DataVizA Tutorial: Data Munging: Solutions

Department of Econometrics and Business Statistics, Monash University

Tutorial 5

Swiss Exports: Full Data

The file *SwissExportsFull.csv* contains the full export data for Switzerland. Each row represents a different date. The first column is the date variable, the second column is the year only and each remaining column measures exports to a different country.

1. Read the data into R.

```
library(tidyverse)
SwissWide<-read_csv('SwissExportsFull.csv')
```

```
## Warning: Missing column names filled in: 'X154' [154]
```

```
#This works but with a quirky warning.  One country code is NA for
#Namibia, however R treats NA as a missing value.  It can be fixed
#with

SwissWide<-read_csv('SwissExportsFull.csv')%>%
  rename(`NA`=X154)
```

```
## Warning: Missing column names filled in: 'X154' [154]
```

```
SwissWide
```

```
## # A tibble: 372 x 247
##    Date        Year     AD     AE     AF     AG     AI     AL    AM     AO     AQ     AR
##    <date>     <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl> <dbl>  <dbl>  <dbl>  <dbl>
##  1 1988-01-01  1988 1.68e5 1.35e7 4.17e5 2.08e3      0 9.52e4     0 6.61e5  5911. 1.14e7
##  2 1988-02-01  1988 4.70e5 2.61e7 2.16e5 4.39e4      0 2.17e5     0 1.36e6      0 1.02e7
##  3 1988-03-01  1988 3.70e5 2.08e7 3.55e4 1.14e4  2497. 1.72e5     0 6.63e5 26106. 1.63e7
##  4 1988-04-01  1988 1.98e5 1.75e7 6.14e5 9.14e3  2100. 5.05e5     0 9.28e5 47189. 1.50e7
##  5 1988-05-01  1988 4.19e5 1.59e7 2.79e5 8.12e5  7511. 1.03e6     0 6.41e5      0 1.78e7
##  6 1988-06-01  1988 6.89e5 2.01e7 3.55e5 1.03e4  1715. 1.31e5     0 1.58e6      0 1.78e7
##  7 1988-07-01  1988 4.68e5 1.96e7 5.08e5 2.49e4  3601. 6.00e5     0 1.20e6 11375. 1.95e7
##  8 1988-08-01  1988 7.36e5 1.50e7 3.04e5 1.13e4   739. 3.47e5     0 1.43e6      0 1.79e7
##  9 1988-09-01  1988 6.06e5 1.76e7 2.04e5 4.10e4 60378. 2.07e5     0 2.17e6      0 1.64e7
## 10 1988-10-01  1988 9.50e5 1.70e7 3.00e5 7.55e4   538. 6.04e5     0 1.13e6      0 1.30e7
## # ... with 362 more rows, and 235 more variables: AS <dbl>, AT <dbl>, AU <dbl>,
## #   AW <dbl>, AZ <dbl>, BA <dbl>, BB <dbl>, BD <dbl>, BE <dbl>, BF <dbl>, BG <dbl>,
## #   BH <dbl>, BI <dbl>, BJ <dbl>, BL <dbl>, BM <dbl>, BN <dbl>, BO <dbl>, BQ <dbl>,
## #   BR <dbl>, BS <dbl>, BT <dbl>, BV <dbl>, BW <dbl>, BY <dbl>, BZ <dbl>, CA <dbl>,
## #   CC <dbl>, CD <dbl>, CF <dbl>, CG <dbl>, CI <dbl>, CK <dbl>, CL <dbl>, CM <dbl>,
## #   CN <dbl>, CO <dbl>, CR <dbl>, CU <dbl>, CV <dbl>, CW <dbl>, CX <dbl>, CY <dbl>,
## #   CZ <dbl>, DE <dbl>, DJ <dbl>, DK <dbl>, DM <dbl>, DO <dbl>, DZ <dbl>, EC <dbl>,
## #   EE <dbl>, EG <dbl>, EH <dbl>, ER <dbl>, ES <dbl>, ET <dbl>, FI <dbl>, FJ <dbl>,
## #   FK <dbl>, FM <dbl>, FO <dbl>, FR <dbl>, GA <dbl>, GB <dbl>, GD <dbl>, GE <dbl>,
## #   GF <dbl>, GH <dbl>, GI <dbl>, GL <dbl>, GM <dbl>, GN <dbl>, GP <dbl>, GQ <dbl>,
## #   GR <dbl>, GS <dbl>, GT <dbl>, GU <dbl>, GW <dbl>, GY <dbl>, HK <dbl>, HM <dbl>,
## #   HN <dbl>, HR <dbl>, HT <dbl>, HU <dbl>, ID <dbl>, IE <dbl>, IL <dbl>, IN <dbl>,
## #   IO <dbl>, IQ <dbl>, IR <dbl>, IS <dbl>, IT <dbl>, JM <dbl>, JO <dbl>, JP <dbl>,
## #   KE <dbl>, ...
```

