Cybersecurity Policy: Data Management and Visulization for Cyber Attacks

Working on actual problems is central to learning. This is your first problem set. The assignments consist of analysis of cyber attacks using basic data management approaches and visualization tools (in R). Late submissions will not be accepted without prior permission. Students are encouraged to discuss the problems together, but must independently produce and submit solutions.

Show your solutions in the chunks below.

Before we dive in

Check 'Intro to Problem Set 0' video on the class website

Familiarize yourself with RMarkdown

Check this video to familiarize yourself with RMarkdown interface

Rename this file

You are in the `Problem-Set-0.Rmd` file now. First, close this file and rename it using your last and first name: `Last-Name-First-Name-Problem-Set-0.Rmd`. Reopen the file. Good!

0 Getting started

Loading packages

Loading packages is boring and time-consuming. First, you need to install packages. Second, you need to run them in R's environment. Delete # before `install.packages("pacman")` and run this chunk of code. Now this package is installed on your system. Put # back.

```
# install.packages("pacman")
# library("pacman")
```

There is and easy way: `pacman` package. This package checks if the package you want (say `dplyr`) has been installed already on your laptop and upload it. No need to use quotes in this package. The following chunk should install and upload all packages that you might need for this problem set project. Just execute it.

Loading datasets

We will use four datasets for this problem set. I describe them in detail later. First, we need to upload these datasets in R memory and store them as data-objects.

Here is the list of data-objects we need and their corresponding files:

Data objects for this assignment	File
`d`	"cyberattacks-across-the-globe-cases.csv"
`d.attacks.by.year`	"cyberattacks-by-year.csv"
`d.attacks.by.year.and.method`	"cyberattacks-by-year-and-method.csv"
`d.attacks.by.attack_on`	"cyberattacks-by-attack_on.csv"

All files are stored in your "0-Data" subfolder. To reed them and store them we use `fread()` function from `data.table` package:

```
d <- fread("0-Data/cyberattacks-across-the-globe-cases.csv")
d.attacks.by.year <- fread("0-Data/cyberattacks-by-year.csv")
d.attacks.by.year.and.method <- fread("0-Data/cyberattacks-by-year-and-method.csv")
d.attacks.by.attack_on <- fread("0-Data/cyberattacks-by-attack_on.csv" )
```

Now, we can start! Good luck!

1 Sources and targets of cyber-attacks *(1 point)*

1.a Explore objects with data on cyber attacks

We start with the most detailed dataset: `d`. Each row in `d` represents a cyber-attack. In the chunk below, use function `names()` to print the names of variables (columns) in `d`

```
# put your answer here
```

What do we know about each attack?

`source` - a territory (country) from which the attack was organized

`target` - a territory (country) where the target (e.g., a private firm or a state agency) was located

`year` - when the attack had a place. Later in this class, we will use precise dates of attacks

`attack_on` - who was under attack (private firms, state agencies, or military objects)

`method`- a method used for this attack

`success` - a dummy-variable. It takes a value of 1 if attackers achieved their goals (money, concessions, physical damage, etc) and 0 otherwise

`num` - another dummy variable. It always takes a value of 1. We will often use it to aggregate data.

We can check what is inside any data object in different ways. For example, we can use `glimpse` (from `dplyr` package) to check the number of rows and columns, names and types of the columns (variables), as well as some examples from this columns.

```
glimpse(d)
```

```
## Rows: 266
## Columns: 7
## $ source    <chr> "US", "US", "Russia", "Russia", "Russia", "US", "Russia", "R~
## $ target    <chr> "Russia", "Russia", "US", "US", "US", "Russia", "US", "US", ~
## $ year      <int> 2008, 2008, 2008, 2008, 2008, 2008, 2009, 2009, 2011, 2013, ~
## $ attack_on <chr> "Government", "Government", "Government", "Government", "Mil~
## $ method    <chr> "Intrusion", "Infiltration", "Infiltration", "Infiltration",~
## $ success   <int> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ num       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
```

