestamp) %>%    # making sure sorted by time
+   pull("Olin_Hall_of_Science")
> acf_out <- acf(
+   x,   # time series
+   na.action = na.pass,    # skips over NAs
+   lag.max = 4,    # max lag
+   plot = FALSE)    # don't plot
> acf_out

Autocorrelations of series 'x', by lag

    0     1     2     3     4
1.000 0.956 0.950 0.934 0.917
> acf_out$acf    # autocorr values
, , 1

          [,1]
[1,] 1.0000000
[2,] 0.9556181
[3,] 0.9502154
[4,] 0.9344803
[5,] 0.9169001
```

```
> acf_out$lag   # lag values
, , 1

     [,1]
[1,]    0
[2,]    1
[3,]    2
[4,]    3
[5,]    4
```

a.

Write a function called `autocor_fun` that will take in:

- the vector of KWH values from one building
- a max lag value

and return a *data frame* with variables

- `autocor` which measures autocorrelation between `energyKWH` values. Note that you will need to extract the `acf` values from the `acf` function and coerce these into a vector with `as.vector`.
- `lag` which tells what lag the `autocor` was computed at

Use the `na.action` and `plot` arguments in `acf` that are shown above when writing your function.

Check your `autocor_fun` on the Olin KWH data that was used above with a max lag of 4.

answer:

```
> autocor_fun <- function(x, maxlag){
+   output <- acf(
+     x,
+     na.action = na.pass,
+     lag.max = maxlag,
+     plot = FALSE)
+   data.frame(
+     autocor = as.vector(output$acf),
+     lag = 0:(maxlag)) %>% as_tibble()
+ }
>
> autocor_fun(energy$"Olin_Hall_of_Science", maxlag = 4)  #
# A tibble: 5 x 2
  autocor   lag
    <dbl> <int>
1   1         0
2   0.956     1
3   0.950     2
4   0.934     3
5   0.917     4
```

b.

Use the `purrr` package to apply `autocor_fun` to the buildings `"Sayles-Hill" ,"Language_&_Dining_Center"`, `"Olin_Hall_of_Science"` with a max lag of 4. Use the wide data frame `energy` and your result should be a data frame that also identifies buildings.

answer:

```
> energy %>%
+     arrange(Timestamp) %>%  # make sure arranged by time
+   select("Sayles-Hill" ,"Language_&_Dining_Center", "Olin_Hall_of_Science") %>%
+  map_dfr(autocor_fun, maxlag = 4, .id = "building")
# A tibble: 15 x 3
   building                 autocor   lag
   <chr>                      <dbl> <int>
 1 Sayles-Hill                1         0
 2 Sayles-Hill                0.936     1
 3 Sayles-Hill                0.929     2
 4 Sayles-Hill                0.915     3
 5 Sayles-Hill                0.894     4
 6 Language_&_Dining_Center   1         0
 7 Language_&_Dining_Center   0.955     1
 8 Language_&_Dining_Center   0.942     2
 9 Language_&_Dining_Center   0.926     3
10 Language_&_Dining_Center   0.908     4
11 Olin_Hall_of_Science       1         0
12 Olin_Hall_of_Science       0.956     1
13 Olin_Hall_of_Science       0.950     2
14 Olin_Hall_of_Science       0.934     3
15 Olin_Hall_of_Science       0.917     4
```

```
> energy %>%
+     arrange(Timestamp) %>%  # make sure arranged by time
+   select("Sayles-Hill" ,"Language_&_Dining_Center", "Olin_Hall_of_Science") %>%
+  map_df(autocor_fun, maxlag = 4, .id = "building")
# A tibble: 15 x 3
   building                 autocor   lag
   <chr>                      <dbl> <int>
 1 Sayles-Hill                1         0
 2 Sayles-Hill                0.936     1
 3 Sayles-Hill                0.929     2
 4 Sayles-Hill                0.915     3
 5 Sayles-Hill                0.894     4
 6 Language_&_Dining_Center   1         0
 7 Language_&_Dining_Center   0.955     1
 8 Language_&_Dining_Center   0.942     2
 9 Language_&_Dining_Center   0.926     3
10 Language_&_Dining_Center   0.908     4
11 Olin_Hall_of_Science       1         0
12 Olin_Hall_of_Science       0.956     1
13 Olin_Hall_of_Science       0.950     2
14 Olin_Hall_of_Science       0.934     3
15 Olin_Hall_of_Science       0.917     4
```

c.

Repeat (b) but this time use the narrow version of the data `energy_narrow`. This time you will use `dplyr` groupings and summarize to apply the `autocor_fun` to each building's KWH. (Don't forget to filter to the three buildings). Your data frame results for (b) and (c) should be identical.

