Lecture 5: Data Transformation

Data Transformation with dplyr

Yujin Jeong

STATS 195

dplyr

dplyr package

Let's start by loading the `dplyr` package:

```
library(dplyr)
```

The flights data set

We will work with the flights data set from the nycflights13 package.:

```
# install.packages("nycflights13")
library(nycflights13)
data(flights)
```

The flights data set

The data set contains ~336,000 flights that departed from New York City (all 3 airports) in 2013.

```
## tibble [336,776 × 19] (S3: tbl_df/tbl/data.frame)
##  $ year          : int [1:336776] 2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
##  $ month         : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
##  $ day           : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
##  $ dep_time      : int [1:336776] 517 533 542 544 554 554 555 557 557 558 ...
##  $ sched_dep_time: int [1:336776] 515 529 540 545 600 558 600 600 600 600 ...
##  $ dep_delay     : num [1:336776] 2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
##  $ arr_time      : int [1:336776] 830 850 923 1004 812 740 913 709 838 753 ...
##  $ sched_arr_time: int [1:336776] 819 830 850 1022 837 728 854 723 846 745 ...
##  $ arr_delay     : num [1:336776] 11 20 33 -18 -25 12 19 -14 -8 8 ...
##  $ carrier       : chr [1:336776] "UA" "UA" "AA" "B6" ...
##  $ flight        : int [1:336776] 1545 1714 1141 725 461 1696 507 5708 79 301 ...
##  $ tailnum       : chr [1:336776] "N14228" "N24211" "N619AA" "N804JB" ...
##  $ origin        : chr [1:336776] "EWR" "LGA" "JFK" "JFK" ...
##  $ dest          : chr [1:336776] "IAH" "IAH" "MIA" "BQN" ...
##  $ air_time      : num [1:336776] 227 227 160 183 116 150 158 53 140 138 ...
##  $ distance      : num [1:336776] 1400 1416 1089 1576 762 ...
##  $ hour          : num [1:336776] 5 5 5 5 6 5 6 6 6 6 ...
##  $ minute        : num [1:336776] 15 29 40 45 0 58 0 0 0 0 ...
##  $ time_hour     : POSIXct[1:336776], format: "2013-01-01 05:00:00" "2013-01-01 05:00:00" "2013-01-01 05:00:00" ...
```

dplyr verbs

These are the main five functions ("verbs") in dplyr. They are immensely helpful for data transformation in R. There are several other functions in dplyr that may be of use, some of which we will introduce as we go, some of which you can take a look at on your own. The dplyr cheat sheet is a great place to start.

- `` `filter()` ``
- `` `select()` ``
- `` `arrange()` ``
- `` `mutate()` ``
- `` `summarize()` ``

magrittr piping

As we start to do more complicated things in R, the `magrittr` package will be very helpful in keeping our code clean and easy to write/read. Take a look at how it works:

```
function1(x,y)

#vs

x %>% function1(y)
```

```
function3(function2(function1(x,y1),y2,z2),y3)

#vs

x %>% function1(y1) %>% function2(y2,z2) %>% function3(y3)

#or

x %>%
  function1(y1) %>%
  function2(y2,z2) %>%
  function3(y3)
```

1. filter

The `filter()` function is used to subset a data frame, retaining all rows that satisfy your conditions.

Since we are here in Stanford, we may only be interested in flights from NYC to SFO. We can use the `filter()` verb to achieve this:

```
flights %>% filter(dest == "SFO")
```

carrier <chr>	flight <int>	tailnum <chr>	origin <chr>	dest <chr>	air_time <dbl>	distance <dbl>	hour <dbl>	minute <dbl>	time_hour <S3: POSIXct>
UA	1124	N53441	EWR	SFO	361	2565	6	0	2013-01-01 06:00:00
UA	303	N532UA	JFK	SFO	366	2586	6	0	2013-01-01 06:00:00
DL	1865	N705TW	JFK	SFO	362	2586	7	0	2013-01-01 07:00:00
VX	11	N635VA	JFK	SFO	356	2586	7	30	2013-01-01 07:00:00
B6	643	N625JB	JFK	SFO	350	2586	7	37	2013-01-01 07:00:00
AA	59	N336AA	JFK	SFO	378	2586	7	45	2013-01-01 07:00:00
UA	1668	N24224	EWR	SFO	373	2565	7	46	2013-01-01 07:00:00
UA	223	N510UA	JFK	SFO	369	2586	8	0	2013-01-01 08:00:00
UA	1480	N76522	EWR	SFO	357	2565	8	17	2013-01-01 08:00:00
AA	179	N325AA	JFK	SFO	389	2586	10	30	2013-01-01 10:00:00

