p8105_hw5_bt2654

Boxiang Tang 2024-11-13

Problem 1

Step 1: Create a function to check for duplicate birthdays

```
# Loading the necessary packages
library(ggplot2)
library(tidyverse)
library(dplyr)
library(purrr)
library(tidyr)


# Function to check if there are duplicate birthdays in a group of size n
check_duplicate_birthday =
  function(n) {
  # Generate random birthdays for n people (1 to 365 represents each day in a
  birthdays = sample(1:365,
                     n,
                     replace = TRUE)
  # Check if there are any duplicate birthdays, return TRUE if there are dupli
  any(duplicated(birthdays))
}

# Test the created function for a group of 30 people
check_duplicate_birthday(30)
```

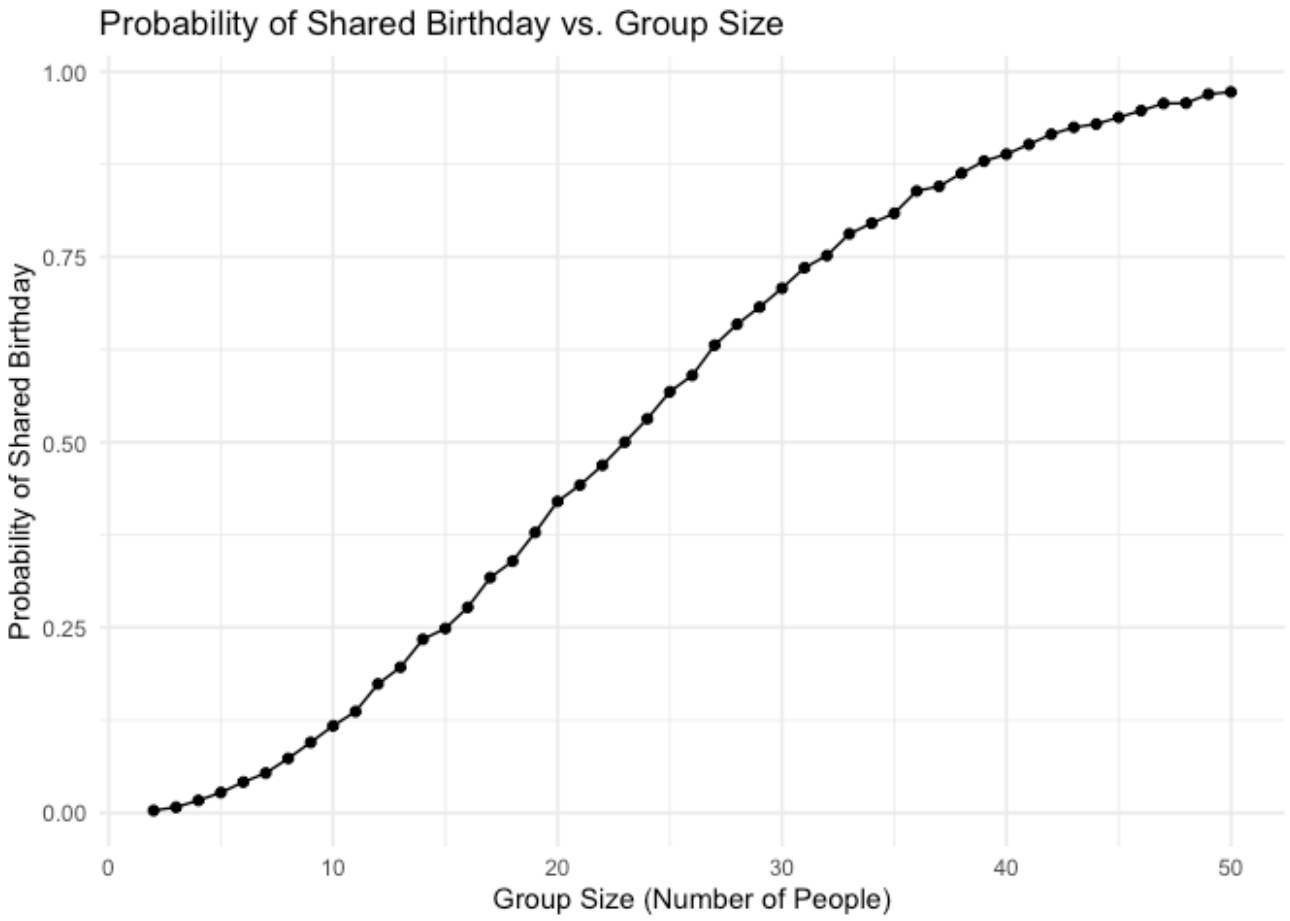
```
## [1] TRUE
```

Step 2: Run the simulation for each group size from 2 to 50

```
# Initialize a vector to store the probabilities for each group size
group_sizes = 2:50
probabilities <- numeric(length(group_sizes))

# Loop over each group size and perform 10,000 simulations
for (i in seq_along(group_sizes)) {
  n = group_sizes[i]
  # Run 10,000 simulations for the current group size
  results = replicate(10000,
                      check_duplicate_birthday(n))
  # Calculate the probability of at least two people sharing a birthday
  probabilities[i] = mean(results)
}
```

Step 3: Plot the probability as a function of group size

```
# Create a data frame for plotting
data = data.frame(GroupSize = group_sizes,
                  Probability = probabilities)

# Plot the probability of shared birthdays as a function of group size
ggplot(data, aes(x = GroupSize,
                 y = Probability)) +
  geom_line() +
  geom_point() +
  labs(title = "Probability of Shared Birthday vs. Group Size",
       x = "Group Size (Number of People)",
       y = "Probability of Shared Birthday") +
  theme_minimal()
```



Step 4: Analysis and Commentary

