MDML Assignment 5

Trent Yu

Due: November 14, 2024. Total Points: 100.

Instructions

1. Fill in the `author` field at the top of this .Rmd file and indicate who you worked with, if anyone. Note that you must turn in your own work (see below).
2. Read the rest of these instructions and the "Grading" section.
3. Starting with the "Assignment" section, complete the rest of this R Markdown file from top to bottom. You will provide R code and some written responses in this Markdown file, and you will fill in the function outlines in the provided `assignment5.R` script. Do **not** write any code outside of the functions included in `assignment5.R` and do **not** add additional arguments to any function in that script. Do **not** edit this Markdown file except as directed in the instructions. After you have filled in a function in `assignment5.R`, you can run the corresponding code block in this Markdown file to progress through the assignment. When you finish the assignment, knit this file to obtain a pdf.
4. You must **only** use the packages loaded in this Markdown file in this assignment. Do not install or load any other packages.
5. I strongly suggest that you create and test your code in a "scratch" R script (don't turn that script in) before putting your answers in this file and in `assignment5.R`. That is, you should consider this file and the R script as the place to put your final answers for each question, not where you figure out how to do each question.
6. None of your functions should take more than a few minutes to run. That is, even if you believe your function answers a question correctly, you will lose points if we cannot run it in a reasonable amount of time (under five minutes); if this occurs, you should write more efficient code for your function.
7. Your submission for this assignment should consist only of these three files; do **not** change the file names:
    - `assignment5_workflow.Rmd`
    - `assignment5_workflow.pdf`
    - `assignment5.R`

Tips

1. Start early! In the past, students have found assignments in this class to be quite time-consuming; there is a reason you are given two weeks for each one.
2. Running code on large datasets can take a long time. You can save time by sampling a small subset of rows to confirm your code works before running it on the whole dataset. Similarly, you may find it helpful to use `dplyr::slice_sample()` to select a random subset of points to plot if your plots are taking a long time to load.

Grading

Submit your assignment files on Brightspace. Assignments submitted after 12:30pm on the due date are considered late. Please see the syllabus for the late policy.

You may discuss this assignment with your professor, course assistant, and classmates, but you must turn in your own work. In particular, *you may not directly copy anyone else's code*; that would constitute plagiarism. You may **not** use ChatGPT or similar generative AI tools when doing your homework unless explicitly instructed to do so.

You will be graded on the following: how accurately you followed instructions; correctness, completeness, and clarity of your code and written answers; creativity (when applicable); and quality of visualizations (when applicable). Also, note that passing a test (implemented by a test function in the R scripts) does **not** mean that you have done a problem correctly; **failing** a test means that the autograder will not be able to grade that part of your script.

Assignment

Setup

Create a folder on your computer for this assignment and put this R Markdown file and `assignment5.R` in that folder. If you are going to create a "scratch" R script to work in, put that in the folder as well. Make sure this folder is your working directory (either using the `setwd()` command or by going to Session –> Set Working Directory in the RStudio menu). Next, within this folder, create a subdirectory called `data`.

In both Parts A and B of this assignment, we will be working with a dataset containing over 100,000 plots of stories from (English) Wikipedia. This includes various types of Wikipedia pages with plot sections (e.g., movies, books, video games).

Download and unzip `plots.zip`. Move the resulting `plots` and `titles` files to your `data/` subdirectory.

Part A: Exploring Wikipedia plots [50 points]

Here, we will examine which words characterize the beginning and end of plots. The data were obtained from this website. Uncomment and run the code block below to load the `tidyverse` and `tidytext` packages (install the `tidytext` package first if necessary), and the `plots` and `titles` files.

setwd("/Users/trentyu/Desktop/Messy Data Machine Learning/Assignment 5")

library(tidyverse) library(tidytext) plots <- read_lines('data/plots') plots <- tibble(text = plots) titles <- read_lines('data/titles')

Question A1: Import and process data [15 points]

In `assignment5.R`, complete the `make_plot_words()` function, which takes two arguments as input, `plots` and `titles`. Inside the function, you should:

1. Create a tibble called `plot_text` with three columns:

- `title`: the title of a story.
- `text`: the plot text for the corresponding title. It's fine if each sentence of a plot is a separate row.

- `story_number`: a unique number for the particular story; the first story in `plots` should have a value of 1, the second should have a value of 2, etc.

  Note: you can keep track of when a story ends by looking for "<EOS>" in the text. You may find the `cumsum()` command helpful to count these markers when creating the `story_number` column.