1-10 of 13,331 rows | 10-19 of 19 columns

Previous 1 2 3 4 5 6 ... 100 Next

1. filter

There are two other international airports near Stanford, San Jose International Airport ("SJC") and Oakland International Airport ("OAK"). So if we want to analyze flights that people take to get from NYC to Stanford, we should probably include these flights.

```
flights %>% filter(dest == "SFO" | dest == "SJC" | dest == "OAK")
```

carrier <chr>	flight <int>	tailnum <chr>	origin <chr>	dest <chr>	air_time <dbl>	distance <dbl>	hour <dbl>	minute <dbl>	time_hour <S3: POSIXct>
VX	27	N847VA	JFK	SFO	354	2586	16	30	2013-01-01 16:00:00
DL	31	N713TW	JFK	SFO	357	2586	17	0	2013-01-01 17:00:00
UA	512	N557UA	JFK	SFO	347	2586	17	29	2013-01-01 17:00:00
UA	1284	N14120	EWR	SFO	360	2565	17	18	2013-01-01 17:00:00
B6	173	N569JB	JFK	SJC	334	2569	18	15	2013-01-01 18:00:00
UA	389	N508UA	JFK	SFO	357	2586	18	45	2013-01-01 18:00:00
AA	177	N332AA	JFK	SFO	361	2586	17	45	2013-01-01 17:00:00
VX	29	N638VA	JFK	SFO	364	2586	18	50	2013-01-01 18:00:00
B6	91	N523JB	JFK	OAK	330	2576	18	45	2013-01-01 18:00:00
DL	853	N727TW	JFK	SFO	361	2586	19	0	2013-01-01 19:00:00

21-30 of 13,972 rows | 10-19 of 19 columns Previous 1 2 3 4 5 6 ... 100 Next

1. filter

We can also use the `%in%` function to similar effect:

```
flights %>% filter(dest %in% c("SFO","SJC","OAK"))
```

The command above filters the dataset and prints it out, but does not retain the output. To keep the extracted dataset for further analysis, we have to assign it to a variable:

```
Stanford <- flights %>% filter(dest %in% c("SFO","SJC","OAK"))
```

1. filter

We now have flights from NYC to SFO/SJC/OAK for the entire year. Let's say that I'm only interested in flights when school is in session (Sep - Jun). Since `month` is a numeric variable, we could do this:

```
Stanford %>% filter(month <= 6 | month >= 9)
```

or

```
Stanford %>% filter(month != 7 & month != 8)
```

2. select

Let's return to the `Stanford` dataset (i.e. all flights from NYC to SFO/SJC/OAK). Notice that we have a total of 19 variables. Sometimes our datasets will have hundreds or thousands of variables! Not all of them may be of interest to us. The `select()` function allows us to choose a subset of variables to form a smaller dataset.

```
Stanford %>% select(year, month, day)
```

year <int>	month <int>	day <int>
2013	1	1
2013	1	1
2013	1	1
2013	1	1
2013	1	1
2013	1	1
2013	1	1
2013	1	1
2013	1	1
2013	1	1

1–10 of 13,972 rows

Previous 1 2 3 4 5 6 ... 100 Next

2. select

If the columns we want form a contiguous block, then we can use simpler syntax. To select rows from `year` to `arr_delay` (inclusive):

```
Stanford %>% select(year:arr_delay)
```

year <int>	month <int>	day <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>
2013	1	1	558	600	-2	923	937	-14
2013	1	1	611	600	11	945	931	14
2013	1	1	655	700	-5	1037	1045	-8
2013	1	1	729	730	-1	1049	1115	-26
2013	1	1	734	737	-3	1047	1113	-26
2013	1	1	745	745	0	1135	1125	10
2013	1	1	746	746	0	1119	1129	-10
2013	1	1	803	800	3	1132	1144	-12
2013	1	1	826	817	9	1145	1158	-13
2013	1	1	1029	1030	-1	1427	1355	32

