Lab: dplyr

36-600

Fall 2022

Data

We'll begin by importing some astronomical data from the 36-290 GitHub site. These data are stored in .Rdata format; such data are saved via `R`'s `save()` function and loaded via `R`'s `load()` function. One wrinkle here: the data are stored on the web, so we also have to apply the `url()` function.

```
file.path = "https://raw.githubusercontent.com/pefreeman/36-290/master/EXAMPLE_DATASETS/
BUZZARD/Buzzard_DC1.Rdata"
load(url(file.path))
rm(file.path)
set.seed(101)
s = sample(nrow(df),4000)
predictors = df[s,-c(7:14)]
response   = as.vector(df[s,14])
rm(df,s)
objects() # Shows the loaded variables. (Redundant with the Environment pane.)
```

```
## [1] "from"           "input"          "output_file"    "predictors"     "proc"
"response"       "rmd_args"
## [8] "self_contained" "to"
```

If everything loaded correctly, you should see two variables in your global environment: `predictors` and `response`. `predictors` is a data frame with 4000 rows and 6 columns, and `response` is a vector of length 4000, and it represents *redshift*, which you can think of as a directly observable proxy for the distance of a galaxy from the Earth. (After all, tape measures aren't going to help us here.)

Questions

Question 1

Apply the `dim()`, `nrow()`, `ncol()`, and `length()` functions to `predictors`, so as to build intuition about what these functions output. (Note: we are not using `dplyr` yet, just base `R` functions here.) Do you know why `length()` returns the value it does? Ask me or the TA if you do not. (But fill in an answer below regardless to reinforce the answer.)

```
dim(predictors)
```

```
## [1] 4000    6
```

```
nrow(predictors)
```

```
## [1] 4000
```

```
ncol(predictors)
```

```
## [1] 6
```

```
length(predictors)
```

```
## [1] 6
```

```
A data frame is a specific type of list; the elements of the list are each column. So le
ngth will return the number of columns.
```

Question 2

Display the names of each column of `predictors`.

```
names(predictors)
```

```
## [1] "u" "g" "r" "i" "z" "y"
```

Time for a digression.

A *magnitude* is a logarithmic measure of brightness that is calibrated such as to be approximately zero for the brightest stars in the night sky (Sirius, Vega, etc.). Every five-unit *increase* in magnitude is a factor of 100 *decrease* in brightness. So a magnitude of 20 means the object is 100 million times fainter than a star like Vega.

Magnitudes are generally measured in particular bands. Imagine that you put a filter in front of a telescope that only lets photons of certain wavelengths pass through. You can then assess the brightness of an object at just those wavelengths. So `u` represents a magnitude determined at ultraviolet wavelengths, with `g`, `r`, and `i` representing green, red, and infrared. (The `z` and `y` are a bit further into the infrared. The names don't represent words.)

So the predictor data consists of six magnitudes spanning from the near-UV to the near-IR.

Question 3

Use the base `R summary()` function to get a textual summary of the predictors. Do you notice anything strange? (Some values you might not expect?)

```
summary(predictors)
```

```
##        u               g               r               i               z
y
##  Min.   :19.55   Min.   :18.51   Min.   :18.00   Min.   :17.74   Min.   :17.56   Min.
:17.34
##  1st Qu.:26.37   1st Qu.:25.86   1st Qu.:25.10   1st Qu.:24.55   1st Qu.:24.21   1st
Qu.:23.86
##  Median :27.34   Median :26.94   Median :26.29   Median :25.74   Median :25.35   Medi
an :25.04
##  Mean   :33.06   Mean   :26.87   Mean   :25.88   Mean   :25.27   Mean   :24.98   Mean
:25.85
##  3rd Qu.:28.51   3rd Qu.:27.74   3rd Qu.:27.00   3rd Qu.:26.47   3rd Qu.:26.12   3rd
Qu.:25.89
##  Max.   :99.00   Max.   :99.00   Max.   :29.41   Max.   :27.00   Max.   :99.00   Max.
:99.00
```

```
The strangeness: the maximum value for some of the columns is 99.00.
```

dplyr

Below, we will practice using `dplyr` package functions to select rows and/or columns of a data frame. `dplyr` is part of the `tidyverse`, and is rapidly becoming the most oft-used way of transforming data. To learn more about "transformatory" functions, read through today's class notes, then through Chapter 5 of the online version of *R for Data Science* (see the syllabus for the link, or just Google for it). In short, you can

action	function
pick features by name	`select()`
pick observations by value	`filter()`
pick observations by category	`group_by()`
create new features	`mutate()`
reorder rows	`arrange()`
collapse rows to summaries	`summarize()`