2. Get the data into long form using the `pivot_longer` function.

```
library(tidyverse)
SwissLong<-pivot_longer(data = SwissWide,
                        cols=c(-Date,-Year), #Do not use these variables
                        names_to = 'Country', #Column names become variable
                        values_to =  'Exports') #All numbers are exports
SwissLong
```

```
## # A tibble: 91,140 x 4
##    Date        Year Country   Exports
##    <date>     <dbl> <chr>       <dbl>
##  1 1988-01-01  1988 AD        167909.
##  2 1988-01-01  1988 AE      13478266.
##  3 1988-01-01  1988 AF        416619.
##  4 1988-01-01  1988 AG          2083.
##  5 1988-01-01  1988 AI             0
##  6 1988-01-01  1988 AL         95239.
##  7 1988-01-01  1988 AM             0
##  8 1988-01-01  1988 AO        660792.
##  9 1988-01-01  1988 AQ          5911.
## 10 1988-01-01  1988 AR      11398162.
## # ... with 91,130 more rows
```

3. Using `group_by` and `summarise` create a new dataset of yearly aggregate exports to each country.

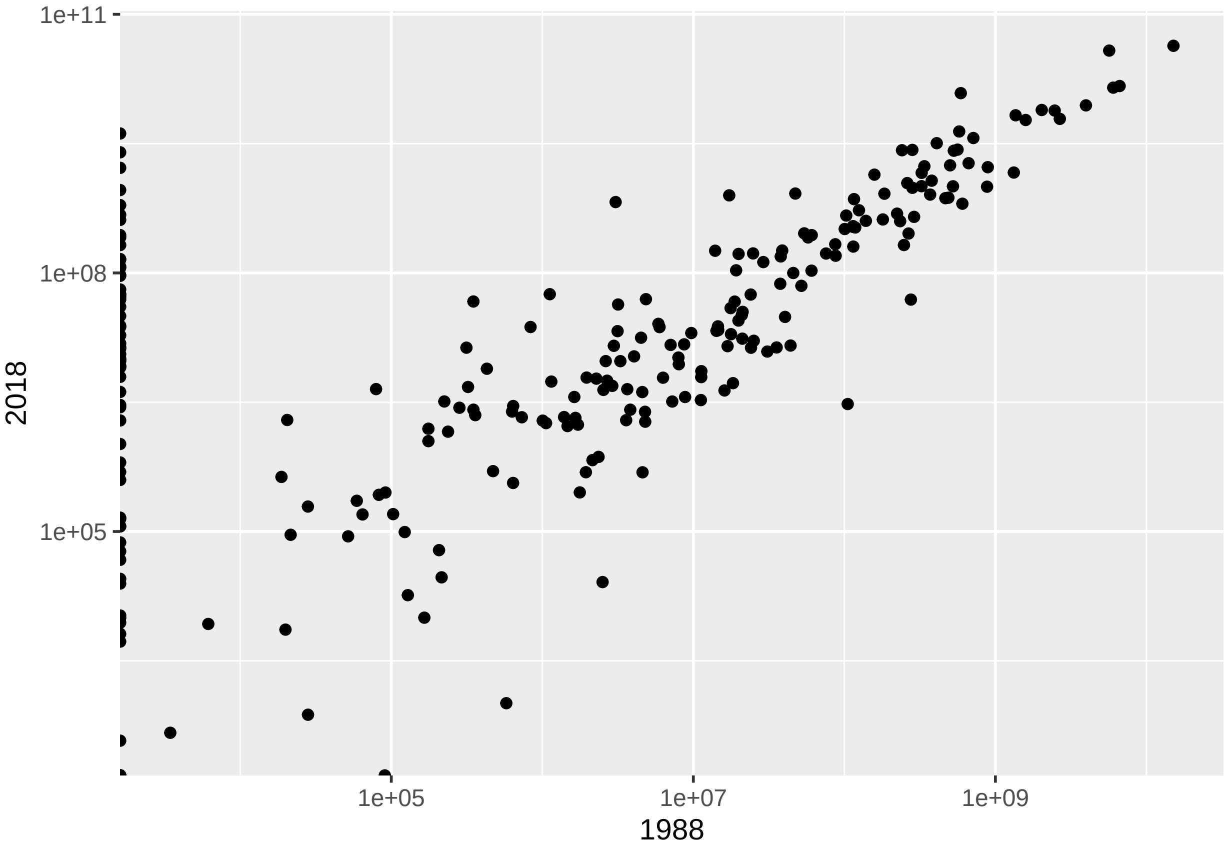
```
SwissLong%>% #This is much easier with long data
  group_by(Year,Country)%>%
  summarise(YearlyExports=sum(Exports))->SwissYearly
SwissYearly
```

```
## # A tibble: 7,595 x 3
## # Groups:   Year [31]
##     Year Country YearlyExports
##    <dbl> <chr>           <dbl>
##  1  1988 AD           6297932.
##  2  1988 AE         240703958.
##  3  1988 AF           3822809.
##  4  1988 AG           1720777.
##  5  1988 AI             82767.
##  6  1988 AL           4852564.
##  7  1988 AM                 0
##  8  1988 AO          14606708.
##  9  1988 AQ             90581.
## 10  1988 AR         184058658.
## # ... with 7,585 more rows
```

