MTHM506: Statistical Data Modelling

I acknowledge the use of ChatGPT(chat.openai.com) to generate materials that I have adapted to include within my final assessment. I confirm that no content generated by AI has been presented as my own work.

Question 1a

The model is defined as:

$$Y_i \sim N\left(\frac{\theta_1 x_i}{\theta_2 + x_i}, \sigma^2\right)$$

for $i = 1, 2, \ldots, 100$, where each Y_i is independent.

The model inherently exhibits non-linearity due to the ratio $\frac{\theta_1 x_i}{\theta_2 + x_i}$ in the mean of Y_i. This ratio means that the relationship between Y_i and x_i is not straightforward or constant; instead, it involves both x_i and the parameters θ_1 and θ_2 in a way that defies the linear model's principle of having a direct, proportional effect of x_i on Y_i. The presence of x_i in both the numerator and denominator creates a complex interaction that alters the effect of x_i on the outcome in a non-linear fashion, as the impact of a change in x_i on the mean of Y_i varies depending on the values of x_i, θ_1, and θ_2. This model's structure, therefore, does not support a constant rate of change or a simple additive relationship, making it inherently non-linear.

Question 1b

To find the likelihood, first, we must find the probability density function which is given by:

$$f(y_i; \theta_1, \theta_2, \sigma^2, x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \frac{\theta_1 x_i}{\theta_2 + x_i})^2}{2\sigma^2}\right)$$

Now we have that, the likelihood can be found which is just the product of the probability density function.

$$L(\theta_1, \theta_2, \sigma^2; y, x) = \prod_{i=1}^{100} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \frac{\theta_1 x_i}{\theta_2 + x_i})^2}{2\sigma^2}\right)$$

Now I have the likelihood I can find the log likelihood which is given by :

$$\ell(\theta_1, \theta_2, \sigma^2; y_1, y_2, \ldots, y_n) = \sum_{i=1}^{100} \log\left(\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) \exp\left(-\frac{(y_i - \frac{\theta_1 x_i}{\theta_2 + x_i})^2}{2\sigma^2}\right)\right)$$

To simplify the expression I will split it in two with the first half being:

$$\sum_{i=1}^{100} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)$$

We first recognise that the term inside the logarithm does not depend on the index i, allowing us to multiply the logarithm by the number of terms in the sum, which is 100:

$$100 \cdot \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)$$

Next, we apply the property of logarithms that $\log\left(\frac{1}{x}\right) = -\log(x)$, to invert the fraction inside the logarithm:

$$-100 \cdot \log\left(\sqrt{2\pi\sigma^2}\right)$$

Since $\log(\sqrt{x}) = \frac{1}{2}\log(x)$, we can simplify the square root inside the logarithm:

$$-100 \cdot \frac{1}{2}\log(2\pi\sigma^2) = -50 \cdot \log(2\pi\sigma^2)$$

Thus, the simplified form of the original sum is:

$$\sum_{i=1}^{100} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) = -50 \cdot \log(2\pi\sigma^2)$$

Alternatively it can be written as

$$\sum_{i=1}^{100} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) = \sum_{i=1}^{100} -1/2 \cdot \log(2\pi\sigma^2)$$

This simplification demonstrates the use of logarithmic identities to consolidate the sum of logarithms into a single term.

Now onto the second part of the expression. Log and exp and inverse operations so this is easily done:

$$\sum_{i=1}^{100} \log\left(\exp\left(-\frac{(y_i - \frac{\theta_1 x_i}{\theta_2 + x_i})^2}{2\sigma^2}\right)\right) = \sum_{i=1}^{100}\left(-\frac{(y_i - \frac{\theta_1 x_i}{\theta_2 + x_i})^2}{2\sigma^2}\right)$$

Now both parts have been simplified it can be put back together:

$$\ell(\theta_1, \theta_2, \sigma^2; y, x) = \sum_{i=1}^{100}\left(-1/2 \cdot \log(2\pi\sigma^2)\right)\left(-\frac{(y_i - \frac{\theta_1 x_i}{\theta_2 + x_i})^2}{2\sigma^2}\right)$$

Question 1c

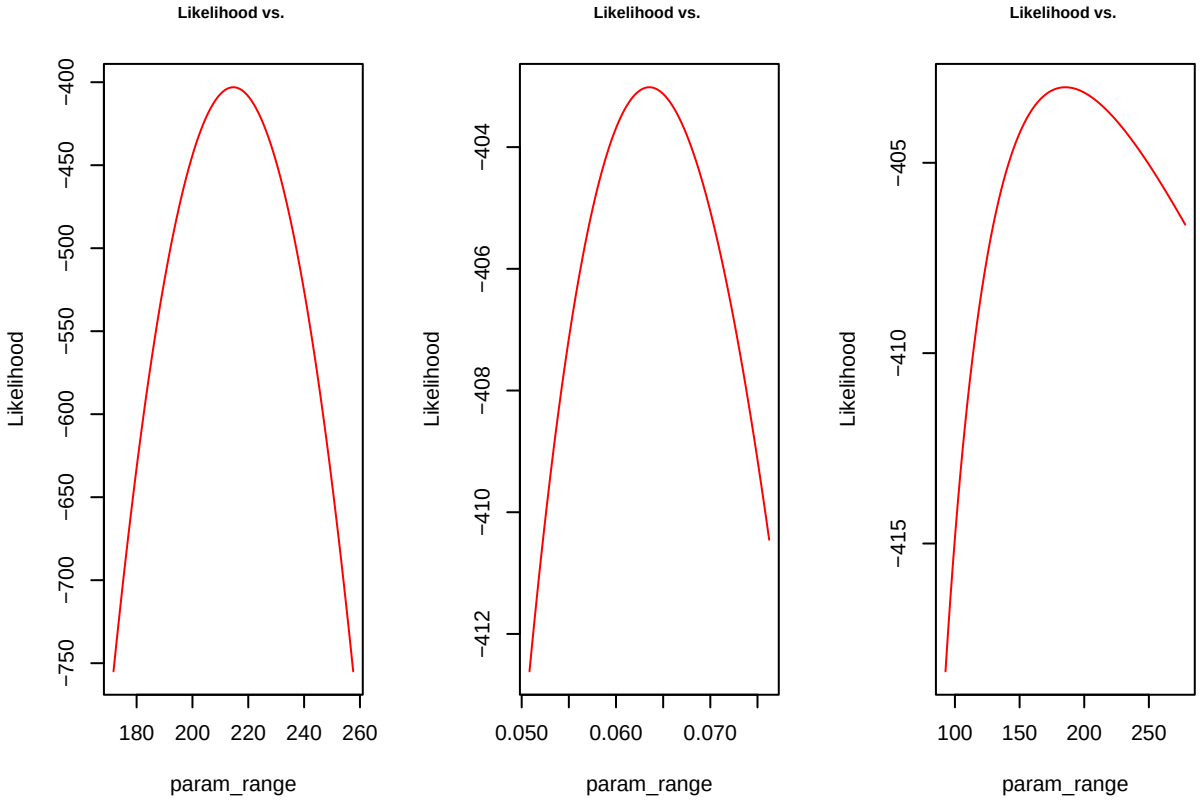
```r
mylike <- function(params, y, x){
  # Extract parameters from the input vector
  theta1 <- params[1] # First parameter
  theta2 <- params[2] # Second parameter
  sigma2 <- params[3] # Third parameter, variance

  # Check if variance is less than or equal to 0, return Inf as error state
  if (sigma2 <= 0) {
    return(Inf)
  }
```

```r
  # Get the number of observations
  n <- length(y)
  # Initialize a vector to store calculated means for each observation
  mu <- numeric(n)
  # Calculate the mean for each observation based on the model equation
  for (i in 1:n) {
    mu[i] <- theta1 * x[i] / (theta2 + x[i])
  }

  # Compute the log-likelihood for the normal distribution
  logL <- sum(-1/2*log(2*pi*sigma2) - ((y-mu)^2) / (2*sigma2))

  # Return the negative log-likelihood to set up for minimization
  -logL
}
```

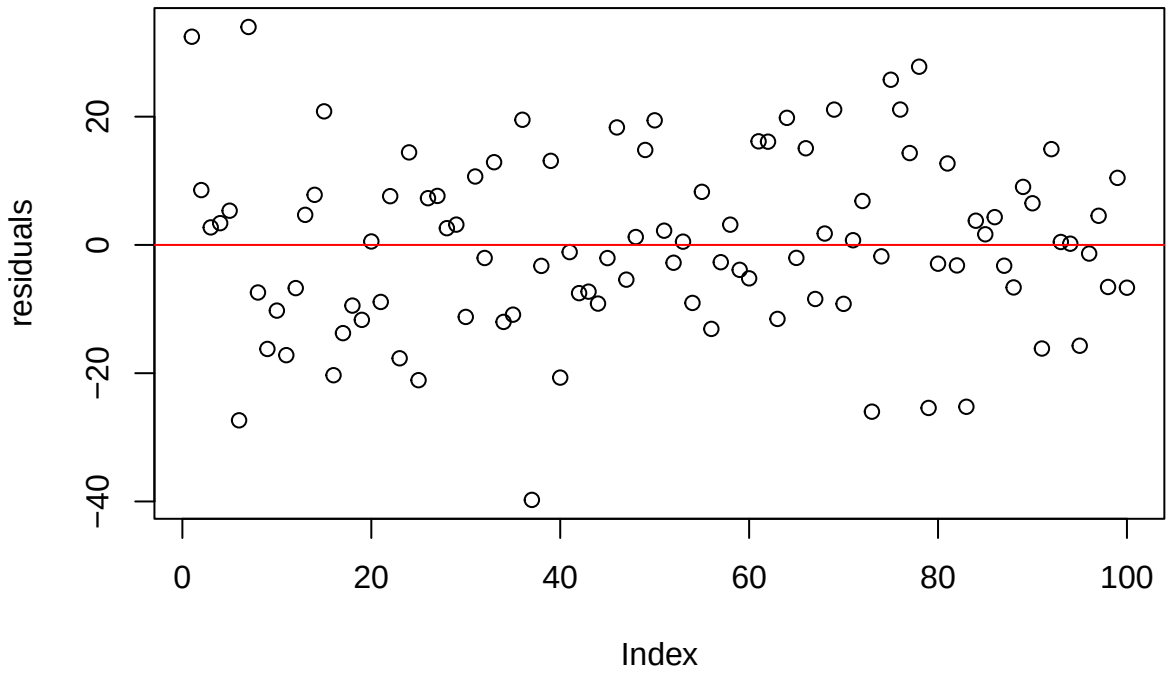
Question 1d

Having the available data enabled me to employ the method of moments to derive an estimate for the variance (σ^2), which served as the initial guess for optimization. I left the other two parameters at zero. Subsequently, through examining the plots of likelihod, it is evident that the estimates are robust. Additional evidence supporting this conclusion includes the observation of a very low gradient and the relatively small number of iterations required by the nlm function during optimization.

```
## [1] "Optimized parameter values: 214.650095582585"
## [2] "Optimized parameter values: 0.0635344784636584"
## [3] "Optimized parameter values: 185.385865181733"

## [1] "Exit code from the optimizer: 1"

## [1] "The minimum value of the objective function (negative log-likelihood) was: 403.015823659113"

## [1] "Gradient of the objective function at the solution: 5.90255019244139e-08"
## [2] "Gradient of the objective function at the solution: -2.84978796116775e-05"
## [3] "Gradient of the objective function at the solution: -7.54290572587658e-10"

## [1] "Number of iterations taken to converge: 45"
```


