analysis_assignment3

sofiascharf

2026-01-08

Load packages and data

```
pacman::p_load(tidyverse, ggplot2, car, lme4)

recipes <- read_csv("recipes.csv")
```

```
## Rows: 62126 Columns: 8
## ── Column specification ────────────────────────────────────────────────────
## Delimiter: ","
## chr (6): recipe_title, category, subcategory, description, ingredients, dire...
## dbl (2): num_ingredients, num_steps
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# print dimensions

print(dim(recipes))
```

```
## [1] 62126     8
```

Data cleaning

First we subset for air fryer and oven:

```
# 1: subset for air fryer and oven
recipes_cleaned <- recipes %>%
  mutate(search_text = tolower(paste(recipe_title, directions))) %>%
  mutate(
    has_airfryer = str_detect(search_text, "air fryer|airfryer"),
    has_oven = str_detect(search_text, "\\boven\\b") # \\b ensures we don't match 'pr
oven'
  ) %>%
  filter(has_airfryer != has_oven) %>%
  mutate(RecipeType = ifelse(has_airfryer, "Airfryer", "Oven")) %>%
  select(-search_text)

# print new dimensions
print(dim(recipes_cleaned))
```

```
## [1] 30760    11
```

Quick substep to create a list counter function:

```
count_items <- function(x) {
  if (is.na(x) || x == "" || x == "[]") return(0)
  # Remove the brackets and split by the quote-comma-quote pattern
  # or simply count the occurrences of '",' which marks the end of an item
  return(str_count(x, '", "') + 1)
}
```

Then we compute our new variables, and remove the ones we are not interested in:

```
final_recipes <- recipes_cleaned %>%
  mutate(
    # 1. Categorization Logic
    search_text = tolower(paste(recipe_title, description)),
    has_airfryer = str_detect(search_text, "air fryer|airfryer"),
    has_oven = str_detect(search_text, "\\boven\\b"),

    recipe_type_bin = case_when(
      has_airfryer & !has_oven ~ 1,
      !has_airfryer & has_oven ~ 0,
      TRUE ~ NA_real_
    )
  ) %>%
  filter(!is.na(recipe_type_bin)) %>%
  mutate(
    # Count items inside the string
    # We use a regex to count the commas that separate quoted items
    num_ingredients = str_count(ingredients, '", "') + 1,

    # 3. Directions:
    # Clean the string to remove brackets and quotes before counting words
    clean_directions_text = directions %>%
      str_remove_all('^\\["|"\\]$') %>% # Remove outer [" and "]
      str_replace_all('", "', " "),      # Replace separators with a space

    # N(Words)
    n_words = str_count(clean_directions_text, "\\w+"),

    # M(WordLength)
    # Strip everything except letters/numbers for character count
    just_chars = str_replace_all(clean_directions_text, "[^a-zA-Z0-9]", ""),
    avg_word_length = nchar(just_chars) / n_words
  ) %>%
  select(n_words, avg_word_length, recipe_type_bin, num_ingredients)

# Check the results
summary(final_recipes$num_ingredients)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   6.000   8.000   8.346  11.000  23.000
```

Check how many of each type

```
# Descriptive Summary Table
recipe_summary <- final_recipes %>%
  group_by(recipe_type_bin) %>%
  summarise(
    n_recipes = n(),

    # Ingredients: Mean and Standard Deviation
    avg_ingredients = mean(num_ingredients, na.rm = TRUE),
    sd_ingredients  = sd(num_ingredients, na.rm = TRUE),

    # Word Count: Mean and Standard Deviation
    avg_words = mean(n_words, na.rm = TRUE),
    sd_words  = sd(n_words, na.rm = TRUE),

    # Word Length: Mean and Standard Deviation
    avg_lexical_length = mean(avg_word_length, na.rm = TRUE),
    sd_lexical_length  = sd(avg_word_length, na.rm = TRUE)
  ) %>%
  # Label the dummy codes for clarity
  mutate(recipe_type = ifelse(recipe_type_bin == 1, "Airfryer", "Oven")) %>%
  select(recipe_type, n_recipes, everything(), -recipe_type_bin)

# Print the table
print(recipe_summary)
```

