In-Class Exercise 4

Nathan Van Ymeren

Preface:

In previous documents I reproduced the question text entirely. This time to make it clearer I'm going to omit the question text and just have some explanatory prose in each section.

First let's load the tidyverse and our dataset:

```r
library(tidyverse)
companies = readRDS("../ICE3/North American Stock Market 1994-2018.rds")
```

Question 1, 1.25 pts

Okay so right away I have a feeling this quiz is going to be an awful lot of `group_by()` which is annoying because underscores are awkward to type quickly. But I digress. The question asks us to compute aggregate values for all companies, per fiscal year, so that's our cue to group by `fyear`.

```r
q1 = companies %>%
  filter(!is.na(fyear)) %>%
  group_by(fyear) %>%
  mutate(mat = median(at, na.rm=TRUE)) %>%
  summarize(mmat = max(mat)) %>%
  filter(mmat == max(mmat)) %>%
  pull(fyear)
```

That's a long pipeline. What we did there is first remove the rows with fiscal years that were missing, then we subset based on fiscal year and calculate the median `at` for each fiscal year across all companies in that fiscal year. Then we add another variable called `mmat` (Maximum Median AT, because why not) and then filter for rows where `mmat` is equal to its maximum value, and then pull out the fiscal year, giving answer D:

```
2018
```

Question 2, 1.25 pts

Pretty straightforward. First we'll group by gvkey and then we'll compute a maximum asset value for each group. Then we will drop all rows where the maximum value is less than 100.

```
 q2 = companies %>%
  group_by(gvkey) %>%
  mutate(mat = max(at)) %>%
  filter(mat >= 100)

nrow(q2)
```

Interestingly enough, the answer is "none of the above":

```
153995
```

Question 3, 1.25 pts

This question asks us to run the following code and describe its contents/what it represents. I'll reproduce only the first few rows for brevity:

```
 new_dataset <-companies %>%
  filter(!is.na(fyear), !is.na(loc), !is.na(sale)) %>%
  group_by(fyear, loc) %>%
  summarise(max_sale = max(sale))

head(new_dataset)
```

Which looks something like this:

fyear	loc	max_sale
1994	ANT	142.949
1994	ARG	4195.119
1994	AUS	12754
1994	AUT	3.392
1994	BHS	69.694
1994	BLZ	23.707

The query is grouping by fiscal year and location, and then computing an aggregate maximum sales figure for each grouping. So I'd say it's "maximum sales, per fiscal year, per country", which is pretty close to what answer D describes.

Question 4, 1.25 pts

Okay so this one is long-winded but pretty straightforward if you work backwards:

1. Drop any rows that have missing values for country, sales, or fiscal year (so that means we'll be calling `filter(!is.na(whatever))` a bunch)
2. retain all the original values (so we'll start from `companies`) and `mutate` new variables into the dataframe before saving it to `q4`, and this implies no use of `select()`
3. the new variable we're going to create will be based on groupings of country and fiscal year (implying `group_by(loc, fyear)` or similar)

Looking through the available answers we can eliminate B because it filters for `is.na()` rather than `!is.na()`, and we can eliminate C because it's calling `select()`. We need to keep the new values and all the old ones, so we need to use `mutate`. To illustrate, we can see what happens if we run the code from D:

```r
  q4 <- companies %>%
   filter(!is.na(fyear), !is.na(loc), !is.na(sale)) %>%
   group_by(fyear,loc) %>%
   summarise(total_country_sales = sum(sale, na.rm = TRUE))

ncol(q4)
```

Notice that q4 has only 3 variables:

```
3
```

Whereas if we run the code from answer A:

```r
  q4 <- companies %>%
   filter(!is.na(fyear), !is.na(loc), !is.na(sale)) %>%
   group_by(fyear,loc) %>%
   mutate(total_country_sales = sum(sale, na.rm = TRUE))

ncol(q4)
```