1–10 of 13,972 rows

Previous 1 2 3 4 5 6 ... 100 Next

2. select

In our example, the `year` column is superfluous, since all the values are all 2013. The code below drops the year column, keeping the rest:

```
Stanford %>% select(-year)
```

month <int>	day <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	carrier <chr>	flight <int>
1	1	558	600	-2	923	937	-14	UA	1124
1	1	611	600	11	945	931	14	UA	303
1	1	655	700	-5	1037	1045	-8	DL	1865
1	1	729	730	-1	1049	1115	-26	VX	11
1	1	734	737	-3	1047	1113	-26	B6	643
1	1	745	745	0	1135	1125	10	AA	59
1	1	746	746	0	1119	1129	-10	UA	1668
1	1	803	800	3	1132	1144	-12	UA	223
1	1	826	817	9	1145	1158	-13	UA	1480
1	1	1029	1030	-1	1427	1355	32	AA	179

1–10 of 13,972 rows | 1–10 of 18 columns

Previous 1 2 3 4 5 6 ... 100 Next

2. select

We can also use `select()` to rearrange the columns. If, for example, we want to have the first 3 columns be day, month, year instead of year, month, day:

```
Stanford %>% select(day, month, year, everything())
```

day <int>	month <int>	year <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	carrier <chr>
1	1	2013	558	600	-2	923	937	-14	UA
1	1	2013	611	600	11	945	931	14	UA
1	1	2013	655	700	-5	1037	1045	-8	DL
1	1	2013	729	730	-1	1049	1115	-26	VX
1	1	2013	734	737	-3	1047	1113	-26	B6
1	1	2013	745	745	0	1135	1125	10	AA
1	1	2013	746	746	0	1119	1129	-10	UA
1	1	2013	803	800	3	1132	1144	-12	UA
1	1	2013	826	817	9	1145	1158	-13	UA
1	1	2013	1029	1030	-1	1427	1355	32	AA

1-10 of 13,972 rows | 1-10 of 19 columns Previous 1 2 3 4 5 6 ... 100 Next

2. select

To rename column names, use the `` `rename()` `` function:

```
Stanford %>% rename(tail_num = tailnum)
```

We can of course combine multiple functions like:

```
Stanford %>% filter(month == 1) %>%
  select(day,tailnum) %>%
  rename(tail_num = tailnum)
```

day <int>	tail_num <chr>
1	N53441
1	N532UA
1	N705TW
1	N635VA
1	N625JB
1	N336AA
1	N24224
1	N510UA
1	N76522
1	N325AA

1-10 of 929 rows Previous 1 2 3 4 5 6 ... 93 Next

3. arrange

Often we get datasets which are not in order, or in an order which we are not interested in. The `arrange()` function allows us to reorder the rows according to an order we want.

The `Stanford` dataset looks like it is already ordered by actual departure time. Perhaps we are most interested in the flights which had the longest departure delay. We could sort the dataset as follows:

```
Stanford %>% arrange(dep_delay)
```

year <int>	month <int>	day <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	carrier <chr>
2013	12	11	710	730	-20	1039	1105	-26	VX
2013	11	16	712	730	-18	1025	1055	-30	VX
2013	9	11	712	730	-18	946	1045	-59	VX
2013	11	19	713	730	-17	1036	1055	-19	VX
2013	7	14	1151	1208	-17	1450	1515	-25	UA
2013	12	10	714	730	-16	1104	1110	-6	VX
2013	3	29	1050	1106	-16	1359	1431	-32	UA
2013	4	20	1420	1436	-16	1737	1755	-18	UA
2013	5	20	719	735	-16	951	1110	-79	VX
2013	1	23	545	600	-15	948	925	23	UA

1–10 of 13,972 rows | 1–10 of 19 columns

Previous 1 2 3 4 5 6 ... 100 Next

3. arrange

It looks like the flights with the shortest delay are at the top instead. To re-order by descending order, use `desc()`:

```
Stanford %>% arrange(desc(dep_delay))
```

year <int>	month <int>	day <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	carrier <chr>
2013	9	20	1139	1845	1014	1457	2210	1007	AA
2013	7	7	2123	1030	653	17	1345	632	VX
2013	7	7	2059	1030	629	106	1350	676	VX
2013	7	6	149	1600	589	456	1935	561	DL
2013	7	10	133	1800	453	455	2130	445	B6
2013	7	10	2342	1630	432	312	1959	433	VX
2013	7	7	2204	1525	399	107	1823	404	UA
2013	7	7	2306	1630	396	250	1959	411	VX
2013	6	23	1833	1200	393	NA	1507	NA	UA
2013	7	10	2232	1609	383	138	1928	370	UA

1–10 of 13,972 rows | 1–10 of 19 columns

Previous 1 2 3 4 5 6 ... 100 Next

(Wow, that’s a really long delay! Almost 17 hours.)