```
> energy_narrow <- energy %>%
+   mutate(month = month(month, label=TRUE)) %>%
+   pivot_longer(
+     cols = 9:90,
+     names_to = "building",
+     values_to = "energyKWH")
```

answer:

```
> energy_narrow %>%
+   filter(building %in% c("Sayles-Hill" ,"Language_&_Dining_Center", "Olin_Hall_of_Science"))  %>%
+   arrange(Timestamp) %>%  # make sure arranged by time
+   group_by(building) %>%
+   summarize(autocor_fun(energyKWH, maxlag = 4))  # get 1 hour of acf's
Warning: Returning more (or less) than 1 row per `summarise()` group was deprecated in
dplyr 1.1.0.
i Please use `reframe()` instead.
i When switching from `summarise()` to `reframe()`, remember that `reframe()`
  always returns an ungrouped data frame and adjust accordingly.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
generated.
# A tibble: 15 x 3
# Groups:   building [3]
   building                 autocor   lag
   <chr>                      <dbl> <int>
 1 Language_&_Dining_Center   1         0
 2 Language_&_Dining_Center   0.955     1
 3 Language_&_Dining_Center   0.942     2
 4 Language_&_Dining_Center   0.926     3
 5 Language_&_Dining_Center   0.908     4
 6 Olin_Hall_of_Science       1         0
 7 Olin_Hall_of_Science       0.956     1
 8 Olin_Hall_of_Science       0.950     2
 9 Olin_Hall_of_Science       0.934     3
10 Olin_Hall_of_Science       0.917     4
11 Sayles-Hill                1         0
12 Sayles-Hill                0.936     1
13 Sayles-Hill                0.929     2
14 Sayles-Hill                0.915     3
15 Sayles-Hill                0.894     4
```

d.

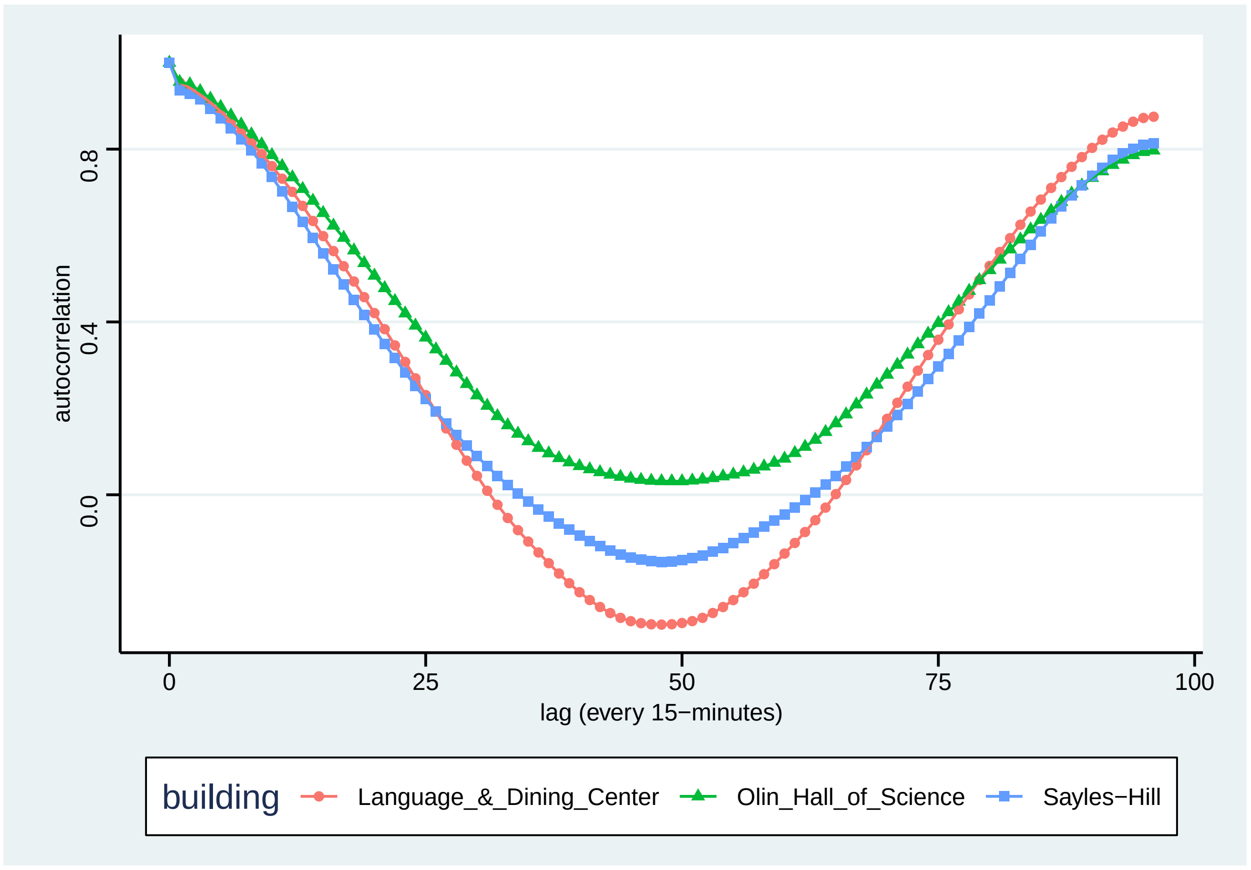
Create a data frame that contains acf values for 24 hours (i.e. a max lag of 24x4=96) for the three buildings used in (b, c). The plog acf values (y) against lag (x) for each of the buildings and describe what trends you see.

answer: We see that correlation is the lowest (and negative) at a lag of about 50, or measurements about 12.5 hours apart (50x15/60). This means there is a negative correlation between, say, KWH usage at noon and KWH usage at midnight. But after this acf increases which means that, say, KWH usage at noon on a Monday is highly correlated with KWH usage at 11:00am on a Tuesday.

