FE8828 Programming Web Applications in Finance

Week 3

Data, visualization, and web: part 2

Dr. Yang Ye <Email:yy@runchee.com>

Nov 16, 2017

Lecture 7: Data Part 2

Joins

left_/right_/anti_/full_join

```
| Position_id | Buy/Sell | Quantity | Risk Factor |
| ----------- |----------|----------|-------------|
```

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.

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

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

```
# order matters, new_data needs to be placed first.
anti_join(new_data, old_data, by = "position_id")
```

In order to find old positions, we will use:

```
# inner_join ignores order
inner_join(new_data, old_data, by = "position_id")
left_join(old_data, new_data, by = "position_id") # produce the same result
right_join(new_data, old_data, by = "position_id") # produce the same result
```

left_join / right_join

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

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.

```
# 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

anti_join(x, y, by = c("job", "education"))
```

0 rows

```
anti_join(y, x, by = c("job", "education"))
```

0 rows

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

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 our way leading to analyze the data.

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

Function can process bulk data: Note: I often forgot there are existing functions that resort to longer versions.

- 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), nth(x, 2), last(x):

group_by / summarize: Examples

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

total
<dbl>
NA

1 row

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

total
<dbl>
8

1 row

```
summarise(df, total = mean(a))
```

total
<dbl>

total
<dbl>
NA

1 row

```
summarise(df, total = mean(a, na.rm = T))
```

total
<dbl>
2.666667

1 row

group_by / summarize: Examples

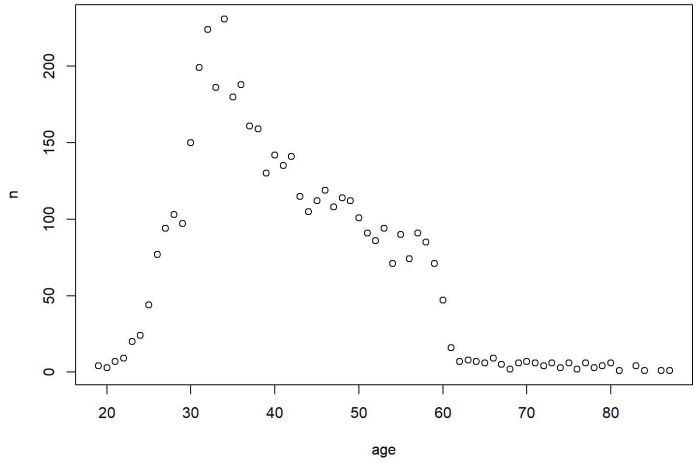
```
# count number of people in each age group
group_by(bank, age) %>% summarise(n = n())
```

age	n
<int>	<int>
19	4
20	3
21	7
22	9
23	20
24	24
25	44
26	77
27	94
28	103

1-10 of 67 rows Previous 1 2 3 4 5 6 7 Next

group_by / summarize: Examples

```
group_by(bank, age) %>% summarise(n = n()) %>% plot
```



group_by / summarize: Examples

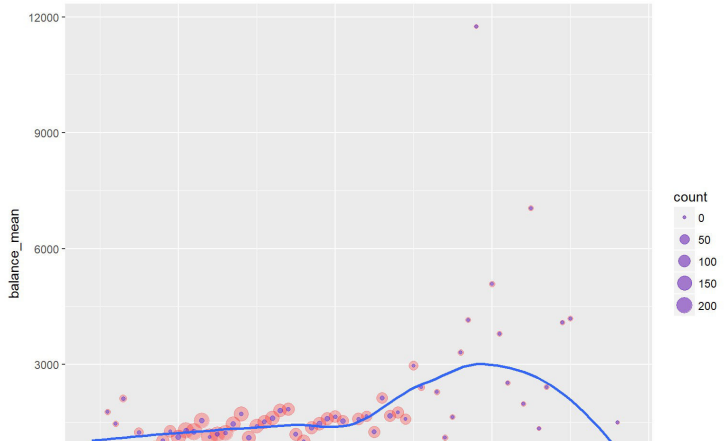
```
bank1 <- group_by(bank, age) %>%
  summarise(balance_mean = mean(balance),
            count = n(),
            default_count = sum(ifelse(default == "no", 0, 1)))

head(bank1)
```

age <int>	balance_mean <dbl>	count <int>	default_count <dbl>
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