3. arrange

To extract just the flights with the top 5 departure delays, we can use the `head()` function:

```
Stanford %>%
    arrange(desc(dep_delay)) %>%
    slice_head(n = 5)
```

year <int>	month <int>	day <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	carrier <chr>
2013	9	20	1139	1845	1014	1457	2210	1007	AA
2013	7	7	2123	1030	653	17	1345	632	VX
2013	7	7	2059	1030	629	106	1350	676	VX
2013	7	6	149	1600	589	456	1935	561	DL
2013	7	10	133	1800	453	455	2130	445	B6

5 rows | 1–10 of 19 columns

3. arrange

The function `` `arrange()` `` also allows us to filter by more than one column, in that each additional column will be used to break ties in the values of the preceding ones. For example, the data set `` `flights` `` seems to be sorted by year, month, day, and actual departure time. If we want to sort by year, month, day and scheduled departure time in reverse instead:

```
Stanford %>% arrange(year, desc(month), desc(day), desc(sched_dep_time))
```

4. mutate

In the `flights` dataset, we have both the time the plane spent in the air (`air_time`) and distance traveled (`distance`). From these, we can figure out the average speed of the plane for the flight using `mutate()`.

The function `mutate()` adds new columns to the end of the dataset.

```
Stanford_small <- Stanford %>%
    select(month, carrier, origin, dest, air_time, distance) %>%
    mutate(speed = distance / air_time * 60)
Stanford_small
```

month <int>	carrier <chr>	origin <chr>	dest <chr>	air_time <dbl>	distance <dbl>	speed <dbl>
1	UA	EWR	SFO	361	2565	426.3158
1	UA	JFK	SFO	366	2586	423.9344
1	DL	JFK	SFO	362	2586	428.6188
1	VX	JFK	SFO	356	2586	435.8427
1	B6	JFK	SFO	350	2586	443.3143
1	AA	JFK	SFO	378	2586	410.4762
1	UA	EWR	SFO	373	2565	412.6005
1	UA	JFK	SFO	369	2586	420.4878
1	UA	EWR	SFO	357	2565	431.0924
1	AA	JFK	SFO	389	2586	398.8689

1-10 of 13,972 rows

Previous 1 2 3 4 5 6 ... 100 Next

4. mutate

The function `mutate()` can be used to create several new variables at once. For example, the following code is valid syntax:

```
Stanford_small %>% mutate(speed_miles_per_min = air_time / distance,
                          speed_miles_per_hour = speed_miles_per_min * 60)
```

month <int>	carrier <chr>	origin <chr>	dest <chr>	air_time <dbl>	distance <dbl>	speed <dbl>	speed_miles_per_min <dbl>	speed_miles_per_hour <dbl>
1	UA	EWR	SFO	361	2565	426.3158	0.1407407	8.444444
1	UA	JFK	SFO	366	2586	423.9344	0.1415313	8.491879
1	DL	JFK	SFO	362	2586	428.6188	0.1399845	8.399072
1	VX	JFK	SFO	356	2586	435.8427	0.1376643	8.259861
1	B6	JFK	SFO	350	2586	443.3143	0.1353442	8.120650
1	AA	JFK	SFO	378	2586	410.4762	0.1461717	8.770302
1	UA	EWR	SFO	373	2565	412.6005	0.1454191	8.725146
1	UA	JFK	SFO	369	2586	420.4878	0.1426914	8.561485
1	UA	EWR	SFO	357	2565	431.0924	0.1391813	8.350877
1	AA	JFK	SFO	389	2586	398.8689	0.1504254	9.025522

1-10 of 13,972 rows

Previous 1 2 3 4 5 6 ... 100 Next

4. mutate

If we only want to keep the newly created variables, use `` `transmute()` `` instead of `` `mutate()` ``.

```
Stanford_small %>% transmute(speed_miles_per_min = air_time / distance,
                  speed_miles_per_hour = speed_miles_per_min * 60)
```

speed_miles_per_min <dbl>	speed_miles_per_hour <dbl>
0.1407407	8.444444
0.1415313	8.491879
0.1399845	8.399072
0.1376643	8.259861
0.1353442	8.120650
0.1461717	8.770302
0.1454191	8.725146
0.1426914	8.561485
0.1391813	8.350877
0.1504254	9.025522

