FE8828 Programming Web Applications in Finance - Session 3 -

Data Manipulation and EDA/2

Dr. Yang Ye yy@runchee.com

Oct 1, 2020

1. Lecture 7: Data Manipulation and EDA/2

2. `group_by` / `summarize`: Examples

3. `summarize/summarise`

4. Assignment

5. Assignment

Section 1

Lecture 7: Data Manipulation and EDA/2

Joins

left_/right_/anti_/full_join

Sample data:

- data_day1

Date	Position_id	Buy/Sell	Quantity	Risk Factor	Traded Price
2019-11-07	00010001	B	100	DCE_IO_1901	505.3
2019-11-07	00010002	B	100	DCE_IO_1901	506.8

- data_day2

Date	Position_id	Buy/Sell	Quantity	Risk Factor	Traded Price
2019-11-07	00010001	B	100	DCE_IO_1901	505.3
2019-11-07	00010002	B	100	DCE_IO_1901	506.8
2019-11-08	00010003	S	-100	DCE_IO_1901	507.9

Positions are additive (to close a position, we won't change the original position but to do a new reverse trade). Suppose we have two days of position data.

left_/right_/anti_/full_join

In order to find the new positions. We will use:

```
# order matters, data_day2 needs to be placed first.
# anti_join is like "data_day2 - data_day1"
anti_join(data_day2, data_day1, by = "position_id")
```

In order to find old positions, we will use:

```
# inner_join ignores order
# find the common positions
inner_join(data_day2, data_day1, by = "position_id")
left_join(data_day1, data_day2, by = "position_id") # produce the sa
right_join(data_day1, data_day2, by = "position_id") # produce the s
left_join(data_day2, data_day1, by = "position_id") # produce all it
```

left_join / right_join

They can be used to do mapping table (aka. vlookup)

Table Product:

type_code	type_name
1	orange
2	banana

Table Transaction:

type_code	quantity	customer_id
1	1	A
2	3	B
3	4	C
2	2	D
1	6	B

Table Customer:

customer_id	customer_phone
A	+123
B	+456
C	+789

Use left_join to create a full report

```
left_join(Transaction, Product, by = "type_code") %>%
left_join(Customer, by = "customer_id")
```

```
| type_code | quantity | customer_id | type_name | customer_phone |
|-----------|----------|-------------|-----------|----------------|
| 1         | 1        | A           | orange    | +123           |
| 2         | 3        | B           | banana    | +456           |
| 3         | 4        | C           | NA        | +789           |
| 2         | 2        | D           | banana    | NA             |
| 1         | 6        | B           | orange    | +456           |
```

full_join and anti_join

- full_join(a, b): Find all combinations between table a and b.
- anti_join(a, b): Find those in a but not in b.

```
# From something simple
df <- full_join(data_frame(a = 1:2), data_frame(a = 2:4), by = "a")
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was
```

a
1
2
3
4

```
df <- anti_join(data_frame(a = 1:2), data_frame(a = 2:4), by = "a")
```

a

full_join and anti_join More

```
# All possible combination between job and education
x <- full_join(distinct(bank, job) %>% mutate(dummy = 1),
               distinct(bank, education) %>% mutate(dummy = 1),
               by = "dummy") %>%
     select(-dummy)
y <- distinct(bank, job, education)

nrow(x)
## [1] 48
nrow(y)
## [1] 48

df1 <- anti_join(x, y, by = c("job", "education"))
df2 <- anti_join(y, x, by = c("job", "education"))
```

- df1: Empty result

job	education

- df2: Empty result

Join is a set operation

- `full_join` is `*`
- `anti_join` is `-`
- `inner_joins` is `-`, `/`
- `left_join/right_join` is either just the same, or `*`, `/`.

group_by / summarize

group_by is the way leading to analyze the data at high-dimension. group_by is used together with summarize

```
group_by(df, ...) ... is the list of variables
summarize(df, new_field = some_func_can_process_bulk_data())
```

Functions can process bulk data:

- sum/mean/median/sd: basic statistics
- min(x), quantile(x, 0.25), max(x): min/max/quantile
- n()/n_distinct(): count and count distint
- ntile: a rough divide into a few groups
- first(x), last(x), nth(x, 2)
- ...

group by / summarize: Examples

```r
# Add paramter na.rm, if there is NA among the data.
df <- data.frame(a = c(1, 3, 4, NA))
```

a
1
3
4
NA

```r
summarise(df, total = sum(a))
```

total
NA

```r
summarise(df, total = sum(a, na.rm = TRUE))
```

total
8

group_by / summarize: Examples

```
# count number of people in each age group
group_by(bank, age) %>% summarise(n = n())
## `summarise()` ungrouping output (override with `.groups` argument
```