6 rows

group_by / summarize: Examples

```
# If combined with ggplot, to be learnt in next session
ggplot(bank1, 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 = F)
## `geom_smooth()` using method = 'loess'
```



Group filter

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

age <int>	job <fctr>	marital <fctr>	education <fctr>	default <fctr>	balance <int>	housing <fctr>	lo <f
19	student	single	unknown	no	0	no	no
19	student	single	unknown	no	1169	no	no
20	student	single	secondary	no	291	no	no
20	student	single	secondary	no	1191	no	no
21	student	single	secondary	no	6	no	no
21	student	single	secondary	no	6844	no	no
22	student	single	unknown	no	47	no	no
22	admin.	single	secondary	no	4111	no	ye
23	technician	single	secondary	no	-306	yes	no
23	student	single	secondary	no	9216	no	no

1-10 of 135 rows | 1-10 of 17 columns Previous 1 2 3 4 5 6 .. 14 Next

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))
d2 <- group_by(bank, job) %>%
  summarise(`balance <= 500` = sum(balance <= 500))
# d collects all jobs, in case some jobs is missing from d1 or d2
d <- distinct(bank, job) %>% arrange(job)
d <- left_join(d, d1, by = "job")
d <- left_join(d, d2, by = "job")
d <- mutate(d, total = `balance > 500` + `balance <= 500`)
d
```

job	balance > 500	balance <= 500	total
<fctr>	<int>	<int>	<int>
admin.	226	252	478
blue-collar	423	523	946
entrepreneur	74	94	168
housemaid	42	70	112
management	521	448	969
retired	127	103	230

job	balance > 500	balance <= 500	total
<fctr>	<int>	<int>	<int>
self-employed	89	94	183
services	154	263	417
student	41	43	84
technician	353	415	768

1-10 of 12 rows Previous 1 2 Next

group_by and summarise/summarize: Further expository

- `group_by` is a like folding a paper without tearing it later.
- `summarise` will tear the paper.
- 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

```
# mutate with group_by
group_by(data.frame(a = 1:10), quartile = ntile(a, 2)) %>%
  mutate(b = a / sum(a))
```

a	quartile	b
<int>	<int>	<dbl>
1	1	0.06666667
2	1	0.13333333
3	1	0.20000000
4	1	0.26666667
5	1	0.33333333
6	2	0.15000000
7	2	0.17500000
8	2	0.20000000
9	2	0.22500000
10	2	0.25000000

1-10 of 10 rows

group_by / 2

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

age	job	marital	education	default	balance	housing	lo
<int>	<fctr>	<fctr>	<fctr>	<fctr>	<int>	<fctr>	<f
22	admin.	single	secondary	no	4111	no	ye
78	housemaid	married	secondary	no	499	no	nc
23	student	single	secondary	no	9216	no	nc
46	management	married	secondary	no	12186	no	nc
64	retired	married	unknown	no	2923	no	nc
77	retired	married	tertiary	no	7802	no	nc
39	management	single	tertiary	no	12437	no	nc
28	student	single	secondary	no	11555	no	nc
81	retired	married	secondary	no	1	no	nc
33	housemaid	single	tertiary	no	23663	yes	nc

1-10 of 68 rows | 1-10 of 17 columns Previous 1 2 3 4 5 6 7 Next

summarize/summarise

```
# summarise with group_by
group_by(data.frame(a = 1:10), quartile = ntile(a, 2)) %>%
  summarise(b = sum(a))
```

quartile	b
<int>	<int>
1	15
2	40

2 rows

```
# summarise without a group_by
summarise(bank, with_housing = sum(housing == "yes") / n(),
          age_min = min(age),
          duration_mean = mean(duration))
```

with_housing	age_min	duration_mean
<dbl>	<dbl>	<dbl>
0.5660252	19	263.9613

1 row

group_by/rowwise/ungroup

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

