Simulation Study on Poisson Regression: Type I and Type II Error Rates

Ryan Summers, M.S. Candidate[1]

Abstract

Researchers in public health often analyze count data, such as the number of people with a given characteristic (prevalence) or the number of new cases of a characteristic in a specific time period (incidence). Because count data is bounded to zero, Poisson regression models are often employed to examine differences such as prevalence or incidence between specific groups of people. However, these models come with strict assumptions about the underlying distribution—most notably, that the mean is equal to the variance. This assumption is often violated in practice. In this simulation study, Type I (false positive) and Type II (false negative) errors are evaluated when the assumptions of a Poisson distribution are not met. Specifically, how these errors change when comparing differences between two groups in cases where the mean does not equal the variance.

Keywords: simulation, poisson, regression, false-positives, false-negatives

Introduction

A Poisson distribution describes count data, but not just any type of count data. Specifically, it models the number of events occurring over a fixed time period in a process with a known constant rate (lambda=λ). The equality (equi-dispersion) of mean and variance in a Poisson distribution stems from this constant-rate process. However, this characteristic can make Poisson models poorly suited for identifying public health concerns. When the variance exceeds the mean, we label the data as over-dispersed. Conversely, when the variance is less than the mean, we describe the data as under-dispersed. Under-dispersion is typically less common than over-dispersion.

$$\phi = \frac{Var[y_{ij}|x_i]}{E[y_{ij}|x_i]} \qquad \phi > 0$$

$$\phi > 1 = \text{over-dispersion}$$

$$\phi < 1 = \text{under-dispersion}$$

Motivating Problem

Standard Poisson regression models lack the flexibility to adequately handle both over-dispersed and under-dispersed data within a single unified framework. While the presence of over- or under-dispersion does not affect the consistency of regression coefficients for prediction, it does result in biased standard errors. Over-dispersion leads to an inflated Type I error rate, causing apparent significant differences between groups when, in reality, none exist. Conversely, under-dispersion increases the Type II error rate, masking truly significant differences between groups. This highlights the importance of testing for violations of the Poisson assumption, properly accounting for dispersion, or identifying a more appropriate model.

Simulation Set-up

Count data was randomly generated from a Poisson distribution in R using the rpois() function ($y_{ij} \sim Pois(\mu_i)$). The simulated data was then modified with a scaling parameter to increase or decrease the variance with respect to the mean, violating the assumptions of the Poisson distribution. The data was transformed to create an ever-increasing or decreasing variance-to-mean ratio ranging from 0.1 to 10. In other words, the variance at the extremes could have been 1/10th of the mean or 10 times the mean. Two groups were randomly sampled from the Poisson distribution, and a generalized linear model (GLM) with the Poisson family was fit to derive p-values.

For Type I error simulation, data was generated so that there was no real difference in the underlying distribution of the two groups. Therefore, the GLM for Poisson should result in p-values < 0.05 in only 5% of the simulations (i.e., 5% Type I error or False Positive Rate). For Type II error simulation, data was generated from two Poisson distributions that differed with varying degrees, using Cohen's d standardized mean difference. Cohen's d essentially measures the effect size of the difference between two means. The effect sizes tested ranged from 0.1 to 1. Interpretations vary for these ranges, but typically 0.2 is considered a small effect size, 0.5 is medium, and 0.8 is large.

The simulation was run 10,000 times, testing five different sample sizes for each simulation. The simulations generated observations based on the five sample sizes tested, which ranged from 10 to 100. The results are presented graphically and in a table, showing the varying variance-to-mean ratios for Type I errors and Cohen's d for Type II errors. R v4.4.1 (2024-06-14: Race for Your Life) was used for all analyses.

