Project 1

Konstantinos Banos

2024-09-20

Table of contents

```r
# load libraries
library(tidyverse)
library(dotwhisker)
library(effects)
library(lemon)
library(MASS)
library(dplyr)
library(glmmTMB)
library(ggplot2)
library(performance)
library(AER)
library(caret)
library(see)
library(DHARMa)
library(patchwork)
library(mlmRev)
library(coefplot)
```

1. Dataset and modelling

```r
# Import and convert the data to tibble
olymp <- read.csv(url("https://raw.githubusercontent.com/bbolker/stats720/main/data/olymp1.cs
olymp <- as_tibble(olymp)  ## could use readr::read_csv directly

## frame, using within() or transform() or dplyr::mutate()
## Separate variables floating around in the workspace can easily
##  get out of sync
gdp_per_cap <- olymp$gdp/olymp$pop ## GDP per Capita
log_gdp <- log(olymp$gdp)          ## Logarithm of GDP
log_pop <- log(olymp$pop)          ## Logarithm of Population
log_gdp_perC <- log_gdp-log_pop    ## Logarithm of GDP per Capita
log_n <- log(olymp$n+1)            ## Logarithm of the Number of Medals



## Add six new columns using mutate to the olymp dataset

 olymp <- olymp %>%
  mutate(gdp_per_cap = gdp_per_cap,
         log_gdp = log_gdp,
         log_pop = log_pop,
         log_gdp_perC = log_gdp_perC,
         log_n = log_n)


mydata = olymp |> filter(medal == "Gold")  ## Filtering by "Gold" Metal and creation of a ne

head(mydata,10)
```

```
# A tibble: 10 x 11
   team   year medal     n    gdp   pop gdp_per_cap log_gdp log_pop log_gdp_perC
   <chr> <int> <chr> <int>  <dbl> <dbl>       <dbl>   <dbl>   <dbl>        <dbl>
 1 Afgh~  2000 Gold      0   6.21  19.5       0.318    1.83    2.97        -
1.15
 2 Afgh~  2004 Gold      0   7.98  23.6       0.339    2.08    3.16        -
1.08
 3 Afgh~  2008 Gold      0  11.1   26.4       0.419    2.40    3.27        -
0.871
 4 Afgh~  2012 Gold      0  17.4   30.5       0.571    2.86    3.42        -
0.561
 5 Afgh~  2016 Gold      0  19.6   34.6       0.565    2.97    3.54        -
0.571
 6 Alge~  2000 Gold      1 110.    30.8       3.57     4.70    3.43         1.27
 7 Alge~  2004 Gold      0 133.    32.5       4.08     4.89    3.48         1.41
 8 Alge~  2008 Gold      0 152.    34.6       4.40     5.02    3.54         1.48
 9 Alge~  2012 Gold      1 170.    37.3       4.57     5.14    3.62         1.52
10 Alge~  2016 Gold      0 195.    40.3       4.83     5.27    3.70         1.57
# i 1 more variable: log_n <dbl>
```

```r
head(olymp,10)
```

```
# A tibble: 10 x 11
   team    year medal     n   gdp   pop gdp_per_cap log_gdp log_pop log_gdp_perC
   <chr>  <int> <chr> <int> <dbl> <dbl>       <dbl>   <dbl>   <dbl>        <dbl>
 1 Afgha~  2000 Bron~     0  6.21  19.5       0.318    1.83    2.97        -
1.15
 2 Afgha~  2000 Gold      0  6.21  19.5       0.318    1.83    2.97        -
1.15
 3 Afgha~  2000 Silv~     0  6.21  19.5       0.318    1.83    2.97        -
1.15
```

```
 4 Afgha~  2004 Bron~     0  7.98  23.6        0.339    2.08    3.16        -
1.08
 5 Afgha~  2004 Gold      0  7.98  23.6        0.339    2.08    3.16        -
1.08
 6 Afgha~  2004 Silv~     0  7.98  23.6        0.339    2.08    3.16        -
1.08
 7 Afgha~  2008 Bron~     1 11.1   26.4        0.419    2.40    3.27        -
0.871
 8 Afgha~  2008 Gold      0 11.1   26.4        0.419    2.40    3.27        -
0.871
 9 Afgha~  2008 Silv~     0 11.1   26.4        0.419    2.40    3.27        -
0.871
10 Afgha~  2012 Bron~     1 17.4   30.5        0.571    2.86    3.42        -
0.561
# i 1 more variable: log_n <dbl>
```

In this analysis, the focus was on estimating gold medals from the `olymp1` dataset. After importing the dataset, it was transformed into a tibble format for more convenient handling. Five new variables were created by applying a logarithmic transformation to five key variables, reducing their dispersion and ensuring a smoother dataset for further analysis. Subsequently, a new tibble (`mydata`) was created, filtered to include only observations related to gold medals. This filtered dataset serves as the primary basis for the following analysis.

a. Model Complexity and Predictor Selection

In accordance with Harrell's guidelines [@harrell], it is recommended that the optimal number of predictor variables in a regression model should be less than $\frac{m}{15}$ (depends on the skewness of the distribution etc.), where m is the limiting sample size, defined as `nrow(mydata)`. Given that there are no strict limitations on the number of predictors in this analysis due to the sufficiently large sample size, I have opted to maintain a relatively simple and interpretable linear model.