- As shown in the plot, the probability of at least two people sharing a birthday increases as the group size grows. This phenomenon is commonly known as the “birthday paradox.” Surprisingly, even with a group size as small as 23, the probability exceeds 50%, which is counterintuitive. This result highlights how seemingly improbable events can have higher probabilities in larger groups due to the combinatorial nature of comparisons.
- In this experiment, we assumed that birthdays are uniformly distributed across 365 days, which simplifies the calculation but may not perfectly reflect real-world birthday distributions.

Problem 2

Step 1: Set up the experiment parameters

```r
# Set sample size and standard deviation
n = 30
sigma = 5

# Define the number of simulations
num_simulations = 5000
```

Step 2: Perform t-tests for mu = 0 and save the results

```r
# Function to perform simulations for a given mean (mu)
simulate_tests =
  function(mu) {
  # Create vectors to store the estimated mean and p-values
  estimates = numeric(num_simulations)
  p_values = numeric(num_simulations)

  # Run simulations
  for (i in 1:num_simulations) {
    # Generate a sample of size n from a normal distribution with mean mu and
    sample_data = rnorm(n,
                        mean = mu,
                        sd = sigma)

    # Perform one-sample t-test against mu = 0
    t_test_result = t.test(sample_data,
                           mu = 0)

    # Extract the estimated mean and p-value
    tidy_result = broom::tidy(t_test_result)
    estimates[i] = tidy_result$estimate
    p_values[i] = tidy_result$p.value
  }

  # Return a data frame with results
  return(data.frame(estimate = estimates,
                    p_value = p_values))
}

# Perform simulations for mu = 0 and save results
results_mu_0 = simulate_tests(mu = 0)

# Display the summary results
results_mu_0 |>
summary() |>
  knitr::kable()
```

	estimate	p_value
	Min. :-3.677139	Min. :0.0001963
	1st Qu.:-0.601035	1st Qu.:0.2458980
	Median : 0.011468	Median :0.4965626
	Mean : 0.009338	Mean :0.4984652
	3rd Qu.: 0.630759	3rd Qu.:0.7501816
	Max. : 3.435530	Max. :0.9995911

