Algoritma Academy: Data Visualization

Samuel Chan

17 September, 2021

Before you go ahead and run the codes in this coursebook, it's often a good idea to go through some initial setup. Under the *Libraries and Setup* tab you'll see some code to initialize our workspace, and the libraries we'll be using for the projects. You may want to make sure that the libraries are installed beforehand by referring back to the packages listed here. Under the *Training Focus* tab we'll outline the syllabus, identify the key objectives and set up expectations for each module.

Background

Algoritma

The following coursebook is produced by the team at Algoritma for its Data Science Academy workshops. The coursebook is intended for a restricted audience only, i.e. the individuals and organizations having received this coursebook directly from the training organization. It may not be reproduced, distributed, translated or adapted in any form outside these individuals and organizations without permission.

Algoritma is a data science education center with bootcamp programs offered in:

- Bahasa Indonesia (Jakarta campus)
- English (Singapore campus)

Lifelong Learning Benefits

If you're an active student or an alumni member, you also qualify for all our future workshops, 100% free of charge as part of your **lifelong learning benefits**. It is a new initiative to help you gain mastery and advance your knowledge in the field of data visualization, machine learning, computer vision, natural language processing (NLP) and other sub-fields of data science. All workshops conducted by us (from 1-day to 5-day series) are available to you free-of-charge, and the benefits **never expire**.

Second Edition

This coursebook is initially written in 2017.

This is the second edition, written in late August 2020. Some of the code has been refactored to work with the latest major version of R, version 4.0. I would like to thank the incredible instructor team at Algoritma for their thorough input and assistance in the authoring and reviewing process.

Libraries and Setup

We'll set-up caching for this notebook given how computationally expensive some of the code we will write can get.

```
options(scipen = 9999)
rm(list=ls())
```

You will need to use `install.packages()` to install any packages that are not already downloaded onto your machine. You then load the package into your workspace using the `library()` function:

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.3
```

```
library(GGally)
```

```
## Warning: package 'GGally' was built under R version 4.0.3
```

```
library(ggthemes)
```

```
## Warning: package 'ggthemes' was built under R version 4.0.2
```

```
library(ggpubr)
library(leaflet)
```

```
## Warning: package 'leaflet' was built under R version 4.0.2
```

```
library(lubridate)
```

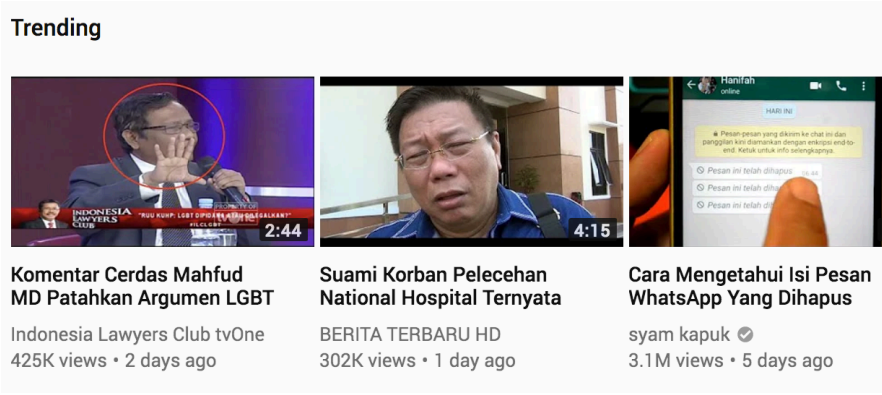
The data we'll be working on is a rather recent Youtube Trending Videos dataset[1]. It has 13,400 records of trending videos between 14th November 2017 to 21st January 2018, and on each record of trending video is a list of variables:

General information relating to video
- Trending date
- Title (video title)
- Channel Title
- Category ID
- Publish Time
- Comments (Disabled?)
- Ratings (Disabled?)
- Video error or removed?

Statistics on a particular date - Views
- Likes
- Dislikes
- Comment Count

[1]This dataset is first contributed under CC0 public domain by Mitchell J (datasnaek on Kaggle), and maintained by other contributors. It has 13,400 records of trending videos between 14th November 2017 to 21st January 2018.

Illustration of a "Trending" section on YouTube:



Training Objectives

The primary objective of this course is to provide a fun and hands-on session to help participants gain full proficiency in data visualization systems and tools. You will learn to create compelling narratives by combining charting elements with a rich grammar under the guidance of the lead instructor and our team of teaching assistants.

- **Plotting Essential**
- Revision: Built-in Plotting Functionalities
- Revision: Scatterplot, Histogram, Line and Column Bars
- Axis, Title and Panel Styles
- Grammar of Graphics
- `ggplot2` Basics
- **Plotting Better**
- Using Themes
- Multi-dimensional Faceting
- Visualizing Geo-Spatial Data with `leaflet`
- Lattice Plotting system

By the end of the workshop, Academy students can choose to complete either of the Learn-By-Building modules as their graded assignment:

Ready for Publication
Applying what you've learned, create a visualization that is polished with the appropriate annotations, aesthetics and some simple commentary. This can be any visualization using the YouTube dataset, but it should communicate a story.

Interactive Map
Create a web page with an interactive map embedded on it. Use a custom icon for the map markers to represent business locations, and show details about each location pin ("markers") upon user's interaction.

This graded assignment is worth **(2) Points**.

Plotting Essentials

R as a statistical computing environment packs a generous amount of tools allowing us to reshape, clean and visualize our data through its built-in capabilities. In the first part of this coursebook, we'll take a look at many of these capabilities and learn how to incorporate these into our day-to-day data science work.

In the second part of this coursebook, we'll shift our focus onto `ggplot`, a plotting system by Hadley Wickham. As you'll see in this 3 days workshop, this plotting system is among the most popular visualization tools today because of its power, extensibility and simplicity (an unlikely combination).

To get started with plotting in R, let's start by reading our data into the environment:

```
vids <- read.csv("data_input/USvideos.csv")
names(vids)
```

```
##  [1] "trending_date"          "title"                 "channel_title"
##  [4] "category_id"            "publish_time"          "views"
##  [7] "likes"                  "dislikes"              "comment_count"
## [10] "comments_disabled"      "ratings_disabled"      "video_error_or_removed"
```

Taking a quick peek at the `trending_date` column reveals that the date values are stored in a year-day-month format, with `.` (dot) being the delimiter:

```
head(vids$trending_date)
```

```
## [1] "17.14.11" "17.14.11" "17.14.11" "17.14.11" "17.14.11" "17.14.11"
```

Because these values follows the `yy.dd.mm` format, our data processing steps would handle this format accordingly through the `format` argument:

```
vids$trending_date <- as.Date(vids$trending_date, format="%y.%d.%m")
```

In real life, most date formats uses a combination that maps to the following:

- `%Y`: 4-digit year (1982)
- `%y`: 2-digit year (82)
- `%m`: 2-digit month (01)
- `%d`: 2-digit day of the month (13)
- `%A`: weekday (Wednesday)
- `%a`: abbreviated weekday (Wed)

- `%B`: month (January)
- `%b`: abbreviated month (Jan)

You should follow your lead instructor on a few in-classroom practice to increase your familiarity with the `as.Date()` function, as processing dates are a fairly common process in many data analysis routines:

```
# Demo:
as.Date("Aug 30,2020", format = "%b %d,%Y")

# Dive Deeper (complete the following):
as.Date("30aug20", format = ___)
as.Date("2020-08-30", format = ___)
as.Date("08.30.2020", format = ___)
```

The raw dataset does not have the proper names for each category, but identify them by an "id" instead. The following code chunk "switches" them by "id" and also convert that to a factor. We will also convert our video titles to a character vector:

```
vids$category_id <- sapply(as.character(vids$category_id), switch,
                           "1" = "Film and Animation",
                           "2" = "Autos and Vehicles",
                           "10" = "Music",
                           "15" = "Pets and Animals",
                           "17" = "Sports",
                           "19" = "Travel and Events",
                           "20" = "Gaming",
                           "22" = "People and Blogs",
                           "23" = "Comedy",
                           "24" = "Entertainment",
                           "25" = "News and Politics",
                           "26" = "Howto and Style",
                           "27" = "Education",
                           "28" = "Science and Technology",
                           "29" = "Nonprofit and Activism",
                           "43" = "Shows")

vids$category_id <- as.factor(vids$category_id)
```

And with that, the next thing we'll need to do is to convert the `publish_time` variable into a date-time class object. Because we're analyzing trending YouTube videos in the US, it makes sense then that we uses a timezone like New York for our analysis:

```
as.POSIXct(head(vids$publish_time),
           format="%Y-%m-%dT%H:%M:%S",
           tz="America/New_York")
```

```
## [1] "2017-11-13 17:13:01 EST" "2017-11-13 07:30:00 EST"
## [3] "2017-11-12 19:05:24 EST" "2017-11-13 11:00:04 EST"
## [5] "2017-11-12 18:01:41 EST" "2017-11-13 19:07:23 EST"
```

Once we've done a sanity check, we can go ahead and apply the conversion to the whole column:

```
vids$publish_time <- as.POSIXct(vids$publish_time,
           format="%Y-%m-%dT%H:%M:%S",
           tz="America/New_York")
```

Once we've done a sanity check, we can go ahead and apply the conversion to the whole column:

```
vids$publish_time <- as.POSIXct(vids$publish_time,
           format="%Y-%m-%dT%H:%M:%S",
           tz="America/New_York")
```

By this point you may be wondering if this could have been made simpler. Allow me to introduce a popular package by the name of `lubridate`. Using `lubridate`, instead of manually specifying the format, you do it "declaratively", like the following:

```
date <-  c("20.30.8", "20.31.8")
date2 <- c("30-8-20", "31-8-20")
date3 <- c("2017-11-13 17:13:01 EST")

# base R
as.Date(date, format="%y.%d.%m")
```

```
## [1] "2020-08-30" "2020-08-31"
```

```
as.Date(date2, format="%d-%m-%y")
```

```
## [1] "2020-08-30" "2020-08-31"
```

```
as.POSIXct(date3, format="%Y-%m-%d %H:%M:%S", tz="UTC")
```

```
## [1] "2017-11-13 17:13:01 UTC"
```

```
# lubridate equivalent:
ydm(date)
```

```
## [1] "2020-08-30" "2020-08-31"
```

```
dmy(date2)
```

```
## [1] "2020-08-30" "2020-08-31"
```

```
ymd_hms(date3, tz="UTC")
```

```
## [1] "2017-11-13 17:13:01 UTC"
```

Observe how simple `lubridate` work with dates and time. In fact, when we use `ymd`, `ymd_hms` or one of its variants, these functions recognize the patterns and will identify the right separators as long as the order of formats is correct. These functions will also parse dates correctly even when the input contain differently formatted dates!

Let's see a few more things that `lubridate` can do. I'll subset the data for the most popular trending video (by number of views) and we'll extract information from the `trending_date` of the most popular trending video:

```r
most <- vids[vids$views == max(vids$views),]
year(most$trending_date)
```

```
## [1] 2017
```

```r
month(most$trending_date)
```

```
## [1] 12
```

```r
day(most$trending_date)
```

```
## [1] 14
```

We will also go ahead and create three new variables for our data frame, storing the hours, period of the day, and the day of the week of each video at the time of publish:

```r
vids$publish_hour <- hour(vids$publish_time)

pw <- function(x){
    if(x < 8){
      x <- "12am to 8am"
    }else if(x >= 8 & x < 16){
      x <- "8am to 3pm"
    }else{
      x <- "3pm to 12am"
    }
}

vids$publish_when <- as.factor(sapply(vids$publish_hour, pw))
vids$publish_wday <- as.factor(weekdays(vids$publish_time))
```

While the `publish_wday` is now a factor, we can also arrange it so our plots will display them in our desired order:

```r
vids$publish_wday <- ordered(vids$publish_wday, levels=c("Monday", "Tuesday", "Wednesday", "Thursday",
```

We'll also go ahead and convert some of the variables into numeric variables as and where appropriate:

```r
vids[,c("views", "likes", "dislikes", "comment_count")] <- lapply(vids[,c("views", "likes", "dislikes",
```

Hopefully up to this point, none of the above data transformation and cleansing process looks too unfamiliar for you! If you do need a refresher, refer to the Programming for Data Science coursebook - we're really applying many of the same ideas to a new dataset, and so you should feel somewhat comfortable up to this point of the course :)

`vids` has 13400 records of trending videos, but there are many videos that were trending for a few days and we really only have a collection of 2,986 unique videos. On a very broad average, each video was trending for ~4.5 days.

Let's create a dataframe, call it `vids.u` that takes only the first observation of each `vids.title` within the data. `match` returns a vector of the positions of matches of its first argument in its second:

```
vids.u <- vids[match(unique(vids$title), vids$title),]
```

We'll also create one more variable `timetotrend` to measure the time it takes for a video to become "trending":

```
vids.u$timetotrend <- vids.u$trending_date - as.Date(vids.u$publish_time)
vids.u$timetotrend <- as.factor(ifelse(vids.u$timetotrend <= 7, vids.u$timetotrend, "8+"))
```

The `ifelse()` command is a very quick way to perform a conditional check, and then do one of two things depending if the conditionals evaluate to TRUE or FALSE. Here's another example to illustrate this concept better:

```
hours <- head(vids$publish_hour)
print(hours)
```

```
## [1] 17  7 19 11 18 19
```

```
ifelse(hours >= 18, "night_shift", "day_shift")
```

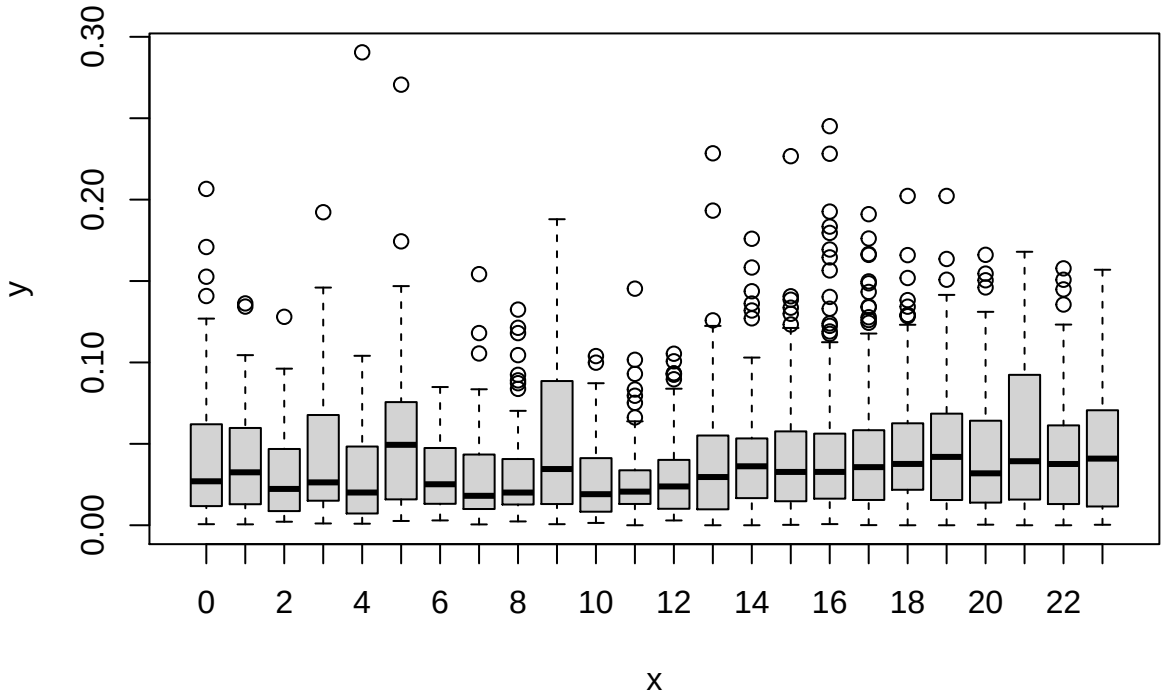
```
## [1] "day_shift"   "day_shift"   "night_shift" "day_shift"   "night_shift"
## [6] "night_shift"
```

With these done, we'll move into the exciting part of this workshop: plotting!

Base Plotting and Statistical Plots

Statistical plots helps us visually inspect our dataset and there are numerous ways to achieve that in R. The simplest of which is through the `plot()` function. In the following code we create two vectors, `x` and `y`, and created a plot:

```
plot(as.factor(vids.u$publish_hour), vids.u$likes/vids.u$views)
```



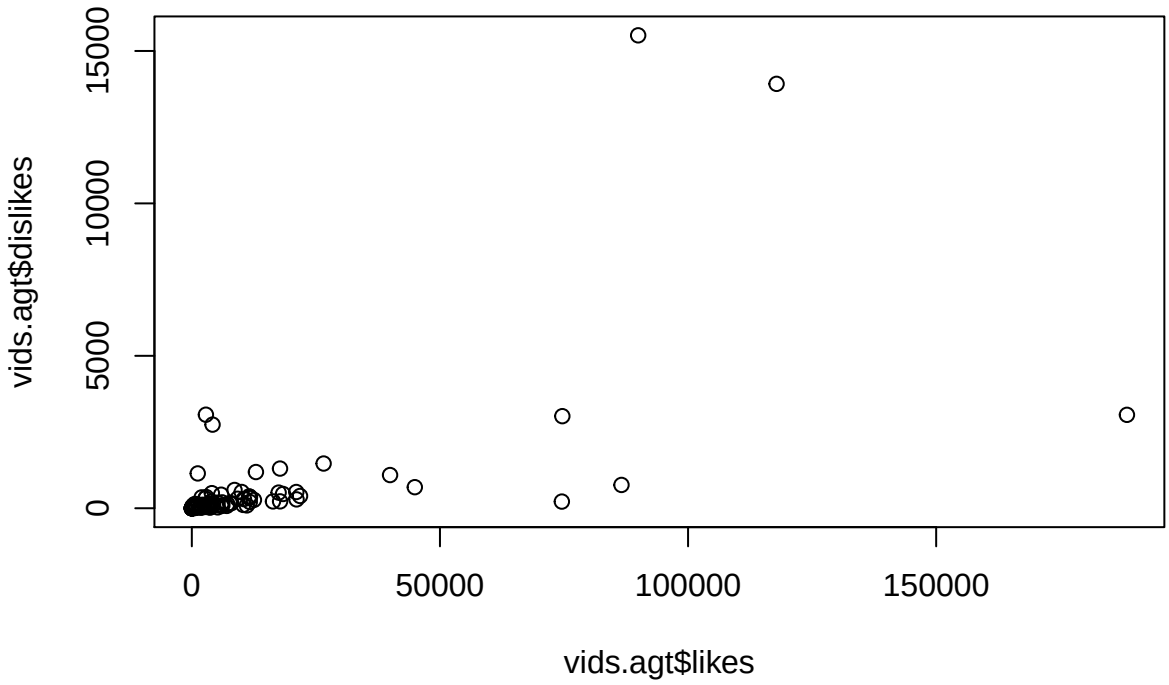
The above gives us a boxplot that compares the "likes ratio" across different period of the day. We want to observe if there is any correlation between the likes-to-view percentage and the period of time when the video was published. As expected, we did not find any obvious patterns, in part because this compares the "likes" a video have on the first day of it being "trending" and the hour when it was published onto YouTube. In a sense, any kind of effect the hour variable can have has been adjusted for (or significantly reduced) by the noise between these two events.

`plot()` knows how to pick sensible defaults based on the input vector it was given. To illustrate this point, I'll subset the data to take only trending videos within the Autos, Gaming and Travel industry (every women's favorite 3 things):

```
vids.agt <- vids.u[vids.u$category_id == "Autos and Vehicles" |
                     vids.u$category_id == "Gaming" |
                     vids.u$category_id == "Travel and Events", ]
```

And notice that as we call `plot` now with two numeric variables, so it creates a scatterplot for us (instead of the boxplot, as seen earlier):

```
plot(vids.agt$likes, vids.agt$dislikes)
```



We'll drop the empty levels from our `category_id` variable, and also create two new variables that measure the likes and dislikes per video view for each observation:

```
vids.agt$category_id <- factor(vids.agt$category_id)
vids.agt$likesp <- vids.agt$likes/vids.agt$views
vids.agt$dislikesp <- vids.agt$dislikes/vids.agt$views
```

Our earlier scatterplot really isn't very informative or even pleasant to look at. The key, as it is with data visualization in general, is to have our plot be effective. A plot that is effective complements how human visual perception works. The scatterplot is ineffective because it could be communicating more with less "visual clutter" at the bottom left.

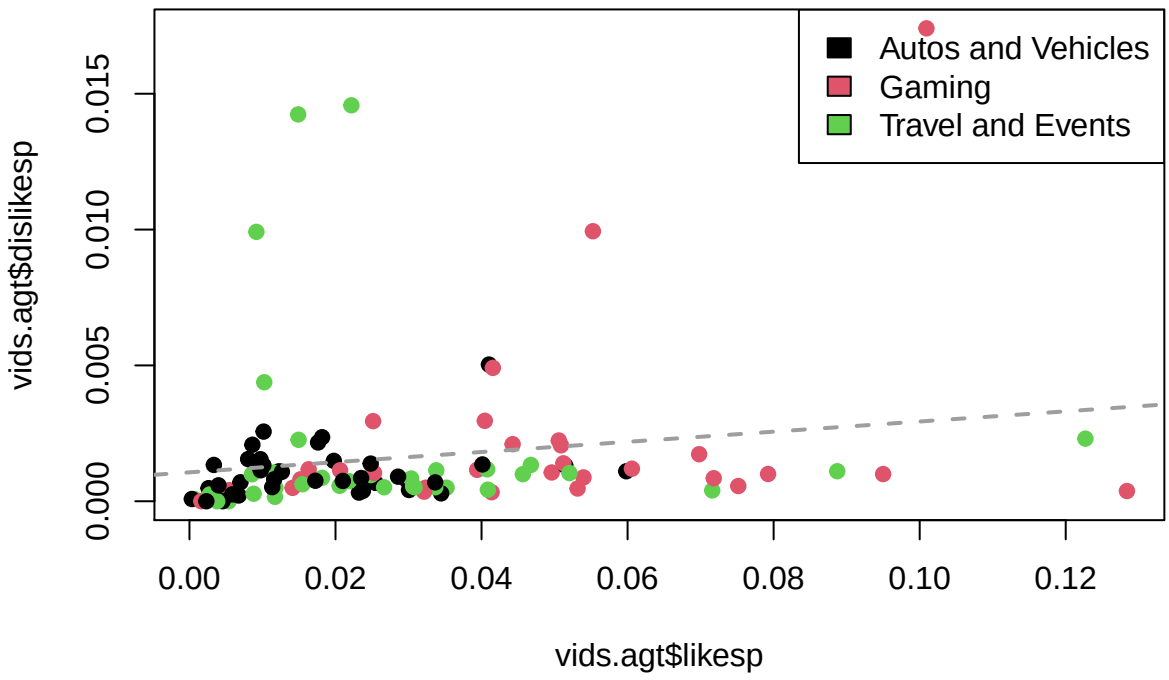
An approach to fix that is by coloring the plots. In the following code chunk, the first line is identical to the code that produces the scatterplot above except for one addition, the `col` (color) parameter. We mapped the color parameter to the category so the points are colored accordingly.

I've also added a dashed line (`lty`) with a width of 2 (`lwd=2`) to show that the correlation between the likes-to-view and dislikes-to-view ratio.

Finally, I added a legend for our plot to show how the colors of our scatterplot points map to each level of our `category_id` variable.

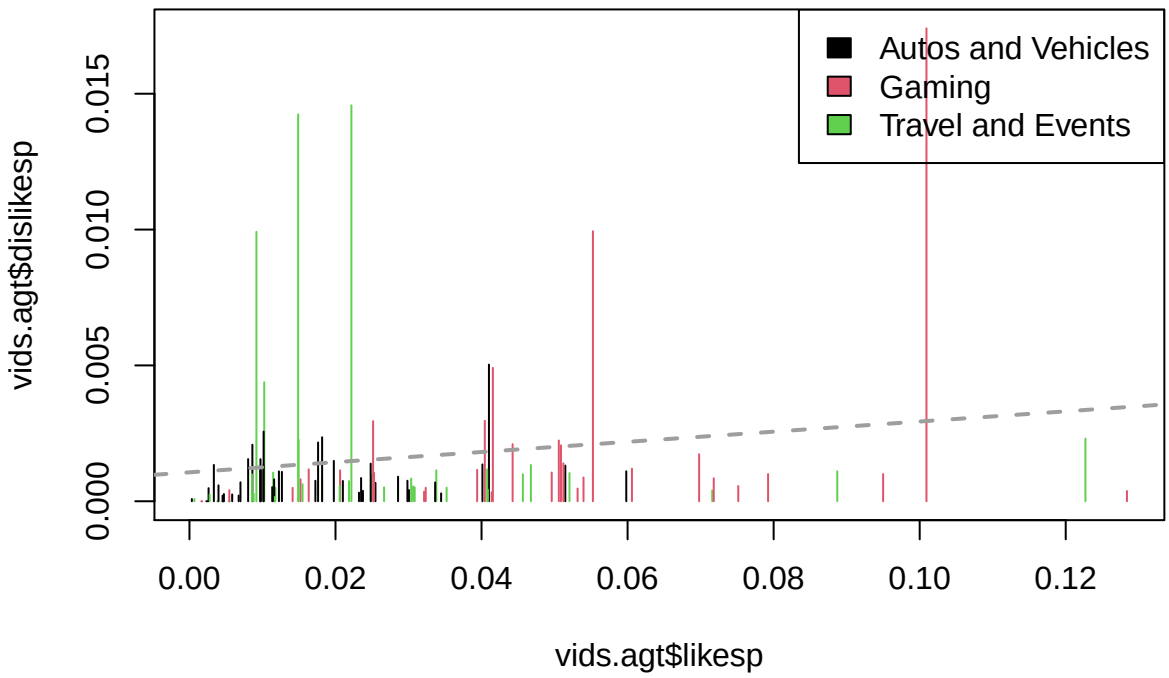
Here's the code:

```
plot(vids.agt$likesp, vids.agt$dislikesp, col=vids.agt$category_id, pch=19)
abline(lm(vids.agt$dislikesp ~ vids.agt$likesp), col=8, lwd=2, lty=2)
legend("topright", legend=levels(vids.agt$category_id), fill=1:3)
```



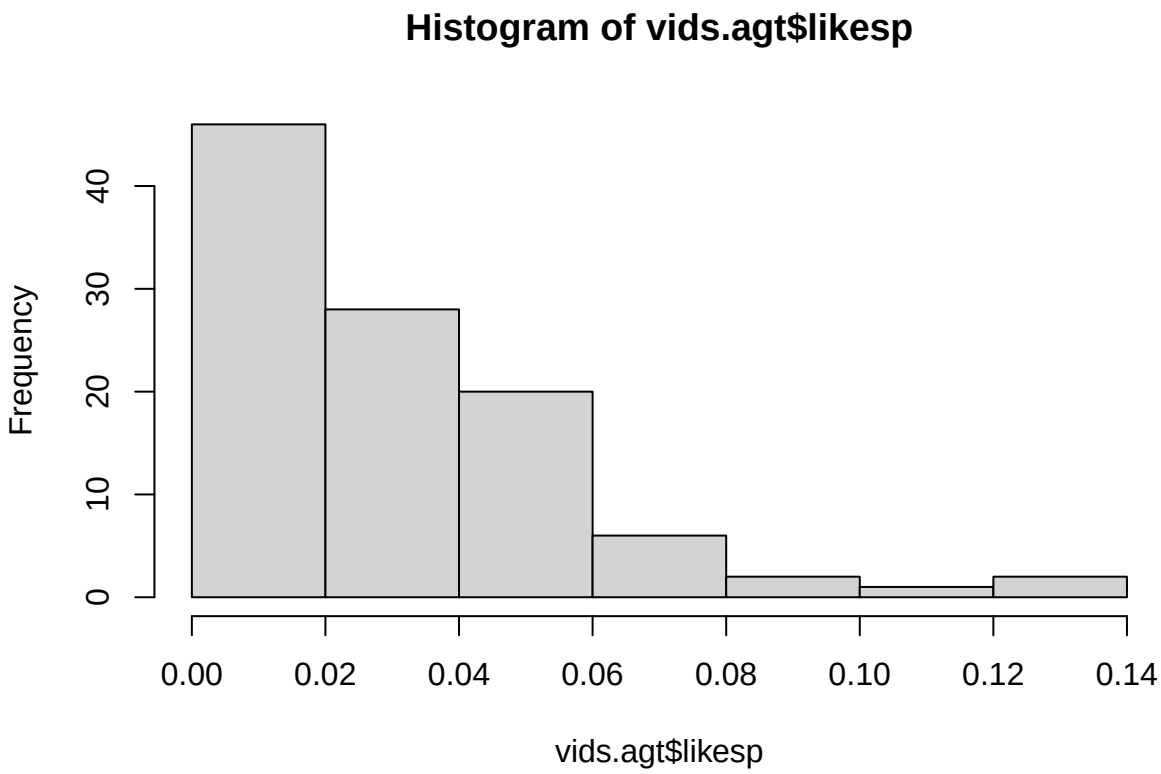
With `plot`, the default for two numerical variables is to plot a scatterplot, but we can override the default parameters with the `type` argument. The following code chunk is identical to the one above, except for the `type="h"` addition:

```
plot(vids.agt$likesp, vids.agt$dislikesp, col=vids.agt$category_id, type="h")
abline(lm(vids.agt$dislikesp ~ vids.agt$likesp), col=8, lwd=2, lty=2)
legend("topright", legend=levels(vids.agt$category_id), fill=1:3)
```



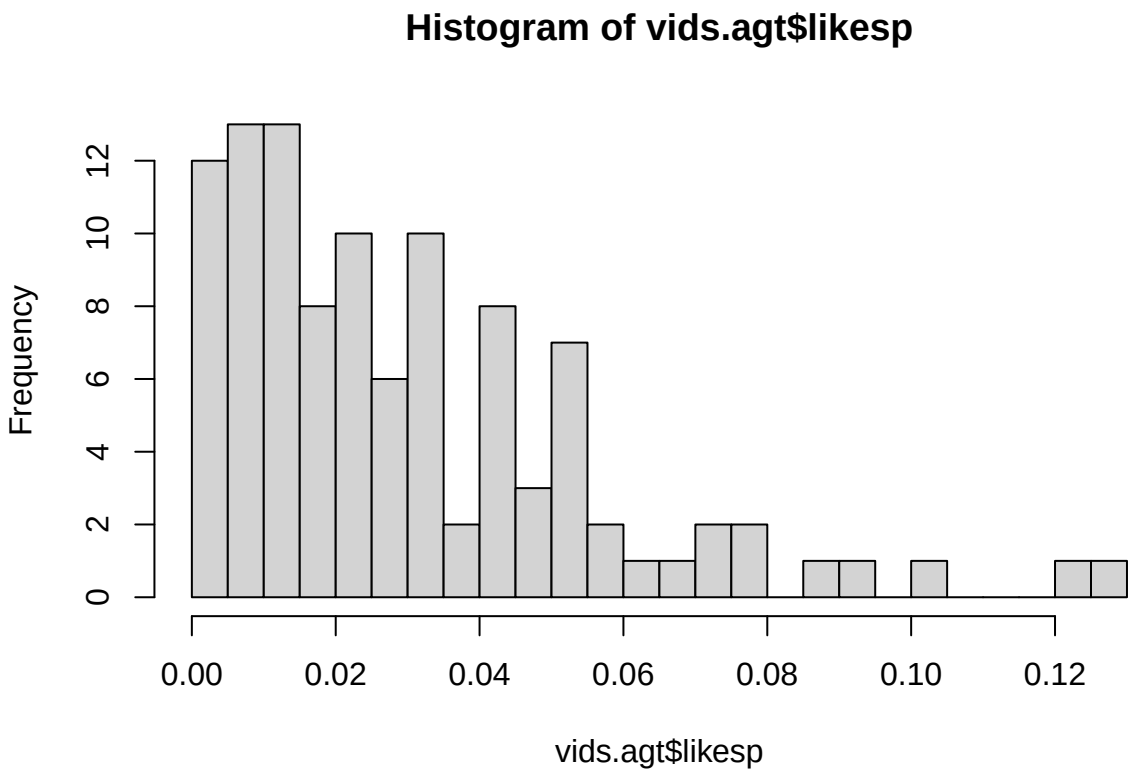
Apart from using `plot()`, we can also create statistical plots using functions such as `hist()`. `hist()` takes a numeric vector and creates a histogram:

```
hist(vids.agt$likesp)
```



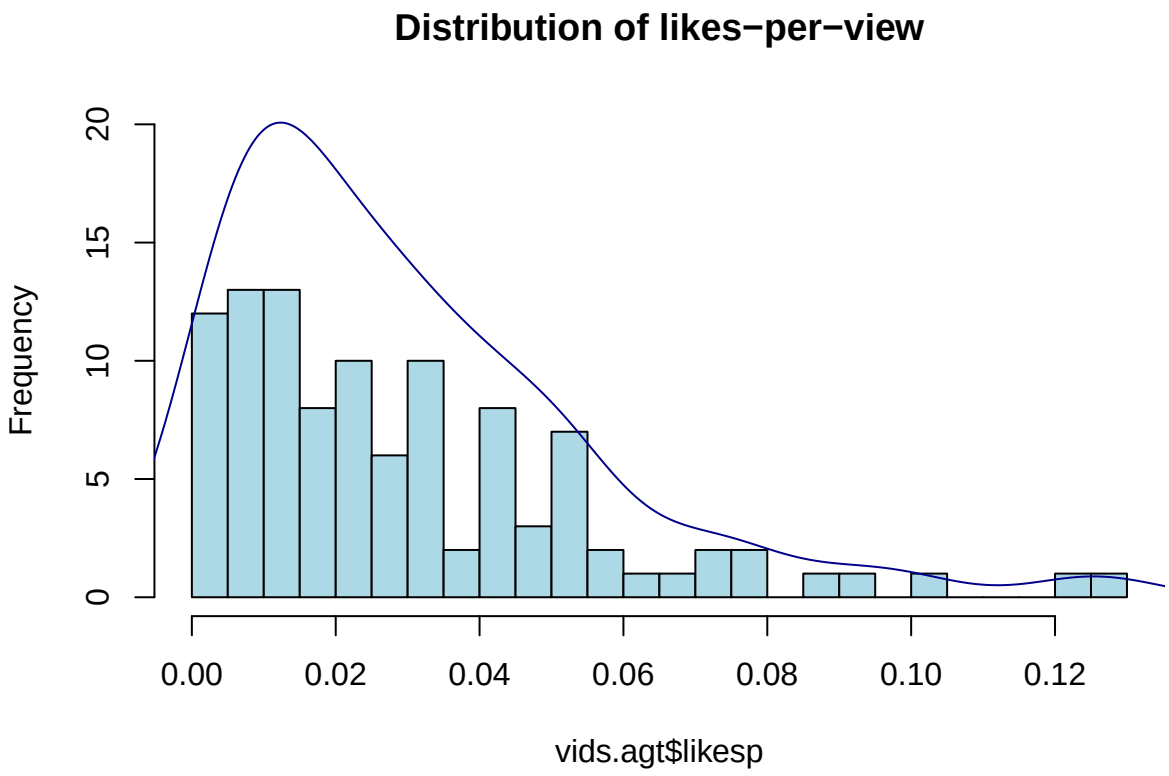
We can additionally use the `breaks` argument to control the number of bins if we were not satisfied with the default values:

```
hist(vids.agt$likesp, breaks=20)
```



Just like how we can add `abline` onto our plot, we can add graphical elements like `lines` onto this histogram too. In fact, let's do that and also use the `main` argument to give our plot a new main title:

```
hist(vids.agt$likesp,
     breaks=20,
     ylim=c(0, 20),
     col="lightblue",
     main="Distribution of likes-per-view")
lines(density(vids.agt$likesp), col="darkblue")
```



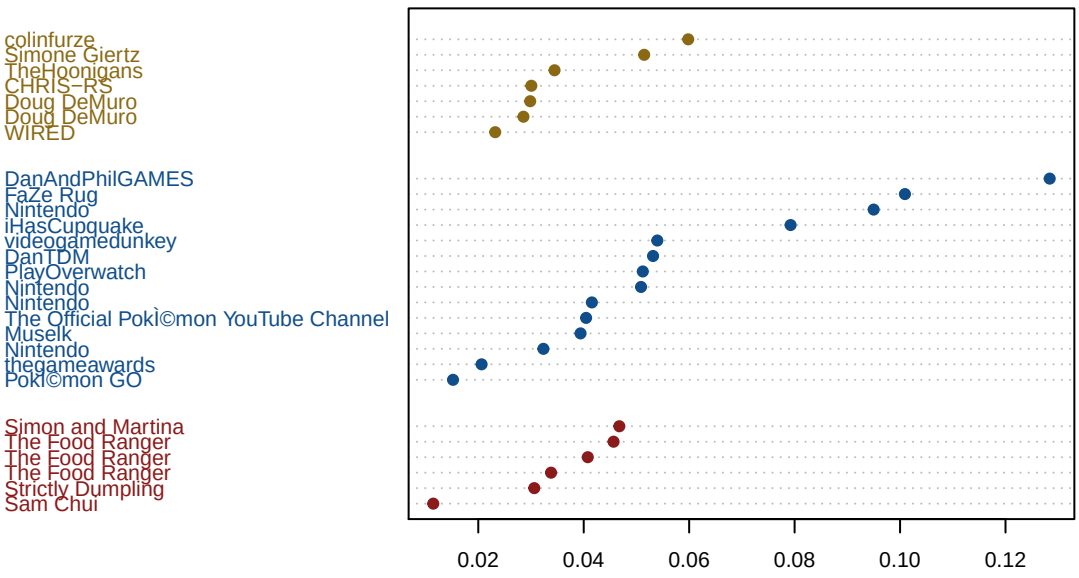
While base plot can be very simple to use, they can be effective too. In fact, with the use of proper coloring, annotation and a little care on the aesthetic touches, you can communicate a lot in a graph using just R's built-in plotting system.

In the following code chunk I'm subsetting from `vids.agt` only trending videos that have more than 10,000 likes and order it by the likes-to-view variable. I added a new variable, `col` to this new dataframe to be used in my following plot:

```
vids.ags <- vids.agt[vids.agt$likes > 10000, ]
vids.ags <- vids.ags[order(vids.ags$likesp), ]

# create color specifications for our dotchart
vids.ags$col[vids.ags$category_id == "Autos and Vehicles"] <- "goldenrod4"
vids.ags$col[vids.ags$category_id == "Gaming"] <- "dodgerblue4"
vids.ags$col[vids.ags$category_id == "Travel and Events"] <- "firebrick4"
```

We're going to create a dot chart (or a Cleveland's Dot Plot) by graphing the likes to view ratio of each trending video in the `vids.ags` dataframe, map `channel_title` to the labels and group these labels by `category_id`.