```
> autocor_df <- energy_narrow %>%
+   filter(building %in% c("Sayles-Hill" ,"Language_&_Dining_Center", "Olin_Hall_of_Science"))  %>%
+   arrange(Timestamp) %>%  # make sure arranged by time
+   group_by(building) %>%
+   summarize(autocor_fun(energyKWH, maxlag = 24*4))  # get 24 hours of acf's
```

```
Warning: Returning more (or less) than 1 row per `summarise()` group was deprecated in
dplyr 1.1.0.
i Please use `reframe()` instead.
i When switching from `summarise()` to `reframe()`, remember that `reframe()`
  always returns an ungrouped data frame and adjust accordingly.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
generated.
> autocor_df
# A tibble: 291 x 3
# Groups:   building [3]
   building                 autocor   lag
   <chr>                      <dbl> <int>
 1 Language_&_Dining_Center   1         0
 2 Language_&_Dining_Center   0.955     1
 3 Language_&_Dining_Center   0.942     2
 4 Language_&_Dining_Center   0.926     3
 5 Language_&_Dining_Center   0.908     4
 6 Language_&_Dining_Center   0.888     5
 7 Language_&_Dining_Center   0.865     6
 8 Language_&_Dining_Center   0.840     7
 9 Language_&_Dining_Center   0.815     8
10 Language_&_Dining_Center   0.789     9
# i 281 more rows
```