2. Use `tidytext::unnest_tokens()` to split the `text` column into separate words so that each row corresponds to a particular word in a particular plot; call the resulting tibble `plot_words`. Then, add a `word_position` column to `plot_words` that, for each word in each plot, divides the location of the word within the plot by the corresponding plot length (in words). For example, if a plot contains 210 words, and is therefore unnested into 210 rows, the value of `word_position` for the first word is `1/210`, and value of `word_position` for the third word is `3/210`, etc.

Your `make_plot_words()` function should return the `plot_words` tibble, which will have four columns: `title`, `story_number`, `word`, and `word_position`. After completing `make_plot_words()`, uncomment and run the code block below, which saves the output of the function as a tibble called `plot_words`.

setwd("/Users/trentyu/Desktop/Messy Data Machine Learning/Assignment 5/data") source('assignment5.R') plot_words <- make_plot_words(plots = plots, titles = titles) test_make_plot_words()

Question A2: Calculate median positions of words. [10 points]

In `assignment5.R`, complete the `make_interesting_words()` function, which takes the `plot_words` tibble you just generated as input. Inside the function, you should calculate, for each unique word that appears in `plot_words` the median position of the word across all plots (call this `median_position`); and the total number of occurrences of the word across all plots (call this `word_count`). Then, restrict to only unique words that occurred at least 2500 times across the plots.

Your `make_interesting_words()` function should return a tibble called `interesting_words`, which contains the unique words with the 10 highest median positions, and the unique words with the 10 lowest median positions. The `interesting_words` tibble should have 20 rows and three columns: `word`, `median_position`, and `word_count`. After completing `make_interesting_words()`, uncomment and run the code block below.

source('assignment5.R') interesting_words <- make_interesting_words(plot_words = plot_words) test_make_interesting_words()

Question A3: Visualize median positions. [10 points]

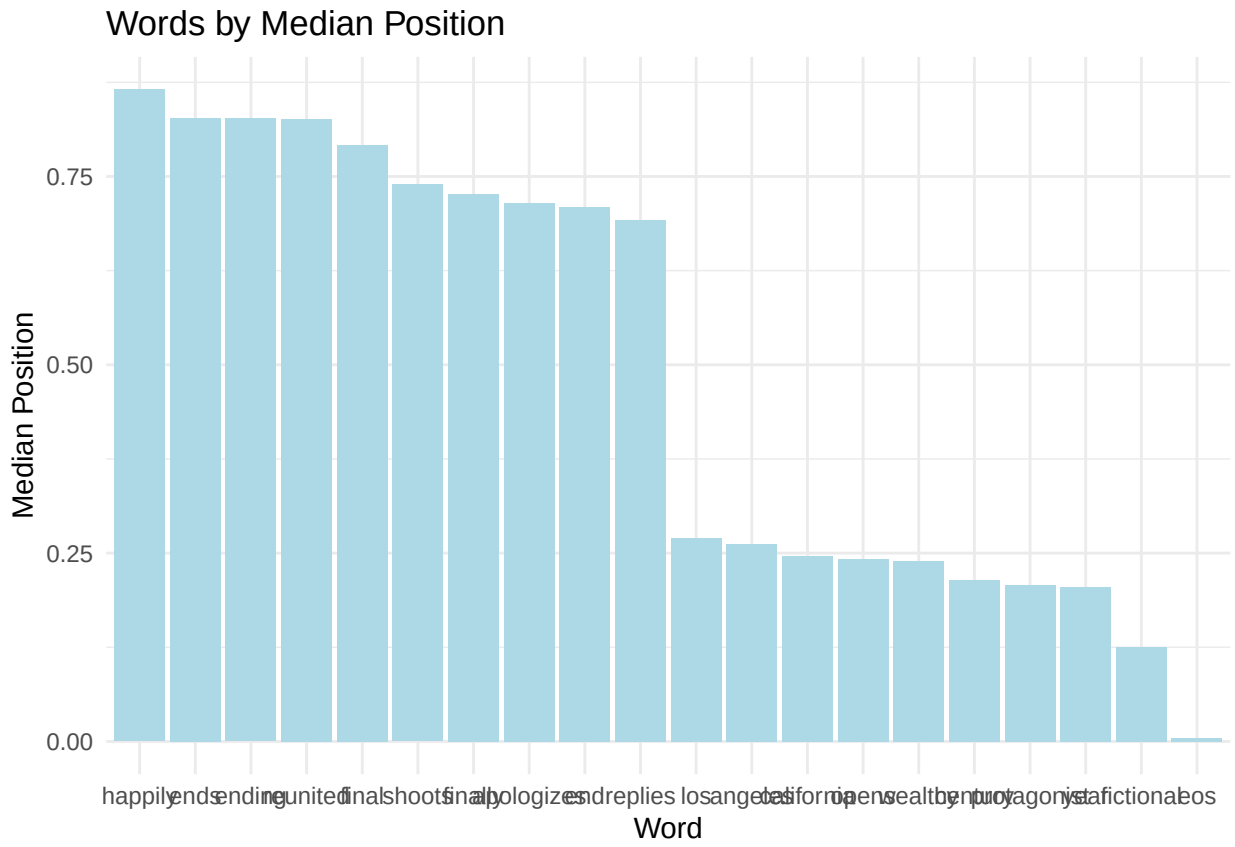
Using the `interesting_words` tibble created in the previous question, create two bar graphs. In the first bar graph, words should appear on the x-axis, and median position on the y-axis, in descending order by median position. In the second bar graph, words should appear on the x-axis, and number of occurrences on the y-axis, in descending order by number of occurrences. Include your code for both plots where indicated, and below, include the plots themselves along with a sentence or two describing what you see.

```
source('assignment5.R')
setwd("/Users/trentyu/Desktop/Messy Data Machine Learning/Assignment 5")

library(tidyverse)
```