4. Now produce a scatterplot on a log-log scale of 1988 exports against 2018 exports.

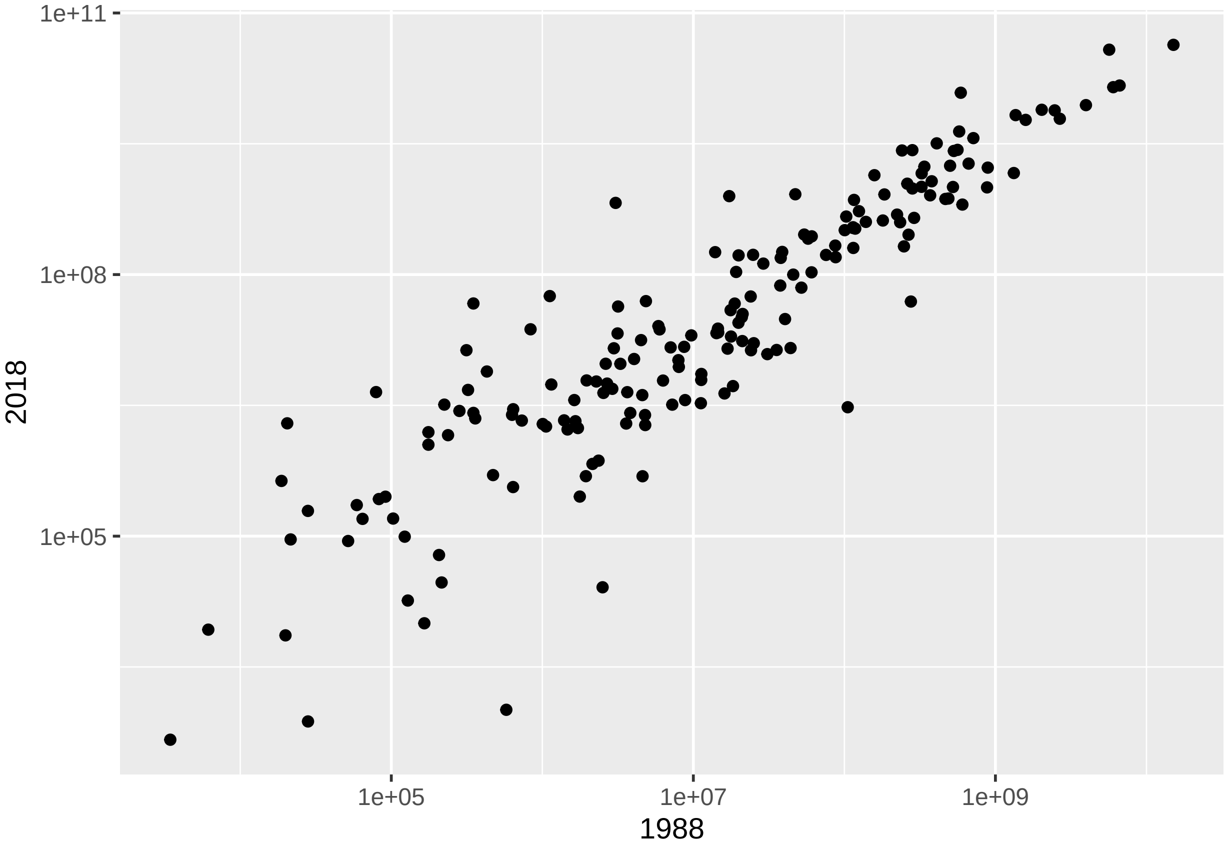
```
SwissYearly%>%
  filter(Year %in% c(1988,2018))%>% #Filter years
  pivot_wider(id_cols = Country,
              names_from = Year,
              values_from = YearlyExports)-> #Need to widen
  SwissYearlyWide

SwissYearlyWide%>%
  ggplot(aes(x=`1988`,y=`2018`))+
  geom_point()+
  scale_x_log10()+scale_y_log10()
```

```
## Warning: Transformation introduced infinite values in continuous x-axis

## Warning: Transformation introduced infinite values in continuous y-axis
```



5. Produce the same plot but remove all countries for which exports are zero in either 1988 or 2018.

```
#Although it works, avoid the temptation to define the logical
#statement as it is written in words !((`1988`==0)|(`2018`==0))

SwissYearlyWide%>%
  filter((`1988`!=0)&(`2018`!=0))%>% #Filter years
  ggplot(aes(x=`1988`,y=`2018`))+
  geom_point()+
  scale_x_log10()+scale_y_log10()
```



Options data

The following example uses Options data from Yahoo Finance. The owner of an put option has the right (but not the obligation) to sell stocks at a predetermined price (the Strike Price) on some fixed date (the Expiry date). A call option is the same but gives the owner the right to buy stocks.

The objective of this exercise is to produce the well-known *volatility smile* result from finance. This result states that for a given Expiry date, a plot of Implied Volatilty against Strike Price is U-shaped. Implied Volatilty is the volatilty of a stock that is computed from stock option data assuming a specific pricing model. The exercise uses the data `apple_options.csv` which can be found on Moodle.

1. Read the data from this csv file into R.

```
apple<-read_csv('apple_options.csv')
```

2. The Implied Volatility has been imported as a character variable. To plot this it must be converted to a numeric variable. Create this using the `mutate` function.

