Coventry University

Faculty of Engineering, Environment and Computing

United Kingdom

A report on

Modeling Brain Responses to Music Using Nonlinear Regression

For STW7089CEM: Introduction to Statistical Methods for Data Science

Submitted by

Dipin Baral

Submitted to

Shrawan Thakur

Date of Submission:

22nd December, 2024

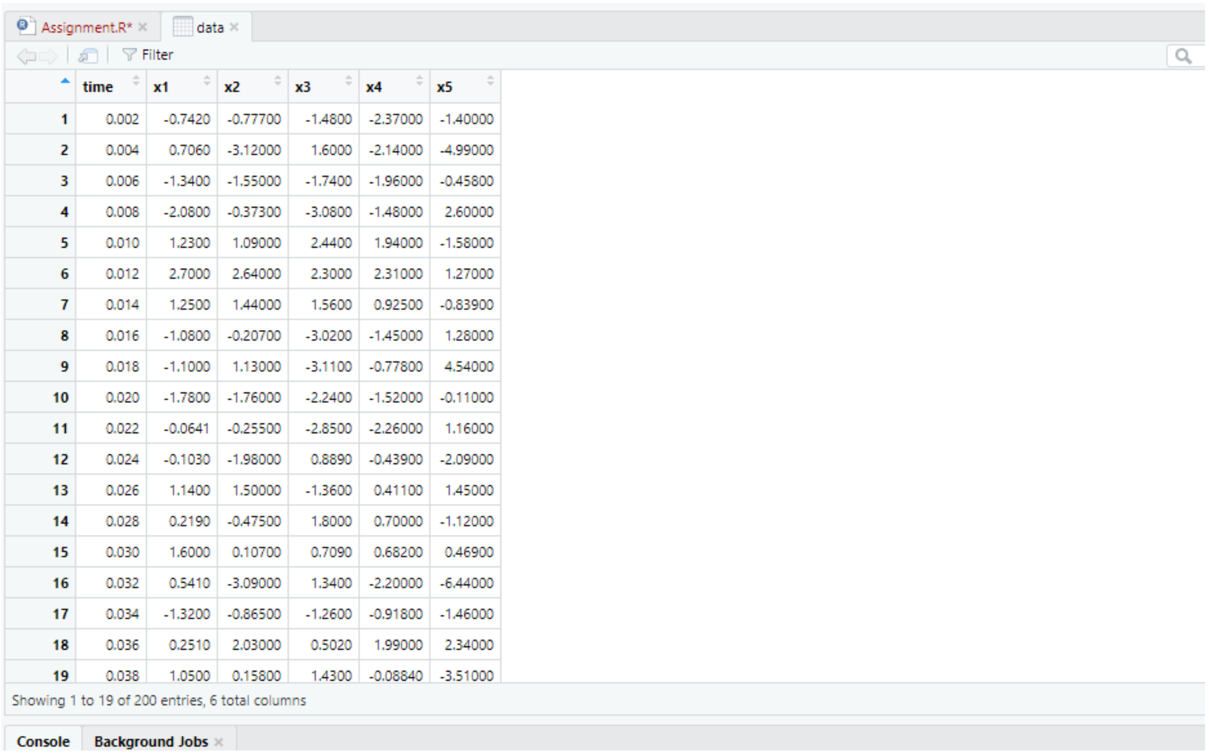
Table of Contents

Figure

Introduction

Modeling brain reactions to music through nonlinear regression entails examining the effect of musical components such as rhythm, melody, and harmony on neural responses. (Jäncke & Sandmann, 2010) Nonlinear regression is especially advantageous as it can identify complex, non-linear correlations between musical stimuli and brain responses that linear models fail to capture. (Koelsch, 2010) Utilizing nonlinear regression on neuroimaging data (such as fMRI and EEG), researchers can reveal the interactions between various musical characteristics and brain networks, offering enhanced understanding of emotional, cognitive, and perceptual mechanisms. (Henson & Friston, 2006).

The scenario presents data for four input signals: X1, X3, X4, and X5, as well as one output signal, X2. The time variation is also provided in seconds as follows:



	time	x1	x2	x3	x4	x5
1	0.002	-0.7420	-0.77700	-1.4800	-2.37000	-1.40000
2	0.004	0.7060	-3.12000	1.6000	-2.14000	-4.99000
3	0.006	-1.3400	-1.55000	-1.7400	-1.96000	-0.45800
4	0.008	-2.0800	-0.37300	-3.0800	-1.48000	2.60000
5	0.010	1.2300	1.09000	2.4400	1.94000	-1.58000
6	0.012	2.7000	2.64000	2.3000	2.31000	1.27000
7	0.014	1.2500	1.44000	1.5600	0.92500	-0.83900
8	0.016	-1.0800	-0.20700	-3.0200	-1.45000	1.28000
9	0.018	-1.1000	1.13000	-3.1100	-0.77800	4.54000
10	0.020	-1.7800	-1.76000	-2.2400	-1.52000	-0.11000
11	0.022	-0.0641	-0.25500	-2.8500	-2.26000	1.16000
12	0.024	-0.1030	-1.98000	0.8890	-0.43900	-2.09000
13	0.026	1.1400	1.50000	-1.3600	0.41100	1.45000
14	0.028	0.2190	-0.47500	1.8000	0.70000	-1.12000
15	0.030	1.6000	0.10700	0.7090	0.68200	0.46900
16	0.032	0.5410	-3.09000	1.3400	-2.20000	-6.44000
17	0.034	-1.3200	-0.86500	-1.2600	-0.91800	-1.46000
18	0.036	0.2510	2.03000	0.5020	1.99000	2.34000
19	0.038	1.0500	0.15800	1.4300	-0.08840	-3.51000

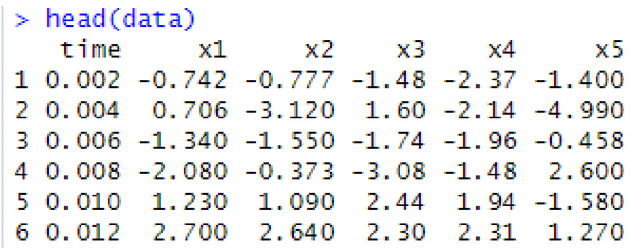
Showing 1 to 19 of 200 entries, 6 total columns

Figure 1: Sample of data

Task 1

1.1 Time series plots (of input and output signal)

Objective: To illustrate the time-related patterns in both input (musical stimuli) and output (brain response) signals.



```
> head(data)
   time     x1     x2    x3    x4     x5
1 0.002 -0.742 -0.777 -1.48 -2.37 -1.400
2 0.004  0.706 -3.120  1.60 -2.14 -4.990
3 0.006 -1.340 -1.550 -1.74 -1.96 -0.458
4 0.008 -2.080 -0.373 -3.08 -1.48  2.600
5 0.010  1.230  1.090  2.44  1.94 -1.580
6 0.012  2.700  2.640  2.30  2.31  1.270
```

Figure 2: Sample data of X (input signals) and time

A time series plot can display the input signals of X as time progresses in a linear fashion.

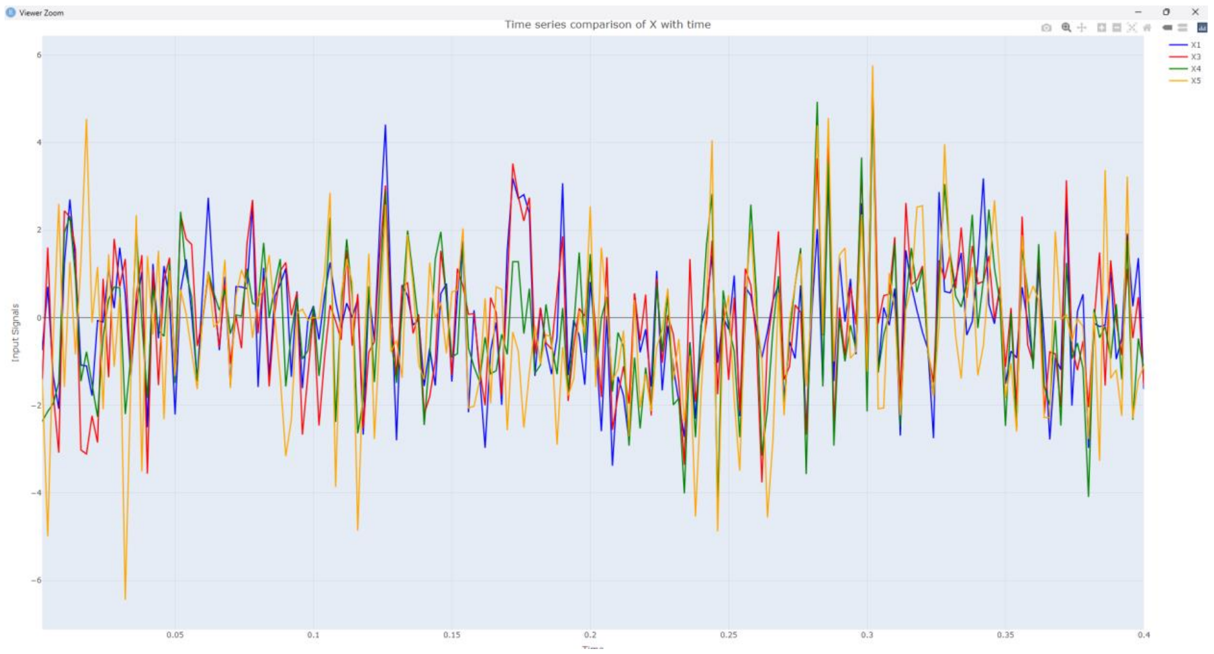


Figure 3: Time series plot of all X inputs

In the above figure, each of the frequencies of X exhibits a comparable frequency spectrum, which may foster the perception that the input data for Brain reaction which is relatively distributed. To enhance comprehension, the time series plot can be further divided into each interpretation as follows.

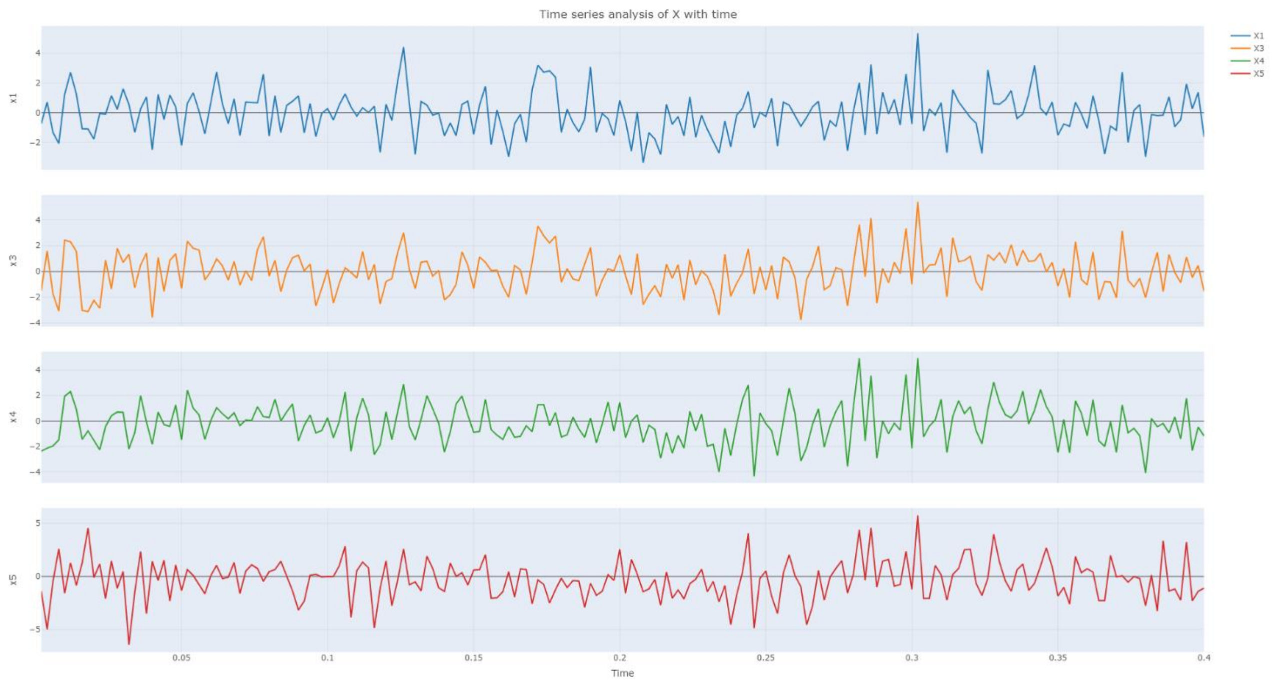
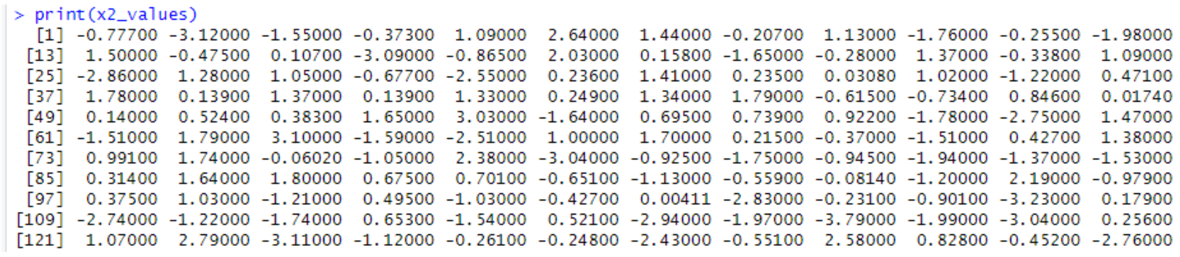


Figure 4: Time series plot of X inputs individually

This time series plot shows four individual signals (X1, X3, X4, X5) plotted against time. These signals can be interpreted in the context of brain signals, where different channels or brain regions generate time-varying electrical activity and it represents as output brain signals.

In addition to input time series analysis, context of brain signals output values can also be plotted for x2.



```
> print(x2_values)
  [1] -0.77700 -3.12000 -1.55000 -0.37300  1.09000  2.64000  1.44000 -0.20700  1.13000 -1.76000 -0.25500 -1.98000
 [13]  1.50000 -0.47500  0.10700 -3.09000 -0.86500  2.03000  0.15800 -1.65000 -0.28000  1.37000 -0.33800  1.09000
 [25] -2.86000  1.28000  1.05000 -0.67700 -2.55000  0.23600  1.41000  0.23500  0.03080  1.02000 -1.22000  0.47100
 [37]  1.78000  0.13900  1.37000  0.13900  1.33000  0.24900  1.34000  1.79000 -0.61500 -0.73400  0.84600  0.01740
 [49]  0.14000  0.52400  0.38300  1.65000  3.03000 -1.64000  0.69500  0.73900  0.92200 -1.78000 -2.75000  1.47000
 [61] -1.51000  1.79000  3.10000 -1.59000 -2.51000  1.00000  1.70000  0.21500 -0.37000 -1.51000  0.42700  1.38000
 [73]  0.99100  1.74000 -0.06020 -1.05000  2.38000 -3.04000 -0.92500 -1.75000 -0.94500 -1.94000 -1.37000 -1.53000
 [85]  0.31400  1.64000  1.80000  0.67500  0.70100 -0.65100 -1.13000 -0.55900 -0.08140 -1.20000  2.19000 -0.97900
 [97]  0.37500  1.03000 -1.21000  0.49500 -1.03000 -0.42700  0.00411 -2.83000 -0.23100 -0.90100 -3.23000  0.17900
[109] -2.74000 -1.22000 -1.74000  0.65300 -1.54000  0.52100 -2.94000 -1.97000 -3.79000 -1.99000 -3.04000  0.25600
[121]  1.07000  2.79000 -3.11000 -1.12000 -0.26100 -0.24800 -2.43000 -0.55100  2.58000  0.82800 -0.45200 -2.76000
```

