Sunday night was a big night for Taylor Swift – not only was she nominated for multiple [Billboard Music Awards](https://www.billboard.com/articles/news/bbma/8456842/billboard-music-awards-2018-winners-list-bbmas); she took home Top Female Artist and Top Selling Album.

statistics-sunday-taylor-swift-vs-lorde

Now you'll want to load geniusR and tidyverse so we can work with our data.

**library**(geniusR)

**library**(tidyverse)

*## ── Attaching packages ────────────────────────────────────────────────────────────────────────────────────────────── tidyverse 1.2.1 ──*

*## ✔ ggplot2 2.2.1 ✔ purrr 0.2.4*

*## ✔ tibble 1.4.2 ✔ dplyr 0.7.4*

*## ✔ tidyr 0.8.0 ✔ stringr 1.3.0*

*## ✔ readr 1.1.1 ✔ forcats 0.3.0*

*## ── Conflicts ───────────────────────────────────────────────────────────────────────────────────────────────── tidyverse\_conflicts() ──*

*## ✖ dplyr::filter() masks stats::filter()*

*## ✖ dplyr::lag() masks stats::lag()*

For today's demonstration, I'll be working with data from two artists I love: Taylor Swift and Lorde. Both dropped new albums last year, Reputation and Melodrama, respectively, and both, though similar in age and friends with each other, have very different writing and musical styles.  
  
geniusR has a function genius\_album that will download lyrics from an entire album, labeling it by track.

swift\_lyrics <- **genius\_album**(artist="Taylor Swift", album="Reputation")

*## Joining, by = c("track\_title", "track\_n", "track\_url")*

lorde\_lyrics <- **genius\_album**(artist="Lorde", album="Melodrama")

*## Joining, by = c("track\_title", "track\_n", "track\_url")*

Now we want to tokenize our datasets, remove stop words, and count word frequency - this code should look familiar, except this time, I'm combining them using the pipeline symbol (%>%) from the tidyverse, which allows you to string together multiple functions without having to nest them.

**library**(tidytext)

tidy\_swift <- swift\_lyrics %>%

**unnest\_tokens**(word,lyric) %>%

**anti\_join**(stop\_words) %>%

**count**(word, sort=TRUE)

*## Joining, by = "word"*

**head**(tidy\_swift)

## # A tibble: 6 x 2

## word n

## <chr> <int>

## 1 call 46

## 2 wanna 37

## 3 ooh 35

## 4 ha 34

## 5 ah 33

## 6 time 32

tidy\_lorde <- lorde\_lyrics %>%

**unnest\_tokens**(word,lyric) %>%

**anti\_join**(stop\_words) %>%

**count**(word, sort=TRUE)

*## Joining, by = "word"*

**head**(tidy\_lorde)

## # A tibble: 6 x 2

## word n

## <chr> <int>

## 1 boom 40

## 2 love 26

## 3 shit 24

## 4 dynamite 22

## 5 homemade 22

## 6 light 22

Looking at the top 6 words for each, it doesn't look like there will be a lot of overlap. But let's explore that, shall we? Lorde's album is 3 tracks shorter than Taylor Swift's. To make sure our word comparisons are meaningful, I'll create new variables that take into account total number of words, so each word metric will be a proportion, allowing for direct comparisons. And because I'll be joining the datasets, I'll be sure to label these new columns by artist name.

tidy\_swift <- tidy\_swift %>%

**rename**(swift\_n = n) %>%

**mutate**(swift\_prop = swift\_n/**sum**(swift\_n))

tidy\_lorde <- tidy\_lorde %>%

**rename**(lorde\_n = n) %>%

**mutate**(lorde\_prop = lorde\_n/**sum**(lorde\_n))

There are multiple types of joins available in the tidyverse. I used an anti\_join to remove stop words. Today, I want to use a full\_join, because I want my final dataset to retain all words from both artists. When one dataset contributes a word not found in the other artist's set, it will fill those variables in with missing values.

compare\_words <- tidy\_swift %>%

**full\_join**(tidy\_lorde, by = "word")

**summary**(compare\_words)

## word swift\_n swift\_prop lorde\_n

## Length:957 Min. : 1.000 Min. :0.00050 Min. : 1.0

## Class :character 1st Qu.: 1.000 1st Qu.:0.00050 1st Qu.: 1.0

## Mode :character Median : 1.000 Median :0.00050 Median : 1.0

## Mean : 3.021 Mean :0.00152 Mean : 2.9

## 3rd Qu.: 3.000 3rd Qu.:0.00151 3rd Qu.: 3.0

## Max. :46.000 Max. :0.02321 Max. :40.0

## NA's :301 NA's :301 NA's :508

## lorde\_prop

## Min. :0.0008

## 1st Qu.:0.0008

## Median :0.0008

## Mean :0.0022

## 3rd Qu.:0.0023

## Max. :0.0307

## NA's :508

The final dataset contains 957 tokens - unique words - and the NAs tell how many words are only present in one artist's corpus. Lorde uses 301 words Taylor Swift does not, and Taylor Swift uses 508 words that Lorde does not. That leaves 148 words on which they overlap.  
  
There are many things we could do with these data, but let's visualize words and proportions, with one artist on the x-axis and the other on the y-axis.

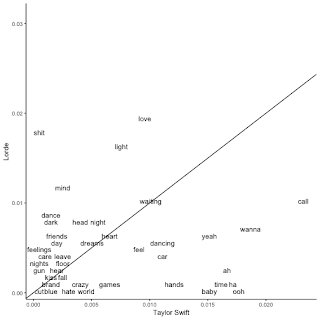
**ggplot**(compare\_words, **aes**(x=swift\_prop, y=lorde\_prop)) +

**geom\_abline**() +

**geom\_text**(**aes**(label=word), check\_overlap=TRUE, vjust=1.5) +

**labs**(y="Lorde", x="Taylor Swift") + **theme\_classic**()

**## Warning: Removed 809 rows containing missing values (geom\_text).**

[](https://4.bp.blogspot.com/-nRnN6fYkO78/WvZcS5ugnbI/AAAAAAAALic/YDNRzq54QSEewUwUJ6KzgC83r-zNEuacgCLcBGAs/s1600/unnamed-chunk-7-1.png)

The warning lets me know there are 809 rows with missing values - those are the words only present in one artist's corpus. Words that fall on or near the line are used at similar rates between artists. Words above the line are used more by Lorde than Taylor Swift, and words below the line are used more by Taylor Swift than Lorde. This tells us that, for instance, Lorde uses "love," "light," and, yes, "shit," more than Swift, while Swift uses "call," "wanna," and "hands" more than Lorde. They use words like "waiting," "heart," and "dreams" at similar rates. Rates are low overall, but if you look at the max values for the proportion variables, Swift's most common word only accounts for about 2.3% of her total words; Lorde's most common word only accounts for about 3.1% of her total words.

When I started this blog back in 2011, my goal was to write deep thoughts on trivial topics – specifically, to overthink and overanalyze pop culture and related topics that appear fluffy until you really dig into them. Recently, I’ve been blogging more about statistics, research, R, and data science, and I’ve loved getting to teach and share.

But sometimes, you just want to overthink and overanalyze pop culture.

Sentiment Analysis

Sentiment analysis works on unigrams - single words - but you can aggregate across multiple words to look at sentiment across a text.  
  
To demonstrate sentiment analysis, I'll use one of my favorite songs: "Hotel California" by the Eagles.  
  
I know, I know.

|  |
| --- |
|  |
| The Dude has had a rough night and he hates the f'ing Eagles. |

Using similar code as last week, let's pull in the lyrics of the song.

**library**(geniusR)

**library**(tidyverse)

hotel\_calif <- **genius\_lyrics**(artist = "Eagles", song = "Hotel California") %>%

**mutate**(line = **row\_number**())

First, we'll chop up these 43 lines into individual words, using the tidytext package and unnest\_tokens function.