Results

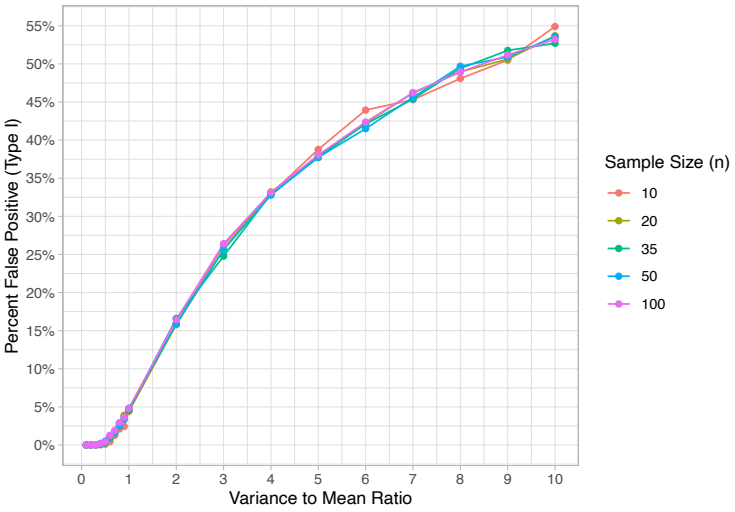

Figure 1: Proportion of Type I Errors Across Range of Variance-to-Mean Ratios

[1]Colorado School of Public Health, University of Colorado Anschutz Medical Campus, Aurora, CO

Table 1: % of Type I Errors With Varying Under-dispersion Rates Across Multiple Sample Sizes

Dispersion	n=10	n=20	n=35	n=50	n=100
0.1	0.0%	0.0%	0.0%	0.0%	0.0%
0.2	0.0%	0.0%	0.0%	0.0%	0.0%
0.3	0.0%	0.0%	0.0%	0.0%	0.0%
0.4	0.0%	0.1%	0.1%	0.1%	0.2%
0.5	0.1%	0.3%	0.3%	0.5%	0.4%
0.6	0.4%	0.8%	0.9%	1.1%	1.3%
0.7	1.3%	1.5%	1.6%	1.7%	1.9%
0.8	2.1%	2.8%	2.4%	2.5%	2.9%
0.9	2.4%	3.9%	3.6%	3.4%	3.6%

Results of the Type I error simulations are plotted in Figure 1. Given that the underlying data truly follows a Poisson process with a variance-to-mean ratio of 1 (mean = variance), the tests result in a false-positive rate of 5% for all sample sizes. However, as soon as the variance-to-mean ratio doubles (variance is twice that of the mean), we see that false-positives more than triple from 5% to approximately 15%. Interestingly, when the variance-to-mean ratio is less than 1, false-positive rates appear much less frequently in a decreasing linear fashion Table 1.

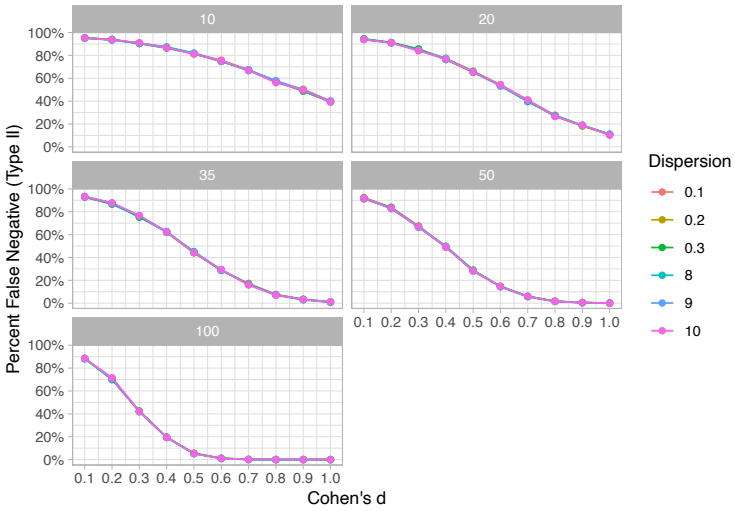

Figure 2: Proportion of Type II Errors Across Range of Cohen's d Stratified by Sample Size

Results of the Type II error simulations are plotted in Figure 2. In this case, simulated data was generated from two different Poisson distributions. Varying Cohen's effect sizes were plotted on the x-axis with lines representing dispersion rates from 0.1 to 10. These results were stratified by the five sample sizes tested. As expected, the plot shows that larger sample sizes lead to fewer false-negatives at smaller effect sizes and decrease more rapidly as the effect size increases. Furthermore, the dispersion rate appeared to have no effect when conditioned on Cohen's d effect size. Only when the sample size is small (n=10) do we see some separation between dispersion rates at around a Cohen's d effect size of 0.8.

Conclusion

Results from the Type I simulations show that the use of Poisson regressions can inflate the risk of false positives if the equal variance assumption is violated. This suggests that standard Poisson regression models are only appropriate when the strict mean-equal-variance assumptions are met. As shown in Figure 1, when the data exhibits over-dispersion, the risk of false positives increases exponentially.