In fact, we can print the content of `d` by putting it in the chunk without any functions (check it by deleting '#' in the chunk below. Do not forget to put '#' back). Do not do it in future. R will print all rows of `d` and your reports will be impossible to read.

```
# d
```

In the chunk below, check the structure of your `d` using at least 2 functions you learnt in DataCamp's "Intro to R".

```
# put your answer here
```

1.b Explore cyberattacks with: Vectors

Let's explore targets and sources of cyberattacks. For example, to obtain a vector of countries that were under attack at least once, we can use `unique()` on variable `target` inside `d`:

```
target.countries <- unique(d$target)
target.countries
```

```
##  [1] "Russia"       "US"           "Iran"         "China"        "N Korea"
##  [6] "Canada"       "UK"           "France"       "Germany"      "Poland"
## [11] "Estonia"      "Lithuania"    "Ukraine"      "Georgia"      "Turkey"
## [16] "Israel"       "Saudi Arabia" "Syria"        "Lebanon"      "Taiwan"
## [21] "Japan"        "India"        "Vietnam"      "Philippines"  "S Korea"
## [26] "Pakistan"
```

In the following chunk, use `d` to create vector `source.countries` that will contain all territories from which at least one attack was organized.

```
# put your answer here
```

It is hard to compare these two vectors. To ease this comparison, we can `sort()` territories alphabetically (increasing order) or in the reverse order (decreasing order). You choose the order by specifying option `decreasing` inside `sort()`

```
target.countries <- sort(target.countries, decreasing = FALSE)
target.countries
```

```
##  [1] "Canada"       "China"        "Estonia"      "France"       "Georgia"
##  [6] "Germany"      "India"        "Iran"         "Israel"       "Japan"
## [11] "Lebanon"      "Lithuania"    "N Korea"      "Pakistan"     "Philippines"
## [16] "Poland"       "Russia"       "S Korea"      "Saudi Arabia" "Syria"
## [21] "Taiwan"       "Turkey"       "UK"           "Ukraine"      "US"
## [26] "Vietnam"
```

```
target.countries <- sort(target.countries, decreasing = TRUE)
target.countries
```

```
##  [1] "Vietnam"      "US"           "Ukraine"      "UK"           "Turkey"
##  [6] "Taiwan"       "Syria"        "Saudi Arabia" "S Korea"      "Russia"
## [11] "Poland"       "Philippines"  "Pakistan"     "N Korea"      "Lithuania"
## [16] "Lebanon"      "Japan"        "Israel"       "Iran"         "India"
## [21] "Germany"      "Georgia"      "France"       "Estonia"      "China"
## [26] "Canada"
```

Sort `source.countries` in the reverse alphabetical order and store the results in the same object:

```
# put your answer here
```

Do these two vectors (`target.countries` and `source.countries`) look alike? Are `source`-territories and `target`-territories often the same? One way to check it is to look at the overlap of the two vectors. In the following chunk, create a vector `target.source.intersection` of countries that present both in `target.countries` and `source.countries`. You can do it manually or using functions like `intersect()`. Also, create a vector `all.countries` of all unique territories presented either in `target.countries` or in `source.countries` (in other words create a union of unique elements from `target.countries` and `source.countries`). *Hint: remember, you can always combine vectors by applying function* `c()`

Looks good!

Question. So what do you see? Are `source`-territories and `target`-territories tend to be the same? Write your answer below.

Your Answer:

To support your answer with some evidence, calculate the share of territories in `target.source.intersection` to all the territories in `all.countries`:

```
# put your answer here
```