**library**(tidytext)

tidy\_hc <- hotel\_calif %>%

**unnest\_tokens**(word,lyric)

This is also probably the point I would remove stop words with anti\_join. But these common words are very unlikely to have a sentiment attached to them, so I'll leave them in, knowing they'll be filtered out anyway by this analysis. We have 4 lexicons to choose from. Loughran is more financial and textual, but we'll still see how well it can classify the words anyway. First, let's create a data frame of our 4 sentiment lexicons.

new\_sentiments <- sentiments %>%

**mutate**(sentiment = **ifelse**(lexicon == "AFINN" & score >= 0, "positive",

**ifelse**(lexicon == "AFINN" & score < 0,

"negative", sentiment))) %>%

**group\_by**(lexicon) %>%

**mutate**(words\_in\_lexicon = **n\_distinct**(word)) %>%

**ungroup**()

Now, we'll see how well the 4 lexicons match up with the words in the lyrics. Big thanks to [Debbie Liske at Data Camp](https://www.datacamp.com/community/tutorials/sentiment-analysis-R) for this piece of code (and several other pieces used in this post):

my\_kable\_styling <- **function**(dat, caption) {

**kable**(dat, "html", escape = FALSE, caption = caption) %>%

**kable\_styling**(bootstrap\_options = **c**("striped", "condensed", "bordered"),

full\_width = FALSE)

}

**library**(kableExtra)

**library**(formattable)

**library**(yarrr)

tidy\_hc %>%

**mutate**(words\_in\_lyrics = **n\_distinct**(word)) %>%

**inner\_join**(new\_sentiments) %>%

**group\_by**(lexicon, words\_in\_lyrics, words\_in\_lexicon) %>%

**summarise**(lex\_match\_words = **n\_distinct**(word)) %>%

**ungroup**() %>%

**mutate**(total\_match\_words = **sum**(lex\_match\_words),

match\_ratio = lex\_match\_words/words\_in\_lyrics) %>%

**select**(lexicon, lex\_match\_words, words\_in\_lyrics, match\_ratio) %>%

**mutate**(lex\_match\_words = **color\_bar**("lightblue")(lex\_match\_words),

lexicon = **color\_tile**("lightgreen","lightgreen")(lexicon)) %>%

**my\_kable\_styling**(caption = "Lyrics Found In Lexicons")

*## Joining, by = "word"*

| Lyrics Found In Lexicons | | | |
| --- | --- | --- | --- |
| **lexicon** | **lex\_match\_words** | **words\_in\_lyrics** | **match\_ratio** |
| AFINN | 18 | 175 | 0.1028571 |
| bing | 18 | 175 | 0.1028571 |
| loughran | 1 | 175 | 0.0057143 |
| nrc | 23 | 175 | 0.1314286 |

NRC offers the best match, classifying about 13% of the words in the lyrics. (It's not unusual to have such a low percentage. Not all words have a sentiment.)

hcsentiment <- tidy\_hc %>%

**inner\_join**(**get\_sentiments**("nrc"), by = "word")

hcsentiment

## # A tibble: 103 x 4

## track\_title line word sentiment

## <chr> <int> <chr> <chr>

## 1 Hotel California 1 dark sadness

## 2 Hotel California 1 desert anger

## 3 Hotel California 1 desert disgust

## 4 Hotel California 1 desert fear

## 5 Hotel California 1 desert negative

## 6 Hotel California 1 desert sadness

## 7 Hotel California 1 cool positive

## 8 Hotel California 2 smell anger

## 9 Hotel California 2 smell disgust

## 10 Hotel California 2 smell negative

## # ... with 93 more rows

Let's visualize the counts of different emotions and sentiments in the NRC lexicon.

theme\_lyrics <- **function**(aticks = **element\_blank**(),

pgminor = **element\_blank**(),

lt = **element\_blank**(),

lp = "none")

{

**theme**(plot.title = **element\_text**(hjust = 0.5), *#Center the title*

axis.ticks = aticks, *#Set axis ticks to on or off*

panel.grid.minor = pgminor, *#Turn the minor grid lines on or off*

legend.title = lt, *#Turn the legend title on or off*

legend.position = lp) *#Turn the legend on or off*

}

hcsentiment %>%

**group\_by**(sentiment) %>%

**summarise**(word\_count = **n**()) %>%

**ungroup**() %>%

**mutate**(sentiment = **reorder**(sentiment, word\_count)) %>%

**ggplot**(**aes**(sentiment, word\_count, fill = -word\_count)) +

**geom\_col**() +

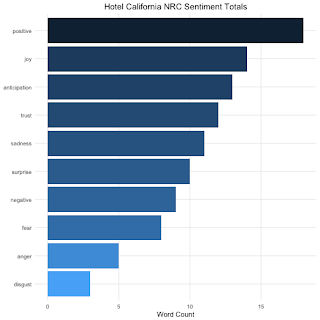
**guides**(fill = FALSE) +

**theme\_minimal**() + **theme\_lyrics**() +

**labs**(x = **NULL**, y = "Word Count") +

**ggtitle**("Hotel California NRC Sentiment Totals") +

**coord\_flip**()

[](https://1.bp.blogspot.com/-S79qwh3n2_U/Wv46zQIK0rI/AAAAAAAALkU/td8fTe2DtfcYeSeF4GGfyTQhXMhrp7ljACLcBGAs/s1600/unnamed-chunk-6-1.png)

Most of the words appear to be positively-valenced. How do the individual words match up?

**library**(ggrepel)

plot\_words <- hcsentiment %>%

**group\_by**(sentiment) %>%

**count**(word, sort = TRUE) %>%

**arrange**(**desc**(n)) %>%

**ungroup**()

plot\_words %>%

**ggplot**(**aes**(word, 1, label = word, fill = sentiment)) +

**geom\_point**(color = "white") +

**geom\_label\_repel**(force = 1, nudge\_y = 0.5,

direction = "y",

box.padding = 0.04,

segment.color = "white",

size = 3) +

**facet\_grid**(~sentiment) +

**theme\_lyrics**() +

**theme**(axis.text.y = **element\_blank**(), axis.line.x = **element\_blank**(),

axis.title.x = **element\_blank**(), axis.text.x = **element\_blank**(),

axis.ticks.x = **element\_blank**(),

panel.grid = **element\_blank**(), panel.background = **element\_blank**(),

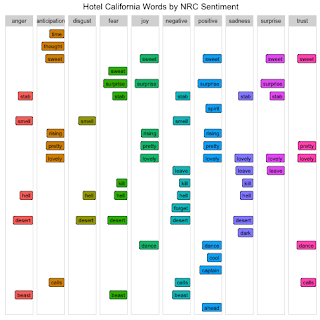
panel.border = **element\_rect**("lightgray", fill = NA),

strip.text.x = **element\_text**(size = 9)) +

**xlab**(**NULL**) + **ylab**(**NULL**) +

**ggtitle**("Hotel California Words by NRC Sentiment") +

**coord\_flip**()

[](https://1.bp.blogspot.com/-EtATflHXVSs/Wv464JMmxQI/AAAAAAAALkc/bCLO-HkdCr8Oau2phcTVN9F89riDTLubQCLcBGAs/s1600/unnamed-chunk-7-1.png)

It looks like some words are being misclassified. For instance, "smell" as in "warm smell of colitas" is being classified as anger, disgust, and negative. But that doesn't explain the overall positive bent being applied to the song. If you listen to the song, you know it's not really a happy song. It starts off somewhat negative - or at least, ambiguous - as the narrator is driving on a dark desert highway. He's tired and having trouble seeing, and notices the Hotel California, a shimmering oasis on the horizon. He stops in and is greated by a "lovely face" in a "lovely place." At the hotel, everyone seems happy: they dance and drink, they have fancy cars, they have pretty "friends."  
  
But the song is in a minor key. Though not always a sign that a song is sad, it is, at the very least, a hint of something ominous, lurking below the surface. Soon, things turn bad for the narrator. The lovely-faced woman tells him they are "just prisoners here of our own device." He tries to run away, but the night man tells him, "You can check out anytime you like, but you can never leave."  
  
The song seems to be a metaphor for something, perhaps fame and excess, which was also the subject of another song on the same album, "Life in the Fast Lane." To someone seeking fame, life is dreary, dark, and deserted. Fame is like an oasis - beautiful and shimmering, an escape. But it isn't all it appears to be. You may be surrounded by beautiful people, but you can only call them "friends." You trust no one. And once you join that lifestyle, you might be able to check out, perhaps through farewell tour(s), but you can never leave that life - people know who you are (or were) and there's no disappearing. And it could be about something even darker that it's hard to escape from, like substance abuse. Whatever meaning you ascribe to the song, the overall message seems to be that things are not as wonderful as they appear on the surface.  
  
So if we follow our own understanding of the song's trajectory, we'd say it starts off somewhat negatively, becomes positive in the middle, then dips back into the negative at the end, when the narrator tries to escape and finds he cannot.  
  
We can chart this, using the line number, which coincides with the location of the word in the song. We'll stick with NRC since it offered the best match, but for simplicity, we'll only pay attention to the positive and negative sentiment codes.

hcsentiment\_index <- tidy\_hc %>%

**inner\_join**(**get\_sentiments**("nrc")%>%

**filter**(sentiment %in% **c**("positive",

"negative"))) %>%

**count**(index = line, sentiment) %>%

**spread**(sentiment, n, fill = 0) %>%

**mutate**(sentiment = positive - negative)

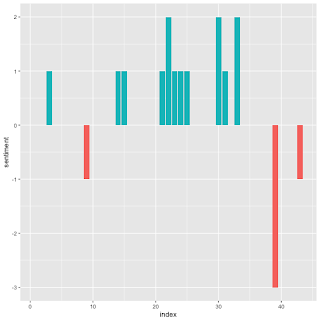
*## Joining, by = "word"*

This gives us a data frame that aggregates sentiment by line. If a line contains more positive than negative words, its overall sentiment is positive, and vice versa. Because not every word in the lyrics has a sentiment, not every line has an associated aggregate sentiment. But it gives us a sort of trajectory over the course of the song. We can visualize this trajectory like this:

hcsentiment\_index %>%

**ggplot**(**aes**(index, sentiment, fill = sentiment > 0)) +

**geom\_col**(show.legend = FALSE)

[](https://4.bp.blogspot.com/-GDjSB5zx3eY/Wv468cfK7kI/AAAAAAAALkg/foyTNo0VmCEk8C5EM69vFDL6YQ5dwlVWQCLcBGAs/s1600/unnamed-chunk-9-1.png)

As the chart shows, the song starts somewhat positive, with a dip soon after into the negative. The middle of the song is positive, as the narrator describes the decadence of the Hotel California. But it turns dark at the end, and stays that way as the guitar solo soars in.