Figure 5: Data sample of x2 output

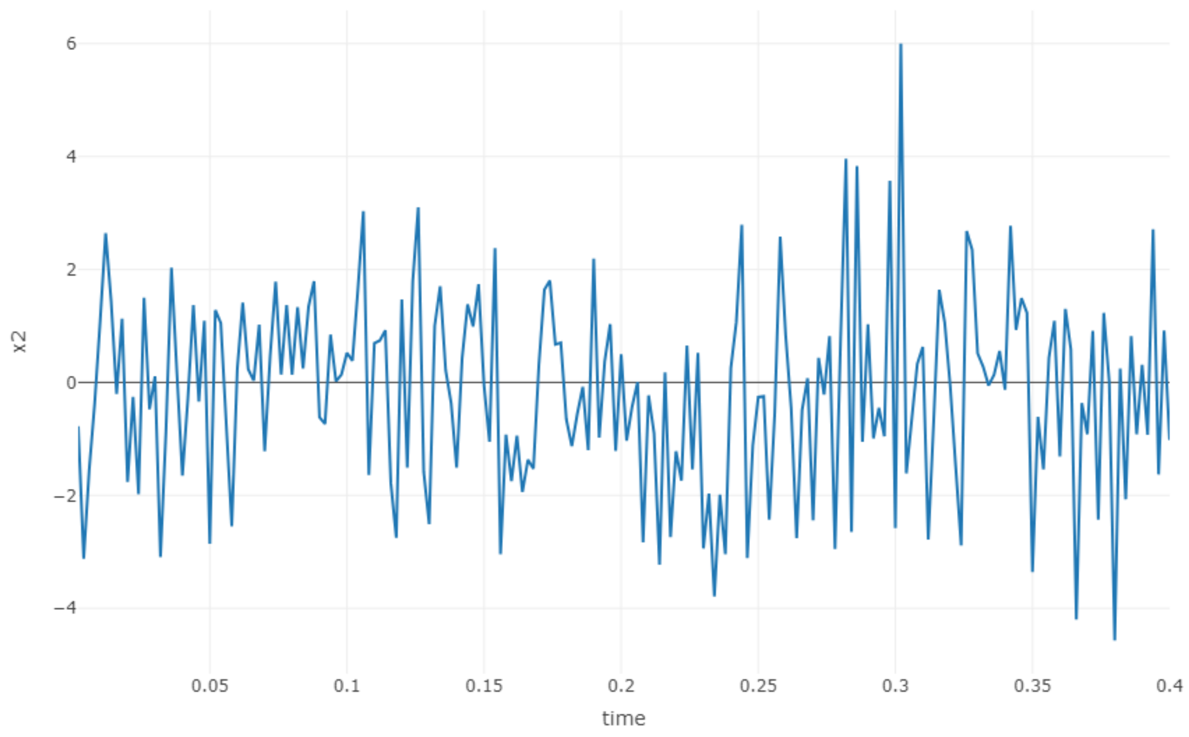


Figure 6: Time series plot of X2

This above plot illustrates, with frequent and noticeable spikes, the first half (0–0.25 seconds) shows a period of high activity that peaks at 0.15 seconds. The signal stabilizes in the second half (0.25–0.4 seconds), exhibiting less frequent fluctuations and a smaller amplitude. This behavior is typical of brain signal recordings, where periods of increased signal fluctuations before stabilizing can be caused by external stimuli or transient neural activity.

1.2: Distribution for each signal (time-series)

When referring to time-series signals, distribution means the statistical pattern or spread of values over time that reflects the signal's general behavior. In order to aid in analysis and prediction, common distributions such as Poisson, exponential, or normal describe the properties of the signal. These distributions are frequently identified through time-series analysis in order to comprehend trends, variability, and forecasts (Shumway and Stoffer, 2017).

Known as density plots, the bell-shaped curves for each input X in the figure below show the areas with the highest concentration of values in the dataset.

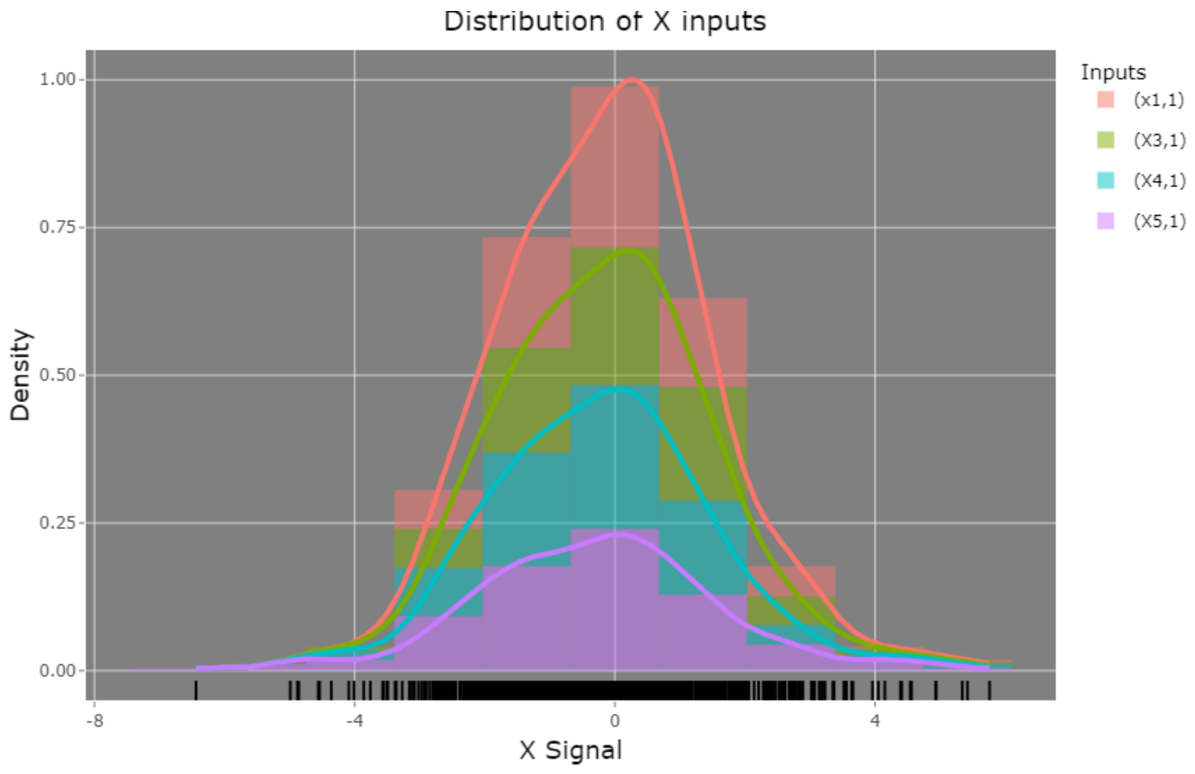


Figure 7: Distribution of X input

The dataset's X input distribution is depicted in the image above. The data point density for each input is shown on the graph. Four different colors are used to represent the inputs (X1, X3, X4, and X5). Each input has a similar distribution that resembles a bell-shaped curve, suggesting that it is roughly normally distributed. The majority of data points for all inputs cluster around zero. For some inputs (X1 and X3), the data point spread is greater than for others (X4 and X5). This implies that there might be a greater variance or spread in the values of some inputs than others.

Similarly, the density distribution is further split down for each X input data as follows.

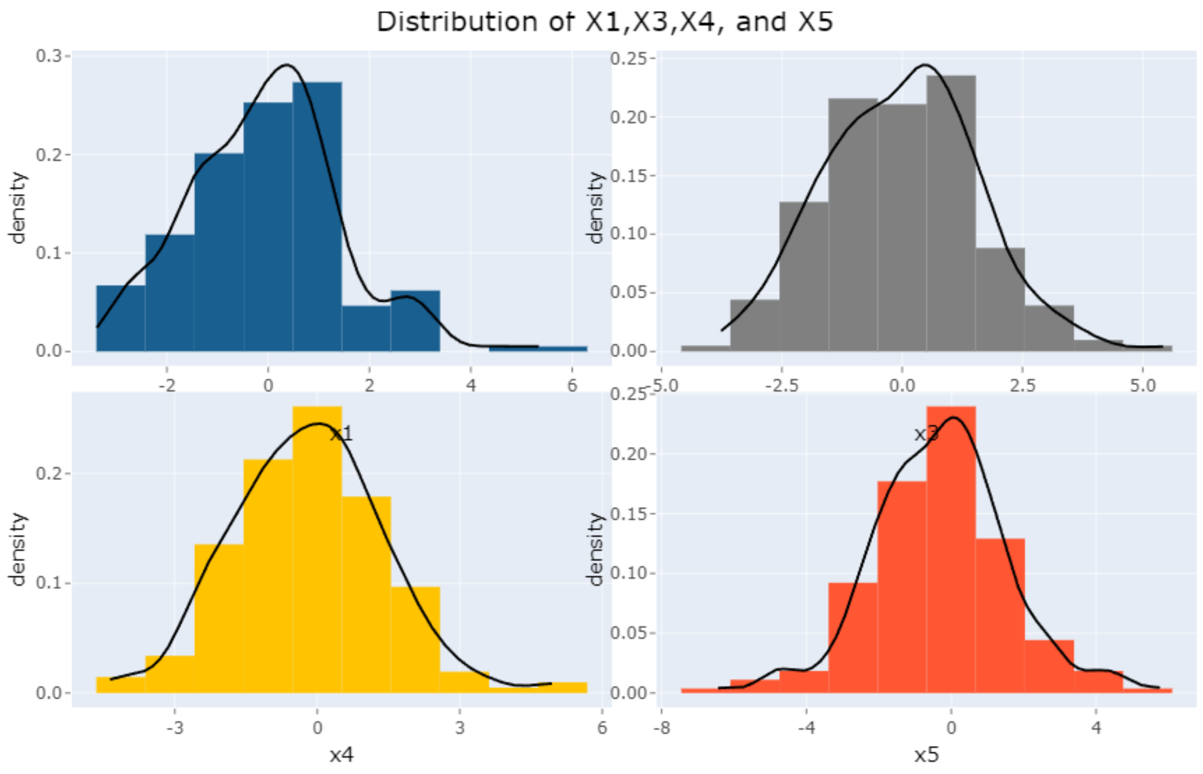


Figure 8: Distribution of individual X inputs

The shapes and tendencies of the signals X1, X3, X4, and X5 distributions show unique features. With peaks around 1, X1 and X5 both show right skewness. Their density curves indicate that they have a normal distribution with means marginally higher than 0. Conversely, X3 exhibits a leftward skew, peaking at approximately 0, suggesting a propensity for negative values and a mean marginally below zero. The symmetrical distribution of X4, which is centered around 0, makes it stand out and suggests more stable data. All things considered, these findings show that the signals exhibit different degrees of variability and skewness, with X4 being the most stable, X1 and X5 having comparable variability, and X3 showing a clear tendency toward negative values.

The same density distribution for output signal X2 can be plotted below.

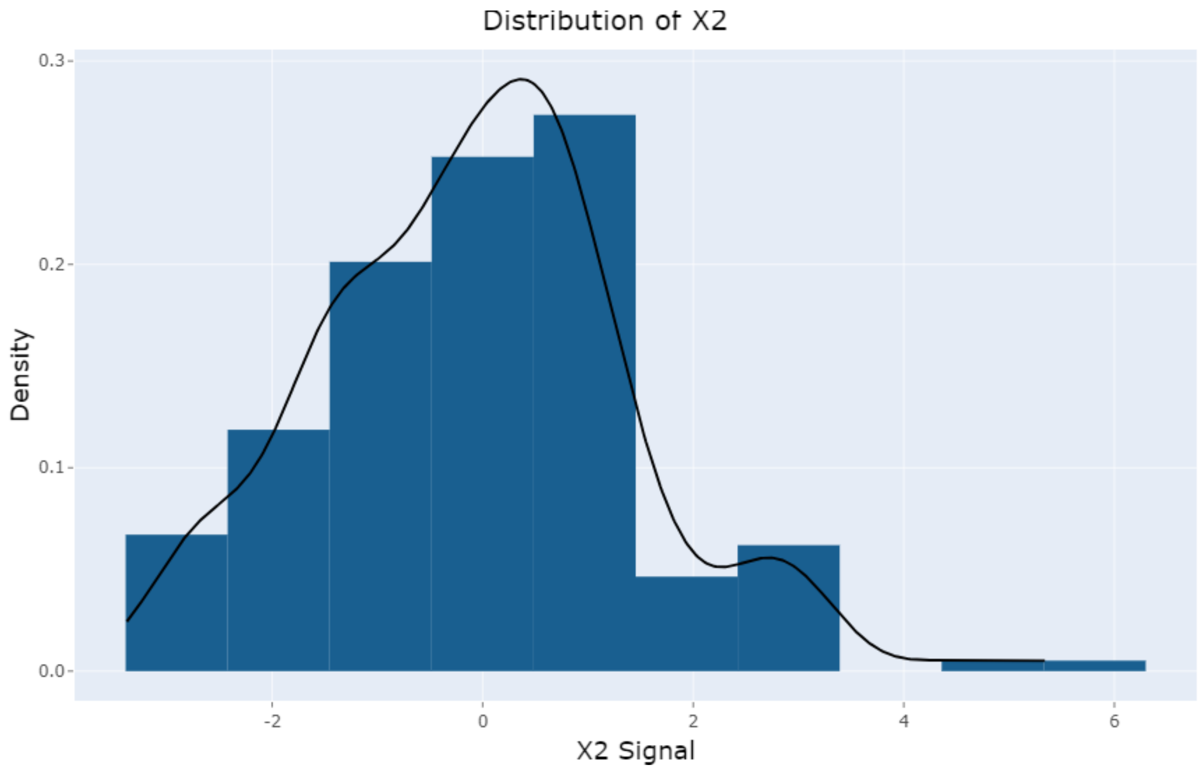


Figure 9: Distribution of X2 output

X2 signals are distributed as shown in the above figure. The majority of data points appear to fall on the right side of the distribution, suggesting that the distribution is slightly skewed to the right. Numerous X2 signals appear to be concentrated around the value of 0, as indicated by the distribution's peak. Some X2 signals, however, have higher values, as indicated by the longer tail that extends to the right. According to the distribution, X2 signals are mostly grouped around 0 with a small number of outliers having higher values.

1.3: Correlation and scatter plots

The correlation between two variables refers to the statistical connection, indicating how one variable may fluctuate in response to changes in the other. Plots of scatter are graphical tools that depict individual data points for both variables, enabling a visual evaluation of their correlation. A positive correlation implies that as one variable rises, so does the other, while a negative correlation suggests an opposite relationship. The intensity and direction of this correlation can be quantified using correlation coefficients, for instance, Pearson's r (Field, 2018).

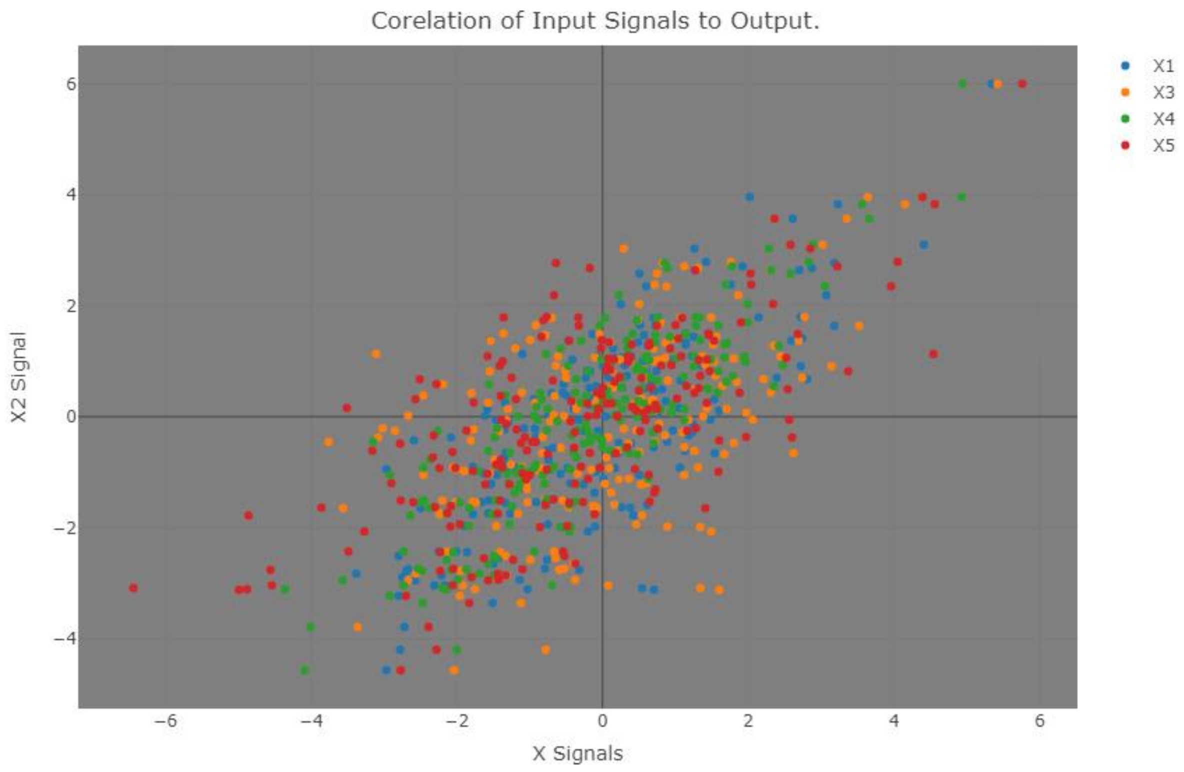


Figure 10: Correlation of all the X inputs with x2 output

Each scatterplot of x inputs with respect to x2 output has been broken down for a better understanding of each X input.

