Sample midterm Programming 1: Sessions 1-4

Contents

1 Solved exercise on samples

Imagine you are in a mall and need to pick yogurt flavors

(1) Set the seed at 1.

```
set.seed(1)
```

(2) Generate a vector with 5 vanilla yogurts, 8 strawberry yogurts, and 10 lemon yogurts; use the function `rep` and put them together in a vector named `yogurts`

```
yogurts <- c(rep('vanilla', 5),
             rep('strawberry', 8),
             rep('lemon', 10))
```

(3) Pick ten yogurts randomly with no replacement.

```
cart <- sample(yogurts, 10, replace = FALSE)
```

(4) Count how many lemon yogurts you selected.

```
sum(cart=='lemon')
```

```
[1] 3
```

(5) Now add 3 chocolate yogurts to your vector of ten yogurts and select one yogurt randomly without replacement; which one did you get?

```
more_yogurts <- c(cart, rep('chocolate', 3))
sample(more_yogurts, 1, replace=FALSE)
```

```
[1] "lemon"
```

2 Solved exercise on strings

(1) Store the following quote from Benoit Mandelbrot into a string and call it `mandelbrot`:

> Time does not run in a straight line, like the markings on a wooden ruler. It stretches and shrinks, as if the ruler were made of balloon rubber. This is true in daily life: We perk up during high drama, nod off when bored. Markets do the same.

```
mandelbrot <- "Time does not run in a straight line, like the markings on a wooden ruler
```

(2) Load the `tidyverse` and count how many characters and words the quote has via the commands `str_length` and `str_count` (which requires the separator to be specified)

```
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.1 --

v ggplot2 3.3.5     v purrr   0.3.4
v tibble  3.1.5     v dplyr   1.0.7
v tidyr   1.1.4     v stringr 1.4.0
v readr   2.0.2     v forcats 0.5.1

-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

```
str_count(mandelbrot, ' ')
```

```
[1] 47
```

```
str_length(mandelbrot)
```

```
[1] 244
```

(3) Using `str_split`, take the words between positions 10 and 20; remember to use `[[]]` to access the content of a list.

```
str_split(mandelbrot, pattern = ' ')[[1]][35:44]
```

```
 [1] "We"     "perk"   "up"     "during" "high"   "drama," "nod"    "off"
 [9] "when"   "bored."
```

(4) Repeat the operation using the `word` command, which returns a string of words instead of a list of words.

```
sentence <- word(mandelbrot, start=35, end=44)
```

(5) In that last substring, substitute "nod off" by "sleep". Use `str_replace`.

```
str_replace(sentence, 'nod off', 'sleep')
```

```
[1] "We perk up during high drama, sleep when bored."
```

3 Solved exercise on conditional statements

The following is an example of a piecewise defined function:

$$\begin{cases} x^2 & x \leq 1 \\ 1 & -1 < x \leq 1 \\ x^2 & 1 < x \end{cases}$$

(1) Write an `if-else` clause that captures the logic of the function. Call the input `x` (give it whatever value you wish) and the output `y`

```
x <- 1
if (x<=-1){
  y <- x**2
} else if(x<=1){
  y <- 1
} else if (x<1){
  y <- x**2
}
```

(2) Wrap the `if-else` clause inside a function called `piece_fun`.

```
piece_fun <- function(x){
  if (x<=-1){
    y <- x**2
  } else if(x<=1){
    y <- 1
  } else if (x<1){
    y <- x**2
  }
}
```

(3) Modify the function so that you only use `ifelse`, and call it `piece_fun_vectorized`:

```
piece_fun_vectorized <- function(x){
  ifelse(x<=-1, x**2, ifelse(x<=1, 1, x**2))
}
```

(4) Generate a vector that goes from -5 to 5 in steps of 0.05 with the command `seq`. Call it `X`.

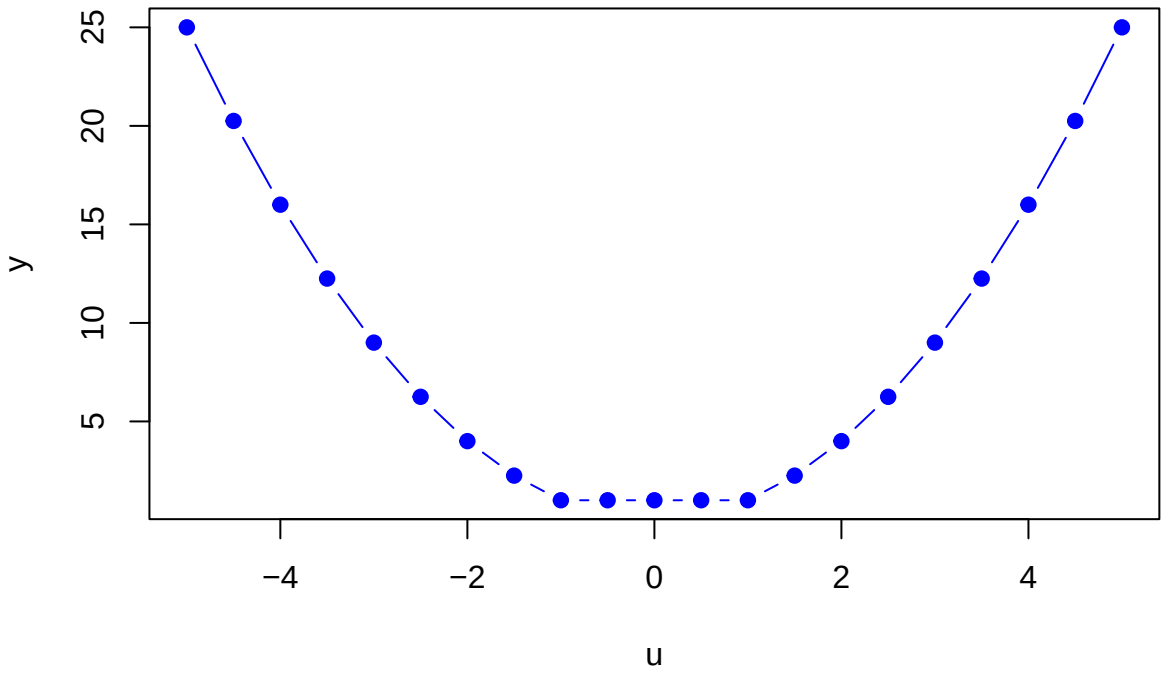
```
X <- seq(from=-5, to=5, by=0.5)
```

(4) Apply the function to `X` and call the output `Y`

```
Y <- piece_fun_vectorized(X)
```

(5) Plot `X` versus `Y` with the `plot` command (don't worry about the options of the `plot` command, they are just intended to make the function pretty and you could just do `plot(X,Y)`:

```
plot(X,Y, type='b', pch=19, col='blue', lty=1, xlab='u', ylab='y')
```



4 Solved exercise on tibbles

Load the tidyverse and access the `txhousing` dataset, which contains information on housing sales in Texas.

(1) Get the data for sales in the Bay Area in year 2000

```
filter(txhousing, city=='Bay Area', year=='2000')
```

```
# A tibble: 12 x 9
   city      year month sales   volume median listings inventory  date
   <chr>    <int> <int> <dbl>    <dbl>  <dbl>    <dbl>     <dbl> <dbl>
 1 Bay Area  2000     1   244 29322659 100700     1766       4.3 2000
 2 Bay Area  2000     2   375 48295141 110400     1817       4.4 2000.
 3 Bay Area  2000     3   391 48779462 113400     1830       4.5 2000.
 4 Bay Area  2000     4   421 52943752 107000     1845       4.6 2000.
```

```
 5 Bay Area  2000     5   533 70983815 115900     1827       4.4 2000.
 6 Bay Area  2000     6   561 75802627 122800     1874       4.5 2000.
 7 Bay Area  2000     7   449 59472007 114900     1876       4.6 2000.
 8 Bay Area  2000     8   471 66461908 119100     1869       4.6 2001.
 9 Bay Area  2000     9   397 51414827 111100     1842       4.5 2001.
10 Bay Area  2000    10   376 46689918 108800     1904       4.7 2001.
11 Bay Area  2000    11   322 42727036 116200     1863       4.6 2001.
12 Bay Area  2000    12   344 45935784 118500     1795       4.4 2001.
```

(2) Get only those sales that happened in Austin in April in either 2000 or 2001.