I’ve used the geniusR package on a couple posts, and I’ll be using it again today to answer this question. I’ll be pulling in some additional code. I’ve shared all my code and tried to credit those who helped me write it where I can.

First, we want to pull in the names of Taylor Swift’s 6 studio albums. I found these and their release dates on Wikipedia. While there are only 6 and I could easily copy and paste them to create my data frame, I wanted to pull that data directly from Wikipedia, to write code that could be used on a larger set in the future. I could, with a couple small tweaks.

library(rvest)

## Loading required package: xml2

TSdisc <- 'https://en.wikipedia.org/wiki/Taylor\_Swift\_discography'  
  
disc <- TSdisc %>%  
 read\_html() %>%  
 html\_nodes(xpath = '//\*[@id="mw-content-text"]/div/table[2]') %>%  
 html\_table(fill = TRUE)

Since html() is deprecated, I replaced it with read\_html(), and I got errors if I didn’t add fill = TRUE. The result is a list of 1, with an 8 by 14 data frame within that single list object. I can pull that out as a separate data frame.

TS\_albums <- disc[[1]]

The data frame requires a little cleaning. First up, there are 8 rows, but only 6 albums. Because the Wikipedia table had a double header, the second header was read in as a row of data, so I want to delete that, because I only care about the first two columns anyway. The last row contains a footnote that was included with the table. So I removed those two rows, first and last, and dropped the columns I don’t need. Second, the information I want with release date was in a table cell along with record label and formats (e.g., CD, vinyl). I don’t need those for my purposes, so I’ll only pull out the information I want and drop the rest. Finally, I converted year from character to numeric – this becomes important later on.

library(tidyverse)

TS\_albums<-TS\_albums[2:7,1:2]  
  
TS\_albums <- TS\_albums %>%  
 separate(`Album details`, c("Released","Month","Day","Year"),  
 extra='drop') %>%  
 select(c("Title","Year"))  
  
TS\_albums$Year<-as.numeric(TS\_albums$Year)

I asked geniusR to download lyrics for all 6 albums. (Note: this code may take a couple minutes to run.) It nests all of the individual album data, including lyrics, into a single column, so I just need to unnest that to create a long file, with album title and release year applied to each unnested line.

library(geniusR)  
  
TS\_lyrics <- TS\_albums %>%  
 mutate(tracks = map2("Taylor Swift", Title, genius\_album))

## Joining, by = c("track\_title", "track\_n", "track\_url")  
## Joining, by = c("track\_title", "track\_n", "track\_url")  
## Joining, by = c("track\_title", "track\_n", "track\_url")  
## Joining, by = c("track\_title", "track\_n", "track\_url")  
## Joining, by = c("track\_title", "track\_n", "track\_url")  
## Joining, by = c("track\_title", "track\_n", "track\_url")

TS\_lyrics <- TS\_lyrics %>%  
 unnest(tracks)

Now we’ll tokenize our lyrics data frame, and start doing our word analysis.

library(tidytext)  
  
tidy\_TS <- TS\_lyrics %>%  
 unnest\_tokens(word, lyric) %>%  
 anti\_join(stop\_words)

## Joining, by = "word"

tidy\_TS %>%  
 count(word, sort = TRUE)

## # A tibble: 2,024 x 2  
## word n  
##   
## 1 time 198  
## 2 love 180  
## 3 baby 118  
## 4 ooh 104  
## 5 stay 89  
## 6 night 85  
## 7 wanna 84  
## 8 yeah 83  
## 9 shake 80  
## 10 ey 72  
## # ... with 2,014 more rows

There are a little over 2,000 unique words across TS’s 6 albums. But how have they changed over time? To examine this, I’ll create a dataset that counts word by year (or album, really). Then I’ll use a binomial regression model to look at changes over time, one model per word. In their book, Julia Silge and David Robinson demonstrated how to use binomial regression to examine word use on the authors’ Twitter accounts over time, including an adjustment to the p-values to correct for multiple comparisons. So I based on my code off that.

words\_by\_year <- tidy\_TS %>%  
 count(Year, word) %>%  
 group\_by(Year) %>%  
 mutate(time\_total = sum(n)) %>%  
 group\_by(word) %>%  
 mutate(word\_total = sum(n)) %>%  
 ungroup() %>%  
 rename(count = n) %>%  
 filter(word\_total > 50)  
  
nested\_words <- words\_by\_year %>%  
 nest(-word)  
  
word\_models <- nested\_words %>%  
 mutate(models = map(data, ~glm(cbind(count, time\_total) ~ Year, .,  
 family = "binomial")))

This nests our regression results in a data frame called word\_models. While I could unnest and keep all, I don’t care about every value the GLM gives me. What I care about is the slope for Year, so the filter selects only that slope and the associated p-value. I can then filter to select the significant/marginally significant slopes for plotting (p < 0.1).

library(broom)  
  
slopes <- word\_models %>%  
 unnest(map(models, tidy)) %>%  
 filter(term == "Year") %>%  
 mutate(adjusted.p.value = p.adjust(p.value))  
  
top\_slopes <- slopes%>%  
 filter(adjusted.p.value < 0.1) %>%  
 select(-statistic, -p.value)

This gives me five words that show changes in usage over time: bad, call, dancing, eyes, and yeah. We can plot those five words to see how they’ve changed in usage over her 6 albums. And because I still have my TS\_albums data frame, I can use that information to label the axis of my plot (which is why I needed year to be numeric). I also added a vertical line and annotations to note.

These geoms add reference lines (sometimes called rules) to a plot, either horizontal, vertical, or diagonal (specified by slope and intercept). These are useful for annotating plots.

geom\_abline(

mapping = NULL,

data = NULL,

...,

slope,

intercept,

na.rm = FALSE,

show.legend = NA

)

geom\_hline(

mapping = NULL,

data = NULL,

...,

yintercept,

na.rm = FALSE,

show.legend = NA

)

geom\_vline(

mapping = NULL,

data = NULL,

...,

xintercept,

na.rm = FALSE,

show.legend = NA

)

Arguments

|  |  |
| --- | --- |
| **mapping** | Set of aesthetic mappings created by [aes()](https://ggplot2.tidyverse.org/reference/aes.html) or [aes\_()](https://ggplot2.tidyverse.org/reference/aes_.html). |
| **data** | The data to be displayed in this layer. There are three options:  If NULL, the default, the data is inherited from the plot data as specified in the call to [ggplot()](https://ggplot2.tidyverse.org/reference/ggplot.html).  A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See [fortify()](https://ggplot2.tidyverse.org/reference/fortify.html) for which variables will be created.  A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)). |
| **...** | Other arguments passed on to [layer()](https://ggplot2.tidyverse.org/reference/layer.html). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| **na.rm** | If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed. |
| **show.legend** | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. |
| **xintercept, yintercept, slope, intercept** | Parameters that control the position of the line. If these are set, data, mapping and show.legend are overridden. |

Details

These geoms act slightly differently from other geoms. You can supply the parameters in two ways: either as arguments to the layer function, or via aesthetics. If you use arguments, e.g. geom\_abline(intercept = 0, slope = 1), then behind the scenes the geom makes a new data frame containing just the data you've supplied. That means that the lines will be the same in all facets; if you want them to vary across facets, construct the data frame yourself and use aesthetics.

Unlike most other geoms, these geoms do not inherit aesthetics from the plot default, because they do not understand x and y aesthetics which are commonly set in the plot. They also do not affect the x and y scales.