Likelihood vs.
Likelihood
−400
−450
−500
−550
−600
−650
−700
−750
180
200
220
240
260
param_range
Likelihood vs.
Likelihood
−404
−406
−408
−410
−412
0.050
0.060
0.070
param_range
Likelihood vs.
Likelihood
−405
−410
−415
100
150
200
250
param_range

Question 1e




```
# Extract the Hessian matrix from the optimization result
hessian_matrix <- result$hessian

# Invert the Hessian matrix to obtain the variance-covariance matrix of the
#parameter estimates
var_cov_matrix <- solve(hessian_matrix)

# Compute standard errors for the parameter estimates (square roots of diagonal
#elements of the var-cov matrix)
standard_errors <- sqrt(diag(var_cov_matrix))

# Print standard errors for each parameter
cat("Standard Error for theta_1:", standard_errors[1], "\n. ")
```

```
## Standard Error for theta_1: 2.674799
## .
```

```
cat("Standard Error for theta_2:", standard_errors[2], "\n. ")
```

```
## Standard Error for theta_2: 0.005140381
## .
```

```r
cat("Standard Error for sigma^2:", standard_errors[3], "\n. ") # Corrected to
```

```
## Standard Error for sigma^2: 26.2231
## .
```

```r
#\\sigma^2 for clarity


# Extract parameter estimates and their standard errors
theta1_estimate <- result$estimate[1]
theta2_estimate <- result$estimate[2]
sigma2_estimate <- result$estimate[3]
SE_theta1 <- standard_errors[1]
SE_theta2 <- standard_errors[2]
SE_sigma2 <- standard_errors[3]
z_99 <- 2.576 # Z-value for 99% confidence level

# Calculate and print 99% confidence intervals for each parameter
CI_theta1_lower <- theta1_estimate - (z_99 * SE_theta1)
CI_theta1_upper <- theta1_estimate + (z_99 * SE_theta1)
CI_theta2_lower <- theta2_estimate - (z_99 * SE_theta2)
CI_theta2_upper <- theta2_estimate + (z_99 * SE_theta2)
CI_sigma2_lower <- sigma2_estimate - (z_99 * SE_sigma2)
CI_sigma2_upper <- sigma2_estimate + (z_99 * SE_sigma2)

# Print confidence intervals
cat("99% Confidence Interval for theta_1: [", CI_theta1_lower, ",", CI_theta1_upper, "]\n. ")
```

```
## 99% Confidence Interval for theta_1: [ 207.7598 , 221.5404 ]
## .
```

```r
cat("99% Confidence Interval for theta_2: [", CI_theta2_lower, ",", CI_theta2_upper, "]\n. ")
```

```
## 99% Confidence Interval for theta_2: [ 0.05029286 , 0.0767761 ]
## .
```

```r
cat("99% Confidence Interval for sigma^2: [", CI_sigma2_lower, ",", CI_sigma2_upper, "]\n. ")
```

```
## 99% Confidence Interval for sigma^2: [ 117.8352 , 252.9366 ]
## .
```

```r
# Corrected to \\sigma^2 for clarity
```

Question 1f

```r
# Given data
theta2_hypothesized <- 0.08
# Calculate the z statistic
z_statistic <- (theta2_estimate - theta2_hypothesized) / SE_theta2
# Critical z values for 10% significance level (two-tailed test)
z_critical <- 1.645 # For one-tailed, but since it's two-tailed, we use +/- this value
# Print the z statistic
cat("z Statistic:", z_statistic, "\n")
```

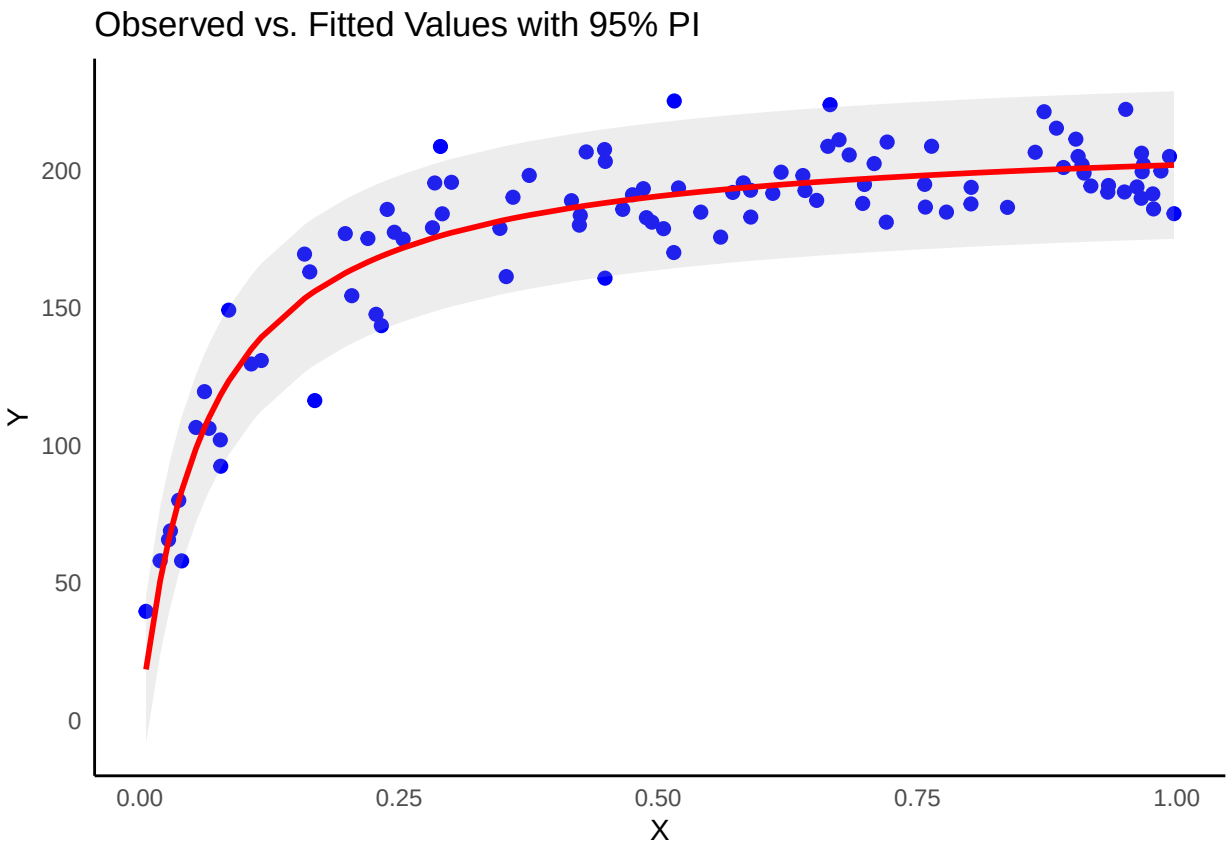
```
## z Statistic: -3.203171
```

```r
# Decision rule
if(abs(z_statistic) > z_critical) {
  cat("Reject H0: The test statistic falls outside the critical region.\n")
} else {
  cat("Fail to reject H0: The test statistic falls within the critical region.\n")
}
```

```
## Reject H0: The test statistic falls outside the critical region.
```

At a significance level of 10%, a z-value is calculated based on the observed value (theta 2 estimate) and the hypothesized value of 0.08. This z-value indicates the magnitude of deviation from the hypothesized value in terms of standard deviations. With a z-value exceeding the critical threshold for a 10% significance level, it is statistically shown that the observed value is significantly different from the hypothesized value of 0.08. Therefore, based on this analysis, it is unlikely that the true value of the parameter is 0.08.

Question 1g



The predicted values closely follow the general trend of the data, indicating that the model effectively captures the relationship between the independent variable (x) and the dependent variable (y). Additionally, the 95% prediction interval encompasses the majority of the data points, suggesting that the models predictions are associated with a reasonable level of uncertainty. However, its worth noting that some data points fall outside the prediction interval, indicating that the model may not capture all the variability in the data. Despite this, overall, the model appears to be a good fit, capturing most of the variability and making reliable predictions. Therefore, it can be considered an appropriate model for the given data

Question 2a

Model 1: While a Poisson regression initially appears suitable for the count-based nature of the dataset, the significant overdispersion observed—where the variance greatly exceeds the mean—contradicts a fundamental assumption of the Poisson distribution, that the mean and variance should be approximately equal. This discrepancy suggests that the Poisson model may not adequately capture the data's underlying variability. In light of this, alternative approaches, such as Negative Binomial regression or a quasi-Poisson model, could offer a more appropriate fit. These models are specifically designed to handle overdispersion, providing a more accurate and robust framework for analyzing count data that deviates from the strict expectations of a Poisson distribution.

Model 2:

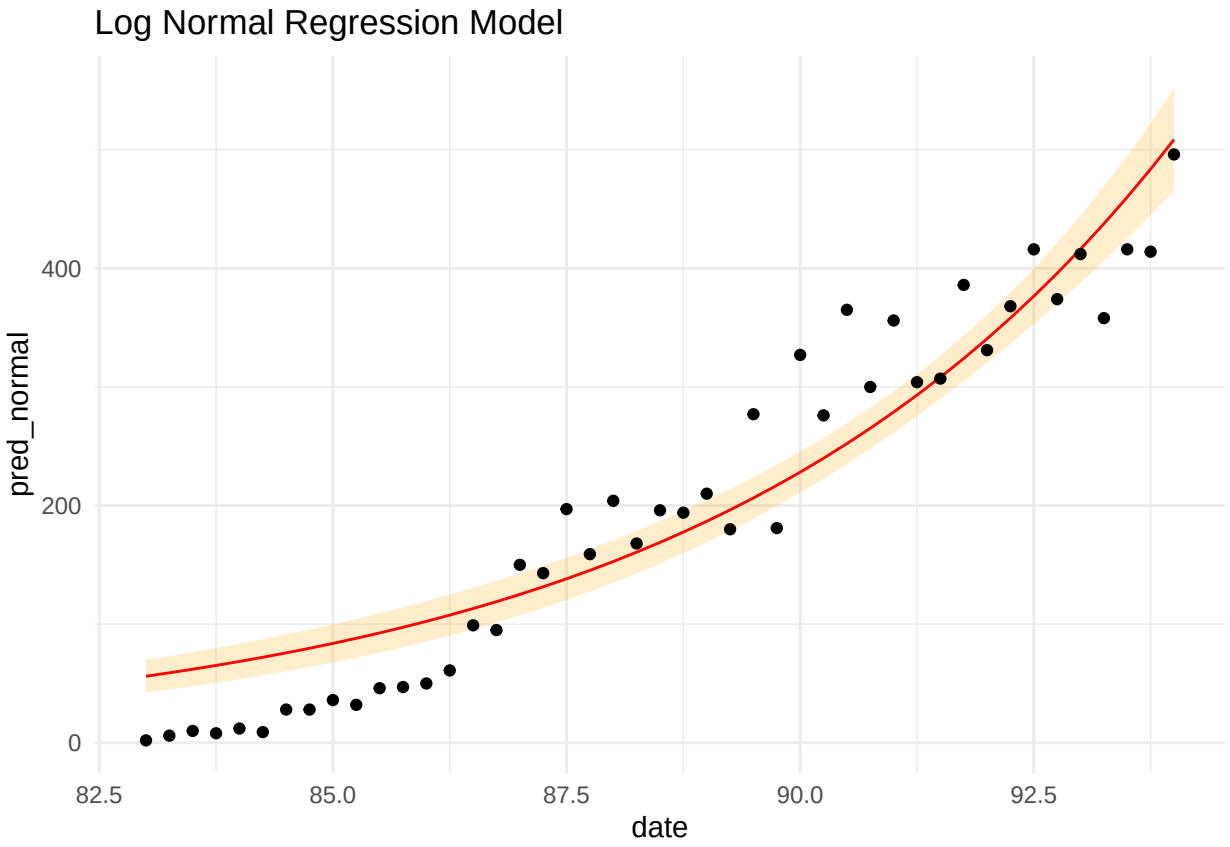
Incorporating the Central Limit Theorem (CLT) into our assessment, we recognise that for large sample sizes, the distribution of sample means tends to approximate a normal distribution, regardless of the original distribution of the data. This principle suggests that, with sufficient aggregation or transformation, including