```
> autocor_df %>%
+   filter(building %in% c("Sayles-Hill" ,"Language_&_Dining_Center", "Olin_Hall_of_Science"))  %>%
+   ggplot(aes(x = lag, y = autocor, color = building)) +
+   geom_point(aes(shape=building)) +
+   geom_line() +
+   labs(y = "autocorrelation", x = "lag (every 15-minutes)")
```



Problem 3: weather

Load the `nasaweather` data and look at the help file for the atmospheric data `?atmos`.

a.

Create a `for` loop that computes the coefficient of variation for all atmospheric measurements except location (lat/long) and time (year, month) variables. The coefficient of variation is the ratio of the standard deviation over the mean of a variable. Print out the output of your loop.

answer:

```
> library(nasaweather)
>
> my_cvs <- rep(NA, 7)  # indexed over the 7 variables
>
> # this option indexes by index position number
> for (i in 1:7){
+   my_cvs[i] <- sd(atmos[[i + 4]], na.rm = TRUE)/mean(atmos[[i + 4]], na.rm = TRUE)
+ }
> my_cvs
[1] 0.01577161 0.01586838 0.04585422 0.07099756 0.56453778 0.63466236 1.02927631
>
> # you could use seq_along but this will also be indexed by 1-7 since you've subsetted the data
> seq_along(atmos[,5:11])
[1] 1 2 3 4 5 6 7
>
```

```
> # this option indexes by column number from the data frame
> for (i in 5:11){
+   my_cvs[i-4] <- sd(atmos[[i]], na.rm = TRUE)/mean(atmos[[i ]], na.rm = TRUE)
+ }
> my_cvs
[1] 0.01577161 0.01586838 0.04585422 0.07099756 0.56453778 0.63466236 1.02927631
```

b.

Use a `map` function (from `purrr` package) to compute the coefficient of variation for all atmospheric measurements except location (lat/long) and time (year, month) variables. Use a function that returns a vector or data frame (and show this output).

answer:

```
> # here is one way: using the special map formula syntax for defining a function: (see ?map for more i
> atmos %>%
+   select(5:11) %>%
+   map(.f = ~sd(.x, na.rm = TRUE)/mean(.x, na.rm=TRUE)) %>% unlist()
  surftemp       temp   pressure      ozone   cloudlow   cloudmid  cloudhigh
0.01577161 0.01586838 0.04585422 0.07099756 0.56453778 0.63466236 1.02927631
>
> # here is another way: using the base-R way of creating a function
> atmos %>%
+   select(5:11) %>%
+   map_dbl(.f = function(x) sd(x, na.rm = TRUE)/mean(x, na.rm=TRUE))
  surftemp       temp   pressure      ozone   cloudlow   cloudmid  cloudhigh
0.01577161 0.01586838 0.04585422 0.07099756 0.56453778 0.63466236 1.02927631
```

c.

Create a function called `my_stats` that computes the following statistics for an input vector: mean, sd, and 5-number summary (min/Q1/median/Q3/max). The function should return an output object that is a "named" vector. E.g. the following is a named vector with the name of the left side of `=` and the value on the right side. Show the output of your function by inputting the `temp` values from `atmos`.

```
> # e.g. named vector with names x and y and values 1 and 2
> c(x = 1, y = 2)
x y
1 2
```

answer:

```
> my_stats <- function(x, na.rm = TRUE){
+   mean <- mean(x, na.rm = na.rm)
+   sd <- sd(x, na.rm = na.rm)
+   percentiles <- quantile(x, na.rm = na.rm)
+   return(c(mean = mean, sd = sd, Q = percentiles))
+ }
> my_stats(atmos$temp)
      mean         sd       Q.0%      Q.25%      Q.50%      Q.75%     Q.100%
297.921125   4.727525 269.100000 295.500000 299.200000 301.400000 310.000000
```

d.

Use `map_df` to compute `my_stats` for all atmospheric measurements except location (lat/long) and time (year, month) variables. Include the variable names as a column in your data frame.

answer:

```
> atmos %>% select(5:11) %>%
+   map_df(my_stats, .id = "variables")
# A tibble: 7 x 8
  variables  mean    sd `Q.0%` `Q.25%` `Q.50%` `Q.75%` `Q.100%`
  <chr>     <dbl> <dbl>  <dbl>   <dbl>   <dbl>   <dbl>    <dbl>
1 surftemp  296.   4.67  266     294.    297.    299.     315.
2 temp      298.   4.73  269.    296.    299.    301.     310
3 pressure  985.  45.2   615     995    1000    1000     1000
4 ozone     267.  19.0   232     254     264     276      390
5 cloudlow   26.2 14.8     0.5    15      23.5    34.5     84.5
6 cloudmid   15.3  9.69    0       7.5    14      22       83.5
7 cloudhigh  12.0 12.4     0       1.5     8.5    18.5     62.5
```

e.

For each year, compute `my_stats` for the `temp` variable. To do this:

- use a year grouping and the `summarize` command
- add a `stats` variable that identifies which statistic is entered in a given row
- then make a wider version of this data that contains a column for each of the `my_stats` for each year in the data. (Final dimensions should be 6x8)

answer:

```
> atmos %>%
+   group_by(year) %>%
+   summarize(values = my_stats(temp)) %>%
+   mutate(stats = c("mean", "sd", "min", "perc_25", "median", "perc_75", "max")) %>%
+   pivot_wider(
+     names_from = stats,
+     values_from = values
+   )
Warning: Returning more (or less) than 1 row per `summarise()` group was deprecated in
dplyr 1.1.0.
i Please use `reframe()` instead.
i When switching from `summarise()` to `reframe()`, remember that `reframe()`
  always returns an ungrouped data frame and adjust accordingly.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
generated.
# A tibble: 6 x 8
# Groups:   year [6]
   year  mean    sd   min perc_25 median perc_75   max
  <int> <dbl> <dbl> <dbl>   <dbl>  <dbl>   <dbl> <dbl>
1  1995  297.  4.94  269.    296.   298.    300.  308.
2  1996  297.  4.55  272.    295    298.    300.  308.
3  1997  298.  4.77  273.    296    299.    301.  308.
4  1998  299.  4.65  273.    296.   300.    302.  310
5  1999  298.  4.49  273.    296.   299.    301.  310.
6  2000  298.  4.75  273.    296    300.    302.  309.
```

Problem 4: String Extraction with titanic train dataset.

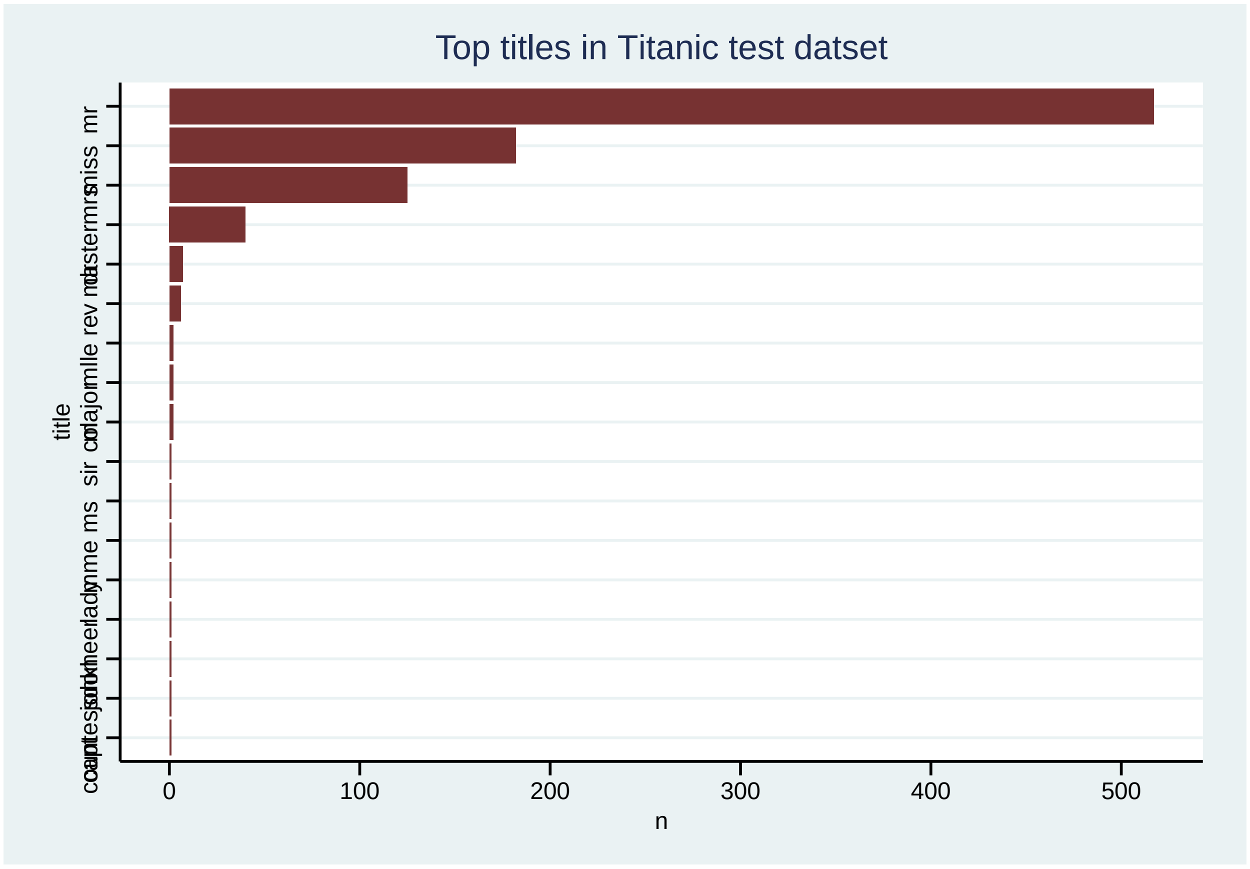
```
> #install.packages("titanic")
> library(titanic)
> set.seed(12233)
> df = tibble(titanic_train)  #load dataset
```

a. The package `titanic` has data on the survival information of Titanic passengers including `Name`. Extract titles from the names of the passengers. These include `Mr`, `Mrs`, etc. Use suitable regex that consists of a look ahead pattern that tells us to look for something followed by a space, one or more instances of string, and a look behind to match patterns to a period after the string, respectively.