Aesthetics

These geoms are drawn using with [geom\_line()](https://ggplot2.tidyverse.org/reference/geom_path.html) so support the same aesthetics: alpha, colour, linetype and size. They also each have aesthetics that control the position of the line:

* geom\_vline(): xintercept
* geom\_hline(): yintercept
* geom\_abline(): slope and intercept

See also

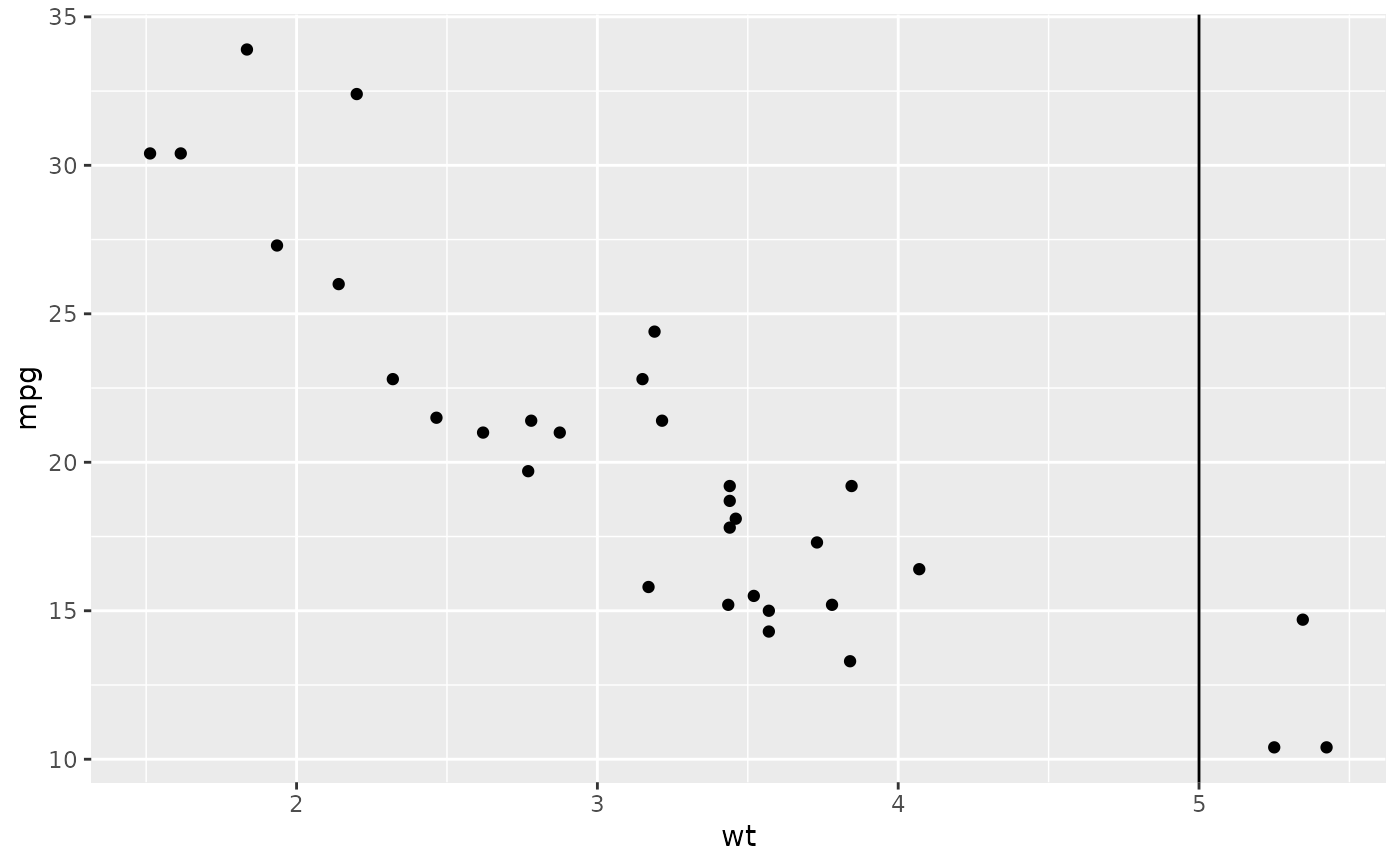
See [geom\_segment()](https://ggplot2.tidyverse.org/reference/geom_segment.html) for a more general approach to adding straight line segments to a plot.

Examples

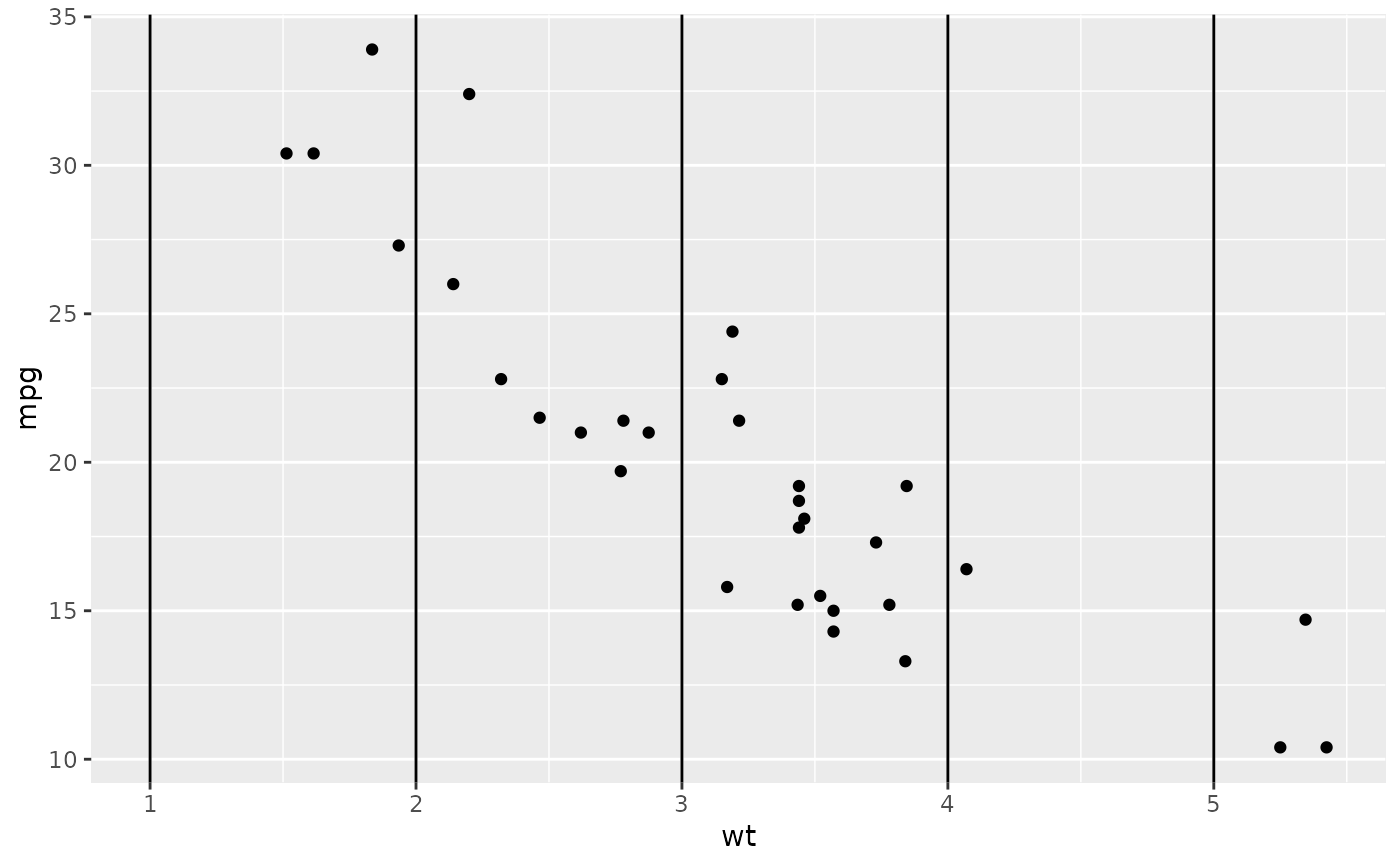
p <- [ggplot](https://ggplot2.tidyverse.org/reference/ggplot.html)(mtcars, [aes](https://ggplot2.tidyverse.org/reference/aes.html)(wt, mpg)) + [geom\_point](https://ggplot2.tidyverse.org/reference/geom_point.html)()