Results from the Type II simulations were quite surprising. Based on the Type I simulations, there was an expectation that Type II errors would be higher when dispersion was less than one. However, as noted in Figure 2, even when the data exhibited extreme under-dispersion (< 0.5), the percentage of Type II errors was identical to when the data exhibited extreme over-dispersion (>7). This suggests that the power of the Poisson regression to detect true differences between groups is robust to violations of the dispersion assumption, at least within the range of parameters tested in this simulation.

A limitation of this simulation study is that the methods used to generate the groups and create violations of equi-dispersion resulted in both groups having either under- or over-dispersion. In practice, due to a phenomenon known as Simpson's Paradox, it's possible for groups to have different levels of dispersion. Future work will explore testing for significance when the groups have differing equi-dispersion violations.

These findings highlight the importance of carefully assessing the assumptions of Poisson regression models in practice. While the simulations indicate the model is robust to violations of the dispersion assumption for detecting true differences (Type II errors), it is highly sensitive to over-dispersion when it comes to false positives (Type I errors). Researchers should consider alternative models, such as negative binomial regression or quasi-Poisson models, when dealing with over-dispersed count data to maintain appropriate control over Type I error rates.

References

Errington, Adam, Jochen Einbeck, Jonathan Cumming, Ute Rössler, and David Endesfelder. 2022. *The International Journal of Biostatistics* 18 (1): 183–202. https://doi.org/doi:10.1515/ijb-2020-0079.

(Errington et al. 2022)