```
# wrong
group_by(bank, age) %>%
  filter(balance == max(balance)) %>%
  summarize(balance = mean(balance)) %>%
  head(n = 3)
```

age	balance
<int>	<dbl>
19	1169
20	1191
21	6844

3 rows

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


balance
<dbl>
13541.21

1 row

group_by/rowwise/ungroup

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

age	job	marital	education	default	balance	housing	loan	cc
<int>	<fctr>	<fctr>	<fctr>	<fctr>	<int>	<fctr>	<fctr>	<f
22	admin.	single	secondary	no	4111	no	yes	ce
78	housemaid	married	secondary	no	499	no	no	tel
23	student	single	secondary	no	9216	no	no	ce

3 rows | 1-10 of 17 columns

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

job	marital	education	default	balance	housing	loan	contac
<fctr>	<fctr>	<fctr>	<fctr>	<int>	<fctr>	<fctr>	<fctr>

job	marital	education	default	balance	housing	loan	contac
<fctr>	<fctr>	<fctr>	<fctr>	<int>	<fctr>	<fctr>	<fctr>
admin.	single	secondary	no	4111	no	yes	cellular
housemaid	married	secondary	no	499	no	no	telepho
student	single	secondary	no	9216	no	no	cellular

3 rows | 1-10 of 16 columns

rowwise

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

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

throw_dices	mean
<int>	<dbl>
1	5.000000
2	1.500000
3	2.000000
4	4.000000
5	3.000000
6	3.833333
7	4.142857
8	3.000000
9	3.888889
10	4.200000

bind_rows

- bind_rows is the + operator for data frames.

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

a
<int>
3
4

2 rows

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

a
<int>
3
4

2 rows

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 = "position_id"))
}

# new_positions contains all new positions on 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 .
arrange(bank, age) %>%
{ bind_rows(head(., n = 5), tail(., n = 5)) }
```

age	job	marital	education	default	balance	housing	loan	cor
<int>	<fctr>	<fctr>	<fctr>	<fctr>	<int>	<fctr>	<fctr>	<fct
19	student	single	primary	no	103	no	no	cellu
19	student	single	unknown	no	0	no	no	cellu
19	student	single	secondary	no	302	no	no	cellu
19	student	single	unknown	no	1169	no	no	cellu
20	student	single	secondary	no	502	no	no	cellu
83	retired	divorced	primary	no	0	no	no	tele
83	retired	divorced	primary	no	1097	no	no	tele
84	retired	divorced	primary	no	639	no	no	tele
86	retired	married	secondary	no	1503	no	no	tele
87	retired	married	primary	no	230	no	no	cellu

bind_rows: Use case

```
# summary
summarise_if(bank, is.numeric, mean)
```

age	balance	day	duration	campa...	pdays	previous
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
41.1701	1422.658	15.91528	263.9613	2.79363	39.76664	0.5425791

1 row

```
# add summary to the records
tail(bind_rows(bank, summarise_if(bank, is.numeric, mean)), n = 1)
```

	age	job	marital	education	default	balance	housing	lo
	<dbl>	<fctr>	<fctr>	<fctr>	<fctr>	<dbl>	<fctr>	<f
4522	41.1701	NA	NA	NA	NA	1422.658	NA	NA

1 row | 1-10 of 18 columns

bind_rows: Use case

```
# 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.
tail(bind_rows(bank, summarise_if(bank, is.numeric, mean)) %>%
  select(balance, day, everything()), n = 1)
```

	age	job	marital	education	default	balance	housing	lo
	<dbl>	<fctr>	<fctr>	<fctr>	<fctr>	<dbl>	<fctr>	<f
4522	41.1701	NA	NA	NA	NA	1422.658	NA	NA

1 row | 1-10 of 18 columns

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.