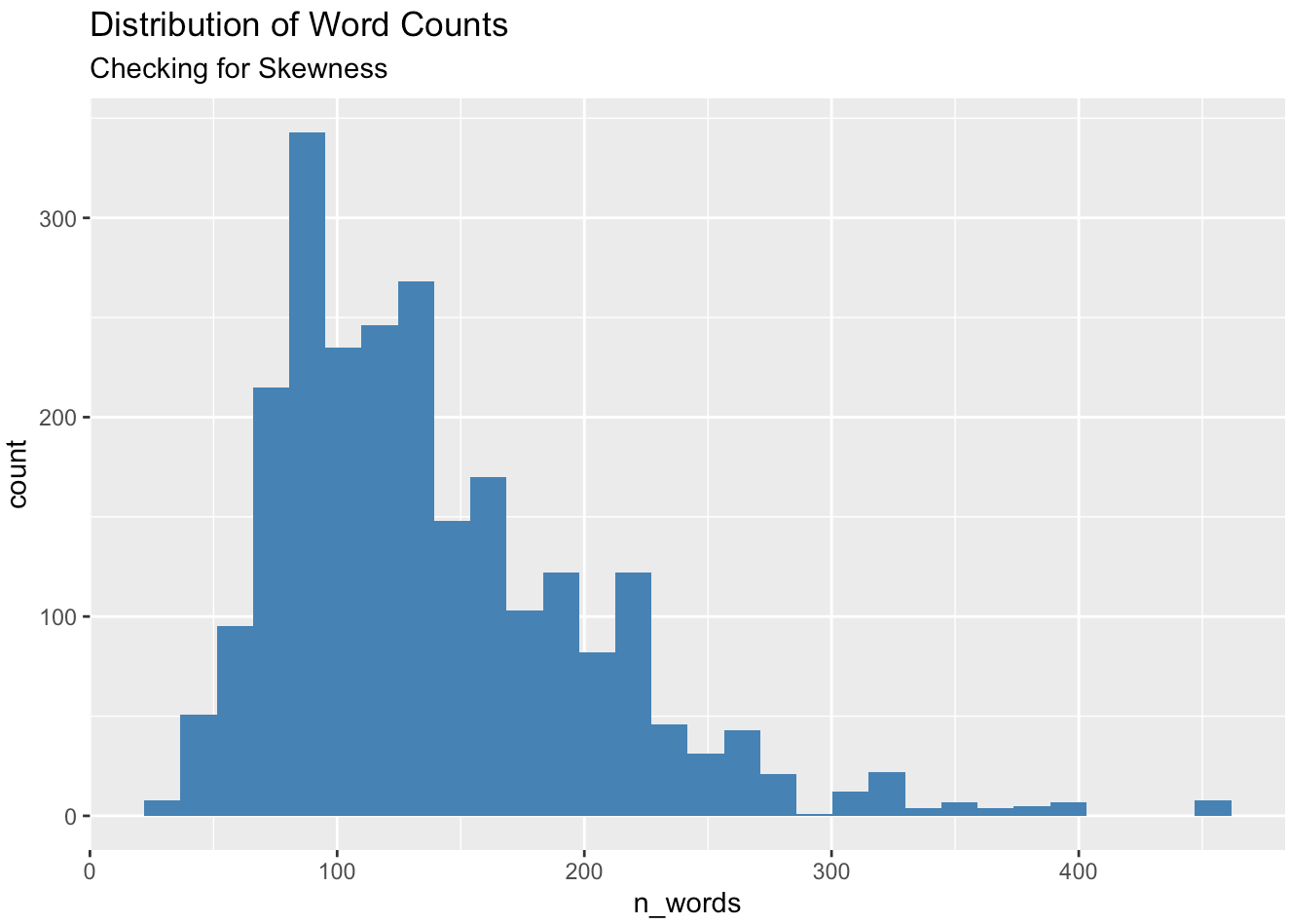
```
## # A tibble: 2 × 8
##   recipe_type n_recipes avg_ingredients sd_ingredients avg_words sd_words
##   <chr>           <int>           <dbl>          <dbl>     <dbl>    <dbl>
## 1 Oven             1417            9.11           3.84      147.     72.8
## 2 Airfryer         1002            7.27           3.42      128.     51.2
## # ℹ 2 more variables: avg_lexical_length <dbl>, sd_lexical_length <dbl>
```

Assumptions check

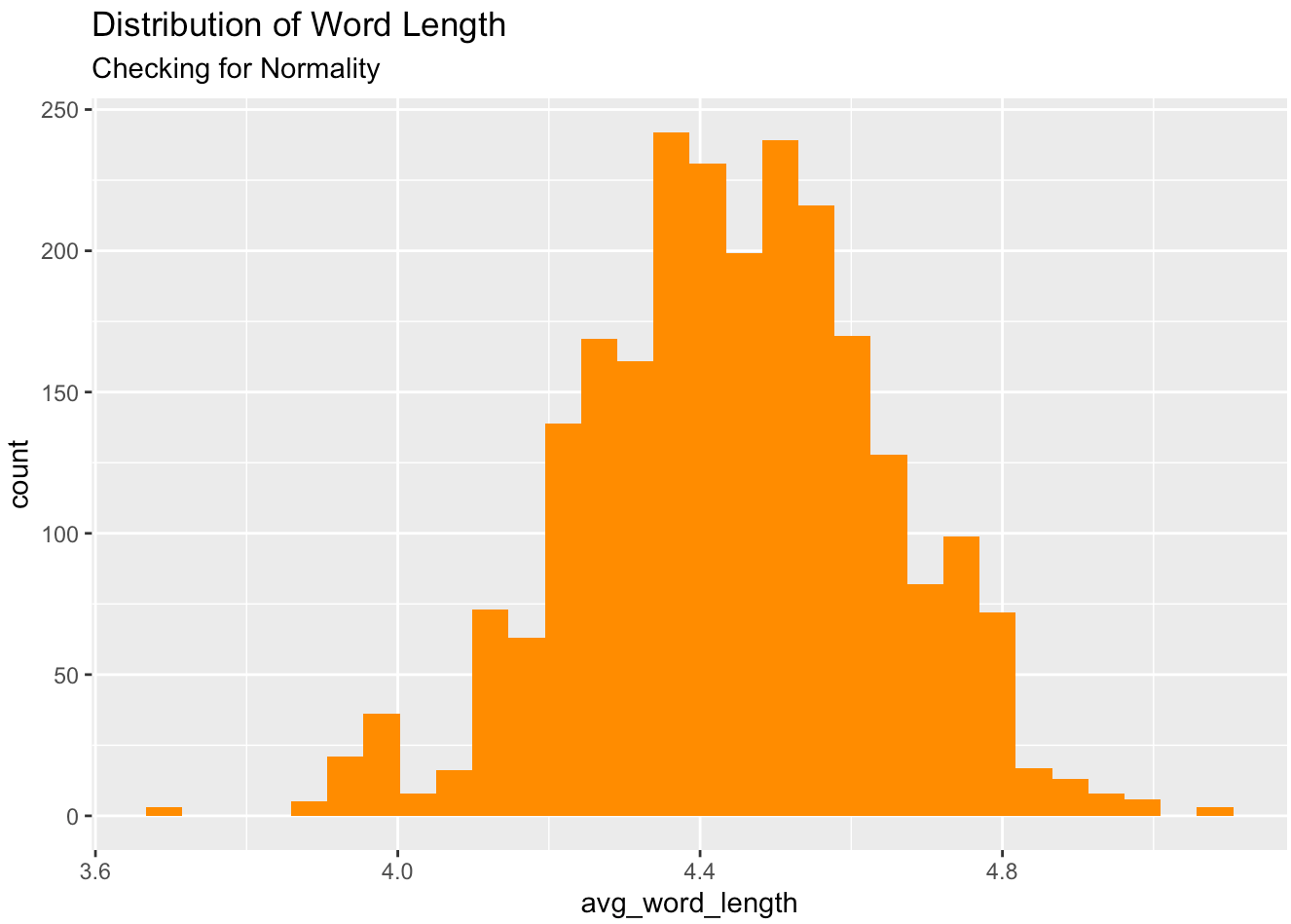
```
# 1. Check Distribution of Dependent Variables
hist_words <- ggplot(final_recipes, aes(x = n_words)) +
  geom_histogram(fill = "steelblue", bins = 30) +
  labs(title = "Distribution of Word Counts", subtitle = "Checking for Skewness")

hist_length <- ggplot(final_recipes, aes(x = avg_word_length)) +
  geom_histogram(fill = "darkorange", bins = 30) +
  labs(title = "Distribution of Word Length", subtitle = "Checking for Normality")

# 2. Check Linearity
scatter_plot <- ggplot(final_recipes, aes(x = num_ingredients, y = n_words, color = f
actor(recipe_type_bin))) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm") +
  labs(title = "Linearity Check", subtitle = "Do the slopes look straight?")

# 3. Check for Multicollinearity
# We run a simple lm just to check VIF (Variance Inflation Factor)
vif_model <- lm(n_words ~ recipe_type_bin + num_ingredients, data = final_recipes)
vif_values <- vif(vif_model)

print(vif_values) # Values > 5 indicate problematic correlation
```

```
## recipe_type_bin num_ingredients
##        1.060825        1.060825
```

```
hist_words
```