log transformations aimed at stabilizing variance and making distributions more symmetric, count data could, under certain conditions, be modelled using a normal distribution framework. Applying a log transformation directly to the data or within the model helps in addressing multiplicative relationships and exponential growth characteristics, necessitating a subsequent evaluation of the normality of either the transformed data or the model residuals to validate the normal distribution assumption. Hence, ensuring that the residuals of the model exhibit a normal distribution becomes a pivotal step when adopting a normal distribution for the model, as it directly affects the appropriateness and the inferential validity of the modelling approach. This validation process, grounded in both the CLT and normality assessments post-transformation, is essential for confirming the suitability of a normal distribution model, especially in contexts initially characterised by count data.

In summary, the Poisson model closely aligns with the inherent nature of the count data but faces challenges due to significant overdispersion, which undermines one of its core assumptions. On the other hand, the Normal distribution model offers greater flexibility and could potentially leverage the Central Limit Theorem for justification; however, the dataset does not sufficiently meet the large sample size criterion necessary for the theorem's applicability. This limitation hampers the ability to confidently rely on normality for the transformed data.

Question 2 b

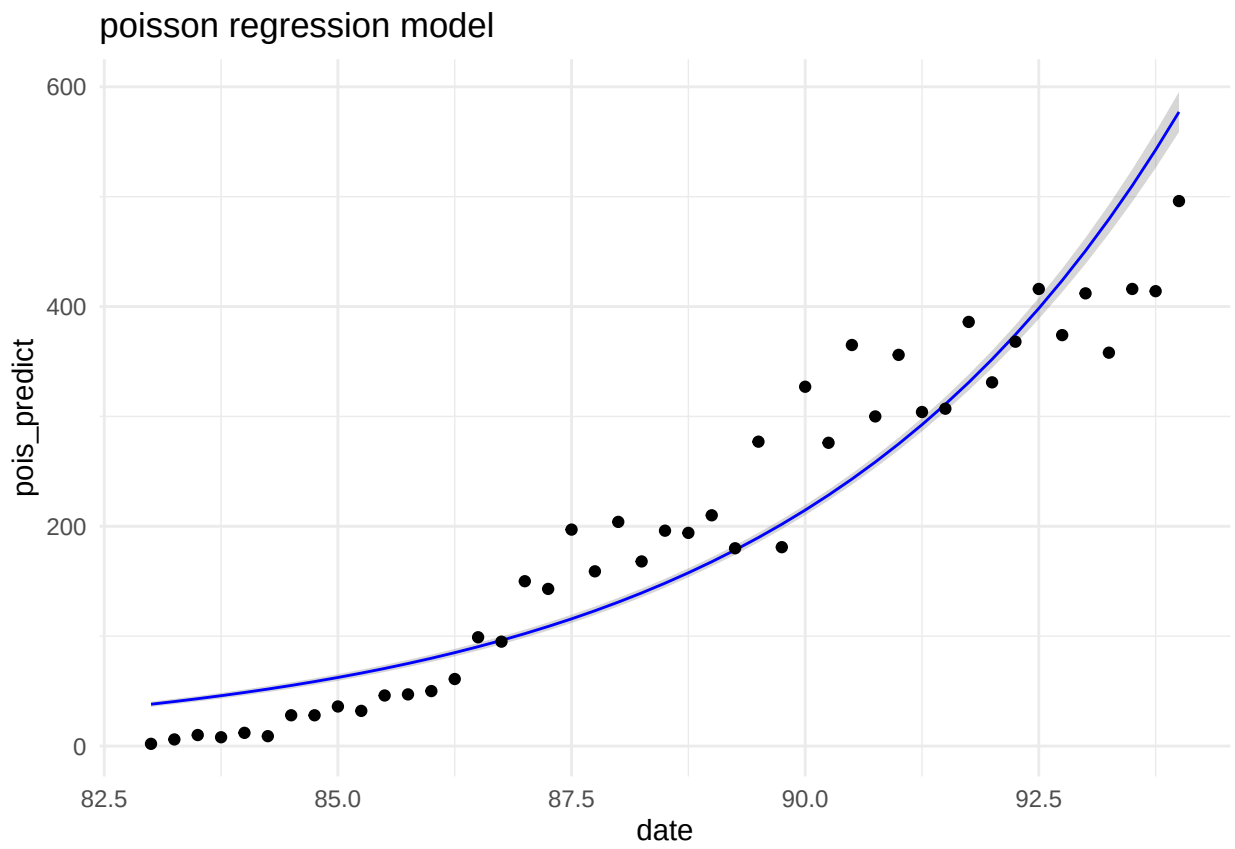
```
normal_model <- glm(cases ~ date, data = aids, family = gaussian("log"))
```



The models predictions don't seem to capture the trend of the actual data with the predicated values deviating significantly from the actual data especially in the earlier dates. The confidence intervals are

fairly narrow showing the model is confident about its predictions, which given the poor fit, isn't justified. Based on the plot the models assumptions don't meet the appropriate relationship between the data.

```
pois_model <- glm(cases ~ date, data = aids, family = poisson(link = 'log'))
```



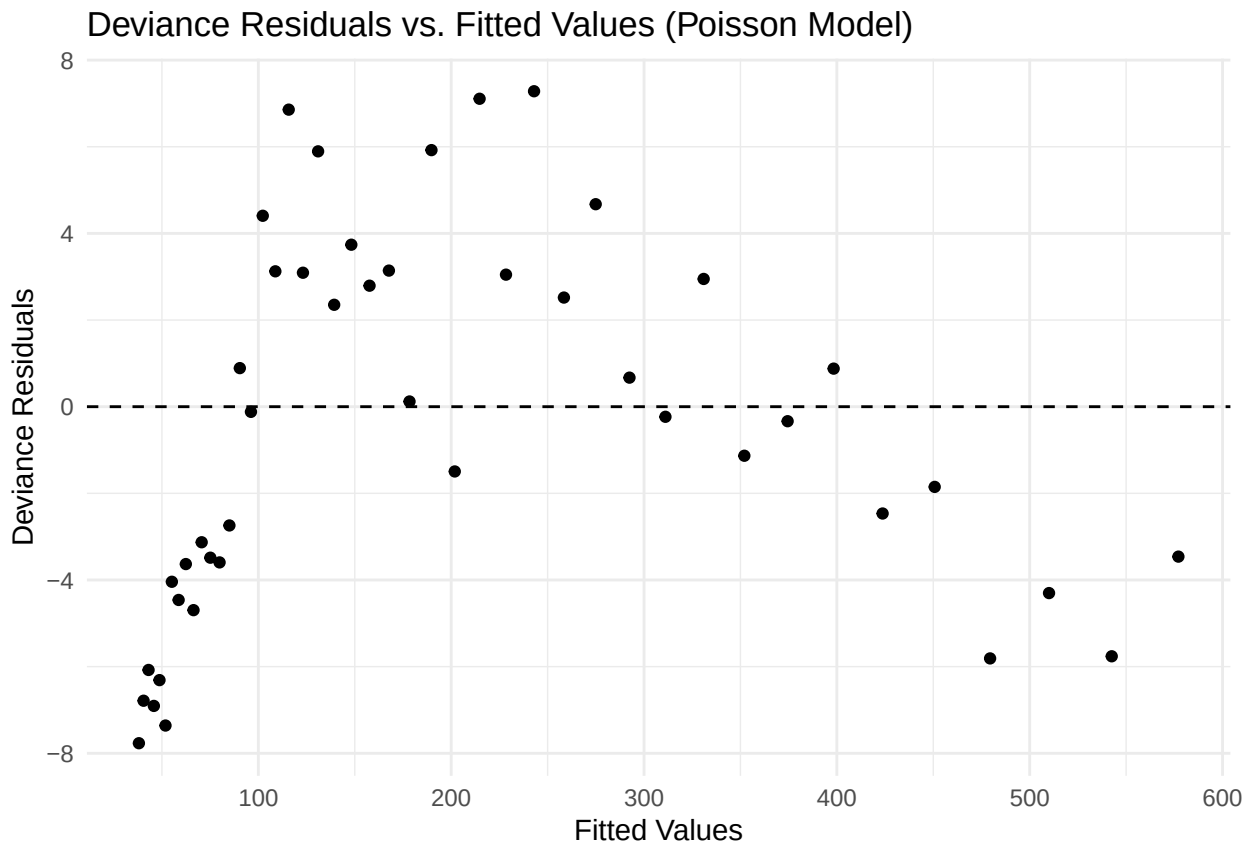
The poissons model predictions are similar to the normal models predictions, with both models having poor predictions in the earlier dates, but the difference is the poisson model has a smaller confidence interval due to in a poisson model the mean and variance are the near equal which doesn't seem to hold up.