For this analysis, I have selected two predictor variables (they are less correlated):

1. The natural logarithm of Gross Domestic Product (log_gdp).
2. The natural logarithm of the Population (log_pop).

The response variable of interest is the natural logarithm of gold medals ($\log_n$), which will be estimated in this model (it was necessary to do a transformation such that: $\log_n = \log(n+1)$, due to zeros in the dataset). This approach aims to facilitate interpretation while ensuring the model remains robust.

b. Variable Scaling, Transformations, and Interpretation Thresholds

The response variable in this analysis is the natural logarithm of the number of gold medals won by each country in a given year, denoted as `log_n`, where n represents the count of gold medals. This transformation is unitless, though it represents a logarithmic transformation of a count variable.

The two predictor variables used in the model are:

- The natural logarithm of Gross Domestic Product (GDP), denoted as log_gdp, where GDP is typically expressed in dollars.
- The natural logarithm of population, denoted as log_pop, with the original variable representing the count of people.

Both of the are basically unitless.

Given the nature of the problem and the logarithmic transformations applied to both the response and predictor variables, defining thresholds in terms of absolute units becomes challenging. For example, determining what constitutes a "small" change in GDP is difficult due to the variation in economic significance across countries.

As a result, I have adopted a threshold of 1% on each variable in the log scale. This approach provides a consistent way to interpret changes in the predictor variables and helps to identify significant relationships while managing noise in the model. Specifically, a 1% change in the log scale reflects the relative change in the original variable, offering a practical and scalable criterion for evaluating small changes in each predictor variable.

c. Model Specification

We applied the following model

```r
# original model
model <- lm(log_n ~ log_gdp+log_pop, data = mydata)

# View summary of the model
summary(model)
```

```

Call:
lm(formula = log_n ~ log_gdp + log_pop, data = mydata)

Residuals:
     Min       1Q   Median       3Q      Max
```

```
-2.08919 -0.65611 -0.07442  0.54266  2.82003

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.77938    0.10621  -7.338 8.03e-13 ***
log_gdp      0.38106    0.03119  12.216  < 2e-16 ***
log_pop     -0.01074    0.03575  -0.300    0.764
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9701 on 538 degrees of freedom
  (94 observations deleted due to missingness)
Multiple R-squared:  0.3604,    Adjusted R-squared:  0.358
F-statistic: 151.6 on 2 and 538 DF,  p-value: < 2.2e-16
```

The model summary indicates that approximately 36% of the variability in the response variable is explained by the predictors included in the model. The natural logarithm of GDP (log_gdp) emerges as a highly significant predictor, with a strong association with the response variable. In contrast, the natural logarithm of Population (log_pop) appears to be statistically insignificant and may not contribute meaningfully to the model.

However, before drawing definitive conclusions, it is essential to conduct diagnostic tests to evaluate the model's reliability and ensure that key assumptions, such as residual normality, homoscedasticity, and independence, are met. These diagnostics will provide a more comprehensive understanding of the model's validity and its potential limitations.

d. Model Diagnostics and Assessment of Assumptions

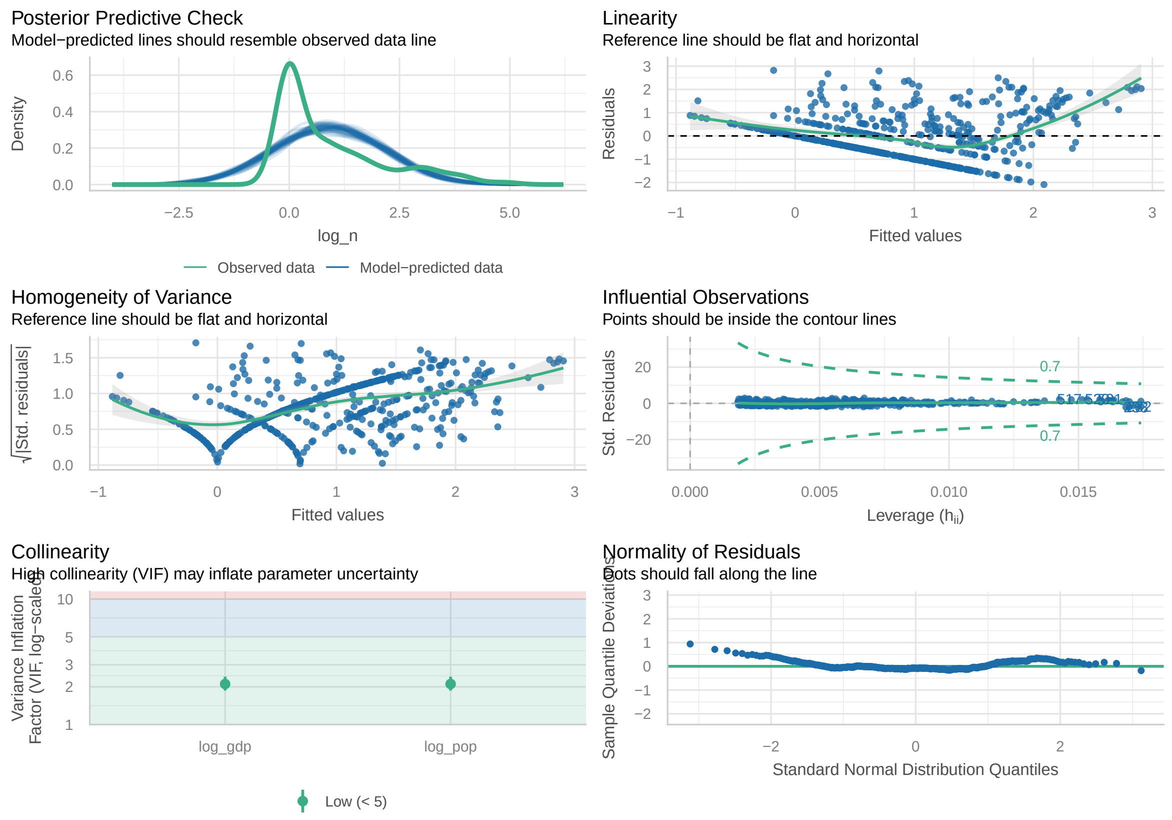
To assess the assumptions of our linear regression model, we examined several diagnostic plots.