A cool thing about `dplyr` is that you can use a piping operator (%>%) to have the output of one function be the input to the next. And you don't have to have only `dplyr` functions within the flow; for instance, you could pipe the first two rows your data frame to head:

```
suppressMessages(library(tidyverse))
head(predictors[,1:2])                          # base R
```

```
##              u       g
## 2873   23.5907 21.6827
## 108639 27.1252 26.2652
## 19665  27.4013 26.7235
## 87391  27.9354 27.0586
## 35772  27.7733 27.4076
## 46326  27.1742 26.9353
```

```
predictors %>% select(.,u,g) %>% head(.)       # dplyr
```

```
##              u       g
## 2873   23.5907 21.6827
## 108639 27.1252 26.2652
## 19665  27.4013 26.7235
## 87391  27.9354 27.0586
## 35772  27.7733 27.4076
## 46326  27.1742 26.9353
```

Let's do a few exercises here. Be sure to tap into, e.g., StackOverflow and *R for Data Science* for any help you may need.

Question 4

Grab all data for which `i` is less than 25 and `g` is greater than 22, and output in order of increasing `y`. (Remember: you combine conditions with & for "and" and | for "or".) Show only the first six lines of output. Note that `head()` by default shows the first six rows of the input data frame.

```
predictors %>% filter(.,i<25 & g>22) %>% arrange(.,y) %>% head(.)
```

```
##         u       g       r       i       z       y
## 1 24.1858 22.0232 20.3907 19.5766 19.1923 18.9443
## 2 24.3510 22.1041 20.4449 19.6847 19.2919 19.0380
## 3 24.4236 22.5979 21.0056 19.8446 19.4433 19.0918
## 4 24.8989 22.8297 21.2118 20.0455 19.6235 19.2732
## 5 23.7571 22.1921 20.6919 19.9896 19.6521 19.3608
## 6 24.1374 22.1437 20.5673 19.9751 19.6583 19.4766
```

Question 5

To get a quick, textual idea of how the data are distributed: select the `g` column, pipe it to the `round()` function, then pipe the output of `round()` to `table()`. You should notice something strange about the output…the same thing you should have noticed when answering Question 3.

```
predictors %>% select(.,g) %>% round(.) %>% table(.)
```

```
## g
##   19   20   21   22   23   24   25   26   27   28   29   30   31   32   34   99
##    5    5   27   46   99  206  388  722 1224  946  247   51   13    9    1   11
```

Time for another digression. Domain scientists are not bound by the `R` convention of using `NA` when data are "not available," or missing. Sometimes the domain scientists will tell you what they use in place of `NA`, sometimes not. In those latter cases, one can usual infer the values. Astronomers for some reason love using -99, -9, 99, etc., to represent missing values. The values of 99 in the table you generated in Question 5 actually represent missing data.

We could change the values of 99 to `NA`, but here we will just filter those rows with values of 99 out of the data frame.

Question 6

Use `filter()` to determine how many rows contain 99's. Since 99's can appear in different columns, you will need to combine conditions together with logical "and"s or "or"s. Note: only four of the six columns have 99's in them.

```
predictors %>% filter(.,u==99|g==99|z==99|y==99) %>% nrow(.)
```

```
## [1] 393
```

Question 7

Now repeat Question 6 (in a sense), and remove all the rows that contain 99's, saving the new data frame as `pred.new`. Hint that may help: the opposite of, e.g., `u==99 | g==99` is `u!=99 & g!=99`.

```
predictors %>% filter(.,u!=99&g!=99&z!=99&y!=99) -> pred.new
```

Question 8

Hold up…wait a minute…you just filtered the predictors data frame without filtering the response vector. A simple solution to this conundrum involves using base-`R` functionality you learned in Week 1: go back to the `predictors` data frame, determine which (hint: `which()`!) rows to keep, or exclude, using the logical operators in either Question 7, or Question 6, then apply the output to `response` directly to define `resp.new`.