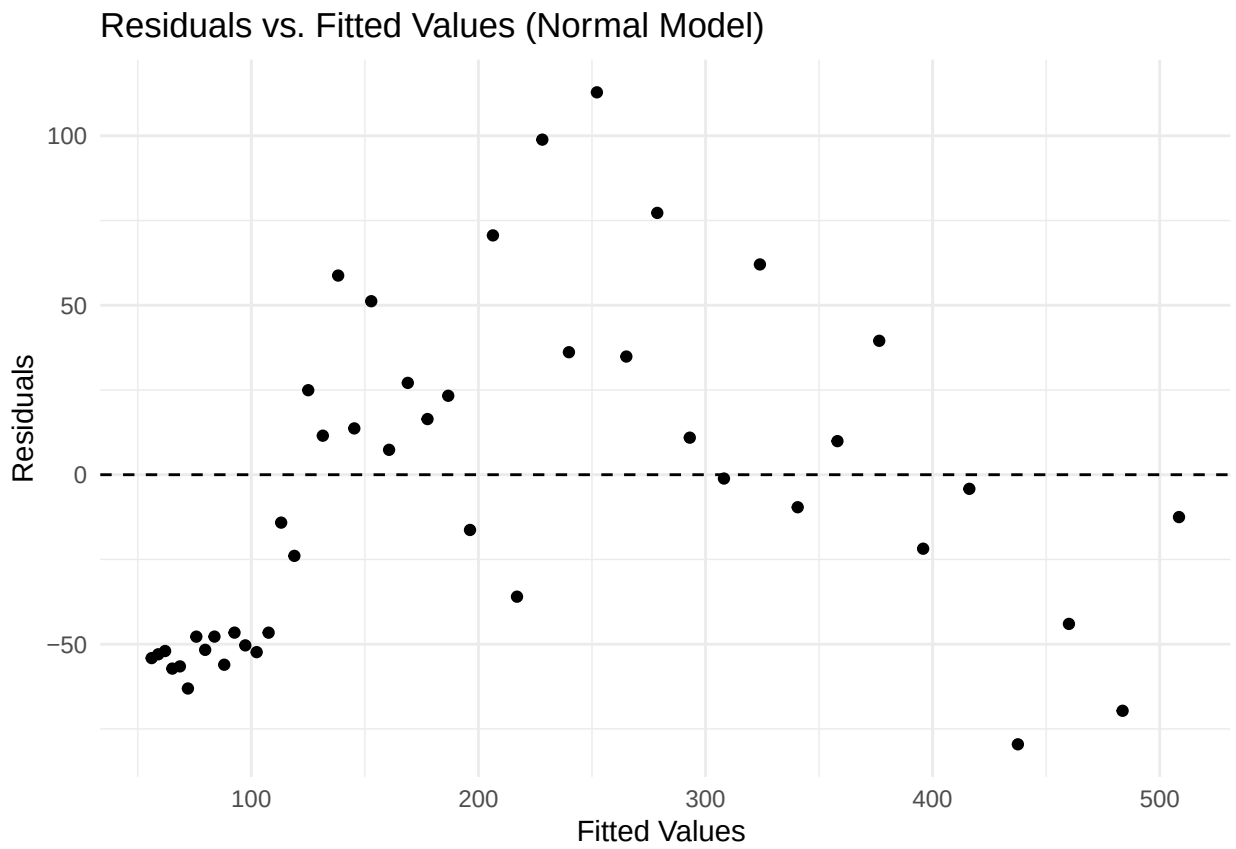
Question 2 c

```
##     Model       AIC       BIC      MSE     RMSE        R2
## 1  Normal  482.8128  488.2328 2340.132 48.37491 0.8918835
## 2 Poisson 1153.8732 1157.4866 3073.856 55.44237 0.8579847
```

Given the statistical metrics provided—AIC, BIC, MSE, RMSE, and R^2—the normal model demonstrates superior performance over the Poisson model for the dataset in question. With significantly lower AIC and BIC values, the normal model suggests a better fit to the data, considering both its accuracy and complexity. This is further supported by lower MSE and RMSE values, indicating that the normal model's predictions are closer to the actual observations. Additionally, the higher R^2 value for the normal model implies it explains a greater proportion of the variance in the response variable. While the normal model is statistically favored based on these metrics, it's crucial to consider the nature of the data and the theoretical assumptions behind each model. The choice between a normal and a Poisson model should also be guided by the data's distribution and the theoretical underpinnings of the models, ensuring that the selected model appropriately reflects the data's characteristics and the research question at hand.

Question 2 d


Deviance Residuals vs. Fitted Values (Poisson Model)
Deviance Residuals
Fitted Values
8
4
0
−4
−8
100
200
300
400
500
600



Both the models over estimate at the start then over estimate in the model then go back to over estimating showing how a polynomial extension would help both models.

Some common extensions are polynomial, seasonal trends, interaction terms, lagging, and GAMs. some not so common extensions would be changing the link function in the model, zero inflation/hurdle models, GAMLSS, auto regression, and Bayesian regression. Specificly for the Poisson model a Quisi model can be used which is relaxes the assumption of a Poisson model that the var is equal to the deviation?

Question 2 e

```
##                             Model       AIC       BIC
## 9      Generalized Additive Model  439.1314  456.6762
## 3      Poisson Polynomial (Cubic)  470.5294  477.7561
## 10            GAM with Cubic Term  475.8327  482.9575
## 11    Poisson Polynomial with Lag  485.7225  492.8593
## 2  Poisson Polynomial (Quadratic)  551.9921  557.4120
## 8       GLM with Square Root Link  636.5564  640.1697
## 5       Poisson with Lagged Cases 1084.2017 1089.5543
## 4                Poisson Seasonal 1125.1102 1134.1435
## 1                   Poisson Model 1153.8732 1157.4866
## 7               GLM with Log Link 1153.8732 1157.4866
## 12            Quasi-Poisson Model 1153.8732 1157.4866
## 6           GLM with Inverse Link 2132.8261 2136.4394
```

```
## Analysis of Deviance Table
```

```
##
## Model 1: cases ~ s(date)
## Model 2: cases ~ date + I(date^2) + I(date^3)
##   Resid. Df Resid. Dev      Df Deviance  Pr(>Chi)
## 1    26.382     72.434
## 2    36.000    125.567 -9.6185  -53.133 4.955e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
## [1] "RMSE for Poisson Polynomial Model: 29.0883417365265"
```

```
## [1] "RMSE for GAM Model: 26.2476798944795"
```

```
## Poisson Polynomial Model (Cubic) - Training RMSE: 27.60452
```

```
## Poisson Polynomial Model (Cubic) - Test RMSE: 35.08846
```

```
## GAM Model - Training RMSE: 22.1972
```

```
## GAM Model - Test RMSE: 44.29743
```

In evaluating the Generalized Additive Model (GAM) and the cubic Poisson Polynomial Model across multiple metrics—AIC, BIC, deviance analysis, and RMSE for training and test datasets—the GAM consistently demonstrates superior performance on the training data, evidenced by lower AIC (439.1314 vs. 470.5294) and BIC (456.6762 vs. 477.7561) values, and a lower training RMSE (22.1972 vs. 27.60452). The analysis of deviance further supports the GAM's better fit, showcasing a significant reduction in unexplained variance compared to the cubic model. However, a nuanced picture emerges when considering the models' generalization capabilities as evidenced by their test RMSE values. Despite the GAM's lower training RMSE, its test RMSE (44.29743) shows a substantial increase from the training set, in contrast to the cubic model's smaller jump (from 27.60452 to 35.08846 in training to test RMSE). This discrepancy suggests that while the GAM may offer a more precise fit to the training data, it potentially overfits, capturing noise as patterns, which hampers its performance on unseen data. Conversely, the cubic model, despite slightly worse fitting metrics, exhibits better generalization, indicated by a less pronounced increase in test RMSE. This evaluation underscores the importance of balancing model complexity and fit with the ability to generalize to new data, highlighting scenarios where the seemingly superior model in terms of training data performance may not necessarily be the best choice for predictions on new, unseen data.

Given the trade-offs between the Generalized Additive Model (GAM) and the cubic Poisson Polynomial Model, my choice leans towards the cubic Poisson Polynomial Model for its better generalization capabilities as evidenced by the smaller increase in RMSE from training to test data. Despite the GAM's superior performance on the training set and its lower AIC and BIC values suggesting a better fit, the significant jump in RMSE on the test set raises concerns about its potential to overfit. In practical applications, the ability of a model to perform well on unseen data is paramount. Therefore, the cubic model's more stable performance across both training and test sets, indicating less overfitting, makes it a more reliable choice for predictions in real-world scenarios where the ultimate goal is to generalize well beyond the training dataset.

```
##                          Model      AIC      BIC
## 3 Polynomial Model 3rd Degree 440.8866 449.9199
## 4                   GAM Model 439.7104 453.6085
## 7               Complex Model 439.3644 459.2377
## 2 Polynomial Model 2nd Degree 447.8615 455.0882
## 6           Interaction Model 445.7260 461.9859
## 1                Normal Model 482.8128 488.2328
## 5              Seasonal Model 485.5348 496.3747
```

```
## Analysis of Deviance Table
##
## Model 1: cases ~ s(date)
## Model 2: cases ~ date + I(date^2) + I(date^3)
##   Resid. Df Resid. Dev      Df Deviance Pr(>Chi)
## 1    37.398      32801
## 2    45.000      37951 -7.6023  -5149.3   0.6045
```

```
## Normal Polynomial Model (Cubic) - Training RMSE: 27.5731
```

```
## Normal Polynomial Model (Cubic) - Test RMSE: 34.28769
```

```
## GAM Model - Training RMSE: 24.73046
```

```
## GAM Model - Test RMSE: 34.6154
```

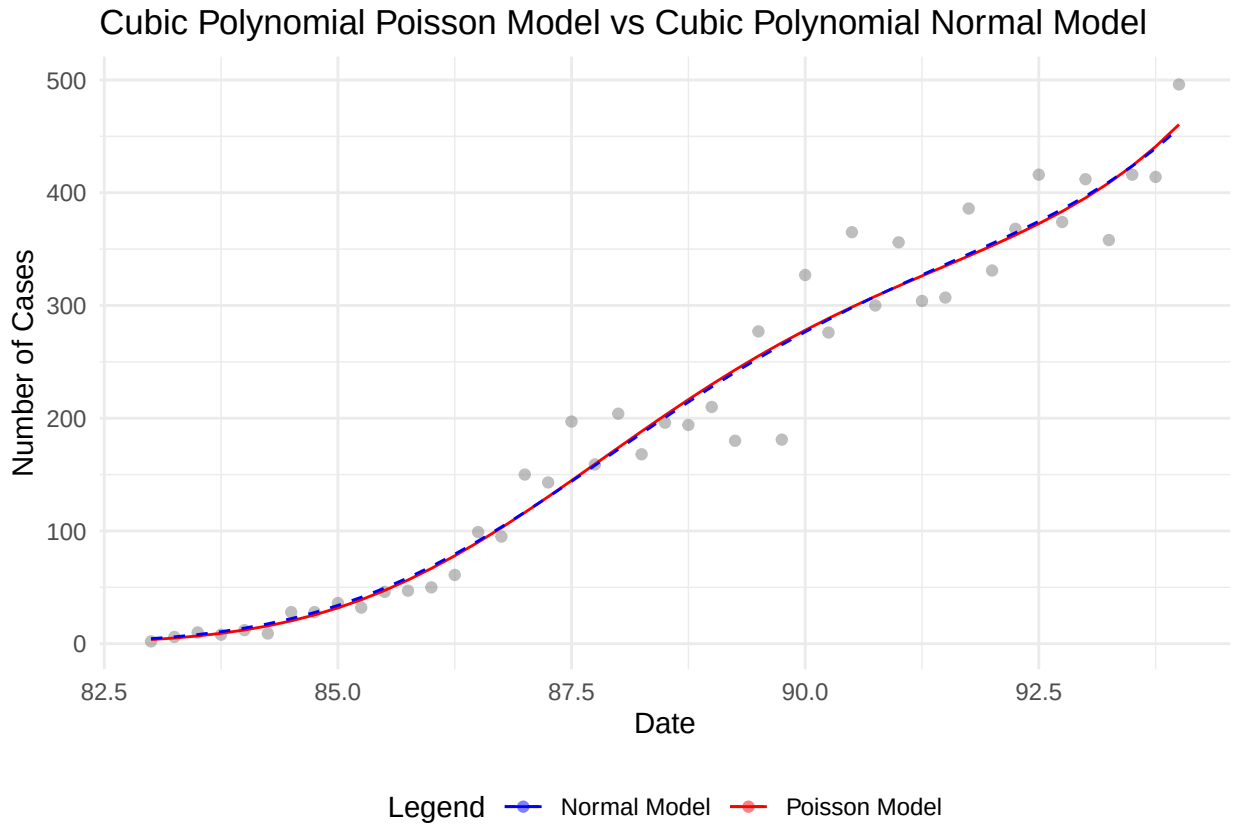
The comparative analysis between the Generalized Additive Model (GAM) and the 3rd Degree Polynomial Model reveals intricate nuances regarding model selection based on AIC, BIC, deviance analysis, and RMSE metrics. The GAM holds a slender advantage in AIC values, hinting at a slightly superior balance between model complexity and its ability to elucidate the variance in the data. This advantage is subtly underscored by the GAM's marginally lower AIC, suggesting a more effective variance explanation, though it is accompanied by a higher BIC, indicating an increase in model complexity.