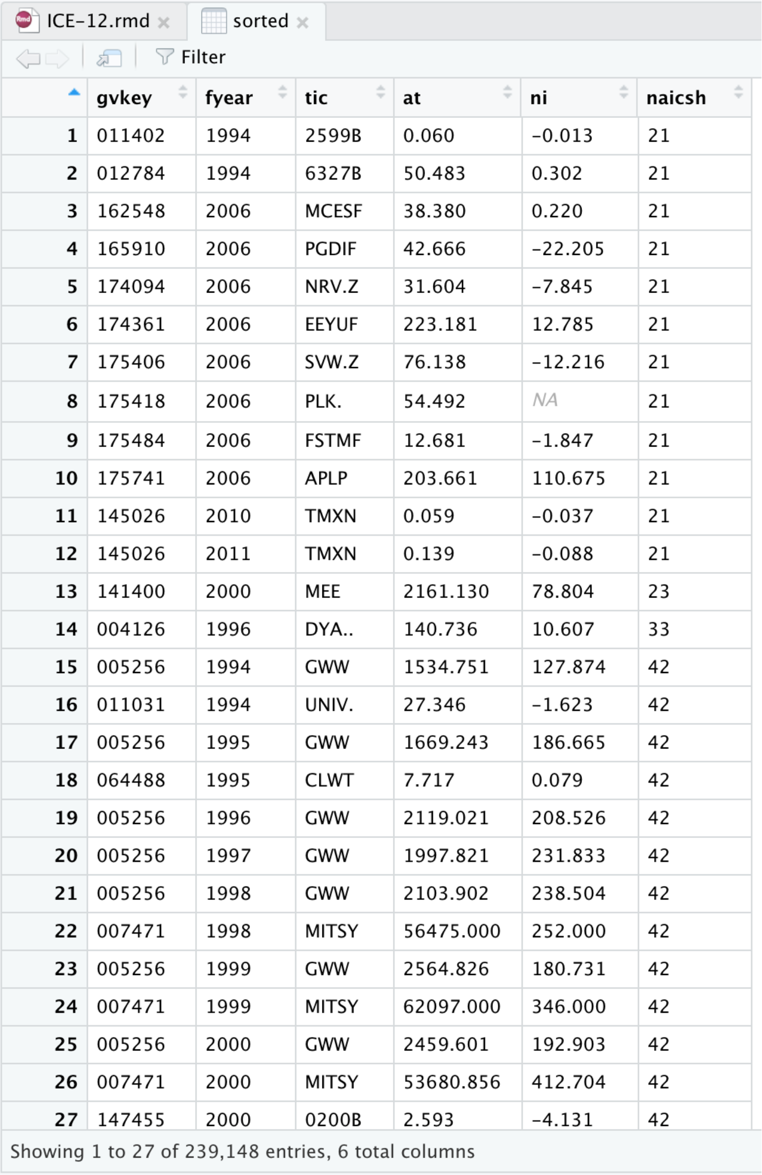
We can see that it has 42 variables, as the question implies it ought to:

```
42
```

Thus the answer is A.

Question 5, 1.25 pts

Okay so we run the code they give us and replace the commented section with something to produce something sorted as such:



	gvkey	fyear	tic	at	ni	naicsh
1	011402	1994	2599B	0.060	-0.013	21
2	012784	1994	6327B	50.483	0.302	21
3	162548	2006	MCESF	38.380	0.220	21
4	165910	2006	PGDIF	42.666	-22.205	21
5	174094	2006	NRV.Z	31.604	-7.845	21
6	174361	2006	EEYUF	223.181	12.785	21
7	175406	2006	SVW.Z	76.138	-12.216	21
8	175418	2006	PLK.	54.492	NA	21
9	175484	2006	FSTMF	12.681	-1.847	21
10	175741	2006	APLP	203.661	110.675	21
11	145026	2010	TMXN	0.059	-0.037	21
12	145026	2011	TMXN	0.139	-0.088	21
13	141400	2000	MEE	2161.130	78.804	23
14	004126	1996	DYA..	140.736	10.607	33
15	005256	1994	GWW	1534.751	127.874	42
16	011031	1994	UNIV.	27.346	-1.623	42
17	005256	1995	GWW	1669.243	186.665	42
18	064488	1995	CLWT	7.717	0.079	42
19	005256	1996	GWW	2119.021	208.526	42
20	005256	1997	GWW	1997.821	231.833	42
21	005256	1998	GWW	2103.902	238.504	42
22	007471	1998	MITSY	56475.000	252.000	42
23	005256	1999	GWW	2564.826	180.731	42
24	007471	1999	MITSY	62097.000	346.000	42
25	005256	2000	GWW	2459.601	192.903	42
26	007471	2000	MITSY	53680.856	412.704	42
27	147455	2000	0200B	2.593	-4.131	42

Showing 1 to 27 of 239,148 entries, 6 total columns

Looking at the image we can see it's pretty clearly sorted by `naicsh`, which is the North American Industry Classification System (NAICS) code identifier, in ascending order, but less obviously it also appears to be sorted by fiscal year in ascending order, within each NAICS code, and then on top of that it's sorted by `gvkey` within each fiscal year. So we'd sort first by `naicsh`, then by `fyear`, and then by `gvkey` in that order.

You could just run each code snippet to see which one produces output like the image but you can just look and see that only answer B will actually do a sort, subsort, and sub-subsort in the correct order. Answer A does the correct sorting but in the wrong order. Answer F might come close but because you're piping each sort operation into the next what it does is just sort the dataframe three times without doing sub-sorts. So, the answer is B.

Question 6, 1.25 pts

The default sort order is ascending, so what this code is doing is sorting by `a`, ascending, and then either sorting by `b` ascending or descending within each `a`. This implies:

1. if `a` has no duplicates then we can only sort `b` one way, since there's only one `b` for any given `a`, so we tick the box for answer A.
2. if `b` has all identical values, then since we're sorting `a` the same way each time, it there's only one way to sort `b` irrespective of how `a` is sorted, so we tick the box for answer D

The other answers are insufficient (this question reminds me of the GMAT, haha)

Question 7, 1.25 pts

Starting with `companies` we'll drop all rows that have `at < 100` or `sale < 100` and we'll drop any rows with missing employment, sales, or assets. Then after that we'll `group_by()` company (aka `gvkey`) and compute average employment per firm where `loc==USA` in fiscal years 2016, 2017, and 2018. The question asks how many firms would be included in this calculation. Let's find out. Reminder that filtering returns rows where the condition is true, so we need to invert some of these inequalities.

```
  q7 = companies %>%
    filter(!is.na(emp), !is.na(sale), !is.na(at)) %>%
    filter(at >= 100 | sale >= 100) %>%
    filter( (fyear <= 2018) & (fyear >= 2016) ) %>%
    filter(loc == "USA") %>%
    group_by(gvkey)

n_distinct(q7$gvkey)
```

The question asks how many companies there are, but because we're filtering by multiple fiscal years we'll end up with an inflated count if we just use `nrow()` so instead we'll count the number of unique `gvkey` values using `n_distinct()` which gives:

```
4181
```

Question 8, 1.25 pts

This question is worded very confusingly. I had no idea what it was even asking for, so thank you to my awesome classmates for interpreting. Apparently "average per firm" means "take an average of each firms' employment figures in the 3 year period, then average those all together". So, here that is:

```
q8 = q7 %>%
  mutate(avgemp = mean(emp)) %>%
  summarize(avgavg = mean(avgemp)) %>%
  summarize(avgavgavg = mean(avgavg))
```

So assuming that's what the question meant it's 10.4 thousand.

```
10.4146899465838
```