# Fixed values

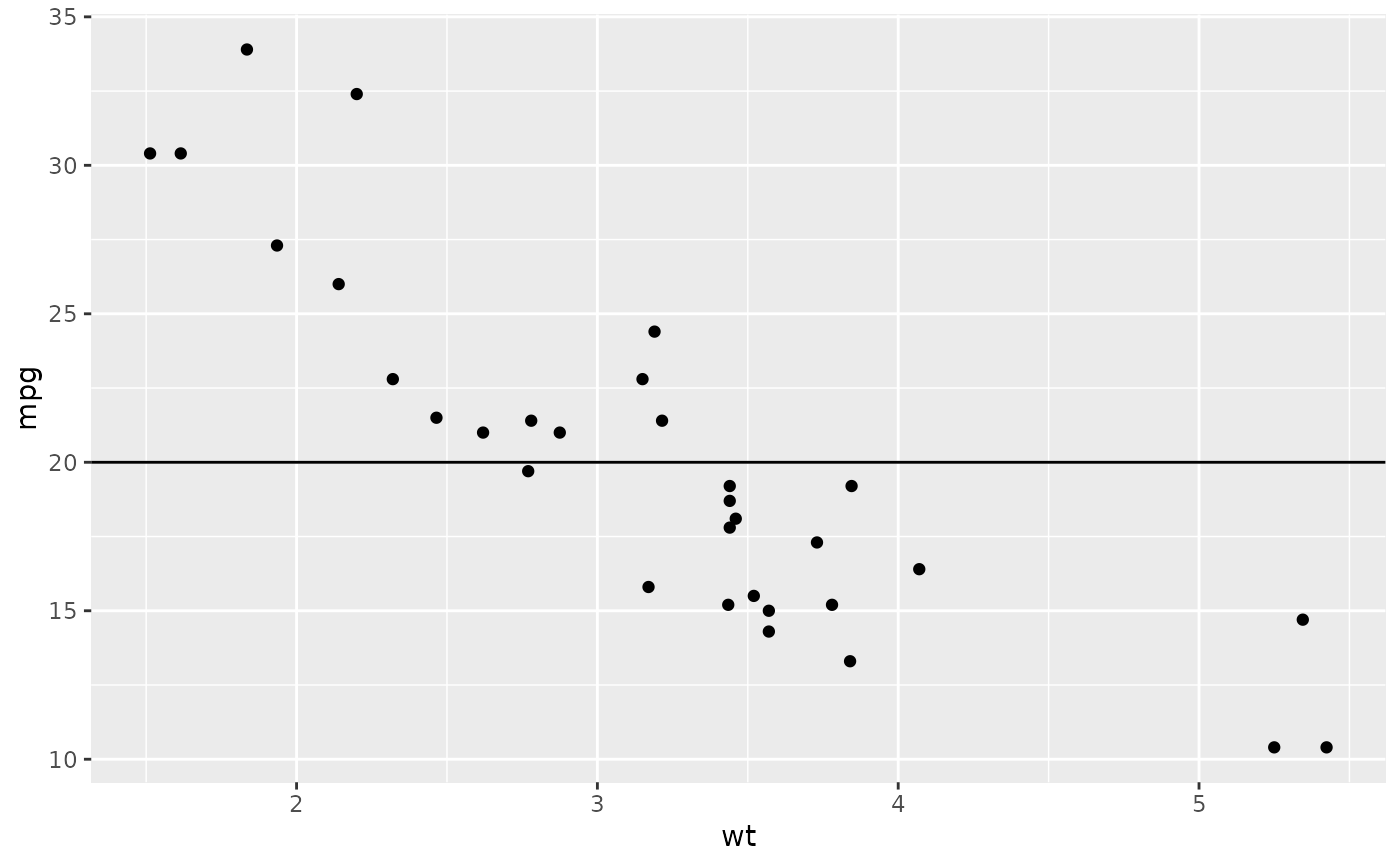
p + geom\_vline(xintercept = 5)



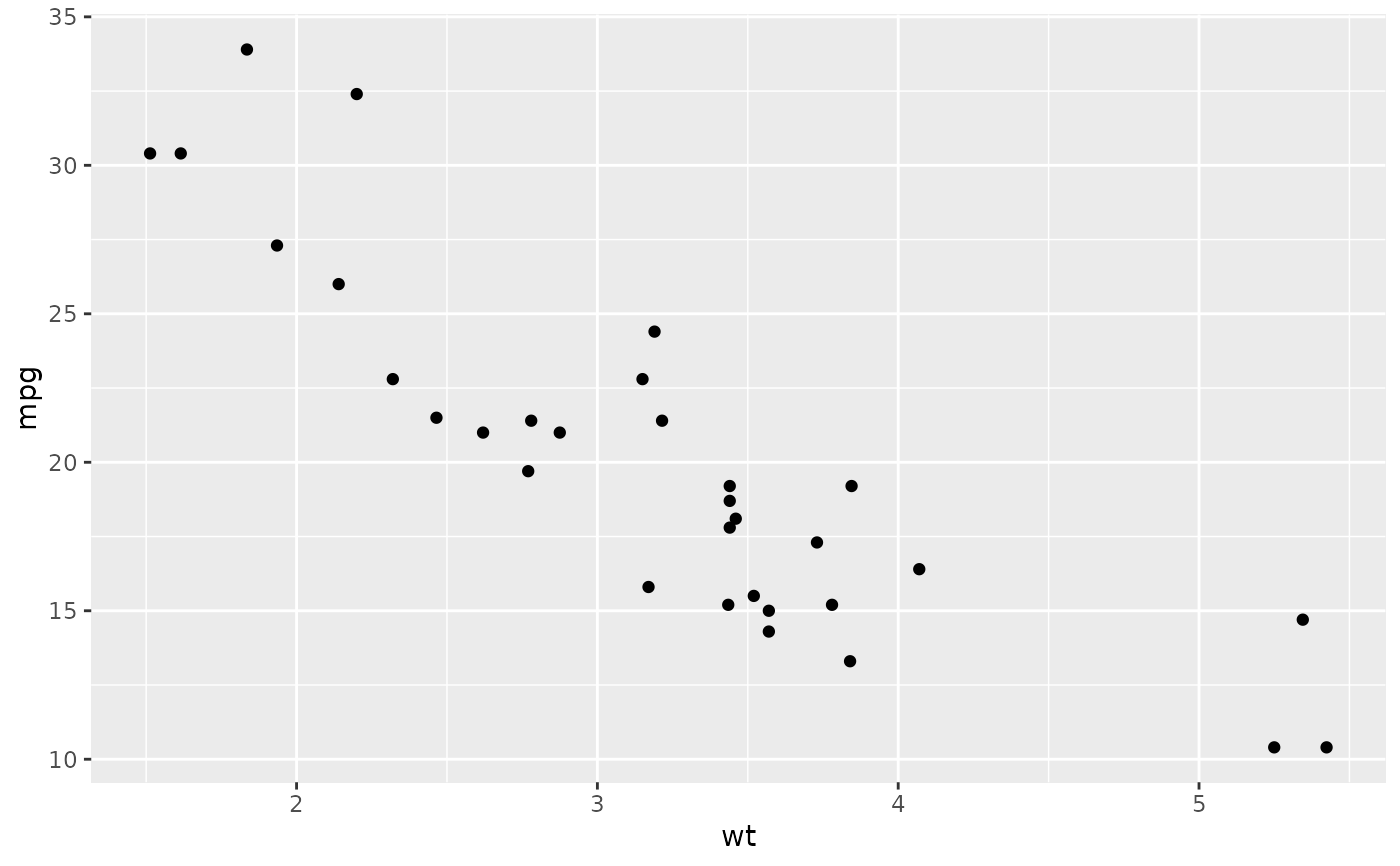
p + geom\_vline(xintercept = 1:5)



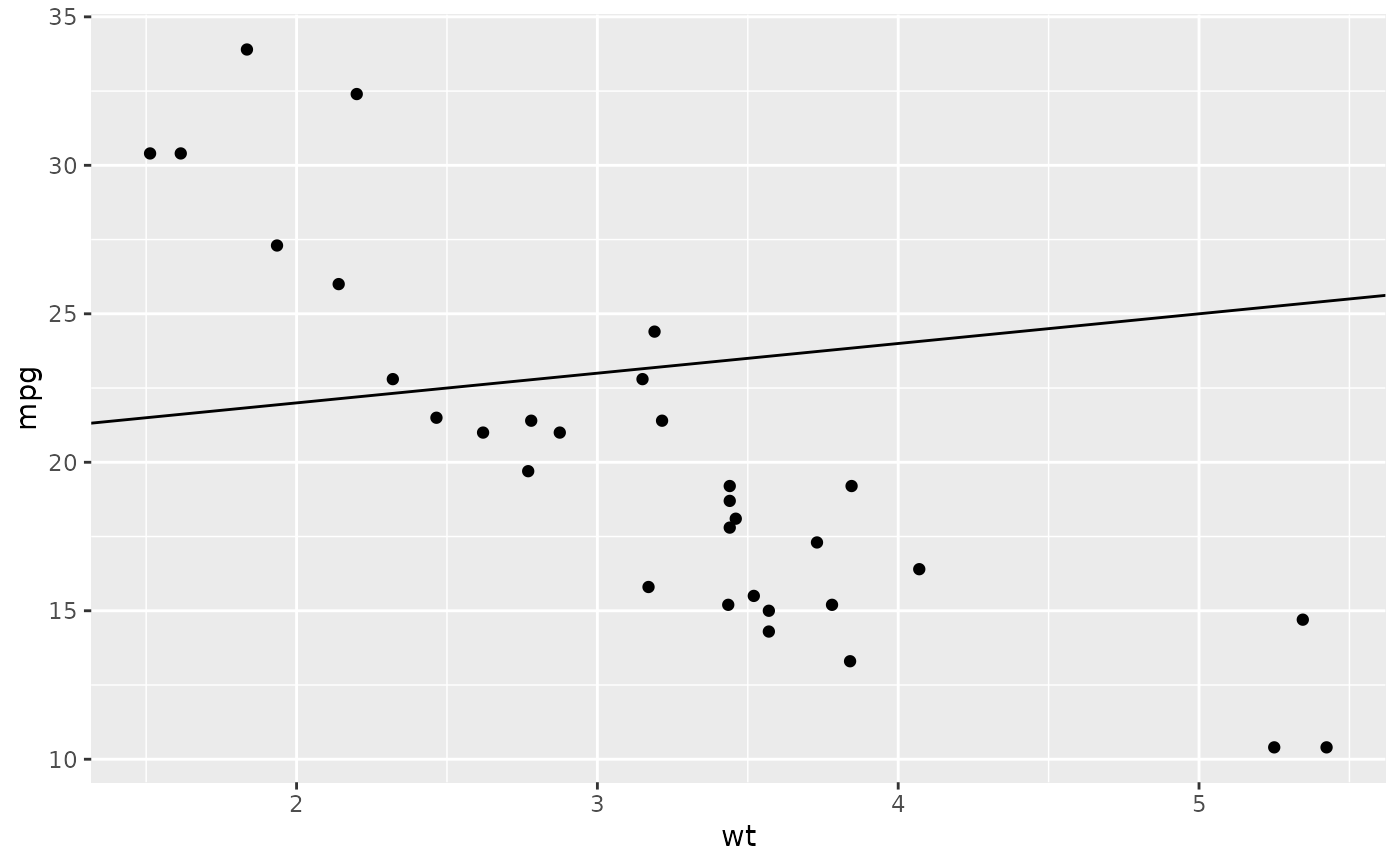
p + geom\_hline(yintercept = 20)



p + geom\_abline() # Can't see it - outside the range of the data



p + geom\_abline(intercept = 20)