Contrastingly, the analysis of deviance introduces a layer of complexity to this evaluation. Despite a notable difference in residual deviance that seems to favor the GAM, the p-value of 0.6045 derived from the likelihood ratio test does not achieve the conventional significance threshold. This result suggests that the improvement in model fit offered by the GAM, while mathematically evident, does not significantly surpass the Polynomial Model in explaining the data's variance from a statistical standpoint.

This juxtaposition of a slight AIC-driven preference for the GAM against the backdrop of a statistically non-significant p-value in deviance analysis highlights the importance of balancing statistical significance with practical considerations in model selection. Furthermore, an examination of overfitting through RMSE comparisons for training and test datasets reveals that the GAM, despite its nuanced fit improvement, exhibits a greater propensity for overfitting compared to the Polynomial Model. This tendency is evidenced by a larger discrepancy between its training and test RMSE values, although not as pronounced as in some previous comparisons, suggesting a more restrained but still present overfitting risk.

This nuanced assessment underscores that while the GAM may offer slight statistical improvements in modeling the data, these advantages do not robustly justify its selection over the simpler, more interpretable 3rd Degree Polynomial Model, especially when considering the practical implications of overfitting. The Polynomial Model's relatively stable performance across training and test datasets, coupled with its simplicity and lower complexity (as indicated by BIC), aligns it more closely with the principle of parsimony. This principle favors models that achieve a reasonable balance between accuracy and simplicity, ensuring robustness and generalizability in practical applications. Therefore, in light of both statistical and practical considerations, the 3rd Degree Polynomial Model emerges as a preferable choice, offering a compelling blend of simplicity, interpretability, and effective data explanation without the heightened risk of overfitting associated with the more complex GAM.

Question 2 f



Both the Poisson and Normal models are fundamentally grounded in the assumptions of their chosen distributions, with their suitability hinging on how well these assumptions align with the characteristics of the dataset. For the Poisson model, the assumption of equal mean and variance does not hold, indicating a significant mismatch with the dataset's inherent nature, particularly evident when even a quasi-Poisson model does not substantially improve model fit. In contrast, the Normal model, predicated on assumptions more amenable to datasets with a large number of observations, such as those concerning national-level AIDS data, finds theoretical support from the Central Limit Theorem (CLT). The CLT suggests that with sufficient data, the distribution of sample means will approximate a normal distribution, making the Normal model a potentially more fitting choice for large datasets but more data will have to be collected for this to hold true as 45 observations isnt a large dataset.

When examining model extensions, both the Poisson and Normal models exhibit improvements with the introduction of Generalized Additive Models (GAM), and quadratic and cubic polynomial extensions. Notably, the Normal model outperforms its Poisson counterparts across these extensions, except for the GAM, where overfitting is a concern. This performance discrepancy could be attributed to the inherent flexibility and the more general applicability of the Normal model to a wider range of data characteristics, including overdispersion and large sample sizes.

A direct comparison using AIC and BIC metrics shows the Normal model in a more favorable light, suggesting a better balance of model fit and complexity. However, when evaluating for overfitting through train-test splits, both models demonstrate comparable performance, which complicates the selection process. Given the mixed evidence, making a definitive choice between these distributions without further data is challenging. Nevertheless, the slight edge in general applicability, alongside marginally better AIC and BIC values for the Normal model, nudges the preference towards it. This inclination is supported by its theoretical foundation in the CLT, offering a more generalized approach suitable for large datasets, despite the close performance

metrics and deviance values observed in comparison to the Poisson model.

In refining the discussion, it's clear that while the Poisson model's strict assumptions limit its applicability in the face of overdispersion and variance-mean discrepancies, the Normal model's broader assumptions and grounding in the CLT render it slightly more versatile for analyzing extensive datasets. This nuanced analysis underscores the importance of considering both statistical metrics and theoretical underpinnings when selecting a model, with a lean towards the Normal distribution for its slightly superior adaptability and performance in the face of large, complex datasets.

Question 2 g

```
# Fit a Negative Binomial model to the data
nb_model_poly3 <- glm.nb(cases ~ date + I(date^2) + I(date^3), data = aids)
```

```
## [1] "AIC: 405.915498090922"
```

```
## [1] "BIC: 414.948810539774"
```

```
## Neg Binom - Training RMSE: 27.68461
```

```
## Neg Binom - Test RMSE: 34.18502
```

The primary challenge encountered with the previous Poisson models stemmed from their foundational assumption that the mean and variance of the distribution are equal—an assumption that led to significant overdispersion in the model's performance. The adoption of a Negative Binomial model addresses this issue directly by incorporating an additional parameter that allows the variance to exceed the mean, thereby offering a more flexible and accurate representation of the data's underlying distribution.

The improvement in model fit facilitated by the Negative Binomial approach is evident in the AIC and BIC scores, which are critical metrics for assessing the quality of statistical models. While these scores indicate a more favorable model fit compared to earlier Poisson and Normal models, it's noteworthy that the Root Mean Square Error (RMSE) remains comparable to the other models. This similarity in RMSE suggests that, in terms of predictive accuracy, the Negative Binomial model performs on par with its counterparts.

However, the true advantage of the Negative Binomial model lies in its alignment with the data's characteristics. By accurately accommodating the overdispersion observed in the dataset—an aspect that Poisson and Normal models struggled with—the Negative Binomial model inherently provides a more reliable and theoretically sound basis for analysis. This alignment with the data's distribution not only justifies the model's use from a statistical standpoint but also enhances the credibility and interpretability of the model's predictions and inferences.

In essence, the adoption of a Negative Binomial model represents a significant methodological improvement by rectifying the misalignment between model assumptions and data characteristics observed in previous models. This adjustment ensures a more accurate representation of the data's distribution, which, despite similar RMSE values, ultimately leads to better model performance due to the more appropriate handling of the underlying statistical properties of the dataset.