```
> # Define the regex
> reg <- "(?<=\\s)[[:alpha:]]+(?=\\.)"
```

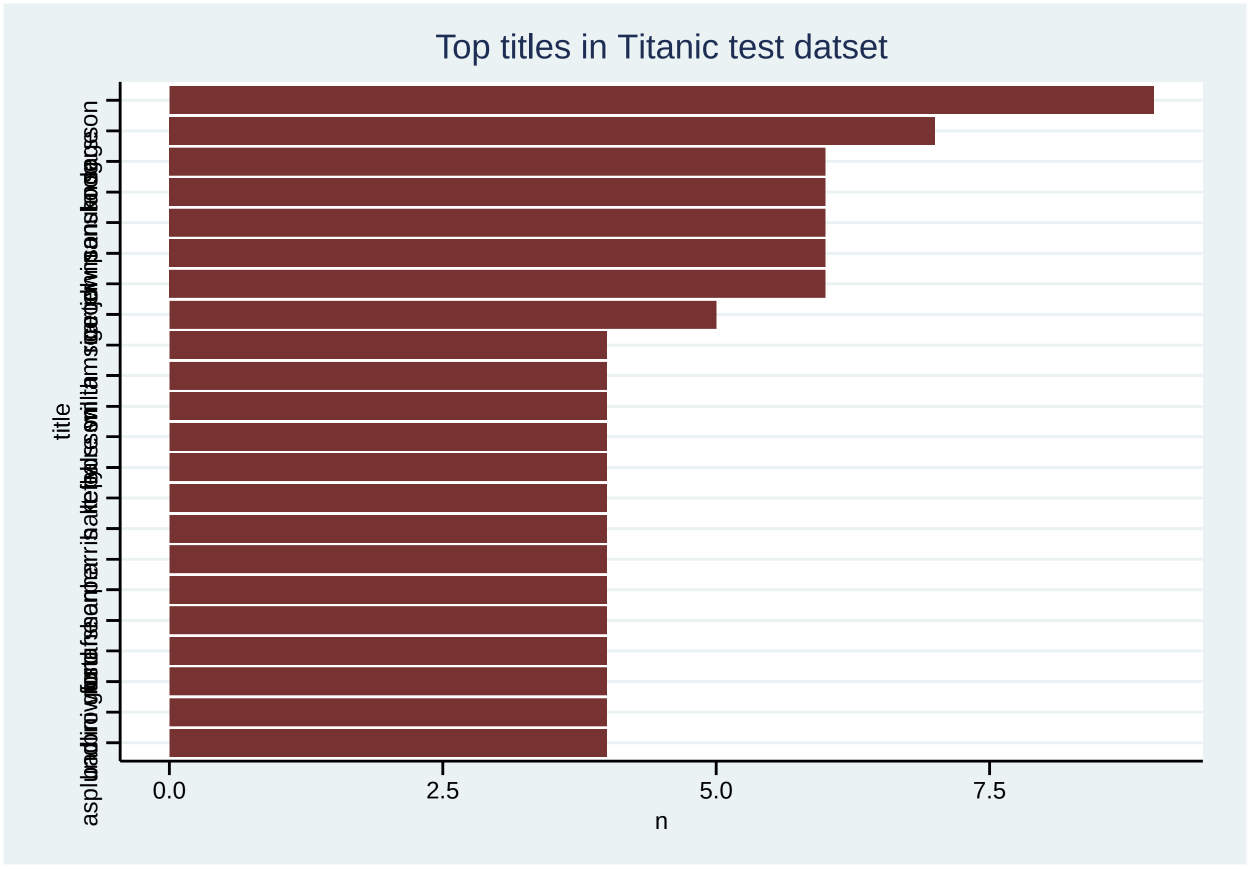
b. Plot the distribution of the the titles using either `geom_col()` or `geom_bar()` after properly reordering the variable based on their counts.