Hint: The following code removes the percentage sign, converts to numeric and divides by 100.

```
gsub('%','','25%')%>%as.numeric()/100
```

```
## [1] 0.25
```

```
gsub('%','','1.32%')%>%as.numeric()/100
```

```
## [1] 0.0132
```

The following code will create the new variable

```r
apple%>%
  mutate(ImpliedVol=gsub('%','',`Implied volatility`)%>%as.numeric()/100)
```

3. The volatility smile is best observed when options with a single expiry date are used. To use as much data as possible, find the expiry date that has the most put options.

```r
apple%>%
  filter(Type=='Put')%>%
  group_by(`Expiry date`)%>%
  summarise(TotalOptions=n())
```

```
## # A tibble: 16 x 2
##    `Expiry date` TotalOptions
##    <date>               <int>
##  1 2020-05-29             120
##  2 2020-06-05             108
##  3 2020-06-12             101
##  4 2020-06-19              94
##  5 2020-06-26              93
##  6 2020-07-02               5
##  7 2020-07-17              92
##  8 2020-09-18              75
##  9 2020-10-16              87
## 10 2020-11-20               2
## 11 2020-12-18              67
## 12 2021-01-15              85
## 13 2021-06-18              73
## 14 2021-09-17              57
## 15 2022-01-21              71
## 16 2022-06-17              59
```

The expiry date with the most options contracts is 2020-05-29 with 120 put options.

4. Options that are very far *out of the money* (very low strike price for a put option) should be excluded from the analysis. Building on previous answers, construct a data frame that only keeps put options from the expiry date in your answer to Question 3, and that have a strike price above 250.

```r
apple%>%
  mutate(ImpliedVol=gsub('%','',`Implied volatility`)%>%as.numeric()/100)%>%
  filter(Type=='Put',`Expiry date`=='2020-05-29',`Strike Price`>250)
```

```
## # A tibble: 74 x 14
##    `Contract name` `Last trade dat~ `Expiry date` `Last price`   Bid   Ask Change
##    <chr>           <chr>            <date>               <dbl> <dbl> <dbl>  <dbl>
##  1 AAPL200529P002~ 2020-05-22 2:12~ 2020-05-29            0.01  0     0.07  -0.06
##  2 AAPL200529P002~ 2020-05-22 3:55~ 2020-05-29            0.02  0.01  0.07  -0.06
##  3 AAPL200529P002~ 2020-05-22 3:30~ 2020-05-29            0.03  0.02  0.03  -0.04
##  4 AAPL200529P002~ 2020-05-22 3:04~ 2020-05-29            0.03  0.03  0.04  -0.07
##  5 AAPL200529P002~ 2020-05-22 9:42~ 2020-05-29            0.09  0     0.1   -0.02
##  6 AAPL200529P002~ 2020-05-22 3:47~ 2020-05-29            0.03  0.02  0.05  -0.12
##  7 AAPL200529P002~ 2020-05-22 3:21~ 2020-05-29            0.05  0     0.06  -0.11
##  8 AAPL200529P002~ 2020-05-22 3:53~ 2020-05-29            0.06  0.04  0.07  -0.1
##  9 AAPL200529P002~ 2020-05-22 3:59~ 2020-05-29            0.06  0.05  0.07  -0.13
## 10 AAPL200529P002~ 2020-05-22 3:58~ 2020-05-29            0.07  0.06  0.08  -0.17
## # ... with 64 more rows, and 7 more variables: `% change` <chr>, Volume <dbl>, `Open
## #   interest` <dbl>, `Implied volatility` <chr>, Type <chr>, `Strike Price` <dbl>,
## #   ImpliedVol <dbl>
```

Note that the filter function could use the AND operator as well.