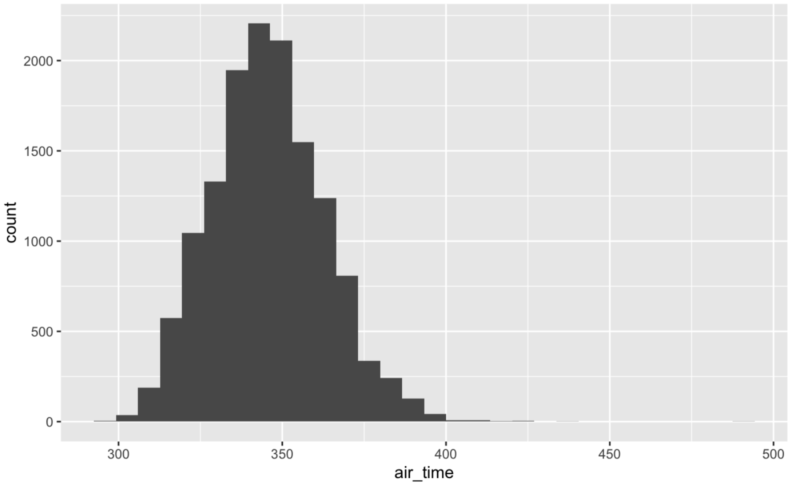
1–10 of 13,972 rows

Previous 1 2 3 4 5 6 ... 100 Next

A digression: plotting our data

Let's make use of our plotting skills from last session to see if there are any trends in air time. First, let's make a histogram of `air_time`:

```
library(ggplot2)
Stanford_small %>% ggplot() + # be careful, remember that ggplot layers need the + sign not piping!
    geom_histogram(aes(x = air_time))
```



A digression: plotting our data

Did you notice the warning message about rows being removed for "containing non-finite values"? If you view the `Stanford_small` dataset, you'll notice that there are some rows which have `NA` for `air_time`. Since we don't know what the air time is, we can't compute the speed and we can't plot it.

As a data analyst, `NA`s are something to watch out for as they could invalidate your analysis. Why are these data missing? Is it completely at random, or is there something going on?

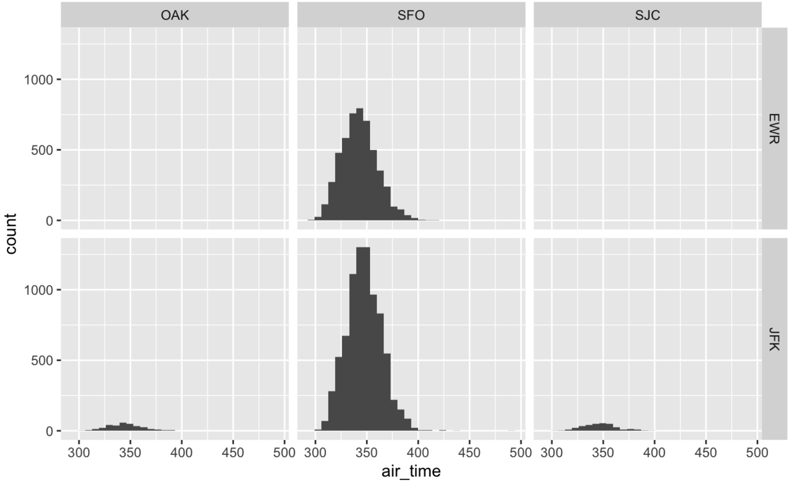
For this session, let's just remove the rows with `air_time` being `NA`:

```
Stanford_small <- Stanford_small %>%
    filter(!is.na(air_time))
```

A digression: plotting our data

It seems like the air time of planes might vary depending on the origin and destination, so let's facet on these 2 variables:

```
Stanford_small %>% ggplot() +
    geom_histogram(aes(x = air_time)) +
    facet_grid(origin ~ dest)
```