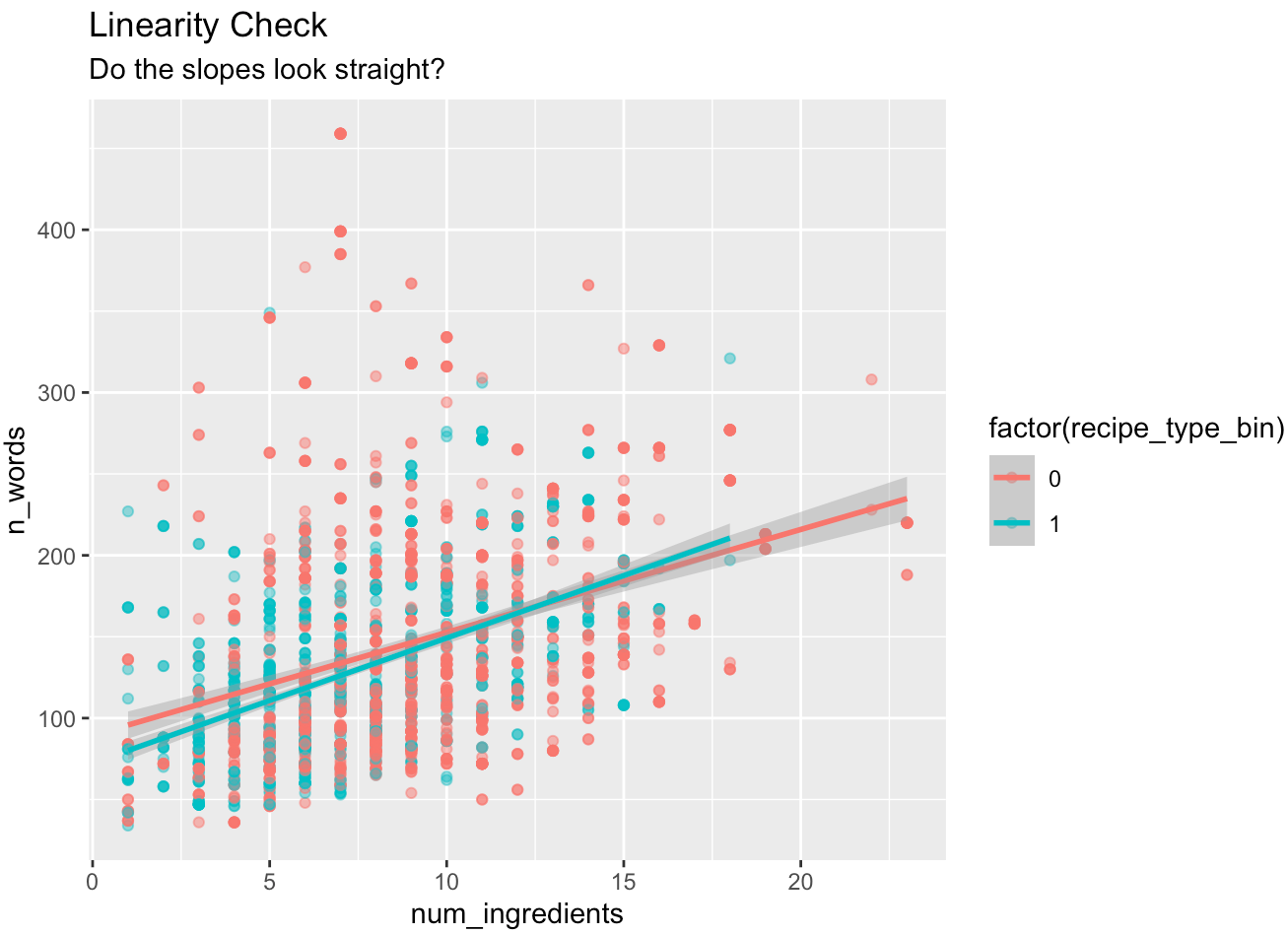

Distribution of Word Counts
Checking for Skewness
count
n_words
0
100
200
300
400

```
hist_length
```