```
dotchart(vids.ags$likesp, labels=vids.ags$channel_title, cex=.7, pch=19, groups=vids.ags$category_id, c
```



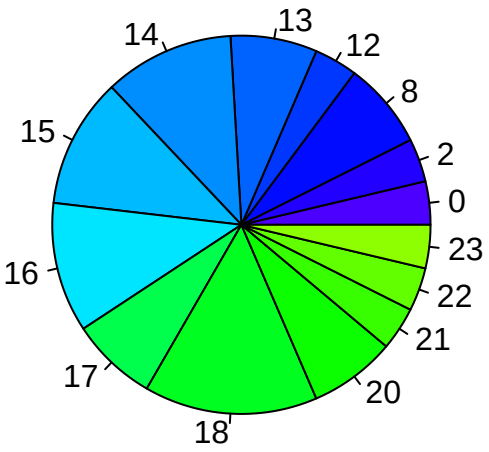
With this we see that between groups, the likes to view proportion (we'll call it "likeability" from now on) of Autos and Vehicles are rather similar with Travel and Events.However, for gaming videos we do see a larger variance in that the top video by likeability is close to 0.13, more than 6 times difference to that of other trending videos in this category. We also observe the rough mean likeability within groups, as well as between them. We would expect Travel and Events videos to have ~4 likes per 100 views, and Gaming videos to have more than that due to the positive skew we observe from above.

Let's talk about another kind of plot, one that most statisticians find cringeworthy for it's undeserved popularity and prevalence in the workplace. Yes, it is the pie chart. In R's official documentation, the pie chart is criticized as being "a very bad way of displaying information [because] the eye is good at judging linear measures and bad at judging relative areas". Almost any data that can be represented in a pie chart can be illustrated with a bar chart or dot chart[2].

If you insist on creating one, here's the code (I've added some colors to make it easier to get a grasp of the measures):

```
pie(table(vids.ags$publish_hour), labels=names(table(vids.ags$publish_hour)), col=topo.colors(24))
```

[2]Full Note on pie charts from the official R Documentation:
"Pie charts are a very bad way of displaying information. The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a preferable way of displaying this type of data.



Grammar of Graphics in R

The motivation of ggplot2

`ggplot2` is created by Hadley Wickham in 2005 as an implementation of Lelad Wilkinson's Grammar of Graphics. The idea with Grammar of Graphics is to create a formal model for data visualization, by breaking down graphics into components that could be systematically added or subtracted by the end user.

With `ggplot2`, plots may be created using `qplot()` where arguments and defaults are handled similarly to the base plotting system, or through `ggplot()` where user can add or alter plot components layer-by-layer with a high level of modularity.

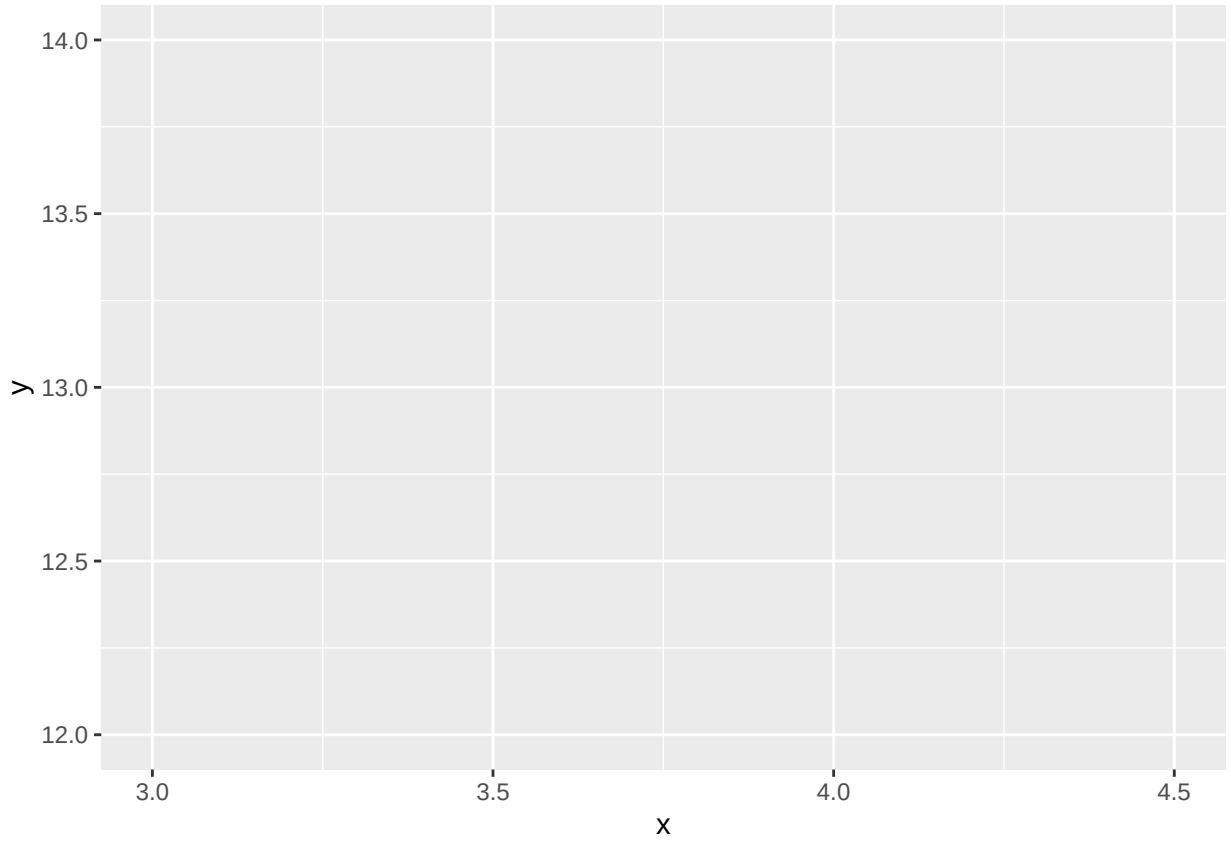
The last point is especially important because it allows the data scientists to work with plots in a system that breaks up these different tasks. Instead of a huge, conceptually flat list of parameters to control every aspect of the plot's final outcome, this system makes plotting a series of distinct task, each focused on one aspect of the plot's final output.

Let us take a look at a simple example, drawing inspiration from the Earthquake incident that happened in the south of Jakarta this week (as of this writing). I've created a dataframe called `gempa`:

```
gempa <- data.frame(
  x=c(3.5,3,4,4.5,4.1),
  y=c(12,14,12.4,12.5,14),
  size=c(14,4,4,6,12)
)
```

And now I'll create a ggplot object using `ggplot()`. Because of the range of my values, this plot will use that and create a plot with these values on each scales (scales, by the way, can be thought of as just the two axis right now). Note that we're just creating a blank plot with no geometry elements (lines, points, etc) on it yet. We save this object as `g`:

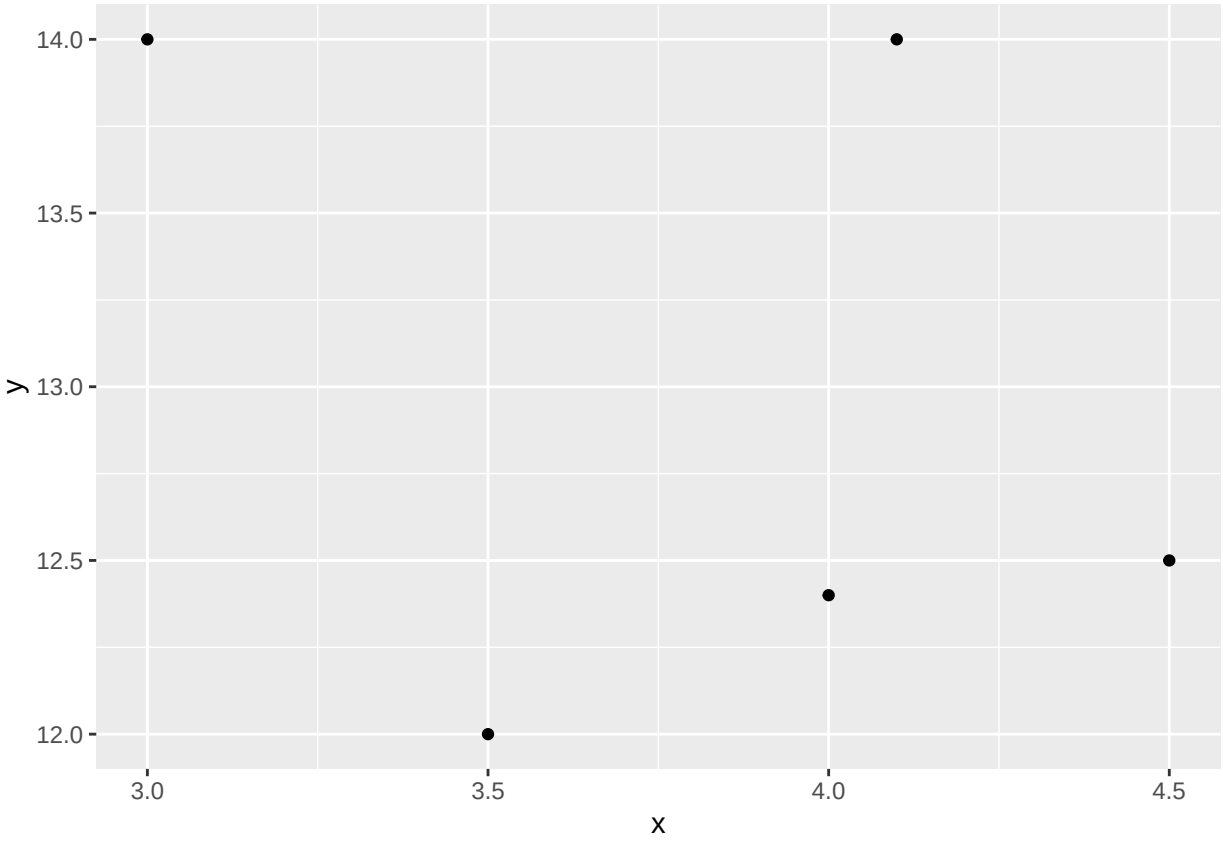
```
g <- ggplot(gempa, aes(x = x, y = y))
g
```



Notice how `ggplot()` takes two arguments: - The data - The `aes` which allow us to specify our mapping of the x and y variables so they are used accordingly by ggplot

Once we created our `ggplot` object (we named it `g`), we can now add a layer onto it using `geom_`. `geom` is ggplot's way of handling geometry, i.e. how the data are represented on the plot. To illustrate this idea, let's add a geom_point and then print the resulting object:

```
g + geom_point()
```



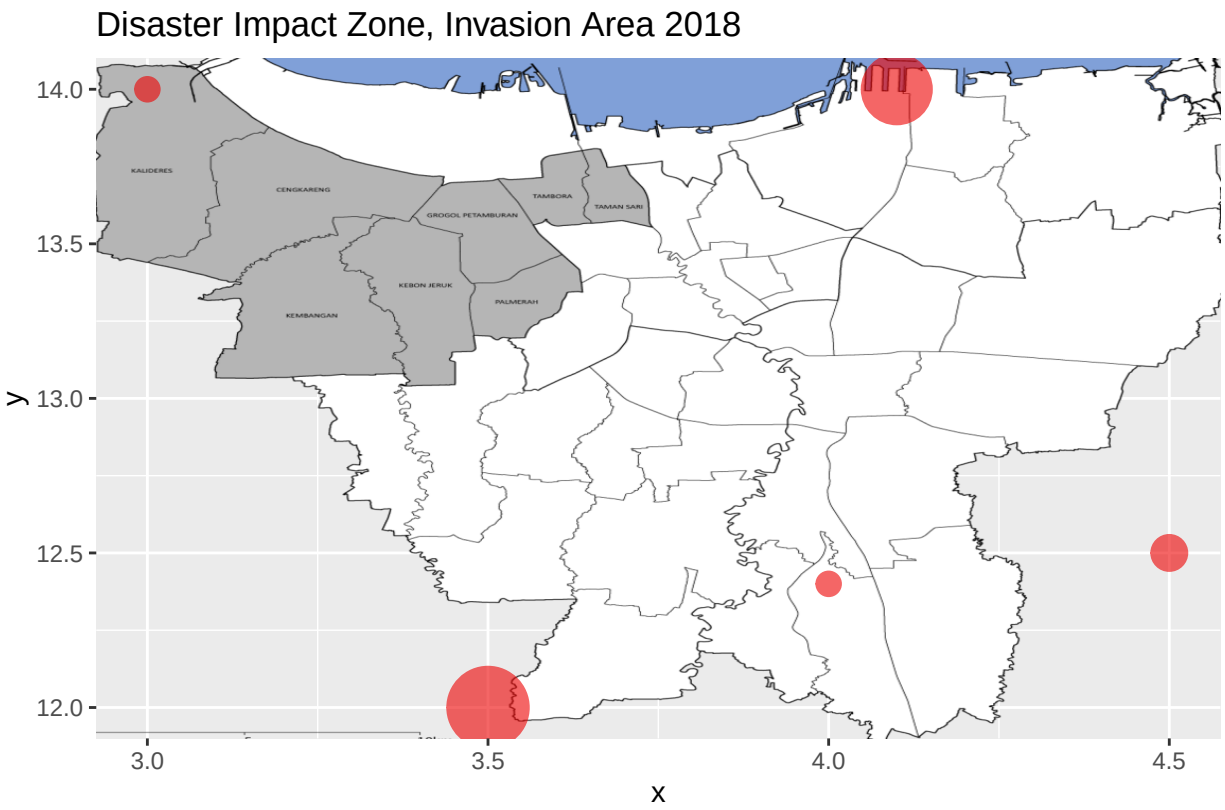
A recap of what we've done so far:

- Creating our ggplot graphics object through `ggplot()`
- We specify 2 arguments in our call to ggplot; It's helpful to note that any argument we pass into `ggplot()` will be used as global options for the plot, i.e. they apply to all layers we add onto that graphics object
- For the second argument we use the `aes()` function, allowing us to map variables from our `gempa` data to aesthetic properties of the plot (in our case we map them to the x and y axis)
- We tell ggplot how we want the graphic objects to be represented by adding (through the "+" operator) our geom layer. Since we added `geom_point`, this is equivalent to adding a layer of scatterplot to represent our x and y variables

As we familiarize ourselves with this system, we will learn to use other functions to obtain a more precise control over the construction of our plot. This could be natively `ggplot` constructs such as scales, legends, geoms and thematic elements or this could be additional constructs that work with `ggplot` through the use of third-party packages. In the following example, we're adding `background_image` to our original plot (`g`) before adding `geom_point` on top of the background image layer. Finally, we add the labels for our title and caption using the `labs` function:

```
library(png)
jak <- png::readPNG('assets/jakarta.png')
g +
  background_image(jak)+
```

```
  geom_point(size=gempa$size, alpha = 0.6, col="red2")+
  labs(title="Disaster Impact Zone, Invasion Area 2018", caption="source: Jakarta Distaster Relief")
```



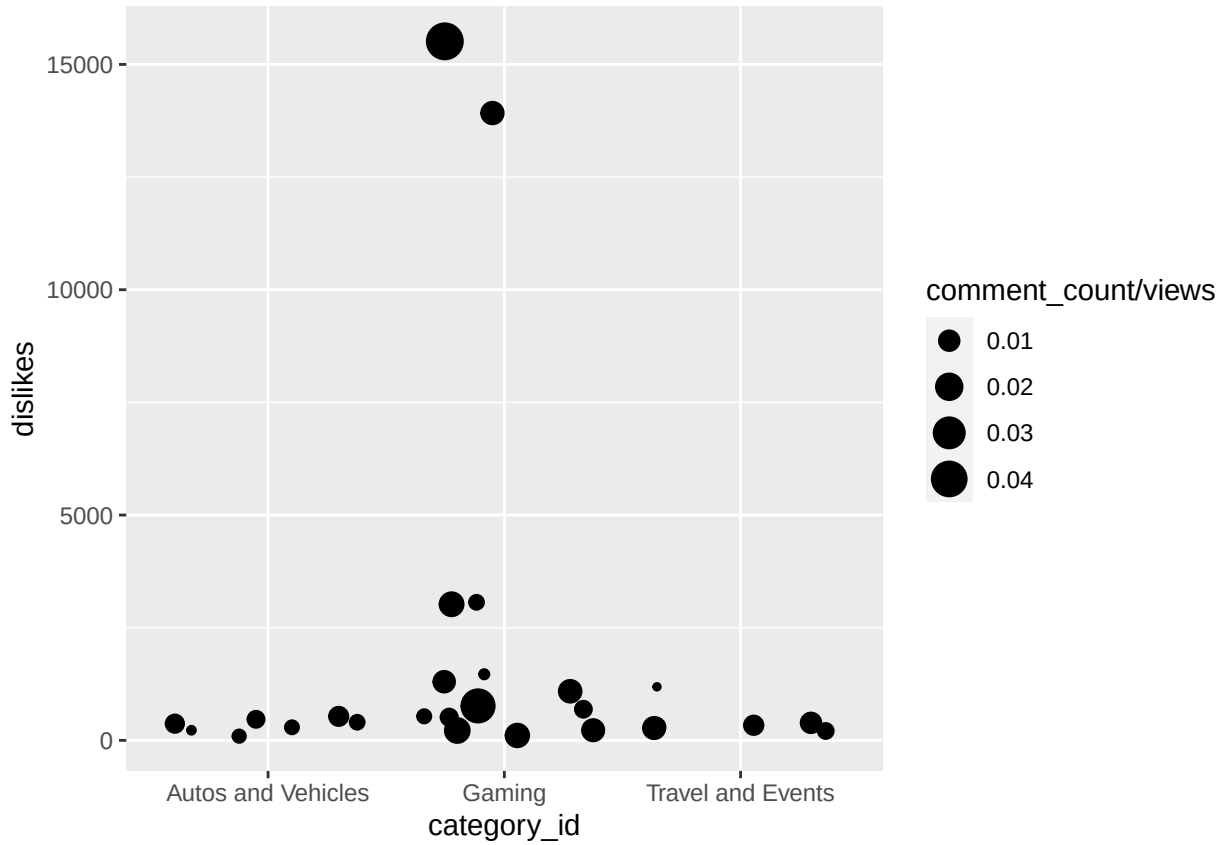
Because of this design philosophy in ggplot, it presents a learning curve that is beginner-friendly and mostly logical. I said beginner-friendly, because as we will see later, all we need to do is to master the starting steps first and not worry about polishing. And with starting steps, this means:
- 1: `ggplot()` with `data` and aesthetics mapping (`aes`) - 2: Add to (1) a single `geom` layer

The State of Trending Videos

Hands-on ggplot: Simple Exploratory Analysis

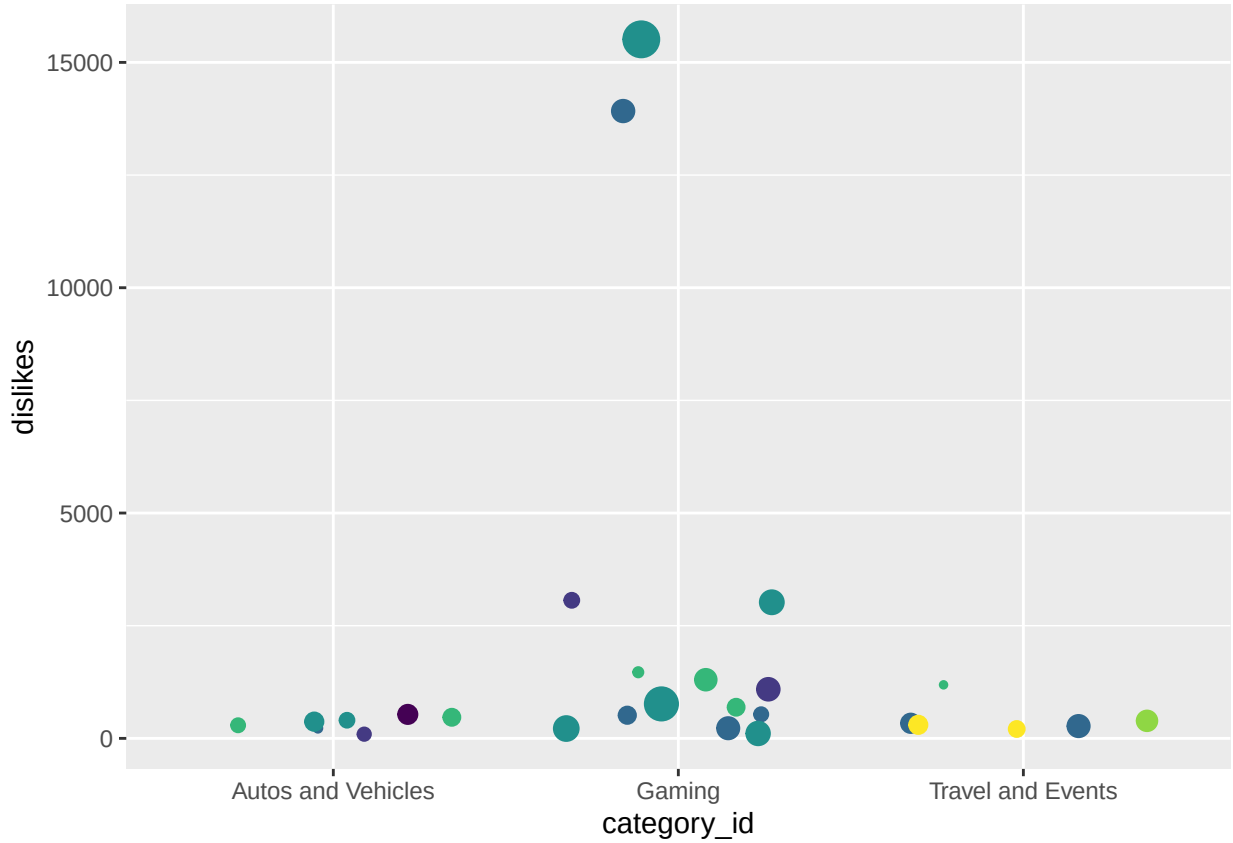
Let's apply what we've learned above to create a simple plot. We will use `vids.ags` as the data, and map our x, y, and size aesthetics to the `category_id`, `dislikes` and comment-per-view respectively.