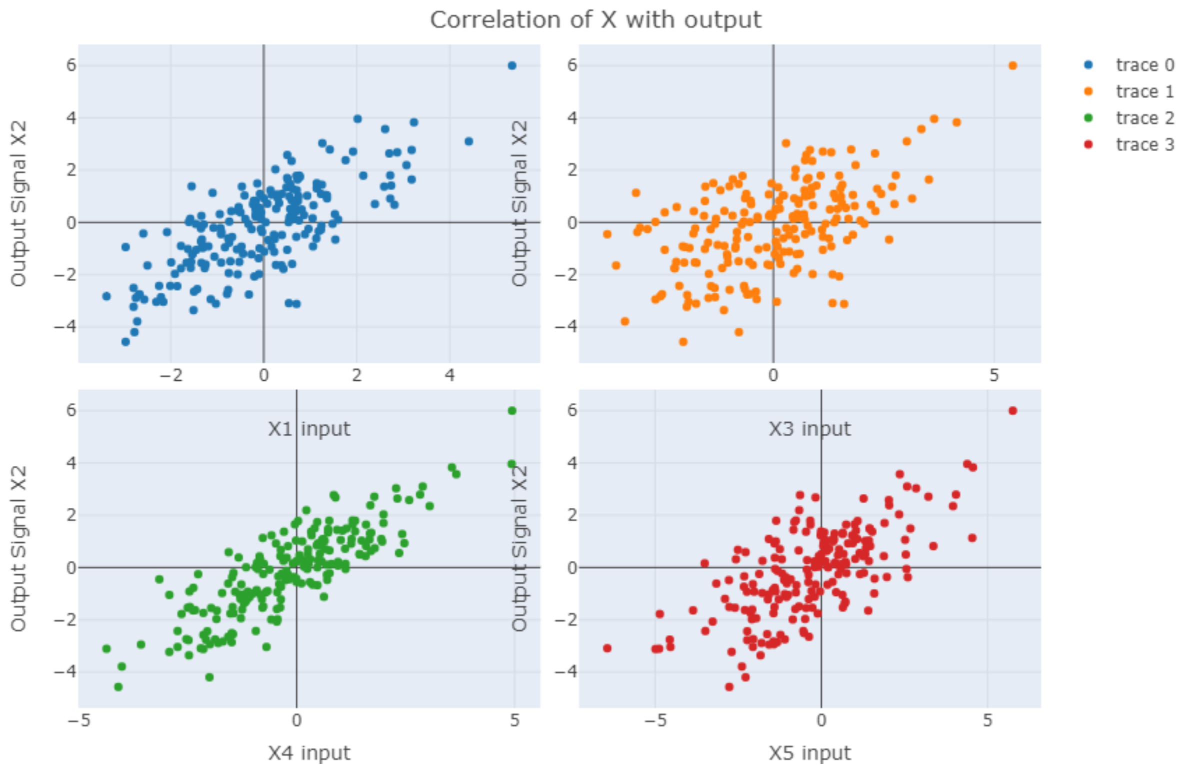


Figure 11: Correlation of all the X input individually with X2 output

The picture shows how the output signal (X2) and the four input variables (X1, X3, X4, and X5) are correlated. Plotted against the output signal, each subplot depicts a distinct input variable. The plots show that X1 and X2 have a direct relationship, with a strong positive correlation. A weak positive correlation between X3 and X2 indicates a less significant relationship. An intermediately positive correlation is shown between X4 and X2. A weak positive correlation between X5 and X2 and a few outliers may indicate an unanticipated influence. In general, the plots show how the input variables and the output signal interact in a complex way, offering important insights into how the system behaves.

Task 2: Regression

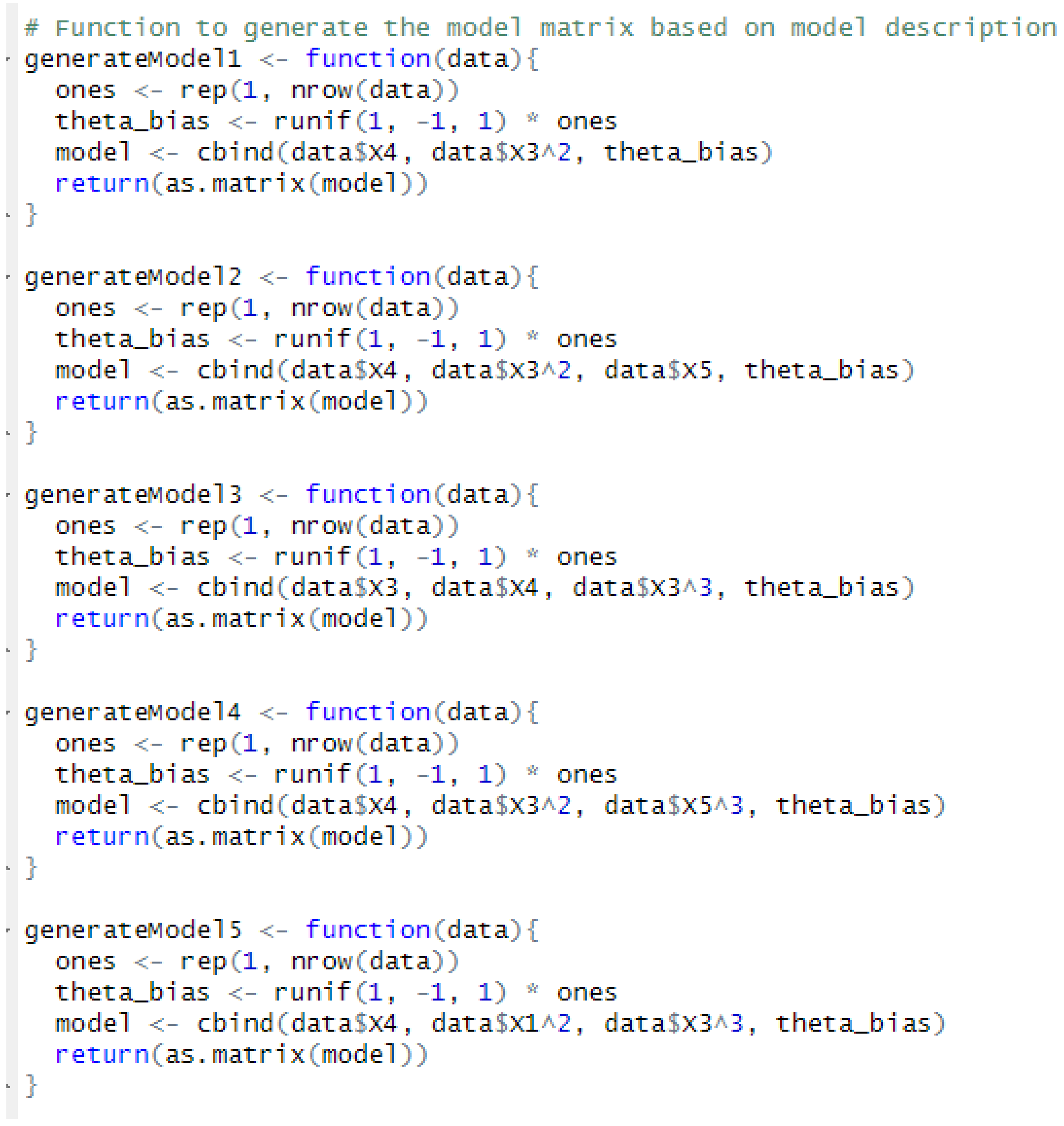
2.1: Regression Model

A statistical method for analyzing the relationship between one or more independent variables and a dependent variable is regression. By enabling predictions and comprehending data trends, it assists in figuring out how changes in predictors affect the result (Montgomery et al. (2012). This idea is expanded upon by nonlinear regression, which models intricate relationships—like polynomial terms—to more accurately depict data from the real world (Seber and Wild, 2003). When linear models are inadequate for identifying patterns, this approach is essential.

$$\text{Model 1:}\quad y = \theta_1 x_4 + \theta_2 x_3^2 + \theta_{bias}$$
$$\text{Model 2:}\quad y = \theta_1 x_4 + \theta_2 x_3^2 + \theta_3 x_5 + \theta_{bias}$$
$$\text{Model 3:}\quad y = \theta_1 x_3 + \theta_2 x_4 + \theta_3 x_5^3$$
$$\text{Model 4:}\quad y = \theta_1 x_4 + \theta_2 x_3^2 + \theta_3 x_5^3 + \theta_{bias}$$
$$\text{Model 5:}\quad y = \theta_1 x_4 + \theta_2 x_1^2 + \theta_3 x_3^2 + \theta_{bias}$$

Figure 12: Regression model equation of all the model

For instance, the Model 1 has the following parameters: X4, X3, and theta bias. The X inputs chosen are from the Modeling Brain Responses to Music signals, and the noise is represented by random values produced by a normal distribution with the specified mean and variance in the specified space. The bias is represented by a random constant value between -1 and 1.



```r
# Function to generate the model matrix based on model description
generateModel1 <- function(data){
  ones <- rep(1, nrow(data))
  theta_bias <- runif(1, -1, 1) * ones
  model <- cbind(data$X4, data$X3^2, theta_bias)
  return(as.matrix(model))
}

generateModel2 <- function(data){
  ones <- rep(1, nrow(data))
  theta_bias <- runif(1, -1, 1) * ones
  model <- cbind(data$X4, data$X3^2, data$X5, theta_bias)
  return(as.matrix(model))
}

generateModel3 <- function(data){
  ones <- rep(1, nrow(data))
  theta_bias <- runif(1, -1, 1) * ones
  model <- cbind(data$X3, data$X4, data$X3^3, theta_bias)
  return(as.matrix(model))
}

generateModel4 <- function(data){
  ones <- rep(1, nrow(data))
  theta_bias <- runif(1, -1, 1) * ones
  model <- cbind(data$X4, data$X3^2, data$X5^3, theta_bias)
  return(as.matrix(model))
}

generateModel5 <- function(data){
  ones <- rep(1, nrow(data))
  theta_bias <- runif(1, -1, 1) * ones
  model <- cbind(data$X4, data$X1^2, data$X3^3, theta_bias)
  return(as.matrix(model))
}
```

Figure 13: Model representation of all the models in R

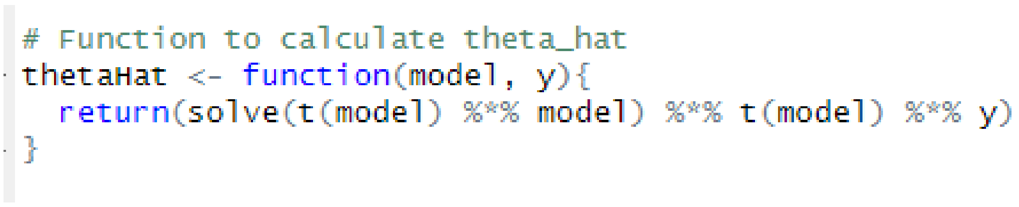
The functions in the above R script represents the data for regression tasks by creating model matrices with bias terms and varying polynomial terms. Every function (generateModel1 to generateModel5) creates distinct feature matrices. For example, Model1 contains X4, X3 squared (X3^2), and a random bias term. Model 2: Incorporates X4, X3^2, X5, and a bias term into Model 1 in addition to X5. Model 3: Contains X3^3 (cubic term) in addition to X3, X4, and bias. Model4: X5^3 is added to Model2 to create X4, X3^3, X5^3, and bias. Model 5: Extended feature extraction using X4, X1^2, X3^3, and bias.

Polynomial regression is the intended application for these models (James et al., 2013) where nonlinear relationships can be captured through variable transformations.

The least-square error is represented as "θ" and the equation for it is as follows:

θ^=(X^T*X)^-1X^T*y

The output signal "x2" from the Brain signal is indicated by the y parameter, while the X parameter represents the model estimator. This can be shown in R as follows.

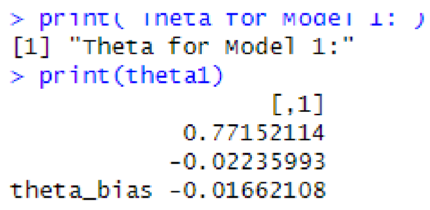


```
# Function to calculate theta_hat
thetaHat <- function(model, y){
  return(solve(t(model) %*% model) %*% t(model) %*% y)
}
```

Figure 14: Listing the Theta hat values from the model

With matrix algebra, the thetaHat function uses the ordinary least squares (OLS) formula θ^=(XTX)−1XTyθ^ to estimate coefficients in a linear regression model. (Weisberg, 2005)

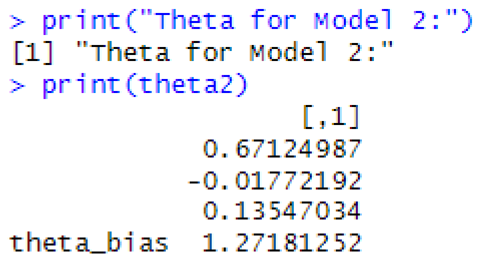
For each model, the theta hat and model value can shown below:



```
> print("Theta for Model 1:")
[1] "Theta for Model 1:"
> print(theta1)
                   [,1]
             0.77152114
            -0.02235993
theta_bias  -0.01662108
```

Figure 15: Model 1 theta hat and theta bias

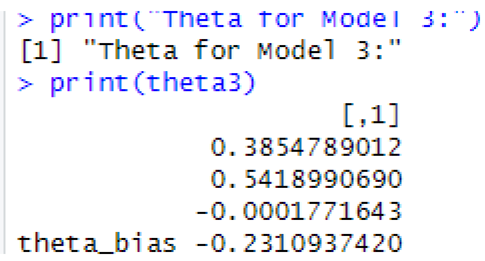
Theta1's output shows the model 1 estimated coefficients. Intercept, slope, and bias terms obtained through OLS estimation are among the values. (Weisberg, 2005)



```
> print("Theta for Model 2:")
[1] "Theta for Model 2:"
> print(theta2)
                   [,1]
             0.67124987
            -0.01772192
             0.13547034
theta_bias   1.27181252
```

Figure 16: Model 2 theta hat and theta bias

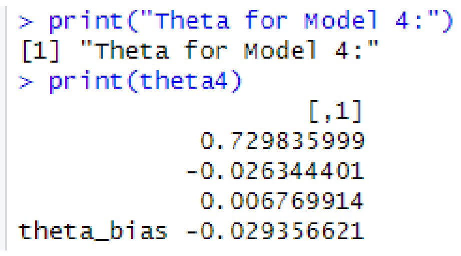
The estimated coefficients for model 2, including intercept, slopes, and other parameters determined through OLS estimation, are included in the theta2 output. (Weisberg, 2005)



```
> print("Theta for Model 3:")
[1] "Theta for Model 3:"
> print(theta3)
                     [,1]
             0.3854789012
             0.5418990690
            -0.0001771643
theta_bias  -0.2310937420
```

Figure 17: Model 3 theta hat and theta bias

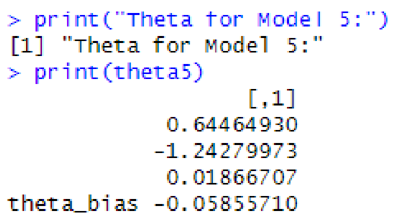
Using the OLS method for linear regression, the theta3 output gives the estimated coefficients for model3, such as intercept, slopes, and bias. (Weisberg, 2005)



```
> print("Theta for Model 4:")
[1] "Theta for Model 4:"
> print(theta4)
                   [,1]
             0.729835999
            -0.026344401
             0.006769914
theta_bias -0.029356621
```