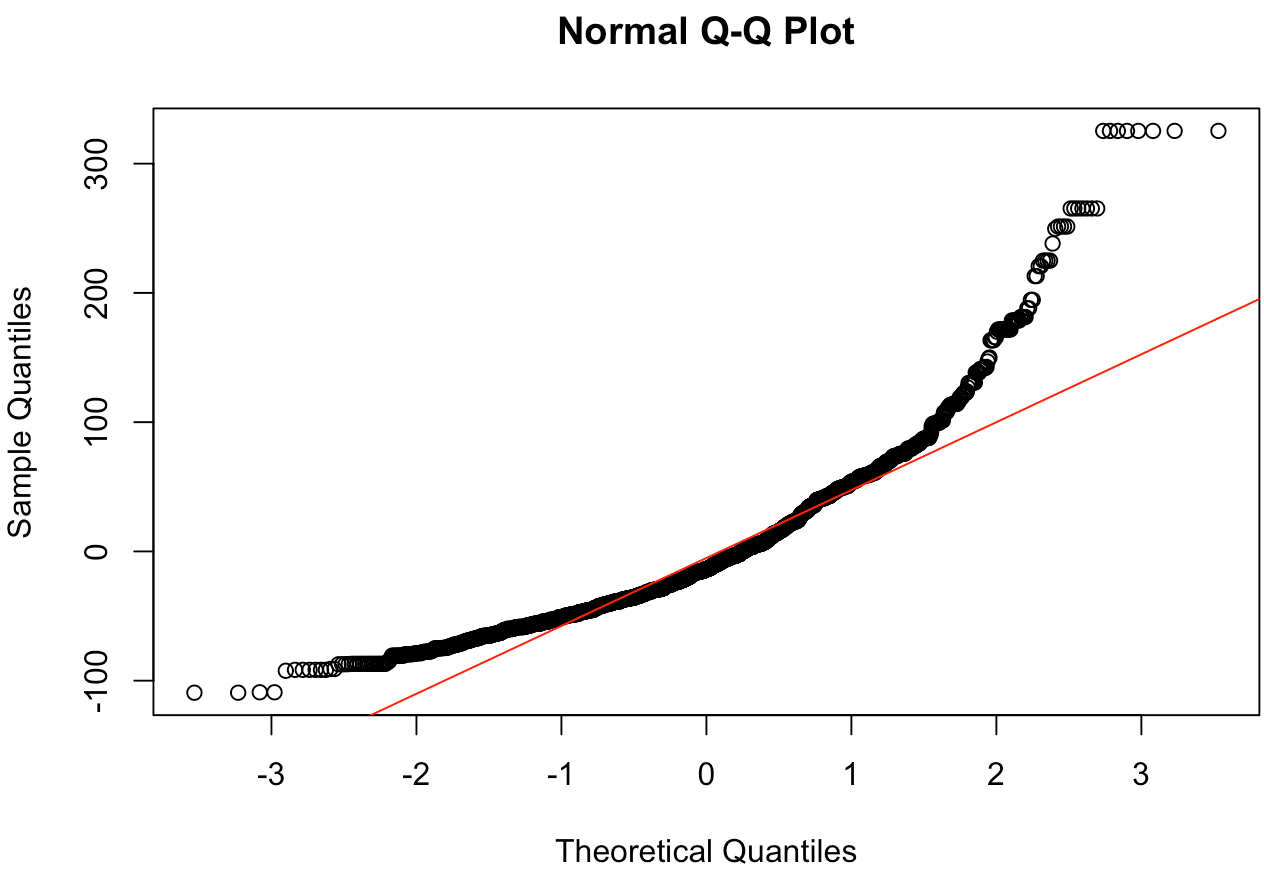



```
scatter_plot
```

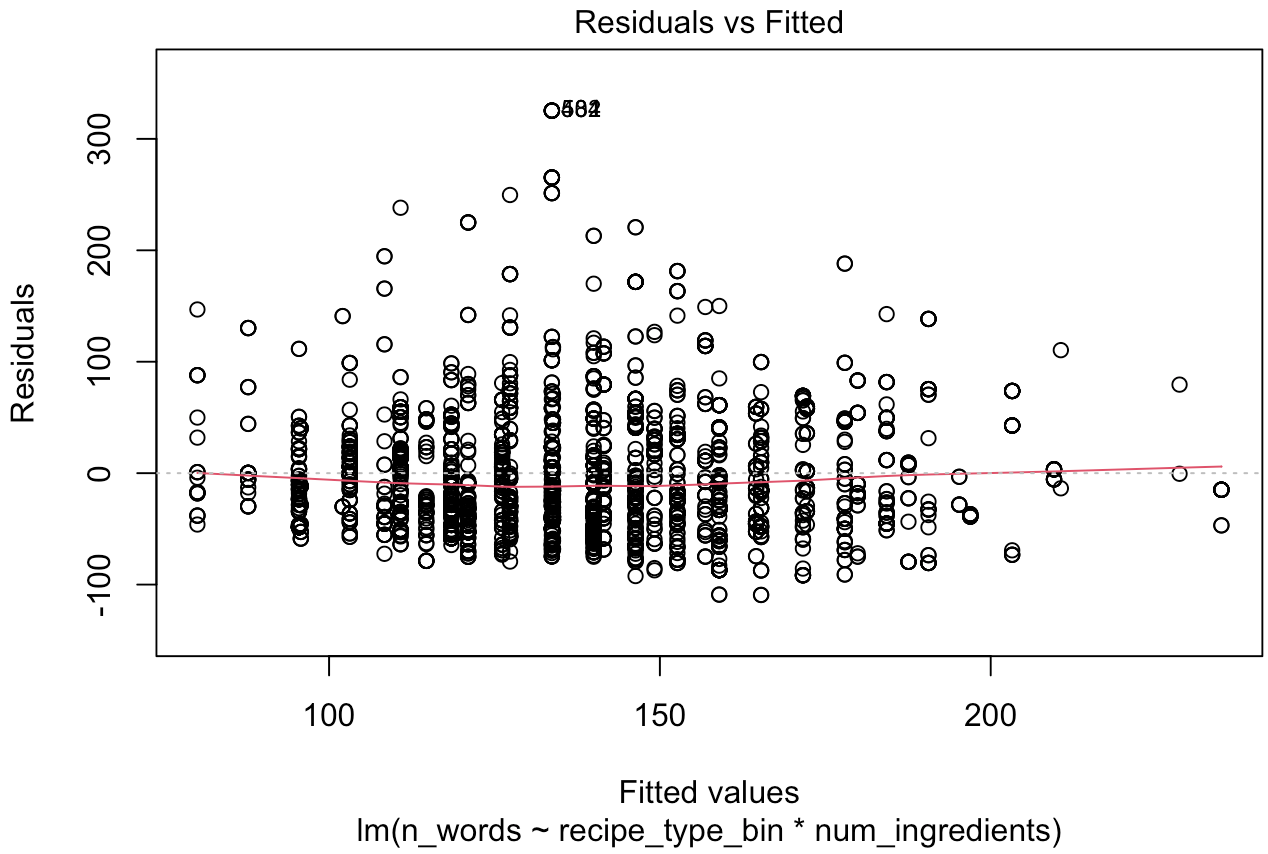
```
## `geom_smooth()` using formula = 'y ~ x'
```




```
# Run a temporary model to check residuals
temp_model <- lm(n_words ~ recipe_type_bin * num_ingredients, data = final_recipes)

# 1. Check Normality (Q-Q Plot)
# If the points follow the line, your residuals are normal.
qqnorm(residuals(temp_model))
qqline(residuals(temp_model), col = "red")
```