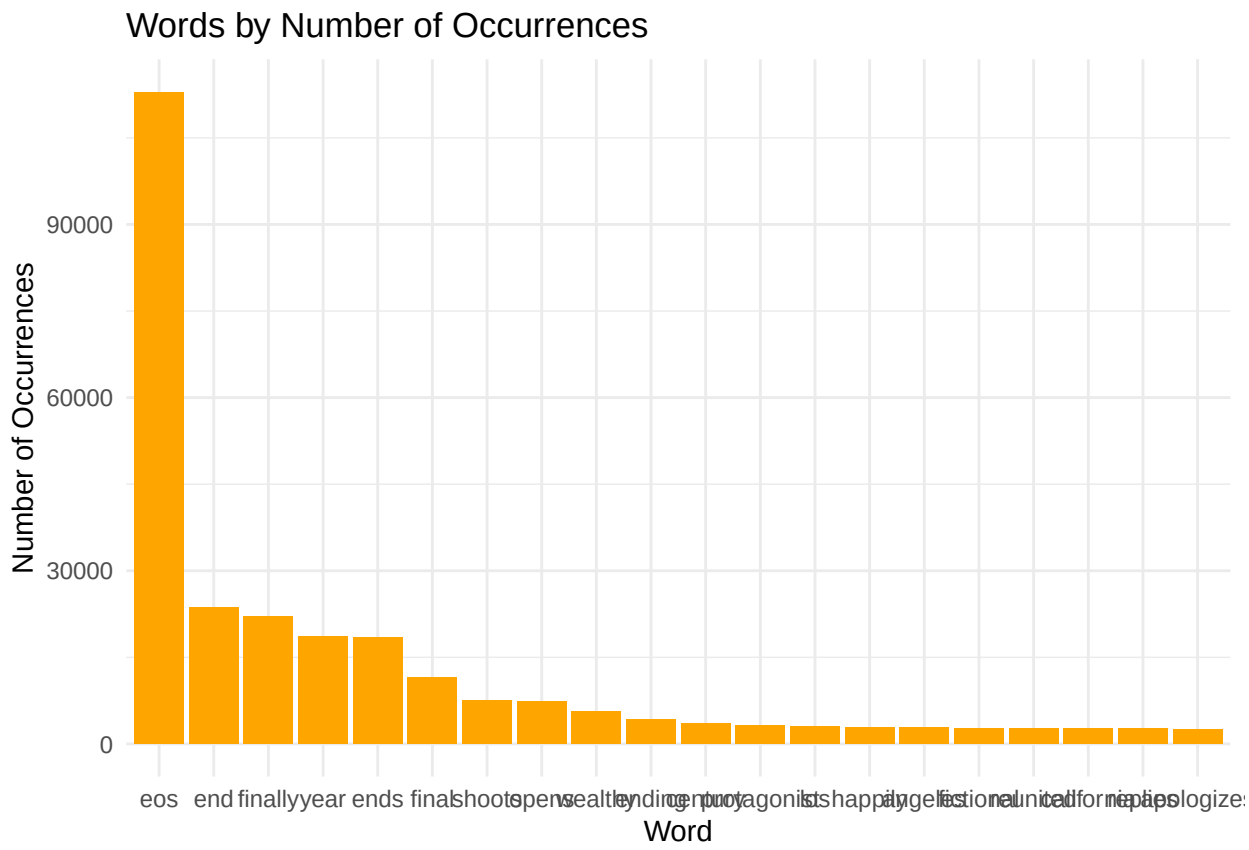
```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
```

```
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(tidytext)
plots <- read_lines('data/plots')
plots <- tibble(text = plots)
titles <- read_lines('data/titles')
plot_words <- make_plot_words(plots = plots, titles = titles)
interesting_words <- make_interesting_words(plot_words = plot_words)

library(dplyr)
library(ggplot2)


ggplot(interesting_words %>% arrange(desc(median_position)), aes(x = reorder(word, -median_position), y
  geom_bar(stat = "identity", fill = "lightblue") +
  labs(x = "Word", y = "Median Position", title = "Words by Median Position") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  theme_minimal()
```



```r
ggplot(interesting_words %>% arrange(desc(word_count)), aes(x = reorder(word, -word_count), y = word_cou
  geom_bar(stat = "identity", fill = "orange") +
  labs(x = "Word", y = "Number of Occurrences", title = "Words by Number of Occurrences") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  theme_minimal()
```