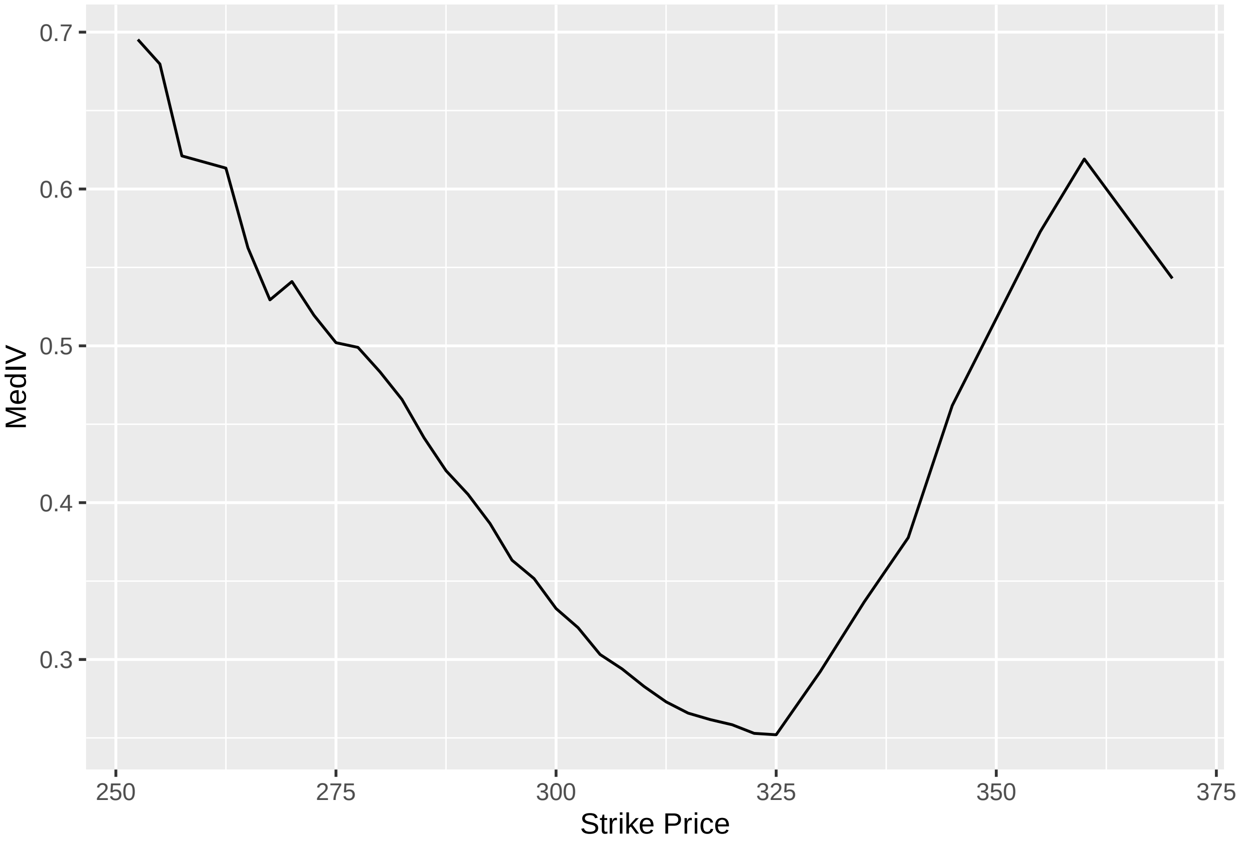
5. Using the data constructed in Q4, find the median value of Implied Volatility for each Strike Price.

```
apple%>%
  mutate(ImpliedVol=gsub('%','',`Implied volatility`)%>%as.numeric()/100)%>%
  filter(Type=='Put',`Expiry date`=='2020-05-29',`Strike Price`>250)%>%
  group_by(`Strike Price`)%>%
  summarise(MedIV=median(`ImpliedVol`,na.rm = T))
```

```
## # A tibble: 37 x 2
##    `Strike Price` MedIV
##             <dbl> <dbl>
##  1           252. 0.695
##  2           255  0.680
##  3           258. 0.621
##  4           260  0.617
##  5           262. 0.613
##  6           265  0.562
##  7           268. 0.529
##  8           270  0.541
##  9           272. 0.520
## 10           275  0.502
## # ... with 27 more rows
```