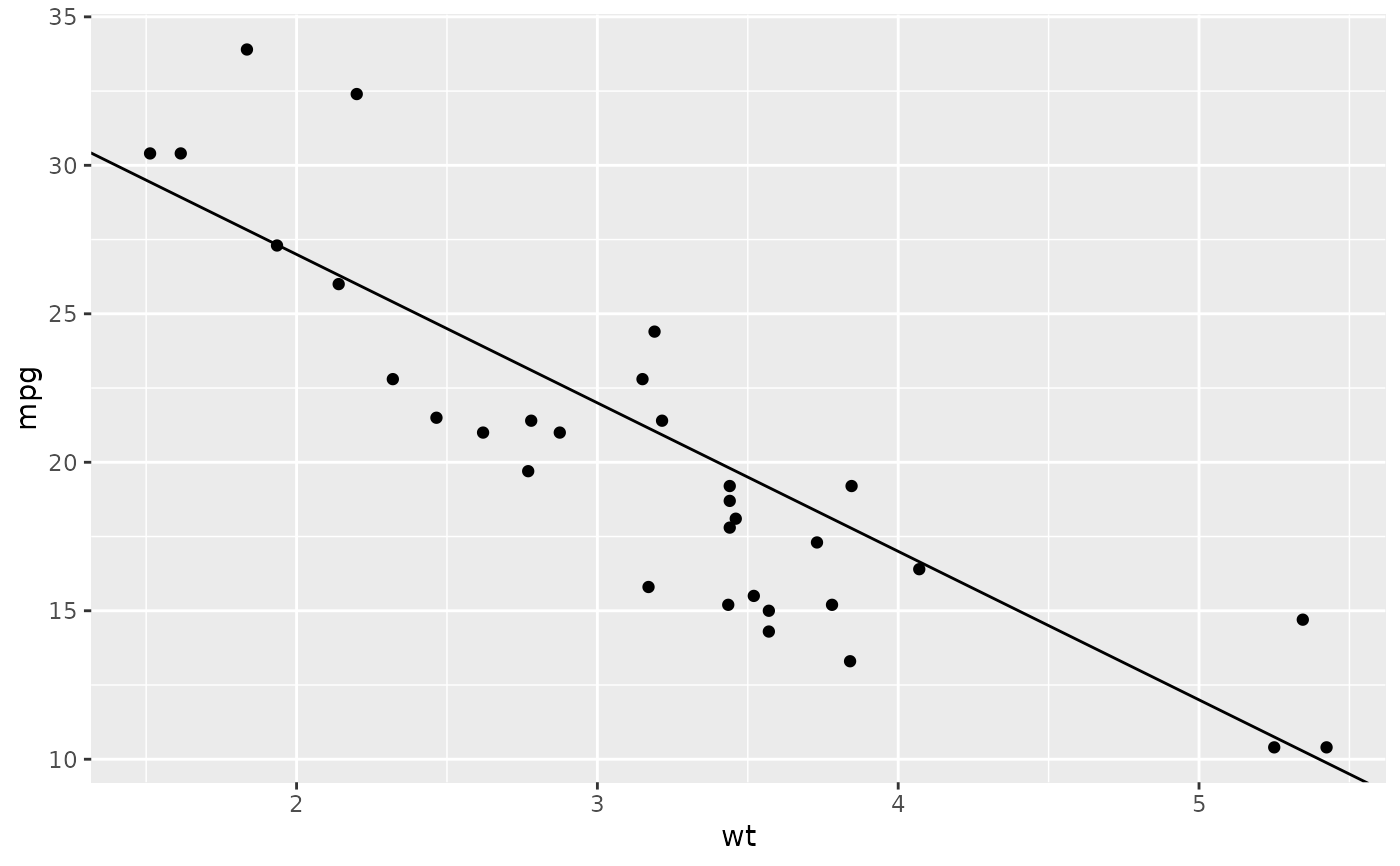
# Calculate slope and intercept of line of best fit

[coef](https://rdrr.io/r/stats/coef.html)([lm](https://rdrr.io/r/stats/lm.html)(mpg ~ wt, data = mtcars))

#> (Intercept) wt

#> 37.285126 -5.344472

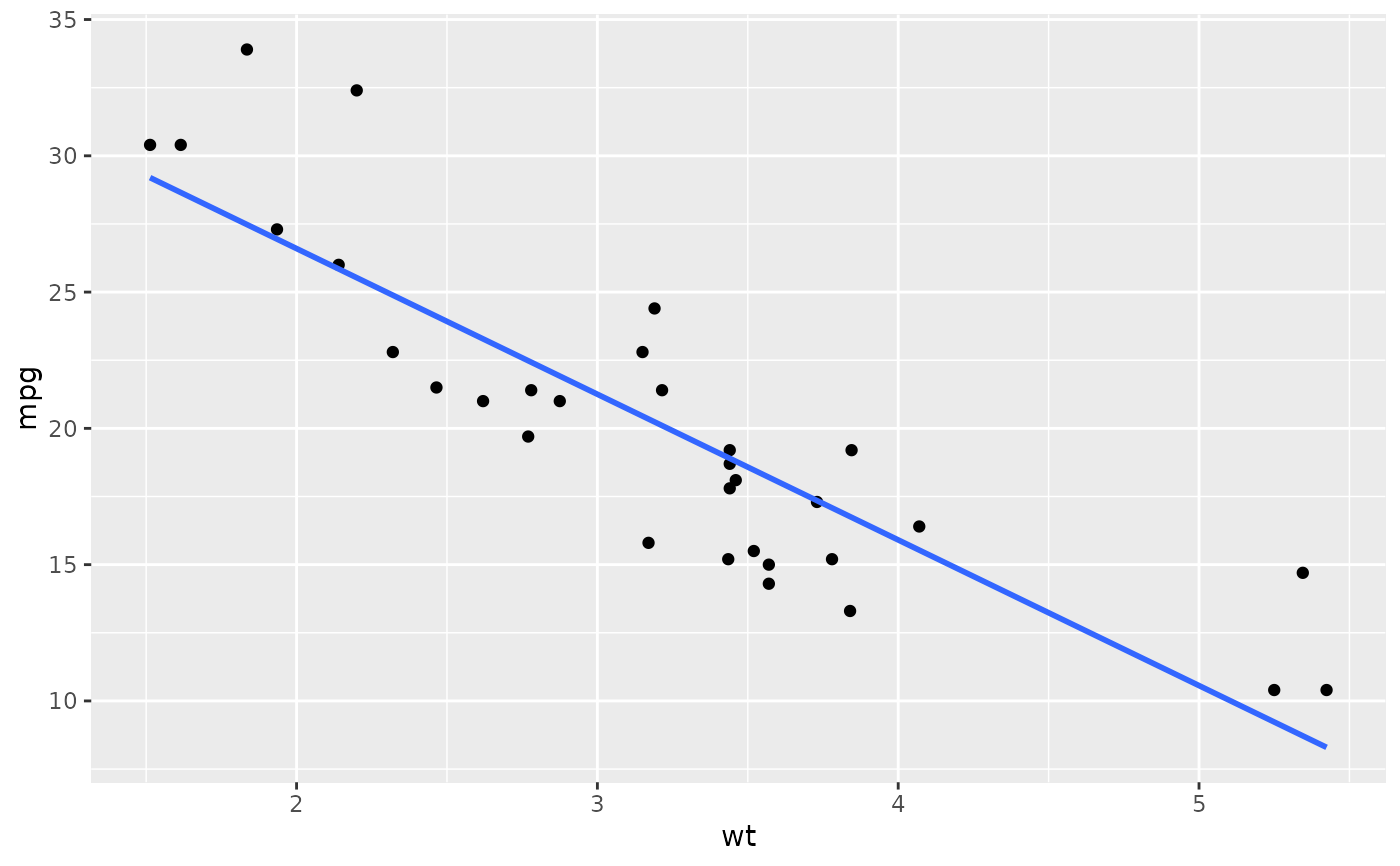
p + geom\_abline(intercept = 37, slope = -5)