PLOT AND WRITTEN EXPLANATION GOES HERE.

The first plot displays the median position of various words in descending order highlighting words that commonly appear toward the end of texts. The second plot shows the frequency of word occurrences in descending order with some words appearing significantly more often than others.

Question A4: Calculate word deciles [5 points]

Although calculating the median position of frequently occurring words across story plots gives a rough sense of which words generally appear near the beginning of plots vs. the end of plots, we can gain more insight by visualizing the distribution of a word's position in a plot across all plots. Specifically, let's look at word frequency by plot *decile* (i.e., the first 10% of the plot, the second 10% of the plot, etc.)

In `assignment5.R`, complete the `make_word_decile_counts()` function, which takes as input the `plot_words` tibble you generated in Question A1, and returns a tibble called `word_decile_counts`. Each row in `word_decile_counts` should include the count of a given word-decile combination, across all plots. There should be three columns, `word`, `decile` (an integer between 1 and 10), and `n` (the count).

For example, suppose word W occurs five times in three plots: in the first decile of plot A, the first and second decile of plot B, and twice in the third decile of plot C. Then `word_decile_counts` would have three

rows corresponding to word W: The first row would have `decile`= 1 and `n`= 2, the second row would have `decile`= 2 and `n`= 1, and the third row would have `decile`= 3 and `n`= 2. (Note that you *don't* need to have rows for word-decile combinations that don't appear in your data.)

After completing `make_word_decile_counts()`, uncomment and run the code block below.

source('assignment5.R') word_decile_counts <- make_word_decile_counts(plot_words = plot_words) test_make_word_decile_counts()

Question A5: Visualize word decile distributions [10 points]

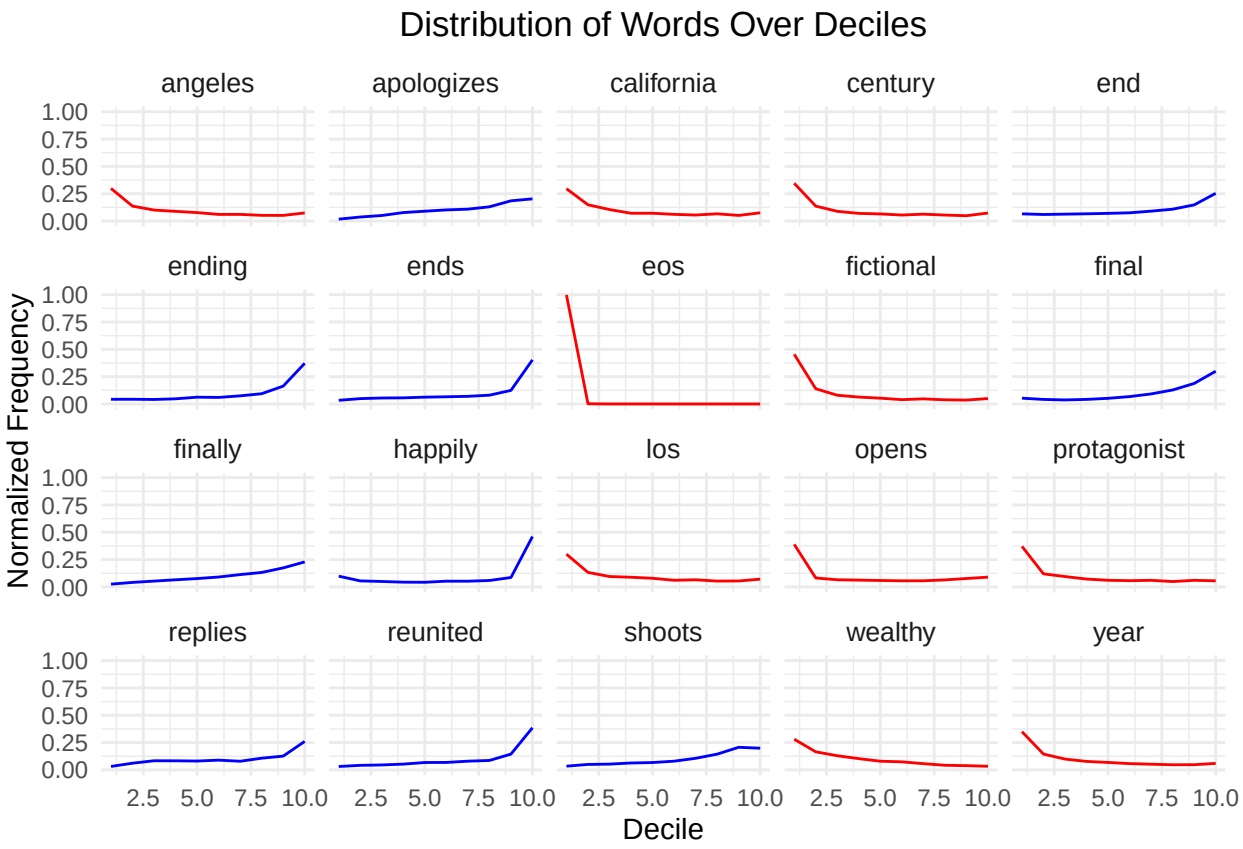
Perform an inner join of `interesting_words` (i.e., the output of the `make_interesting_words()` function from Question A2) and `word_decile_counts` (i.e., the output of the `make_word_decile_counts()` function from Question A4), keeping track of whether the words in `interesting_words` are in the group with highest median position, or the group with lowest median position.