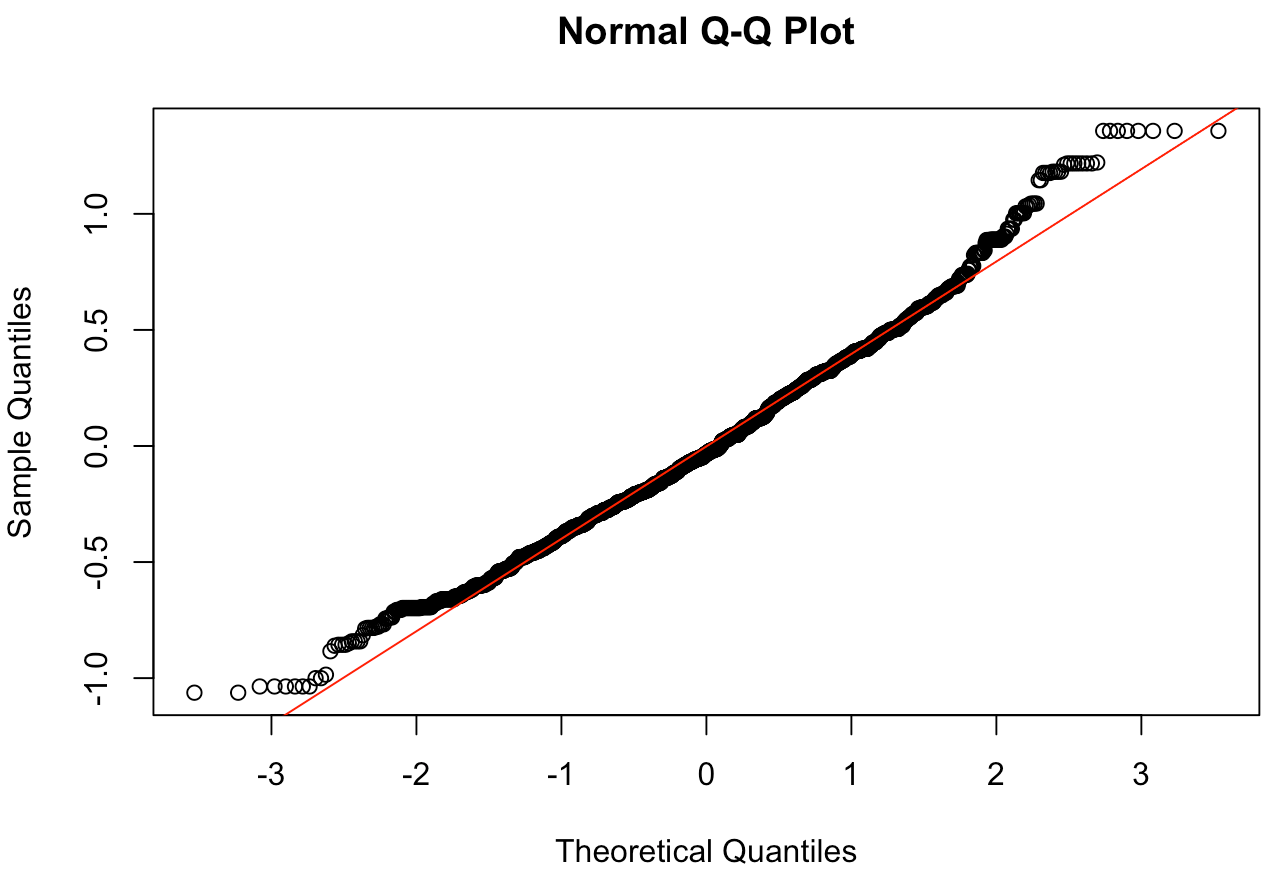



```
# 2. Check Homoscedasticity (Residuals vs Fitted)
# You want a random cloud of points. A "funnel" shape means you need a log-transform.
plot(temp_model, which = 1)
```