1. **Residuals vs. Fitted**: The plot reveals a moderate V-shaped pattern, indicating a potential non-linear relationship between the predictors and the response variable. This suggests that the linear model may not adequately capture the underlying relationship.
2. **Q-Q Plot**: The Q-Q plot demonstrates deviations in the tails from the theoretical quantiles of the Normal distribution. While this suggests violations of the normality assumption, it is important to note that Q-Q plots are not always the most reliable method for assessing normality.
3. **Scale-Location Plot**: This plot indicates a clear upward trend, suggesting that the assumption of homoscedasticity (constant variance of the residuals) is violated. The presence of heteroscedasticity could impact the validity of the regression results.

4. **Posterior Predictive Plot**: The model's ability to simulate the distribution of the observed data is insufficient, as indicated by the discrepancies in the posterior predictive plot. This further underscores the limitations of the current model.

5. **Collinearity Assessment**: In contrast, the collinearity diagnostics suggest that there is no significant collinearity among the predictor variables. While collinearity is generally not a major concern for linear models, it is still a positive finding.

```r
# plot model evaluation graphs
performance::check_model(model)
```



In summary, the diagnostic plots indicate several violations of model assumptions, particularly regarding linearity, normality, and homoscedasticity, which may necessitate model reassessment or alternative modeling approaches.

e. Model Revision and Selection of an Appropriate Count Regression

The initial linear regression model was found to be inappropriate for the data at hand. The response variable, the number of medals (n), is a count variable, and using a linear model to predict count data can lead to biased estimates and violations of key assumptions, as evidenced by the diagnostic plots.

Given the nature of the response variable, I have opted for a Negative Binomial (NB) regression model. The Negative Binomial model is particularly well-suited for count data and is commonly used when the data exhibits over-dispersion—a scenario where the variance exceeds the mean. Over-dispersion appears to be a significant issue in this case, making the NB regression an appropriate choice.