Step 3: Repeat the experiment for multiple values of mu

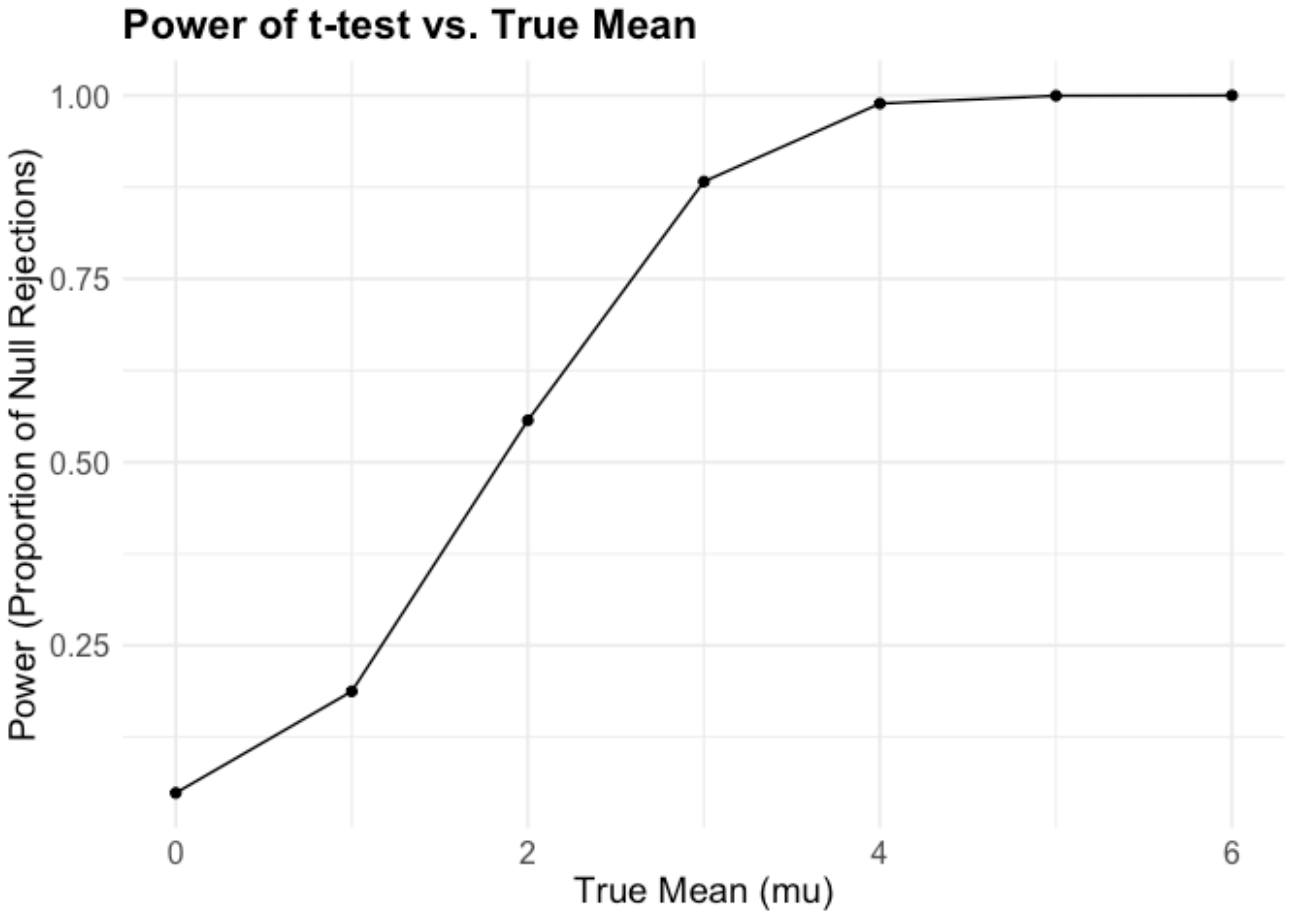
```
# Define the different values of mu to test
mu_values = 0:6

# Create a list to store the results for each mu
results_list = lapply(mu_values,
                      simulate_tests)
names(results_list) = paste0("mu_",
                             mu_values)
```

Step 4: Calculate the power (proportion of times the null hypothesis was rejected) for each mu

```
# Calculate power for each value of mu
power_values = sapply(results_list,
                      function(df) {
  mean(df$p_value < 0.05) # Proportion of p-values below significance level
})

# Create a data frame to store power results
power_data = data.frame(mu = mu_values,
                        power = power_values)
```

Step 5: Plot power vs. effect size

```
# Plot power as a function of the true mean (mu)
ggplot(power_data,
       aes(x = mu, y = power)) +
  geom_line() +
  geom_point() +
  labs(title = "Power of t-test vs. True Mean",
       x = "True Mean (mu)",
       y = "Power (Proportion of Null Rejections)") +
  theme_minimal() +
   theme(
    axis.title = element_text(size = 14),
    axis.text = element_text(size = 12),
    plot.title = element_text(size = 16,
                              face = "bold")
  )
```