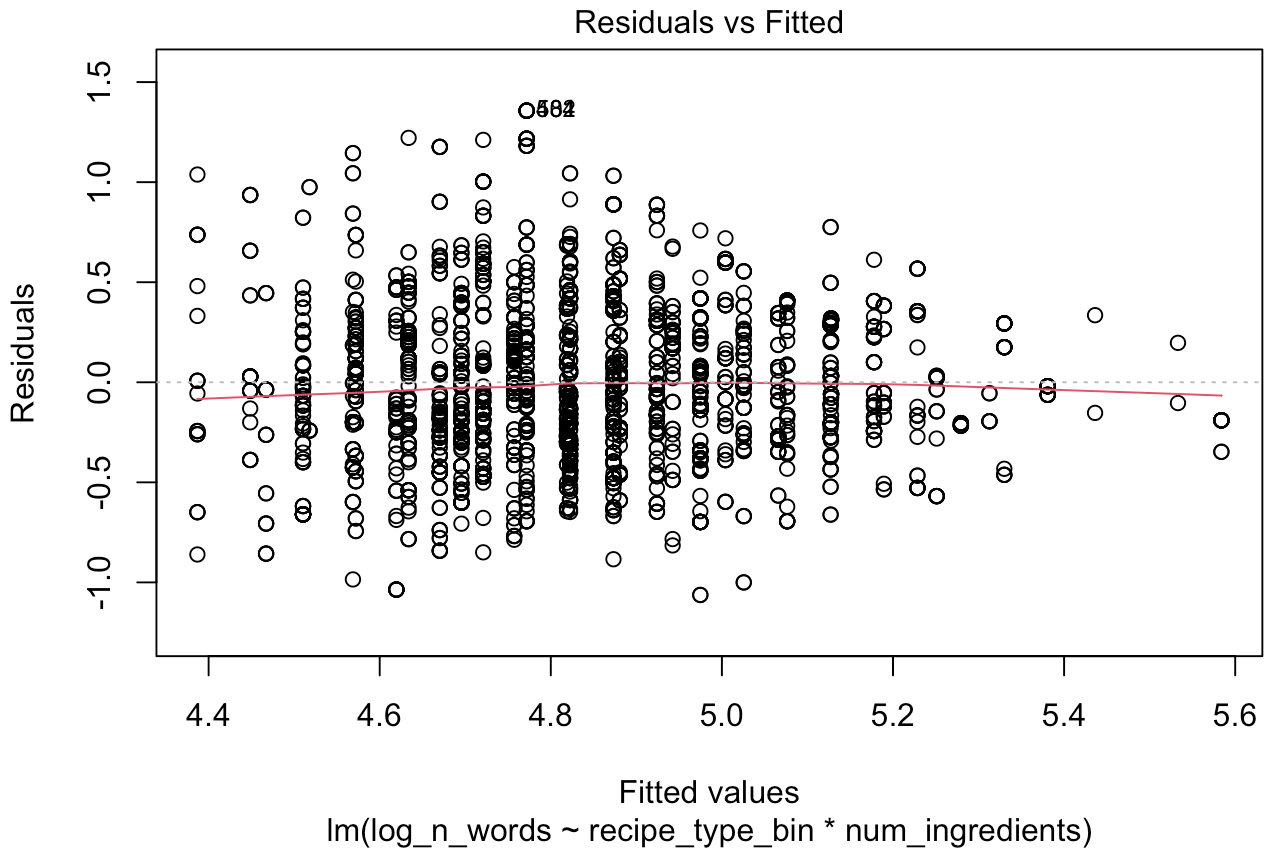



```
final_recipes <- final_recipes %>%
  mutate(log_n_words = log(n_words))
```

```
# Run a temporary model to check residuals
temp_model <- lm(log_n_words ~ recipe_type_bin * num_ingredients, data = final_recipe
s)

# 1. Check Normality (Q-Q Plot)
# If the points follow the line, your residuals are normal.
qqnorm(residuals(temp_model))
qqline(residuals(temp_model), col = "red")
```




```
# 2. Check Homoscedasticity (Residuals vs Fitted)
# You want a random cloud of points. A "funnel" shape means you need a log-transform.
plot(temp_model, which = 1)
```



Amazing, let's go!

Comparing the models

First we build all the models:

```
### --- LOG WORD COUNT MODELS (Procedural Complexity) ---
# Model 1: Main Effects
m1 <- lm(log_n_words ~ recipe_type_bin + num_ingredients,
           data = final_recipes)

# Model 2: Interaction Effect
m2 <- lm(log_n_words ~ recipe_type_bin * num_ingredients,
           data = final_recipes)

### --- WORD LENGTH MODELS (Lexical Complexity) ---
# Model 3: Main Effects
m3 <- lm(avg_word_length ~ recipe_type_bin + num_ingredients,
           data = final_recipes)

# Model 4: Interaction Effect
m4 <- lm(avg_word_length ~ recipe_type_bin * num_ingredients,
           data = final_recipes)
```

