Lab 2 Notebook

Table of contents

## Lab 2

In this class, we will go through the dplyr package in R.

## Introduction to dplyr

### Pipe Operator

In computation, we often have to chain multiple operations together. Consider a toy example. We want to calculate:

Try and write it as is in the following code chunk:

atan(sin(exp(gamma(1))))

[1] 0.3897659

However, as you may notice, this is difficult to read and error-prone (imagine counting parentheses whenever you write codes!).

In mathematics, we chain operations by composition:

In R specifically, we have the pipe operator %>%, provided by the magrittr package. It allows you write the above as:

library(magrittr)  
x <- 1  
x %>% gamma %>% exp %>% sin %>% atan

[1] 0.3897659

The pipe %>% takes the expression on its LHS, and pass it as the FIRST argument to the function on the RHS.

It’s helpful to read %>% as “and then”. We applied the gamma function, and then the exponential function, and then the sine function, so on and so forth . . .

You can also supply additional arguments for piped functions (that the pipe feed into). As an example:

x <- c(1, 2, 3, NA)  
x %>% max(na.rm=T) #is the same as

[1] 3

max(x,na.rm=T)

[1] 3

Amongst the readers familiar with Object Oriented Programming, compare this with method chaining techniques.

As of the time of writing, R is considering implementing a native pipe operator.

### Tidyverse collection of packages

* From the tidyverse website: The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.
* tidyverse packages help to steamline the data cleansing and exploring process and also smooth out some quirks of R.
* It is not a single package but a collection of packages. We will introduce them judiciously as we need them.
* dplyr (pronounced d-plier) is a package designed for cleansing and manipulating a data set. It is an amalgamation of “data frames” and the package plyr. Pliers, of course, are handy tools to have.
* We will look at the 5 main functions: mutate, select, filter, group\_by and arrange. For full potential of the package, you are encouraged to look at its documentation.
* Although not compulsory, you are encouraged to try recreating all examples in these notes in base R.

### Dplyr - mutate

* The general syntax is: mutate(dataset, output\_column = f(input\_columns))
* output\_column will be added in as a new column in the dataset. If there is already a column named output\_column, it will be overwritten.
* f is a function returning a vector of the same length. The length preservation is important - since the output will be pushed into the data frame
* As demonstration, we will consider the two built-in data sets in R: mtcars and iris. For demo purposes, there are no benefits gained from working on a large data sets. Worry not, you will be given a “proper” data set to work with.

Let’s take a look at the mtcars as an example:

head(mtcars)

mpg cyl disp hp drat wt qsec vs am gear carb  
Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4  
Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4  
Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1  
Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1  
Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2  
Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1

* The row names are not actually part of the data frame. It is stored as an additional attributes. The technical implication is that this information might be lost if fed to a function that doesn’t preserve attributes. For instance:

# Load the 'dplyr' package. The 'warn.conflicts = FALSE' option ensures that   
# no warning messages are printed when there are naming conflicts with other packages.  
library(dplyr, warn.conflicts = FALSE)

Warning: package 'dplyr' was built under R version 4.2.3

# 'as\_tibble' converts the given dataset (in this case, 'mtcars') into a tibble,   
# which is a modern version of data frames in R provided by the 'tibble' package.   
# Tibbles are more user-friendly and have some advantages over traditional data frames.  
as\_tibble(mtcars) %>%  
# Display the first 6 rows of the resulting tibble.  
head()

# A tibble: 6 × 11  
 mpg cyl disp hp drat wt qsec vs am gear carb  
 <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
1 21 6 160 110 3.9 2.62 16.5 0 1 4 4  
2 21 6 160 110 3.9 2.88 17.0 0 1 4 4  
3 22.8 4 108 93 3.85 2.32 18.6 1 1 4 1  
4 21.4 6 258 110 3.08 3.22 19.4 1 0 3 1  
5 18.7 8 360 175 3.15 3.44 17.0 0 0 3 2  
6 18.1 6 225 105 2.76 3.46 20.2 1 0 3 1

**Q: What does warn.conflicts - FALSE and as\_tibble mean? Comment your description in the code above.**

* We want to create a new column in the data frame that stores the column name. This is done simply by:

mutate(mtcars, modelNames = rownames(mtcars)) %>% head(n=2)

mpg cyl disp hp drat wt qsec vs am gear carb modelNames  
Mazda RX4 21 6 160 110 3.9 2.620 16.46 0 1 4 4 Mazda RX4  
Mazda RX4 Wag 21 6 160 110 3.9 2.875 17.02 0 1 4 4 Mazda RX4 Wag

Within a function call, <- and = are not equivalent. Strange behavior might result. Please copy the code above and see what happens :)

# mutate(mtcars, modelNames <- rownames(mtcars)) %>% head(n=2)  
  
# Here, you've used the assignment operator (<-) instead of the equals sign. What this does is it assigns the row names of the mtcars dataset to a new variable modelNames in the global environment, not within the mutate function. As a result, the mutate function doesn't actually add a new column, and you'll just get the original mtcars dataset without any modifications.  
#   
# Using <- within function calls like this can lead to confusing outcomes and is generally considered bad practice. Always use = when specifying arguments or making assignments within function calls.