Comments on the first plot:

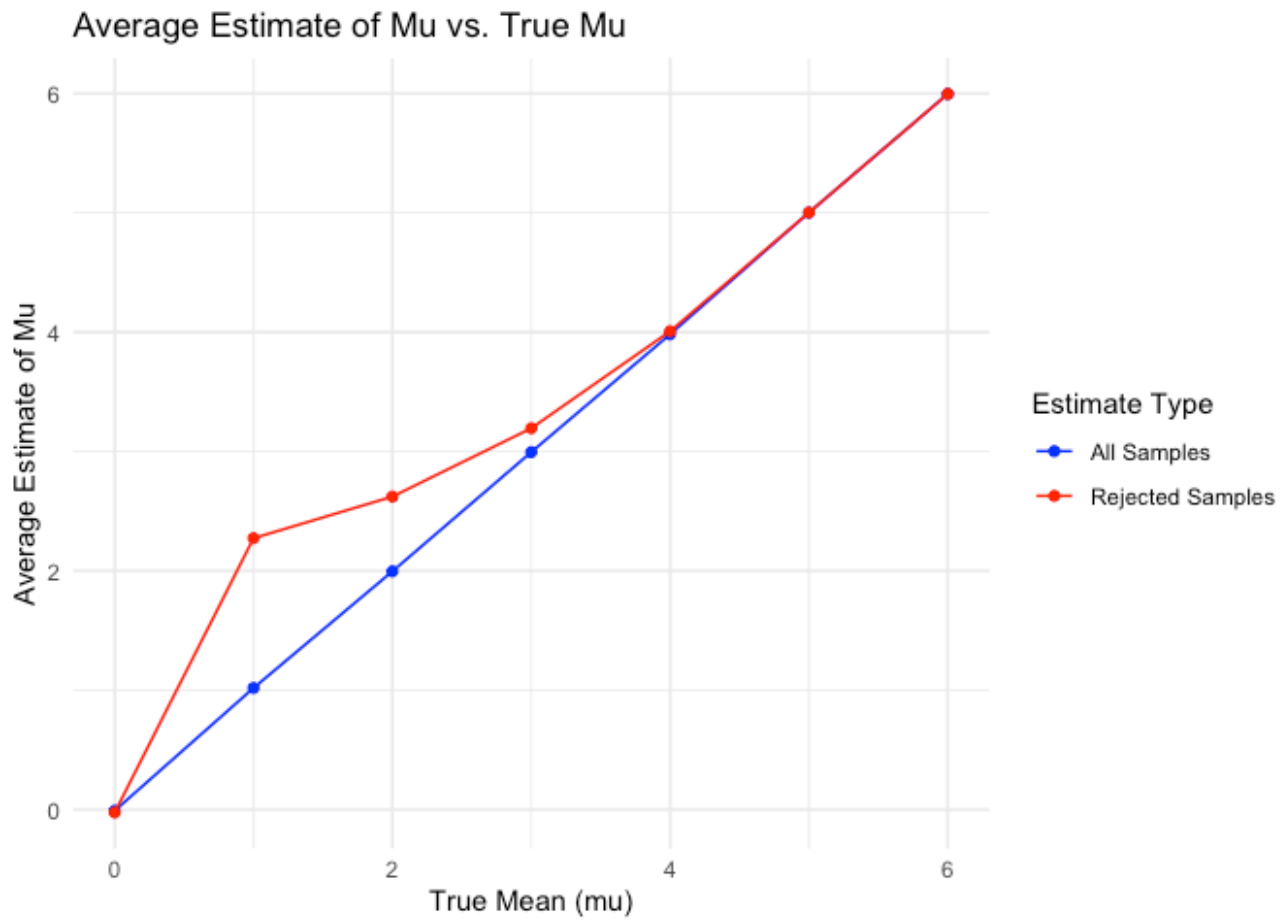
The first plot shows the power of the one-sample t-test as a function of the true mean, μ. Power is defined as the probability of correctly rejecting a false null hypothesis. From the plot, we observe that when $\mu = 0$, the power is near zero, as expected, since there is no true effect to detect. As μ increases, the power of the test also increases, eventually

reaching close to 1 (100%) for higher values of μ. This trend aligns with the concept that a larger effect size (in this case, a greater deviation of μ from 0) enhances the likelihood of detecting a true effect. This plot illustrates the positive association between effect size and test power, demonstrating that with larger true means, the test is more sensitive to correctly identifying a deviation from the null hypothesis.