Call me or the TA over (or come to office hours) if you need help with this.

```
w = which(predictors$u!=99&predictors$g!=99&predictors$z!=99&predictors$y!=99)
resp.new = response[w]

# or
#w = which(predictors$u==99|predictors$g==99|predictors$z==99|predictors$y==99)
#resp.new = response[-w]
```

The data we are working with have no factor variables, so I'm going to create one on the fly:

```
type = rep("FAINT",nrow(pred.new))
w = which(pred.new$i<25)
type[w] = "BRIGHT"
type = factor(type)
unique(type)
```

```
## [1] BRIGHT FAINT
## Levels: BRIGHT FAINT
```

```
pred.new = cbind(type,pred.new)
```

So I defined my factor variable using character strings, and then coerced the vector of strings into a factor variable with two levels. Note that by default, the levels order themselves into alphabetical order. You can override that default behavior if there is actually a natural ordering to your factor variables. See the documentation for `factor()` to see how to do that.

Question 9

Use `group_by()` and `summarize()` to determine the numbers of `BRIGHT` and `FAINT` galaxies. (If you are adventuresome: try implementing the `tally()` function instead of `summarize()`. For our purposes here, it's easier.)

```
pred.new %>% group_by(.,type) %>% summarize(.,Number=n())
```

```
## # A tibble: 2 × 2
##   type   Number
##   <fct>   <int>
## 1 BRIGHT   1295
## 2 FAINT    2312
```

```
# and/or
pred.new %>% group_by(.,type) %>% tally(.)
```

```
## # A tibble: 2 × 2
##   type       n
##   <fct>  <int>
## 1 BRIGHT  1295
## 2 FAINT   2312
```

Question 10

Repeat Question 9, but show the median value of the `u` magnitude instead of the numbers in each factor group. (You do need `summarize()` here.)

```
pred.new %>% group_by(.,type) %>% summarize(.,Median=median(u))
```

```
## # A tibble: 2 × 2
##   type   Median
##   <fct>   <dbl>
## 1 BRIGHT   25.9
## 2 FAINT    27.7
```

Time for yet another digression.

Magnitudes of galaxies at particular wavelengths are heavily influenced by two factors: physics (what is going on with the gas, dust, and stars within the galaxy itself), and distance (the further away a galaxy is, the less bright it tends to be, so the magnitude generally goes up). To attempt to mitigate (somewhat, not necessarily completely) the effect of distance, astronomers often use *colors*, which are differences in magnitude for two adjacent filters.

Question 11

Use `mutate()` to define two new columns for `pred.new`: `gr`, which would be the `g` magnitude minus the `r` magnitude, and `ri`, for `r` and `i`. Save your result.

```
pred.newer = pred.new %>% mutate(.,gr=g-r,ri=r-i)
```

Question 12

Are the mean values of `g-r` and `r-i` roughly the same for `BRIGHT` galaxies versus `FAINT` ones? Heck if I know. Use `dplyr` functions to attempt to answer this question.

```
pred.newer %>% group_by(.,type) %>% summarize(.,mean.gr=mean(gr),mean.ri=mean(ri))
```

```
## # A tibble: 2 × 3
##   type   mean.gr mean.ri
##   <fct>    <dbl>   <dbl>
## 1 BRIGHT   0.942   0.620
## 2 FAINT    0.694   0.558
```

Question 13

Let's go back to hypothesis testing for a moment. Let's extract two vectors of data from `pred.newer`:

```
gr.faint = pred.newer$gr[pred.newer$type=="FAINT"]
gr.bright = pred.newer$gr[pred.newer$type=="BRIGHT"]
```

The first vector has all the `g-r` colors for faint galaxies, and the second vector has all the ones for bright galaxies. The code here *should* be familiar given what we covered in Week 01, but if you are not sure what this code is doing, call me or the TA over.

Now: test the hypothesis that the means of the distributions that `gr.faint` and `gr.bright` are sampled from are the same. Use a two-sample t test. What do you conclude? Is a two-sample t test actually strictly appropriate here? (Maybe visualize the data in each vector to find out, with, e.g., the base `R hist()` function.) (Even if the answer is no, do the t test anyway.)

```
t.test(gr.faint,gr.bright)
```

```
##
##  Welch Two Sample t-test
##
## data:  gr.faint and gr.bright
## t = -14.389, df = 2979.8, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.2816093 -0.2140635
## sample estimates:
## mean of x mean of y
## 0.6942116 0.9420480
```

```
The p-value is incredibly small (effectively zero): we conclude that the mean of the dis
tribution of g-r colors for bright galaxies differs significantly from those for faint g
alaxies.

Strictly, the two-sample t-test assumes the means for each sample are normally distribut
ed. If you histogram the data, they appear to be kinda normally distributed, but not qui
te...particularly the bright data. But this is actually OK, because the sample sizes are
so large that the Central Limit Theorem takes over: the samples means will be, at least
approximately, normally distributed, so using the two-sample t-test here is fine.
```