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

	job <fctr>	education <fctr>
1	unemployed	primary
2	services	secondary
3	management	tertiary

3 rows

bind_cols

```
dt2 <- bind_cols(dt1, dt1)
dt2[1:3,]
```

	job <fctr>	education <fctr>	job1 <fctr>	education1 <fctr>
1	unemployed	primary	unemployed	primary
2	services	secondary	services	secondary
3	management	tertiary	management	tertiary

3 rows

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)

bind_cols(d1, d2, d3)
```

Duration Sep <dbl>	Duration Oct <dbl>	Duration Nov <dbl>
215.7308	272.8	272.0668

1 row

Exercise

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

```
df <- data.frame(x = c("2017-10-01", "2017-31-12", "2017-03-17", "2017
df
```

x <fctr>
2017-10-01
2017-31-12
2017-03-17
2017-02-29
2017-09-30

5 rows

Output:

```
## Wrong dates on: 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 = T),
                 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

date <date>	Copper_qty <dbl>	Gold_qty <dbl>	Silver_qty <dbl>
2017-01-01	127	261	500
2017-01-02	490	538	567
2017-01-03	393	306	802
2017-01-04	234	356	684
2017-01-05	177	664	869

5 rows

Long format

date <date>	key <chr>	value <dbl>
2017-01-01	Copper_qty	127
2017-01-02	Copper_qty	490
2017-01-03	Copper_qty	393

date	key	value
<date>	<chr>	<dbl>
2017-01-04	Copper_qty	234
2017-01-05	Copper_qty	177
2017-01-01	Gold_qty	261
2017-01-02	Gold_qty	538
2017-01-03	Gold_qty	306
2017-01-04	Gold_qty	356
2017-01-05	Gold_qty	664

1-10 of 15 rows Previous 1 2 Next

spread/gather conversion for Wide format <=> Long format

```
gather(data, key, value, ...)
```

… is where you want to make as independent columns. You need to specify all columns that should be gathers (or remove all columns that should _not_be gathered):

```
mini_iris <- iris[1:3,]
gather(mini_iris,
       key = flower_att,
       value = measurement,
       -Species)
```

Species	flower_att	measurement
<fctr>	<chr>	<dbl>
setosa	Sepal.Length	5.1
setosa	Sepal.Length	4.9
setosa	Sepal.Length	4.7
setosa	Sepal.Width	3.5
setosa	Sepal.Width	3.0

Species	flower_att	measurement
<fctr>	<chr>	<dbl>
setosa	Sepal.Width	3.2
setosa	Petal.Length	1.4
setosa	Petal.Length	1.4
setosa	Petal.Length	1.3
setosa	Petal.Width	0.2

1-10 of 12 rows Previous 1 2 Next

spread/gather what columns to remove/add

```
lfmt <- gather(mini_iris,
               key = flower_att,
               value = measurement,
               Sepal.Length, Sepal.Width, Petal.Length, Petal.Width)
lfmt
```

Species	flower_att	measurement
<fctr>	<chr>	<dbl>
setosa	Sepal.Length	5.1
setosa	Sepal.Length	4.9
setosa	Sepal.Length	4.7
setosa	Sepal.Width	3.5
setosa	Sepal.Width	3.0
setosa	Sepal.Width	3.2
setosa	Petal.Length	1.4
setosa	Petal.Length	1.4
setosa	Petal.Length	1.3
setosa	Petal.Width	0.2

Spread

```
spread(lfmt, key, value)
```

Example: get row sum.

```
library(tidyr)
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")
```

date	Copper_qty	Gold_qty	Silver_qty	Total
<date>	<dbl>	<dbl>	<dbl>	<dbl>
2017-01-01	127	261	500	888
2017-01-02	490	538	567	1595
2017-01-03	393	306	802	1501
2017-01-04	234	356	684	1274
2017-01-05	177	664	869	1710

5 rows


```
# although this works...
# Hard coding of column names "Copper_qty Gold_qty Silver_qty".
wfmt %>% mutate(total = Copper_qty + Gold_qty + Silver_qty)
```

date	Copper_qty	Gold_qty	Silver_qty	total
<date>	<dbl>	<dbl>	<dbl>	<dbl>
2017-01-01	127	261	500	888
2017-01-02	490	538	567	1595
2017-01-03	393	306	802	1501
2017-01-04	234	356	684	1274
2017-01-05	177	664	869	1710