```
> df %>%
+   mutate(title = str_extract(str_to_lower(Name), reg)) %>%
+   count(title, sort = TRUE) %>%
+   ggplot(aes(fct_reorder(title,n), n)) +
+   geom_col(fill = "#773232") +
+   coord_flip() +
+   labs(x = "title") +
+   ggtitle("Top titles in Titanic test datset")
```



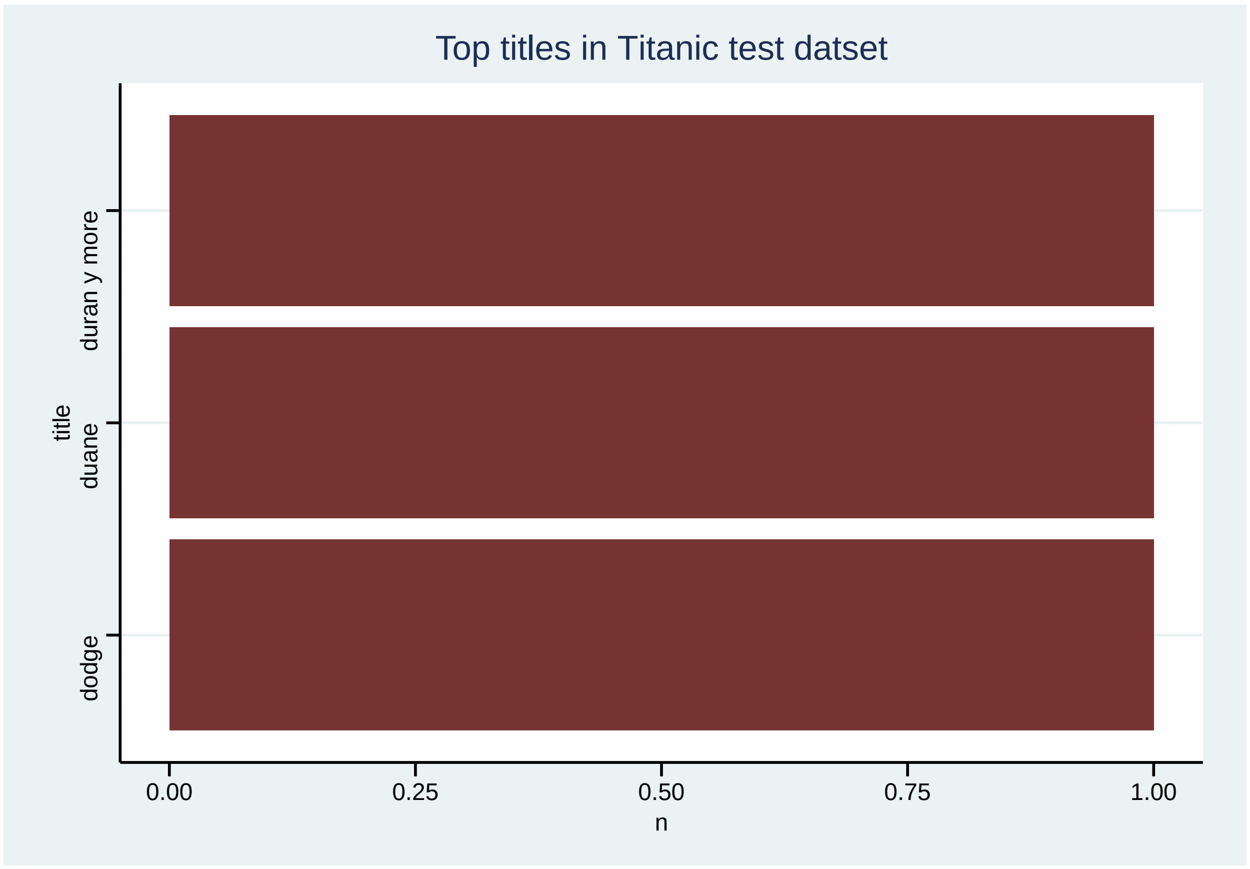
c. Extract family names from `Names` and plot the 10 most frequent family names. Remember to extract the pattern to include any kinds of characters (.) that are 1 character or more in length (+) and are followed by a comma(?=\,). What is/are the most popular last name/names?