* mutate and other dplyr functions are designed to work seamlessly with the pipe operator - so let’s do just that.
* For another example, suppose that you suspect that mpg is a linear function of the square root of disp. Let us create a column for that:

mtcars %>% mutate(root\_disp = sqrt(disp))

mpg cyl disp hp drat wt qsec vs am gear carb  
Mazda RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4  
Mazda RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4  
Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1  
Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1  
Hornet Sportabout 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2  
Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1  
Duster 360 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4  
Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2  
Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2  
Merc 280 19.2 6 167.6 123 3.92 3.440 18.30 1 0 4 4  
Merc 280C 17.8 6 167.6 123 3.92 3.440 18.90 1 0 4 4  
Merc 450SE 16.4 8 275.8 180 3.07 4.070 17.40 0 0 3 3  
Merc 450SL 17.3 8 275.8 180 3.07 3.730 17.60 0 0 3 3  
Merc 450SLC 15.2 8 275.8 180 3.07 3.780 18.00 0 0 3 3  
Cadillac Fleetwood 10.4 8 472.0 205 2.93 5.250 17.98 0 0 3 4  
Lincoln Continental 10.4 8 460.0 215 3.00 5.424 17.82 0 0 3 4  
Chrysler Imperial 14.7 8 440.0 230 3.23 5.345 17.42 0 0 3 4  
Fiat 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1  
Honda Civic 30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2  
Toyota Corolla 33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1  
Toyota Corona 21.5 4 120.1 97 3.70 2.465 20.01 1 0 3 1  
Dodge Challenger 15.5 8 318.0 150 2.76 3.520 16.87 0 0 3 2  
AMC Javelin 15.2 8 304.0 150 3.15 3.435 17.30 0 0 3 2  
Camaro Z28 13.3 8 350.0 245 3.73 3.840 15.41 0 0 3 4  
Pontiac Firebird 19.2 8 400.0 175 3.08 3.845 17.05 0 0 3 2  
Fiat X1-9 27.3 4 79.0 66 4.08 1.935 18.90 1 1 4 1  
Porsche 914-2 26.0 4 120.3 91 4.43 2.140 16.70 0 1 5 2  
Lotus Europa 30.4 4 95.1 113 3.77 1.513 16.90 1 1 5 2  
Ford Pantera L 15.8 8 351.0 264 4.22 3.170 14.50 0 1 5 4  
Ferrari Dino 19.7 6 145.0 175 3.62 2.770 15.50 0 1 5 6  
Maserati Bora 15.0 8 301.0 335 3.54 3.570 14.60 0 1 5 8  
Volvo 142E 21.4 4 121.0 109 4.11 2.780 18.60 1 1 4 2  
 root\_disp  
Mazda RX4 12.649111  
Mazda RX4 Wag 12.649111  
Datsun 710 10.392305  
Hornet 4 Drive 16.062378  
Hornet Sportabout 18.973666  
Valiant 15.000000  
Duster 360 18.973666  
Merc 240D 12.111978  
Merc 230 11.865918  
Merc 280 12.946042  
Merc 280C 12.946042  
Merc 450SE 16.607227  
Merc 450SL 16.607227  
Merc 450SLC 16.607227  
Cadillac Fleetwood 21.725561  
Lincoln Continental 21.447611  
Chrysler Imperial 20.976177  
Fiat 128 8.871302  
Honda Civic 8.700575  
Toyota Corolla 8.432082  
Toyota Corona 10.959015  
Dodge Challenger 17.832555  
AMC Javelin 17.435596  
Camaro Z28 18.708287  
Pontiac Firebird 20.000000  
Fiat X1-9 8.888194  
Porsche 914-2 10.968136  
Lotus Europa 9.751923  
Ford Pantera L 18.734994  
Ferrari Dino 12.041595  
Maserati Bora 17.349352  
Volvo 142E 11.000000

* The input part can contain multiple columns, as in the following example. The calculations are nonsensical, but they demonstrate the technique at a mechanical level:

mtcars %>% mutate(outputCol = mpg + cyl\*disp)

mpg cyl disp hp drat wt qsec vs am gear carb  
Mazda RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4  
Mazda RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4  
Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1  
Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1  
Hornet Sportabout 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2  
Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1  
Duster 360 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4  
Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2  
Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2  
Merc 280 19.2 6 167.6 123 3.92 3.440 18.30 1 0 4 4  
Merc 280C 17.8 6 167.6 123 3.92 3.440 18.90 1 0 4 4  
Merc 450SE 16.4 8 275.8 180 3.07 4.070 17.40 0 0 3 3  
Merc 450SL 17.3 8 275.8 180 3.07 3.730 17.60 0 0 3 3  
Merc 450SLC 15.2 8 275.8 180 3.07 3.780 18.00 0 0 3 3  
Cadillac Fleetwood 10.4 8 472.0 205 2.93 5.250 17.98 0 0 3 4  
Lincoln Continental 10.4 8 460.0 215 3.00 5.424 17.82 0 0 3 4  
Chrysler Imperial 14.7 8 440.0 230 3.23 5.345 17.42 0 0 3 4  
Fiat 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1  
Honda Civic 30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2  
Toyota Corolla 33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1  
Toyota Corona 21.5 4 120.1 97 3.70 2.465 20.01 1 0 3 1  
Dodge Challenger 15.5 8 318.0 150 2.76 3.520 16.87 0 0 3 2  
AMC Javelin 15.2 8 304.0 150 3.15 3.435 17.30 0 0 3 2  
Camaro Z28 13.3 8 350.0 245 3.73 3.840 15.41 0 0 3 4  
Pontiac Firebird 19.2 8 400.0 175 3.08 3.845 17.05 0 0 3 2  
Fiat X1-9 27.3 4 79.0 66 4.08 1.935 18.90 1 1 4 1  
Porsche 914-2 26.0 4 120.3 91 4.43 2.140 16.70 0 1 5 2  
Lotus Europa 30.4 4 95.1 113 3.77 1.513 16.90 1 1 5 2  
Ford Pantera L 15.8 8 351.0 264 4.22 3.170 14.50 0 1 5 4  
Ferrari Dino 19.7 6 145.0 175 3.62 2.770 15.50 0 1 5 6  
Maserati Bora 15.0 8 301.0 335 3.54 3.570 14.60 0 1 5 8  
Volvo 142E 21.4 4 121.0 109 4.11 2.780 18.60 1 1 4 2  
 outputCol  
Mazda RX4 981.0  
Mazda RX4 Wag 981.0  
Datsun 710 454.8  
Hornet 4 Drive 1569.4  
Hornet Sportabout 2898.7  
Valiant 1368.1  
Duster 360 2894.3  
Merc 240D 611.2  
Merc 230 586.0  
Merc 280 1024.8  
Merc 280C 1023.4  
Merc 450SE 2222.8  
Merc 450SL 2223.7  
Merc 450SLC 2221.6  
Cadillac Fleetwood 3786.4  
Lincoln Continental 3690.4  
Chrysler Imperial 3534.7  
Fiat 128 347.2  
Honda Civic 333.2  
Toyota Corolla 318.3  
Toyota Corona 501.9  
Dodge Challenger 2559.5  
AMC Javelin 2447.2  
Camaro Z28 2813.3  
Pontiac Firebird 3219.2  
Fiat X1-9 343.3  
Porsche 914-2 507.2  
Lotus Europa 410.8  
Ford Pantera L 2823.8  
Ferrari Dino 889.7  
Maserati Bora 2423.0  
Volvo 142E 505.4