```
ggplot(data = vids.ags, aes(x=category_id, y=dislikes, size=comment_count/views))+
  geom_jitter()
```



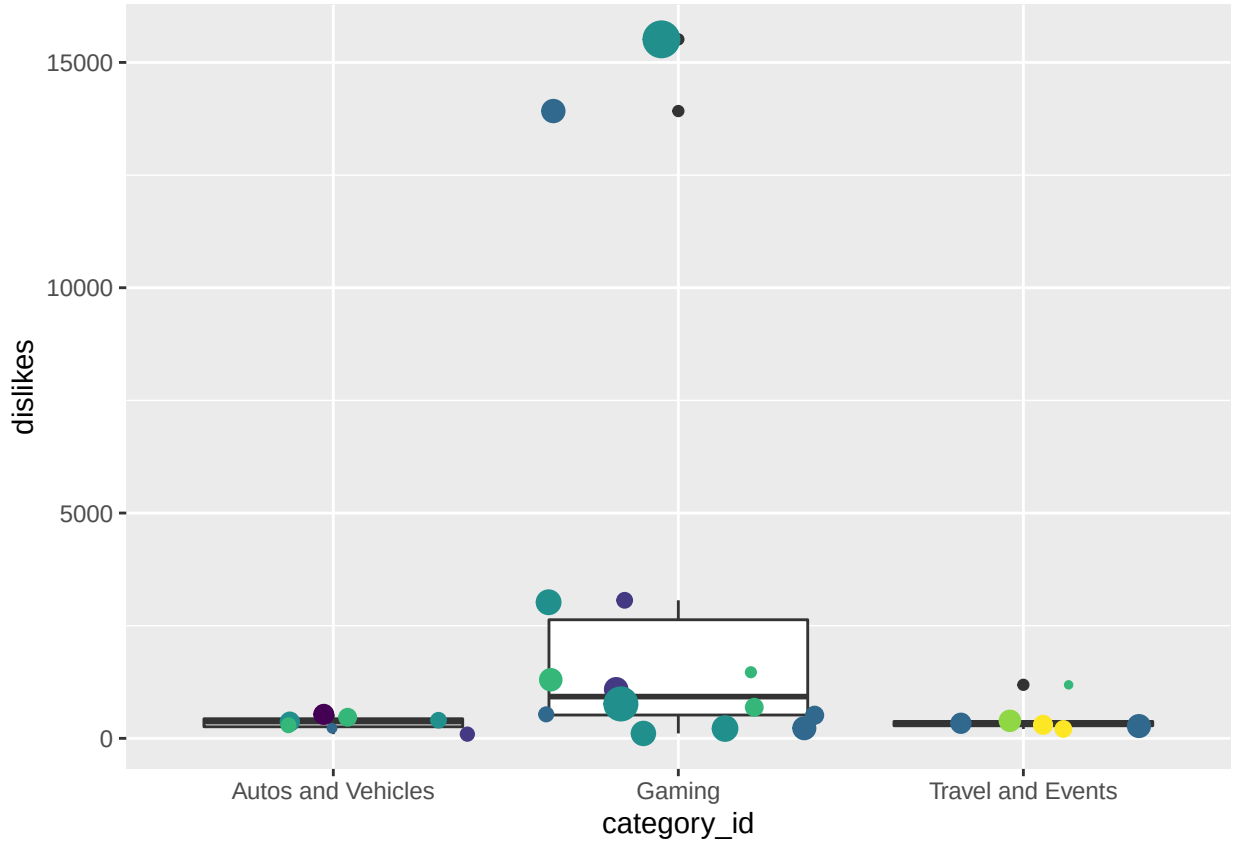
Mostly, I'm copy-and-pasting the code from earlier chunks and then adding ggplot2 elements onto the plot layer-by-layer. Here I mapped the color of my jitter points to the day of week for each of these videos. I also added a `theme` component to specify the position of my legend:

```
ggplot(data = vids.ags, aes(x=category_id, y=dislikes, size=comment_count/views))+
  geom_jitter(aes(col=publish_wday))+
  theme(legend.position = "none")
```



I hope you're getting the pattern now, but just to drive home the point, let's copy the code above and add one more layer: `geom_boxplot()`. I want the jitter points to be above the boxplot, so I'll add the elements in that order such that the last item will be on the top-most:

```
ggplot(data = vids.ags, aes(x=category_id, y=dislikes, size=comment_count/views))+
  geom_boxplot()+
  geom_jitter(aes(col=publish_wday))+
  theme(legend.position = "none")
```



Dive Deeper: Give our plot a main title and caption:

Recall that we can use `labs()` to change the labels on our plot and we did just that earlier when we gave our plot a main title and caption. Copy (or re-write) the code above, and gave them an appropriate title and caption. Your code should look something like this:

```
### Copy and Paste or Finish the following pseudo-code

## psuedo-code
## ggplot() +
##   geom_boxplot() +
##   geom_jitter()+
##   theme() +
##   labs(title="Dislikes to View Comparison", subtitle="Youtube Trending Data, 2018", x="Video Categor

### End of Exercise
```

Hopefully that was a fun introduction to ggplot!

Now let's start thinking about a more useful application of plotting. Imagine you're working with the insights and business intelligence team at a media firm, and were asked to produce a report that illustrates the most prolific producers of trending videos in recent weeks.

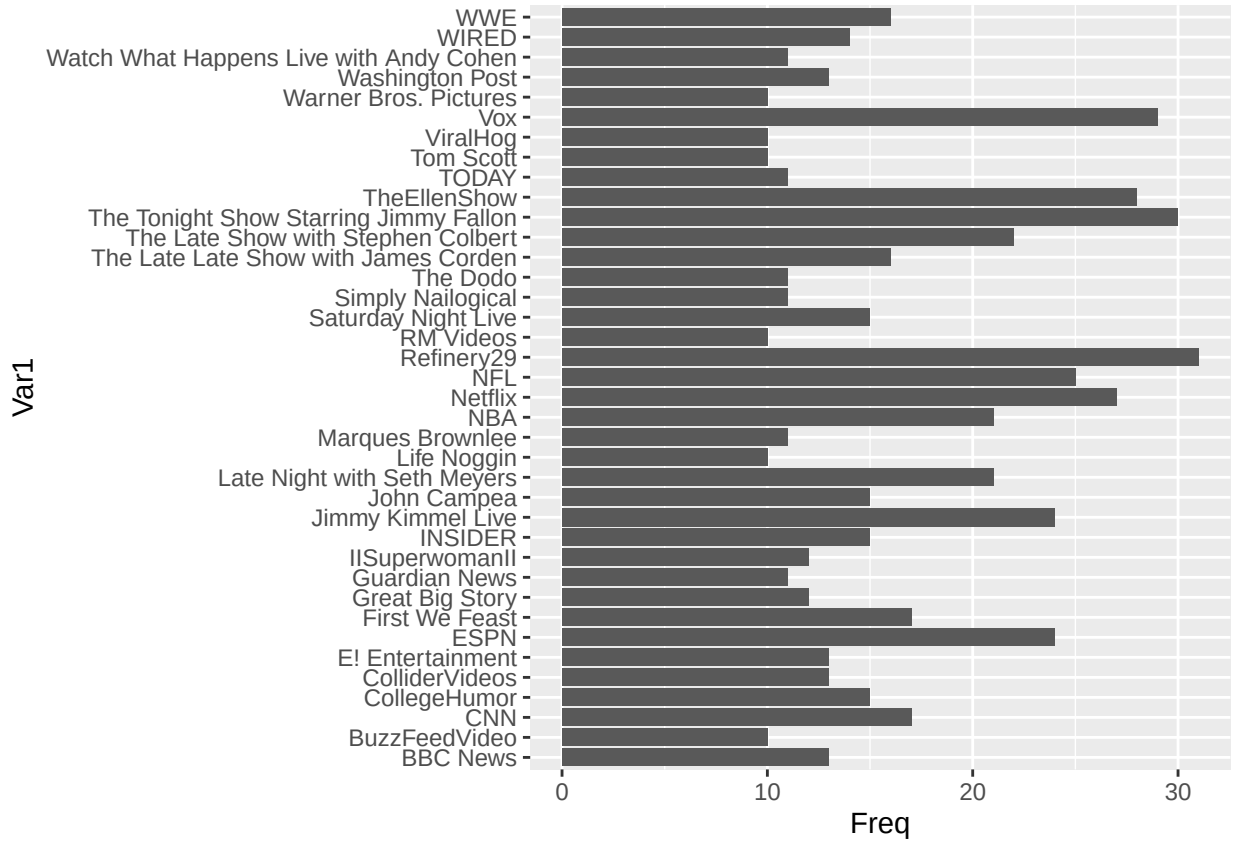
Since we're concerned about the quantity of videos (talking about being *prolific!*) we will create another subset of the full dataframe, but take only the channels that have at least 10 videos being trending!

```
temp1 <- as.data.frame(table(vids.u$channel_title))
temp1 <- temp1[temp1$Freq >= 10,]
temp1 <- temp1[order(temp1$Freq, decreasing = T), ]
head(temp1)
```

```
##                                       Var1 Freq
## 1019                           Refinery29   31
## 1226 The Tonight Show Starring Jimmy Fallon   30
## 1354                                   Vox   29
## 1241                          TheEllenShow   28
## 881                                Netflix   27
## 889                                    NFL   25
```

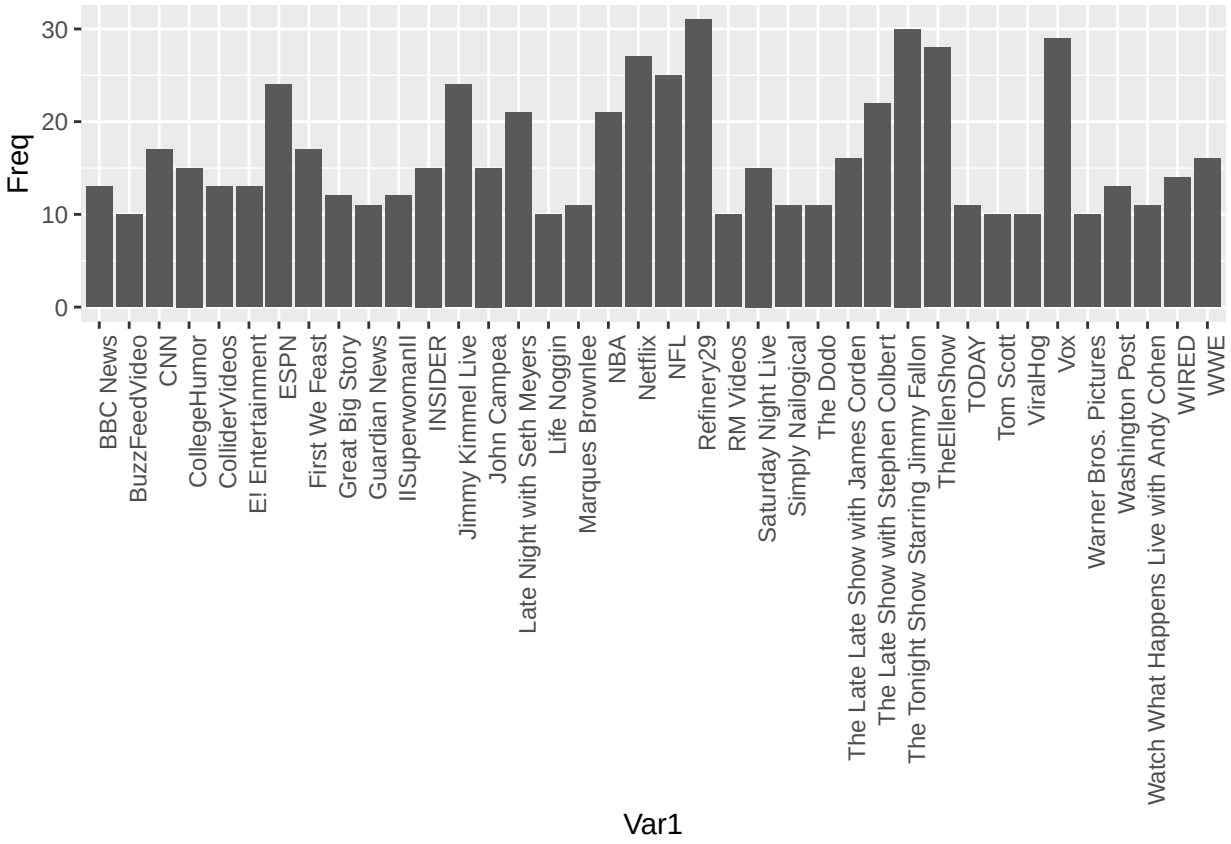
`temp1` is a dataframe with two variables, and contain a list of channels that have at least 10 videos being trending in the observation period. Let's create a column chart (`geom_col`). The category names can be displayed horizontally by placing it on the *x-axis* while the frequency is placed on the *y-axis*. That makes it easier for the user to read and identify the videos that are more prolific than others in producing trending videos:

```
ggplot(temp1, aes(x=Freq, y=Var1))+
  geom_col()
```



If flipping the coordinates (swapping x and y axis) isn't an option, then another approach is to rotate the axis-text for x by 90 degree. This is done with the following code:

```
ggplot(temp1, aes(x=Var1, y=Freq))+
  geom_col()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



To maximize the time you spent writing code, let's hop into another Dive Deeper challenger. This time we'll create a dataframe with two variables: first is the `category_id` and second being the mean (average) of dislikes by each of these category. Using what we've learned in Programming for Data Science we create this dataframe using `aggregate()`

Here's the code:

```
temp2 <- aggregate.data.frame(vids.u$dislikes, by=list(vids.u$category_id), mean)
head(temp2)
```

```
##              Group.1         x
## 1 Autos and Vehicles  155.1220
## 2             Comedy  863.9414
## 3          Education  363.7477
## 4      Entertainment 2303.7745
## 5 Film and Animation  572.9079
## 6             Gaming 1473.9667
```

Dive Deeper: Can you use `geom_col` to create a column chart that plots the amount of average likes by category? Refer to earlier exercises if you need to. Add a `labs()` component to label the x- and y-axis appropriately as a bonus.

```
# Write your code here
```

Now what is important here is that the above did not account for the views each of these category has, and so before jumping into any conclusion it's paramount we think with clarity and be aware of the "biases" our plot or statistics may show. Perhaps a better idea is to have our dislikes "adjusted for" the views, so it is more representative of the true events!

Just to freshen things up, we'll shift the focus from **video categories** to **video producers** for this next exercise. We're now creating a subset of the dataframe that measures the dislikes-to-views and comment count, but group it on a channel / producer level instead of a categorical level. We'll make sure to take only observations with a non-zero value for the comment count:

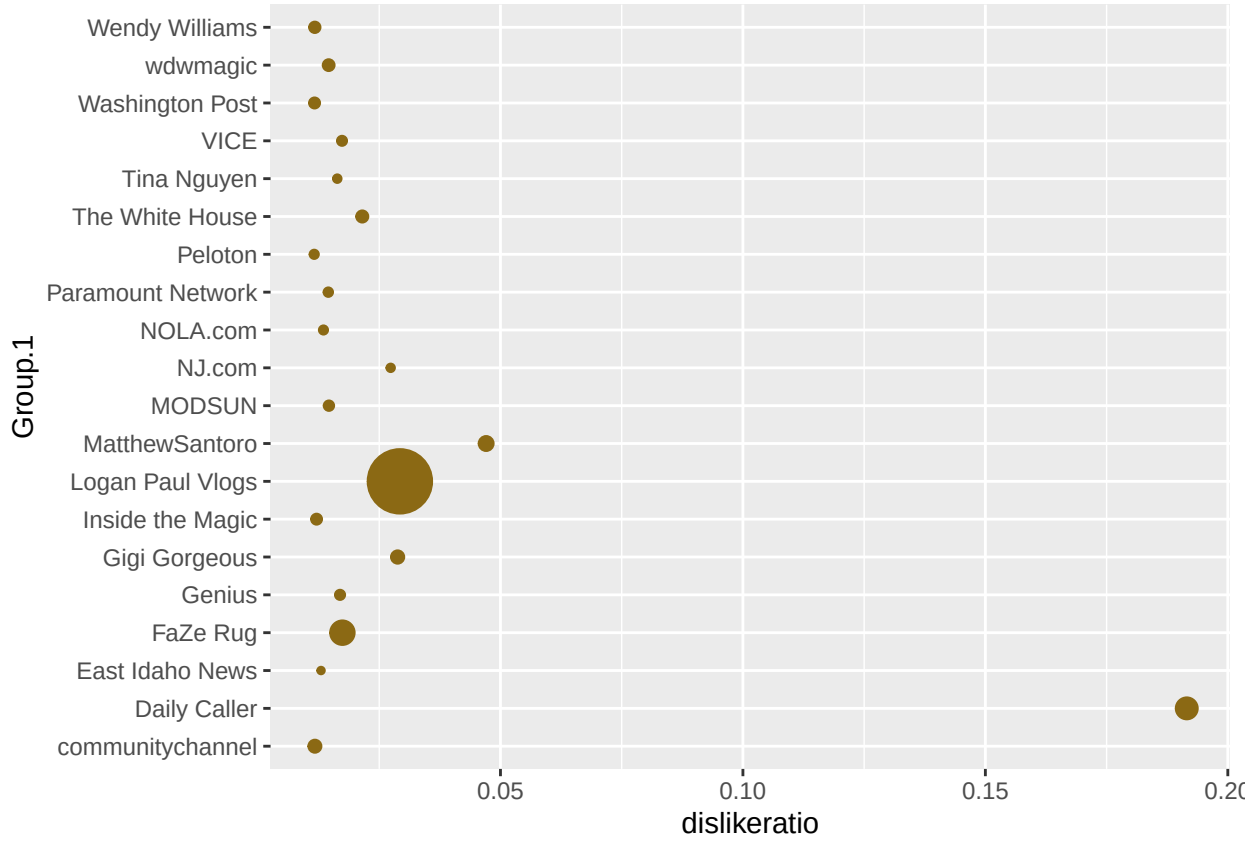
```
temp3 <- aggregate.data.frame(list(dislikeratio = vids.u$dislikes/vids.u$views, comment = vids.u$commen
temp3 <- temp3[order(temp3$dislikeratio, decreasing = T), ]
temp3 <- temp3[temp3$comment != 0, ]
head(temp3)
```

```
##               Group.1 dislikeratio  comment
## 292      Daily Caller   0.19153148  28013.0
## 777    MatthewSantoro   0.04705022   7484.0
## 723  Logan Paul Vlogs   0.02926234 434266.0
## 461     Gigi Gorgeous   0.02879144   4477.0
## 904             NJ.com   0.02736082    147.0
## 1234  The White House   0.02150297   2870.5
```

With the newly created dataframe `temp3`, let's try and create a plot similar to the dot plot we see in an earlier exercise. For now, don't worry about coloring the points or labels and just use a fixed color. However, have the size of each point correspond to the comment count of the videos.