Statistical comparison

```
# Summarize Word Count Models
summary(m1) # Main Effects
```

```
##
## Call:
## lm(formula = log_n_words ~ recipe_type_bin + num_ingredients,
##     data = final_recipes)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.07021 -0.26617 -0.03471  0.25889  1.36561
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      4.380550   0.022696 193.007   <2e-16 ***
## recipe_type_bin -0.004307   0.016917  -0.255    0.799
## num_ingredients  0.054699   0.002205  24.803   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3979 on 2416 degrees of freedom
## Multiple R-squared:  0.2135, Adjusted R-squared:  0.2129
## F-statistic: 327.9 on 2 and 2416 DF,  p-value: < 2.2e-16
```

```
summary(m2) # Interaction
```

```
##
## Call:
## lm(formula = log_n_words ~ recipe_type_bin * num_ingredients,
##     data = final_recipes)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.06275 -0.27006 -0.03367  0.26721  1.35731
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    4.416433   0.027201 162.363   <2e-16 ***
## recipe_type_bin               -0.091273   0.040146  -2.274   0.0231 *
## num_ingredients                0.050759   0.002752  18.441   <2e-16 ***
## recipe_type_bin:num_ingredients  0.010966   0.004592   2.388   0.0170 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3975 on 2415 degrees of freedom
## Multiple R-squared:  0.2154, Adjusted R-squared:  0.2144
## F-statistic:   221 on 3 and 2415 DF,  p-value: < 2.2e-16
```

```
# Summarize Word Length Models
summary(m3) # Main Effects
```

```
##
## Call:
## lm(formula = avg_word_length ~ recipe_type_bin + num_ingredients,
##     data = final_recipes)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.65915 -0.12676  0.00189  0.13254  0.71819
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     4.319670   0.010963 394.036  < 2e-16 ***
## recipe_type_bin 0.040388   0.008171   4.943 8.23e-07 ***
## num_ingredients 0.012835   0.001065  12.050  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1922 on 2416 degrees of freedom
## Multiple R-squared:  0.05834,    Adjusted R-squared:  0.05756
## F-statistic: 74.84 on 2 and 2416 DF,  p-value: < 2.2e-16
```

```
summary(m4) # Interaction
```

```
##
## Call:
## lm(formula = avg_word_length ~ recipe_type_bin * num_ingredients,
##     data = final_recipes)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.67184 -0.13006 -0.00086  0.13438  0.70799
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     4.342299   0.013127 330.779  < 2e-16 ***
## recipe_type_bin                -0.014454   0.019375  -0.746  0.45574
## num_ingredients                 0.010351   0.001328   7.792 9.72e-15 ***
## recipe_type_bin:num_ingredients  0.006915   0.002216   3.121  0.00183 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1919 on 2415 degrees of freedom
## Multiple R-squared:  0.06212,    Adjusted R-squared:  0.06096
## F-statistic: 53.32 on 3 and 2415 DF,  p-value: < 2.2e-16
```

```
print("--- Comparison for Log Word Count ---")
```

```
## [1] "--- Comparison for Log Word Count ---"
```

```
anova(m1, m2)
```

```
## Analysis of Variance Table
##
## Model 1: log_n_words ~ recipe_type_bin + num_ingredients
## Model 2: log_n_words ~ recipe_type_bin * num_ingredients
##   Res.Df    RSS Df Sum of Sq      F  Pr(>F)
## 1   2416 382.55
## 2   2415 381.65  1   0.90128 5.7032 0.01701 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
print("--- Comparison for Average Word Length ---")
```

```
## [1] "--- Comparison for Average Word Length ---"
```

```
anova(m3, m4)
```

```
## Analysis of Variance Table
##
## Model 1: avg_word_length ~ recipe_type_bin + num_ingredients
## Model 2: avg_word_length ~ recipe_type_bin * num_ingredients
##   Res.Df    RSS Df Sum of Sq      F   Pr(>F)
## 1   2416 89.249
## 2   2415 88.891  1   0.35842 9.7376 0.001827 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

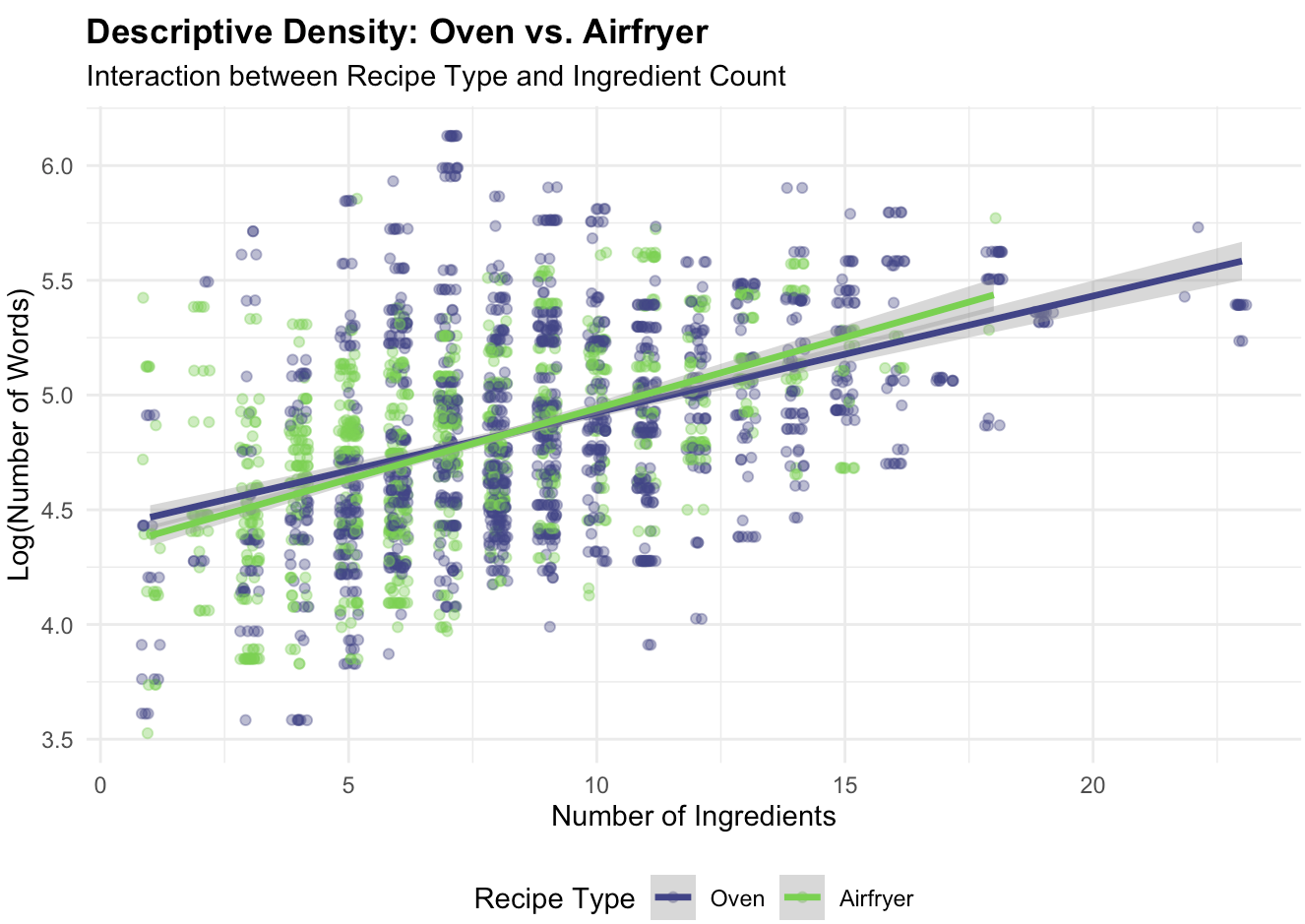
Plots