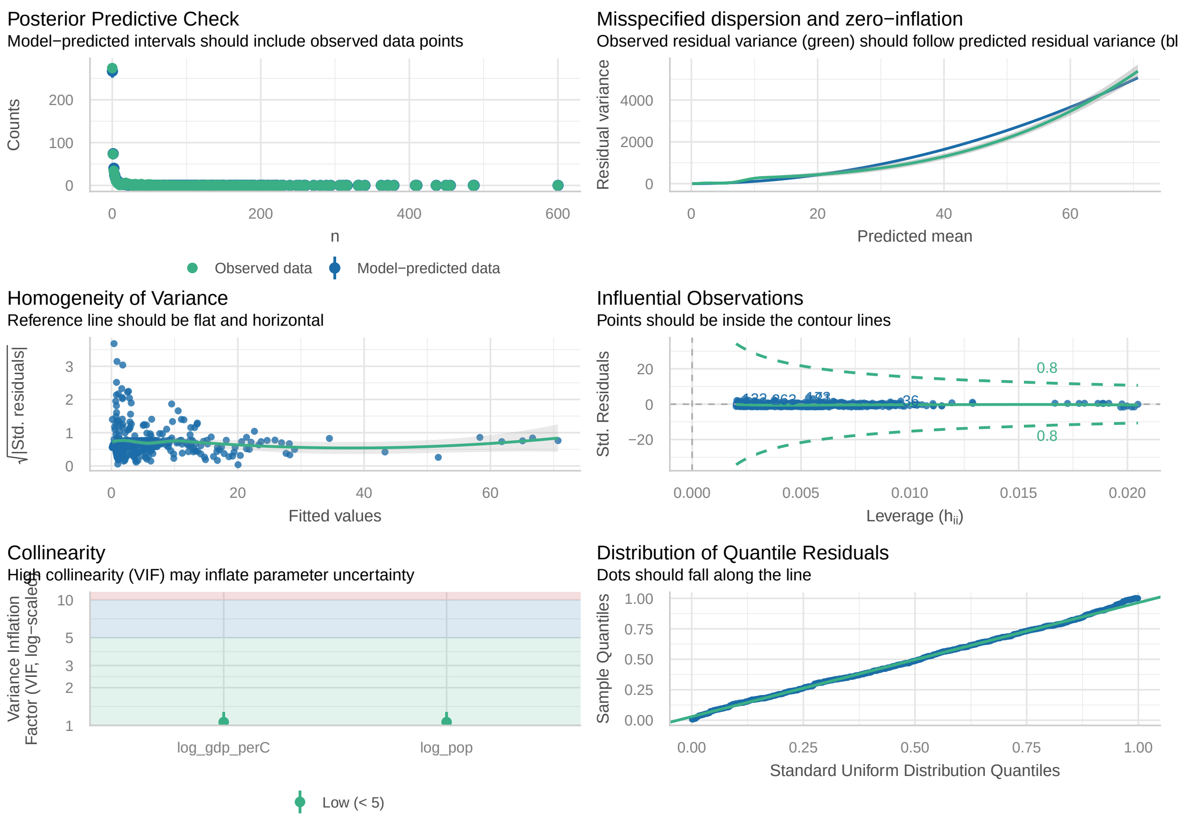
In the NB regression model, I will use the original response variable (n) rather than its logarithmic transformation. The predictors will be the natural logarithm of GDP per capita (log_gdp_perC) and the natural logarithm of population (log_pop). This approach allows for a better fit of the data and addresses the limitations observed in the initial linear model, particularly in handling the over-dispersion inherent in the data.

```r
## Test for Over-dispersion : dispersiontest(nb_model)

# Fit Negative Binomial regression model
nb_model <- glm.nb(n ~ log_pop + log_gdp_perC, data = mydata)

# View summary of the nb_model
summary(nb_model)
```

```
Call:
glm.nb(formula = n ~ log_pop + log_gdp_perC, data = mydata, init.theta = 0.3548300421,
    link = log)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.90041    0.22471  -8.457   <2e-16 ***
log_pop       0.61990    0.04987  12.431   <2e-16 ***
log_gdp_perC  0.63563    0.06174  10.295   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Negative Binomial(0.3548) family taken to be 1)

    Null deviance: 795.35  on 540  degrees of freedom
Residual deviance: 492.81  on 538  degrees of freedom
  (94 observations deleted due to missingness)
AIC: 2230.6
```

```
Number of Fisher Scoring iterations: 1

              Theta:  0.3548
          Std. Err.:  0.0312

 2 x log-likelihood:  -2222.5810
```

```r
# plot nb_model evaluation graphs
performance::check_model(nb_model)
```



```r
### check for collinearity
### vif less than 3 indicate no significant
vif(nb_model)
```

```
  log_pop log_gdp_perC
 1.062566     1.062566
```

```r
## Pseudo R^2
performance::r2(nb_model)
```

```
# R2 for Generalized Linear Regression
  Nagelkerke's R2: 0.556
```

```r
## Predictive Performance (Cross-validation)
## train_control <- trainControl(method="cv", number=10)
## train(ln ~ log_gdp_perC + log_pop, data=mydata, method="glm.nb", trControl=train_control)
```

Looking at the summary results, the model reveals that both population size and GDP per capita are significant predictors of Olympic gold medal success. Specifically, a 1% increase in population is associated with approximately a 0.62% increase in the expected number of gold medals, while a 1% increase in GDP per capita corresponds to about a 0.64% increase in expected medals. These findings suggest that countries with larger populations and higher economic resources are more likely to perform well in the Olympics, underscoring the importance of demographic and economic factors in athletic achievement. Moreover, theta = 0.3548, which is something that creates evidence for the over-dispersion nature of data.