2 Explore cyberattacks with: `table()` *(1 point)*

We often want to narrow our focus to specific cases. For example, we are only interested in the territories that experienced a lot of attacks. Function `table()` will calculate the number of time each territory appeared in column `target`.

```
table(d$target) # check the results without storing them in an object
```

```
##
##       Canada        China      Estonia       France      Georgia      Germany
##            2            7            4            3            6            3
##        India         Iran       Israel        Japan      Lebanon    Lithuania
##           20           14           11           14            1            4
##      N Korea     Pakistan  Philippines       Poland       Russia      S Korea
##            5            7            5            3           11           23
## Saudi Arabia        Syria       Taiwan       Turkey           UK      Ukraine
##            7            1            7            4            3           15
##           US      Vietnam
##           82            4
```

```
target.by.frequency <- table(d$target) # store the results in an object
target.by.frequency # check what is inside this object
```

```
##
##       Canada        China      Estonia       France      Georgia      Germany
##            2            7            4            3            6            3
##        India         Iran       Israel        Japan      Lebanon    Lithuania
##           20           14           11           14            1            4
##      N Korea     Pakistan  Philippines       Poland       Russia      S Korea
##            5            7            5            3           11           23
## Saudi Arabia        Syria       Taiwan       Turkey           UK      Ukraine
##            7            1            7            4            3           15
##           US      Vietnam
##           82            4
```

By examining `target.by.frequency` we see that top 3 countries on the list inlcude "US", "S Korea", and "India". We can store them in a new object manually:

```
target.by.frequency.top.3 <- c("US", "S Korea", "India")
target.by.frequency.top.3
```

```
## [1] "US"      "S Korea" "India"
```

We could also specify the position of these three elements inside `target.by.frequency`:

```
target.by.frequency.top.3 <- target.by.frequency[c(7,18,25)]
target.by.frequency.top.3
```

```
##
```

```
##   India S Korea      US
##      20      23      82
```

But there is a difference. `table()` basically creates a vector of elements. Each element reflect the number of times a territory occurred in our input column `d$target`. Each element also has a name. To obtain just the list of names, we need to use `names()` on `target.by.frequency.top.3`:

```
names(target.by.frequency.top.3)
```

```
## [1] "India"   "S Korea" "US"
```

```
target.by.frequency.top.3 <- names(target.by.frequency.top.3)
target.by.frequency.top.3
```

```
## [1] "India"   "S Korea" "US"
```

Use `table()` on `d$sources` to create vector `source.by.frequency`. Next, manually create a list of top-3 territories that launched attacks. Store the results in `source.by.frequency.top.3`:

```
# put your answer here
```

Now, re-write `source.by.frequency.top.3` by specifying the positions of top-3 elements in `source.by.frequency`:

```
# put your answer here
```

Nice! But can we obtain the top-3 territories in an automated way? Yes. For example, we can use `sort()` on `target.by.frequency` and subset first or last elements of this vector. The following chunk obtains a list of three territories that experienced the *least* number of attacks: *Note: the elements in `target.by.frequency` are numbers. So, `sort()` will use them for ordering (not the names of territories)*

```
target.by.frequency <- table(d$target) # Obtain number of attacks for each territory
target.by.frequency <-  sort(target.by.frequency) # Sort the vector, if 'decreasing = TRUE' is not spec
target.by.frequency.bottom.3 <- target.by.frequency[1:3] # Extract the first three elements
target.by.frequency.bottom.3 # check what is inside the object
```

```
##
## Lebanon   Syria  Canada
##       1       1       2
```

```
target.by.frequency.bottom.3 <- names(target.by.frequency.bottom.3) # extract the names of the elements
target.by.frequency.bottom.3 # check what is inside the object again.
```

```
## [1] "Lebanon" "Syria"   "Canada"
```

OK! Now let's put together things we've learnt so far. In the following chunk, create vectors `target.by.frequency.top.5` and `source.by.frequency.top.5`. `source.by.frequency.top.5` should contain names of top-5 territories that launched attacks; `target.by.frequency.top.5` should contain names of top-5 territories that experienced cyberattacks.

```
# put your answer here
```

The other important feature of `table()` is to contrast variables against each other. For example, we can contrast different methods of cyberattacks with the type of targets:

```
table(d$method, d$attack_on)
```

```
##
##                Government Military Private
##   DDoS                 24        6      16
##   Defacement           20        1       7
##   Infiltration         18       18      12
##   Intrusion            70       24      50
```

Question. What is the most common method used against the government agencies? Military? Private? What is the most common cyber method overall?