Step 6: Calculate and plot the average estimate of mu

```
# Calculate the average estimate of mu for each value of true mu
average_estimates = sapply(results_list,
                           function(df) {
  mean(df$estimate)
})

# Calculate the average estimate of mu only for samples where the null was rej
average_estimates_rejected = sapply(results_list,
                                    function(df) {
  mean(df$estimate[df$p_value < 0.05])
})

# Create a data frame for plotting
estimate_data = data.frame(mu = mu_values,
                           avg_estimate = average_estimates,
                           avg_estimate_rejected = average_estimates_rejected


# Plot average estimate of mu and overlay rejected estimates
ggplot(estimate_data,
       aes(x = mu)) +
  geom_line(aes(y = avg_estimate,
                color = "All Samples")) +
  geom_point(aes(y = avg_estimate,
                 color = "All Samples")) +
  geom_line(aes(y = avg_estimate_rejected,
                color = "Rejected Samples")) +
  geom_point(aes(y = avg_estimate_rejected,
                 color = "Rejected Samples")) +
  labs(
    title = "Average Estimate of Mu vs. True Mu",
    x = "True Mean (mu)",
    y = "Average Estimate of Mu",
    color = "Estimate Type" # Legend title
  ) +
  theme_minimal() +
  scale_color_manual(
    values = c("All Samples" = "blue",
```

```
                "Rejected Samples" = "red"),
    labels = c("All Samples",
               "Rejected Samples")
  )
```



Comments on the second plot:

The second plot overlays two lines: the average estimate of $\hat{\mu}$ across all samples (blue) and the average estimate of $\hat{\mu}$ in samples where the null hypothesis was rejected (red). When $\mu = 0$, both lines start at the origin, with no observable discrepancy. However, as μ increases, we see that the average $\hat{\mu}$ for all samples continues to closely track the true mean μ, as expected. The average $\hat{\mu}$ for samples where the null was rejected, however, consistently remains higher than the true mean, especially at lower values of μ. This discrepancy results from selection bias: samples with larger deviations from 0 (positive or negative) are more likely to yield a significant p-value, thus appearing in the rejected group. As μ increases, the difference diminishes, illustrating that the bias is more pronounced for smaller effect sizes and nearly negligible when μ becomes large.

Question Answer Is the sample average of $\hat{\mu}$ across tests for which the null is rejected approximately equal to the true value of μ? Why or why not?

No, the sample average of $\hat{\mu}$ across tests where the null is rejected is generally not equal to the true mean μ, especially for smaller μ values. This bias occurs because tests that produce extreme values of $\hat{\mu}$ (farther from 0) are more likely to reject the null hypothesis. Thus, the rejected samples tend to have inflated average estimates. As μ becomes larger, the bias lessens because the effect size is naturally large enough to reject the null hypothesis more consistently without relying on selection bias.

Problem 3

Overall Data Description

Data Dimensions: This dataset contains 49 rows and 2 columns.

This dataset provides information on homicides across various U.S. cities, capturing details about each case, including victim demographics, case status, and location. Below is a description of each key variables:

- `uid` : Unique identifier for each homicide case, with 0 unique values in the dataset.
- `reported_date` : The date the homicide was reported, in the format YYYYMMDD. Ensuring consistent date format is essential for accurate time-based analysis.
- `victim_last` and `victim_first` : The last and first names of the victim, respectively. These fields allow for identification but are not typically used for statistical analysis.
- `victim_race` : The race of the victim, including categories such as .
- `victim_age` : Age of the victim at the time of the homicide, ranging from to -. Filtering out unrealistic ages (e.g., below 0 or above 120) may be necessary for accurate analysis.
- `victim_sex` : The gender of the victim, including .
- `city` and `state` : The location of the homicide. The dataset covers 0 cities and 0 states.
- `lat` and `lon` : Latitude and longitude coordinates for the location of each homicide, enabling geospatial analysis.
- `disposition` : The case status, indicating whether it was solved. Categories include: . This variable is essential for analyzing the proportion of unsolved cases.