6. Plot Implied Volatility against Strike Price using a line plot.

```
apple%>%
  mutate(ImpliedVol=gsub('%','',`Implied volatility`)%>%as.numeric()/100)%>%
  filter(Type=='Put',`Expiry date`=='2020-05-29',`Strike Price`>250)%>%
  group_by(`Strike Price`)%>%
  summarise(MedIV=median(`ImpliedVol`,na.rm = T))%>%
  ggplot(aes(x=`Strike Price`,y=MedIV))+geom_line()
```



Web scraping

The options data were obtained using web scraping, the following exercise helps you to scrape data for a single strike price. Go to this link and then click on the first strike price. Scrape the data that you find after clicking on a strike price.

The exact link will depend on when the data is retrieved but will be similar to https://au.finance.yahoo.com/quote/AAPL/options?strike=180&straddle=false

The following code scrapes the data.

```
#The following line should match the exact URL
url<-'https://au.finance.yahoo.com/quote/AAPL/options?strike=180&straddle=false'

url%>%
    read_html%>%
    html_table()->data
```

This returns a list with first element:

```
## # A tibble: 12 x 11
##    `Contract name` `Last trade dat~ `Expiry date` `Last price`   Bid   Ask Change
##    <chr>           <chr>            <chr>                <dbl> <dbl> <dbl>  <dbl>
##  1 AAPL200612C001~ 2020-05-07 6:37~ 2020-06-12            123.   149  153.      0
##  2 AAPL200619C001~ 2020-06-05 12:4~ 2020-06-19            150.     0     0      0
##  3 AAPL200717C001~ 2020-05-22 9:50~ 2020-07-17            137.     0     0      0
##  4 AAPL200918C001~ 2020-06-08 2:27~ 2020-09-18            153.     0     0      0
##  5 AAPL201016C001~ 2020-05-18 9:38~ 2020-10-16            134.     0     0      0
##  6 AAPL201218C001~ 2020-03-31 10:2~ 2020-12-18            85.9  113.  115.       0
##  7 AAPL210115C001~ 2020-06-05 2:39~ 2021-01-15           151.     0     0       0
##  8 AAPL210618C001~ 2020-06-03 3:19~ 2021-06-18           148.     0     0       0
##  9 AAPL210917C001~ 2020-05-29 1:01~ 2021-09-17           144.     0     0       0
## 10 AAPL220121C001~ 2020-05-28 10:2~ 2022-01-21           146      0     0       0
## 11 AAPL220617C001~ 2020-06-08 11:2~ 2022-06-17           156.     0     0       0
## 12 AAPL220916C001~ 2020-05-29 2:31~ 2022-09-16           149.     0     0       0
## # ... with 4 more variables: `% change` <chr>, Volume <chr>, `Open interest` <chr>,
## #   `Implied volatility` <chr>
```

First Normal Form

Discuss whether the following databases satisfy first normal form.

Database A:

Name	Social Media Username
Jane Smith	Facebook: jsChampion
Kamal Usman	Twitter: kusman, LinkedIn: ku87
Li Xiao	WeChat: lx99

Database B:

Name	Social Media Username
Jane Smith	Facebook: jsChampion
Kamal Usman	Twitter: kusman
Kamal Usman	LinkedIn: ku87
Li Xiao	WeChat: lx99

Database A has the same issue as seen in lectures. Kamal Usman has two social media accounts so the entry is not atomic. The second Database resolves this issue. However arguably the variable Social Media Username is still not atomic. There are two separate pieces of information, the social media platform and the username. A better database would store these as two separate variables.

Another point worth mentioning is that even the variable name might not be considered to be atomic. For example in some contexts it may be important to serparate family name from given names.