# But this is easier to do with geom\_smooth:

p + [geom\_smooth](https://ggplot2.tidyverse.org/reference/geom_smooth.html)(method = "lm", se = FALSE)

#> `geom\_smooth()` using formula 'y ~ x'



# To show different lines in different facets, use aesthetics

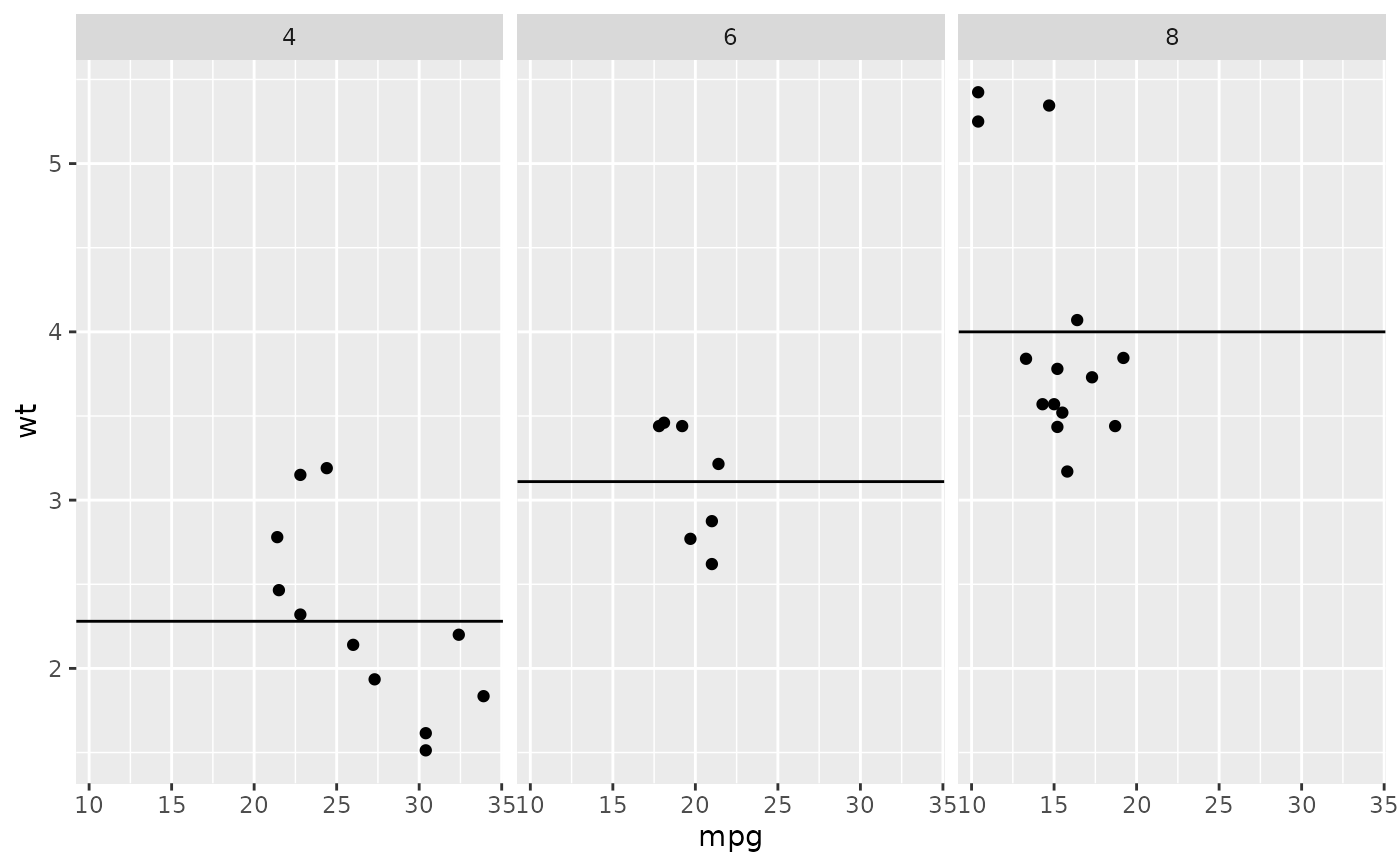
p <- [ggplot](https://ggplot2.tidyverse.org/reference/ggplot.html)(mtcars, [aes](https://ggplot2.tidyverse.org/reference/aes.html)(mpg, wt)) +

[geom\_point](https://ggplot2.tidyverse.org/reference/geom_point.html)() +

[facet\_wrap](https://ggplot2.tidyverse.org/reference/facet_wrap.html)(~ cyl)

mean\_wt <- [data.frame](https://rdrr.io/r/base/data.frame.html)(cyl = [c](https://rdrr.io/r/base/c.html)(4, 6, 8), wt = [c](https://rdrr.io/r/base/c.html)(2.28, 3.11, 4.00))

p + geom\_hline([aes](https://ggplot2.tidyverse.org/reference/aes.html)(yintercept = wt), mean\_wt)



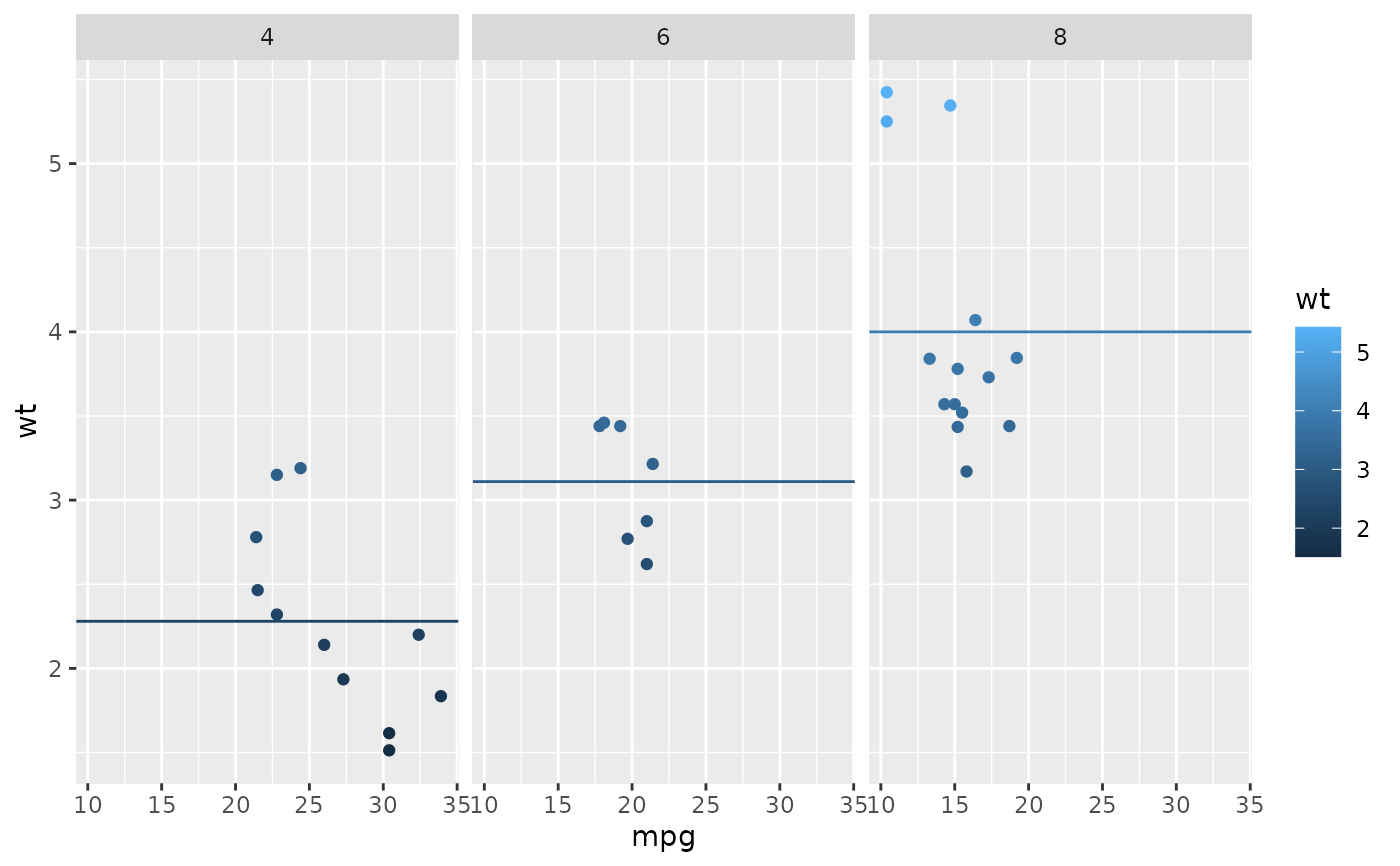
# You can also control other aesthetics

[ggplot](https://ggplot2.tidyverse.org/reference/ggplot.html)(mtcars, [aes](https://ggplot2.tidyverse.org/reference/aes.html)(mpg, wt, colour = wt)) +