Use this tibble to create a collection of plots, one for each word, with decile on the x-axis, and the frequency of the word on the y-axis, normalized by the total number of occurrences of the word (i.e., you are visualizing the distribution of each word over deciles). Your plot should have 20 facets (individual subplots), one for each word; the lines for the 10 words with the largest median positions should all be one color (e.g., blue), and the lines for the 10 words with the smallest median positions should all be another color (e.g. red).

Include your code for the plots where indicated, and below, include the plots themselves along with a few sentences describing what you see.

```
library(dplyr)
library(ggplot2)


source('assignment5.R')
setwd("/Users/trentyu/Desktop/Messy Data Machine Learning/Assignment 5")

library(tidyverse)
library(tidytext)
plots <- read_lines('data/plots')
plots <- tibble(text = plots)
titles <- read_lines('data/titles')
plot_words <- make_plot_words(plots = plots, titles = titles)
interesting_words <- make_interesting_words(plot_words = plot_words)
word_decile_counts <- make_word_decile_counts(plot_words = plot_words)

interesting_words <- interesting_words %>%
  mutate(color = ifelse(rank(desc(median_position)) <= 10, "blue", "red"))

merged_data <- word_decile_counts %>%
  inner_join(interesting_words, by = "word")

merged_data <- merged_data %>%
  mutate(normalized_frequency = n / word_count)

ggplot(merged_data, aes(x = decile, y = normalized_frequency, color = color, group = word)) +
  geom_line() +
  scale_color_identity() +
  labs(x = "Decile", y = "Normalized Frequency", title = "Distribution of Words Over Deciles") +
  facet_wrap(~ word, ncol = 5) +
```

```
  theme_minimal() +
  theme(legend.position = "none",
        strip.text = element_text(size = 10),
        plot.title = element_text(hjust = 0.5))
```



PLOT AND WRITTEN EXPLANATION GOES HERE.

The plots show the distribution of each word's occurrence across deciles, with blue lines representing words in the top 10 highest median positions and red lines representing words in the bottom 10. The normalized frequency on the y-axis indicates how often each word appears in each decile, revealing different trends, such as "ends" and "happily" peaking in later deciles, while "eos" and "fictional" are more frequent in earlier deciles.

Part B: Surprise endings [50 points]

Suppose you are an author who wants to write a surprise ending for your story. One challenge you face is that stories generally have similar arcs (e.g. https://en.wikipedia.org/wiki/The_Seven_Basic_Plots). In this question, you'll build a classifier that predicts whether an ending will be happy or sad (based on the sentiment of words) using all the text up to the ending. After you write the first part of the plot, you can then use the classifier to predict the *least likely* ending type for your plot, which you can use to sketch a new, surprising ending.

Question B1: Introduction to sentiment analysis. [10 points]

In this question, you will learn how a sentiment lexicon can be used to determine the overall sentiment of parts of a text.