Step 1: Load the data

```
data = read.csv("data/homicide-data.csv")
```

Step 2: Data Cleaning

```
# Create `city_state` column by combining `city` and `state`
data =
  data |>
  mutate(city_state = paste(city,
                            state,
                            sep = ", "))

# Define unsolved dispositions for identification of unsolved cases
unsolved_dispositions = c("Closed without arrest",
                          "Open/No arrest")

# Remove the "Tulsa, AL" record as it is incorrect
data =
  data |>
  filter(!(city == "Tulsa" & state == "AL"))
```

Step 3: Summarize data by city

```
# Calculate total homicides and unsolved homicides for each city
homicide_summary =
  data |>
  mutate(unsolved = disposition %in% unsolved_dispositions) |>
  group_by(city_state) |>
  summarise(
    total_homicides = n(),
    unsolved_homicides = sum(unsolved)
  ) |>
  ungroup()

knitr::kable(homicide_summary,
             caption = "Summary of Homicides by City and State",
             col.names = c("City", "Total", "Unsolved"))
```

City	Total	Unsolved
Albuquerque, NM	378	146
Atlanta, GA	973	373

City	Total	Unsolved
Baltimore, MD	2827	1825
Baton Rouge, LA	424	196
Birmingham, AL	800	347
Boston, MA	614	310
Buffalo, NY	521	319
Charlotte, NC	687	206
Chicago, IL	5535	4073
Cincinnati, OH	694	309
Columbus, OH	1084	575
Dallas, TX	1567	754
Denver, CO	312	169
Detroit, MI	2519	1482
Durham, NC	276	101
Fort Worth, TX	549	255
Fresno, CA	487	169
Houston, TX	2942	1493
Indianapolis, IN	1322	594
Jacksonville, FL	1168	597
Kansas City, MO	1190	486
Las Vegas, NV	1381	572
Long Beach, CA	378	156
Los Angeles, CA	2257	1106
Louisville, KY	576	261
Memphis, TN	1514	483

City	Total	Unsolved
Miami, FL	744	450
Milwaukee, wI	1115	403
Minneapolis, MN	366	187
Nashville, TN	767	278
New Orleans, LA	1434	930
New York, NY	627	243
Oakland, CA	947	508
Oklahoma City, OK	672	326
Omaha, NE	409	169
Philadelphia, PA	3037	1360
Phoenix, AZ	914	504
Pittsburgh, PA	631	337
Richmond, VA	429	113
Sacramento, CA	376	139
San Antonio, TX	833	357
San Bernardino, CA	275	170
San Diego, CA	461	175
San Francisco, CA	663	336
Savannah, GA	246	115
St. Louis, MO	1677	905
Stockton, CA	444	266
Tampa, FL	208	95
Tulsa, OK	583	193
Washington, DC	1345	589

Summary of Homicides by City and State

Step 4: Single City Analysis - Baltimore, MD

```
# Filter data for Baltimore, MD and perform a proportion test
baltimore_data =
  homicide_summary |>
  filter(city_state == "Baltimore, MD")

# Conduct the proportion test for unsolved homicides in Baltimore
baltimore_test =
  prop.test(baltimore_data$unsolved_homicides,
            baltimore_data$total_homicides)

# Use broom::tidy to organize test results into a clean format
baltimore_results =
  baltimore_test|>
  broom::tidy() |>
  print()
```

```
## # A tibble: 1 × 8
##   estimate statistic  p.value parameter conf.low conf.high method    alterr
##      <dbl>     <dbl>    <dbl>     <int>    <dbl>     <dbl> <chr>     <chr>
## 1    0.646      239. 6.46e-54         1    0.628     0.663 1-sample… two.si
```