Your Answer:

Use `table()` to check how successful are cyberattacks (variable `success` in `d`) against different targets (variable `attack_on`)

```
# put your answer here
```

Question. Attackers have higher chances when they attack ...

Your Answer:

Now, analyze what methods are usually more successful than the others (variable `method` in `d`)

```
# put your answer here
```

Question. Calculate the ratio of successful attacks for each method ...

Your Answer:

3 Subset data on cyber attacks *(1 point)*

We often conduct the analysis on a subset of data. One way to subset data is to specify the value of some variable in our dataset. (Remember, that in this case we need to use `==` not `=`.) For example, we can subset all observations from 2012 using `data.table` syntax:

```
d.2012 <- d[year ==  2012,]
# Note 1: Because class of `d` is `data.table` we do not need to specify the data object inside the squ
# If we use some old school formats like `data.frame`, we will need to be explicit: d[d$year == 2012,]
# Note 2: Do not forget to put comma after your logical expression. In this way, R understand that we a
```

The same is true, if we want to focus on a specific territory. Here is an example for India as a target:

```
d.india <- d[target == "India",]
```

We can also make complex logical expressions with '&' and subset on them:

```
d.india.2012 <- d[target == "India" & year == 2012,]
d.india.2012
```

```
##    source target year attack_on    method success num
## 1:  China  India 2012   Private Intrusion       0   1
```

We can also subset with multiple values. The operator `%in%` is our good friend here:

```
countries.of.interest <- c("S Korea", "N Korea")
d.subset <- d[target %in% countries.of.interest,]
head(d.subset)
```

```
##     source  target year  attack_on       method success num
## 1:      US N Korea 2008 Government         DDoS       1   1
## 2:      US N Korea 2010   Military Infiltration       0   1
## 3:      US N Korea 2015   Military Infiltration       0   1
## 4: S Korea N Korea 2008 Government Infiltration       0   1
## 5: N Korea S Korea 2007   Military Infiltration       0   1
## 6: N Korea S Korea 2009 Government         DDoS       0   1
```

Now, subset three territories that experienced the least number of attacks (remember `target.by.frequency.bottom.3`?). Also, only keep observations from 2010 to 2014. Store the results in `d.bottom.3.from.2010.to.2014`.

```
# put your answer here
```

How many observations do you have in `d.bottom.3.from.2010.to.2014`? Use `nrow()` on your data object to answer this question.

```
# put your answer here
```