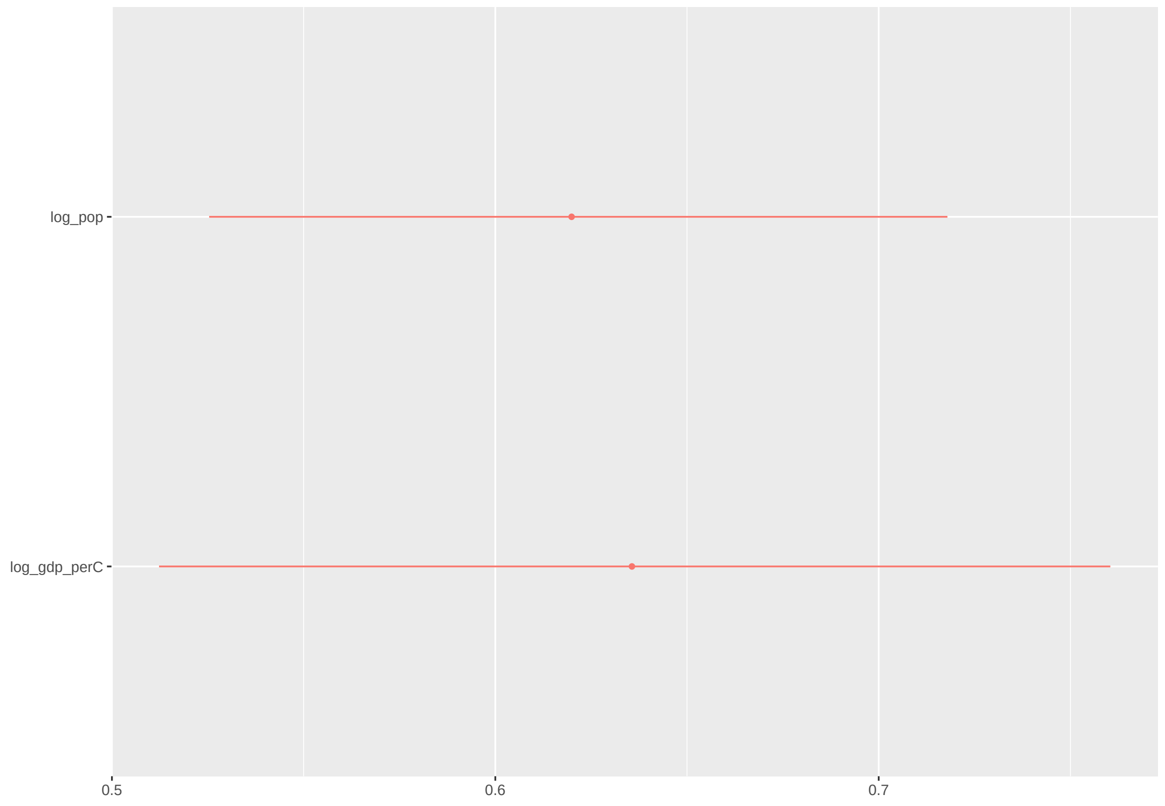
The diagnostic graphs provide valuable insights into the model's performance. The Q-Q plot indicates a relatively strong fit, suggesting that the residuals conform closely to a normal distribution. One critical assumption of the model is the absence of influential observations, which is confirmed by the analysis of influential observations. While there are some minor discrepancies between the observed and predicted residual variances, the overall model fit remains satisfactory.

Furthermore, the analysis shows no signs of collinearity among the predictor variables. The results of the posterior predictive checks are also favorable. However, the robustness of these findings may warrant further investigation, as the overall goodness of fit raises the possibility of potential overfitting or other specification issues that should be explored to ensure the reliability of the model.

f. Coefficient Plot

To draw a coefficient plot:

```r
# model coefficient plot
dwplot(x=nb_model)
```



I have chosen not to scale and center the predictors because we have only two quantitative variables, which allows for straightforward interpretation of the coefficients.

g. Effect Plot

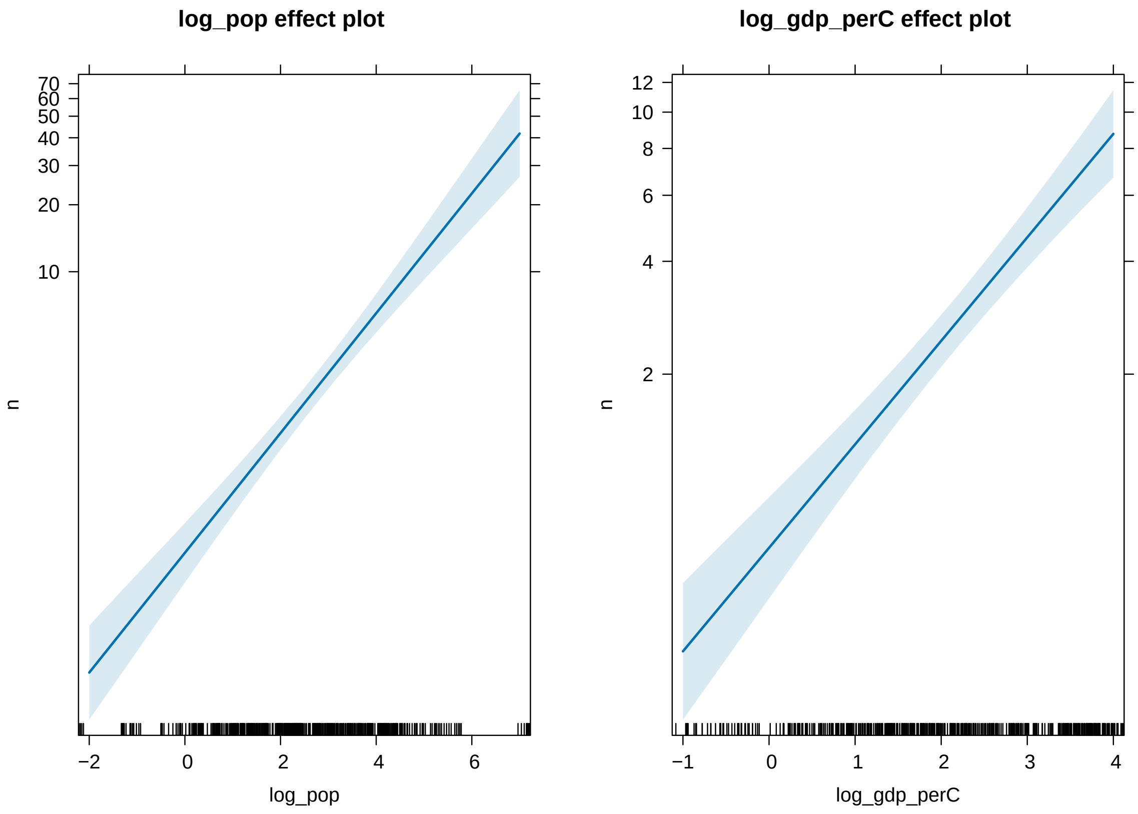
To draw an effects plot, we have:

```r
# Plot an effect plot and Effects of Predictors on the Model
effects::allEffects(nb_model)
```

```
 model: n ~ log_pop + log_gdp_perC

 log_pop effect
log_pop
        -2        0.1          3          5          7 
 0.1577998  0.5800481  3.5010293 12.0957119 41.7894949 

 log_gdp_perC effect
```

```
log_gdp_perC
       -1       0.1         2         3         4
0.3645483 0.7335204 2.4542054 4.6340731 8.7501368
```

```r
plot(allEffects(nb_model))
```