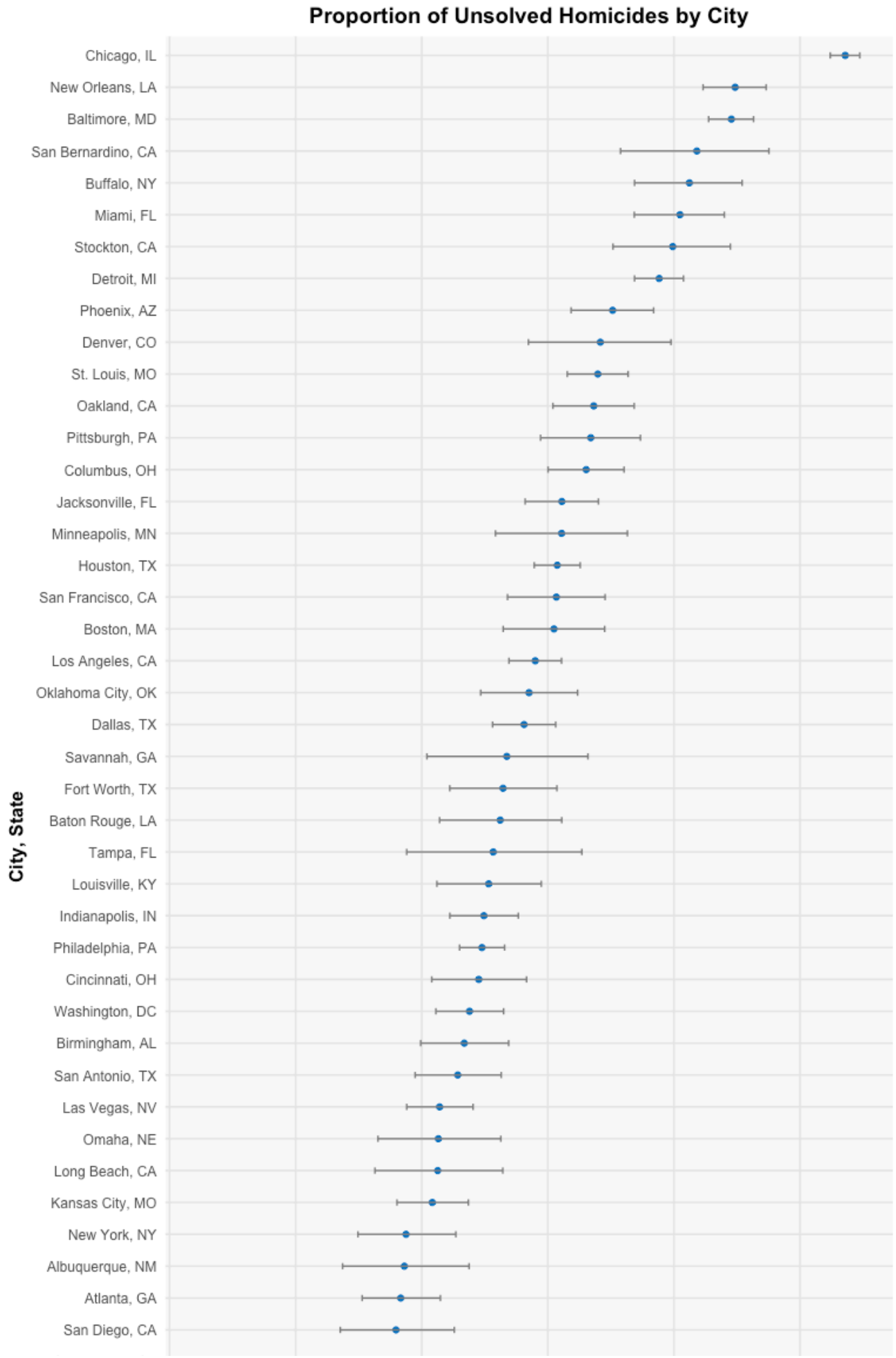
Step 5: Multi-City Analysis - All Cities