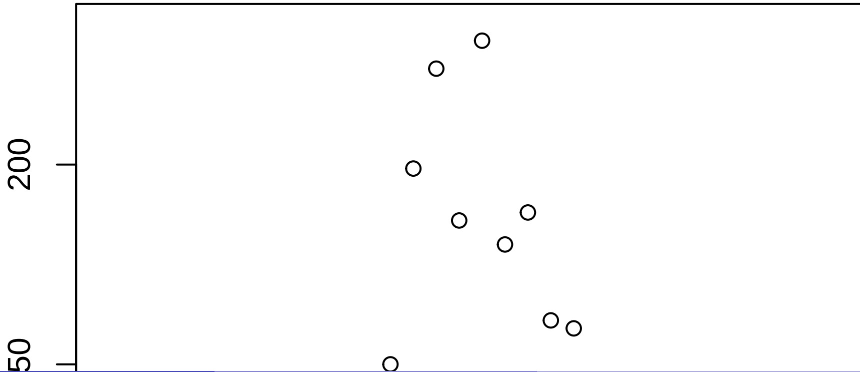
```
## `summarise()` ungrouping output (override with `.groups` argument
```

age	n
19	4
20	3
21	7
22	9
23	20
24	24

...

group by / summarize: Examples

```
group_by(bank, age) %>% summarise(n = n()) %>% plot
## `summarise()` ungrouping output (override with `.groups` argumen
```



Section 2

`group_by` / `summarize`: Examples

group_by / summarize: Examples

```
bank_age <- group_by(bank, age) %>%
        summarise(balance_mean = mean(balance),
                  count = n(),
                  default_count = sum(ifelse(default == "no", 0, 1)
## `summarise()` ungrouping output (override with `.groups` argument
```

age	balance_mean	count	default_count
19	393.5000	4	0
20	661.3333	3	0
21	1774.2857	7	0
22	1455.3333	9	0
23	2117.9500	20	1
24	634.6250	24	1
25	1240.0682	44	1
26	788.5584	77	3
27	851.7766	94	4
28	1025.0971	103	1

...

group_by / summarize: Examples

```
# If combined with ggplot, to be learnt in next session
bank_age %>%
  ggplot(aes(x = age, y = balance_mean)) +
  geom_point(aes(size = count), alpha = 1/4, color = "red") +
  geom_point(aes(size = default_count), alpha = 1/3, color = "blue")
  geom_smooth(se = FALSE)
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

Group filter

```
# Find the maximum and minimum balance on each age.
df <- bank %>%
  group_by(age) %>%
  filter(min_rank(balance) == 1 | min_rank(desc(balance)) == 1) %>%
  arrange(age, balance)
```

age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	pre
19	student	single	unknown	no	0	no	no	cellular	11	feb	123	3	-1	0
19	student	single	unknown	no	1169	no	no	cellular	6	feb	463	18	-1	0
20	student	single	secondary	no	291	no	no	telephone	11	may	172	5	371	5
20	student	single	secondary	no	1191	no	no	cellular	12	feb	274	1	-1	0
21	student	single	secondary	no	6	no	no	unknown	9	may	622	1	-1	0
21	student	single	secondary	no	6844	no	no	cellular	14	aug	126	3	127	7
22	student	single	unknown	no	47	no	no	cellular	3	jul	69	3	-1	0
22	admin.	single	secondary	no	4111	no	yes	cellular	19	aug	65	1	-1	0
23	technician	single	secondary	no	-306	yes	no	unknown	4	jun	217	2	-1	0
23	student	single	secondary	no	9216	no	no	cellular	5	jun	471	2	-1	0

Count for condition

TRUE => 1, FALSE => 0

```
# Generate a report for balance and job
d1 <- group_by(bank, job) %>%
  summarise(`balance > 500` = sum(balance > 500))
## `summarise()` ungrouping output (override with `.groups` argumen
d2 <- group_by(bank, job) %>%
  summarise(`balance <= 500` = sum(balance <= 500))
## `summarise()` ungrouping output (override with `.groups` argumen
# df collects all jobs, in case some jobs are missing from either d
# This is a typical example for collecting data.
df <- distinct(bank, job) %>% arrange(job)
df <- left_join(df, d1, by = "job")
df <- left_join(df, d2, by = "job")
df <- mutate(df, total = `balance > 500` + `balance <= 500`)
```

job	balance > 500	balance <= 500	total
admin.	226	252	478
blue-collar	423	523	946
entrepreneur	74	94	168
housemaid	48	70	118

group_by and summarise/summarize: Further explain

- group_by is a like folding a paper without tearing it later.
- summarise tears the paper to do individual pieces.
- Therefore, group_by can be used with other verbs, mutate, filter, which will work within the group.
- summarise can be used without group_by, then it will apply to entire data as one whole group.

group_by