Read Chapter 2 of Text Mining with R. Then, in `assignment5.R`, complete the `make_sentiments()` function, which takes two inputs: the `plot_words` tibble generated by the function `make_plot_words()` in Question A1, and `lexicon`, a dictionary of words and associated sentiment values. You will pass the `bing` lexicon from the `tidytext` package to the lexicon argument of `make_sentiments()`. Inside the `make_sentiments()` function, do the following:

1. Add a `decile` column to `plot_words`, as in Question A4, which records the decile (an integer between 1 and 10) of the plot position in which each word appears.
2. Edit the `bing` lexicon to remove all rows with words that have multiple sentiment values (there should be three such words, corresponding to six rows).
3. Add a column called `sentiment` to `plot_words`, assigning to each word the corresponding sentiment value from the `bing` lexicon (if a word doesn't have a sentiment value in the `bing` lexicon, it's fine to drop it).
4. Recode `positive` sentiment values as `1`, and `negative` sentiment values as `0`; then, grouping by `story_number` and `decile`, compute the mean sentiment for each decile of each plot that has at least one word with a sentiment value.

Your `make_sentiments()` function should return a tibble called `plots`, which has four columns: `title`, `story_number`, `decile`, and `mean_sentiment`, a number between 0 and 1 encoding the average sentiment for that decile in that story. You should ignore `NA` values when computing `mean_sentiment` for each decile of each plot. (Don't worry about missing deciles for some stories; you'll deal with that issue next.)

After completing `make_sentiments()`, uncomment and run the code block below, which saves the output of the function as a tibble called `plots`.

source('assignment5.R') lexicon <- tidytext::get_sentiments('bing') plots <- make_sentiments(plot_words = plot_words, lexicon = lexicon) test_make_sentiments()

Question B2: Process data and fit a logistic regression. [25 points]

Now we need to construct a "ground truth" label indicating whether each plot has a sad ending or happy ending. We also need to handle missing values.

In `assignment5.R`, complete the `process_plots_for_modeling()` function, which takes the `plots` tibble generated in Question B1 as input. Inside the `process_plots_for_modeling()` function, do the following:

1. Use `dplyr::pivot_wider` to reshape `plots` into a dataset where each row represents exactly one plot, and where we can use sentiment values from the first nine deciles to predict the ending. This tibble should have 12 columns: `story_number`, `title`, and `decile_1`, `decile_2`, ..., `decile_10`, where the value in each `decile_j` column is the mean sentiment for that decile and that story.
2. Quite a few plot-deciles will have missing `mean_sentiment` values for various deciles. Drop all plots that have a missing value for `decile_10` (since we'll use this feature to construct our outcome variable), then drop all plots that additionally have 4 or more missing values anywhere in deciles 1-9.

   Then, for each remaining missing `mean_sentiment` value in deciles 1-9, replace it with the average sentiment from the other deciles in the same plot (except decile 10), ignoring `NA` values. For example, if plot P has `mean_sentiment` 0.5 for deciles 1 and 2, `mean_sentiment` 0 for deciles 3-7, and a missing

`mean_sentiment` value for deciles 8 and 9, then you should impute a `mean_sentiment` value of 1/7 for deciles 8 and 9. After this step, you should have no missing values in any row of your dataset.[1]

(You may find the `base::rowSums()` and `base::rowMeans()` commands helpful for this question.)

3. Add a column called `is_happy_ending` that records if the 10th decile of the plot has `mean_sentiment` greater than or equal to 0.5 (we'll say this was a happy ending, and put the value in `is_happy_ending` as 1), or less than 0.5 (we'll say this was a sad ending, and put the value in `is_happy_ending` as 0). Then, drop the `decile_10` column.

After completing `process_plots_for_modeling()`, uncomment and run the code block below, which saves the output of the function as a tibble called `modeling_data` and fits a logistic regression model, predicting `is_happy_ending` as a function of the average sentiments of the first nine deciles.

source('assignment5.R') modeling_data <- process_plots_for_modeling(plots = plots) ending_model <- glm(is_happy_ending ~ ., data = select(modeling_data, -story_number, -title), family='binomial') test_process_plots_for_modeling()

Question B3: Write a surprising plot ending. [15 points]

Now, you'll find a plot which is not in this dataset, and use the classifier you just build to help write an "unpredictable" ending.

1. Find a plot (from Wikipedia, or somewhere else) that is not in the `plots` dataset and hardcode it into the code block below. Include the title of your plot where indicated in the `make_plot_words()` functions in the code block.

2. Using the `make_plot_words()`, `make_sentiments()`, and `process_plots_for_modeling()` functions from previous questions, process your plot so that we can obtain predictions for this story, i.e., unnest the tokens, join with the `bing` sentiment lexicon, and calculate the average sentiment for each decile. (Note: if your plot has too many missing sentiment values, you may have to choose a different plot).