Code Appendix

```
# ---
# title: 'Simulation Study on Poisson Regression: Type I and Type II Error Rates'
# author: Ryan Summers, M.S. Candidate, Colorado School of Public Health, University
#   of Colorado Anschutz Medical Campus, Aurora, CO]
# date: "2024-10-12"
# output:
#   bookdown::html_document2: default
#   bookdown::pdf_document2: default
#   html_document:
#     df_print: paged
# bibliography: summers.bib
# ---
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(number_sections = FALSE)
knitr::opts_chunk$set(fig.align='center')

library(tidyverse)
library(ggplot2)
library(magrittr)
library(kableExtra)
library(naniar)
library(tidylog)
library(psych)
library(knitr)
library(AER)
library(pastecs)
library(doParallel)
library(parallel)
library(foreach)
library(data.table)
library(tictoc)
library(gt)
library(bookdown)
library(kableExtra)
library(scales)
### Type I Error Graph

# Load csv (computationally expensive so will need to import csv)
type1_results <- read_csv('type1_results.csv')

# Convert wide to long for graphing
fig1_long <- type1_results %>%
  pivot_longer(cols = perc_type1,
               names_to = "Error",
               values_to = "FP")


ggplot(data = fig1_long,#[fig1_long$scalar<5,],
       aes(x = scalar, y = FP, group=sample_size,
           color = factor(sample_size))) +
```

```
  geom_line() +
  geom_point() +
  labs(
    y="Percent False Positive (Type I)", x="Variance to Mean Ratio",
       color="Sample Size (n)") +
  scale_x_continuous(breaks = seq(0, 10, by = 1)) +
  scale_y_continuous(
    breaks = seq(0, 1, by = 0.05),
    #breaks = scales::pretty_breaks(n = 10),
    labels = function(x) paste0(x*100, "%")) +
  theme_light()

type1_results %>%
  filter(scalar<1) %>%
  select(scalar, sample_size, perc_type1) %>%
  spread(key = sample_size, value = perc_type1) -> test

test %>%
  # create table
  gt() %>%
  tab_style(
    style = cell_text(weight = "bold"),
    locations = cells_column_labels()) %>%

  # modify row cells to be gray
  tab_style(
    style = cell_fill(color='gray95'),
    locations = list(
      cells_body(rows = c(1,3,5,7,9)),
      cells_title())) %>%

  # modify column names
  cols_label(
    scalar ~ 'Dispersion',
    `10` ~ "n=10",
    `20` ~ "n=20",
    `35` ~ "n=35",
    `50` ~ "n=50",
    `100` ~ "n=100"
  ) %>%

  # Scale values by 100 and add % symbol
  fmt_percent(
    columns = c(`10`, `20`, `35`, `50`, `100`),
    decimals = 1
  ) %>%

  # add a source note
  # tab_source_note(
  #   source_note = md(c('Table 1: % of Type I Errors With Varying Under-dispersion Rates Across Multiple Sample

  # add a footnote (if needed)
  # tab_footnote(
  #   footnote = "This is a footnote",
  #   locations = cells_column_labels(2)) %>%

  # add a header and subtitle
  tab_header(
    title = 'Table 1: % of Type I Errors With Varying Under-dispersion Rates Across Multiple Sample Sizes')

  # adjust column widths of table
```

```
  #tab_options(table.width = '35%')

test %>%
  mutate(across(2:6, ~ percent(., accuracy = 0.1))) %>%
  kbl(
  caption = '% of Type I Errors for Under-dispersed Data',
  booktabs = TRUE,
  align = 'cc',
  linesep = "",
  col.names = c('Dispersion', 'n=10', 'n=20', 'n=35', 'n=50', 'n=100')
  ) %>%
  add_header_above(c(" " = 1, "Sample Sizes" = 5), bold = TRUE) %>%
  collapse_rows(columns = 1, latex_hline = "major", valign = "middle") %>%
  kable_styling(
    latex_options = c("hold_position", "HOLD_position", "scale_down"),
    full_width = F) %>%
  column_spec(1, width = "4cm") %>%
  column_spec(2:6, width = "2cm") %>%
  row_spec(0, bold = TRUE, hline_after = TRUE) %>%
  kable_classic()

### Type II Error Graph

# Load csv (computationally expensive so will need to import csv)
type2_results3 <- read_csv('type2_results3.csv')

# Convert wide to long for graphing
fig3_long <- type2_results3 %>%
  pivot_longer(cols = perc_type2,
               names_to = "Error",
               values_to = "FN")

fig3_long %>%
  filter((scalar<=0.3 | scalar>=8) & scalar <=10) %>%
  ggplot(aes(x = cohensd, y = FN, group=scalar,
           color = as.factor(scalar))) +
  geom_line() +
  geom_point() +
  labs(y="Percent False Negative (Type II)", x="Cohen's d",
       color="Dispersion") +
  scale_x_continuous(breaks = seq(0.1, 1, by = 0.1)) +
  scale_y_continuous(
    breaks = seq(0, 1, by = 0.2),
    labels = function(x) paste0(x*100, "%")) +
  facet_wrap(~ sample_size, ncol = 2) +
  theme_light()

type2_results3 %>%
  #filter(scalar>1) %>%