As the logarithm of population increases, the expected number of gold medals (n) increases significantly, demonstrating a strong positive relationship. Similarly, as the logarithm of GDP per capita increases, the expected number of gold medals also increases, indicating that higher economic resources are associated with better Olympic performance. These effects illustrate the influence of population size and economic capacity on the success of countries in the Olympic Games.

2: Contrasts

Since we have $\overline{y} = C\beta$ where C is our contrasts matrix, $\beta = C^{-1}\overline{y}$.

Below is our contrasts matrix as requested in the question:

```r
# Define the inverse of contrast matrix
c_inv <- matrix(c(
  -1, 1/3, 1/3, 1/3,      ## C vs Average of Treatments
   0, 1, -1, 0,           ## I vs II
   0, 0, 1, -1,           ## II vs III
   1, 0, 0, 0),           ## Treat Control as an Intercept
  nrow=4,byrow=TRUE)

MASS::fractions(c_inv)
```

```
     [,1] [,2] [,3] [,4]
[1,]  -1  1/3  1/3  1/3
[2,]   0    1   -1    0
[3,]   0    0    1   -1
[4,]   1    0    0    0
```

In this part we are gonna define the contrast matrix and creating a sample to employ the model

```r
# inverse contrasts matrix
c_mat <- solve((c_inv))
MASS::fractions(c_mat)
```

```
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    1
[2,]    1  2/3  1/3    1
[3,]    1 -1/3  1/3    1
[4,]    1 -1/3 -2/3    1
```

```r
data = data.frame(
  treatment = factor(c("C", "I", "II", "III")),
  response = c(2,4,6,8)
)
print(data())
```

At this point, I applied the contrast matrix to the data set

```r
model = lm(response ~ treatment, contrasts = list(treatment = c_mat[,-4]), data = data)
coef(summary(model))
```

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept)        2        NaN     NaN      NaN
treatment1         4        NaN     NaN      NaN
treatment2        -2        NaN     NaN      NaN
treatment3        -2        NaN     NaN      NaN
```

The estimated coefficients from the model provide insights into the differences between the control and the treatment groups. Specifically:

- The coefficient for the control group is **2**, serving as the baseline for comparison.
- The coefficient for the overall effect of the treatments (the average of all treatments) is **4**, indicating that, on average, the treatments differ from the control group by **4 units**.
- The difference between Treatment 1 and Treatment 2 is represented by a coefficient of **-2**, suggesting that Treatment 2 results in **2 units** less than Treatment 1, on average.
- The difference between Treatment 2 and Treatment 3 is also represented by a coefficient of **-2**, meaning that Treatment 3 results in **2 units** less than Treatment 2, on average.

These coefficients reflect the relative differences between the control and treatment groups, as well as the differences among the treatments themselves. The results was completely expected.

3. Simulation exercises misspecification problems in Linear Regression

```r
sim_fun <- function(n = 100, slope = 1, s = 1, m = 0, df = 10) {
  # Generate predictor
  x <- runif(n)
  # Mean structure: E[y | x] = m + slope * x
  mu <- m + slope * x
  # t-distributed errors with mean 0 and scale s
  e <- s * rt(n, df = df)
  # Response
  y <- mu + e
  data.frame(x = x, y = y)
}


run_simulations <- function(df, num_simulations = num_simulations) {
  true_slope <- 1  # True slope value

  results <- data.frame(
    bias =  numeric(num_simulations),    ## data frame creation for later usage
    rmse = numeric(num_simulations),
    coverage = numeric(num_simulations),
    power = numeric(num_simulations)
    )

  for (i in 1:num_simulations) {
    sim_data <- sim_fun(n = 100, slope = 1, s = 1, m = 0, df = df)

    model <- lm(y ~ x, data = sim_data)

    # Validating the model and the results
    if (is.na(summary(model)$coefficients["x", "Estimate"])) {
      # Invalid model results
      results[i, ] <- c(NA, NA, NA)
    } else {
      #coefficients
      coef_summary <- coef(summary(model))

      ##  RMSE
      rmse <- sqrt(mean((coef_summary["x", "Estimate"] - true_slope)^2))

      ## Bias
      bias <- coef_summary["x", "Estimate"] - true_slope

      # power (p < alpha)
      alpha <- 0.05
      p_value <- coef_summary["x", "Pr(>|t|)"]
      power <- mean(p_value < alpha)