Figure 18: Model 4 theta hat and theta bias

Using ordinary least squares (OLS) regression, the theta4 output displays the estimated coefficients for model 4, including intercept, slopes, and bias. (Weisberg, 2005)



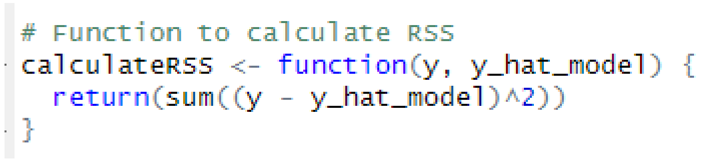
```
> print("Theta for Model 5:")
[1] "Theta for Model 5:"
> print(theta5)
                  [,1]
            0.64464930
           -1.24279973
            0.01866707
theta_bias -0.05855710
```

Figure 19: Model 5 theta hat and theta bias

Using generateModel5 and thetaHat, the code calculates the theta parameters, generating estimates for three values as well as a bias term that shows the performance of the model. (Weisberg, 2005)

2.2: Residual Sum of Squares

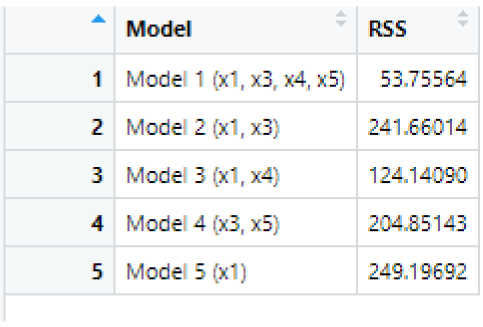
The error between observed values (y i) and predicted values (y ^ i) in a regression model is measured by the Residual Sum of squares (RSS). A smaller RSS suggests a better fit, as it implies the model predictions closely match the actual data. It is calculated as the sum of squared differences, which is given by $R\ S\ S = \sum\ i = 1\ n\ (\ y\ i - y\ \hat{}\ i\ )\ 2$ RSS= ∑i=1n(yi−y^i)2. A crucial metric for assessing model performance and contrasting models is RSS. Nevertheless, it does not penalize overfitting and is sensitive to the size of the data. Metrics like RMSE or R-squared may be preferred for normalized error. (Draper & Smith, 1998).



```
# Function to calculate RSS
calculateRSS <- function(y, y_hat_model) {
  return(sum((y - y_hat_model)^2))
}
```

Figure 20: Function to calculate the RSS

For each model, the RSS can be found below:



	Model	RSS
1	Model 1 (x1, x3, x4, x5)	53.75564
2	Model 2 (x1, x3)	241.66014
3	Model 3 (x1, x4)	124.14090
4	Model 4 (x3, x5)	204.85143
5	Model 5 (x1)	249.19692

Figure 21: RSS values for each model

2.3: Log-likelihood Function

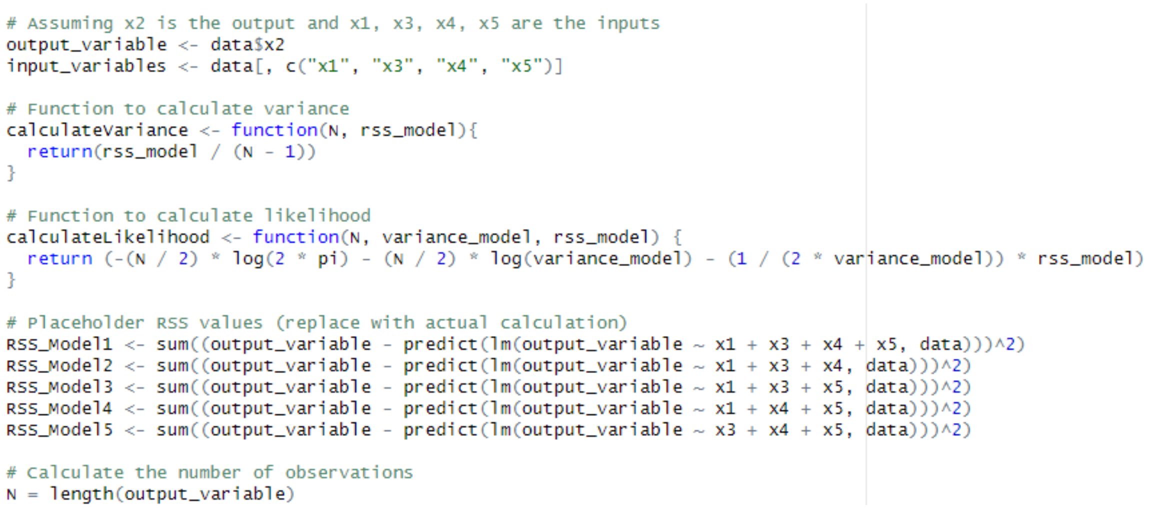
The probability of observing the given data under a particular statistical model is represented by the logarithm of the likelihood function, which is the log-likelihood function. To determine the parameter values that optimize the likelihood of the observed data, maximum likelihood estimation, or MLE, frequently uses this technique. The log transformation converts products into sums and helps prevent numerical underflow, which makes computations easier, particularly when working with big data sets or complex models. (Bishop, 2006)

The likelihood function, given the parameters of a model, represents the probability of observing a particular set of data.

$$\ln p(D|\hat{\theta}) = -\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln(\hat{\sigma}^2) - \frac{1}{2\hat{\sigma}^2}\text{RSS}$$

Figure 22: Equation for Likelihood

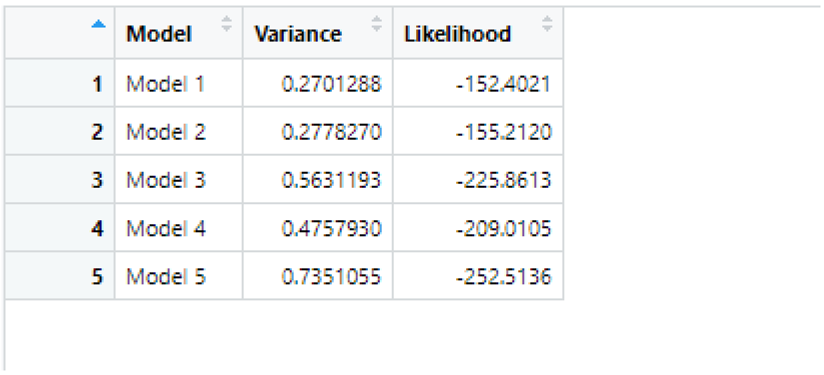
The above figure 22 equation can be represented in R as follows:



```
# Assuming x2 is the output and x1, x3, x4, x5 are the inputs
output_variable <- data$x2
input_variables <- data[, c("x1", "x3", "x4", "x5")]

# Function to calculate variance
calculateVariance <- function(N, rss_model){
  return(rss_model / (N - 1))
}

# Function to calculate likelihood
calculateLikelihood <- function(N, variance_model, rss_model) {
  return (-(N / 2) * log(2 * pi) - (N / 2) * log(variance_model) - (1 / (2 * variance_model)) * rss_model)
}

# Placeholder RSS values (replace with actual calculation)
RSS_Model1 <- sum((output_variable - predict(lm(output_variable ~ x1 + x3 + x4 + x5, data)))^2)
RSS_Model2 <- sum((output_variable - predict(lm(output_variable ~ x1 + x3 + x4, data)))^2)
RSS_Model3 <- sum((output_variable - predict(lm(output_variable ~ x1 + x3 + x5, data)))^2)
RSS_Model4 <- sum((output_variable - predict(lm(output_variable ~ x1 + x4 + x5, data)))^2)
RSS_Model5 <- sum((output_variable - predict(lm(output_variable ~ x3 + x4 + x5, data)))^2)

# Calculate the number of observations
N = length(output_variable)
```

Figure 23: Representation of Variance and Likelihood in R

The results for each model from (model 1 to model 5) are as follows:



	Model	Variance	Likelihood
1	Model 1	0.2701288	-152.4021
2	Model 2	0.2778270	-155.2120
3	Model 3	0.5631193	-225.8613
4	Model 4	0.4757930	-209.0105
5	Model 5	0.7351055	-252.5136

Figure 24: Variance and likelihood data

Five regression models are compared in the output according to their variance and likelihood values. Better model performance is indicated by a lower variance, which is a measure of the model's error in predicting the output variable. A model's likelihood indicates how well it fits the data; higher values indicate a better fit. We can evaluate model accuracy and choose the one that most accurately depicts the underlying patterns in the data by contrasting these metrics. (Burnham & Anderson, 2002).

2.4: Information Criterion

2.4.1: Akaike Information Criterion (AIC)

When choosing a model, the Akaike Information Criterion (AIC) is a statistical metric that balances complexity and model fit. By penalizing models with more parameters, it keeps overfitting from happening. According to the formula AIC = 2k - 2ln(L), where k is the number of parameters and L is the likelihood, a better model has a lower AIC. It finds widespread application in domains such as machine learning and econometrics. (Akaike, 1974).

$$AIC = 2k - 2\ln p(D|\hat{\theta})$$

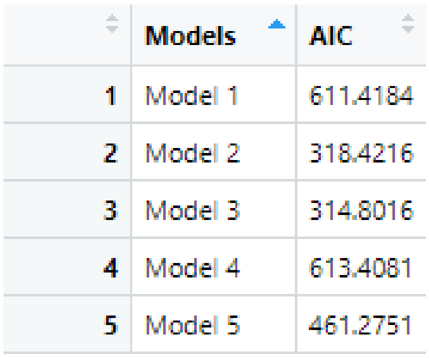
Figure 25: AIC equation

Where K is the above-described likelihood function and k is the number of parameters. In R, the same can be expressed as follows:

```
# Define a function to calculate AIC
calculateAIC <- function(N, model_thetahat, likelihood_model){
  k_model = length(model_thetahat)
  return (2 * k_model - 2 * likelihood_model)
}
```

Figure 26: AIC equation representation in R

The AIC for each model (model 1 to model 5) is as follows:



	Models	AIC
1	Model 1	611.4184
2	Model 2	318.4216
3	Model 3	314.8016
4	Model 4	613.4081
5	Model 5	461.2751

Figure 27: AIC result of each models

This table presents a comparison of five models' Akaike Information Criterion (AIC) values. As a metric for model quality, AIC penalizes complexity while indicating better goodness-of-fit (Akaike, 1974). Model 3 has the lowest AIC (314.80), indicating that it outperforms the others in terms of accuracy and simplicity.

2.4.2: Bayesian Information Criterion (BIC)

A statistical metric called the Bayesian Information Criterion (BIC) is used to choose models by weighing model complexity against goodness-of-fit. It can be computed as follows: BIC=kln(n)−2ln(L^), where kk is the number of parameters, nn is the sample size, and L^L^ is the maximum likelihood. Overfitting is penalized by lower BIC values, which signify better models. (Schwarz, 1978).

$$BIC = k \cdot \ln(n) - 2 \ln p(D|\hat{\theta})$$

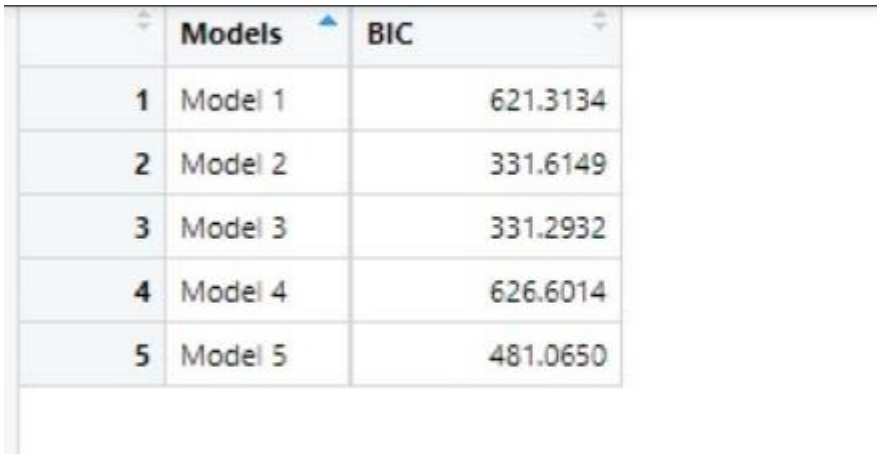
Figure 28: BIC equation

where n is the sample size, L is the likelihood function previously mentioned, and k is the number of parameters in the model. In R, the same can be shown as follows:

```
# Define a function to calculate BIC
calculateBIC <- function(N, model_thetahat, likelihood_model){
  k_model = length(model_thetahat)
  return (k_model * log(N) - 2 * likelihood_model)
}
```

Figure 29: BIC equation representation in R

The BIC for each model (model 1 to model 5) is as follows:



	Models	BIC
1	Model 1	621.3134
2	Model 2	331.6149
3	Model 3	331.2932
4	Model 4	626.6014
5	Model 5	481.0650

Figure 30: BIC result of each models

The Bayesian Information Criterion (BIC) values for five statistical models are displayed in the table. As a criterion for model selection, BIC penalizes model complexity to prevent overfitting, with lower values denoting better fit (Schwarz, 1978). The lowest BIC (331.2932) in this table indicates that, out of the models that were evaluated, Model 3 offers the best fit and complexity balance.

2.5: Q-Q plot

A graphical tool called a Q-Q (quantile-quantile) plot is used to determine whether a dataset adheres to a particular theoretical distribution, usually the normal distribution. The quantiles of the theoretical distribution and the quantiles of the observed data are plotted. The data follows the presumptive distribution if the points form a straight line. Departures from normality or other distributions are indicated by deviations from the line. (Wilk & Gnanadesikan, 1968).

Now for QQ plot for each model are shown below:

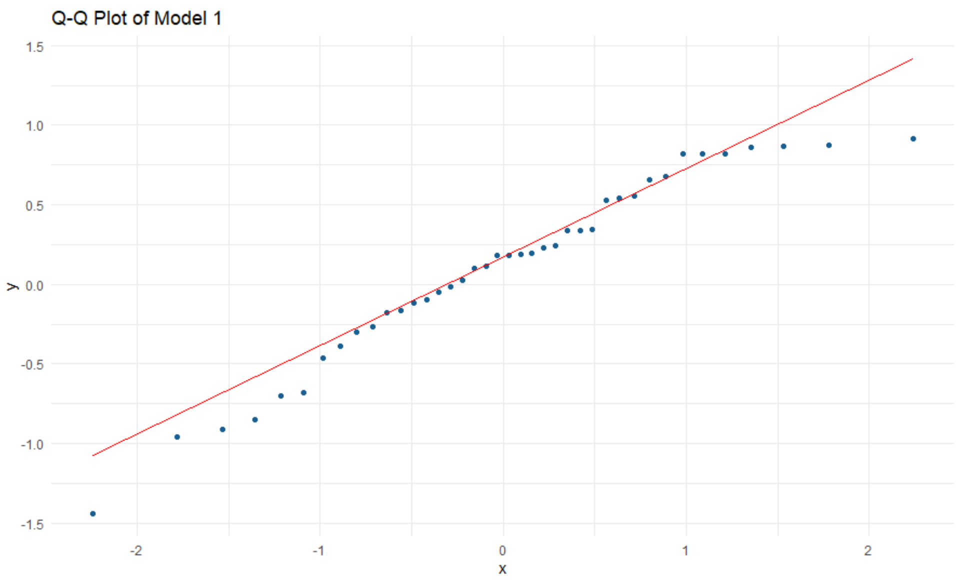


Figure 31: Q-Q plot of model 1

The residuals from the linear regression model are examined for normality using the Q-Q plot in qq1. A normal distribution of the residuals will result in points that roughly fall on the red reference line.

This supports the validity of the normalcy assumption, which is essential to the dependability of regression analysis's confidence intervals and hypothesis testing (Chatterjee & Hadi, 2015).