  group_by(cohensd) %>%
  select(scalar, sample_size, perc_type2) %>%
  spread(key = sample_size, value = perc_type2) -> test2

test2 %>%
  # create table
  gt() %>%
  tab_style(
    style = cell_text(weight = "bold"),
    locations = cells_column_labels()) %>%

  # modify row cells to be gray
```

```
  tab_style(
    style = cell_fill(color='gray95'),
    locations = list(
      cells_body(rows = c(1,3,5,7,9)),
      cells_title())) %>%

  # modify column names
  cols_label(
    scalar ~ 'Dispersion',
    `10` ~ "n=10",
    `20` ~ "n=20",
    `35` ~ "n=35",
    `50` ~ "n=50",
    `100` ~ "n=100"
  ) %>%

  # Scale values by 100 and add % symbol
  fmt_percent(
    columns = c(`10`, `20`, `35`, `50`, `100`),
    decimals = 1
  ) %>%

  # add a source note
  # tab_source_note(
  #   source_note = md(c('Table 1: % of Type I Errors With Varying Under-dispersion Rates Across Multiple Sample

  # add a footnote (if needed)
  # tab_footnote(
  #   footnote = "This is a footnote",
  #   locations = cells_column_labels(2)) %>%

  # add a header and subtitle
  tab_header(
    title = 'Table 1: % of Type I Errors With Varying Under-dispersion Rates Across Multiple Sample Sizes')

  # adjust column widths of table
  #tab_options(table.width = '35%')


library(tidyverse)
library(ggplot2)
library(magrittr)
library(kableExtra)
library(naniar)
library(tidylog)
library(psych)
library(knitr)
library(AER)
library(pastecs)
library(doParallel)
library(parallel)
library(foreach)
library(data.table)
library(tictoc)
library(gt)




gen_plot_data <- function(ct, mean_val, var_val) {
```

```r
  # Input validation
  if (ct <= 0) stop("The number of data points (ct) must be positive.")
  if (mean_val < 0) stop("Mean value (mean_val) must be non-negative.")
  if (var_val < 0) stop("Variance value (var_val) must be non-negative.")

  # Generate sample data using a normal distribution
  sample_data <- round(rnorm(n = ct, mean = mean_val, sd = sqrt(var_val)))

  # Remove negative values and zeros, ensuring at least some data remains
  sample_data <- sample_data[sample_data > 0]

  # If there are no positive values, regenerate until we get at least one
  while (length(sample_data) == 0) {
    sample_data <- round(rnorm(n = ct, mean = mean_val, sd = sqrt(var_val)))
    sample_data <- sample_data[sample_data > 0]
  }

  # Create a df and return it
  sample_data <- data.frame('dv' = sample_data)
  return(sample_data)
}




create_pois_sample_plot <- function(sample_data){

  max_val <- max(sample_data$dv)
  sample_lambda <- MASS::fitdistr(sample_data$dv, "Poisson")[[1]][[1]]

  # Create observed count proportions
  cts_table <- prop.table(table(factor(sample_data$dv, levels = 0:max_val)))
  cts_table <- data.frame(cts_table)
  names(cts_table) = c("Count", "Density")

  # Fit Poisson distribution
  pois_fit_to_sample <- dpois(0:max_val, sample_lambda)
  pois_cts = data.frame('Count' = 0:max_val, 'Density' = pois_fit_to_sample)

  # Calculate variance-to-mean ratio
  var_mean_ratio <- round(var(sample_data$dv) / mean(sample_data$dv))

  # Title with variance-to-mean ratio
  plot_title <- paste0("Variance to Mean Ratio ~", var_mean_ratio)

  # Create the plot
  sample_plot <-  ggplot(data = cts_table, aes(x = as.numeric(Count), y = Density)) +
    geom_col(fill = "#999999") +
    geom_area(data = pois_cts, aes(x = Count, y = Density),
              fill = "#E69F00", alpha = 0.4) +
    ggtitle(plot_title) +
    scale_x_continuous(breaks = seq(from = 0, to = max_val, by = 5)) +
    theme_light() +
    scale_y_continuous(labels = function(x) paste0(x * 100, "%"), limits = c(0, .1))

  show(sample_plot)
  return(sample_plot)
}
```

```r
set.seed(42)

ct = 1000
mean_val = 20
var_mults = c(1, 2, 3, 4, 10, 20)

# Function to generate data and create plots
generate_data_and_plot = function(ct, mean_val, var_mult) {
  data = gen_plot_data(ct, mean_val, mean_val * var_mult)
  plot = create_pois_sample_plot(data)
  return(plot)
}

# Create a list to store plots
poisson_plots = lapply(var_mults, function(var_mult) {
  generate_data_and_plot(ct, mean_val, var_mult)
})

# Access each plot as poisson_plots[[1]], poisson_plots[[2]], etc.
```

```r
##########################
###### Simulations ######
##########################


# Function to generate Poisson data for comparison of two groups w/ null of
# no difference as default and include scalar parameter to vary dispersion
gen_pois_data <- function(count, lambda, scalar, mean_diff = 0) {