5. summarize

Instead of looking at plots, we can try to look at summary statistics instead. What was the mean/median air time for flights in our `Stanford_small` dataset? We can use the `summarize()` function to help us

```
Stanford_small %>% summarize(mean_airtime = mean(air_time))
```

```
## # A tibble: 1 × 1
##   mean_airtime
##          <dbl>
## 1           NA
```

```
Stanford_small %>% summarize(median_airtime = median(air_time))
```

```
## # A tibble: 1 × 1
##   median_airtime
##            <dbl>
## 1             NA
```

5. summarize

The `NA`s are causing us trouble! We need to specify the `na.rm = TRUE` option to remove `NA`s from consideration:

```
Stanford_small %>% summarize(mean_airtime = mean(air_time, na.rm = TRUE))
```

```
## # A tibble: 1 × 1
##   mean_airtime
##          <dbl>
## 1         346.
```

```
Stanford_small %>% summarize(median_airtime = median(air_time, na.rm = TRUE))
```

```
## # A tibble: 1 × 1
##   median_airtime
##            <dbl>
## 1            345
```

5. summarize and group_by

The function `summarize()` gives me a summary of the entire dataset. If we want summaries by group, then we have to use `summarize()` in conjunction with `group_by()`. The function `group_by()` changes the unit of analysis from the whole dataset to individual groups. The following code groups the dataset by carrier, then computes the summary statistic for each group:

```
Stanford_small %>%
    group_by(carrier) %>%
    summarize(mean_airtime = mean(air_time, na.rm = TRUE)) %>%
    arrange(desc(mean_airtime))
```

```
## # A tibble: 5 × 2
##   carrier mean_airtime
##   <chr>          <dbl>
## 1 AA              348.
## 2 VX              348.
## 3 DL              347.
## 4 B6              347.
## 5 UA              344.
```

5. summarize and group_by

We can also group by more than one variable. For example, if we want to count the number of flights for each carrier in each month, we could use the following code:

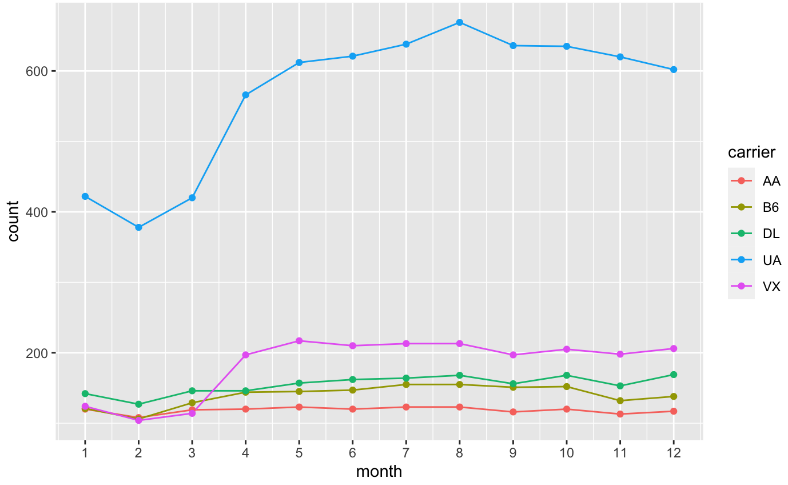
```
Stanford_small %>%
    group_by(month, carrier) %>%
    summarize(count = n())
```

```
## # A tibble: 60 × 3
## # Groups:   month [12]
##    month carrier count
##    <int> <chr>   <int>
##  1     1 AA        120
##  2     1 B6        121
##  3     1 DL        142
##  4     1 UA        422
##  5     1 VX        124
##  6     2 AA        108
##  7     2 B6        106
##  8     2 DL        127
##  9     2 UA        378
## 10     2 VX        104
## # … with 50 more rows
## # ℹ Use `print(n = ...)` to see more rows
```

5. summarize and group_by

We can even pipe the summarize dataset to `ggplot()` to plot the data.

```
Stanford_small %>%
    group_by(month, carrier) %>%
    summarize(count = n()) %>%
    ggplot(mapping = aes(x = month, y = count, col = carrier)) +
        geom_line() +
        geom_point() +
        scale_x_continuous(breaks = 1:12)
```



Wordle Example

Here's a fun example with the recently viral word game Wordle: Wordle Game

Wordle is basically a combination of the classic games Mastermind and Hangman. You try to guess a five letter word and after each guess you get information about which letters you have guessed correctly. If we want to win, we should probably guess words with very common letters first. How could we come up with the best first two words to cover the most common ten letters?