5 rows

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 design data storage

- Use list to store app data, i.e. configuration.
- User data frame to store repeating data of similar structure
- Every data frame is better to have a column with the function of **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, store it somewhere.
- Delete is not good for customer data. add a column name with a common name, e.g. SYS_DEL.

```
| 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
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"

# use dplyr
bank1 <- mutate(bank, taxable = ifelse(job == "student" & age < 22, "no", "yes"))
```



```
distinct(bank1, taxable)
```

Delete:

- Use filter to exclude the row(s).
- (Advanced version) Create a column `DELETED` of logic type, if we need to delete the row, set it T. We need to change all read commands to have `!DELETED`

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: "Nov 15, 2017"
output: html_document
---

```{r setup, include=FALSE}
library(tidyverse)
library(lubridate)
library(bizdays)
knitr::opts_chunk$set(echo = FALSE, fig.align="center", collapse = TRUE, cache = T)
bank <- read.csv("https://goo.gl/PBQnBt", sep = ";")
```

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

# Finding #2
```{r, echo = F}
# Find the big age group
bank %>%
  group_by(age_group = (age %/% 10) * 10) %>%
  summarise(count = n()) %>%
  arrange(age_group) -> res
````



````
res

plot(res$age_group, res$count)
```

# Discover insights of data
- Employment
- Social attributes.
- Count for sub-total / total, plot graph
````

Assignment

2. Book option trades

1.1 Store the options from https://finance.google.com/finance/option_chain?q=NASDAQ%3AAMZN&ei=iIoAWvDmF8GqugSsj5mICw

```
    | Date | Strike | Quantity | Underlying | Long/Short | Call/Put
```

1.2 Count the total valuation of call alone, put alone, call and put.

1.3 Find those in the money.

1.4. Plot the volatility curve, strike v.s. vol

```
# GBSVolatility(price, TypeFlag, S, X, Time, r, b, tol, maxiter)
# Use Price to back-out implied volatility
GBSVolatility(867.30, "c", 1135.67, 240,
              as.numeric((as.Date("2018-01-19") - as.Date("2017-11-15"
## [1] 8.101273e-18
GBSVolatility(256.50, "c", 1135.67, 880.00,
              as.numeric((as.Date("2018-01-19") - as.Date("2017-11-15"
## [1] 0.3107899
GBSVolatility(53.62, "c", 1135.67, 1120.00,
```



```
              as.numeric((as.Date("2018-01-19") - as.Date("2017-11-15"
## [1] 0.2394823
```


Assignment

3. Bank (Group - choose a coordinator to send me the report)
   This is the 1st installment of the assignment. There will be more installment. Due on 6th week. Please pace your group.

```
Data frame 1: Account
| AcountNo | Name |

Data frame 2: Transaction
| TransactionNo | Date | AccountNo | TransactionType | Amount | Currency |

Data frame 3: Currency to SGD
| Currency | Conversion | Date |
```

TransactionType can be: Withdraw/Deposit/Spend Write follow functions and use them to initialize the data.

1. Create 10 accounts with initial random deposit and credit in SGD.
2. Create 3 currencies: CNY, USD, SGD. Download their conversion rate between 2017-07-01 and 2017-09-30.
3. Generate random transaction data for 10 accounts during 2017-07-01

and 2017-09-30. Make it more realistic, deposite is 1-2 times per month, a random number of 3000-5000. Spend/Withdraw can be any times and any amount. Deposit is positive, Withdraw/Spend is negative. You can't withdraw more than the deposit, can't spend more than credit + deposit.

4. Generate report for transaction as month-end statement in SGD.

```
{ Client Name }
{ Month }

# Transaction History

Date | TransactionType | Amount | Currency | Deposit Balance | Credit Balance |
     |                 |        |          |                 |                |
     |                 |        |          |                 |                |
     |                 |        |          |                 |                |
-------------------------------------------------------------------------------
                        Month-End Balance | Deposit         | Credit         |

# Summary

TransactionType | Amount |
...
```

Submission:

1. A report describing interesting learning points on design and coding (1-2 pages, just be concise)
2. Code with decent amount of comments
3. Example running result.