Bibliogrophy

```r
# Load required libraries
library(ggplot2)
library(tidyverse)
library(mgcv)
library(stats)
library(MASS)
library(gbm)

# Set seed for reproducibility
set.seed(02032024)

# Load data from a file
load("C:/Users/archi/Downloads/datasets_exercises.RData")

# Plot the data
plot(nlmodel$x, nlmodel$y, main = "Simple Scatter Plot", xlab = "X-axis", ylab = "Y-axis")

# Extract data from the loaded dataset
x <- nlmodel$x
y <- nlmodel$y
n <- 100  # Not used in the code, seems redundant
var <- var(y)  # Calculate variance of y

# Define the negative log-likelihood function for optimization
mylike <- function(params, y, x) {
  # Extract parameters from the input vector
  theta1 <- params[1]  # First parameter
  theta2 <- params[2]  # Second parameter
  sigma2 <- params[3]  # Third parameter, variance

  # Check if variance is less than or equal to 0, return Inf as error state
  if (sigma2 <= 0) {
    return(Inf)
  }

  # Get the number of observations
  n <- length(y)

  # Initialize a vector to store calculated means for each observation
  mu <- numeric(n)

  # Calculate the mean for each observation based on the model equation
  for (i in 1:n) {
    mu[i] <- theta1 * x[i] / (theta2 + x[i])
  }

  # Compute the log-likelihood for the normal distribution
  logL <- sum(-1/2*log(2*pi*sigma2) - ((y-mu)^2) / (2*sigma2))

  # Return the negative log-likelihood to set up for minimization
  -logL
}
```

```r
# Initial parameter guesses for the optimization
initial_values <- c(0, 0, var)

# Optimize the parameters using the 'nlm' function
result <- nlm(f = mylike, p = initial_values, y = y, x = x, hessian = TRUE, iterlim = 1000)

# Print out the optimization results
print(paste("Optimized parameter values:", result$estimate))
print(paste("Exit code from the optimizer:", result$code))
print(paste("The minimum value of the objective function (negative log-likelihood) was:", result$minimu
print(paste("Gradient of the objective function at the solution:", result$gradient))
print(paste("Number of iterations taken to converge:", result$iterations))


# Define a function to perform the optimization of the nonlinear model
nlmodel.optimize <- function(x, y) {
  # Initial guesses for the parameters, setting sigma2 as the variance of y
  initial_values <- c(theta1 = 0, theta2 = 0, sigma2 = var(y))

  # Wrapper function for mylike to include additional arguments (x, y)
  mylike_wrapper <- function(p) {
    mylike(p, y = y, x = x)
  }

  # Perform nonlinear minimization using nlm
  nlm_res <- nlm(f = mylike_wrapper, p = initial_values, hessian = TRUE, iterlim = 1000)

  # Return the result of the optimization
  return(nlm_res)
}


# Optimizing parameters
optim_results <- nlmodel.optimize(x, y) # Call the optimization function with the data
optimized_params <- optim_results$estimate # Extract optimized parameters from the results

# Define a function to plot the likelihood for a range of values for each parameter
plot_param_likelihood <- function(x, y, optimized_params, param_index, param_range) {
  # Calculate likelihoods for a range of values for the specified parameter
  likelihoods <- sapply(param_range, function(param) {
    params <- optimized_params # Start with optimized parameters
    params[param_index] <- param # Replace the specified parameter with a value from the range
    -mylike(params, y, x) # Calculate the negative log-likelihood and convert to likelihood
  })

  # Plot the likelihoods against the parameter values
  plot(param_range, likelihoods, type = 'l',
       main = paste("Likelihood vs.", names(optimized_params)[param_index]), # Title with the parameter
       xlab = names(optimized_params)[param_index], # X-axis label with the parameter name
       ylab = "Likelihood", # Y-axis label
       col = 'red', # Line color
       cex.main = 0.8) # Adjust the title size
}
```

```r
# Define parameter ranges for plotting based on the optimized parameters
theta1_range <- seq(from = optimized_params[1] * 0.8, to = optimized_params[1] * 1.2, length.out = 100)
theta2_range <- seq(from = optimized_params[2] * 0.8, to = optimized_params[2] * 1.2, length.out = 100)
sigma2_range <- seq(from = optimized_params[3] * 0.5, to = optimized_params[3] * 1.5, length.out = 100)

# Setup the plotting area to display three plots in a single row
par(mfrow=c(1,3))

# Plot the likelihood as a function of each parameter using the defined ranges
plot_param_likelihood(x, y, optimized_params, param_index = 1, param_range = theta1_range)
plot_param_likelihood(x, y, optimized_params, param_index = 2, param_range = theta2_range)
plot_param_likelihood(x, y, optimized_params, param_index = 3, param_range = sigma2_range)

# Define a function for making predictions based on the optimized model
predict_function <- function(params, x) {
  theta1 <- params[1]
  theta2 <- params[2]
  # Note: sigma2 is not used for prediction, only for fitting the model
  mu <- theta1 * x / (theta2 + x) # Model equation for predictions
  return(mu)
}

# Calculate predicted values based on the optimized parameters
predicted_values <- predict_function(result$estimate, x)

# Calculate residuals (difference between observed and predicted values)
residuals <- y - predicted_values

# Plot residuals to assess model fit
plot(residuals, type = 'p') # Plot as points
abline(h = 0, col = "red") # Horizontal line at 0 for reference

# Extract the Hessian matrix from the optimization result
hessian_matrix <- result$hessian

# Invert the Hessian matrix to obtain the variance-covariance matrix of the parameter estimates
var_cov_matrix <- solve(hessian_matrix)

# Compute standard errors for the parameter estimates (square roots of diagonal elements of the var-cov
standard_errors <- sqrt(diag(var_cov_matrix))

# Print standard errors for each parameter
cat("Standard Error for theta_1:", standard_errors[1], "\n")
cat("Standard Error for theta_2:", standard_errors[2], "\n")
cat("Standard Error for sigma^2:", standard_errors[3], "\n")

# Extract parameter estimates and their standard errors
theta1_estimate <- result$estimate[1]
theta2_estimate <- result$estimate[2]
sigma2_estimate <- result$estimate[3]
SE_theta1 <- standard_errors[1]
SE_theta2 <- standard_errors[2]
SE_sigma2 <- standard_errors[3]
```

```r
z_99 <- 2.576 # Z-value for 99% confidence level

# Calculate and print 99% confidence intervals for each parameter
CI_theta1_lower <- theta1_estimate - (z_99 * SE_theta1)
CI_theta1_upper <- theta1_estimate + (z_99 * SE_theta1)
CI_theta2_lower <- theta2_estimate - (z_99 * SE_theta2)
CI_theta2_upper <- theta2_estimate + (z_99 * SE_theta2)
CI_sigma2_lower <- sigma2_estimate - (z_99 * SE_sigma2)
CI_sigma2_upper <- sigma2_estimate + (z_99 * SE_sigma2)

# Print confidence intervals
cat("99% Confidence Interval for theta_1: [", CI_theta1_lower, ",", CI_theta1_upper, "]\n")
cat("99% Confidence Interval for theta_2: [", CI_theta2_lower, ",", CI_theta2_upper, "]\n")
cat("99% Confidence Interval for sigma^2: [", CI_sigma2_lower, ",", CI_sigma2_upper, "]\n")

# Given data
theta2_hypothesized <- 0.08
# Calculate the z statistic
z_statistic <- (theta2_estimate - theta2_hypothesized) / SE_theta2
# Critical z values for 10% significance level (two-tailed test)
z_critical <- 1.645 # For one-tailed, but since it's two-tailed, we use +/- this value
# Print the z statistic
cat("z Statistic:", z_statistic, "\n")
# Decision rule
if(abs(z_statistic) > z_critical) {
  cat("Reject H0: The test statistic falls outside the critical region.\n")
} else {
  cat("Fail to reject H0: The test statistic falls within the critical region.\n")
}

# Predicted values
sigma2 <- result$estimate[3]
nlmodel$mu <- theta1_estimate * nlmodel$x/ (theta2_estimate + nlmodel$y)

# Calculate standard error of the estimate
n <- nrow(nlmodel)
x_bar <- mean(nlmodel$x)
t_critial <- qt(0.975, df=n-2)
nlmodel$se <- sqrt(sigma2*(1+1/n+((nlmodel$x-x_bar)^2)/sum((nlmodel$x-x_bar)^2)))

# Calculate 95% confidence intervals for predictions
nlmodel$lower_95 <- nlmodel$mu - t_critial * nlmodel$se
nlmodel$upper_95 <- nlmodel$mu + t_critial * nlmodel$se

# Calculate standard error of predictions
residual_variance <- var(y - predicted_values)
se_pred <- sqrt(residual_variance)  # Standard error of predictions
z_95 <- 1.96  # Critical value for 95% CI

# Calculate lower and upper bounds of the 95% prediction interval
lower_bound <- predicted_values - z_95 * se_pred
upper_bound <- predicted_values + z_95 * se_pred
```

```r
# Create a data frame for plotting
data_plot <- data.frame(x = x, y = y, lower_bound = lower_bound, upper_bound = upper_bound, predicted_va

suppressWarnings({
  # Plot using ggplot2
  ggplot(data_plot, aes(x = x)) +
    geom_point(aes(y = y), colour = "blue", size = 2) +
    geom_ribbon(aes(ymin = lower_bound, ymax = upper_bound), fill = "darkgrey", alpha = 0.2) +
    geom_line(aes(y = predicted_values), colour = "red", size = 1) +
    labs(title = "Observed vs. Fitted Values with 95% PI", x = "X", y = "Y") +
    theme_minimal() +
    scale_colour_manual("",
                        breaks = c("Observed", "Fitted", "95% PI"),
                        values = c("blue", "red", "darkgrey")) +
    theme(panel.grid.major = element_blank(),  # Remove major gridlines
          panel.grid.minor = element_blank(),  # Remove minor gridlines
          panel.background = element_blank(),   # Remove background
          axis.line = element_line(color = "black"))  # Add axis line
})

# Plot the original scatter plot from the "aids" dataset
plot(aids$cases, aids$date, main = "Simple Scatter Plot", xlab = "X-axis", ylab = "Y-axis")

# Summary of the 'cases' variable in the 'aids' dataset
summary(aids$cases)

# Histogram with density overlay of 'cases' variable
ggplot(aids, aes(x = cases)) +
  geom_histogram(aes(y = after_stat(density)), binwidth = 30, colour = "black", fill = "grey") +
  geom_density(alpha = .2, fill = "#FF6666") +
  labs(title = "Histogram of AIDS Cases with Density Overlay", x = "Cases", y = "Density") +
  theme_minimal()

# Poisson regression model
pois_model <- glm(cases ~ date, data = aids, family = poisson(link = 'log'))

# Pearson residuals and dispersion statistic for Poisson model
pearson_resid <- residuals(pois_model, type = "pearson")
dispersion_statistic <- sum(pearson_resid^2) / pois_model$df.residual
dispersion_statistic

# Normal regression model
normal_model <- glm(cases ~ date, data = aids, family = gaussian("log"))

# Confidence intervals for normal regression model
pred_normal <- predict(normal_model, newdata = aids, se = TRUE, type = "response")
aids$pred_normal <- pred_normal$fit
aids$ci_lower_normal <- pred_normal$fit - 1.96 * pred_normal$se.fit
aids$ci_upper_normal <- pred_normal$fit + 1.96 * pred_normal$se.fit

# Plotting normal regression model
ggplot(aids, aes(x = date)) +
  geom_ribbon(aes(ymin = ci_lower_normal, ymax = ci_upper_normal), alpha = 0.2, fill = 'orange') +
```

```
  geom_line(aes(y = pred_normal), color = 'red') +
  geom_point(aes(y = cases)) +
  ggtitle("Log Normal Regression Model") +
  theme_minimal()

# Poisson regression model with confidence intervals
aids$pois_predict <- predict(pois_model, type = 'response')
pred_with_se <- predict(pois_model, type = 'response', se.fit = TRUE)
aids$ci_lower_pois <- pred_with_se$fit - 1.96 * pred_with_se$se.fit
aids$ci_upper_pois <- pred_with_se$fit + 1.96 * pred_with_se$se.fit

# Plotting Poisson regression model
ggplot(aids, aes(x = date)) +
  geom_ribbon(aes(ymin = ci_lower_pois, ymax = ci_upper_pois), alpha = 0.2) +
  geom_line(aes(y = pois_predict), color = 'blue') +
  geom_point(aes(y = cases)) +
  ggtitle('Poisson Regression Model') +
  theme_minimal()

# Metrics calculation for both models
aic_normal <- AIC(normal_model)
bic_normal <- BIC(normal_model)
mse_normal <- mean((aids$cases - aids$pred_normal)^2)
rmse_normal <- sqrt(mse_normal)
r2_normal <- 1 - sum((aids$cases - aids$pred_normal)^2) / sum((aids$cases - mean(aids$cases))^2)

aic_pois <- AIC(pois_model)