  ###   count: Poisson count
  ###   lambda: Poisson rate (count/time-period)
  ###   scalar: Dispersion effect (under or over)
  ###   mean_diff: Mean difference between groups

  # Generate random data from Poisson process
  pois_dat1 <- rpois(count, lambda) * scalar
  pois_dat2 <- rpois(count, lambda) * scalar + (mean_diff * scalar)

  # Combine data into one vector and create a data table (faster than df)
  df <- data.table(response = c(pois_dat1, pois_dat2),
                   group = factor(c(rep(0, count),
                                    rep(1, count))))
  return(df)
}
```

```r
# Function to derive p-values using a GLM for Poisson data
gen_pvalue <- function(data) {
  ###   data: Data generated from random Poisson process (rpois)
```

```r
  # Fit Poisson GLM
  p_fit <- glm(response ~ group, data = data, family = "poisson")

  # Extract p-value from model summary
  p_val <- summary(p_fit)$coefficients[8]
  return(p_val)
}




# Function to get calculate type1 errors
type1_perc <- function(iters, n, mean, scalar, mean_diff = 0) {
  ###   iters: Number of iterations
  ###   n: Sample size
  ###   mean: Baseline mean value chosen (arbitrary)
  ###   Scalar: Dispersion effect (under or over)

  # Use parallel execution
  res <- foreach(i = 1:iters, .combine = 'c', .export = 'gen_pvalue') %dopar% {
    gen_pvalue(gen_pois_data(count=n,
                             mean_diff=mean,
                             scalar=scalar,
                             mean_diff))
  }
  # Calculate the proportion of significant results
  prop <- mean(res < .05)
  return(ifelse(is.na(prop), 0.0001, prop))
}




# Function to derive percentage of type1 errors for a custom range of scalars
# and sample sizes
gen_type1_results <- function(iterations = 10000, n_range, mean_val = 1) {
  ###   iterations: Number of iterations
  ###   n_range: Sample size range
  ###   mean_val: Baseline mean value chosen (arbitrary)

  # Create a sequence for the scalars:
  # 0.1 to 1 (increment = 0.1)
  # 2 to 10 (increment = 1)
  scalar_range <- c(seq(0.1, 1, by = 0.1),  # under-dispersion
                    seq(2, 10, by = 1))     # over-dispersion

  # Initialize an empty list to store results
  results_list <- list()

  # Loop over sample sizes and scalar values
  for (n_t1 in n_range) {
    # Initialize a results vector for sample size
    type1_res <- numeric(length(scalar_range))

    # Loop through each scalar value
    for (i in seq_along(scalar_range)) {
      type1_res[i] <- type1_perc(iters=iterations,
                                 n=n_t1,
```

```
                                mean=mean_val,
                                scalar = scalar_range[i])
    }

    # Store results in a data table for this sample size
    results_list[[as.character(n_t1)]] <- data.table(sample_size = n_t1,
                                                     scalar = scalar_range,
                                                     perc_type1 = type1_res)
  }

  # Combine the individual data tables into one
  dt <- rbindlist(results_list, use.names = TRUE, fill = TRUE)

  return(dt)
}




##############################################
##### Simulation for Type I Error Rates #####
##############################################

# Range of samples sizes to simulate
n_vector <- c(10, 20, 35, 50, 100)

suppressWarnings({
  # tic() & toc() = code execution timer
  tic()

  # Reproducibility
  set.seed(42)

  # Run Type I simulation
  type1_results <- gen_type1_results(iterations = 10000,
                                     n_range = n_vector,
                                     mean_val = 1)
  toc()
})

# Write to csv
#write.csv(type1_results, file='type1_results.csv', row.names = F)



# Load csv (computationally expensive so will need to import csv)
type1_results <- read_csv('type1_results.csv')



### Type I Error Graph

# Convert wide to long for graphing
fig1_long <- type1_results %>%
  pivot_longer(cols = perc_type1,
               names_to = "Error",
               values_to = "FP")
```

```r
ggplot(data = fig1_long,#[fig1_long$scalar<5,],
       aes(x = scalar, y = FP, group=sample_size,
           color = factor(sample_size))) +
  geom_line() +
  geom_point() +
  labs(y="Percent False Positive (Type I)", x="Variance to Mean Ratio",
       color="Sample Size (n)") +
  scale_x_continuous(breaks = seq(0, 10, by = 1)) +
  scale_y_continuous(
    breaks = seq(0, 1, by = 0.05),
    #breaks = scales::pretty_breaks(n = 10),