### Dplyr - select

* In insurance, you often have enormous data sets with tens or even hundreds of columns, many of which might be irrelevant.
* You can use the select function to retain only the desired columns.
* mtcars %>% select(cyl, disp, hp) %>% head(n = 2)
* cyl disp hp  
  Mazda RX4 6 160 110  
  Mazda RX4 Wag 6 160 110
* You can chain mutate and select by the pipe operator. It works because these functions take a data frame as input and return a data frame as output, which can then be fed into another data-frame-consuming function.
* mtcars %>% mutate(outputCol = mpg + cyl\*disp) %>% select(mpg,cyl,disp,outputCol)
* mpg cyl disp outputCol  
  Mazda RX4 21.0 6 160.0 981.0  
  Mazda RX4 Wag 21.0 6 160.0 981.0  
  Datsun 710 22.8 4 108.0 454.8  
  Hornet 4 Drive 21.4 6 258.0 1569.4  
  Hornet Sportabout 18.7 8 360.0 2898.7  
  Valiant 18.1 6 225.0 1368.1  
  Duster 360 14.3 8 360.0 2894.3  
  Merc 240D 24.4 4 146.7 611.2  
  Merc 230 22.8 4 140.8 586.0  
  Merc 280 19.2 6 167.6 1024.8  
  Merc 280C 17.8 6 167.6 1023.4  
  Merc 450SE 16.4 8 275.8 2222.8  
  Merc 450SL 17.3 8 275.8 2223.7  
  Merc 450SLC 15.2 8 275.8 2221.6  
  Cadillac Fleetwood 10.4 8 472.0 3786.4  
  Lincoln Continental 10.4 8 460.0 3690.4  
  Chrysler Imperial 14.7 8 440.0 3534.7  
  Fiat 128 32.4 4 78.7 347.2  
  Honda Civic 30.4 4 75.7 333.2  
  Toyota Corolla 33.9 4 71.1 318.3  
  Toyota Corona 21.5 4 120.1 501.9  
  Dodge Challenger 15.5 8 318.0 2559.5  
  AMC Javelin 15.2 8 304.0 2447.2  
  Camaro Z28 13.3 8 350.0 2813.3  
  Pontiac Firebird 19.2 8 400.0 3219.2  
  Fiat X1-9 27.3 4 79.0 343.3  
  Porsche 914-2 26.0 4 120.3 507.2  
  Lotus Europa 30.4 4 95.1 410.8  
  Ford Pantera L 15.8 8 351.0 2823.8  
  Ferrari Dino 19.7 6 145.0 889.7  
  Maserati Bora 15.0 8 301.0 2423.0  
  Volvo 142E 21.4 4 121.0 505.4
* In certain cases, you want to exclude columns from a data frame instead. Here’s an example.
* mtcars %>% select(!cyl) %>% head(n=3)
* mpg disp hp drat wt qsec vs am gear carb  
  Mazda RX4 21.0 160 110 3.90 2.620 16.46 0 1 4 4  
  Mazda RX4 Wag 21.0 160 110 3.90 2.875 17.02 0 1 4 4  
  Datsun 710 22.8 108 93 3.85 2.320 18.61 1 1 4 1
* If you want to exclude multiple columns, this doesn’t work mtcars %>% select(!cyl, !am)
* Either of these work though:
* mtcars %>% select(!cyl & !am)
* mpg disp hp drat wt qsec vs gear carb  
  Mazda RX4 21.0 160.0 110 3.90 2.620 16.46 0 4 4  
  Mazda RX4 Wag 21.0 160.0 110 3.90 2.875 17.02 0 4 4  
  Datsun 710 22.8 108.0 93 3.85 2.320 18.61 1 4 1  
  Hornet 4 Drive 21.4 258.0 110 3.08 3.215 19.44 1 3 1  
  Hornet Sportabout 18.7 360.0 175 3.15 3.440 17.02 0 3 2  
  Valiant 18.1 225.0 105 2.76 3.460 20.22 1 3 1  
  Duster 360 14.3 360.0 245 3.21 3.570 15.84 0 3 4  
  Merc 240D 24.4 146.7 62 3.69 3.190 20.00 1 4 2  
  Merc 230 22.8 140.8 95 3.92 3.150 22.90 1 4 2  
  Merc 280 19.2 167.6 123 3.92 3.440 18.30 1 4 4  
  Merc 280C 17.8 167.6 123 3.92 3.440 18.90 1 4 4  
  Merc 450SE 16.4 275.8 180 3.07 4.070 17.40 0 3 3  
  Merc 450SL 17.3 275.8 180 3.07 3.730 17.60 0 3 3  
  Merc 450SLC 15.2 275.8 180 3.07 3.780 18.00 0 3 3  
  Cadillac Fleetwood 10.4 472.0 205 2.93 5.250 17.98 0 3 4  
  Lincoln Continental 10.4 460.0 215 3.00 5.424 17.82 0 3 4  
  Chrysler Imperial 14.7 440.0 230 3.23 5.345 17.42 0 3 4  
  Fiat 128 32.4 78.7 66 4.08 2.200 19.47 1 4 1  
  Honda Civic 30.4 75.7 52 4.93 1.615 18.52 1 4 2  
  Toyota Corolla 33.9 71.1 65 4.22 1.835 19.90 1 4 1  
  Toyota Corona 21.5 120.1 97 3.70 2.465 20.01 1 3 1  
  Dodge Challenger 15.5 318.0 150 2.76 3.520 16.87 0 3 2  
  AMC Javelin 15.2 304.0 150 3.15 3.435 17.30 0 3 2  
  Camaro Z28 13.3 350.0 245 3.73 3.840 15.41 0 3 4  
  Pontiac Firebird 19.2 400.0 175 3.08 3.845 17.05 0 3 2  
  Fiat X1-9 27.3 79.0 66 4.08 1.935 18.90 1 4 1  
  Porsche 914-2 26.0 120.3 91 4.43 2.140 16.70 0 5 2  
  Lotus Europa 30.4 95.1 113 3.77 1.513 16.90 1 5 2  
  Ford Pantera L 15.8 351.0 264 4.22 3.170 14.50 0 5 4  
  Ferrari Dino 19.7 145.0 175 3.62 2.770 15.50 0 5 6  
  Maserati Bora 15.0 301.0 335 3.54 3.570 14.60 0 5 8  
  Volvo 142E 21.4 121.0 109 4.11 2.780 18.60 1 4 2
* mtcars %>% select(!cyl) %>% select(!am)
* mpg disp hp drat wt qsec vs gear carb  
  Mazda RX4 21.0 160.0 110 3.90 2.620 16.46 0 4 4  
  Mazda RX4 Wag 21.0 160.0 110 3.90 2.875 17.02 0 4 4  
  Datsun 710 22.8 108.0 93 3.85 2.320 18.61 1 4 1  
  Hornet 4 Drive 21.4 258.0 110 3.08 3.215 19.44 1 3 1  
  Hornet Sportabout 18.7 360.0 175 3.15 3.440 17.02 0 3 2  
  Valiant 18.1 225.0 105 2.76 3.460 20.22 1 3 1  
  Duster 360 14.3 360.0 245 3.21 3.570 15.84 0 3 4  
  Merc 240D 24.4 146.7 62 3.69 3.190 20.00 1 4 2  
  Merc 230 22.8 140.8 95 3.92 3.150 22.90 1 4 2  
  Merc 280 19.2 167.6 123 3.92 3.440 18.30 1 4 4  
  Merc 280C 17.8 167.6 123 3.92 3.440 18.90 1 4 4  
  Merc 450SE 16.4 275.8 180 3.07 4.070 17.40 0 3 3  
  Merc 450SL 17.3 275.8 180 3.07 3.730 17.60 0 3 3  
  Merc 450SLC 15.2 275.8 180 3.07 3.780 18.00 0 3 3  
  Cadillac Fleetwood 10.4 472.0 205 2.93 5.250 17.98 0 3 4  
  Lincoln Continental 10.4 460.0 215 3.00 5.424 17.82 0 3 4  
  Chrysler Imperial 14.7 440.0 230 3.23 5.345 17.42 0 3 4  
  Fiat 128 32.4 78.7 66 4.08 2.200 19.47 1 4 1  
  Honda Civic 30.4 75.7 52 4.93 1.615 18.52 1 4 2  
  Toyota Corolla 33.9 71.1 65 4.22 1.835 19.90 1 4 1  
  Toyota Corona 21.5 120.1 97 3.70 2.465 20.01 1 3 1  
  Dodge Challenger 15.5 318.0 150 2.76 3.520 16.87 0 3 2  
  AMC Javelin 15.2 304.0 150 3.15 3.435 17.30 0 3 2  
  Camaro Z28 13.3 350.0 245 3.73 3.840 15.41 0 3 4  
  Pontiac Firebird 19.2 400.0 175 3.08 3.845 17.05 0 3 2  
  Fiat X1-9 27.3 79.0 66 4.08 1.935 18.90 1 4 1  
  Porsche 914-2 26.0 120.3 91 4.43 2.140 16.70 0 5 2  
  Lotus Europa 30.4 95.1 113 3.77 1.513 16.90 1 5 2  
  Ford Pantera L 15.8 351.0 264 4.22 3.170 14.50 0 5 4  
  Ferrari Dino 19.7 145.0 175 3.62 2.770 15.50 0 5 6  
  Maserati Bora 15.0 301.0 335 3.54 3.570 14.60 0 5 8  
  Volvo 142E 21.4 121.0 109 4.11 2.780 18.60 1 4 2
* It is difficult to explain why mtcars%>%select(!cyl,!am) doesn’t work without digging into the mechanics of tidyselect. We shall not do so.
* These methods require one to explicitly name the columns. It is possible to select/deselect columns based on pattern matching or regular expression. See this for helper functions.