```r
# mutate with group_by
df <- group_by(data.frame(a = 1:10), quantile = ntile(a, 2)) %>%
  mutate(b = a / sum(a))
```

a	quantile	b
1	1	0.0666667
2	1	0.1333333
3	1	0.2000000
4	1	0.2666667
5	1	0.3333333
6	2	0.1500000
7	2	0.1750000
8	2	0.2000000
9	2	0.2250000
10	2	0.2500000

group by / 2

```
# filter with group_by
df <- group_by(bank, age) %>% filter(balance == max(balance))
```

age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	pr
22	admin.	single	secondary	no	4111	no	yes	cellular	19	aug	65	1	-1	
78	housemaid	married	secondary	no	499	no	no	telephone	16	mar	80	4	-1	
23	student	single	secondary	no	9216	no	no	cellular	5	jun	471	2	-1	
46	management	married	secondary	no	12186	no	no	unknown	20	jun	29	3	-1	
64	retired	married	unknown	no	2923	no	no	cellular	12	mar	120	1	-1	
77	retired	married	tertiary	no	7802	no	no	telephone	4	may	421	1	92	
39	management	single	tertiary	no	12437	no	no	telephone	18	nov	40	1	-1	
28	student	single	secondary	no	11555	no	no	cellular	8	apr	125	2	-1	
81	retired	married	secondary	no	1	no	no	cellular	19	aug	65	5	-1	
33	housemaid	single	tertiary	no	23663	yes	no	cellular	16	apr	199	2	146	
40	self-employed	married	tertiary	no	13669	no	no	cellular	15	oct	138	1	136	
31	housemaid	single	primary	no	26965	no	no	cellular	21	apr	654	2	-1	
30	management	single	tertiary	no	19358	no	no	cellular	19	nov	258	2	-1	
67	blue-collar	married	secondary	no	16353	no	no	cellular	27	oct	223	2	-1	

Section 3

summarize/summarise

summarize/summarise

```
# summarise with group_by
df <- group_by(data.frame(a = 1:10), quantile = ntile(a, 2)) %>%
  summarise(b = sum(a))
## `summarise()` ungrouping output (override with `.groups` argumen
```

quantile	b
1	15
2	40

```
# summarise without a group_by. It will treat entire df as a whole.
df <- summarise(bank,
                with_housing = sum(housing == "yes") / n(),
                age_min = min(age),
                duration_mean = mean(duration))
```

with_housing	age_min	duration_mean
0.5660252	19	263.9613

group_by/ungroup

`ungroup()` removes group definition, restores the “ungrouped”" data frame back to entire data. Because `group_by` will leave a trace

```
# wrong
df_wrong <- group_by(bank, age) %>%
  filter(balance == max(balance)) %>%
  summarize(balance = mean(balance)) %>%
  head(n = 3)
## `summarise()` ungrouping output (override with `.groups` argumen

# correct
df_correct <- group_by(bank, age) %>%
  filter(balance == max(balance)) %>%
  ungroup %>%
  summarize(balance = mean(balance))
```

age	balance
19	1169
20	1191
21	6844

group by/ungroup

```
# We can't remove age
# R will prompt for "Adding missing grouping variables: `age`"
df1 <- group_by(bank, age) %>%
  filter(balance == max(balance)) %>%
  select(-age) %>% head(n = 3)
## Adding missing grouping variables: `age`

# We can remove age with ungroup
df2 <- group_by(bank, age) %>%
  filter(balance == max(balance)) %>%
  ungroup %>%
  select(-age) %>% head(n = 3)
```

age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	pre
22	admin.	single	secondary	no	4111	no	yes	cellular	19	aug	65	1	-1	0
78	housemaid	married	secondary	no	499	no	no	telephone	16	mar	80	4	-1	0
23	student	single	secondary	no	9216	no	no	cellular	5	jun	471	2	-1	0

job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previ

rowwise

Sometimes, we need to use `rowwise()` which is a special `group_by` which makes every one row a group. `rowwise()` use case, it applies to complex logic that can't be applied as a group.