bic_pois <- BIC(pois_model)
mse_pois <- mean((aids$cases - aids$pois_predict)^2)
rmse_pois <- sqrt(mse_pois)
r2_pois <- 1 - sum((aids$cases - aids$pois_predict)^2) / sum((aids$cases - mean(aids$cases))^2)

# Creating a table of metrics
metrics_table <- data.frame(
  Model = c("Normal", "Poisson"),
  AIC = c(aic_normal, aic_pois),
  BIC = c(bic_normal, bic_pois),
  MSE = c(mse_normal, mse_pois),
  RMSE = c(rmse_normal, rmse_pois),
  R2 = c(r2_normal, r2_pois)
)

# Printing the table
print(metrics_table)

# Summary of both models
summary(normal_model)
summary(pois_model)

# Plotting deviance residuals vs. fitted values for the Poisson model
poisson_residuals <- residuals(pois_model, type = 'deviance')
poisson_fitted <- fitted(pois_model)
```

```r
ggplot(aids, aes(x = poisson_fitted, y = poisson_residuals)) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed") +
  labs(title = "Deviance Residuals vs. Fitted Values (Poisson Model)",
       x = "Fitted Values", y = "Deviance Residuals") +
  theme_minimal()

# Assuming 'normal_model' is your fitted model on transformed data
normal_residuals <- residuals(normal_model)
normal_fitted <- fitted(normal_model)

# Plotting residuals vs. fitted values for the normal model
ggplot(aids, aes(x = normal_fitted, y = normal_residuals)) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed") +
  labs(title = "Residuals vs. Fitted Values (Normal Model)",
       x = "Fitted Values", y = "Residuals") +
  theme_minimal()

# Fit a quadratic polynomial Poisson regression model
pois_model_poly <- glm(cases ~ date + I(date^2), data = aids, family = poisson(link = 'log'))
aids$pred_poly <- predict(pois_model_poly, type = "response")

# Plotting the quadratic polynomial Poisson model
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases), color = "blue", alpha = 0.5) +
  geom_line(aes(y = pred_poly), color = "red") +
  labs(title = "Polynomial Poisson Model", y = "Number of Cases", x = "Date") +
  theme_minimal()

# Fit a cubic polynomial Poisson regression model
pois_model_poly3 <- glm(cases ~ date + I(date^2) + I(date^3), data = aids, family = poisson(link = 'log
aids$pred_poly3 <- predict(pois_model_poly3, type = "response")

# Plotting the cubic polynomial Poisson model
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases), color = "blue", alpha = 0.5) +
  geom_line(aes(y = pred_poly3), color = "red") +
  labs(title = "Cubic Polynomial Poisson Model", y = "Number of Cases", x = "Date") +
  theme_minimal()

# Fit a Poisson regression model with a seasonal factor (quarter as a categorical variable)
pois_model_seasonal <- glm(cases ~ date + factor(quarter), data = aids, family = poisson(link = 'log'))
aids$pred_seasonal <- predict(pois_model_seasonal, type = "response")

# Plotting the seasonal Poisson model
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases), color = "blue", alpha = 0.5) +
  geom_line(aes(y = pred_seasonal), color = "green") +
  labs(title = "Seasonal Poisson Model", y = "Number of Cases", x = "Date") +
  theme_minimal()

# Fit a Poisson regression model with a lagged variable for cases
```

```r
aids$lag_cases <- c(NA, head(aids$cases, -1))
pois_model_lag <- glm(cases ~ date + lag_cases, data = aids, family = poisson(link = "log"), na.action =
aids$pred_lag_cases <- predict(pois_model_lag, type = "response")

# Plotting the Poisson model with lagged cases
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases), color = "black", alpha = 0.5, size = 2) +
  geom_line(aes(y = pred_lag_cases), color = "red", size = 1) +
  labs(title = "Poisson Model with Lagged Cases",
       x = "Date",
       y = "Cases (Observed and Predicted)") +
  theme_minimal()

# Fit a Poisson regression model with an interaction term between date and quarter
pois_model_interaction <- glm(cases ~ date * factor(quarter), data = aids, family = poisson(link = 'log
date_range <- seq(min(aids$date), max(aids$date), length.out = 100)
quarters <- unique(aids$quarter)
prediction_data <- expand.grid(date = date_range, quarter = quarters)
prediction_data$cases_predicted <- predict(pois_model_interaction, newdata = prediction_data, type = "r

# Plotting predicted cases with interaction between date and quarter
ggplot() +
  geom_point(data = aids, aes(x = date, y = cases, color = quarter), alpha = 0.5) +
  geom_line(data = prediction_data, aes(x = date, y = cases_predicted, color = quarter)) +
  labs(title = "Predicted Cases with Interaction between Date and Quarter",
       x = "Date",
       y = "Predicted/Observed Number of Cases") +
  theme_minimal() +
  scale_color_brewer(palette = "Set1", name = "Quarter")

# Fit Poisson regression models with different link functions
glm_log <- glm(cases ~ date, data = aids, family = poisson(link = "log"))
glm_sqrt <- glm(cases ~ date, data = aids, family = poisson(link = "sqrt"))
glm_inverse <- glm(cases ~ date, data = aids, family = poisson(link = "inverse"))
aids$pred_log <- predict(glm_log, type = "response")
aids$pred_sqrt <- predict(glm_sqrt, type = "response")
aids$pred_inverse <- predict(glm_inverse, type = "response")

# Plotting Poisson regression with different link functions
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases), color = "black", alpha = 0.5, size = 2) +
  geom_line(aes(y = pred_log, color = "log Link"), size = 1) +
  geom_line(aes(y = pred_sqrt, color = "Square Root Link"), size = 1, linetype = "dashed") +
  geom_line(aes(y = pred_inverse, color = "Inverse Link"), size = 1, linetype = "dotdash") +
  scale_color_manual(values = c("log Link" = "red", "Square Root Link" = "blue", "Inverse Link" = "green
  labs(title = "Poisson Regression with Different Link Functions",
       x = "Date",
       y = "Cases (Observed and Predicted)") +
  theme_minimal() +
  guides(color = guide_legend(title = "Model"))

# Fit a GAM with a Poisson response
gam_model <- gam(cases ~ s(date), data = aids, family = poisson(link = "log"))
```

```r
aids$pred_gam = predict(gam_model, type = "response")

# Plotting GAM models
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases), color = "blue", alpha = 0.5) +
  geom_line(aes(y = pred_gam), color = "red") +
  labs(title = "GAM Models", y = "Number of Cases", x = "Date") +
  theme_minimal()

# Fit a GAM with a cubic term and a different link function
gam_model3 <- gam(cases ~ s(date, k = 2) + I(date^2), data = aids, family = poisson(link = "sqrt"))
aids$pred_gam3 <- predict(gam_model3, type = "response")

# Plotting GAM model predictions vs. actual cases
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases), color = "blue", alpha = 0.5) +
  geom_line(aes(y = pred_gam3), color = "red") +
  labs(title = "GAM Model3 Predictions vs. Actual Cases", y = "Number of Cases", x = "Date") +
  theme_minimal()

# Fit a Poisson regression model with a quadratic term and a lagged variable
pois_model_lag_poly <- glm(cases ~ date + I(date^2) + lag_cases, data = aids, family = poisson(link = "
aids$pred_lag_cases1 <- predict(pois_model_lag_poly, type = "response")

# Plotting Poisson model with lagged cases
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases), color = "black", alpha = 0.5, size = 2) +
  geom_line(aes(y = pred_lag_cases1), color = "red", size = 1) +
  labs(title = "Poisson Model with Lagged Cases",
       x = "Date",
       y = "Cases (Observed and Predicted)") +
  theme_minimal()

# Fit a quasi-Poisson regression model to account for overdispersion
quas_poi <- glm(cases ~ date, data = aids, family = quasipoisson(link = 'log'))
aids$quas_poi <- predict(quas_poi, type = "response")

# Plotting quasi-Poisson model
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases), color = "black", alpha = 0.5, size = 2) +
  geom_line(aes(y = quas_poi), color = "red", size = 1) +
  labs(title = "Poisson Model with Lagged Cases",
       x = "Date",
       y = "Cases (Observed and Predicted)") +
  theme_minimal()

# Define a function to create a QQ plot for assessing the normality of residuals
plot_qq <- function(model, model_name) {
  # Extract Pearson residuals from the model
  pearson_res <- residuals(model, type = "pearson")

  # Create a QQ plot of the residuals
  # qqnorm plots the standardized residuals against a standard normal distribution
```

```r
  qqnorm(pearson_res, main = paste("QQ Plot for", model_name))

  # qqline adds a reference line to the plot, here in red, which helps in visually assessing the normal
  qqline(pearson_res, col = "red")
}

# Usage of the function for various models
# 'pois_model' should be replaced with the actual model object, same for the other model names.
# The function is called with each model object and a descriptive name for the model.
plot_qq(pois_model, "Poisson Model")
plot_qq(quas_poi, "Quasi-Poisson Model")
plot_qq(pois_model_poly, "Poisson Polynomial Model (Quadratic)")
plot_qq(pois_model_poly3, "Poisson Polynomial Model (Cubic)")
plot_qq(pois_model_seasonal, "Poisson Seasonal Model")
plot_qq(pois_model_lag, "Poisson Model with Lagged Cases")
plot_qq(glm_inverse, "Poisson Model with Inverse Link")
plot_qq(glm_log, "Poisson Model with Log Link")
plot_qq(glm_sqrt, "Poisson Model with Square Root Link")
plot_qq(gam_model, "Generalized Additive Model (GAM)")
plot_qq(gam_model3, "Generalized Additive Model with Cubic Term (GAM V3)")
plot_qq(pois_model_lag_poly, "Poisson Polynomial Model with Lag (Quadratic)")

# Define the function to create Residuals vs. Fitted values plot
plot_resid_fitted <- function(model, model_name) {
  # Extract fitted values from the model
  fitted_values <- fitted(model)

  # Extract Pearson residuals from the model
  residuals <- residuals(model, type = "pearson")

  # Create a ggplot
  ggplot(data = data.frame(Fitted = fitted_values, Residuals = residuals), aes(x = Fitted, y = Residuals
    geom_point() +  # Scatter plot of residuals vs fitted values
    geom_hline(yintercept = 0, linetype = "dashed", color = "red") +  # Horizontal line at 0 to aid vis
    labs(title = paste("Residuals vs. Fitted for", model_name),
         x = "Fitted Values",
         y = "Residuals") +  # Labels and title
    theme_minimal()  # Minimal theme for a clean look
}

# Example usage of the function for different models
# Replace 'pois_model' with the actual model object
plot_resid_fitted(pois_model, "Poisson Model")
plot_resid_fitted(quas_poi, "Quasi-Poisson Model")
plot_resid_fitted(pois_model_poly, "Poisson Model (Quadratic)")
plot_resid_fitted(pois_model_poly3, "Poisson Model (Cubic)")
plot_resid_fitted(pois_model_seasonal, "Poisson Seasonal Model")
plot_resid_fitted(pois_model_lag, "Poisson Lagged Model")
plot_resid_fitted(glm_inverse, "Poisson Model with Inverse Link")
plot_resid_fitted(glm_log, "Poisson Model with Log Link")
plot_resid_fitted(glm_sqrt, "Poisson Model with Square Root Link")
plot_resid_fitted(gam_model, "Generalized Additive Model")
plot_resid_fitted(gam_model3, "GAM with Cubic Term")
```

```r
plot_resid_fitted(pois_model_lag_poly, "Poisson Model with Lag (Quadratic)")

# Define a function to calculate AIC and BIC for a given model
calculate_aic_bic <- function(model, model_name) {
  # Use built-in functions to calculate AIC and BIC
  aic_value <- AIC(model)
  bic_value <- BIC(model)

  # Return a named list with the model name, AIC, and BIC
  return(list(Model = model_name, AIC = aic_value, BIC = bic_value))
}

# Calculate AIC and BIC for a predefined list of models using the function
aic_bic_list <- list(
  calculate_aic_bic(pois_model, "Poisson Model"),
  calculate_aic_bic(pois_model_poly, "Poisson Polynomial (Quadratic)"),
  calculate_aic_bic(pois_model_poly3, "Poisson Polynomial (Cubic)"),
  calculate_aic_bic(pois_model_seasonal, "Poisson Seasonal"),
  calculate_aic_bic(pois_model_lag, "Poisson with Lagged Cases"),
  calculate_aic_bic(glm_inverse, "GLM with Inverse Link"),
  calculate_aic_bic(glm_log, "GLM with Log Link"),
  calculate_aic_bic(glm_sqrt, "GLM with Square Root Link"),
  calculate_aic_bic(gam_model, "Generalized Additive Model"),