The solution code is here:

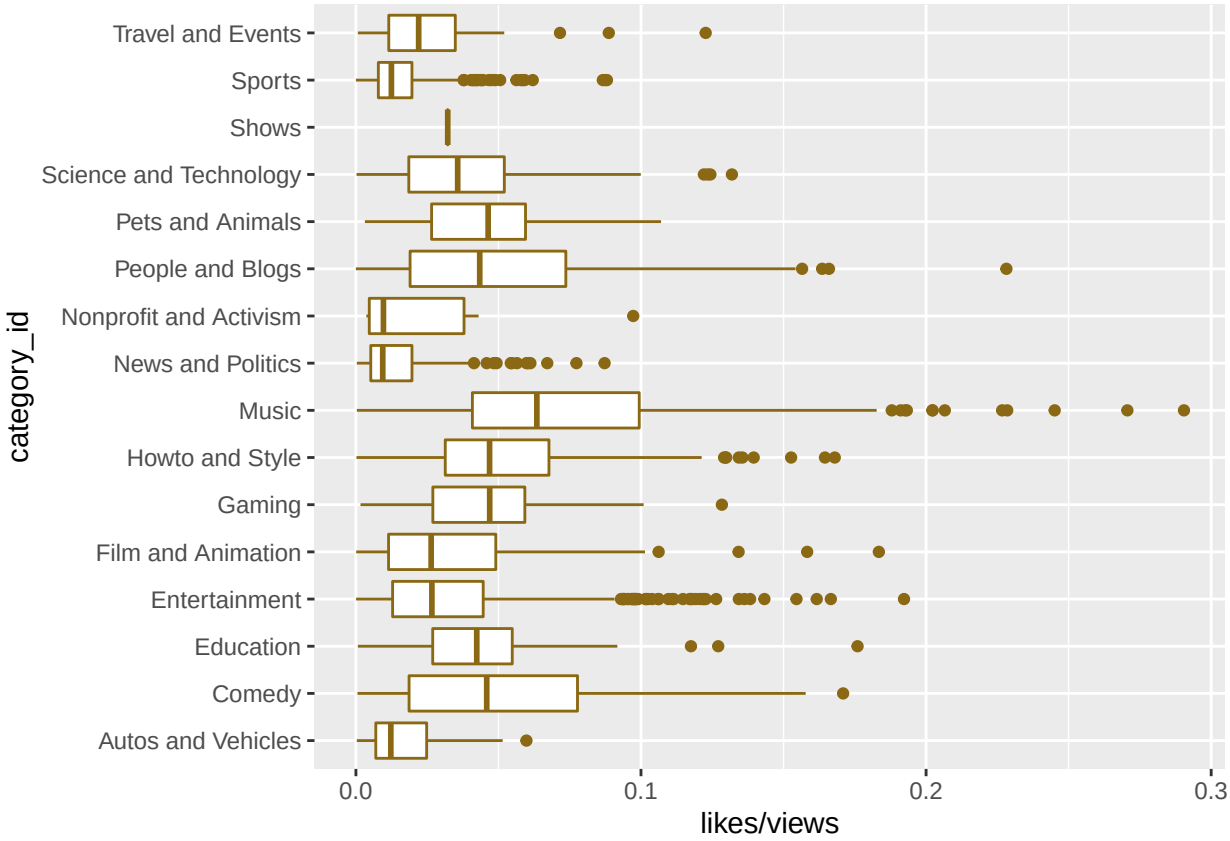
```
ggplot(temp3[1:20,], aes(x=dislikeratio, y=Group.1))+
  geom_point(aes(size=comment), color="goldenrod4",show.legend = F)+
  scale_size(range=c(1,11))
```



Notice for example that among the top 20 most disliked trending videos, **Daily Channel** is credited with the most disastrous dislike ratio - close to 18 dislikes per 100 view! Note also that channels such as **Logan Paul Vlogs** and to a smaller extent **FaZe Rug** have videos that garnered a fair bit of presumably hate / unkind comments, more so than the rest of the crop. It is important to be reminded that here we're looking at 20 of the most disliked videos in our entire dataset!

We can swap the axis to any other geoms as well to get a better visualization. Such flexbility clearly has much to offer when the data we want to visualize is relatively large, making the right arrangement of the plot's element especially important. Let's see an example below:

```
ggplot(vids.u[vids.u$ratings_disabled == F, ], aes(x=likes/views, y=category_id))+
  geom_boxplot(show.legend = F, color="goldenrod4")
```

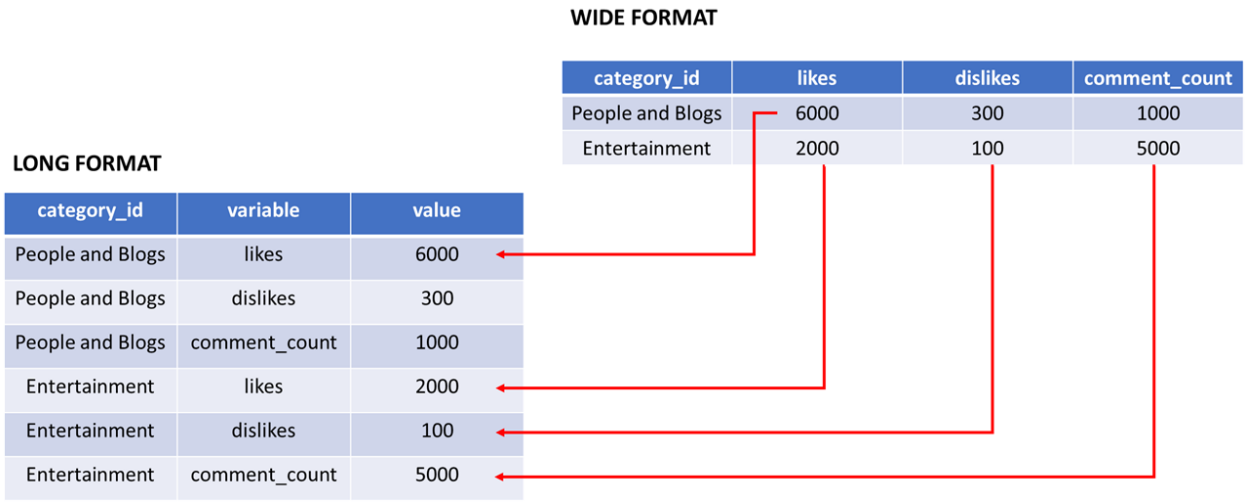


A potentially noisy plot, with some clever arrangement, can be made simple and effortless with ggplot. Spend a couple of minutes before hopping into the next section to understand, and practice, what you've learned so far. Hopefully you've seen enough to be convinced of the merits and advantages of this plotting system.

Hands-on ggplot: Multivariate Plots

To take things up a notch, let's see how we can apply what we've learned to create multivariate plots, or plots designed to reveal the relationship among multiple variables, which in turn help us examine the underlying patterns between pairs of variables.

To facilitate our tasks, I'm introducing a `package` called `tidyr`, which allow us to reshape a data frame between "wide" (measurement spread across horizontally) and "long" (measurement collected vertically). The principle of wide and long format can be illustrated with the following figures.



In the following code, I'm reshaping the `vids.u` dataframe from a wide one to a long one, with `category_id` being the ID and measurements being `likes`, `dislikes` and `comment_count`:

```
library(tidyr)
```

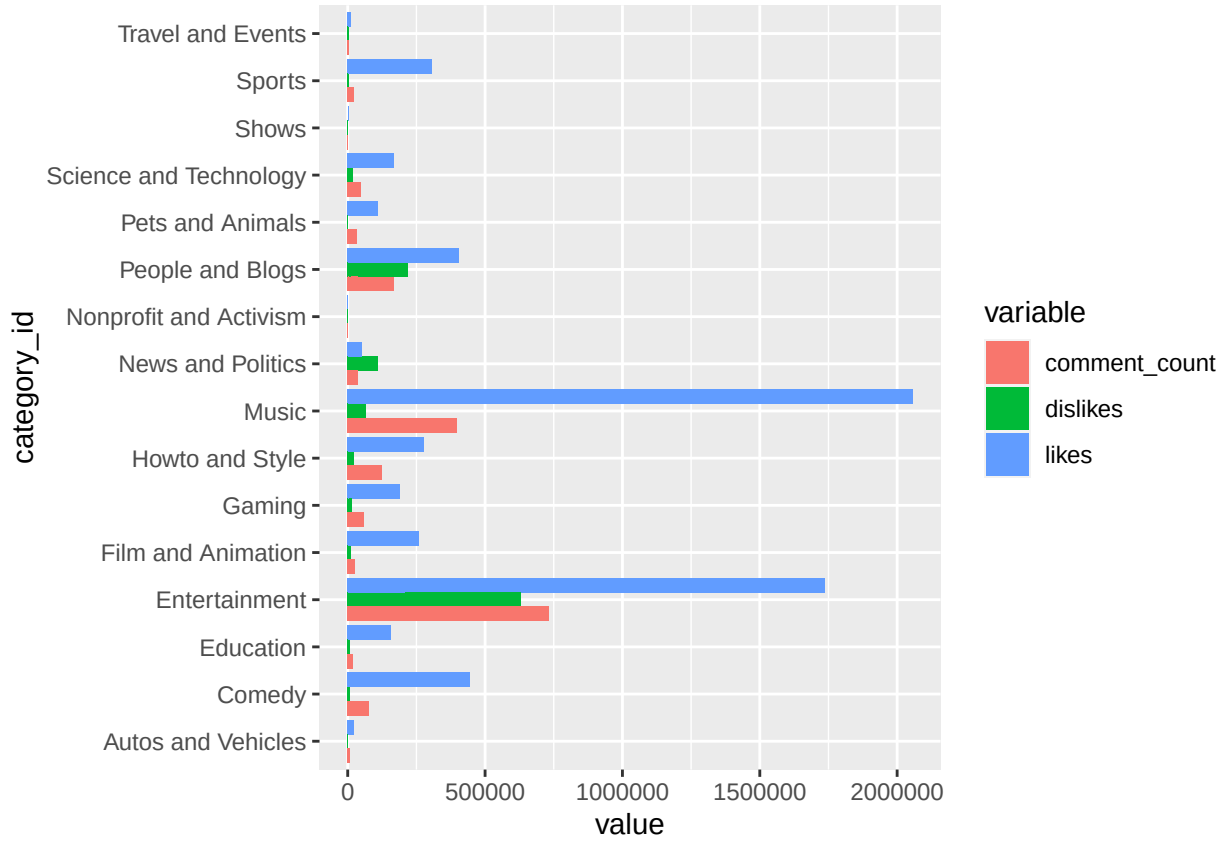
```
## Warning: package 'tidyr' was built under R version 4.0.4
```

```
# 4,8,7,9 pointing to category_id, likes, dislikes, comment_count
vids.select <- vids.u[vids.u$comments_disabled == F,
                      c(4,7,8,9)]

# reshape data from wide format to long format
vids.m <- pivot_longer(data = vids.select,
                       cols = -category_id,
                       names_to = "variable",
                       values_to = "value")

rbind(head(vids.m), tail(vids.m))
```

```
## # A tibble: 12 x 3
##    category_id      variable      value
##    <fct>            <chr>         <dbl>
##  1 People and Blogs likes         57527
##  2 People and Blogs dislikes       2966
##  3 People and Blogs comment_count 15954
##  4 Entertainment    likes         97185
##  5 Entertainment    dislikes       6146
##  6 Entertainment    comment_count 12703
##  7 Sports           likes           136
##  8 Sports           dislikes          5
##  9 Sports           comment_count    16
## 10 Entertainment    likes          3246
## 11 Entertainment    dislikes         28
## 12 Entertainment    comment_count   386
```

And using the melted dataframe, we can now create `ggplot`. As a start, we'll create a column plot and visualize the `value` of each `variable` in each of the `category_id`. We map the color of each bars to `variable_id` using `fill` like the following:

```
ggplot(vids.m, aes(x=value, y=category_id))+
  geom_col(position="dodge", aes(fill=variable))
```



If you have wanted a stacked bar plot instead of side-by-side bar plots, all we need to do is to substitute the `position="dodge"` parameter with `position="stack"`. Try that in the code chunk above and see the resulting plot adjust to that!

Using the same technique above, we'll create another melted dataframe using only videos that have more dislikes than likes (`dislikes >= likes`) with the exclusion of any video that have their ratings disabled:

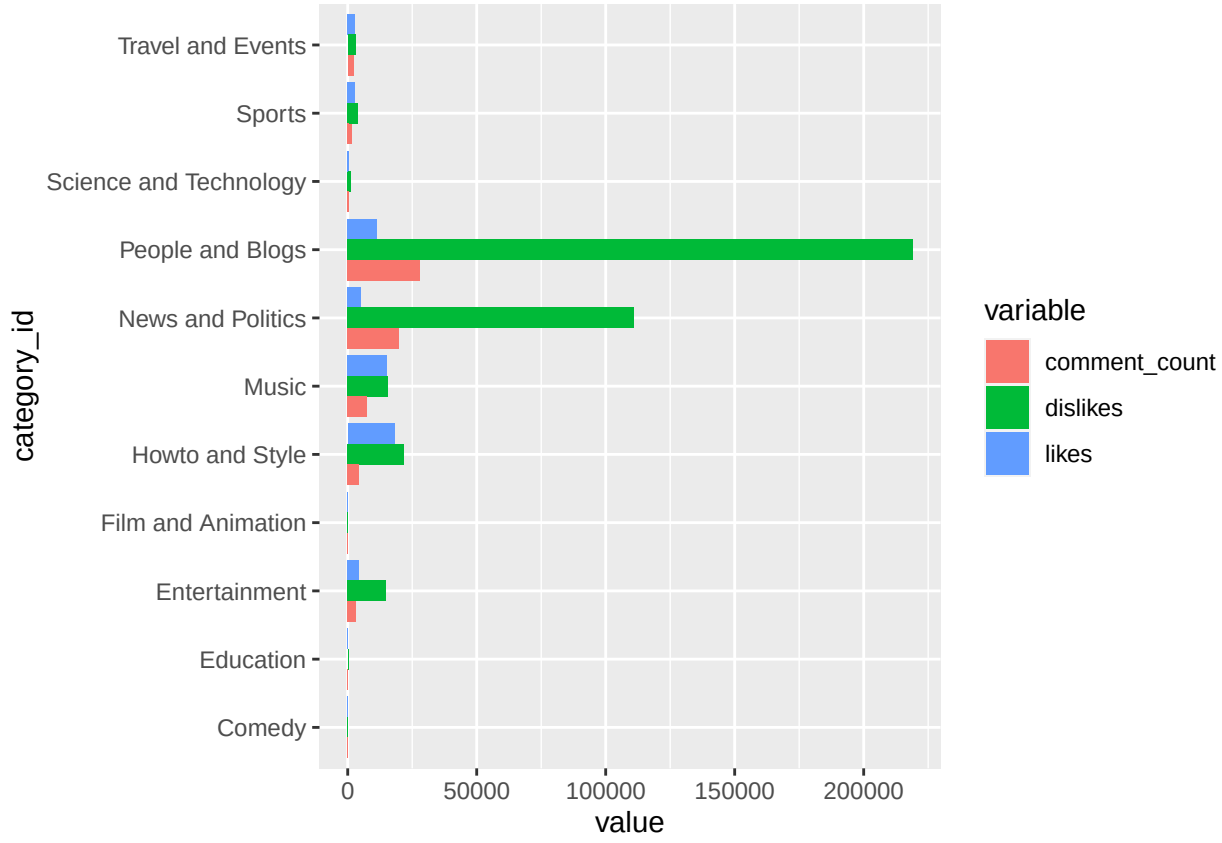
```
# 4,8,7,9 pointing to category_id, likes, dislikes, comment_count
vids.select2 <- vids.u[vids.u$dislikes >= vids.u$likes & vids.u$ratings_disabled == F,
                       c(4,7,8,9)]

vids.hate <- pivot_longer(data = vids.select2,
                          cols = -category_id,
                          names_to = "variable",
                          values_to = "value")
head(vids.hate)
```

```
## # A tibble: 6 x 3
##   category_id variable      value
```

```
##   <fct>       <chr>          <dbl>
## 1 Music       likes          15186
## 2 Music       dislikes       15448
## 3 Music       comment_count   7484
## 4 Sports      likes           2017
## 5 Sports      dislikes        2425
## 6 Sports      comment_count   1447
```

And let's create a dodge-type bar plot just like above:

```
ggplot(vids.hate, aes(x=value, y=category_id))+
  geom_col(position="dodge", aes(fill=variable))
```



From the plot above, the trending videos that were disliked by YouTube users seem to have come from the **People and Blogs** and **News and Politics** categories in no small amount. This may indicate a category-level pattern, or such skew may be attributed to only a handful of "bad actors". To investigate further, let's apply the techniques we've learned above but this time we'll melt the dataframe by `channel_title` instead of categories:

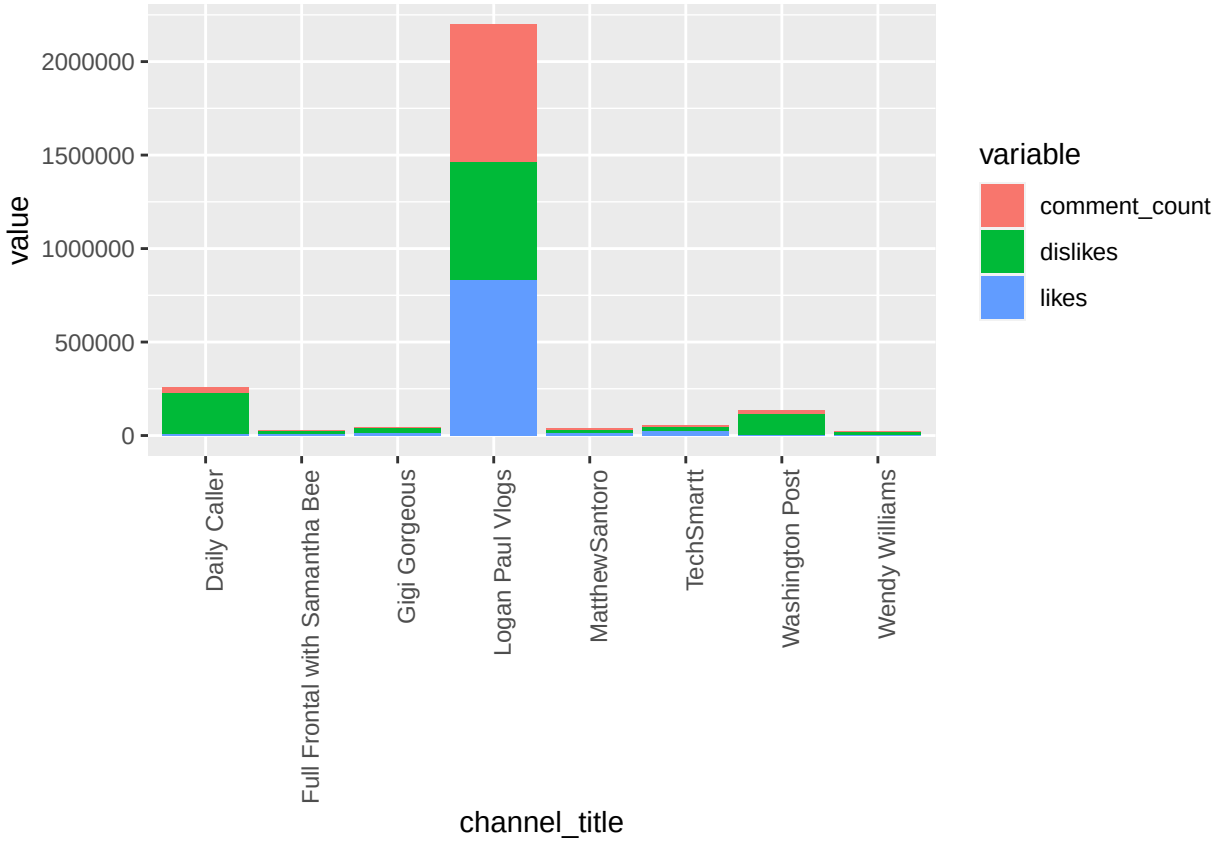
```
vids.hate2 <- pivot_longer(data = vids.u[vids.u$dislikes >= 10000 &
                                           vids.u$dislikes*2 >= vids.u$likes &
                                           vids.u$ratings_disabled == F,
                                         c(3,7,8,9)],
                           cols = -channel_title,
                           names_to = "variable",
                           values_to = "value")
```

```
head(vids.hate2[order(vids.hate2$channel_title),])
```

```
## # A tibble: 6 x 3
##   channel_title                  variable       value
##   <chr>                          <chr>          <dbl>
## 1 Daily Caller                   likes           9100
## 2 Daily Caller                   dislikes      218841
## 3 Daily Caller                   comment_count  28013
## 4 Full Frontal with Samantha Bee likes          11383
## 5 Full Frontal with Samantha Bee dislikes       12122
## 6 Full Frontal with Samantha Bee comment_count   4289
```