### Dplyr - filter

* This allows you to select a subset of the original data set satisfying a certain condition.
* As an example, let us select the rows of mtcars where cyl has the value 6.
* mtcars %>% filter(cyl == 6) %>% head()
* mpg cyl disp hp drat wt qsec vs am gear carb  
  Mazda RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4  
  Mazda RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4  
  Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1  
  Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1  
  Merc 280 19.2 6 167.6 123 3.92 3.440 18.30 1 0 4 4  
  Merc 280C 17.8 6 167.6 123 3.92 3.440 18.90 1 0 4 4
* You can also combine multiple conditions using Boolean algebra. For example:
* mtcars %>% filter(cyl == 6 & mpg > 15) %>% head()
* mpg cyl disp hp drat wt qsec vs am gear carb  
  Mazda RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4  
  Mazda RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4  
  Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1  
  Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1  
  Merc 280 19.2 6 167.6 123 3.92 3.440 18.30 1 0 4 4  
  Merc 280C 17.8 6 167.6 123 3.92 3.440 18.90 1 0 4 4
* Since the “and” operator is used so often, filter has its own built-in syntax, where you can supply an arbitrary number of conditions:
* mtcars %>% filter(cyl == 6, mpg > 15) %>% head()
* mpg cyl disp hp drat wt qsec vs am gear carb  
  Mazda RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4  
  Mazda RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4  
  Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1  
  Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1  
  Merc 280 19.2 6 167.6 123 3.92 3.440 18.30 1 0 4 4  
  Merc 280C 17.8 6 167.6 123 3.92 3.440 18.90 1 0 4 4
* The original code combines the conditions cyl == 6 AND mpg > 15 and filters based on this one condition. The second version filters twice, based on two conditions. The desired effect is the same in the end.