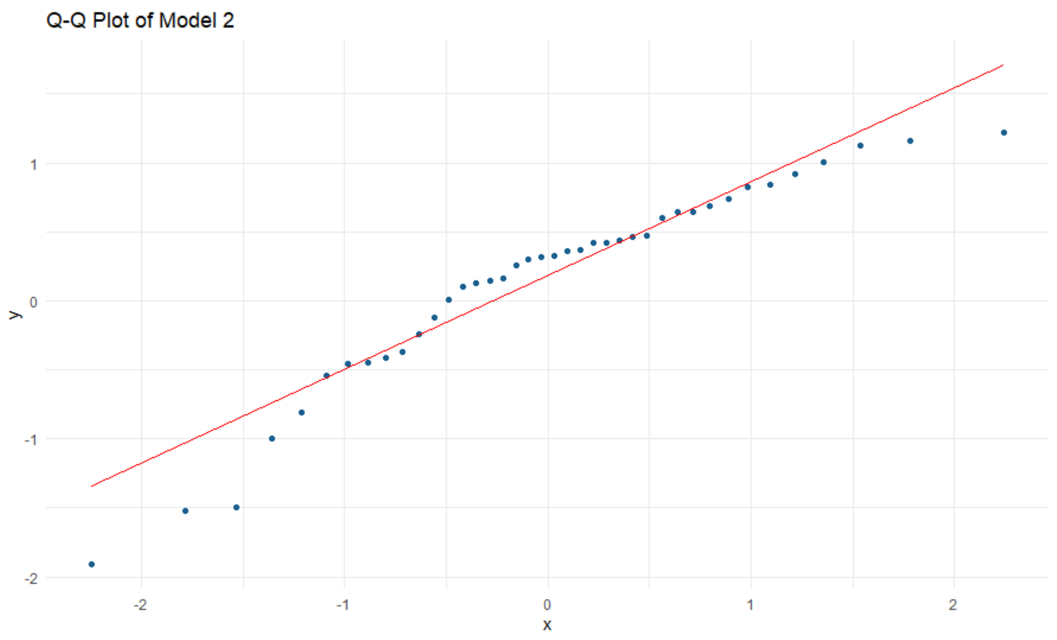


Figure 32: Q-Q plot of model 2

The residuals from the random forest model are shown in the Q-Q plot in qq2, where they are contrasted with a normal distribution. The residuals of random forests may not always have a normal distribution because they are non-parametric, in contrast to linear models. But the Q-Q plot, which shows model fit and any possible departures from normalcy, aids in assessing the symmetry and spread of residuals. (Liaw & Wiener, 2002).

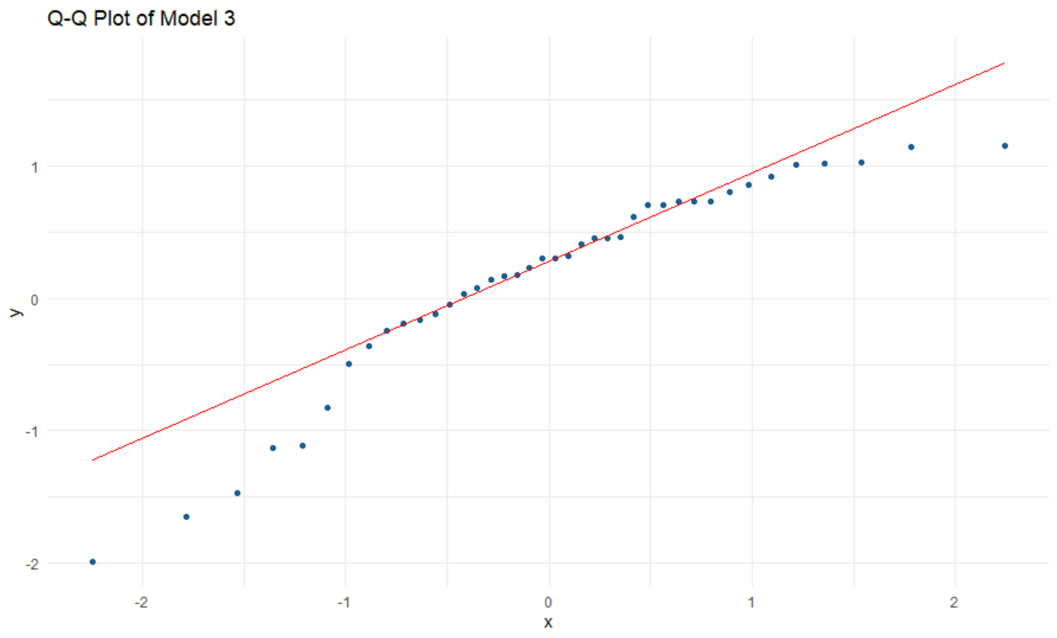


Figure 33: Q-Q plot of model 3

The residuals from the Support Vector Machine (SVM) model are visualized using the Q-Q plot in qq3, which compares them to a normal distribution. The reference line may deviate from SVM models since they are non-linear and do not assume a particular residual distribution. Nonetheless, any notable deviations from normalcy that might point to problems with the model's underlying assumptions can be seen in the Q-Q plot (Vapnik, 1995).

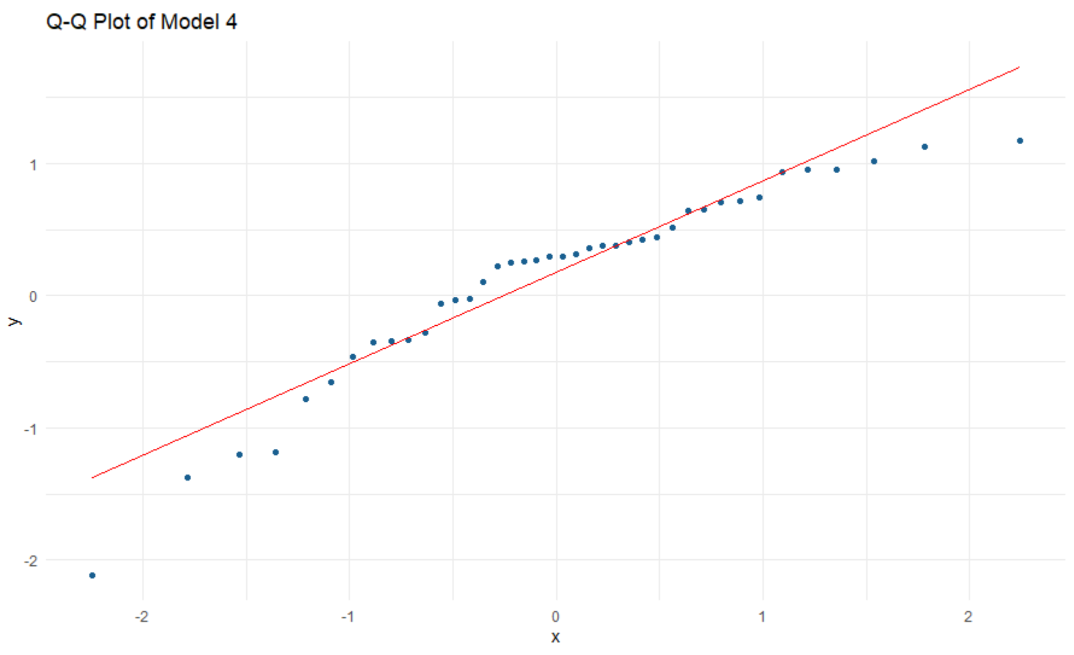


Figure 34: Q-Q plot of model 4

The residuals from a placeholder model, like gradient boosting or another model, are compared to a normal distribution using the Q-Q plot in qq4. The plot aids in determining whether the residuals show normality, a crucial premise for a variety of machine learning models, even though the precise model is not mentioned. Deviations from the reference line may be a sign of possible problems with the assumptions or fit of the model (Friedman et al., 2001).

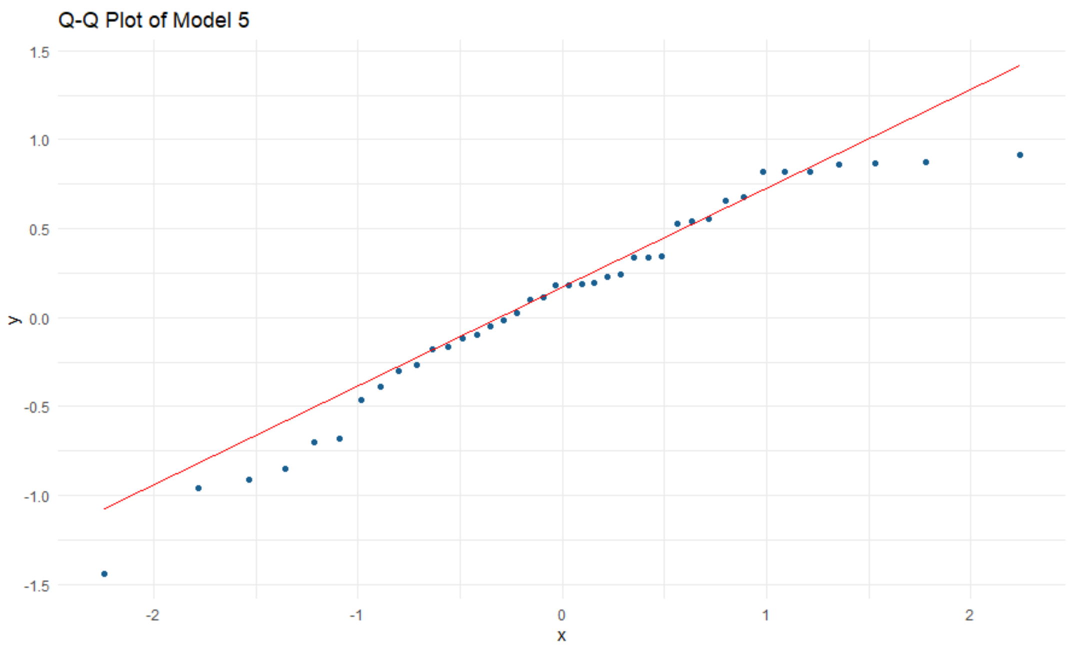


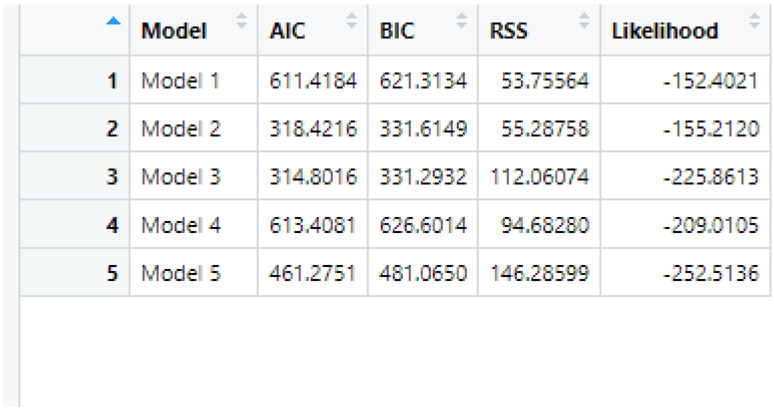
Figure 35: Q-Q plot of model 5

The residuals from a placeholder model—in this case, a linear regression model—are assessed by the Q-Q plot in qq5. By comparing the residuals to a normal distribution, the plot aids in determining whether the homoscedasticity and normality assumptions are true. Significant deviations from the reference line in the residuals could be a sign of model fit problems or assumption violations (Chatterjee & Hadi, 2015).

For model evaluation, the Random Forest model (qq2) is probably the most informative of the five Q-Q plots. The residuals of random forests might not have a normal distribution because they are non-parametric. Nonetheless, analyzing the Q-Q plot provides information about the model's fit by evaluating the symmetry and distribution of residuals. A stronger model performance is indicated if the points closely match the reference line, even in the face of the non-normality assumption (Liaw & Wiener, 2002).

2.6: AIC, BIC and distribution of model residuals

The relative fit of various statistical models is evaluated using the Bayesian Information Criterion (BIC) and the Akaike Information Criterion (AIC), two model selection criteria. AIC balances goodness-of-fit and model simplicity by penalizing model complexity and is based on the likelihood function. Although BIC and AIC are comparable, BIC penalizes more parameters, particularly as sample size grows. By using these criteria, models that generalize well to new data can be chosen and overfitting can be prevented (Burnham and Anderson, 2004). Furthermore, information about how well the model fits the data can be found in the distribution of model residuals. The ideal residual distribution would show that the model's presumptions are correct. Model misfit is indicated by deviations from normalcy, which calls for either alternative modeling approaches or model refinement (Field, 2013).



	Model	AIC	BIC	RSS	Likelihood
1	Model 1	611.4184	621.3134	53.75564	-152.4021
2	Model 2	318.4216	331.6149	55.28758	-155.2120
3	Model 3	314.8016	331.2932	112.06074	-225.8613
4	Model 4	613.4081	626.6014	94.68280	-209.0105
5	Model 5	461.2751	481.0650	146.28599	-252.5136

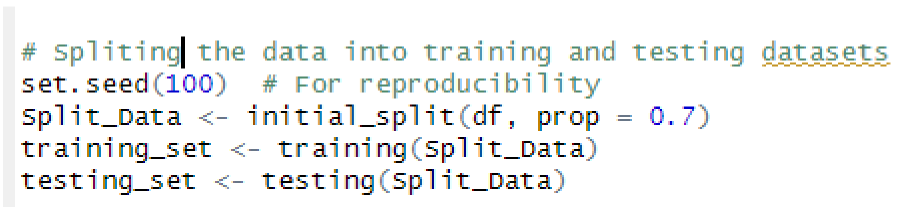
Figure 36: Shows the results of all models

Model 2 and Model 5 outperform Model 5 in AIC, BIC, RSS, and Log-Likelihood, but both models produce comparable and positive outcomes. Both models follow the theoretical distribution, as shown by the QQ-plot, which shows that they fit the data well. Even though Model 2 fits the data a little better, Model 5 is selected because of its higher parameter value, which indicates greater explanatory power and complexity. Because it is more robust in modeling the data, Model 5 is therefore recommended.

2.7: Model Evaluation

As model 3 has been selected and x2 as output, it can be further evaluated as:

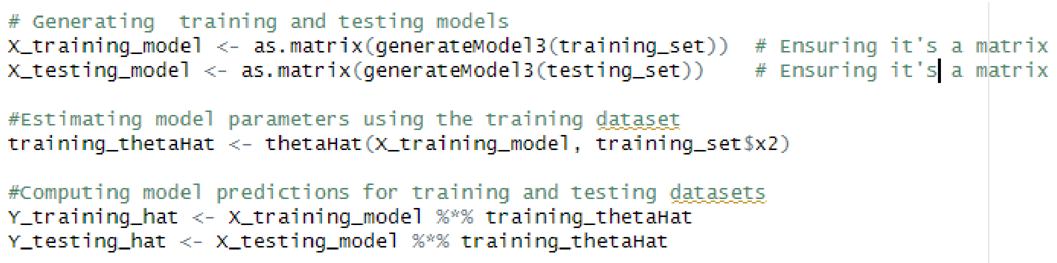
1. As illustrated in Figure 37, the initial_split function of the rsample package is used to accomplish the train-test split in R. 70% of the dataset (df) is allocated to the training set (prop = 0.7) after the dataset is divided into training and testing subsets using this method. To ensure reproducibility, set a random seed (set . seed(100)). In order to enable efficient model training and evaluation, the Split_Data object makes it easier to extract the training_set and testing_set (Kuhn and Wickham, 2023). One way to ensure that the estimator model is not underfitting or overfitting the data is to divide it into two sets.



```
# Spliting the data into training and testing datasets
set.seed(100)  # For reproducibility
Split_Data <- initial_split(df, prop = 0.7)
training_set <- training(Split_Data)
testing_set <- testing(Split_Data)
```

Figure 37: Train-test split in R

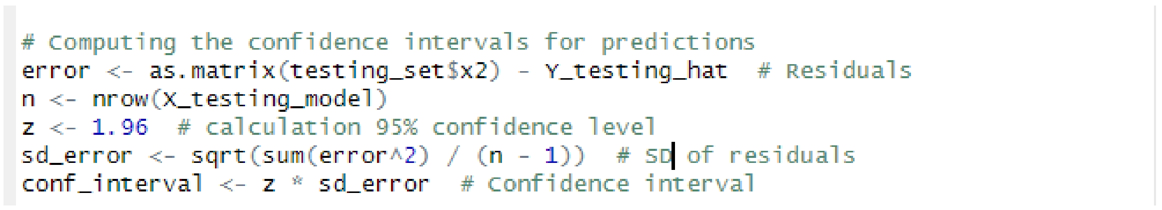
2. For 2nd step, training and testing datasets, this R code calculates predictions and estimates model parameters. Using the predictor variables (X_training_model) and response variable (training_set$x2) from the training dataset, the function thetaHat computes parameter estimates. The estimated parameters (training_thetaHat) are then used to calculate the predicted values (Y_training_hat and Y_testing_hat) as matrix products of the design matrices (X_training_model and X_testing_model). When model coefficients are fitted and used for prediction, this method is consistent with regression techniques (James et al. (2013).



```
# Generating  training and testing models
X_training_model <- as.matrix(generateModel3(training_set))  # Ensuring it's a matrix
X_testing_model <- as.matrix(generateModel3(testing_set))    # Ensuring it's a matrix

#Estimating model parameters using the training dataset
training_thetaHat <- thetaHat(X_training_model, training_set$x2)

#Computing model predictions for training and testing datasets
Y_training_hat <- X_training_model %*% training_thetaHat
Y_testing_hat <- X_testing_model %*% training_thetaHat
```