```
filter(txhousing, city=='Austin' & (month == 4) & (year == 2000 | year == 2001))
```

```
# A tibble: 2 x 9
  city    year month sales    volume median listings inventory  date
  <chr>  <int> <int> <dbl>     <dbl>  <dbl>    <dbl>     <dbl> <dbl>
1 Austin  2000     4  1556 289197960 136900     3192       2.1 2000.
2 Austin  2001     4  1579 302565123 151100     6800       4.4 2001.
```

(3) Sort the dataframe by descending order of `year` and then by ascending order of `month` (in a single statement).

```
arrange(txhousing, desc(year), month)
```

```
# A tibble: 8,602 x 9
   city                year month sales   volume median listings inventory  date
   <chr>              <int> <int> <dbl>    <dbl>  <dbl>    <dbl>     <dbl> <dbl>
 1 Abilene             2015     1   158   2.35e7 134100      801       4.4  2015
 2 Amarillo            2015     1   204   3.32e7 138500     1120       4.3  2015
 3 Arlington           2015     1   261   4.61e7 159700      552       1.3  2015
 4 Austin              2015     1  1656   5.12e8 237500     5567       2.2  2015
 5 Bay Area            2015     1   401   8.12e7 172200     1910       2.9  2015
 6 Beaumont            2015     1   151   2.17e7 122000     1558       7.3  2015
 7 Brazoria County     2015     1    69   1.04e7 146000      301       2.8  2015
 8 Brownsville         2015     1    41   5.40e6  97000      733      10.7  2015
 9 Bryan-College Sta~  2015     1   173   3.82e7 189300      988       3.8  2015
10 Collin County       2015     1   776   2.42e8 268000     1780       1.3  2015
# ... with 8,592 more rows
```

(4) Select only the columns `city` and `sales`:

```
select(txhousing, 'city', 'sales')
```

```
# A tibble: 8,602 x 2
   city    sales
   <chr>   <dbl>
 1 Abilene    72
 2 Abilene    98
 3 Abilene   130
 4 Abilene    98
 5 Abilene   141
 6 Abilene   156
 7 Abilene   152
 8 Abilene   131
 9 Abilene   104
10 Abilene   101
# ... with 8,592 more rows
```

(5) Generate a new column called `mean` that is equal to `volume` divided by `sales`

```
mutate(txhousing, mean=volume/sales)
```

```
# A tibble: 8,602 x 10
   city     year month sales   volume median listings inventory  date   mean
   <chr>   <int> <int> <dbl>    <dbl>  <dbl>    <dbl>     <dbl> <dbl>  <dbl>
 1 Abilene  2000     1    72  5380000  71400      701       6.3 2000  74722.
 2 Abilene  2000     2    98  6505000  58700      746       6.6 2000. 66378.
 3 Abilene  2000     3   130  9285000  58100      784       6.8 2000. 71423.
 4 Abilene  2000     4    98  9730000  68600      785       6.9 2000. 99286.
 5 Abilene  2000     5   141 10590000  67300      794       6.8 2000. 75106.
 6 Abilene  2000     6   156 13910000  66900      780       6.6 2000. 89167.
 7 Abilene  2000     7   152 12635000  73500      742       6.2 2000. 83125
 8 Abilene  2000     8   131 10710000  75000      765       6.4 2001. 81756.
 9 Abilene  2000     9   104  7615000  64500      771       6.5 2001. 73221.
10 Abilene  2000    10   101  7040000  59300      764       6.6 2001. 69703.
# ... with 8,592 more rows
```

(6) Get the average `median` (price of sale) by `city` by means of the `group_by` and `summarize` commands:

```
by_city <- group_by(txhousing, city)
summarise(by_city, avg_disp=mean(median))
```

```
# A tibble: 46 x 2
   city                 avg_disp
   <chr>                   <dbl>
 1 Abilene                98028.
 2 Amarillo              115122.
 3 Arlington             128314.
 4 Austin                181998.
 5 Bay Area              147132.
 6 Beaumont              115171.
 7 Brazoria County           NA
 8 Brownsville               NA
 9 Bryan-College Station 139472.
10 Collin County         201613.
# ... with 36 more rows
```