And creating our `ggplot()`:

```
ggplot(vids.hate2, aes(x=channel_title, y=value))+
  geom_col(aes(fill=variable))+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



Sure enough, we see that there are three peculiar cases from "Daily Caller", "Logan Paul Vlogs" and to a lesser extent "Washington Post".

Daily Caller is a far-right news and opinion website that posted a controversial and potentially offensive video titled "PSA from Chairman of the FCC Ajit Pai". In this video, the FCC Chairman dismisses concerns over net neutrality while dancing around in a Santa suit to "Harlem Shake" and swinging a lightsaber. Logan Paul, back in December 2017 published a disturbing video titled "Found Dead Body in Forest" that featured

actual footage of what appears to be a dead body when he and his friends ventured into Aokigahara (known also as the Suicide Forest) and subsequently a video titled "so sorry".

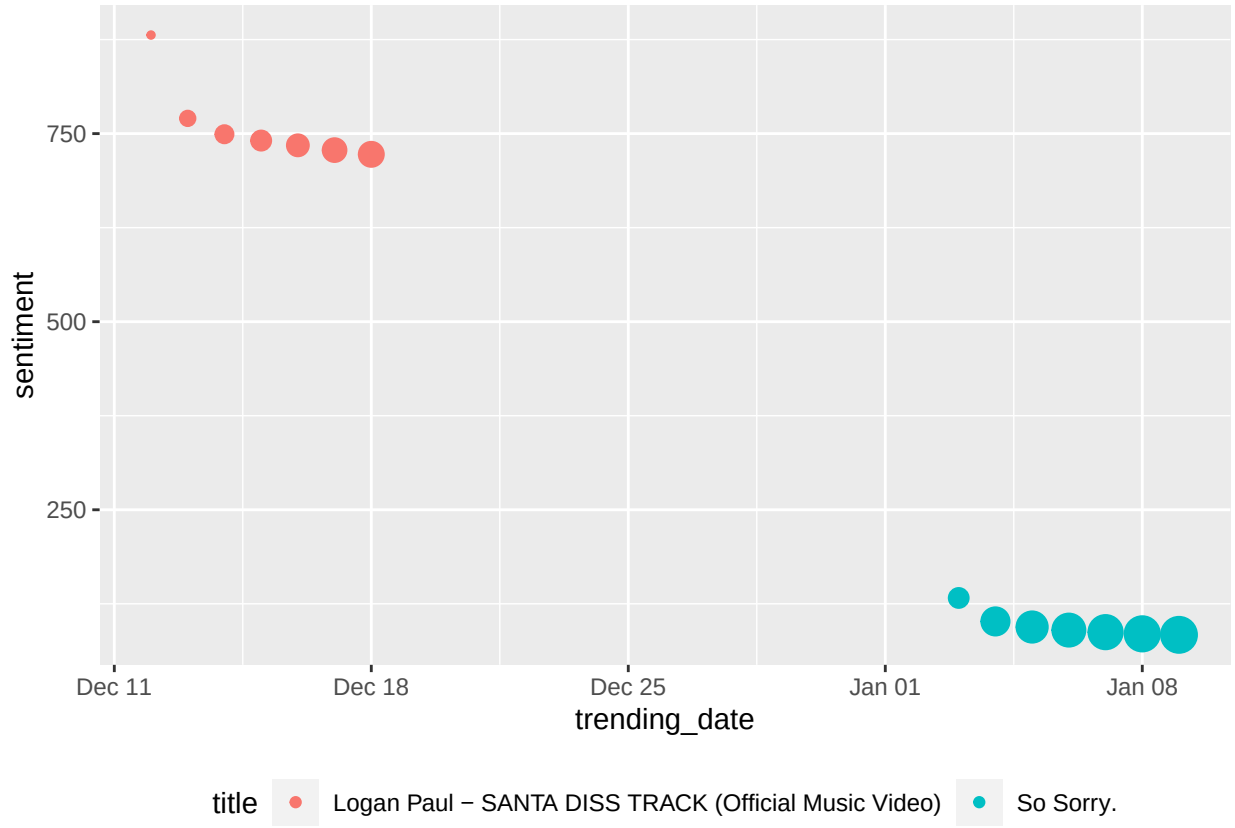
Washington Post's "The FCC repeals its net neutrality rules" was third on the most disliked trending videos, and it is actually a 3-hour long live stream in which federal regulators vote to allow Internet providers to speed up service for some apps and websites — and block or slow down others. The decision is considered highly unpopular.

Let's pick out the Trending videos published by Logan Paul and create a new variable that calculates the likes-to-dislikes ratio:

```
vids.logan <- vids[vids$channel_title == "Logan Paul Vlogs",]
vids.logan$sentiment <- (vids.logan$likes/vids.logan$dislikes)*100
```

The following is one way to compare the sentiments. There is nothing new in the following code and you should be able to fully understand what the code is doing:

```
ggplot(vids.logan, aes(x=trending_date, y=sentiment, color=title))+
  geom_point(aes(size=views))+
  guides(size=F)+
  theme(legend.position = "bottom")
```

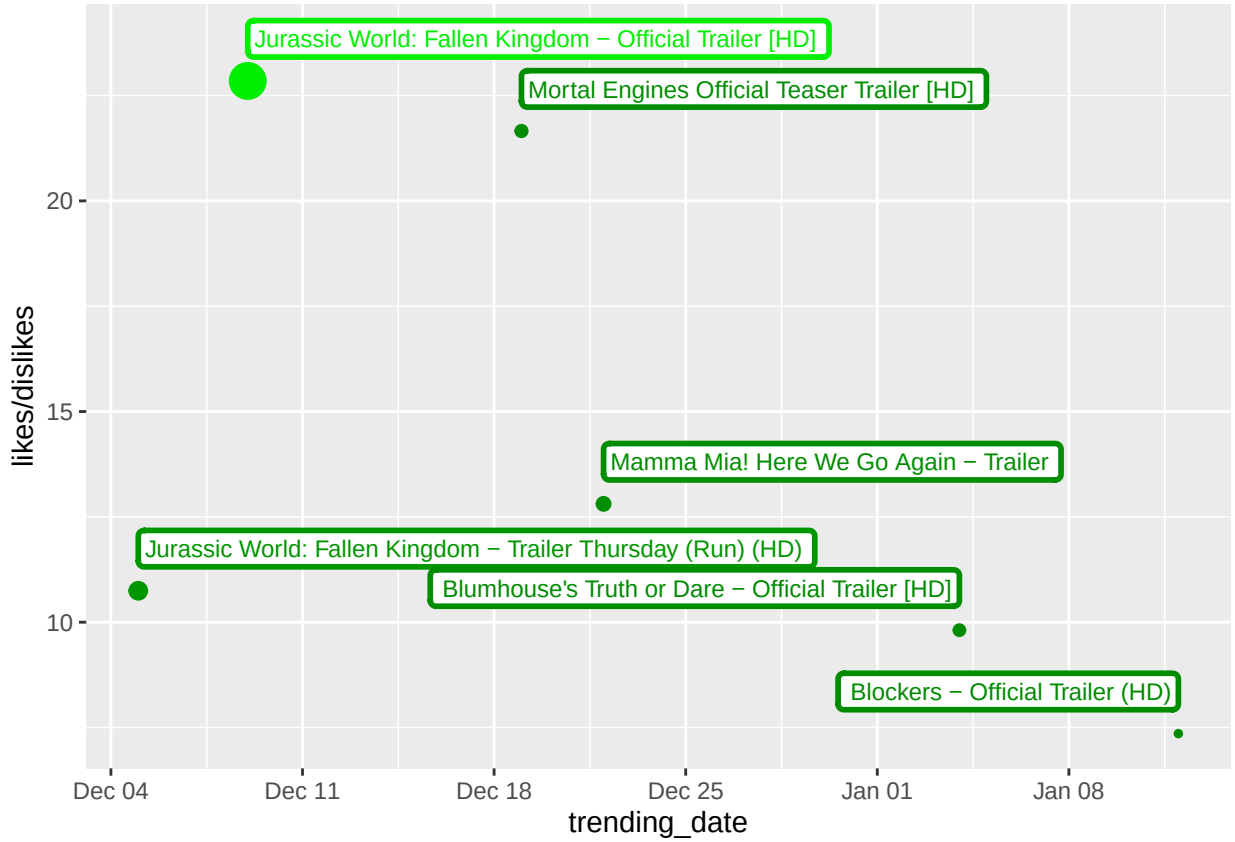


Observe that we're using the size to represent views, but that wasn't very obvious to our readers. We can add a legend or accompanying annotation to explain that, but as we'll see in the coming exercises there are better way to represent these data.

While the y-axis still represents the sentiment, we will add a layer of `geom_label` to label the movie names instead. A similar function would have been `geom_text` but since we're practically labelling our points, I've

opted for the `geom_label` instead. Here, we're looking at the trending videos from Universal Pictures during that period.

```
ggplot(vids.u[vids.u$channel_title == "Universal Pictures", ], aes(x=trending_date, y=likes/dislikes, c
  geom_point(aes(size=views),show.legend = F)+
  geom_label(aes(label=title), nudge_y = 1, label.size = 1, hjust="inward", size=3)+
  scale_color_gradient(low="green4", high="green2")+
  theme(legend.position = "none")
```



Now this plot communicates about the same level of information with more directness (no need for the use of legends), and it's immediately clear that **Jurassic World: Fallen Kingdom - Official Trailer [HD]** is the most-viewed, as well as the most-loved (highest likes to dislikes ratio) among them.

Hands-on ggplot: Multiple groups of data

To add a bit of diversity in our exercise, we'll now subset from our original data any videos produced by the channel **Tasty** and we will also compute the mean on each of these videos across the different days it was featured on - these are represented as `tasty` and `tasty.agg` respectively:

```
tasty <- vids[vids$channel_title == "Tasty", ]
tasty.agg <- aggregate.data.frame(tasty$views, by = list(tasty$title), mean)
names(tasty.agg) <- c("title", "mean")
tasty.agg
```

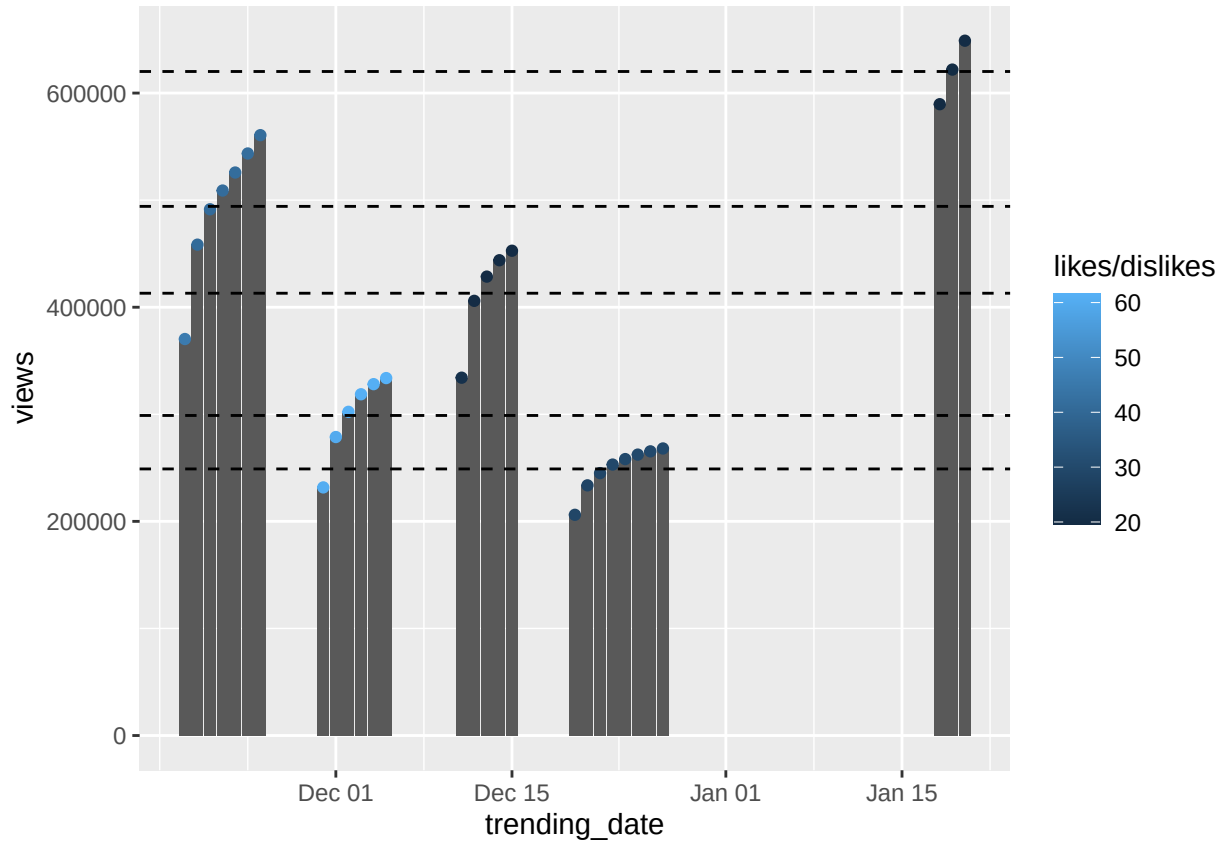
```
##                                                    title     mean
```

```
## 1 5 Desserts To Share with your BFF at Max Brenner Chocolate Bar 298912.2
## 2                                 Homemade Vs. Store-bought: Pasta 620141.7
## 3                       Restaurant Vs. Homemade Chicken Parm Pizza 494174.7
## 4                       The Best Ever Vegan Chocolate Chip Cookies 248983.6
## 5                                            Ultimate Bacon Recipes 413071.2
```

Sometimes we want to visualize relationships between variables in multiple subsets of the data - in these situations a particularly elegant solution is to have the plots appearing in panels defined in our plot construction process. These types of plots are called facet plots and is an incredible feature of ggplot. Let's see how we can take advantage of that!

Here we're creating a plot using the `tasty` dataset, first by creating the columns and then adding a second layer of points, and finally horizontal lines `geom_hline` (an equivalent for vertical line would be `geom_vline`) for each mean in our `tasty.agg` dataframe. By passing in `data=tasty.agg`, it overrides the initial data (`tasty`) allowing us to add plot elements using aesthetic from a different dataframe. Because we have 5 rows of means, we would expect 5 dashed lines (`linetype=2`).

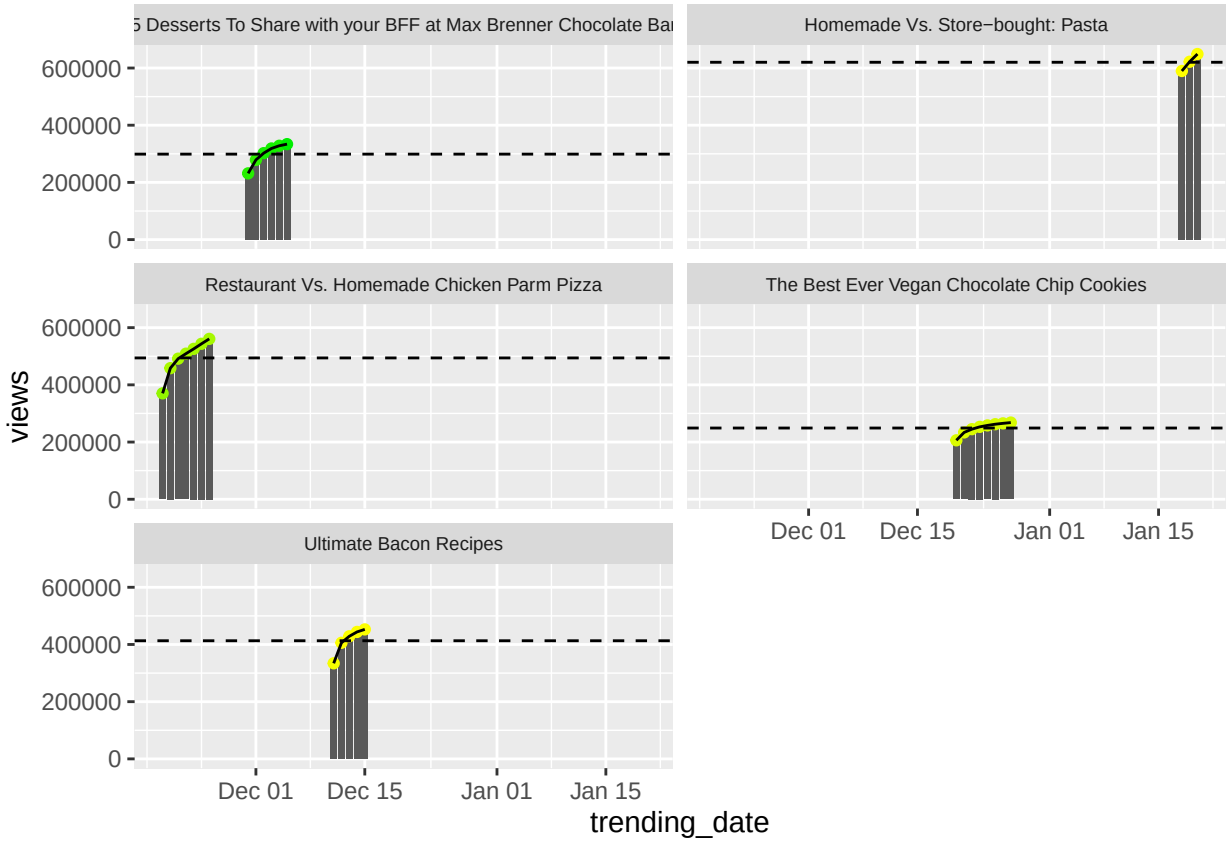
```
g1 <- ggplot(tasty, aes(x=trending_date, y=views))+
  geom_col()+
  geom_point(aes(col=likes/dislikes))+
  geom_hline(data=tasty.agg, aes(yintercept=mean),linetype=2)

g1
```



It isn't a bad start, but it may have been a better design decision to break the plot up into panels by each video so we can have a better observation of each video's trend. This could be established using

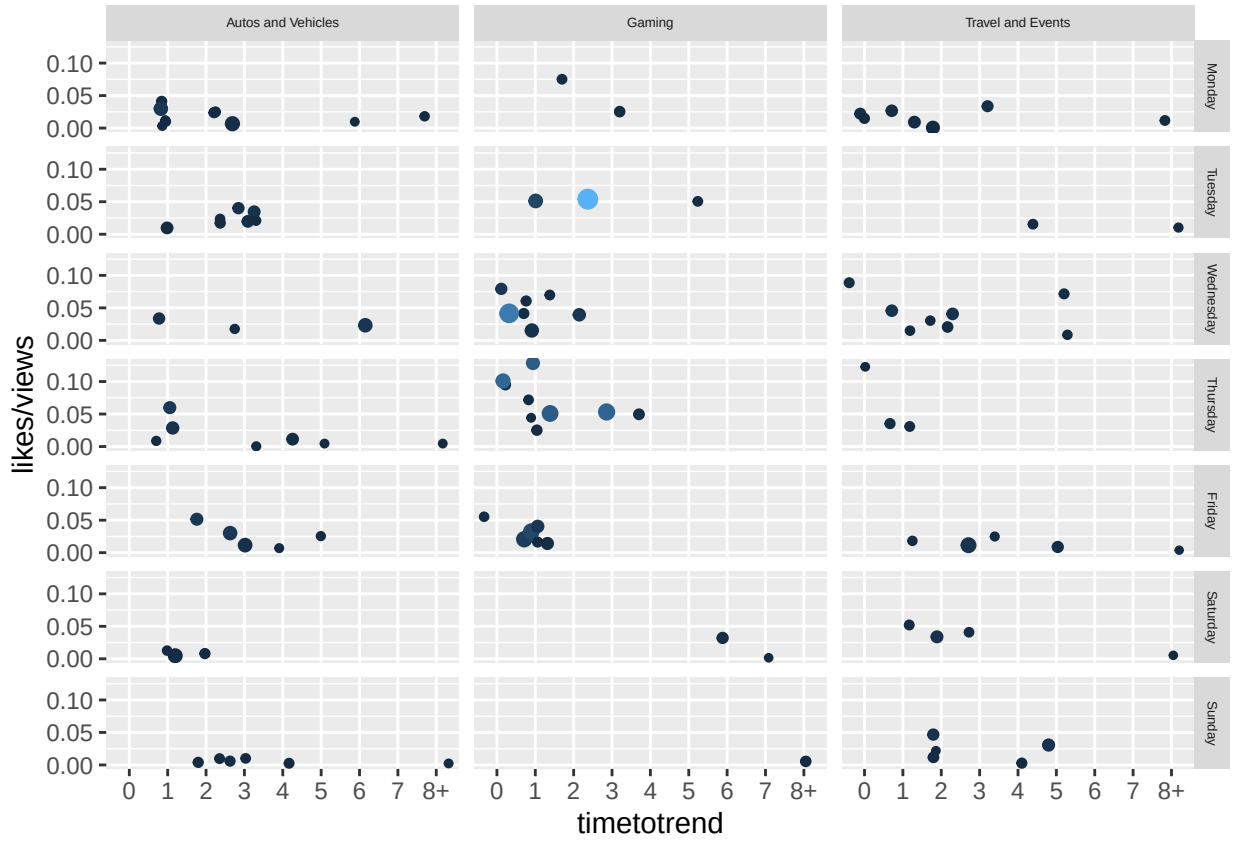
`facet_wrap()`. `facet_wrap` wraps our plots into a multirow panel of plots. Here we'll specify the number of columns to be 2 using `ncol=2`:

```
g1 +
  geom_line(col="black")+
  facet_wrap(~as.factor(title), ncol=2)+
  scale_color_gradient(low="yellow", high="green2")+
  theme(legend.position = "none",
        strip.text=element_text(size=7))
```



In a similar way to `facet_wrap`, we can use `facet_grid` to specify which variables are used to split our plots along the rows and columns. If we want the row to be the day of the week and columns to represent each category, we could achieve that by adding `facet_grid(publish_wday~as.factor(category_id))`:

```
ggplot(data=vids.agt, aes(x=timetotrend, y=likes/views))+
  geom_jitter(aes(size=views, col=likes))+
  facet_grid(publish_wday~as.factor(category_id))+
  scale_size(range=c(1,3))+
  theme(legend.position = "none",
        strip.text=element_text(size=5))
```



Dive Deeper: Using facet_grids or facet_wraps on Vox media Imagine working with Vox media and tasked to produce a visualization using data of their trending videos in past recent weeks. Use either `facet_wrap` or `facet_grid` in your visualization.