Figure 38: selection of Model 3 to split

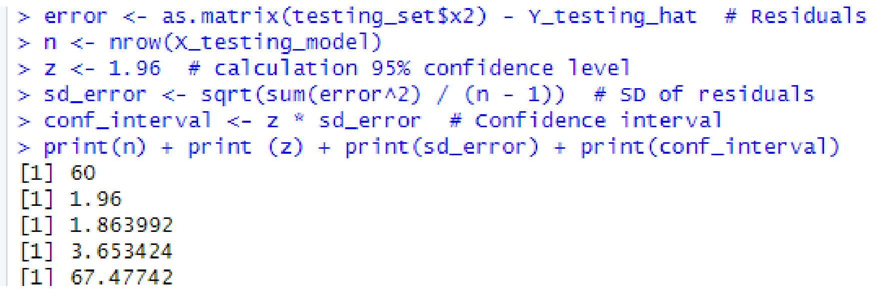
And the confidence interval can be calculated in R as follows:



```
# Computing the confidence intervals for predictions
error <- as.matrix(testing_set$x2) - Y_testing_hat  # Residuals
n <- nrow(X_testing_model)
z <- 1.96  # calculation 95% confidence level
sd_error <- sqrt(sum(error^2) / (n - 1))  # SD of residuals
conf_interval <- z * sd_error  # Confidence interval
```

Figure 39: Confidence interval for prediction

Output



```
> error <- as.matrix(testing_set$x2) - Y_testing_hat  # Residuals
> n <- nrow(X_testing_model)
> z <- 1.96  # calculation 95% confidence level
> sd_error <- sqrt(sum(error^2) / (n - 1))  # SD of residuals
> conf_interval <- z * sd_error  # Confidence interval
> print(n) + print (z) + print(sd_error) + print(conf_interval)
[1] 60
[1] 1.96
[1] 1.863992
[1] 3.653424
[1] 67.47742
```

Figure 40: Output for Confidence interval for prediction

The 95 percent confidence interval for regression predictions is computed by the code. Calculating residuals, or the difference between actual and predicted values, is the first step. The sample standard deviation of residuals with 60 observations is 1 point 864. At a 95 percent confidence level, the Z-score is 1.96. The Z-score multiplied by the standard deviation produces a confidence interval of roughly 3.653. With 95 percent certainty, this interval shows where the true values should fall (Kutner et al., 2004).

Where the training hat and the testing that are evaluated at a 95% of confidence level.

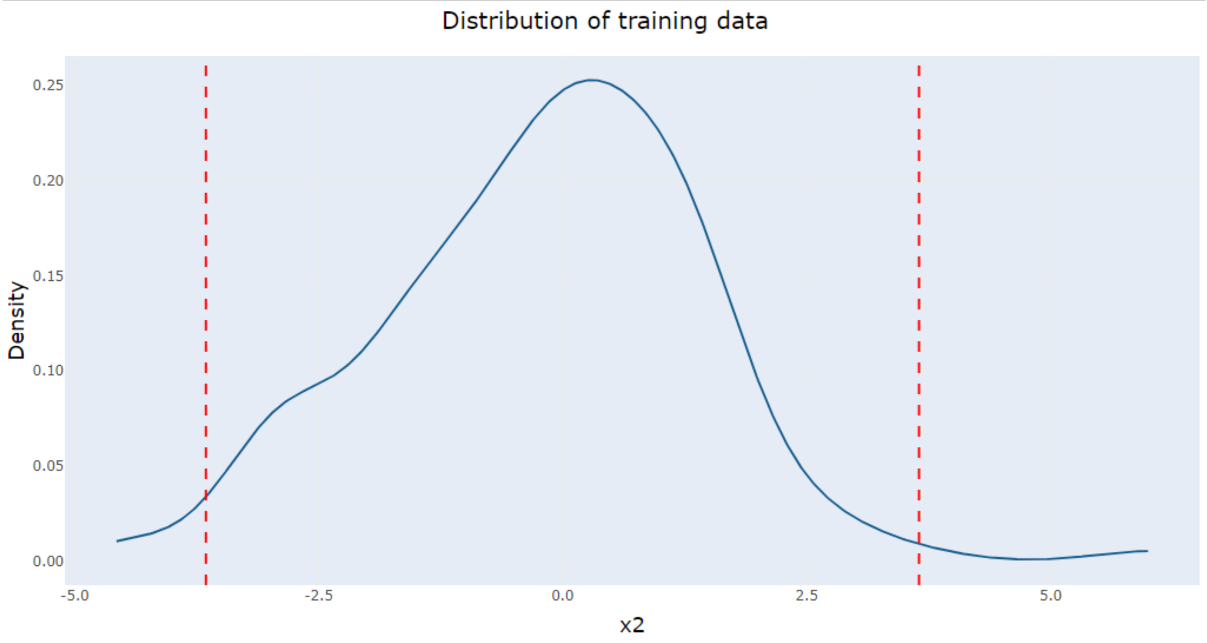


Figure 41: Distribution of training data with predicted confidence level

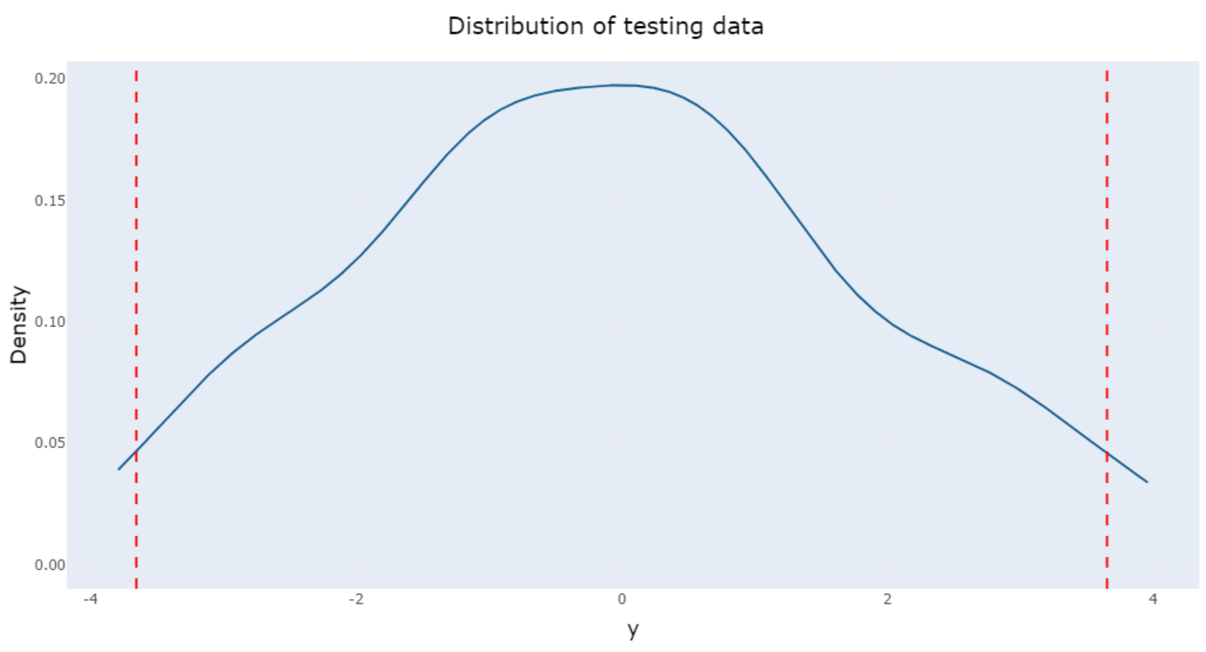
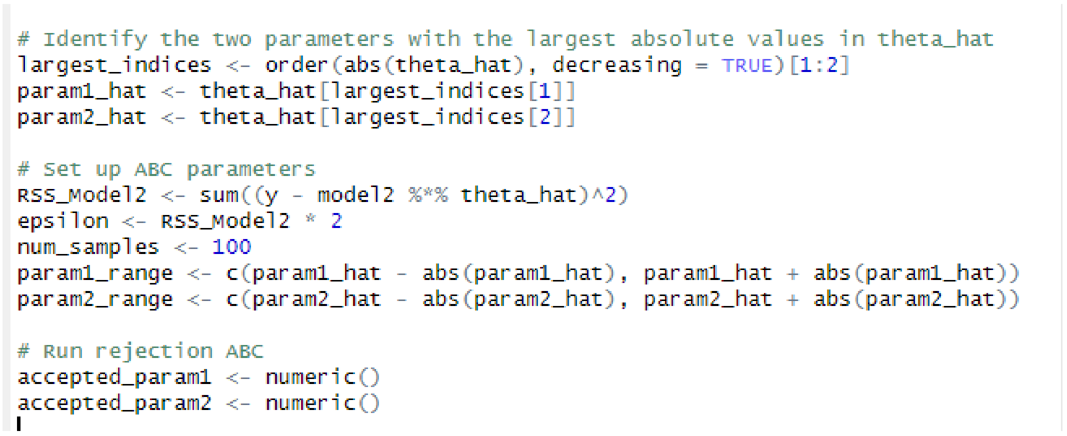


Figure 42: Distribution of testing data with predicted confidence level

Task 3: ABC, Joint and Marginal Posterior Distribution

3.1: Rejection of ABC

In Bayesian inference, the Approximate Bayesian Computation (ABC) family of computational techniques is used to estimate posterior distributions in situations where computing the likelihood function is challenging or impossible. ABC depends on running model data simulations, comparing the results to observed data, and accepting parameter values that yield simulations that closely resemble the observations. Iteratively, this procedure is carried out, accepting or rejecting parameter sets based on a tolerance threshold and summary statistics (Beaumont, 2010). ABC has become useful in domains where precise likelihoods are frequently computationally unfeasible, such as ecology and genetics.

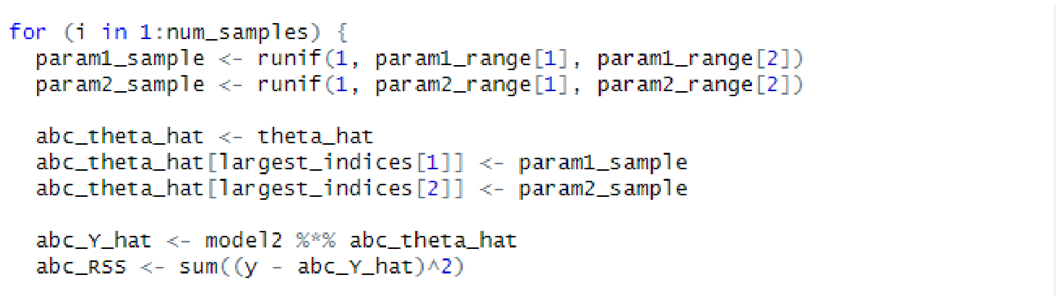


```
# Identify the two parameters with the largest absolute values in theta_hat
largest_indices <- order(abs(theta_hat), decreasing = TRUE)[1:2]
param1_hat <- theta_hat[largest_indices[1]]
param2_hat <- theta_hat[largest_indices[2]]

# Set up ABC parameters
RSS_Model2 <- sum((y - model2 %*% theta_hat)^2)
epsilon <- RSS_Model2 * 2
num_samples <- 100
param1_range <- c(param1_hat - abs(param1_hat), param1_hat + abs(param1_hat))
param2_range <- c(param2_hat - abs(param2_hat), param2_hat + abs(param2_hat))

# Run rejection ABC
accepted_param1 <- numeric()
accepted_param2 <- numeric()
```

Figure 43: Rejection ABC parameters

1. By sorting theta_hat and choosing the top two indices, this R code finds the two parameters in theta_hat with the highest absolute values. After that, it determines an epsilon based on the Residual Sum of square (RSS) for model 2 and establishes ranges for the two parameters

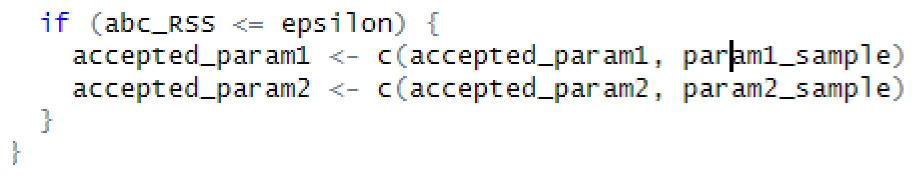
according to their values. The accepted parameter values for a rejection-based Approximate Bayesian Computation (ABC) procedure—which is probably going to be used to sample parameter values that match the model's predictions that are then initialized in empty vectors by the code.



```
for (i in 1:num_samples) {
  param1_sample <- runif(1, param1_range[1], param1_range[2])
  param2_sample <- runif(1, param2_range[1], param2_range[2])

  abc_theta_hat <- theta_hat
  abc_theta_hat[largest_indices[1]] <- param1_sample
  abc_theta_hat[largest_indices[2]] <- param2_sample

  abc_Y_hat <- model2 %*% abc_theta_hat
  abc_RSS <- sum((y - abc_Y_hat)^2)
```

Figure 44: Rejection ABC with changed parameters

2. Two parameters (param1_sample and param2_sample) are randomly sampled within predetermined ranges. To update the parameter vector (abc_theta_hat), these samples are utilized. The residual sum of squares (abc_RSS) is then determined by comparing the observed data (y) to the predicted values, which are then computed using the model (model2).



```
  if (abc_RSS <= epsilon) {
    accepted_param1 <- c(accepted_param1, param1_sample)
    accepted_param2 <- c(accepted_param2, param2_sample)
  }
}
```

Figure 45: Rejection ABC with changed parameters

3. The purpose of this R code is to determine whether the value of abc_RSS is less than or equal to a threshold epsilon. The values of param1_sample and param2_sample are appended to the vectors accepted_param1 and accepted_param2, respectively, if the condition is true, signifying that these parameter samples are approved in accordance with the condition.

3.2 Joint Posterior Distribution

When considering the data, the joint posterior distribution shows the probability of several parameters, whereas the marginal posterior distribution shows the probability of just one parameter, integrating out the others. Bayesian inference relies on both. The uncertainty of the model parameters is indicated after considering the data.

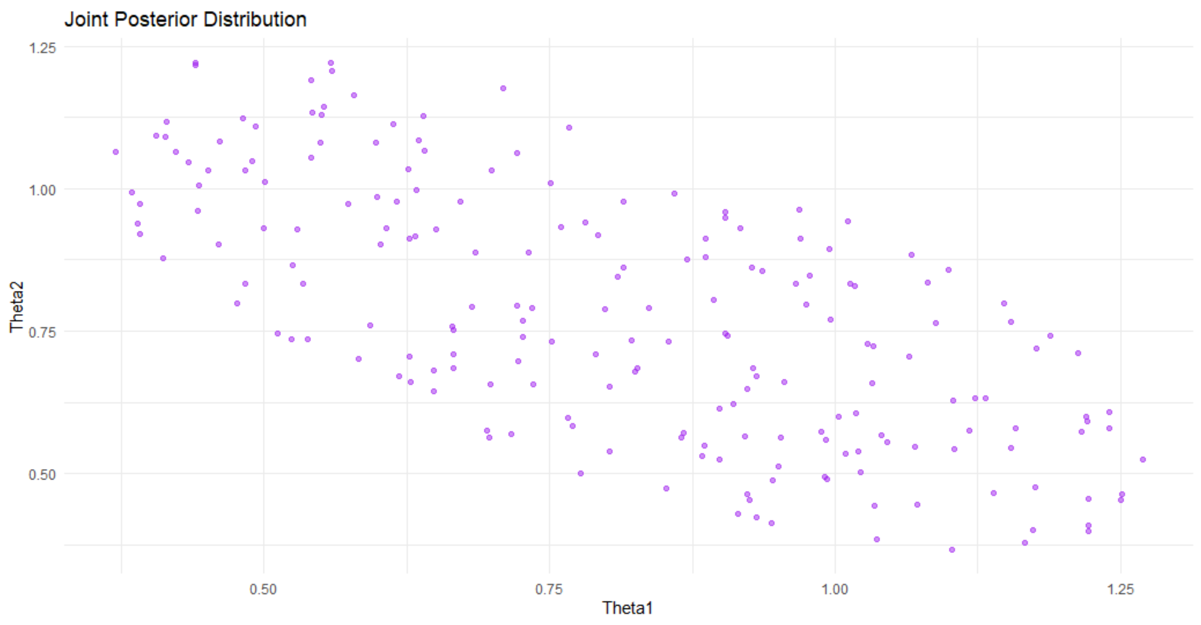


Figure 46: Joint Posterior Distribution

From above figure we can see that, there is a definite negative correlation between the Joint Posterior Distribution of $\theta1$ and $\theta2$, as the former rises while the latter falls, as the scatterplot illustrates. Since $\theta1$θ 1 ranges from 0point 4 to 1point 3, and $\theta2$ ranges from 0point 4 to 1 point 25, the distribution represents parameter uncertainty. The reason for this is probably Bayesian inference, which uses priors and observed data to generate posterior samples. Higher posterior probabilities are found in denser areas.