### Dplyr::group\_by and dplyr::summarise

**Q: What does :: mean in R?**

* An Extremely powerful techniques when data entires can be categorised into groups.
* For our example, consider the iris dataset. Each row is an observation from a flower, which can be one of the three species: vetosa, versicolor, and virginica.
* Suppose that we want to find the mean Sepal.Length of each group. This involves partitioning iris into three sub-datasets, and compute the mean Sepal.Length three times.
* A solution with dplyr can look like this:
* iris%>%group\_by(Species)%>%summarise(meanLength = mean(Sepal.Length))
* # A tibble: 3 × 2  
   Species meanLength  
   <fct> <dbl>  
  1 setosa 5.01  
  2 versicolor 5.94  
  3 virginica 6.59
* You can compute an arbitrary number of by-group summary statistics in one call to summarise, for instance:
* iris%>%group\_by(Species)%>%summarise(meanLength = mean(Sepal.Length),maxLength = max(Sepal.Length))
* # A tibble: 3 × 3  
   Species meanLength maxLength  
   <fct> <dbl> <dbl>  
  1 setosa 5.01 5.8  
  2 versicolor 5.94 7   
  3 virginica 6.59 7.9
* Note that group\_by partiions by unique values of the grouping column. Although it makes sense to group by a categorical column, we can equally well group by numeric columns.
* To demonstrate this, let’s consider the mtcars dataset again. The cyl column is a numeric column showing the number of cylinders in a vehicle, but we can treat it as a categorical variable for vehicle type.
* mtcars %>% group\_by(cyl) %>% summarise(avgDisp = mean(disp))
* # A tibble: 3 × 2  
   cyl avgDisp  
   <dbl> <dbl>  
  1 4 105.  
  2 6 183.  
  3 8 353.
* If you want to categorise your data at a finer level than offered by one categorical variable, we can group\_by multiple columns. As an example, given the number of cyl, we want to further distinguish between automatic (am == 0) and manual (am==1) vehicles.

```{r}  
mtcars %>% group\_by(cyl, am) %>% summarise(avgDisp = mean(disp))  
```

**Q: We get the following note `summarise()` has grouped output by 'cyl'. You can override using the `.groups` argument, please investigate what this means and comment in the code above.**

### Dplyr::arrange

* As the name suggests, this arranges the rows in a certain order. As an example, let’s arrange mtcars in the increasing order of mpg:
* mtcars %>% arrange(mpg) %>% head()
* mpg cyl disp hp drat wt qsec vs am gear carb  
  Cadillac Fleetwood 10.4 8 472 205 2.93 5.250 17.98 0 0 3 4  
  Lincoln Continental 10.4 8 460 215 3.00 5.424 17.82 0 0 3 4  
  Camaro Z28 13.3 8 350 245 3.73 3.840 15.41 0 0 3 4  
  Duster 360 14.3 8 360 245 3.21 3.570 15.84 0 0 3 4  
  Chrysler Imperial 14.7 8 440 230 3.23 5.345 17.42 0 0 3 4  
  Maserati Bora 15.0 8 301 335 3.54 3.570 14.60 0 1 5 8
* For descending order:
* mtcars %>% arrange(desc(mpg)) %>% head()
* mpg cyl disp hp drat wt qsec vs am gear carb  
  Toyota Corolla 33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1  
  Fiat 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1  
  Honda Civic 30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2  
  Lotus Europa 30.4 4 95.1 113 3.77 1.513 16.90 1 1 5 2  
  Fiat X1-9 27.3 4 79.0 66 4.08 1.935 18.90 1 1 4 1  
  Porsche 914-2 26.0 4 120.3 91 4.43 2.140 16.70 0 1 5 2
* You can also arrange by multiple columns, first by the number of cyl, then mpg:
* mtcars %>% arrange(cyl, mpg) %>% head()
* mpg cyl disp hp drat wt qsec vs am gear carb  
  Volvo 142E 21.4 4 121.0 109 4.11 2.780 18.60 1 1 4 2  
  Toyota Corona 21.5 4 120.1 97 3.70 2.465 20.01 1 0 3 1  
  Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1  
  Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2  
  Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2  
  Porsche 914-2 26.0 4 120.3 91 4.43 2.140 16.70 0 1 5 2

### Lexical Scoping & NSE (Optional)

* This section is optional, but it will help you understand more thoroughly how R looks for its variables, i.e. its scoping rules.
* Let’s have a look at the following example. I have selected only the first two columns to make the output more succinct.
* external = 1:32  
  mtcars%>%select(mpg,cyl)%>%mutate(out1 = cyl^2,out2=external^2)
* mpg cyl out1 out2  
  Mazda RX4 21.0 6 36 1  
  Mazda RX4 Wag 21.0 6 36 4  
  Datsun 710 22.8 4 16 9  
  Hornet 4 Drive 21.4 6 36 16  
  Hornet Sportabout 18.7 8 64 25  
  Valiant 18.1 6 36 36  
  Duster 360 14.3 8 64 49  
  Merc 240D 24.4 4 16 64  
  Merc 230 22.8 4 16 81  
  Merc 280 19.2 6 36 100  
  Merc 280C 17.8 6 36 121  
  Merc 450SE 16.4 8 64 144  
  Merc 450SL 17.3 8 64 169  
  Merc 450SLC 15.2 8 64 196  
  Cadillac Fleetwood 10.4 8 64 225  
  Lincoln Continental 10.4 8 64 256  
  Chrysler Imperial 14.7 8 64 289  
  Fiat 128 32.4 4 16 324  
  Honda Civic 30.4 4 16 361  
  Toyota Corolla 33.9 4 16 400  
  Toyota Corona 21.5 4 16 441  
  Dodge Challenger 15.5 8 64 484  
  AMC Javelin 15.2 8 64 529  
  Camaro Z28 13.3 8 64 576  
  Pontiac Firebird 19.2 8 64 625  
  Fiat X1-9 27.3 4 16 676  
  Porsche 914-2 26.0 4 16 729  
  Lotus Europa 30.4 4 16 784  
  Ford Pantera L 15.8 8 64 841  
  Ferrari Dino 19.7 6 36 900  
  Maserati Bora 15.0 8 64 961  
  Volvo 142E 21.4 4 16 1024
* When mutate evaluates the computations supplied to it, it will: Treat the columns of the master data frame as if they were variables, then If such a column cannot be found (e.g. the external variable above), it will search outside of the data frame for a variable whose name matches.
* This mechanism is called lexical scoping. We will not explore this further, but the interested students are suggested to consult [Advanced R by Wickham].
* Unfortunately this opens up the following issue, if you have another cyl variable defined outside the data frame, e.g: cyl <- 1:32.
* How to you target it in mutate? The easiest solution is to enforce some naming scheme to avoid this situation.
* Let’s take a look at a tangential, but related, example.There are several ways in which one can load a package:
* library(dplyr)  
  library("dplyr")
* library can interpret both a string of characters and the codes. If supply codes, library will first convert this into a character strings.
* In general, R can access the codes you type and convert them into strings. This behavior is called Non-Standard Evaluation (NSE).
* The situation is a bit more complicated, for example:
* dplyr <- "purrr"  
  library(dplyr)
* This seems rather ambiguous. What do you think R will do?
* NSE is employed extensively in R, and thoroughly in dplyr and tidyverse.
* Again, we will not explore this further. But if you’re not getting the expected results, scoping issues and NSE often are the source of errors. It is worth bearing in mind these concepts