    labels = function(x) paste0(x*100, "%")) +
  theme_light()
```

```r
{r result-table-2, echo=F}
type1_results %>%
  filter(scalar<1) %>%
  select(scalar, sample_size, perc_type2) %>%
  spread(key = sample_size, value = perc_type2) -> test

test %>%
  # create table
  gt() %>%
  tab_style(
    style = cell_text(weight = "bold"),
    locations = cells_column_labels()) %>%

  # modify row cells to be gray
  tab_style(
    style = cell_fill(color='gray95'),
    locations = list(
      cells_body(rows = c(1,3,5,7,9)),
      cells_title())) %>%

  # modify column names
  cols_label(
    scalar ~ 'Dispersion',
    `10` ~ "n=10",
    `20` ~ "n=20",
    `35` ~ "n=35",
    `50` ~ "n=50",
    `100` ~ "n=100"
  ) %>%

  # Scale values by 100 and add % symbol
  fmt_percent(
    columns = c(`10`, `20`, `35`, `50`, `100`),
    decimals = 2
  ) %>%

  # add a source note
  #tab_source_note(
  #  source_note = md(c('Test'))) %>%

  # add a footnote (if needed)
  # tab_footnote(
  #   footnote = "This is a footnote",
```

```
  #   locations = cells_column_labels(2)) %>%

  # add a header and subtitle
tab_header(
  title = 'Table 1: % of Type I Errors Across Multiple Sample Sizes') %>%

  # Add bold borders to the title
  tab_style(
    style = list(
      cell_borders(sides = "top", weight = px(2)),
      cell_borders(sides = "bottom", weight = px(2))
    ),
    locations = cells_title(groups = "title")
  ) %>%

  # adjust column widths of table
  tab_options(table.width = '85%')
```

```
## Generate Poisson data for different Cohen's d values
# Cohen's d is a measure of effect size that quantifies the difference between
# two group means in terms of SD units

gen_diff_pois_data <- function(count, cohensd) {
  ###   count: Poisson count
  ###   cohensd: Cohen's d standardized effect

  # Vectorized mean values for different Cohen's d
  # values are compared to a baseline mean of 1. (e.g. if cohensd=0.3, group 1
  # has mean of 1 and group 2 has mean of 1.33 [(1.33-1)/1]=0.3)
  lambda_vals <- c("0.1" = 1.11, "0.2" = 1.21, "0.3" = 1.33, "0.4" = 1.44,
                   "0.5" = 1.57, "0.6" = 1.695, "0.7" = 1.83, "0.8" = 1.99,
                   "0.9" = 2.115, "1" = 2.28)

  # Assign group 2 lambda based on cohensd values above
  lambda2 <- lambda_vals[as.character(cohensd)]

  # Generate Poisson data
  pois_dat1 <- rpois(count, 1)
  pois_dat2 <- rpois(count, lambda2)

  # Combine data into one vector and create a data table (faster than df)
  dt <- data.table(response = c(pois_dat1, pois_dat2),
                   group = factor(c(rep(0, count),
                                    rep(1, count))))

  return(dt)
}
```

```
gen_diff_pois_data <- function(count, cohensd, scalar = 1) {
  ###   count: Poisson count
```

```r
  ###   cohensd: Cohen's d standardized effect
  ###   scalar: Dispersion effect (under or over-dispersion)

  # Vectorized mean values for different Cohen's d
  # The values are compared to a baseline mean of 1.
  lambda_vals <- c("0.1" = 1.11, "0.2" = 1.21, "0.3" = 1.33, "0.4" = 1.44,
                   "0.5" = 1.57, "0.6" = 1.695, "0.7" = 1.83, "0.8" = 1.99,
                   "0.9" = 2.115, "1" = 2.28)

  # Assign group 2 lambda based on Cohen's d values
  lambda2 <- lambda_vals[as.character(cohensd)]

  # Generate Poisson data and scale by scalar (for under/over-dispersion)
  pois_dat1 <- rpois(count, 1 * scalar)
  pois_dat2 <- rpois(count, lambda2 * scalar)

  # Combine data into one vector and create a data table
  dt <- data.table(response = c(pois_dat1, pois_dat2),
                   group = factor(c(rep(0, count),
                                    rep(1, count))))

  return(dt)
}
```

```r
# Function to calculate type2 errors based on cohensd
type2_perc <- function(iters, count, cohensd, scalar=1, mean_diff=0) {
  ###   iters: Number of iterations
  ###   count: Poisson count
  ###   cohensd: Cohen's d standardized effect
  ###   scalar: Dispersion effect (under or over)
  ###   mean_diff: Mean difference between groups

  # Use parallel execution
  res <- foreach(i = 1:iters, .combine = 'c', .export = 'gen_pvalue') %dopar% {
    gen_pvalue(gen_diff_pois_data(count,cohensd))
  }
  # Calculate the proportion of significant results
  prop <- mean(res >= .05)
  return(ifelse(is.na(prop), 0.0001, prop))
}
```

```r
gen_type2_results <- function(cd_iters, n_range) {
  ###   cd_iters: Number of iterations
  ###   n_range: Sample size range

  # Define Cohen's d range
  cd <- c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)