[geom\_point](https://ggplot2.tidyverse.org/reference/geom_point.html)() +

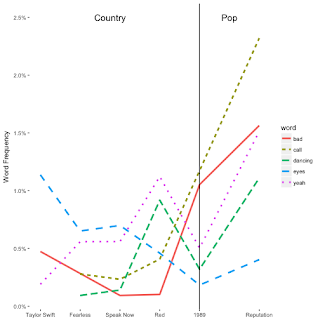
geom\_hline([aes](https://ggplot2.tidyverse.org/reference/aes.html)(yintercept = wt, colour = wt), mean\_wt) +

[facet\_wrap](https://ggplot2.tidyverse.org/reference/facet_wrap.html)(~ cyl)



library(scales)

words\_by\_year %>%  
 inner\_join(top\_slopes, by = "word") %>%  
 ggplot(aes(Year, count/time\_total, color = word, lty = word)) +  
 geom\_line(size = 1.3) +  
 labs(x = NULL, y = "Word Frequency") +  
 scale\_x\_continuous(breaks=TS\_albums$Year,  
 labels=TS\_albums$Title) +  
 scale\_y\_continuous(labels=scales::percent) +  
 geom\_vline(xintercept = 2014) +  
 theme(panel.grid.major = element\_blank(),  
 panel.grid.minor = element\_blank(),  
 panel.background = element\_blank()) +  
 annotate("text", x = c(2009.5,2015.5), y = c(0.025,0.025),  
 label = c("Country", "Pop") , size=5)

[](https://i2.wp.com/4.bp.blogspot.com/-9NIC3WQKzmY/Wv9ogsWH2QI/AAAAAAAALlU/aZ82D4Ifgi4knWDmiiaDjofcu5c209FKQCPcBGAYYCw/s1600/unnamed-chunk-8-1.png?ssl=1)

The biggest change appears to be in the word “call,” which she didn’t use at all in her self-titled album, and used at low rates until “1989” and, especially, “Reputation.” I can ask for a few examples of “call” in her song lyrics, with grep.

Grep – An Example

# NOT RUN {

grep("[a-z]", letters)

txt <- c("arm","foot","lefroo", "bafoobar")

if(length(i <- grep("foo", txt)))

cat("'foo' appears at least once in\n\t", txt, "\n")

i # 2 and 4

txt[i]

## Double all 'a' or 'b's; "\" must be escaped, i.e., 'doubled'

gsub("([ab])", "\\1\_\\1\_", "abc and ABC")

txt <- c("The", "licenses", "for", "most", "software", "are",

"designed", "to", "take", "away", "your", "freedom",

"to", "share", "and", "change", "it.",

"", "By", "contrast,", "the", "GNU", "General", "Public", "License",

"is", "intended", "to", "guarantee", "your", "freedom", "to",

"share", "and", "change", "free", "software", "--",

"to", "make", "sure", "the", "software", "is",

"free", "for", "all", "its", "users")

( i <- grep("[gu]", txt) ) # indices

stopifnot( txt[i] == grep("[gu]", txt, value = TRUE) )

## Note that in locales such as en\_US this includes B as the

## collation order is aAbBcCdEe ...

(ot <- sub("[b-e]",".", txt))

txt[ot != gsub("[b-e]",".", txt)]#- gsub does "global" substitution

txt[gsub("g","#", txt) !=

gsub("g","#", txt, ignore.case = TRUE)] # the "G" words

regexpr("en", txt)

gregexpr("e", txt)

## Using grepl() for filtering

## Find functions with argument names matching "warn":

findArgs <- function(env, pattern) {

nms <- ls(envir = as.environment(env))

nms <- nms[is.na(match(nms, c("F","T")))] # <-- work around "checking hack"

aa <- sapply(nms, function(.) { o <- get(.)

if(is.function(o)) names(formals(o)) })

iw <- sapply(aa, function(a) any(grepl(pattern, a, ignore.case=TRUE)))

aa[iw]

}

findArgs("package:base", "warn")

## trim trailing white space

str <- "Now is the time "

sub(" +$", "", str) ## spaces only

## what is considered 'white space' depends on the locale.

sub("[[:space:]]+$", "", str) ## white space, POSIX-style

## what PCRE considered white space changed in version 8.34: see ?regex

sub("\\s+$", "", str, perl = TRUE) ## PCRE-style white space

## capitalizing

txt <- "a test of capitalizing"

gsub("(\\w)(\\w\*)", "\\U\\1\\L\\2", txt, perl=TRUE)

gsub("\\b(\\w)", "\\U\\1", txt, perl=TRUE)

txt2 <- "useRs may fly into JFK or laGuardia"

gsub("(\\w)(\\w\*)(\\w)", "\\U\\1\\E\\2\\U\\3", txt2, perl=TRUE)

sub("(\\w)(\\w\*)(\\w)", "\\U\\1\\E\\2\\U\\3", txt2, perl=TRUE)

## named capture

notables <- c(" Ben Franklin and Jefferson Davis",

"\tMillard Fillmore")

# name groups 'first' and 'last'

name.rex <- "(?<first>[[:upper:]][[:lower:]]+) (?<last>[[:upper:]][[:lower:]]+)"

(parsed <- regexpr(name.rex, notables, perl = TRUE))

gregexpr(name.rex, notables, perl = TRUE)[[2]]

parse.one <- function(res, result) {

m <- do.call(rbind, lapply(seq\_along(res), function(i) {

if(result[i] == -1) return("")

st <- attr(result, "capture.start")[i, ]

substring(res[i], st, st + attr(result, "capture.length")[i, ] - 1)

}))

colnames(m) <- attr(result, "capture.names")

m

}

parse.one(notables, parsed)

## Decompose a URL into its components.

## Example by LT (http://www.cs.uiowa.edu/~luke/R/regexp.html).

x <- "http://stat.umn.edu:80/xyz"

m <- regexec("^(([^:]+)://)?([^:/]+)(:([0-9]+))?(/.\*)", x)

m

regmatches(x, m)

## Element 3 is the protocol, 4 is the host, 6 is the port, and 7

## is the path. We can use this to make a function for extracting the

## parts of a URL:

URL\_parts <- function(x) {

m <- regexec("^(([^:]+)://)?([^:/]+)(:([0-9]+))?(/.\*)", x)

parts <- do.call(rbind,

lapply(regmatches(x, m), `[`, c(3L, 4L, 6L, 7L)))

colnames(parts) <- c("protocol","host","port","path")

parts

}

URL\_parts(x)

## There is no gregexec() yet, but one can emulate it by running

## regexec() on the regmatches obtained via gregexpr(). E.g.:

pattern <- "([[:alpha:]]+)([[:digit:]]+)"

s <- "Test: A1 BC23 DEF456"

lapply(regmatches(s, gregexpr(pattern, s)),

function(e) regmatches(e, regexec(pattern, e)))

# }

library(expss)

callsubset <- TS\_lyrics[grep("call", TS\_lyrics$lyric),]  
callsubset <- callsubset %>%  
 select(Title, Year, track\_title, lyric)  
set.seed(2012)  
callsubset<-callsubset[sample(nrow(callsubset), 3), ]  
callsubset<-callsubset[order(callsubset$Year),]  
as.etable(callsubset, rownames\_as\_row\_labels = FALSE)

| **Title** | **Year** | **track\_title** | **lyric** |
| --- | --- | --- | --- |
| Speak Now | 2010 | Back to December (Acoustic) | When your birthday passed, and I didn’t call |
| Red | 2012 | All Too Well | And you call me up again just to break me like a promise |
| Reputation | 2017 | Call It What You Want | Call it what you want, call it what you want, call it |

On the other hand, she doesn’t sing about “eyes” as much now that she’s moved from country to pop.

eyessubset <- TS\_lyrics[grep("eyes", TS\_lyrics$lyric),]  
eyessubset <- eyessubset %>%  
 select(Title, Year, track\_title, lyric)  
set.seed(415)  
eyessubset<-eyessubset[sample(nrow(eyessubset), 3), ]  
eyessubset<-eyessubset[order(eyessubset$Year),]  
as.etable(eyessubset, rownames\_as\_row\_labels = FALSE)

| **Title** | **Year** | **track\_title** | **lyric** |
| --- | --- | --- | --- |
| Taylor Swift | 2006 | A Perfectly Good Heart | And realized by the distance in your eyes that I would be the one to fall |
| Speak Now | 2010 | Better Than Revenge | I’m just another thing for you to roll your eyes at, honey |
| Red | 2012 | State of Grace | Just twin fire signs, four blue eyes |

Bet you’ll never listen to Taylor Swift the same way again.

A few notes: I opted to examine any slopes with p < 0.10, which is greater than conventional levels of significance; if you look at the adjusted p-value column, though, you'll see that 4 of the 5 are < 0.05 and one is only slightly greater than 0.05. But I made the somewhat arbitrary choice to include only words used more than 50 times across her 6 albums, so I could get different results by changing that filtering value when I create the words\_by\_time data frame. Feel free to play around and see what you get by using different values!