```
vids.vox <- vids.u[vids.u$channel_title == "Vox", ]
vids.vox$sentiment <- vids.vox$likes/vids.vox$dislikes
vids.vox$title <- as.factor(vids.vox$title)
head(vids.vox)
```

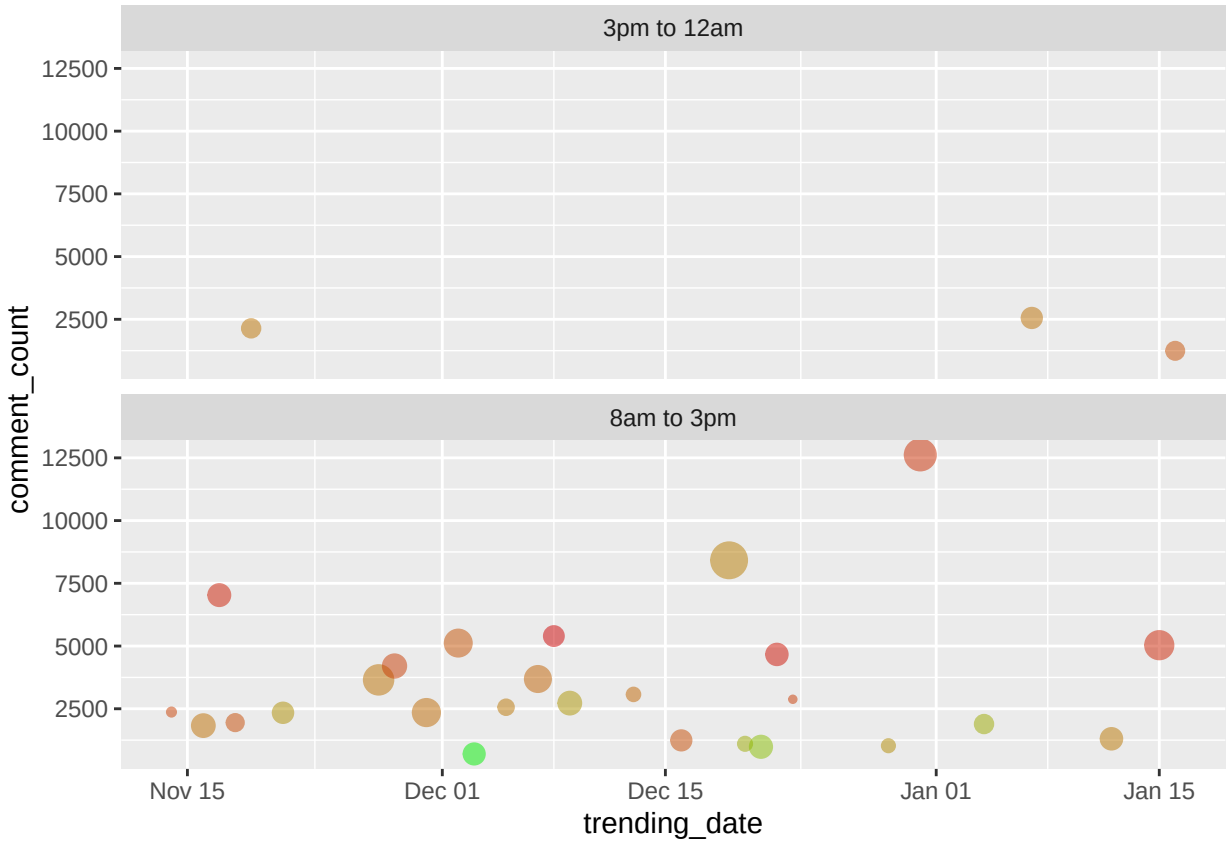
```
##      trending_date                                                 title
## 10      2017-11-14  Why the rise of the robots won%Ûªt mean the end of work
## 411     2017-11-16    The all-American fruit you've probably never heard of
## 614     2017-11-17                                    Walking while black
## 807     2017-11-18          The environmental cost of free two-day shipping
## 1003    2017-11-19                The military coup in Zimbabwe, explained
## 1419    2017-11-21 How job surveillance is transforming trucking in America
##      channel_title       category_id        publish_time  views likes dislikes
## 10             Vox News and Politics 2017-11-13 13:45:16 256426 12654     1363
## 411            Vox News and Politics 2017-11-15 11:30:00 533940 12633      597
## 614            Vox News and Politics 2017-11-16 12:55:39 505886 23207     5375
## 807            Vox News and Politics 2017-11-17 13:00:12 366048 14742     1308
## 1003           Vox News and Politics 2017-11-18 16:09:31 396346 11277      499
## 1419           Vox News and Politics 2017-11-20 13:01:34 455289 14025      441
##      comment_count comments_disabled ratings_disabled video_error_or_removed
## 10            2368             FALSE            FALSE                  FALSE
## 411           1828             FALSE            FALSE                  FALSE
```

```
## 614           7030            FALSE           FALSE                  FALSE
## 807           1948            FALSE           FALSE                  FALSE
## 1003          2140            FALSE           FALSE                  FALSE
## 1419          2340            FALSE           FALSE                  FALSE
##      publish_hour publish_when publish_wday timetotrend sentiment
## 10             13   8am to 3pm       Monday           1  9.283933
## 411            11   8am to 3pm    Wednesday           1 21.160804
## 614            12   8am to 3pm     Thursday           1  4.317581
## 807            13   8am to 3pm       Friday           1 11.270642
## 1003           16  3pm to 12am     Saturday           1 22.599198
## 1419           13   8am to 3pm       Monday           1 31.802721
```

```
### Write your answer here and execute the code to produce your visualization

# -- Scroll up to refer to earlier code and copy / paste any if you like
# -- Try and attempt this challenge by discussion with your neighbor if necessary but do not blindly co
```

While I'll show you a reference answer here below, I strongly recommend you only use this as a reference upon completion of the **Dive Deeper** exercise.

```
voxp <- ggplot(vids.vox, aes(x=trending_date, y=comment_count, color=sentiment))+
  # geom_point is a reference, geom_col or other geoms (sensible choices) would work too
  geom_point(aes(size=views),show.legend = F, alpha=0.5)+
  scale_color_gradient(low="red3", high="green2")+
  # facet_wrap by publish when is a reference, any other variables (sensible choices) would work
  facet_wrap(~publish_when, ncol=1)

voxp
```



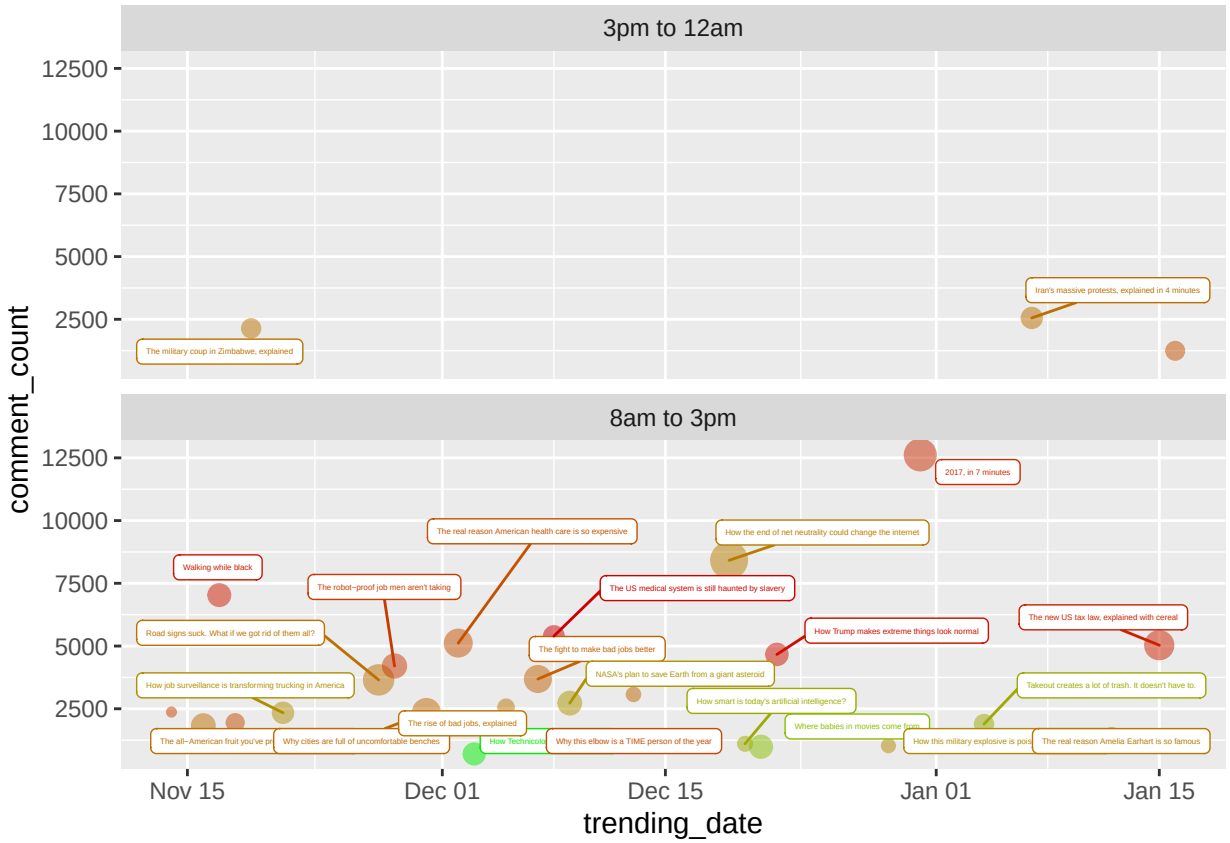
There are some pretty interesting points in the above plot. The bright green one near "December 01" is an interesting video we may want to further study, and the one that crosses 12,500 comment count is just about as interesting. One quick fix is to replace each points with `geom_text` or use `geom_label` to label them:

```
voxp +
  geom_label(aes(label=title), size=1.4, nudge_y = 800) +
  theme(legend.position = "none")
```

Notice that a lot of these labels overlapped with each other and are generally just noisy to look at (a phenomenon sometimes called "overplotting"). We have a few ways to fix that. First, is to only plot points that could be potentially interesting. Second, is to have the text or labels "repel" each other, creating more space between these different points.

This can be achieved using a library called `ggrepel`:

```
library(ggrepel)
vox.pop <- vids.vox[vids.vox$views >= 400000 | vids.vox$sentiment >= median(vids.vox$sentiment), 2]

voxp +
  geom_label_repel(aes(label=title), size=1.2, data=vids.vox[vids.vox$title %in% vox.pop,], box.padding
  theme(legend.position = "none")
```



Now this result is a lot neater, and definitely more comfortable on the eyes. We see that "2017, in 7 minutes" isn't at all well-liked in terms of sentiment (more dislikes to likes ratio than many other videos on average) even though it does seem to be trendy in every other measures: views, indicated by the point's size; and comment counts, indicated by it's position on the y-axis.

Topics that are trendy (lots of views, lots of comment) but unpopular do also seem to share a thematic commonality:
- The real reason American health care is so expensive
- How Trump makes extreme things look normal
- The new US law, explained with cereal
- Walking while black
- The US medical system is still haunted by slavery

These 5 videos are closest to the negative end of the sentiment scale, and the topics are sensitive or political in nature. Videos that do seem to do very well (high approval ratings) are:
- How technicolor changed movies
- Where babies in movies come from
- How smart is today's artificial intelligence?
- Takeout creates a lot of trash. It doesn't have to
- Nasa's plan to save Earth from a giant asteroid

These videos seem to be scientific in nature - so this visualization do seem to suggest a certain correlation between thematic elements and approval ratings.

Can you discuss any other findings or propose any preliminary observations from the plot above? How would you improve the plot above?

Learn-by-building Module: Data Visualization

To help us get ready to the learn-by-building assignment, we'll walk through a simple exercise together. There isn't any new concept being introduced, but rather a start-to-finish recap of the data visualization process.

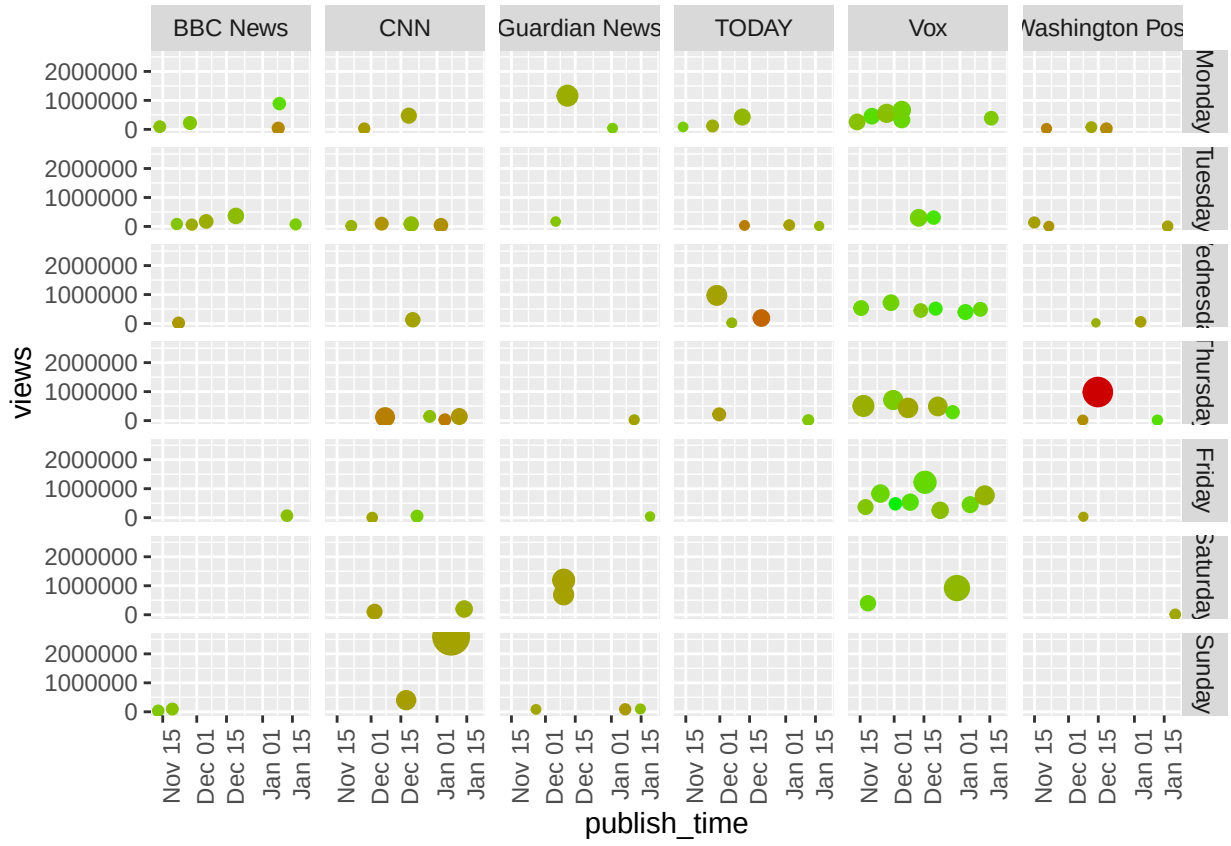
First, we'll have to subset any videos within the News and Politics category and take only the ones that have more than 10 trending videos during the explanatory period. We run the following code and see that 6 channels / publishers satisfy that condition:

```
news <- vids.u[vids.u$category_id == "News and Politics", ]
news <- aggregate(trending_date ~ channel_title, news, length)
news <- news[news$trending_date > 10, ]
news <- news[order(news$trending_date, decreasing = T),]
news
```

```
##      channel_title trending_date
## 83             Vox            29
## 30             CNN            17
## 10        BBC News            12
## 84 Washington Post            12
## 40   Guardian News            11
## 78           TODAY            11
```

We can now use `vids.u$channel_title %in% news$channel_title` as our data source, indicating that we only wish to create our ggplot using channels that are in the list of 6 news channels above. The rest of the code is relatively straightforward:

```
ggplot(data=vids.u[vids.u$channel_title %in% news$channel_title,], aes(x=publish_time, y=views))+
  geom_point(aes(col=log(likes/dislikes), size=comment_count))+
  facet_grid(publish_wday~channel_title)+
  scale_color_gradient(low="red3", high="green2")+
  theme(legend.position = "none",
        axis.text.x = element_text(angle = 90, hjust = 1))
```



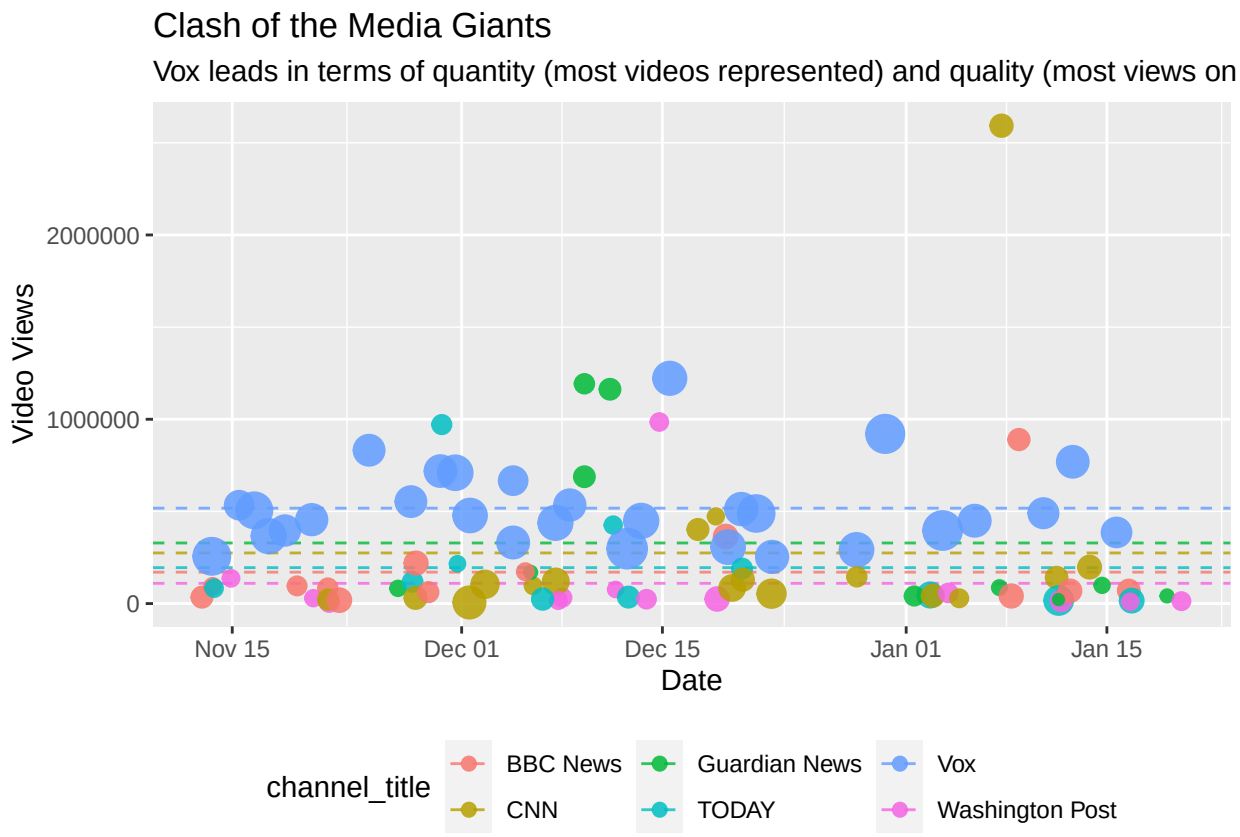
We don't always need to use a facet plot, if the information you wish to convey doesn't require that. The following plot does not tells us the sentiment or comment count of each video, but does enough to tell us about each channel's representation in the trending videos.