  # Initialize an empty list to store results for each sample size
  results_list <- list()
```

```r
  # Loop through the different sample sizes
  for (n_cd in n_range) {
    # Initialize a vector for storing results
    type2_res <- numeric(length(cd))

    # Loop through Cohen's d range and get results for each sample size
    for (i in seq_along(cd)) {
      type2_res[i] <- type2_perc(cd_iters, n_cd, cd[i])
    }
    # Store results for the current sample size in a data table
    dt <- data.table(
      sample_size = rep(n_cd, length(cd)),
      cohensd = cd,
      perc_type1 = type2_res
    )
    # Add the result to the results list
    results_list[[as.character(n_cd)]] <- dt
  }
  # Combine all results into a single data table
  type2_results <- rbindlist(results_list, use.names = TRUE, fill = TRUE)

  return(type2_results)
}
```

```r
library(parallel)
library(data.table)

gen_type2_results_parallel <- function(cd_iters, n_range,
                                       scalar_range = c(seq(0.1, 1, by = 0.1),
                                                        seq(2, 10, by = 1)),
                                       num_cores = detectCores() - 1) {
  ### cd_iters: Number of iterations
  ### n_range: Sample size range
  ### scalar_range: Range of scalar values (under- and over-dispersion)
  ### num_cores: Number of cores to use for parallelization

  # Define Cohen's d range
  cd <- c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)

  # Function to generate Type II error results for each combination
  calc_type2_for_scalar <- function(n_cd, scalar) {
    # Initialize a vector to store Type II results for each Cohen's d
    type2_res <- numeric(length(cd))

    # Loop through each Cohen's d value
    for (i in seq_along(cd)) {
      # Pass the scalar value to the type2_perc function
      type2_res[i] <- type2_perc(cd_iters, n_cd, cohensd = cd[i], scalar = scalar)
    }

    # Return the result as a data.table
    return(data.table(
      sample_size = rep(n_cd, length(cd)),
      scalar = rep(scalar, length(cd)),
      cohensd = cd,
      perc_type2 = type2_res
```

```
    ))
  }

  # Create a list of all combinations of n_range and scalar_range
  param_list <- expand.grid(n_cd = n_range, scalar = scalar_range,
                            stringsAsFactors = FALSE)

  # Use mclapply for parallel execution
  results_list <- mclapply(1:nrow(param_list), function(idx) {
    n_cd <- param_list$n_cd[idx]
    scalar <- param_list$scalar[idx]

    # Call the helper function to compute results for this combination
    calc_type2_for_scalar(n_cd, scalar)
  }, mc.cores = num_cores)

  # Combine all the results into a single data table
  type2_results <- rbindlist(results_list, use.names = TRUE, fill = TRUE)

  return(type2_results)
}




##############################################
#### Simulation for Type II Error Rates #####
##############################################

# Range of samples sizes to simulate
sample_sizes <- c(10, 20, 35, 50, 100)


# tic() & toc() = code execution timer
tic()

# reproducibility
set.seed(42)

# run Type II sim
type2_results3 <- gen_type2_results_parallel(cd_iters=10000, n_range=sample_sizes,
                                             num_cores=6)
toc()

# Write to csv
#write.csv(type2_results3, file='type2_results3.csv', row.names = F)



# Load csv (computationally expensive so will need to import csv)
type2_results3 <- read_csv('type2_results3.csv')




### Type II Error Graph

# Convert wide to long for graphing
fig3_long <- type2_results3 %>%
  pivot_longer(cols = perc_type2,
```

```
               names_to = "Error",
               values_to = "FN")

fig3_long %>%
  filter((scalar<=0.3 | scalar>=8) & scalar <=10) %>%
  ggplot(aes(x = cohensd, y = FN, group=scalar,
             color = as.factor(scalar))) +
  geom_line() +
  geom_point() +
  labs(y="Percent False Negative (Type II)", x="Cohen's d",
       color="Dispersion") +
  scale_x_continuous(breaks = seq(0.1, 1, by = 0.1)) +
  scale_y_continuous(
    breaks = seq(0, 1, by = 0.10),
    labels = function(x) paste0(x*100, "%")) +
  facet_wrap(~ sample_size, ncol = 2) +
  theme_light()
```

```
$$y_{ij} \sim Pois(\mu_i)$$
```

```
$$
\phi=\frac{Var[y_{ij}|x_i]}{E[y_{ij}|x_i]} \phantom{0000}\phi>0 \\
\phantom{00} \\
\phi>1 = \text{over-dispersion} \\
\phi<1 = \text{under-dispersion}
$$
```

```
$$
\text{Cohen's d}=\frac{\lambda_{group1} - \lambda_{group2}}{\sigma_{pooled}}
$$
```