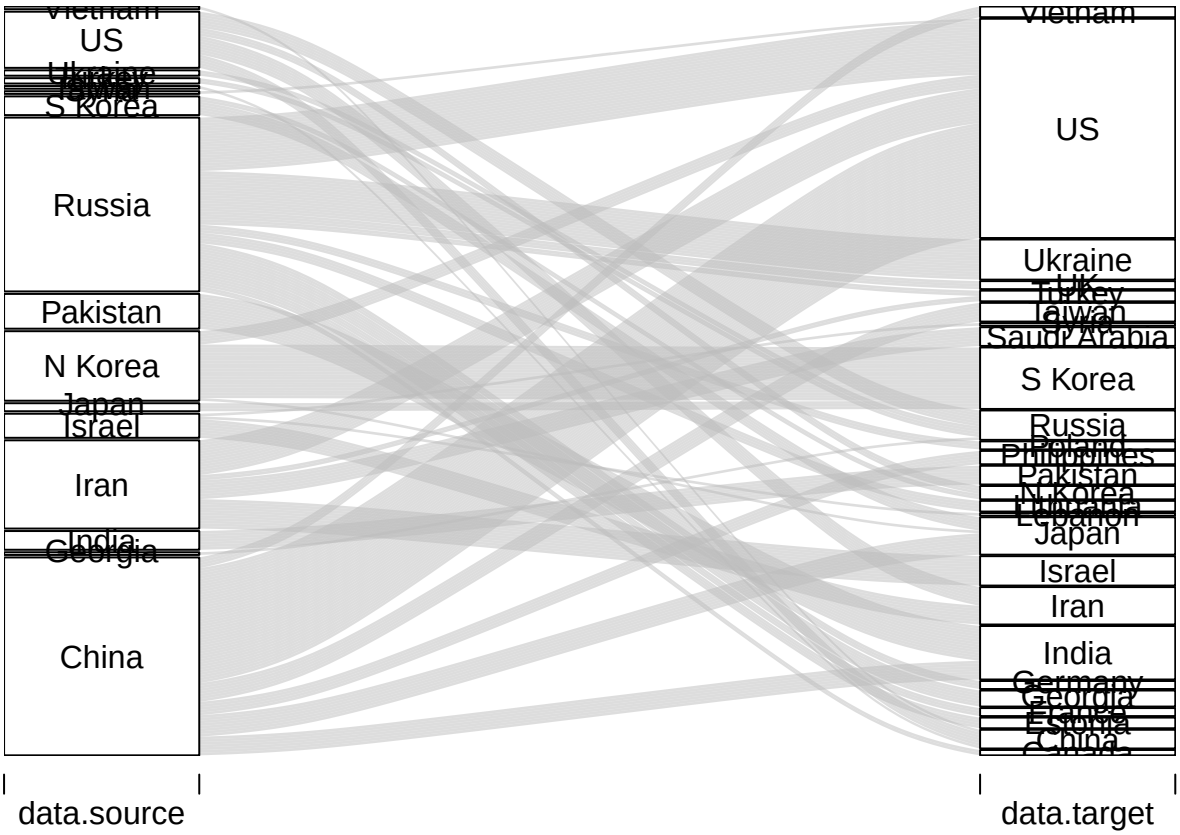
4 Explore sources and targets with Sankey Diagrams *(1 point)*

Tables are great. But it is usually hard to infer systematic patterns in your data while looking at them. For example, do the territories usually mirror each other attacks? A simple way to check it is to produce a Sankey Diagram.

To plot our Sankey Diagrams we will use function `alluvial()` from `alluvial` package (yes, the names are confusing, but it is what it is). Inside `alluvial()` we need to specify to options. For `data` we need to specify which columns represent targets and sources. We can reference them by their names in the dataset `data = d[,c("source","target")]`. Alternatively, we can reference them by their position `data = d[,1:2]`.

We also need to specify frequency with `freq = d$num`. This option is useful when each row represents not a single attack, but for example all the attacks from `source == "Country A"` to `target == "Country B"`. It is not our case, yet we are required to specify this option.

```
alluvial(data = d[,1:2], freq=d$num)
```



Well, this looks awful and messy. This is because we have a lot territores that appear in `d` only a couple of times. Instead, let's subset data and keep only territories appeared either in `target.by.frequency.top.5`, or in `source.by.frequency.top.5`. Store this subset of data in in `d.for.alluvial`.

```
# put your answer here
```

Now, make your Sankey Diagram with `alluvial()` using `d.for.alluvial` as your input dataset.

```
# put your answer here
```

Question. Describe interesting patterns you see in this diagram. And what about attacks? Do we see the symmetry?

Your Answer:

5 Explore trends in cyberattacks with `ggplot` *(1 point)*

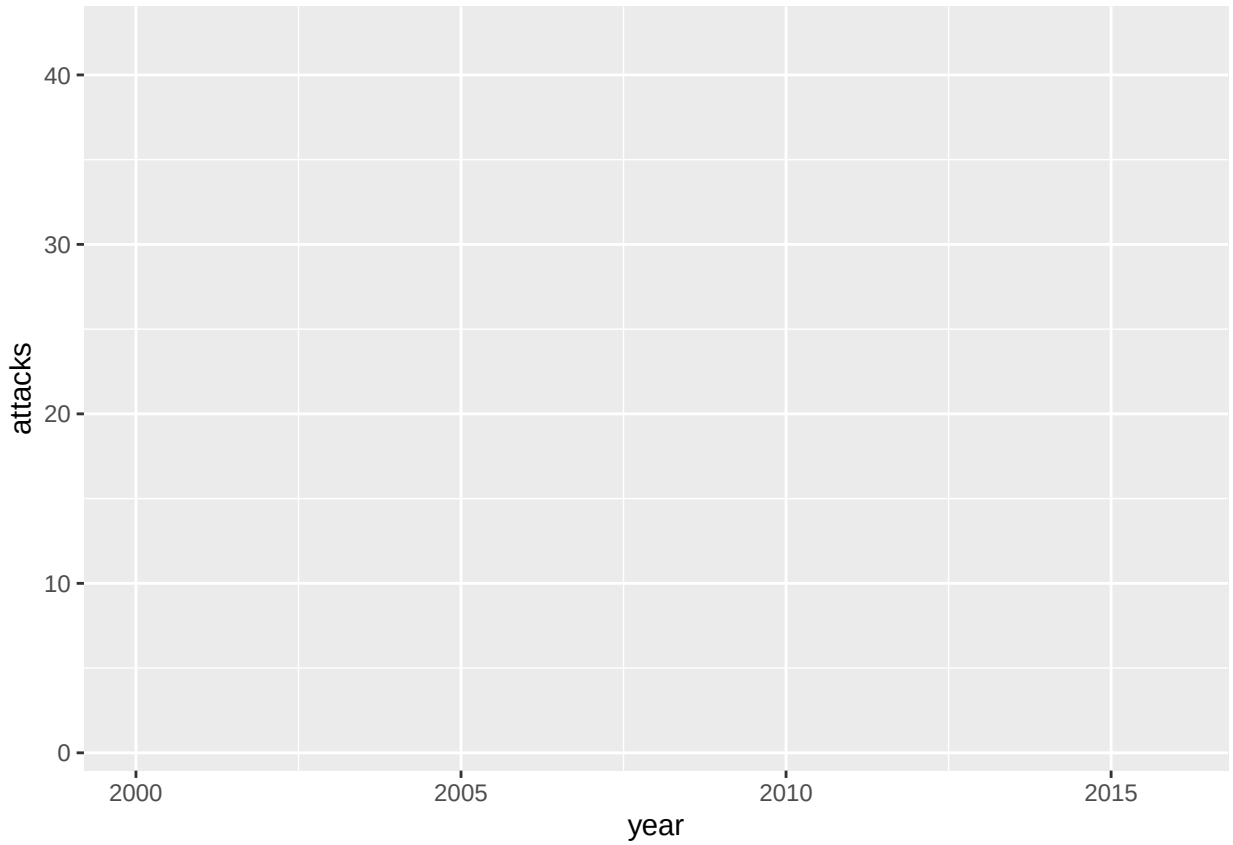
Finally, we can use visual tools to explore trends in attacks with `ggplot` function. For example, we can look how the number of attack changes from year to year. We will use `d.attacks.by.year`. This dataset has only two columns `year` and `attacks`:

```
head(d.attacks.by.year)
```

```
##    year attacks
## 1: 2008      27
## 2: 2009      28
## 3: 2011      27
## 4: 2013      20
## 5: 2014      42
## 6: 2015      34
```