## Case Study: A Motor Third-party Liability Policies

As an example, we look at a data set containing risk features and claim numbers collected for nearly 700,000 MTPL policies observed over a year.

We conduct an initial numerical analysis to prepare for regression modelling. The response variable for the regression is the number of claims of each policy during the exposure period. Among all the policies, 94.98% have no claims. 9 variables are selected to explain the response, including:

1. IDpol: policy ID.
2. ClaimNb: Number of claims during the exposure period
3. Exposure : The exposure period.
4. Area: The area code.
5. VehPower: The power of the car (ordered categorical).
6. VehAge: The vehicle age, in years
7. DrivAge: The driver age, in years
8. BonusMalus: Bonus/malus, between 50 and 350: <100 means bonus, >100 means malus in France.
9. VehBrand: The car brand (unordered categories).
10. VehGas: The car gas, Diesel or regular.
11. Density: The density of inhabitants (number of inhabitants per km2) in the city the driver of the car lives in.
12. Region: the policy regions in France (based on a standard French classification)

Visit this [link](https://www.openml.org/search?type=data&sort=runs&id=41214) and download the .csv spreadsheet. Install and load the package readr. Type ?readr::read\_csv to read its documentation. Use the following code to interactively load the data set. (Note: set session to current directory, save the data set in the same folder to run the code below). You can also find the freMTPL2freq.csv on my github website.

library(readr)

Warning: package 'readr' was built under R version 4.2.3

dat <- read\_csv("freMTPL2freq.csv")

Rows: 678013 Columns: 12  
── Column specification ────────────────────────────────────────────────────────  
Delimiter: ","  
chr (4): Area, VehBrand, VehGas, Region  
dbl (8): IDpol, ClaimNb, Exposure, VehPower, VehAge, DrivAge, BonusMalus, De...  
  
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.

Use the function str to have a quick overview of the dataset:

str(dat)

spc\_tbl\_ [678,013 × 12] (S3: spec\_tbl\_df/tbl\_df/tbl/data.frame)  
 $ IDpol : num [1:678013] 1 3 5 10 11 13 15 17 18 21 ...  
 $ ClaimNb : num [1:678013] 1 1 1 1 1 1 1 1 1 1 ...  
 $ Exposure : num [1:678013] 0.1 0.77 0.75 0.09 0.84 0.52 0.45 0.27 0.71 0.15 ...  
 $ Area : chr [1:678013] "D" "D" "B" "B" ...  
 $ VehPower : num [1:678013] 5 5 6 7 7 6 6 7 7 7 ...  
 $ VehAge : num [1:678013] 0 0 2 0 0 2 2 0 0 0 ...  
 $ DrivAge : num [1:678013] 55 55 52 46 46 38 38 33 33 41 ...  
 $ BonusMalus: num [1:678013] 50 50 50 50 50 50 50 68 68 50 ...  
 $ VehBrand : chr [1:678013] "B12" "B12" "B12" "B12" ...  
 $ VehGas : chr [1:678013] "Regular" "Regular" "Diesel" "Diesel" ...  
 $ Density : num [1:678013] 1217 1217 54 76 76 ...  
 $ Region : chr [1:678013] "R82" "R82" "R22" "R72" ...  
 - attr(\*, "spec")=  
 .. cols(  
 .. IDpol = col\_double(),  
 .. ClaimNb = col\_double(),  
 .. Exposure = col\_double(),  
 .. Area = col\_character(),  
 .. VehPower = col\_double(),  
 .. VehAge = col\_double(),  
 .. DrivAge = col\_double(),  
 .. BonusMalus = col\_double(),  
 .. VehBrand = col\_character(),  
 .. VehGas = col\_character(),  
 .. Density = col\_double(),  
 .. Region = col\_character()  
 .. )  
 - attr(\*, "problems")=<externalptr>

## Tasks

1. VehGas is a character column. For regressions - it should be a factor column. A factor vector is made of underlying numeric data (for mathematical categorisation) and factor labels (to retain qualitative data). Use mutate to convert the VehGas column to a factor column. *Hint: Consult ?factor*

* dat <- dat %>% mutate(VehGas = factor(VehGas))

1. Some entries have an excessively high ClaimNb. One possible reason is that they might be bulk policies. Similar issues are seen with Exposure. One crude way to address this is to bound the ClaimNb and Exposure columns from above by 4 and 1, respectively. *Hint: consult the documentation of the pmin and pmax functions in base R.*

* dat <- dat %>%  
   mutate(ClaimNb = pmin(ClaimNb, 4),   
   Exposure = pmin(Exposure, 1))

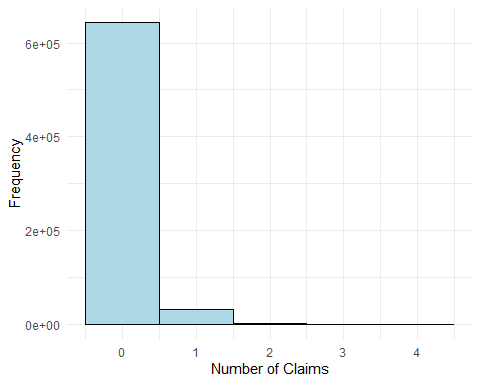
1. By using mutate, create a new column storing the logarithm of the Exposure column.