Let's get some data. Below we use the `readr` package to load a competitive Scrabble dictionary I found online (I did some testing to confirm that its weirder words were legal in Wordle). While we don't know the exact dictionary used by Wordle, this seems good enough for our purposes.

```
library(readr)
library(stringr)
dict<-read_delim("http://norvig.com/ngrams/TWL06.txt", delim = '\n', col_names = FALSE, show_col_types = FALSE)
```

Wordle Example

Now remember, we don't want all of the words, just words that are 5 letters long. How is our data stored?

```
dict
```

```
## # A tibble: 178,691 × 1
##      X1
##     <chr>
##   1 AA
##   2 AAH
##   3 AAHED
##   4 AAHING
##   5 AAHS
##   6 AAL
##   7 AALII
##   8 AALIIS
##   9 AALS
##  10 AARDVARK
## # … with 178,681 more rows
## # ℹ Use `print(n = ...)` to see more rows
```

Wordle Example

Luckily for us, there is a nice package for dealing with strings in the tidyverse called `` `stringr` `` and it has just the function we need `` `str_length` ``. How should we filter words that are 5 letters long?

Wordle Example

Luckily for us, there is a nice package for dealing with strings in the tidyverse called `stringr` and it has just the function we need `str_length`. How should we filter words that are 5 letters long?

```
dict %>% filter(str_length(X1) == 5) %>%
  rename(word = X1) -> wordle_tb
wordle_tb
```

```
## # A tibble: 8,938 × 1
##    word
##    <chr>
##  1 AAHED
##  2 AALII
##  3 AARGH
##  4 ABACA
##  5 ABACI
##  6 ABACK
##  7 ABAFT
##  8 ABAKA
##  9 ABAMP
## 10 ABASE
## # … with 8,928 more rows
## # ℹ Use `print(n = ...)` to see more rows
```

Wordle Example

Remember that we want to guess words with very common letters first.

What are the most common letters?

We can use

- the function `str_count(string, pattern)` which counts the number of times pattern is found within each element of string.
- the function `str_detect(string, pattern)` which detects the presence of the pattern in the string.

Wordle Example

Remember that we want to guess words with very common letters first.

What are the most common letters?

We can use

- the function `` `str_count(string, pattern)` `` which counts the number of times pattern is found within each element of string.
- the function `` `str_detect(string, pattern)` `` which detects the presence of the pattern in the string.

```
atleast <- rep(0, 26)
total <- rep(0, 26)
```

```
for (i in seq_along(LETTERS)){
  total[i] <- # FILL IN!
  atleast[i] <- # FILL IN!
}
```

Wordle Example

Remember that we want to guess words with very common letters first.

What are the most common letters?

We can use

- the function `` `str_count(string, pattern)` `` which counts the number of times pattern is found within each element of string.
- the function `` `str_detect(string, pattern)` `` which detects the presence of the pattern in the string.

```
atleast <- rep(0, 26)
total <- rep(0, 26)
```

```
for (i in seq_along(LETTERS)){
  total[i] <- wordle_tb$word %>% str_count(LETTERS[i]) %>% mean()
  atleast[i] <- wordle_tb$word %>% str_detect(LETTERS[i]) %>% mean()
}
```

Wordle Example

Let’s look at the results!

```
final <- tibble(Letter = LETTERS, total = total, atleast = atleast)
```

How can we print the top 10 most common letters?

Wordle Example

Let’s look at the results!

```
final %>% arrange(desc(total)) %>% slice_head(n=10) %>% select(Letter,total)
```

```
## # A tibble: 10 × 2
##    Letter total
##    <chr>  <dbl>
##  1 S      0.520
##  2 E      0.513
##  3 A      0.447
##  4 O      0.334
##  5 R      0.326
##  6 I      0.295
##  7 L      0.273
##  8 T      0.260
##  9 N      0.227
## 10 D      0.194
```

Wordle Example

Let's look at the results!

```
final %>% arrange(desc(atleast)) %>% slice_head(n=10) %>% select(Letter,atleast)
```

```
## # A tibble: 10 × 2
##    Letter atleast
##    <chr>    <dbl>
##  1 S        0.461
##  2 E        0.447
##  3 A        0.405
##  4 R        0.308
##  5 O        0.294
##  6 I        0.281
##  7 L        0.250
##  8 T        0.239
##  9 N        0.215
## 10 U        0.185
```

What words should we guess?

Wordle Example

What could we do to improve this analysis?