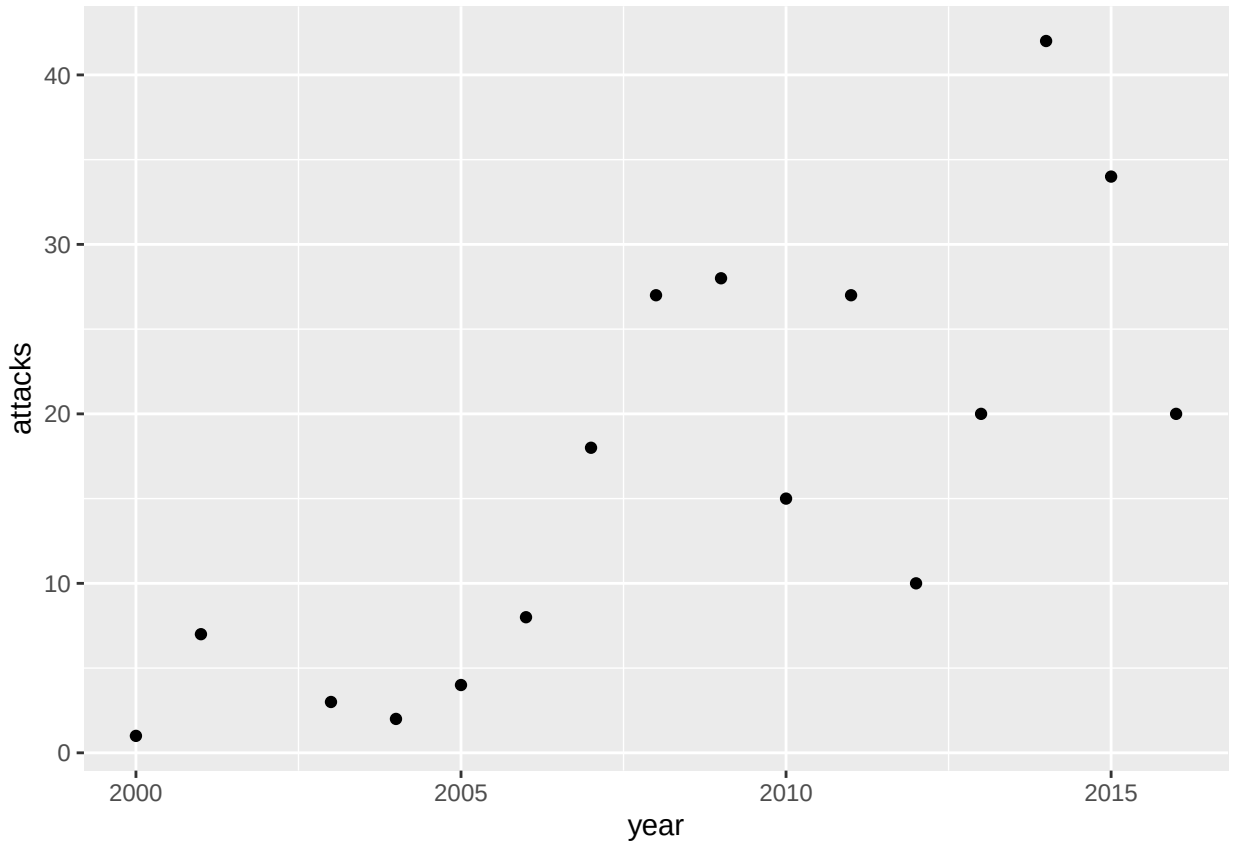
`ggplot` works like Lego. You add up different elements to build your final figure. First you need to specify your baseplate. Your baseplate is `ggplot()`. It has two major options: `data` (works like in `alluvial()`, `names()` and other functions) and `aes()` – stands for aesthetics. Within `aes()` we specify the parameters of the plot we need, like x-axis or y-axis:

```
ggplot(data = d.attacks.by.year,
       aes(x = year, y = attacks)
       )
```



We specified the baseplate but it is blank. This is because we can use a different layers to visualize our data-points. For example, we can add `geom_point()` layer to produce points:

```
ggplot(data = d.attacks.by.year,
       aes(x = year, y = attacks)
       ) +
  geom_point()
```