* dat <- dat %>% mutate(logExposure = log(Exposure))

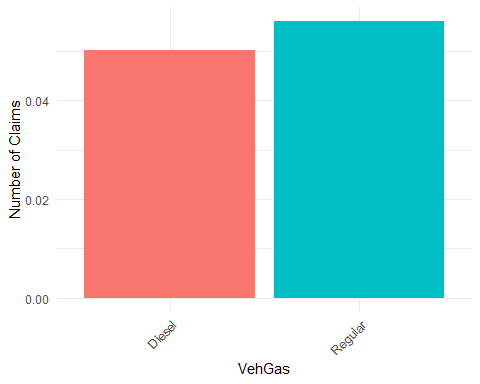
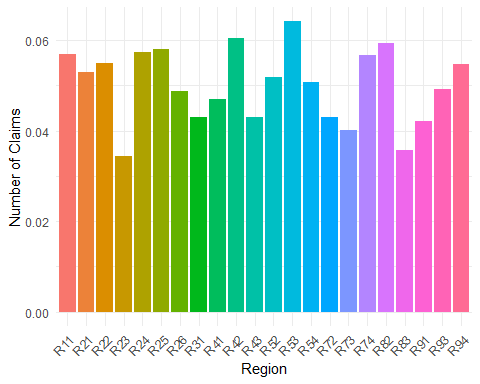
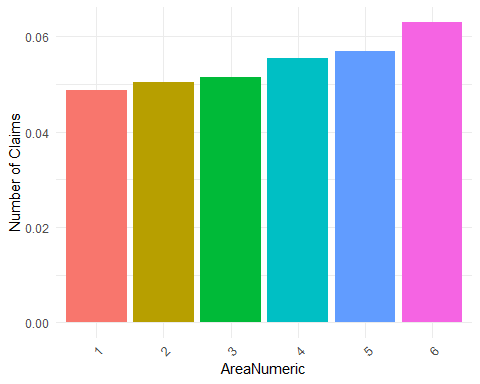
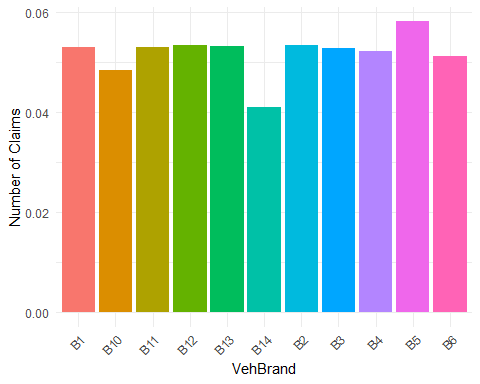
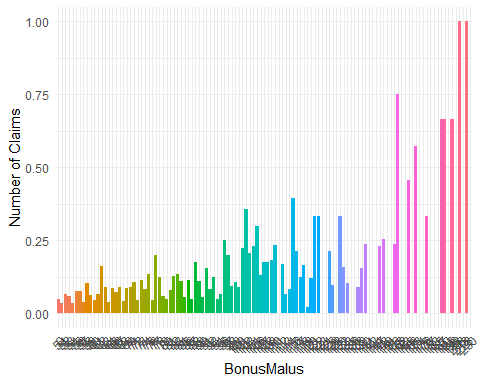
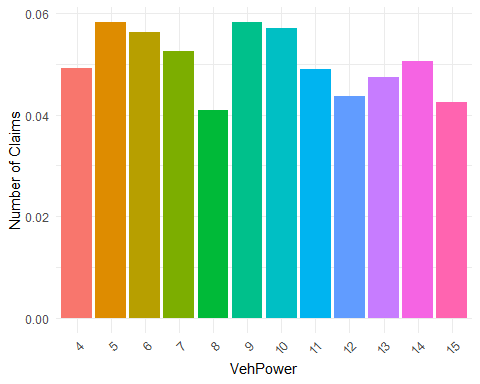
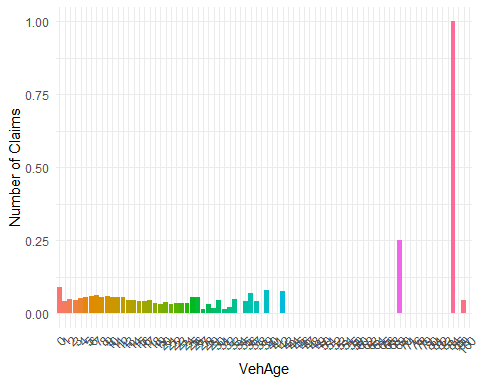
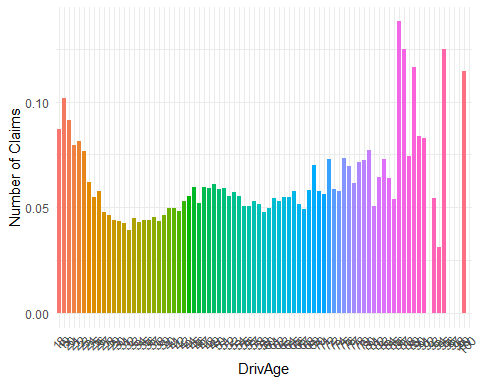
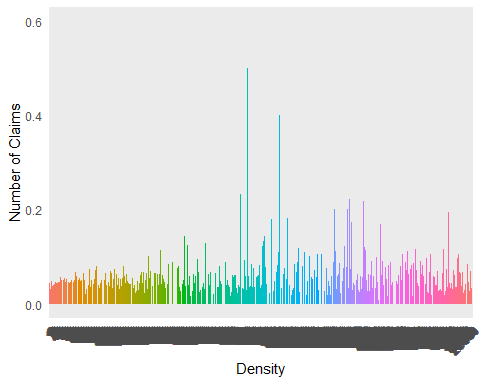
1. Create a new column that maps Area to numerical data. For instance, map “A” to 1, “B” to 2, etc. *Hint: write a function for this, then use mutate.*