```
df <- data.frame(throw_dices = 1:10)
df <- rowwise(df) %>% mutate( mean = mean(sample(1:6, throw_dices, r
```

throw_dices	mean
1	5.000000
2	4.500000
3	5.666667
4	4.500000
5	4.000000
6	2.833333
7	3.428571
8	3.500000
9	3.777778
10	4.100000

bind_rows

- bind_rows is the + operator for data frames.

```
# add empty data frame is the same.
df1 <- bind_rows(data.frame(a = 3:4), data.frame())
```

a
3
4

```
df2 <- bind_rows(data.frame(), data.frame(a = 3:4))
```

a
3
4

bind_rows: Use case

I usually use bind_rows to collect results. For example,

```
new_positions <- data.frame()
closed_positions <- data.frame()

for (i in length(dates)-1) {
  old_date <- dates[i]
  new_date <- dates[i+1]

  new_data <- filter(position, date == new_date)
  old_data <- filter(position, date == old_date)

  new_positions <- bind_rows(new_positions,
                             anti_join(new_data, old_data, by = "pos
}

# new_positions contains all new positions on their day 1
```

bind_rows: Use case

If row order matters, bind_row can be used to re-order/splice and recombine.

```
# Get head and tail
# Note: use { } to use the .
df <- arrange(bank, age) %>%
      { bind_rows(head(., n = 5), tail(., n = 5)) }
```

age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	pre
19	student	single	primary	no	103	no	no	cellular	10	jul	104	2	-1	0
19	student	single	unknown	no	0	no	no	cellular	11	feb	123	3	-1	0
19	student	single	secondary	no	302	no	no	cellular	16	jul	205	1	-1	0
19	student	single	unknown	no	1169	no	no	cellular	6	feb	463	18	-1	0
20	student	single	secondary	no	502	no	no	cellular	30	apr	261	1	-1	0
83	retired	divorced	primary	no	0	no	no	telephone	31	may	664	1	77	3
83	retired	divorced	primary	no	1097	no	no	telephone	5	mar	181	1	-1	0
84	retired	divorced	primary	no	639	no	no	telephone	18	may	353	3	-1	0
86	retired	married	secondary	no	1503	no	no	telephone	18	mar	165	3	101	1
87	retired	married	primary	no	230	no	no	cellular	30	oct	144	1	-1	0

bind_rows: Use case

```r
# summary
df1 <- summarise_if(bank, is.numeric, mean)
```

age	balance	day	duration	campaign	pdays	previous
41.1701	1422.658	15.91528	263.9613	2.79363	39.76664	0.5425791

```r
# add summary to the records
df2<- tail(bind_rows(bank, summarise_if(bank, is.numeric, mean)), n
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	pr
4522	41.1701	NA	NA	NA	NA	1422.658	NA	NA	NA	15.91528	NA	263.9613	2.79363	39.76664	0.

bind_rows: Use case

```r
# bind_rows can match column names and type.
# let's adjust the column order.
# As due-deligence, better to check the result.
# I remember earlier version of dplyr doesn't do match.
df <- tail(bind_rows(bank, summarise_if(bank, is.numeric, mean) %>%
  select(balance, day, everything())), n = 1)
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	pr
4522	41.170	NA	NA	NA	NA	1422.658	NA	NA	NA	15.9152	NA	263.961	2.7936	39.766	0.4

bind_cols

- bind_cols is to extend the data frame in width.

Use cases

- It’s a lazyman’s `left_join` or `select`
- It copies the columns
- I usually find it useful to generate data frame for reports.

```r
dt1 <- bind_cols(select(bank, job), select(bank, education))
dt1[1:3,]
```

job	education
unemployed	primary
services	secondary
management	tertiary

bind_cols

```
dt2 <- bind_cols(dt1, dt1)
## New names:
## * job -> job...1
## * education -> education...2
## * job -> job...3
## * education -> education...4
dt2[1:3,]
```

job...1	education...2	job...3	education...4
unemployed	primary	unemployed	primary
services	secondary	services	secondary
management	tertiary	management	tertiary

bind_cols: Use cases

```
d1 <- filter(bank, month == "sep") %>%
  summarize(duration = mean(duration)) %>%
  rename(`Duration Sep` = duration)
d2 <- filter(bank, month == "oct") %>%
  summarize(duration = mean(duration)) %>%
  rename(`Duration Oct` = duration)
d3 <- filter(bank, month == "nov") %>%
  summarize(duration = mean(duration)) %>%
  rename(`Duration Nov` = duration)

df <- bind_cols(d1, d2, d3)
```

Duration Sep	Duration Oct	Duration Nov
215.7308	272.8	272.0668

Exercise

1. How to know the row number of the wrong date