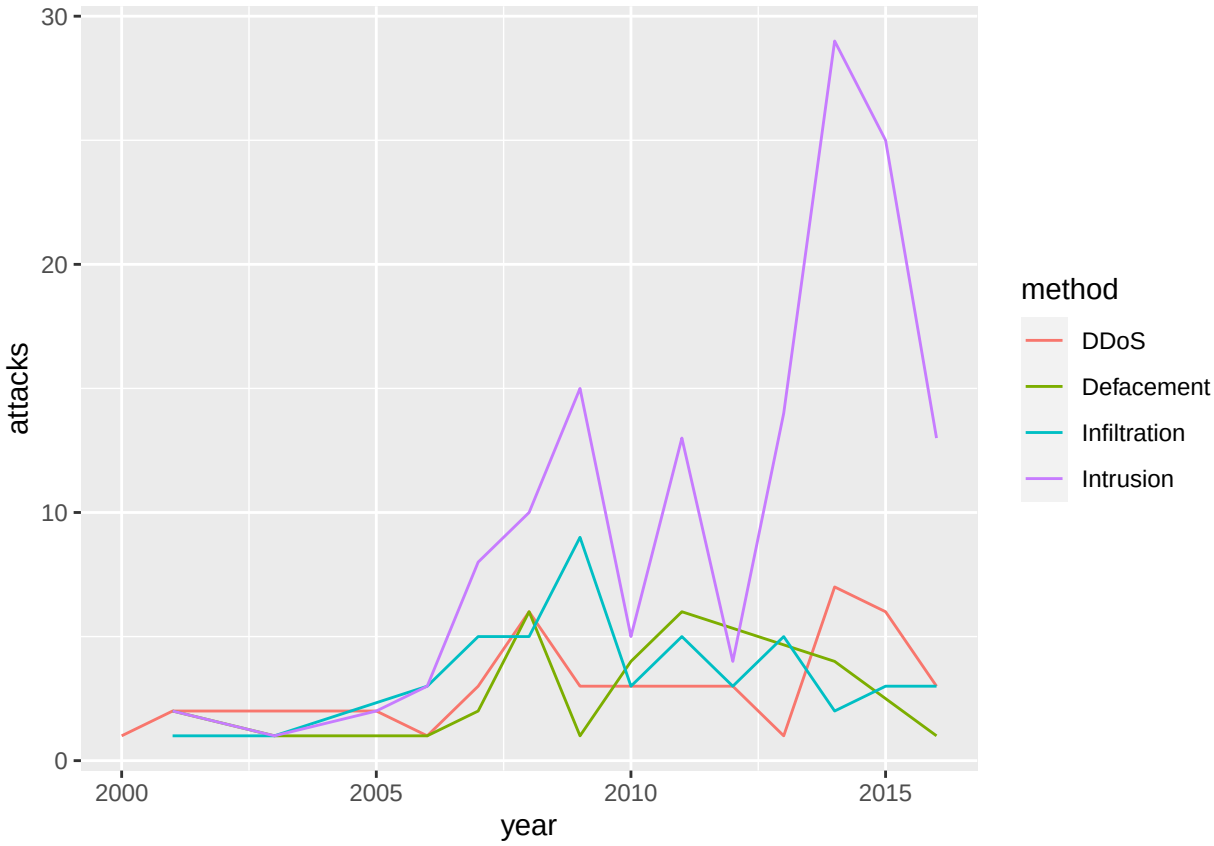


Use `geom_line()` instead of `geom_point()` and reproduce our plot from the previous chunk:

```
# put your answer here
```

We can also specify that our observations are organized as groups. We use `color = group_variable` to produce separate trends for each of the group. Let's take `d.attacks.by.year.and.method` and make trends separately for each method used for cyberattacks:

```
ggplot(data = d.attacks.by.year.and.method,
       aes(x = year, y = attacks, color = method) # We explicitly require ggplot to treat observations
       ) + # Use the plus to add layers and other elements to the baseplate. This plus should always ap
  geom_line()
```



Question. Describe what you see here

Your Answer:

For the final task we will use data from `d.attacks.by.attack_on`. This dataset summarizes cyberattacks according to the type of their target. As in our main dataset `attack_on` in `d.attacks.by.attack_on` has three values: "Government", "Military", and "Private". In the chunk below, use `ggplot` to plot the year-trends in the number of attacks by each type of their target. Your plot should only show the attacks on private firms and government agencies (you will need to exclude military). Also, plot the results only for years from 2009 to 2014. Your plot should include both `geom_line` and `geom_point` layers.

```
# put your answer here
```

That's it!

Now use triangle at the 'knit' button to compile html version of your report. Submit your html file to the class website under "Problem Set 0". If you cannot compile html that means you have some errors in your code. R console shows you the line with an error. Check it and try to compile your report again. If (after spending some time) you cannot compile your html, submit your Rmd file for partial credit.

Good job!