  calculate_aic_bic(gam_model3, "GAM with Cubic Term"),
  calculate_aic_bic(pois_model_lag_poly, "Poisson Polynomial with Lag")
)

# Manually calculate AIC and BIC for the 'quas_poi' model
num_params <- length(coef(quas_poi))  # Includes intercept
n <- nrow(aids)  # Total observations
log_likelihood <- sum(dpois(aids$cases, lambda = predict(quas_poi, type = "response"), log = TRUE))
aic_quas_poi <- -2 * log_likelihood + 2 * num_params
bic_quas_poi <- -2 * log_likelihood + log(n) * num_params

# Convert the list of model AIC/BIC values to a data frame
aic_bic_results <- do.call(rbind, lapply(aic_bic_list, function(x) data.frame(Model = x$Model, AIC = x$

# Append the manually calculated AIC and BIC for 'quas_poi' to the results
aic_bic_results <- rbind(aic_bic_results, data.frame(Model = "Quasi-Poisson Model", AIC = aic_quas_poi,

# Order the results by the sum of AIC and BIC for each model as a proxy for combined criterion
aic_bic_results$Sum_AIC_BIC <- aic_bic_results$AIC + aic_bic_results$BIC
aic_bic_results_ordered <- aic_bic_results[order(aic_bic_results$Sum_AIC_BIC), ]

# Remove the auxiliary 'Sum_AIC_BIC' column and display the ordered results
aic_bic_results_ordered <- subset(aic_bic_results_ordered, select = -Sum_AIC_BIC)
print(aic_bic_results_ordered)

# Calculate RMSE for each model
predicted_pois_poly3 <- predict(pois_model_poly3, type = "response")
predicted_gam <- predict(gam_model, type = "response")
rmse_pois_poly3 <- rmse(aids$cases, predicted_pois_poly3)
rmse_gam <- rmse(aids$cases, predicted_gam)
```

```r
# Calculate the size of the dataset
data_size <- nrow(aids)
# Determine the size of the training set (e.g., 80% of the dataset)
train_size <- floor(0.8 * data_size)
# Randomly sample indices for the training data
train_indices <- sample(seq_len(data_size), size = train_size)
# Create training and test sets
trainData <- aids[train_indices, ]
testData <- aids[-train_indices, ]
# Now, you can fit your models to the training data and evaluate them on the test data
# Fitting models on the training data
pois_model_poly3 <- glm(cases ~ date + I(date^2) + I(date^3), data = trainData, family = poisson(link =
gam_model <- gam(cases ~ s(date), data = trainData, family = poisson(link = "log"))
# Predicting on training and test sets
trainPredict_pois_poly3 <- predict(pois_model_poly3, newdata = trainData, type = "response")
testPredict_pois_poly3 <- predict(pois_model_poly3, newdata = testData, type = "response")
trainPredict_gam <- predict(gam_model, newdata = trainData, type = "response")
testPredict_gam <- predict(gam_model, newdata = testData, type = "response")
# Define RMSE function
rmse <- function(actual, predicted) {
  sqrt(mean((predicted - actual)^2))
}
# Calculating RMSE for training and test sets
rmse_train_pois_poly3 <- rmse(trainData$cases, trainPredict_pois_poly3)
rmse_test_pois_poly3 <- rmse(testData$cases, testPredict_pois_poly3)
rmse_train_gam <- rmse(trainData$cases, trainPredict_gam)
rmse_test_gam <- rmse(testData$cases, testPredict_gam)

# Output the ordered AIC/BIC results
print(aic_bic_results_ordered)
# Perform ANOVA between GAM and Poisson model with cubic term
anova(gam_model, pois_model_poly3, test = "Chisq")
# Output RMSE values
print(paste("RMSE for Poisson Polynomial Model:", rmse_pois_poly3))
print(paste("RMSE for GAM Model:", rmse_gam))
# Output RMSE values for training and test sets
cat("Poisson Polynomial Model (Cubic) - Training RMSE:", rmse_train_pois_poly3, "\n")
cat("Poisson Polynomial Model (Cubic) - Test RMSE:", rmse_test_pois_poly3, "\n")
cat("GAM Model - Training RMSE:", rmse_train_gam, "\n")
cat("GAM Model - Test RMSE:", rmse_test_gam, "\n")

# 1. Normal model with log link
normal_model <- glm(cases ~ date, data = aids, family = gaussian(link = "log"))
aids$normal_predict <- predict(normal_model, type = "response")
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases), color = "blue", alpha = 0.5) +  # Original data points
  geom_line(aes(y = normal_predict), color = "red") +  # Use the predicted values for the line
  labs(title = "GAM Model Predictions vs. Actual Cases", y = "Number of Cases", x = "Date") +
  theme_minimal()


# 2. Polynomial models with log link
poly_nor <- glm(cases ~ date + I(date^2), data = aids, family = gaussian(link = "log"))
```

```r
aids$poly_nor <- predict(poly_nor, type = "response")
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases), color = "blue", alpha = 0.5) +  # Original data points
  geom_line(aes(y = poly_nor), color = "red") +  # Use the predicted values for the line
  labs(title = "GAM Model Predictions vs. Actual Cases", y = "Number of Cases", x = "Date") +
  theme_minimal()


poly_nor3 <- glm(cases ~ date + I(date^2) + I(date^3), data = aids, family = gaussian(link = "log"))
aids$poly_nor3 <- predict(poly_nor3, type = "response")
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases), color = "blue", alpha = 0.5) +  # Original data points
  geom_line(aes(y = poly_nor3), color = "red") +  # Use the predicted values for the line
  labs(title = "GAM Model Predictions vs. Actual Cases", y = "Number of Cases", x = "Date") +
  theme_minimal()


# 3. GAM model with log link
gam_nor <- gam(cases ~ s(date), data = aids, family = gaussian(link = "log"))
aids$gam_nor <- predict(gam_nor, type = "response")
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases), color = "blue", alpha = 0.5) +  # Original data points
  geom_line(aes(y = gam_nor), color = "red") +  # Use the predicted values for the line
  labs(title = "GAM Model Predictions vs. Actual Cases", y = "Number of Cases", x = "Date") +
  theme_minimal()


# 4. Seasonal model with log link
nor_seasonal <- glm(cases ~ date + factor(quarter), data = aids, family = gaussian(link = "log"))
aids$nor_seasonal <- predict(nor_seasonal, type = "response")
# Plotting for the seasonal model
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases), color = "blue", alpha = 0.5) +
  geom_line(aes(y = nor_seasonal), color = "green") +
  labs(title = "Seasonal Poisson Model", y = "Number of Cases", x = "Date") +
  theme_minimal()

# 6. Model with interaction and polynomial terms
model1 <- glm(cases ~ date * factor(quarter) + I(date^2) + I(date^3), data = aids, family = gaussian())
prediction_data_model1 <- predict(model1, newdata = aids, type = "response")

# 7. Model with interatcion
nor_inter <- glm(cases ~ date * factor(quarter), data = aids, family = gaussian())
# Create a new data frame for predictions as before
date_range <- seq(min(aids$date), max(aids$date), length.out = 100)
quarters <- unique(aids$quarter)
prediction_data <- expand.grid(date = date_range, quarter = quarters)
# Predict cases using the interaction model
prediction_data$cases_predicted <- predict(nor_inter, newdata = prediction_data, type = "response")
# Convert 'quarter' in 'aids' to a factor if it's not already, to match 'prediction_data'
aids$quarter <- factor(aids$quarter)
# Plotting with observed data points and predicted trends
ggplot() +
```

```
  geom_point(data = aids, aes(x = date, y = cases, color = quarter), alpha = 0.5) +
  geom_line(data = prediction_data, aes(x = date, y = cases_predicted, color = quarter)) +
  labs(title = "Predicted Cases with Interaction between Date and Quarter",
       x = "Date",
       y = "Predicted/Observed Number of Cases") +
  theme_minimal() +
  scale_color_brewer(palette = "Set1", name = "Quarter")

# Function to Create Q-Q Plots for Model Residuals
create_qqplot <- function(model, data, model_name) {
  # Calculate residuals from the model
  residuals <- resid(model)

  # Generate the QQ plot
  qqnorm(residuals)  # QQ plot against the theoretical normal distribution
  qqline(residuals, col = "red")  # Add a reference line

  # Add titles using model_name
  title(main = paste("QQ Plot of Residuals for", model_name),
        sub = "Comparing to Standard Normal Distribution")
}

# Apply the QQ plot function to various models
create_qqplot(normal_model, aids, "Normal Model")
create_qqplot(poly_nor, aids, "Polynomial Model (Quadratic)")
create_qqplot(poly_nor3, aids, "Polynomial Model (Cubic)")
create_qqplot(gam_nor, aids, "GAM")
create_qqplot(nor_seasonal, aids, "Seasonal Model")
create_qqplot(nor_inter, aids, "Normal Model with Interaction")
create_qqplot(model1, aids, "Complex Model")

# Define a list of models for comparison
model_list <- list(normal_model, poly_nor, poly_nor3, gam_nor, nor_seasonal, nor_inter, model1)
model_names <- c("Normal Model", "Polynomial Model 2nd Degree", "Polynomial Model 3rd Degree", "GAM Mod

# Calculate AIC and BIC for each model
aic_values <- sapply(model_list, AIC)
bic_values <- sapply(model_list, BIC)

# Sample size for the log-likelihood calculation
n <- nrow(aids)

# Combine AIC and BIC values including the GBM model
results <- data.frame(Model = c(model_names),
                      AIC = c(aic_values),
                      BIC = c(bic_values))

# Sort results by the sum of AIC and BIC for each model in ascending order
results$Combined <- results$AIC + results$BIC
results_sorted <- results[order(results$Combined), ]

# Remove the temporary 'Combined' column before displaying the results
results_sorted$Combined <- NULL
```

```r
# Predicting on training and test sets
trainPredict_poly_nor3 <- predict(poly_nor3, newdata = trainData, type = "response")
testPredict_poly_nor3 <- predict(poly_nor3, newdata = testData, type = "response")
trainPredict_gam_nor <- predict(gam_nor, newdata = trainData, type = "response")
testPredict_gam_nor <- predict(gam_nor, newdata = testData, type = "response")
# Define RMSE function
rmse <- function(actual, predicted) {
  sqrt(mean((predicted - actual)^2))
}
# Calculating RMSE for training and test sets
rmse_train_poly_nor3 <- rmse(trainData$cases, trainPredict_poly_nor3)
rmse_test_poly_nor3 <- rmse(testData$cases, testPredict_poly_nor3)
rmse_train_gam_nor <- rmse(trainData$cases, trainPredict_gam_nor)
rmse_test_gam_nor <- rmse(testData$cases, testPredict_gam_nor)

print(results_sorted)
anova(gam_nor, poly_nor3, test = "Chisq")
# Output RMSE values
cat("Normal Polynomial Model (Cubic) - Training RMSE:", rmse_train_poly_nor3, "\n")
cat("Normal Polynomial Model (Cubic) - Test RMSE:", rmse_test_poly_nor3, "\n")
cat("GAM Model - Training RMSE:", rmse_train_gam_nor, "\n")
cat("GAM Model - Test RMSE:", rmse_test_gam_nor, "\n")

# Plotting predictions from different models
ggplot(aids, aes(x = date)) +
  geom_point(aes(y = cases, color = "Actual Cases"), alpha = 0.5) +  # Actual cases
  geom_line(aes(y = pred_poly3, color = "Poisson Model")) +  # Poisson model predictions
  geom_line(aes(y = poly_nor3, color = "Normal Model"), linetype = "dashed") +  # Normal model predicti
  scale_color_manual(name = "Legend",
                     values = c("Poisson Model" = "red", "Normal Model" = "blue")) +  # Customize legen
  labs(title = "Cubic Polynomial Poisson Model vs Cubic Polynomial Normal Model", y = "Number of Cases"
  theme_minimal() +  # Minimal theme for a clean look
  theme(legend.position = "bottom")  # Adjust legend position to bottom

# Fit a Negative Binomial model to the data
nb_model_poly3 <- glm.nb(cases ~ date + I(date^2) + I(date^3), data = aids)

# Calculate AIC and BIC for the Negative Binomial model
aic <- AIC(nb_model_poly3)
bic <- BIC(nb_model_poly3)
print(paste("AIC:", aic))
print(paste("BIC:", bic))

# Predicting on training and test sets
trainnb <- predict(nb_model_poly3, newdata = trainData, type = "response")
testnb <- predict(nb_model_poly3, newdata = testData, type = "response")

# Define RMSE function
rmse <- function(actual, predicted) {
  sqrt(mean((predicted - actual)^2))
}

# Calculating RMSE for training and test sets
```

```r
rmsenb_train <- rmse(trainData$cases, trainnb)
rmsenb_test <- rmse(testData$cases, testnb)

# Output RMSE values for Negative Binomial model
cat("Neg Binom - Training RMSE:", rmsenb_train, "\n")
cat("Neg Binom - Test RMSE:", rmsenb_test, "\n")
```