```
df <- data.frame(dt = c("2019-10-01", "2019-31-12", "2019-03-17", "2
```

dt
2019-10-01
2019-31-12
2019-03-17
2019-02-29
2019-09-30

Output:

```
## Wrong dates on rows: 2, 4
```

Exercise

2. How to get sub-total and total on mean of age and balance, group by job and education?

job	education	mean(Age)	median(Balance)
services	primary	...	...
services			
services	+	...	...
...			
+	+	...	...

Exercise

3. To evaluate a portfolio of options for its total value.

```
GBSOption(TypeFlag = "p", S = 3500, X = 3765,
          Time = 1/12, r = 0, b = 0, sigma = 0.3)@price
## [1] 300.0049
df <- data.frame(type = sample(c("c", "p"), 100, replace = TRUE),
                 strike = round(runif(100) * 100, 0),
                 underlying = round(runif(100) * 100, 0),
                 Time = 1,
                 r = 0.01,
                 b = 0,
                 sigma = 0.3)
```

tidyr: `gather/spread`

Wide format <=> Long format

- Wide format is more familiar to us. Column name is the data attribute.
- Long format is what we reformat the data that common attributes are gathered together as a single variable.
- Reference: Tidy data https://en.wikipedia.org/wiki/Tidy_data

Wide v.s. Long

Wide format

```
wfmt <- data_frame(date = seq(from = as.Date("2019-01-01"), by = "da
                   Copper_qty = round(runif(5) * 1000, 0),
                   Gold_qty = round(runif(5) * 1000, 0),
                   Silver_qty = round(runif(5) * 1000, 0))
```

date	Copper_qty	Gold_qty	Silver_qty
2019-01-01	157	733	893
2019-01-02	295	626	462
2019-01-03	929	139	650
2019-01-04	335	137	440
2019-01-05	809	465	235

Wide v.s. Long

Long format

```r
library(tidyr)
df <- gather(wfmt, key, value, -date)
```

date	key	value
2019-01-01	Copper_qty	157
2019-01-02	Copper_qty	295
2019-01-03	Copper_qty	929
2019-01-04	Copper_qty	335
2019-01-05	Copper_qty	809
2019-01-01	Gold_qty	733
2019-01-02	Gold_qty	626
2019-01-03	Gold_qty	139
2019-01-04	Gold_qty	137
2019-01-05	Gold_qty	465
2019-01-01	Silver_qty	893
2019-01-02	Silver_qty	462
2019-01-03	Silver_qty	650
2019-01-04	Silver_qty	440

`spread/gather` convert for Wide format <=> Long format

```
# Original help
gather(data, key, value, ...)
# My annotated version
gather(data,
       new_key_col_name,
       new_value_col_name,
       -columns_to_be_included_in_the_left)
```

… is where you want to make as independent columns. You need to specify all columns that should be `gathered` (or before `gather`, remove all columns that should *not* be `gathered`).

gather example with *Bank* dataset

```
wfmt <- group_by(bank, job) %>% summarize(yy = sum(ifelse(default ==
## `summarise()` ungrouping output (override with `.groups` argumen
df <- gather(wfmt, default, value, -job) %>% arrange(job, default)
```

job	yy	nn
admin.	6	472
blue-collar	14	932
entrepreneur	7	161
housemaid	2	110
management	14	955
retired	3	227
self-employed	4	179

…

job	default	value
admin.	nn	472
admin.	yy	6
blue-collar	nn	932

spread

```
# Original help
spread(data, key, value)
# My annotated version
spread(data, colname_to_be_header, value_to_be_filled_under_header)
```

spread example with *Bank* dataset

```r
lfmt <- group_by(bank, job, default) %>% summarize(nn = n())
## `summarise()` regrouping output by 'job' (override with `.groups
df <- spread(lfmt, default, nn)
# How to take care of converting NA to zero?
```

job	default	nn
admin.	no	472
admin.	yes	6
blue-collar	no	932
blue-collar	yes	14
entrepreneur	no	161
entrepreneur	yes	7
housemaid	no	110

…

job	no	yes
admin.	472	6
blue-collar	932	14

Combine different columns' Quantity

date	Copper_qty	Gold_qty	Silver_qty
2019-01-01	243	99	654
2019-01-02	936	512	326
2019-01-03	597	91	301
2019-01-04	990	251	319
2019-01-05	606	375	134

...