```
news.agg <- vids.u[vids.u$channel_title %in% news$channel_title, ]
news.agg <- aggregate(views ~ channel_title, news.agg, mean)
names(news.agg) <- c("channel_title", "mean_views")
news.agg
```

```
##     channel_title mean_views
## 1        BBC News   170525.0
## 2             CNN   274701.6
## 3   Guardian News   328777.5
## 4           TODAY   194893.6
## 5             Vox   517597.6
## 6 Washington Post   109853.3
```

The plot:

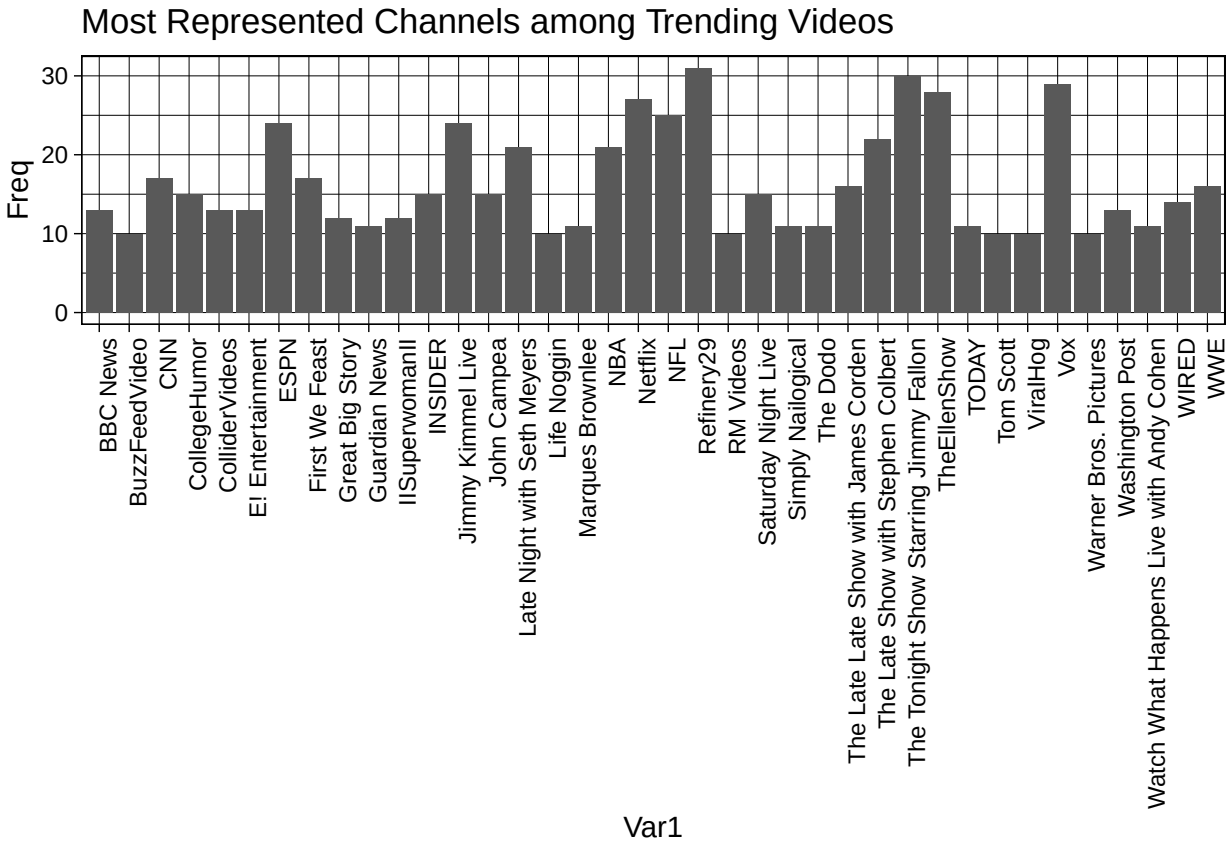
```
ggplot(data=vids.u[vids.u$channel_title %in% news$channel_title,], aes(x=publish_time, y=views))+
  geom_hline(data=news.agg, aes(yintercept=mean_views, col=channel_title), linetype=2, alpha=0.8)+
  geom_point(aes(size=likes/views, col=channel_title), stroke=1, alpha=0.85)+
  guides(size=F)+
  labs(title="Clash of the Media Giants", x="Date", y="Video Views", subtitle="Vox leads in terms of qua
  theme(legend.position = "bottom")
```



Using Themes

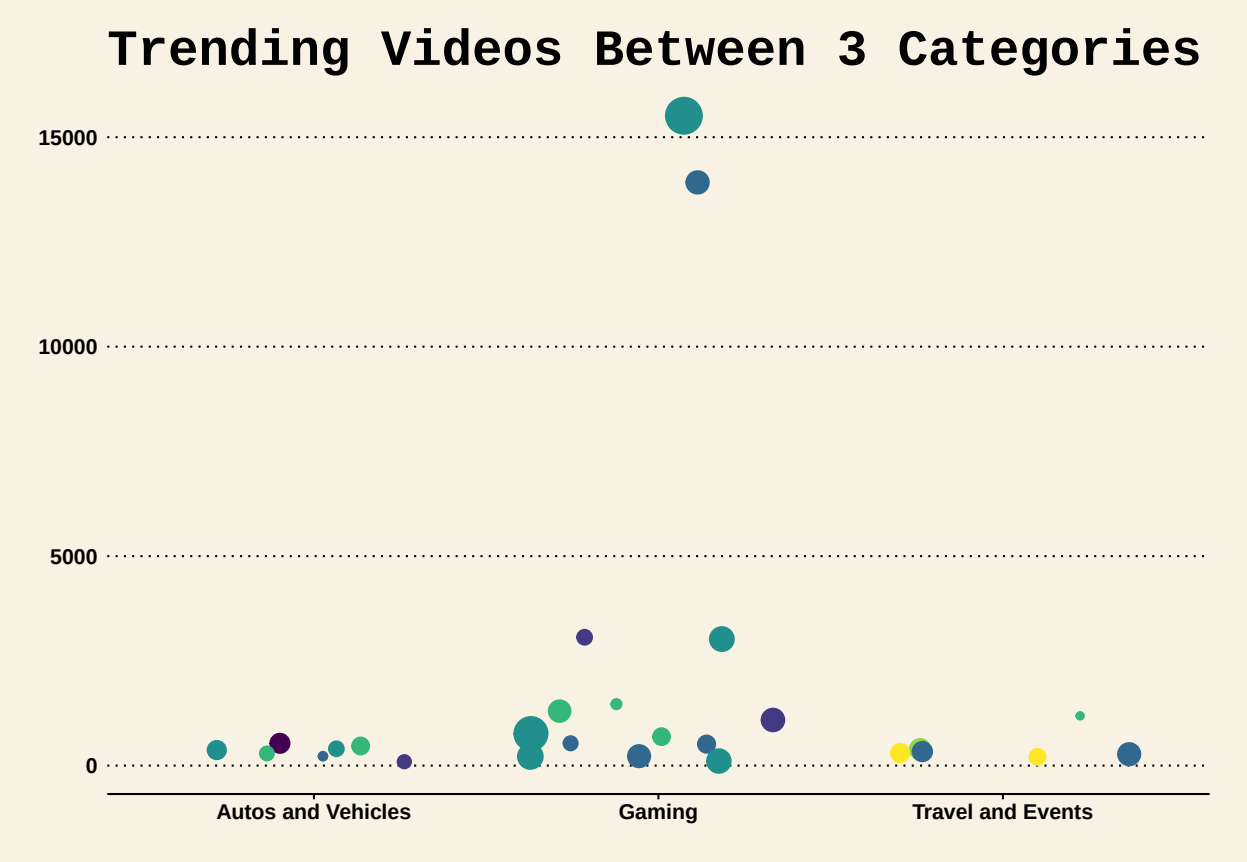
We can spice up our visualizations using another nifty feature of `ggplot`: themes! I've copied and pasted the code from our earlier exercise and added a theme using `theme_linedraw()`. I invite you to go ahead and swap out the theme and replace it with one of the other themes. Examples:
- `theme_calc()`
- `theme_excel()` - `theme_gdocs()`
- `theme_classic()`

```
ggplot(temp1, aes(x=Var1, y=Freq))+
  geom_col()+
  theme_linedraw()+
  labs(title="Most Represented Channels among Trending Videos")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



Now try one more example - try and replace `theme_wsj` with one of the other themes in the following plot. A simple switch of the overall presentation "theme" using a short function like `theme_` makes it easy to experiment with different aesthetics and is one of yet another benefit of using ggplot!

```
ggplot(data = vids.ags, aes(x=category_id, y=dislikes, size=comment_count/views))+
  geom_jitter(aes(col=publish_wday))+
  theme_wsj(base_size = 8)+
  labs(title = "Trending Videos Between 3 Categories")+
  theme(legend.position = "none")
```



Introduction to Leaflet [Optional]

Leaflet is among the most popular JS library for interative maps, used by websites such as The New York Times, The Washington Post, GitHub and Flickr[3]. The R package `leaflet` allows us to create leaflet maps directly in R code. The steps are as follow:

1. Create a map widget by calling leaflet().
2. Add layers (i.e., features) to the map by using layer functions (`addTiles`, `addMarkers`, `addPolygons`) to modify the map widget.

Sounds similar enough to the ggplot system? Let's see a simple example.

I'm going to create two objects to be used for our Leaflet map later. First, an icon! Here I'm using Algoritma's main icon and saving it to an object called `ico`. Next, we'll create `loca`, a data frame that has two variables (`lat` and `lng`) with some randomly generated numbers. The code is straightforward:

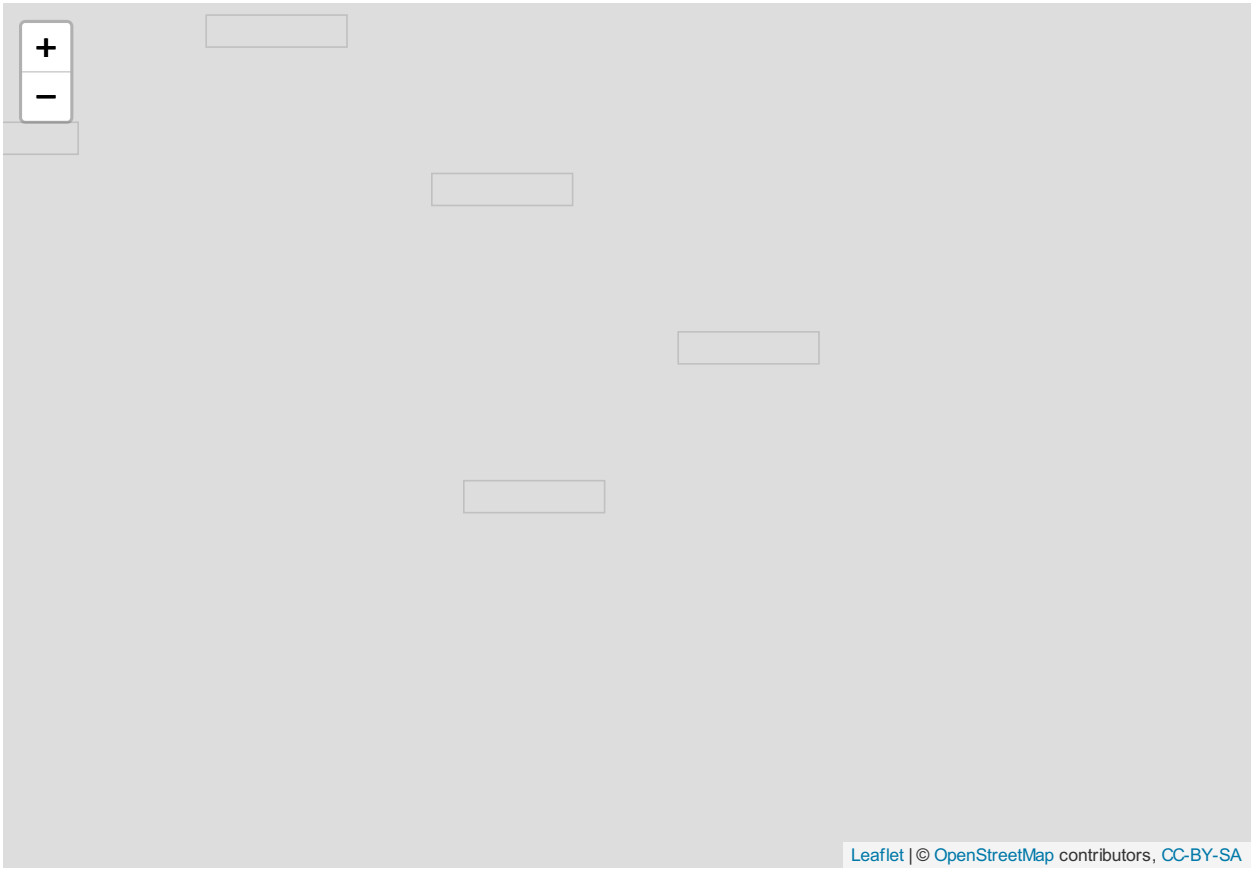
```
set.seed(418)
library(leaflet)

ico <- makeIcon(
    iconUrl = "https://algorit.ma/wp-content/uploads/2017/07/logo_light_trans.png",
    iconWidth=177/2, iconHeight=41/2
```

[3]Official Documentation, Leaflet

```
)


loca <- data.frame(lat=runif(5, min = -6.24, max=-6.23),
                   lng=runif(5, min=106.835, max=106.85))
```

And we'll now create our map:

```
# create a leaflet map widget
map1 <- leaflet()

# add tiles from open street map
map1 <- addTiles(map1)

# add markers
map1 <- addMarkers(map1, data = loca, icon=ico)

map1
```



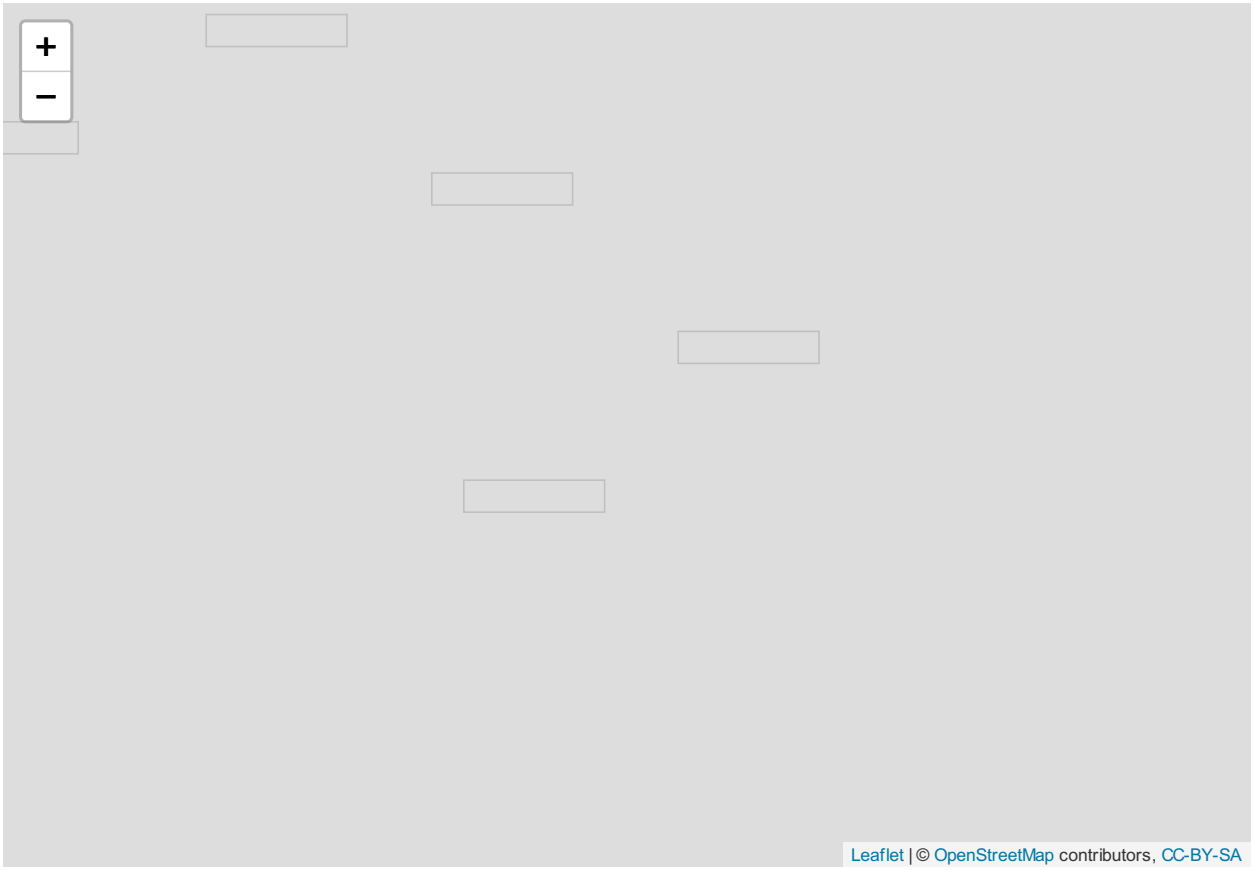
Supposed we want the end user to be able to click on each of these icons and have a simple pop-up description, we can add that to our map too!

Create the pop-up text:

```
pops <- c(
    "<h3>Algoritma Main HQ</h3><p>Visit us here!</p>",
    "<strong>Algoritma Business Campus</strong>",
    "<h3>In-Construction</h3><p>New Secondary Campus</p>",
    "<strong>Secondary Campus</strong>",
    "<strong>The Basecamp (business-school)</strong>"
)
```

Adding them to our map:

```
map1 <- leaflet()
map1 <- addTiles(map1)
map1 <- addMarkers(map1, data = loca, icon=ico, popup = pops)

map1
```



`leaflet` does a lot more than the simple demonstration above, but since it belongs to the optional part of this coursebook - I'll leave it up to you, the readers, to further explore its possibilities! While I would recommend you to use `ggplot` as the main focus of your graded assignment, I want to leave the choice up to you. Work with your academic mentors to produce a visualization as specified in the learn-by-building module and good luck!

Summary

The coursebook covers many aspects of plotting, including using visualization libraries such as `ggplot2`, `leaflet` and a few other supporting libraries. I hope you've managed to get a good grasp of the plotting philosophy behind `ggplot2`, and have built a few visualizations with it yourself!

Happy coding!

Samuel

Annotations

Cleveland (1985), page 264: "Data that can be shown by pie charts always can be shown by a dot chart. This means that judgements of position along a common scale can be made instead of the less accurate angle judgements." This statement is based on the empirical investigations of Cleveland and McGill as well as investigations by perceptual psychologists."