      # coverage
      conf_interval <- confint(model)
      coverage <- (true_slope >= conf_interval["x", 1] & true_slope <= conf_interval["x", 2]

      results[i, ] <- c(bias, rmse, coverage, power)
    }
  }

  return(results)
}
```

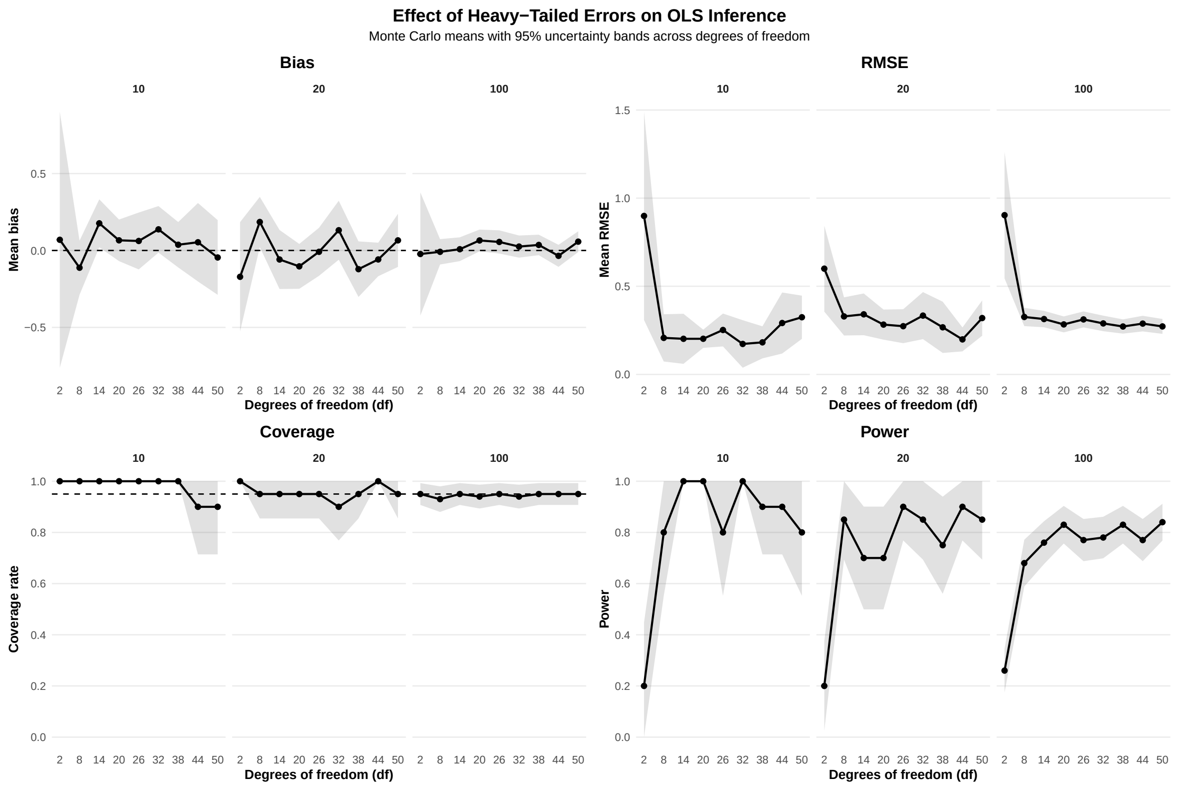
```r
set.seed(123)

deg <- seq(2, 50, by = 6)
n_vec <- c(10, 20, 100)

all_results <- do.call(rbind, lapply(n_vec, function(nsim) {
  do.call(rbind, lapply(deg, function(df_val) {
    res <- run_simulations(df = df_val, num_simulations = nsim)

    # add identifiers so you can group later
    res$n_sim <- nsim
    res$df <- df_val
    res$sim_id <- seq_len(nrow(res))

    res
  }))
}))

# str(all_results)
# head(all_results)



summary_results <- all_results %>%
  group_by(n_sim, df) %>%
  summarise(
    bias_mean = mean(bias, na.rm = TRUE),
    bias_sd   = sd(bias, na.rm = TRUE),
    rmse_mean = mean(rmse, na.rm = TRUE),
    rmse_sd   = sd(rmse, na.rm = TRUE),
    coverage_rate = mean(coverage, na.rm = TRUE),  # should approach ~0.95 if calibrated
    power_rate    = mean(power, na.rm = TRUE),
    .groups = "drop"
  )

head(summary_results,10)
```

```
# A tibble: 10 x 8
   n_sim    df bias_mean bias_sd rmse_mean rmse_sd coverage_rate power_rate
   <dbl> <dbl>     <dbl>   <dbl>     <dbl>   <dbl>         <dbl>      <dbl>
 1    10     2    0.0695   1.34      0.899   0.953             1        0.2
 2    10     8   -0.112    0.284     0.207   0.216             1        0.8
 3     10     14     0.177     0.251      0.202  0.229           1           1
 4     10     20     0.0660    0.218      0.202  0.0834          1           1
 5     10     26     0.0617    0.298      0.252  0.151           1           0.8
 6     10     32     0.137     0.244      0.173  0.218           1           1
 7     10     38     0.0373    0.238      0.182  0.147           1           0.9
 8     10     44     0.0534    0.412      0.292  0.279           0.9         0.9
 9     10     50    -0.0456    0.392      0.324  0.198           0.9         0.8
10     20      2    -0.171     0.811      0.600  0.556           1           0.2
```