```r
df <- wfmt %>%
      gather(key, value, -date) %>%
      group_by(date) %>%
      summarize(value1 = sum(value)) %>%
      rename(value = value1) %>%
      mutate(key = "Total") %>%
      spread(key = key, value = value) %>%
      inner_join(wfmt, ., by = "date")
## `summarise()` ungrouping output (override with `.groups` argumen
```

separate/unite

```
separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE,
  convert = FALSE, extra = "warn", fill = "warn", ...)
```

```
#> # A tibble: 6 × 3
#>       country  year              rate
#> *       <chr> <int>             <chr>
#> 1 Afghanistan  1999      745/19987071
#> 2 Afghanistan  2000     2666/20595360
#> 3      Brazil  1999   37737/172006362
#> 4      Brazil  2000   80488/174504898
#> 5       China  1999 212258/1272915272
#> 6       China  2000 213766/1280428583
```

```
separate(df, rate, into = c("cases", "population"))
separate(df, rate, into = c("cases", "population"), convert = TRUE)
```

```
unite(df, century, year) # default sep is "_"
unite(df, century, year, sep = "") # seamless unite
```

Rules of Thumb for use list of data frame

- Use list to store app data, i.e. configuration. `conf <- list(use_calendar_days = TRUE, do_fx_conversion = FALSE, year_convention = 252)`
- User data frame to store repeating data of similar structure.
- Every data frame is better to have a id column, like **item_id**. It can be number or character. Make it unique. If **item_id** is a number, when insert new record to the data frame, we need to increment it somewhere. So, use a variable to keep it somewhere, or use `max(item_id) + 1` (It will do calculation for all ids. Performance still good with small data set)
- Delete is not good for enterprise. We need to leave an audit trail. And we can prevent from wrong operation. Add a column name with a common name, e.g. SYS_DEL. Its default value is FALSE, when you want to delete it, set it to TRUE. When extracting data, use `filter(df1, !SYS_DEL)`. The advanced version involves the user and datetime, i.e. SYS_DEL_USER, SYS_DEL_DATETIME.

```
| position_id | call_put | amount | strike | SYS_DEL |
| X123        |          |        |        |
```

CRUD in dplyr

Create:

- add new rows. bind_rows()

Read:

- You have known enough: filter/select/joins/... to get what you need.

Update:

- Use either data frame way or mutate.

```
# get all row numbers for students
# . refers to the output of the pipe %>%. .$nnn => df$nnn
row_nums <- mutate(bank, nnn = 1:n()) %>%
            filter(job == "student" & age < 22) %>%
            select(nnn) %>%
            .$nnn

bank1 <- bank
bank1[row_nums, "taxable"] <- "no"
bank1[setdiff(1:nrow(bank), row_nums), "taxable"] <- "yes"
```

Section 4

Assignment

Assignment

1. Exploratory Data Work on the bank dataset. Find 10 findings from data. Use R Markdown.

~~~~
---
title: "FE8828 Assignment for Exploratory Data Analysis"
author: "Yang Ye <sub> <Email:yy@runchee.com> </sub>"
date: "Sep 2019"
output: html_document
---

```{r setup, include=FALSE}
library(tidyverse)
library(lubridate)
library(bizdays)
# Use echo = TRUE for assignment is an exception, so code is visible
knitr::opts_chunk$set(echo = TRUE, fig.align="center", collapse = TF
bank <- read.csv("https://goo.gl/PBQnBt", sep = ";")
```

# Finding #1
This data contains `r nrow(data)` rows
~~~~

Section 5

Assignment

Assignment

2. Book option trades

1.1 Copy the options data from https://www.nasdaq.com/symbol/goog/option-chain?dateindex=1

```
Gather data for "Dec 20, 2019" and store into following data frame f

    | Expiry Date | Strike | Open Interest | Underlying | Call/Put |
```

1.2 Count the total valuation of 1) call alone, 2) put alone, 3) call and put. `Open Interest * (Bid + Ask) / 2`

1.3 Find those in the money and get their total `Open Interest`.

1.4. Plot the volatility curve, strike v.s. vol. For strike < current price, use puts' price; for strike > current price, use calls' price.

```
# GBSVolatility(price, TypeFlag, S, X, Time, r, b, tol, maxiter)
# Use Price to back-out implied volatility. Assume r = 0.03
# Example:

GBSVolatility(867.30, "c", 1135.67, 240,
```