```
> reg <- ".+(?=,)"
>
>
> df %>%
+   mutate(family = str_extract(str_to_lower(Name), reg)) %>%
+   count(family, sort = TRUE) %>%
+   top_n(10) %>%
+   ggplot(aes(fct_reorder(family,n), n)) +
+   geom_col(fill = "#773232") +
+   coord_flip() +
+   labs(x = "title") +
+   ggtitle("Top titles in Titanic test datset")
```



d. Write a function that plots the top 10 last names of passengers beginning and ending with certain letters passed as an argument to the function. Pass the `data tibble`, `letter corresponding to the beginning of the last name`, and `letter corresponding to the ending of the last name` as arguments to the function, and a plot (a distribution plot depicting the last names and their count) as the output of the function.

```
> plot_top_ten <- function(data = df, first, last){
+   reg1 <- ".+(?=,)"
+   reg2 <- str_glue("^[",{first}, "].*[",{last},"]$")
+   p1 <- df %>% na.omit() %>%
+   mutate(family = str_extract(str_to_lower(Name), reg1)) %>%
+   count(family, sort = TRUE) %>%
+   filter(str_detect(family, reg2)) %>%
+   top_n(10) %>%
+   ggplot(aes(fct_reorder(family,n), n)) +
+   geom_col(fill = "#773232") +
+   coord_flip() +
+   labs(x = "title") +
+   ggtitle("Top titles in Titanic test datset")
+ p1
+ }
>
> plot_top_ten(data = df, first = "Dd" , last = "e")
```


Top titles in Titanic test datset
title
duran y more
duane
dodge
n