3. Using the `ending_model` classifier from Question B2, predict whether your story has a happy ending or a sad ending using the mean sentiment of the first nine deciles. Report the predicted probability of a happy ending below. Upon reading the end of the plot, do you agree with the classifier?

4. Rewrite the ending of your story to be "surprising" (your rewrite must be in coherent English!). That is, rewrite the last decile of the plot so that it has the opposite type of ending; if your plot was predicted to have a "happy" ending, your rewritten ending should have a mean sentiment lower than 0.3, and if your plot was predicted to have a "sad" ending, your rewritten ending should have a mean sentiment greater than 0.7. Include your rewritten ending in the code block where indicated.

5. Does the rewritten ending *need* to be coherent English in order to be "happy" or "sad" according to our definition? That is, do the written sentences have to actually make sense? Explain in a sentence or two below.

6. Can you think of ways to more accurately capture (using the plot text) whether the ending to a story is "happy" or "sad"? Write a few sentences below.

```
source('assignment5.R')

library(tidytext)
lexicon <- tidytext::get_sentiments('bing')
plots <- make_sentiments(plot_words = plot_words, lexicon = lexicon)
```

[1] Note that this kind of imputation has a number of significant drawbacks; don't blindly use it whenever you are dealing with missing data!

```
## `summarise()` has grouped output by 'title', 'story_number'. You can override
## using the `.groups` argument.
```

```
modeling_data <- process_plots_for_modeling(plots = plots)
ending_model <- glm(is_happy_ending ~ .,  data = select(modeling_data, -story_number, -title), family='
```

```
my_plot <- tibble(text = c(
  "In the magical animated land of Andalasia, a young maiden named Giselle dreams of true love.",
  "Giselle imagines meeting her perfect prince, and her animal friends help her create a statue of him.
  "Meanwhile, Prince Edward hears Giselle's singing voice and is immediately captivated.",
  "He rushes to find her, and they fall in love at first sight.",
  "They decide to marry the next day, much to the dismay of Edward's stepmother, Queen Narissa.",
  "Queen Narissa fears losing her throne if Edward marries and becomes king.",
  "On Giselle and Edward's wedding day, Narissa disguises herself as an old hag and lures Giselle to a
  "Narissa pushes Giselle into the well, which transports her to a world 'where there are no happily ev
  "Giselle emerges from a manhole in modern-day New York City, confused and frightened.",
  "She wanders around Manhattan in her bridal gown, feeling completely out of place.",
  "Robert, a cynical divorce lawyer, and his young daughter, Morgan, come across Giselle.",
  "Though skeptical, Robert lets Giselle stay in his apartment for the night.",
  "Giselle tries to adjust to New York life but is bewildered by its fast pace and lack of kindness.",
  "Meanwhile, Prince Edward jumps into the well to rescue Giselle and ends up in New York as well.",
  "Accompanied by his sidekick, Pip the chipmunk, Edward begins his search for Giselle.",
  "Giselle uses her magical singing abilities to clean Robert's apartment with the help of local animal
  "Robert's girlfriend, Nancy, is shocked when she finds Giselle in his apartment and becomes jealous."
  "Over time, Giselle and Robert begin to bond, and she learns about the complexities of real relations
  "Giselle realizes that love isn't as simple as she thought and feels confused about her feelings for
  "Prince Edward finally finds Giselle, but she seems hesitant to leave with him.",
  "Queen Narissa's minion, Nathaniel, follows Edward to New York to prevent him from finding Giselle.",
  "Nathaniel tries to stop Giselle and Edward from reuniting by tempting her with poisoned apples.",
  "Giselle takes a bite from one of the apples and begins to feel unwell.",
  "At a ball, Narissa appears and, seeing her plan unravel, transforms into a dragon to capture Giselle
  "Robert fights Narissa to save Giselle, showing his newfound feelings for her.",
  "Narissa, in dragon form, kidnaps Robert, but Giselle bravely goes after them.",
  "With help from Pip, Giselle defeats Narissa, who falls from a great height.",
  "Giselle realizes that she loves Robert, and they share a kiss that breaks the spell.",
  "Prince Edward returns to Andalasia with Nancy, where they find their own happy ending.",
  "Giselle stays in New York with Robert and Morgan, living happily ever after in a blend of fairy tale
))

my_plot_words <- make_plot_words(plots = my_plot, titles = "enchanted")
my_plot_sentiments <- make_sentiments(my_plot_words, lexicon = lexicon)
```

```
## `summarise()` has grouped output by 'title', 'story_number'. You can override
## using the `.groups` argument.
```

```
my_modeling_data <- process_plots_for_modeling(plots = my_plot_sentiments)