```r
# --- Add uncertainty bands (95% CI around the Monte Carlo mean) ---
summary_results2 <- summary_results %>%
  mutate(
    bias_se = bias_sd / sqrt(n_sim),
    rmse_se = rmse_sd / sqrt(n_sim),

    bias_lo = bias_mean - 1.96 * bias_se,
    bias_hi = bias_mean + 1.96 * bias_se,

    rmse_lo = rmse_mean - 1.96 * rmse_se,
    rmse_hi = rmse_mean + 1.96 * rmse_se,

    # conservative SE for proportions (coverage/power)
    cov_se = sqrt(pmax(coverage_rate * (1 - coverage_rate), 0) / n_sim),
    pow_se = sqrt(pmax(power_rate * (1 - power_rate), 0) / n_sim),

    cov_lo = pmax(0, coverage_rate - 1.96 * cov_se),
    cov_hi = pmin(1, coverage_rate + 1.96 * cov_se),

    pow_lo = pmax(0, power_rate - 1.96 * pow_se),
    pow_hi = pmin(1, power_rate + 1.96 * pow_se)
  )
```

```r
theme_portfolio <- theme_minimal(base_size = 12) +
  theme(
    plot.title.position = "plot",
    plot.title = element_text(face = "bold", size = 15, hjust = 0.5),
    axis.title = element_text(face = "bold"),
    strip.text = element_text(face = "bold"),
    panel.grid.minor = element_blank(),
    panel.grid.major.x = element_blank()
  )

```

```r
p_bias <- ggplot(summary_results2, aes(x = df, y = bias_mean)) +
  geom_hline(yintercept = 0, linetype = "dashed", linewidth = 0.6) +
  geom_ribbon(aes(ymin = bias_lo, ymax = bias_hi), alpha = 0.15) +
  geom_line(linewidth = 0.9) +
  geom_point(size = 2) +
  facet_wrap(~ n_sim, nrow = 1) +
  scale_x_continuous(breaks = unique(summary_results2$df)) +
  labs(x = "Degrees of freedom (df)", y = "Mean bias",
       title = "Bias") +
  theme_portfolio


p_rmse <- ggplot(summary_results2, aes(x = df, y = rmse_mean)) +
  geom_ribbon(aes(ymin = rmse_lo, ymax = rmse_hi), alpha = 0.15) +
  geom_line(linewidth = 0.9) +
  geom_point(size = 2) +
  facet_wrap(~ n_sim, nrow = 1) +
  scale_x_continuous(breaks = unique(summary_results2$df)) +
  labs(x = "Degrees of freedom (df)", y = "Mean RMSE",
       title = "RMSE") +
  theme_portfolio


p_cov <- ggplot(summary_results2, aes(x = df, y = coverage_rate)) +
  geom_hline(yintercept = 0.95, linetype = "dashed", linewidth = 0.6) +
  geom_ribbon(aes(ymin = cov_lo, ymax = cov_hi), alpha = 0.15) +
  geom_line(linewidth = 0.9) +
  geom_point(size = 2) +
  facet_wrap(~ n_sim, nrow = 1) +
  scale_x_continuous(breaks = unique(summary_results2$df)) +
  scale_y_continuous(limits = c(0, 1), breaks = seq(0, 1, 0.2)) +
  labs(x = "Degrees of freedom (df)", y = "Coverage rate",
       title = "Coverage") +
  theme_portfolio


p_power <- ggplot(summary_results2, aes(x = df, y = power_rate)) +
  geom_ribbon(aes(ymin = pow_lo, ymax = pow_hi), alpha = 0.15) +
  geom_line(linewidth = 0.9) +
  geom_point(size = 2) +
  facet_wrap(~ n_sim, nrow = 1) +
  scale_x_continuous(breaks = unique(summary_results2$df)) +
  scale_y_continuous(limits = c(0, 1), breaks = seq(0, 1, 0.2)) +
  labs(x = "Degrees of freedom (df)", y = "Power",
       title = "Power") +
  theme_portfolio



(p_bias | p_rmse) /
(p_cov  | p_power) +
  plot_annotation(
    title = "Effect of Heavy-Tailed Errors on OLS Inference",
    subtitle = "Monte Carlo means with 95% uncertainty bands across degrees of freedom",
    theme = theme(
      plot.title = element_text(face = "bold", size = 16, hjust = 0.5),
      plot.subtitle = element_text(size = 12, hjust = 0.5)
    )
  )
```



In this set of simulations, I tested the effects of violating the normality assumption by generating data from a t-distribution with varying degrees of freedom (df). As the degrees of freedom increase, the t-distribution converges to the normal distribution, theoretically satisfying the normality assumption when df$\rightarrow\infty$.

As we expected, due to the converges nature of T distribution to the normal distribution, as we increase the number of simulations to converge to the real value and when we increase the df the violation is eliminated. For example, when evaluating model coverage, with the true coverage rate set at 95%, the results from the simulation demonstrate that the estimated coverage rate converges to 95% as df increases. Additionally, both the bias and the root mean square error (RMSE) exhibit convergence; the bias tends towards 0, and the RMSE stabilizes at approximately 0.27.