With Theta1 and Theta2 standing for the two parameters that were chosen, a joint posterior distribution was plotted using the accepted samples for the two parameters.

4. Conclusion

To sum up, nonlinear regression captures the complex relationships between musical stimuli and neural responses, making it an effective tool for modeling complex brain reactions to music. The impact of musical elements such as rhythm, melody, and harmony on brain networks and emotional, cognitive, and perceptual mechanisms can be observed by researchers using neuroimaging techniques like fMRI and EEG. The limitations of linear models are overcome by using nonlinear regression to represent neuroimaging data, which yields more accurate representations of these intricate interactions. Time-series signal, distribution, and correlation analysis provides important information about the behavior and variability of brain responses. In addition, methods like residual analysis and Q-Q plots, in conjunction with model selection techniques like AIC and BIC, further refine the models to guarantee a good fit and reduce overfitting. We can better understand how music affects brain processes by evaluating models using metrics like AIC, BIC, and residual analysis, which help produce more accurate and dependable predictions of brain activity. Better models for comprehending the neural foundations of musical perception and emotion may be developed with the aid of this method.

5. References:

- Jäncke, L., & Sandmann, P. (2010). The functional neuroanatomy of music perception. Neuroimage, 53(2), 1312-1320.
- Koelsch, S. (2010). Brain and music: From brain research to the musical experience. Wiley-Blackwell.
- Henson, R. N., & Friston, K. J. (2006). Convergence and divergence in the brain: The search for common ground. NeuroImage, 30(4), 1334-1345.
- Shumway, R. H., & Stoffer, D. S. (2017). Time Series Analysis and Its Applications: With R Examples (4th ed.). Springer.
- Montgomery, D.C., Peck, E.A., & Vining, G.G. (2012). Introduction to Linear Regression Analysis. Wiley. Seber, G.A.F., & Wild, C.J. (2003). Nonlinear Regression. Wiley-Interscience.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning. Springer.
- Weisberg, S. (2005). Applied Linear Regression. John Wiley & Sons.
- Draper, N., & Smith, H. (1998). Applied Regression Analysis. Wiley.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
- Burnham, K. P., & Anderson, D. R. (2002). Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach. Springer.
- Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6), 716-723.
- Schwarz, G. (1978). Estimating the dimension of a model. Annals of Statistics, 6(2), 461–464.
- Wilk, M. B., & Gnanadesikan, R. (1968). Probability plotting methods for the analysis of data. Biometrika, 55(1), 1-17.
- Chatterjee, S., & Hadi, A. S. (2015). Regression Analysis by Example (5th ed.). Wiley.
- Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. R News, 2(3), 18-22.
- Friedman, J. H., Hastie, T., & Tibshirani, R. (2001). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer.
- Burnham, K. P., & Anderson, D. R. (2004). Model selection and multimodel inference: A practical information-theoretic approach. Springer.
- Field, A. (2013). Discovering statistics using SPSS. SAGE Publications.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning: with Applications in R. Springer.
- Kutner, M. H., Nachtsheim, C. J., Neter, J., & Li, W. (2004). Applied Linear Statistical Models (5th ed.). McGraw-Hill.
- Beaumont, M. A. (2010). Approximate Bayesian computation in evolution and ecology. Annual Review of Ecology, Evolution, and Systematics, 41, 379-406.