```
# plot a: count words interaction
plot_words <- ggplot(final_recipes, aes(x = num_ingredients, y = log_n_words, color =
factor(recipe_type_bin))) +
  # Add raw points with some transparency (jittered to prevent overlap on the ordinal
x-axis)
  geom_jitter(alpha = 0.4, width = 0.2) +
  # Add the regression lines from the models
  geom_smooth(method = "lm", se = TRUE, size = 1.2) +
  # Styling
  scale_color_viridis_d(begin = 0.2, end = 0.8, labels = c("Oven", "Airfryer")) +
  labs(
    title = "Descriptive Density: Oven vs. Airfryer",
    subtitle = "Interaction between Recipe Type and Ingredient Count",
    x = "Number of Ingredients",
    y = "Log(Number of Words)",
    color = "Recipe Type"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom", plot.title = element_text(face = "bold"))
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

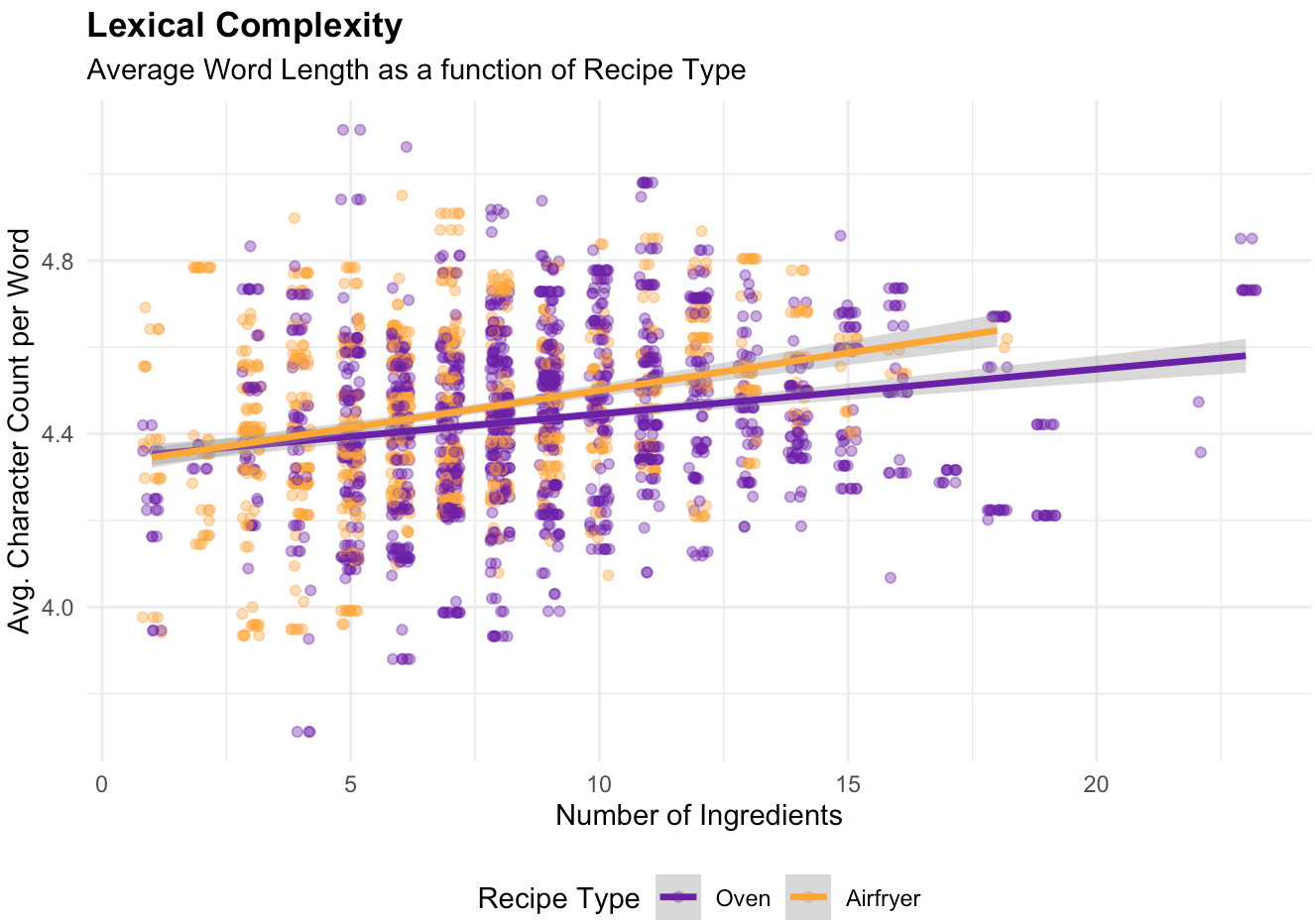
```
# plot B: word length
plot_length <- ggplot(final_recipes, aes(x = num_ingredients, y = avg_word_length, co
lor = factor(recipe_type_bin))) +
  geom_jitter(alpha = 0.4, width = 0.2) +
  geom_smooth(method = "lm", se = TRUE, size = 1.2) +
  scale_color_viridis_d(begin = 0.2, end = 0.8, labels = c("Oven", "Airfryer"), optio
n = "plasma") +
  labs(
    title = "Lexical Complexity",
    subtitle = "Average Word Length as a function of Recipe Type",
    x = "Number of Ingredients",
    y = "Avg. Character Count per Word",
    color = "Recipe Type"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom", plot.title = element_text(face = "bold"))

# Display the plots
print(plot_words)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```




```
print(plot_length)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```


Lexical Complexity
Average Word Length as a function of Recipe Type
Avg. Character Count per Word
4.8
4.4
4.0
0
5
10
15
20
Number of Ingredients
Recipe Type
Oven
Airfryer