predict(ending_model, newdata = my_modeling_data, type = 'response')
```

```
##         1
## 0.5595921
```

```r
source('assignment5.R')

library(tidytext)
lexicon <- tidytext::get_sentiments('bing')
plots <- make_sentiments(plot_words = plot_words, lexicon = lexicon)
```

```
## `summarise()` has grouped output by 'title', 'story_number'. You can override
## using the `.groups` argument.
```

```r
modeling_data <- process_plots_for_modeling(plots = plots)
ending_model <- glm(is_happy_ending ~ .,  data = select(modeling_data, -story_number, -title), family='

my_rewritten_plot <- tibble(text = c(
  "In the magical land of Andalasia, a young maiden named Giselle dreams of finding true love, but is o
  "Giselle spends her days singing to herself, hoping that one day a prince will come and take her away
  "One day, Prince Edward hears Giselle's voice and rushes to find her, enchanted by her innocence.",
  "They fall in love at first sight and decide to marry the next day, but Queen Narissa, Edward's stepm
  "Narissa fears that if Edward marries, she will lose her throne, so she disguises herself as an old ha
  "On her wedding day, Narissa pushes Giselle into a well, sending her to a world where fairy tales do n
  "Giselle awakens in the gritty streets of New York City, feeling lost, scared, and utterly alone in a
  "Wandering through the unfamiliar city, Giselle is shocked by the harshness of the real world, where n
  "Eventually, a cynical lawyer named Robert and his young daughter, Morgan, find Giselle and reluctantl
  "Giselle tries to adjust to this harsh new world, but she feels the weight of her shattered dreams mo
  "Meanwhile, Prince Edward arrives in New York to rescue Giselle, but quickly becomes lost in the confu
  "Giselle, desperate to feel some sense of home, uses her singing to call the local animals, but the ci
  "Robert's girlfriend, Nancy, becomes jealous when she finds Giselle in his apartment, leading to argum
  "As the days pass, Giselle starts to fall for Robert, though she's conflicted and heartbroken over her
  "When Edward finally finds Giselle, he is dismayed to see that her spirit has changed, dulled by the l
  "Queen Narissa's minion, Nathaniel, arrives in New York to prevent Giselle and Edward from finding hap
  "Nathaniel tempts Giselle with a poisoned apple, promising her that it will help her forget her sadnes
  "In a moment of despair, Giselle takes a bite of the apple, hoping it will numb the pain she feels fr
  "As Giselle grows weaker, Robert realizes that he has feelings for her, but his confession comes too l
  "At a masquerade ball, Narissa confronts Giselle, mocking her belief in fairy tales and happy endings
  "Realizing that she has no place in either world, Giselle's spirit breaks, and she accepts her fate."
  "Narissa, seeing her victory, transforms into a dragon and threatens to destroy everything Giselle ev
  "Robert tries to protect Giselle, but Narissa's strength is too great, and he is gravely injured in th
  "With no one left to save her, Giselle watches in despair as her dreams of love and happiness crumble
  "Alone and heartbroken, Giselle wanders the streets of New York, feeling abandoned in a world that no
  "Days turn into weeks, and Giselle becomes a shadow of her former self, her once-bright spirit dulled
  "The fairy tale she had dreamed of all her life has ended in tragedy, leaving her isolated and forgott
  "As she disappears into the endless streets, no one remembers the maiden from Andalasia, and her story
  "Giselle's dreams are lost forever, and the magic she once believed in is gone, leaving her to live ou
))

my_rewritten_plot_words <- make_plot_words(plots = my_rewritten_plot, titles = "Enchanted - Sad Version
my_rewritten_plot_sentiments <- make_sentiments(my_rewritten_plot_words, lexicon = lexicon)
```

```
## `summarise()` has grouped output by 'title', 'story_number'. You can override
## using the `.groups` argument.
```

```r
my_rewritten_plot_sentiments %>% filter(decile == 10)
```

```
## # A tibble: 1 x 4
##   title                    story_number decile mean_sentiment
##   <chr>                           <dbl>  <int>          <dbl>
## 1 Enchanted - Sad Version             1     10          0.286
```

WRITTEN EXPLANATION GOES HERE.

The rewritten ending does not necessarily need to be coherent English to be classified as "happy" or "sad" according to the sentiment model. Sentiment analysis relies on individual words or phrases rather than the overall coherence of sentences. As a result, using strongly positive or negative words can influence the sentiment score.

To better catch a "happy" or "sad" ending, we could analyze sentiment trends throughout the plot and look for key thematic words in the final deciles. Additionally, using NLP models trained on narrative structures could capture emotional outcomes more accurately than simple sentiment scores.