* # Create a function to map Area to numerical data  
  map\_area\_to\_number <- function(area) {  
   areas <- c("A", "B", "C", "D", "E", "F", "G", "H", "I", "J")  
   return(which(areas == area))  
  }  
    
  dat <- dat %>% mutate(AreaNumeric = sapply(Area, map\_area\_to\_number))

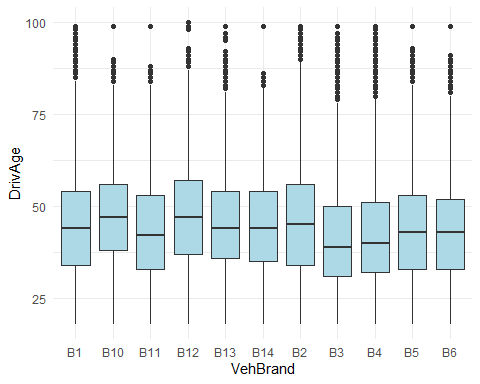
1. Now create a histogram of ClaimNb using ggplot2. As you may see, the y-axis has numbers in the form of 0e+00 to 6e+05. How can change this to numerical variables?

* library(ggplot2)
* Warning: package 'ggplot2' was built under R version 4.2.3
* # ClaimNb  
  fig1 <- ggplot(dat, aes(x = ClaimNb)) +  
   geom\_histogram(fill = "lightblue", color = "black", bins = 5) +  
   labs(x = "Number of Claims", y = "Frequency") +  
   theme\_minimal()  
  fig1
* 

1. Create a function which allows you to plot each explanatory variable according to the number of claims. (*Hint: Start with plotting the ClaimNb against a single covariate and see if you can create a function that you can use for all covariates.)* As you can see, some of the continuous variables do not look very nice. Can you transform them to categorical variables?

* # Define a function to create bar plots for categorical variables  
  plot\_categorical <- function(var) {  
   ggplot(dat %>%  
   group\_by(.data[[var]]) %>%  
   summarise(mean\_count = mean(ClaimNb, na.rm = TRUE)),  
   aes(x = factor(.data[[var]]), y = mean\_count, fill = factor(.data[[var]]))) +  
   geom\_col(show.legend = FALSE) +  
   labs(x = var,  
   y = "Number of Claims") +  
   theme\_minimal() +  
   theme(axis.text.x = element\_text(angle = 45, hjust = 1))  
  }  
    
  # List of categorical variables  
  categorical\_vars <- c("VehGas", "Region", "AreaNumeric",  
   "VehBrand", "BonusMalus","VehPower", "VehAge", "DrivAge",  
   "Density")  
    
  # Generate plots  
  categorical\_plots <- lapply(categorical\_vars, plot\_categorical)  
  categorical\_plots
* [[1]]
* 
* [[2]]
* 
* [[3]]
* 
* [[4]]
* 
* [[5]]
* 
* [[6]]
* 
* [[7]]
* 
* [[8]]
* 
* [[9]]
* 

1. Try and see if you can find any interactions between the variables.

* # For example, this is a plot DrivAge against VehBrand  
  ggplot(dat, aes(x = factor(VehBrand), y = DrivAge)) +  
   geom\_boxplot(fill = "lightblue") +  
   labs(x = "VehBrand",  
   y = "DrivAge") +  
   theme\_minimal()
* 

## Additional Exercises Functions in R

1. Write an R function, choose.members(n,c,p), that returns the number of ways to choose members from an organization of people to serve on an executive committee consisting of “named positions” (e.g., president, vice-president, treasurer, and so on…), and at-large members of equal rank, as the sample output below suggests. *Hint: Use the built-in functions: factorial() and choose(). First please use help and the name of each function, for example, help(factorial).*

* #Exercise 1  
    
  choose.members = function(n,c,p) {  
   return(factorial(n)\*choose(p,n)\*choose(p-n,c))  
  }

1. Write an R function, number.sequence(n), that returns a vector that has the form and has exactly terms, as the sample output below suggests.

* #Exercise 2  
    
  number.sequence = function(n) {  
   squares = (2:(n+1))^2  
   odds = 2\*(1:n)+1  
   return(squares \* odds)  
  }

1. Write a function is.even(n) in R that returns TRUE when is even and FALSE otherwise. Then, using is.even(n), write a function evens.in(v) that returns a vector comprised of the even integers in a vector of integers.

* #Exercise 3  
    
  is.even = function(n) {  
   return( n %% 2 == 0 )  
  }  
    
  evens.in = function(v) {  
   return(v[is.even(v)])  
  }

1. The sequence of consecutive differences of a given sequence of numbers, , is the sequence . Write a function consecutive.differences(v) that computes the consecutive differences of the elements of some vector .

* #Exercise 4   
    
  consecutive.differences = function(v) {  
   a = v[-1]  
   b = v[-length(v)]  
   return(a-b)  
  }

1. Write a function histogram.for.simulated.uniform.data(n,num.bars) that will simulate values following a uniform distribution and then plot the corresponding histogram. The histogram should have num.bars bars (although depending on the random data produced, it is possible that some interior bars might be zero-high).

* #Exercise 5  
  histogram.for.simulated.uniform.data = function(n,num.bars) {  
   data = runif(n)  
   bin.width = (max(data)-min(data))/num.bars  
   cols = c("red","white")  
   brks = seq(from=min(data),to=max(data),by=bin.width)  
   hist(data,breaks=brks,col=cols)  
  }