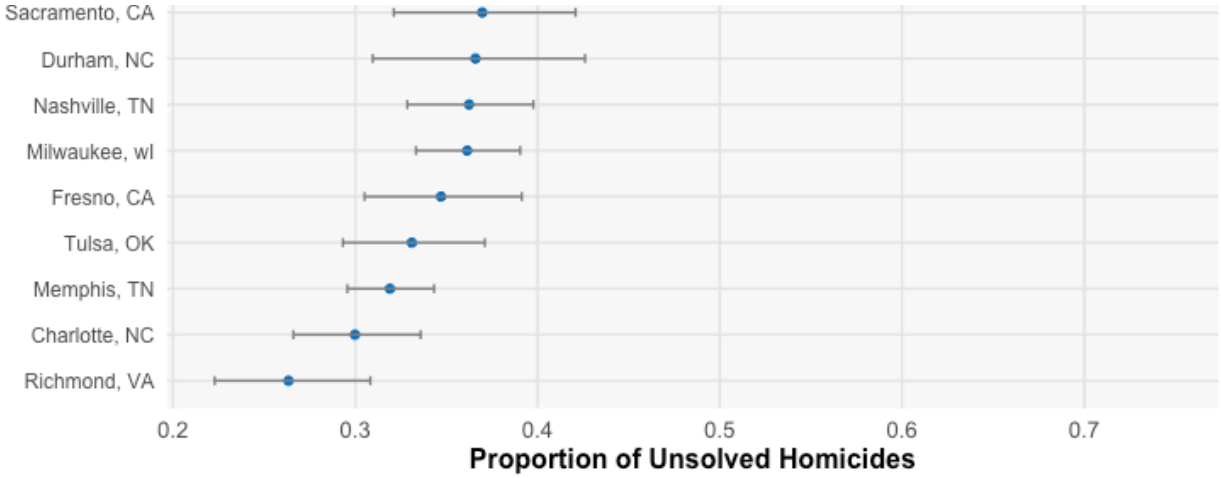
```
# Define a function to perform prop.test and tidy the output for each city
city_prop_test =
  function(total, unsolved) {
  result <- prop.test(unsolved, total)
  broom::tidy(result)
}

# Apply the function to each city and generate results for all cities
homicide_results =
  homicide_summary |>
  mutate(test_results = map2(total_homicides,
                             unsolved_homicides,
                             city_prop_test)) |>
  unnest(test_results) |>
  select(city_state,
         estimate,
```

```
          conf.low,
          conf.high)
```

Step 6: Plotting the Results with Enhanced Readability

```
# Reorder city_state by estimated proportion to make the plot more readable
homicide_results =
  homicide_results |>
  arrange(estimate) |>
  mutate(city_state = factor(city_state, levels = unique(city_state)))


# Create the plot showing the proportion of unsolved homicides with confidence
ggplot(homicide_results,
       aes(x = estimate,
           y = reorder(city_state, estimate))) +
  geom_point(color = "#0073C2") +
  geom_errorbar(aes(xmin = conf.low, xmax = conf.high),
                width = 0.2, color = "#808080") + #
  labs(
    title = "Proportion of Unsolved Homicides by City",
    x = "Proportion of Unsolved Homicides",
    y = "City, State"
  ) +
  theme_minimal(base_size = 12) +
  theme(
    plot.title = element_text(size = 14,
                              face = "bold",
                              hjust = 0.5),
    axis.text.y = element_text(size = 9,
                               color = "#4D4D4D"),
    axis.text.x = element_text(size = 10,
                               color = "#4D4D4D"),
    axis.title = element_text(face = "bold"),
    panel.grid.major = element_line(color = "#E5E5E5"),
    panel.grid.minor = element_blank(),
    panel.background = element_rect(fill = "#FAFAFA",
                                    color = NA)
  )
```


Proportion of Unsolved Homicides by City
City, State
Chicago, IL
New Orleans, LA
Baltimore, MD
San Bernardino, CA
Buffalo, NY
Miami, FL
Stockton, CA
Detroit, MI
Phoenix, AZ
Denver, CO
St. Louis, MO
Oakland, CA
Pittsburgh, PA
Columbus, OH
Jacksonville, FL
Minneapolis, MN
Houston, TX
San Francisco, CA
Boston, MA
Los Angeles, CA
Oklahoma City, OK
Dallas, TX
Savannah, GA
Fort Worth, TX
Baton Rouge, LA
Tampa, FL
Louisville, KY
Indianapolis, IN
Philadelphia, PA
Cincinnati, OH
Washington, DC
Birmingham, AL
San Antonio, TX
Las Vegas, NV
Omaha, NE
Long Beach, CA
Kansas City, MO
New York, NY
Albuquerque, NM
Atlanta, GA
San Diego, CA


Sacramento, CA
Durham, NC
Nashville, TN
Milwaukee, wI
Fresno, CA
Tulsa, OK
Memphis, TN
Charlotte, NC
Richmond, VA
0.2
0.3
0.4
0.5
0.6
0.7
Proportion of Unsolved Homicides