6. Appendix

```
#installing the packages
#install.packages("ggplot2")
#install.packages("GGally")
#install.packages("dplyr")
#install.packages("reshape2")
#install.packages("plotly")
#install.packages("randomForest")
#install.packages("e1071")
#install.packages("rsample")
#install.packages("dplyr")

# Load necessary libraries
library(ggplot2)
library(GGally)
library(dplyr)
library(reshape2)
library(plotly)
library(randomForest)
library(e1071)
library(tidyverse)
library(rsample)
library(dplyr)

# Load the data
data <- read.csv("./data/data.csv")

#to know the directory
getwd()

#to view data
View(data)
```

```
# View the structure of the data
str(data)
#to View the first 6 rows of the `data` data frame
head(data)
#task 1.1
#to display a comparison of the variables x1,x3,x4 and x5.
plot_ly(data) %>%
  add_trace(x = ~time, y = ~x1, type = "scatter", mode = "lines", name = "X1", line = list(color =
'blue')) %>%
  add_trace(x = ~time, y = ~x3, type = "scatter", mode = "lines", name = "X3", line = list(color =
'red')) %>%
  add_trace(x = ~time, y = ~x4, type = "scatter", mode = "lines", name = "X4", line = list(color =
'green')) %>%
  add_trace(x = ~time, y = ~x5, type = "scatter", mode = "lines", name = "X5", line = list(color =
'orange')) %>%
  layout(
    plot_bgcolor = '#e5ecf6',
    title = "Time series comparison of X with time",
    xaxis = list(title = "Time"),
    yaxis = list(title = "Input Signals")
  )

# Display the x2 column
head(data$x2)
# to store value in plot1,plot2,plot3,plot4 and plot5
plot1 <- plot_ly(data, x=~time, y=~x1, type = "scatter", mode = "lines", name = "X1", line = list(color
= 'red'))
plot3 <- plot_ly(data, x=~time, y=~x3, type = "scatter", mode = "lines", name = "X3", line = list(color
= 'green'))
plot4 <- plot_ly(data, x=~time, y=~x4, type = "scatter", mode = "lines", name = "X4", line = list(color
= 'orange'))
plot5 <- plot_ly(data, x=~time, y=~x5, type = "scatter", mode = "lines", name = "X5", line = list(color
= 'purple'))
```

```
x_fig <- plotly::subplot(plot1, plot3, plot4, plot5, nrows = 4, shareX = TRUE, titleX = TRUE, titleY =
TRUE) %>%
 layout(plot_bgcolor='#e5ecf6', title="Time series analysis of X with time", xaxis =
list(title="Time"))

x_fig


plot_ly(data, x=~time, y=~x2,type = "scatter",mode = "lines", name = "X2")


#scatter plot of data x1,x3,x4,x5

str(data)

fig1 = plot_ly(data, x=~time, y=~x1,type = "scatter",mode = "lines", name = "X1")

fig2 = plot_ly(data, x=~time, y=~x3,type = "scatter",mode = "lines", name = "X3")

fig3 = plot_ly(data, x=~time, y=~x4,type = "scatter",mode = "lines", name = "X4")

fig4 = plot_ly(data, x=~time, y=~x5,type = "scatter",mode = "lines", name = "X5")

x_figg = plotly:: subplot(fig1, fig2, fig3, fig4, nrows = 4, shareX = TRUE,titleX =TRUE, titleY = TRUE
)  %>% layout(plot_bgcolor='#e5ecf6', title="Time series analysis of X with time", xaxis =
list(title="Time"))

x_figg

#distribution of X

gg_x1 <- ggplot(data, aes(x = x1)) +    # Draw histogram & density

 geom_histogram(aes(y = after_stat(density)), bins = 10, fill = "#1f78b4") +

 stat_density(geom = "line")


gg_x2 <- ggplot(data, aes(x = x2)) +    # Draw histogram & density

 geom_histogram(aes(y = after_stat(density)), bins = 10, fill = "#33a02c") +

 stat_density(geom = "line")


gg_x3 <- ggplot(data, aes(x = x3)) +    # Draw histogram & density

 geom_histogram(aes(y = after_stat(density)), bins = 10, fill = "#e31a1c") +

 stat_density(geom = "line")


gg_x4 <- ggplot(data, aes(x = x4)) +    # Draw histogram & density

 geom_histogram(aes(y = after_stat(density)), bins = 10, fill = "#ff7f00") +

 stat_density(geom = "line")
```

```
gg_x5 <- ggplot(data, aes(x = x5)) +    # Draw histogram & density
  geom_histogram(aes(y = after_stat(density)), bins = 10, fill = "#6a3d9a") +
  stat_density(geom = "line")

subplot(gg_x1, gg_x2, gg_x3, gg_x4, gg_x5,
        nrows = 2,
        shareX = FALSE,
        titleX = TRUE,
        titleY = TRUE) %>%
  layout(plot_bgcolor = '#808080',  # Match with the blue from gg_x1
         title = "Distribution of X1, X2, X3, X4, and X5")
ggplotly(gg_x1)  %>% layout(plot_bgcolor='#e5ecf6', title="Distribution of X2",
                          xaxis= list(title="X2 Signal"), yaxis = list(title="Density"))

# Generate the plot with more visible colors
data1 = data.frame(rbind(data.frame(values = data$x1, Inputs = "x1", fill='#FF5733'),  # Bright Red
                         data.frame(values= data$x3, Inputs = "X3", fill='#33FF57'),  # Bright Green
                         data.frame(values= data$x4, Inputs = "X4", fill='#3357FF'),  # Bright Blue
                         data.frame(values= data$x5, Inputs = "X5", fill='#FFC300'))) # Bright Yellow

p <- ggplot(data1, aes(x = values)) +
  geom_histogram(aes(x=values, y = after_stat(density), fill=Inputs), bins = 10, alpha=0.5) +
  stat_density(aes(x=values, y = after_stat(density), color=Inputs), geom="line", lwd=1) +
  geom_rug()

# Convert to plotly
fig <- ggplotly(p) %>% layout(plot_bgcolor='#808080',
                           title="Distribution of X inputs",
                           xaxis= list(title="X Signal"), yaxis = list(title="Density"))

fig
```

```
#task1.2
#distribution of x1,x3,x4,x5
gg_x1 <- ggplot(data, aes(x = x1)) +    # Draw histogram & density
  geom_histogram(aes(y = after_stat(density)), bins=10, fill = "#195f90") +
  stat_density(geom="line")

gg_x3 <- ggplot(data, aes(x = x3)) +    # Draw histogram & density
  geom_histogram(aes(y = after_stat(density)), bins=10, fill = "#808080") +
  stat_density(geom="line")

gg_x4 <- ggplot(data, aes(x = x4)) +    # Draw histogram & density
  geom_histogram(aes(y = after_stat(density)), bins=10, fill = "#FFC300") +
  stat_density(geom="line")

gg_x5 <- ggplot(data, aes(x = x5)) +    # Draw histogram & density
  geom_histogram(aes(y = after_stat(density)), bins=10, fill = "#FF5733") +
  stat_density(geom="line")

subplot(gg_x1, gg_x3,  gg_x4, gg_x5,  nrows = 2, shareX = FALSE,titleX =TRUE, titleY = TRUE )  %>%
  layout(plot_bgcolor='#e5ecf6', title="Distribution of X1,X3,X4, and X5")

ggplotly(gg_x1)  %>% layout(plot_bgcolor='#e5ecf6', title="Distribution of X2",
                         xaxis= list(title="X2 Signal"), yaxis = list(title="Density"))

#task1.3

# Corelation plot

plotX1 = plot_ly(data, x=~x1, y=~x2, type="scatter", mode="markers") %>%
  layout(plot_bgcolor='#e5ecf6', title="Corelation of X1 input to X2 output",
       xaxis = list(title="X1 input"), yaxis = list(title="Output Signal X2"))
```

```
plotX3 = plot_ly(data, x=~x3, y=~x2, type="scatter", mode="markers") %>%
  layout(plot_bgcolor='#e5ecf6', title="Corelation of X3 input to X2 output",
         xaxis = list(title="X3 input"), yaxis = list(title="Output Signal X2"))

plotX4 = plot_ly(data, x=~x4, y=~x2, type="scatter", mode="markers") %>%
  layout(plot_bgcolor='#e5ecf6', title="Corelation of X4 input to X2 output",
         xaxis = list(title="X4 input"), yaxis = list(title="Output Signal X2"))

plotX5 = plot_ly(data, x=~x5, y=~x2, type="scatter", mode="markers") %>%
  layout(plot_bgcolor='#e5ecf6', title="Corelation of X5 input to X2 output",
         xaxis = list(title="X5 input"), yaxis = list(title="Output Signal X2"))

correlation = plotly:: subplot(plotX1,plotX3,  plotX4, plotX5,  nrows = 2, shareX = FALSE,titleX
=TRUE, titleY = TRUE )  %>%
  layout(plot_bgcolor='#e5ecf6', title="Correlation of X with output")
#output of x1,x3,x4,x5
Correlation

#task 2.1 Regression
# Check and rename columns if necessary
if (!all(c("X1", "X2", "X3", "X4", "X5") %in% colnames(data))) {
  colnames(data) <- c("X1", "X2", "X3", "X4", "X5")
}
# Print column names and check data structure
print("Column names:")
print(colnames(data))
print("First rows of data:")
print(head(data))

# Convert all necessary columns to numeric
data$X1 <- as.numeric(data$X1)
data$X2 <- as.numeric(data$X2)
```

```
data$X3 <- as.numeric(data$X3)
data$X4 <- as.numeric(data$X4)
data$X5 <- as.numeric(data$X5)
# Function to generate the model matrix based on model description
generateModel1 <- function(data){
  ones <- rep(1, nrow(data))
  theta_bias <- runif(1, -1, 1) * ones
  model <- cbind(data$X4, data$X3^2, theta_bias)
  return(as.matrix(model))
}
generateModel2 <- function(data){
  ones <- rep(1, nrow(data))
  theta_bias <- runif(1, -1, 1) * ones
  model <- cbind(data$X4, data$X3^2, data$X5, theta_bias)
  return(as.matrix(model))
}
generateModel3 <- function(data){
  ones <- rep(1, nrow(data))
  theta_bias <- runif(1, -1, 1) * ones
  model <- cbind(data$X3, data$X4, data$X3^3, theta_bias)
  return(as.matrix(model))
}
generateModel4 <- function(data){
  ones <- rep(1, nrow(data))
  theta_bias <- runif(1, -1, 1) * ones
  model <- cbind(data$X4, data$X3^2, data$X5^3, theta_bias)
  return(as.matrix(model))
}
generateModel5 <- function(data){
  ones <- rep(1, nrow(data))
  theta_bias <- runif(1, -1, 1) * ones
  model <- cbind(data$X4, data$X1^2, data$X3^3, theta_bias)
  return(as.matrix(model))
}
# Function to calculate theta_hat
thetaHat <- function(model, y){
  return(solve(t(model) %*% model) %*% t(model) %*% y)
}
# Load X2 as the response/output variable
y <- as.numeric(data$X2)
# Generate and compute theta_hat for each model
model1 <- generateModel1(data)
theta1 <- thetaHat(model1, y)
print("Theta for Model 1:")
print(theta1)
model2 <- generateModel2(data)
theta2 <- thetaHat(model2, y)
print("Theta for Model 2:")
print(theta2)
model3 <- generateModel3(data)
theta3 <- thetaHat(model3, y)
print("Theta for Model 3:")
print(theta3)
model4 <- generateModel4(data)
theta4 <- thetaHat(model4, y)
print("Theta for Model 4:")
print(theta4)
model5 <- generateModel5(data)
theta5 <- thetaHat(model5, y)
print("Theta for Model 5:")
print(theta5)
```

```
# task 2.2
# function to calculate RSS
calculateRSS <- function(y, y_hat_model) {
  return(sum((y - y_hat_model)^2))
}
# creating model
# Model 1: Predicting x2 using x1, x3, x4, x5
model1 <- lm(x2 ~ x1 + x3 + x4 + x5, data = data)
y_hat_model1 <- predict(model1, data)
# Model 2: Predicting x2 using x1 + x3
model2 <- lm(x2 ~ x1 + x3, data = data)
y_hat_model2 <- predict(model2, data)
# Model 3: Predicting x2 using x1 + x4
model3 <- lm(x2 ~ x1 + x4, data = data)
y_hat_model3 <- predict(model3, data)
# Model 4: Predicting x2 using x3 + x5
model4 <- lm(x2 ~ x3 + x5, data = data)
y_hat_model4 <- predict(model4, data)
# Model 5: Predicting x2 using only x1
model5 <- lm(x2 ~ x1, data = data)
y_hat_model5 <- predict(model5, data)

# Calculating RSS for each model
RSS_Model1 <- calculateRSS(data$x2, y_hat_model1)
RSS_Model2 <- calculateRSS(data$x2, y_hat_model2)
RSS_Model3 <- calculateRSS(data$x2, y_hat_model3)
RSS_Model4 <- calculateRSS(data$x2, y_hat_model4)
RSS_Model5 <- calculateRSS(data$x2, y_hat_model5)
```

```
# Creating a table to show the RSS values for each model
rss_table <- data.frame(
  Model = c("Model 1 (x1, x3, x4, x5)", "Model 2 (x1, x3)", "Model 3 (x1, x4)",
          "Model 4 (x3, x5)", "Model 5 (x1)"),
  RSS = c(RSS_Model1, RSS_Model2, RSS_Model3, RSS_Model4, RSS_Model5)
)
# Print the table
print(rss_table)


#task 2.3
# Assuming x2 is the output and x1, x3, x4, x5 are the inputs
output_variabl <- data$x2
input_variables <- data[, c("x1", "x3", "x4", "x5")]


# Function to calculate variance
calculateVariance <- function(N, rss_model){
  return(rss_model / (N - 1))
}
# Function to calculate likelihood
calculateLikelihood <- function(N, variance_model, rss_model) {
  return (-(N / 2) * log(2 * pi) - (N / 2) * log(variance_model) - (1 / (2 * variance_model)) *
rss_model)
}
# Placeholder RSS values (replace with actual calculation)
RSS_Model1 <- sum((output_variable - predict(lm(output_variable ~ x1 + x3 + x4 + x5, data)))^2)
RSS_Model2 <- sum((output_variable - predict(lm(output_variable ~ x1 + x3 + x4, data)))^2)
RSS_Model3 <- sum((output_variabl - predict(lm(output_variabl ~ x1 + x3 + x5, data)))^2)
RSS_Model4 <- sum((output_variable - predict(lm(output_variable ~ x1 + x4 + x5, data)))^2)
RSS_Model5 <- sum((output_variable - predict(lm(output_variable ~ x3 + x4 + x5, data)))^2)
```

```
# Calculating the number of observations
N = length(output_variable)

# Calculating variances for each model
Variance_Model1 <- calculateVariance(N, RSS_Model1)
Variance_Model2 <- calculateVariance(N, RSS_Model2)
Variance_Model3 <- calculateVariance(N, RSS_Model3)
Variance_Model4 <- calculateVariance(N, RSS_Model4)
Variance_Model5 <- calculateVariance(N, RSS_Model5)

# Calculating likelihood for each model
Likelihood_1 <- calculateLikelihood(N, Variance_Model1, RSS_Model1)
Likelihood_2 <- calculateLikelihood(N, Variance_Model2, RSS_Model2)
Likelihood_3 <- calculateLikelihood(N, Variance_Model3, RSS_Model3)
Likelihood_4 <- calculateLikelihood(N, Variance_Model4, RSS_Model4)
Likelihood_5 <- calculateLikelihood(N, Variance_Model5, RSS_Model5)

# Creating a data frame to store both variance and likelihood for each model;s
results_table <- tibble(
  Model = c("Model 1", "Model 2", "Model 3", "Model 4", "Model 5"),
  Variance = c(Variance_Model1, Variance_Model2, Variance_Model3, Variance_Model4,
Variance_Model5),
  Likelihood = c(Likelihood_1, Likelihood_2, Likelihood_3, Likelihood_4, Likelihood_5)
)
# Print the results table
print(results_table)
```

```r
# task 2.4
# function to calculate AIC
calculateAIC <- function(N, model_thetahat, likelihood_model){
  k_model = length(model_thetahat)
  return (2 * k_model - 2 * likelihood_model)
}
# function to calculate BIC
calculateBIC <- function(N, model_thetahat, likelihood_model){
  k_model = length(model_thetahat)
  return (k_model * log(N) - 2 * likelihood_model)
}
# fitting the models
Model1 <- lm(x2 ~ x1 + x3, data = data)
Model2 <- lm(x2 ~ x1 + x3 + x4, data = data)
Model3 <- lm(x2 ~ x1 + x3 + x4 + x5, data = data)
Model4 <- lm(x2 ~ poly(x1, 2) + x3, data = data)
Model5 <- lm(x2 ~ poly(x1, 2) + x3 + poly(x5, 2), data = data)
# Extracting log-likelihood and parameter counts
logLikelihood <- function(model) logLik(model)[1]
numParams <- function(model) length(coef(model))
N <- nrow(data)
# Calculating AIC and BIC
AIC_Model1 <- calculateAIC(N, coef(Model1), logLikelihood(Model1))
BIC_Model1 <- calculateBIC(N, coef(Model1), logLikelihood(Model1))

AIC_Model2 <- calculateAIC(N, coef(Model2), logLikelihood(Model2))
BIC_Model2 <- calculateBIC(N, coef(Model2), logLikelihood(Model2))

AIC_Model3 <- calculateAIC(N, coef(Model3), logLikelihood(Model3))
BIC_Model3 <- calculateBIC(N, coef(Model3), logLikelihood(Model3))

AIC_Model4 <- calculateAIC(N, coef(Model4), logLikelihood(Model4))
```

```
BIC_Model4 <- calculateBIC(N, coef(Model4), logLikelihood(Model4))

AIC_Model5 <- calculateAIC(N, coef(Model5), logLikelihood(Model5))
BIC_Model5 <- calculateBIC(N, coef(Model5), logLikelihood(Model5))

# Combining results into a table
results <- data.frame(
  Models = c("Model 1", "Model 2", "Model 3", "Model 4", "Model 5"),
  AIC = c(AIC_Model1, AIC_Model2, AIC_Model3, AIC_Model4, AIC_Model5),
  BIC = c(BIC_Model1, BIC_Model2, BIC_Model3, BIC_Model4, BIC_Model5)
)
#print data value of results
print(results)

#task 2.5 Q-Q Plot
# Spliting data into training and testing sets
set.seed(42)
sample_index <- sample(1:nrow(data), 0.8 * nrow(data))
train <- data[sample_index, ]
test <- data[-sample_index, ]
# input features and target variable
X_train <- train[, c("time", "x1", "x3", "x4", "x5")]
y_train <- train$x2
X_test <- test[, c("time", "x1", "x3", "x4", "x5")]
y_test <- test$x2
# Train Model 1: Linear Regression
model1 <- lm(x2 ~ time + x1 + x3 + x4 + x5, data = train)
y_hat_model1 <- predict(model1, newdata = test)
# Train Model 2: Random Forest
model2 <- randomForest(x2 ~ time + x1 + x3 + x4 + x5, data = train, ntree = 100)
y_hat_model2 <- predict(model2, newdata = test)
```

```
# Train Model 3: Support Vector Regressor
model3 <- svm(x2 ~ time + x1 + x3 + x4 + x5, data = train, kernel = "radial")
y_hat_model3 <- predict(model3, newdata = test)
# Train Model 4: Placeholder Model
model4 <- randomForest(x2 ~ time + x1 + x3 + x4 + x5, data = train, ntree = 50)
y_hat_model4 <- predict(model4, newdata = test)
# Train Model 5: Placeholder Model
model5 <- lm(x2 ~ time + x1 + x3 + x4 + x5, data = train)
y_hat_model5 <- predict(model5, newdata = test)
# function to calculate residuals
calculateError <- function(y, y_hat) {
  return(y - y_hat)
}

# Calculating residuals for each model
residuals_model1 <- calculateError(y_test, y_hat_model1)
residuals_model2 <- calculateError(y_test, y_hat_model2)
residuals_model3 <- calculateError(y_test, y_hat_model3)
residuals_model4 <- calculateError(y_test, y_hat_model4)
residuals_model5 <- calculateError(y_test, y_hat_model5)

# function to generate Q-Q plot
plotQQ <- function(model_error, title) {
  ggplot(data.frame(model_error = model_error), aes(sample = model_error)) +
    geom_qq(color = "#195f90") +
    geom_qq_line(color = "red") +
    ggtitle(title) +
    theme_minimal()
}
# generating Q-Q plots for each model
qq1 <- plotQQ(residuals_model1, "Q-Q Plot of Model 1 ")
qq2 <- plotQQ(residuals_model2, "Q-Q Plot of Model 2 ")
```

```
qq3 <- plotQQ(residuals_model3, "Q-Q Plot of Model 3 ")
qq4 <- plotQQ(residuals_model4, "Q-Q Plot of Model 4 ")
qq5 <- plotQQ(residuals_model5, "Q-Q Plot of Model 5 ")

# displaying the Q-Q plots
print(qq1)
print(qq2)
print(qq3)
print(qq4)
print(qq5)

#2.6 AIC, BIC, RSS and likelihood
# Creating a table to show the results
results_table_a <- tibble(
  Model = c("Model 1 ", "Model 2 ", "Model 3 ",
          "Model 4", "Model 5 "),
  AIC = c(AIC_Model1, AIC_Model2, AIC_Model3, AIC_Model4, AIC_Model5),
  BIC = c(BIC_Model1, BIC_Model2, BIC_Model3, BIC_Model4, BIC_Model5),
  RSS = c(RSS_Model1, RSS_Model2, RSS_Model3, RSS_Model4, RSS_Model5),
  Likelihood = c(Likelihood_1, Likelihood_2, Likelihood_3, Likelihood_4, Likelihood_5)
)
# Print the results table
print(results_table)

#task 2.7
#loading data agin for task 2.7
df <- read.csv('./data/data.csv')
# splitting the data into training and testing data sets
set.seed(100)  # For reproducibility
Split_Data <- initial_split(df, prop = 0.7)
training_set <- training(Split_Data)
testing_set <- testing(Split_Data)
```

```
# generating  training and testing models
X_training_model <- as.matrix(generateModel3(training_set))  # Ensuring it's a matrix
X_testing_model <- as.matrix(generateModel3(testing_set))    # Ensuring it's a matrix

# defining the thetaHat function
thetaHat <- function(X, y) {
  # Ensure X is a matrix and y is numeric
  if (!is.matrix(X)) stop("X must be a matrix")
  if (!is.numeric(y)) stop("y must be numeric")
  y <- as.matrix(y)  # Convert y to a column matrix
  if (nrow(X) != nrow(y)) stop("X and y must have the same number of rows")

  # Return the estimated coefficients
  solve(t(X) %*% X) %*% t(X) %*% y
}
#estimating model parameters using the training dataset
training_thetaHat <- thetaHat(X_training_model, training_set$x2)

#Computing model predictions for training and testing datasets
Y_training_hat <- X_training_model %*% training_thetaHat
Y_testing_hat <- X_testing_model %*% training_thetaHat

# computing the confidence intervals for predictions
error <- as.matrix(testing_set$x2) - Y_testing_hat  # Residuals
n <- nrow(X_testing_model)
z <- 1.96  # calculation 95% confidence level
sd_error <- sqrt(sum(error^2) / (n - 1))  # SD of residuals
conf_interval <- z * sd_error  # Confidence interval
print(n) + print (z) + print(sd_error) + print(conf_interval)

# creating the plots for training and testing data
```

```
    # Plot for Training Data
    training_plot <- ggplot(training_set, aes(x = x2)) +
      stat_density(geom = "line", color = "#195f90") +
      geom_vline(xintercept = conf_interval, linetype = "dashed", color = "red") +
      geom_vline(xintercept = -conf_interval, linetype = "dashed", color = "red") +
      ggtitle("Distribution of Training Data") +
      theme_minimal() +
      xlab("y") +
      ylab("Density")
    ggplotly(training_plot) %>%
      layout(
        plot_bgcolor = '#e5ecf6',
        title = "Distribution of training data",
        xaxis = list(title = "x2"),
        yaxis = list(title = "Density")
      )
# plot for Testing Data
    testing_plot <- ggplot(testing_set, aes(x = x2)) +
      stat_density(geom = "line", color = "#195f90") +
      geom_vline(xintercept = conf_interval, linetype = "dashed", color = "red") +
      geom_vline(xintercept = -conf_interval, linetype = "dashed", color = "red") +
      ggtitle("Distribution of Testing Data") +
      theme_minimal() +
      xlab("y") +
      ylab("Density")
    ggplotly(testing_plot) %>%
      layout(
        plot_bgcolor = '#e5ecf6',
        title = "Distribution of testing data",
        xaxis = list(title = "y"),
        yaxis = list(title = "Density")
      )
    #  adding error Bars to Predictions
    testing_data_with_predictions <- data.frame(
      Actual = testing_set$x2,
      Predicted = Y_testing_hat,
      Lower_CI = Y_testing_hat - conf_interval,
      Upper_CI = Y_testing_hat + conf_interval
    )
    # scatter plot with Error Bars
    error_bar_plot <- ggplot(testing_data_with_predictions, aes(x = Actual, y = Predicted)) +
      geom_point(color = "#195f90") +
      geom_errorbar(aes(ymin = Lower_CI, ymax = Upper_CI), color = "red") +
      ggtitle("Model Predictions with Confidence Intervals") +
      xlab("Actual y") +
      ylab("Predicted y") +
      theme_minimal()

    ggplotly(error_bar_plot)

    #task 3
# Convert all necessary columns to numeric
data$x1 <- as.numeric(data$x1)
data$x2 <- as.numeric(data$x2)
data$x3 <- as.numeric(data$x3)
data$x4 <- as.numeric(data$x4)
data$x5 <- as.numeric(data$x5)

# Generate the regression model
# Model includes x1, x3, x4, and x5 as inputs and x2 as output
generateModel <- function(data) {
  ones <- rep(1, nrow(data))
  theta_bias <- runif(1, -1, 1) * ones
```

```r
  model <- cbind(data$x1, data$x3, data$x4, data$x5, theta_bias)
  return(as.matrix(model))
}

# Calculate theta_hat
thetaHat <- function(model, y) {
  return(solve(t(model) %*% model) %*% t(model) %*% y)
}

# Define the response variable (y)
y <- as.numeric(data$x2)

# Generate the model and calculate theta_hat
model <- generateModel(data)
theta_hat <- thetaHat(model, y)

# Identify the two parameters with the largest absolute values in theta_hat
largest_indices <- order(abs(theta_hat), decreasing = TRUE)[1:2]
param1_hat <- theta_hat[largest_indices[1]]
param2_hat <- theta_hat[largest_indices[2]]

# Define ABC parameters
RSS_Model <- sum((y - model %*% theta_hat)^2)
epsilon <- RSS_Model * 2
num_samples <- 1000
param1_range <- c(param1_hat - abs(param1_hat), param1_hat + abs(param1_hat))
param2_range <- c(param2_hat - abs(param2_hat), param2_hat + abs(param2_hat))

# Perform rejection ABC
accepted_param1 <- numeric()
accepted_param2 <- numeric()
```

```
for (i in 1:num_samples) {
  param1_sample <- runif(1, param1_range[1], param1_range[2])
  param2_sample <- runif(1, param2_range[1], param2_range[2])

  abc_theta_hat <- theta_hat
  abc_theta_hat[largest_indices[1]] <- param1_sample
  abc_theta_hat[largest_indices[2]] <- param2_sample

  abc_Y_hat <- model %*% abc_theta_hat
  abc_RSS <- sum((y - abc_Y_hat)^2)

  if (abc_RSS <= epsilon) {
    accepted_param1 <- c(accepted_param1, param1_sample)
    accepted_param2 <- c(accepted_param2, param2_sample)
  }
}

# Create a data frame for the results
abc_results <- data.frame(Theta1 = accepted_param1, Theta2 = accepted_param2)

# Plot the joint posterior distribution
plot <- ggplot(abc_results, aes(x = Theta1, y = Theta2)) +
  geom_point(alpha = 0.5, color = "purple") +
  theme_minimal() +
  labs(title = "Joint Posterior Distribution", x = "Theta1", y = "Theta2")

# Display the plot
print(plot)
```

7. Github Link

https://github.com/dipinbaral96/Assignment-of-Introduction-To-Statistical-Methods-For-Data-Science.git