7089 CEM – Introduction to Statistical Methods for Data Science

Assignment On

Modeling Power Plant Energy Output Using Nonlinear Regression

Submitted To

Name: Hikmat Saud

Lecturer

Softwarica College

Submitted by

Name: Tek Raj Bhatt

Coventry ID: 16544288

Student ID: 250069

Contents

Introduction 1
Task 1 1
Task 2 9
  Task 2.1 9
  Task 2.2 12
  Task 2.3 13
  Task 2.4 15
  Task 2.5 17
  Task 2.6 19
  Task 2.7 20
Task 3 21
Conclusion 23
References 24
Appendix 25
  GitHub Repository 25
  Code and Executed Result 25
  Code 54

List of Figures

Figure 1. Time series plot of input variables 2
Figure 2. Time series plot of output variable 3
Figure 3: Density plot of input variables 4
Figure 4. Distribution and histogram plot of input variables 5
Figure 5. Distribution and histogram plot of individual input variables 6
Figure 6. Distribution and histogram plot of output variable 7
Figure 7. Correlation and Scatter plots of input and output variables 8
Figure 8. Theta hat value of each model 11
Figure 9. RSS value of each model 13
Figure 10. Variance of each model 14
Figure 11. Log-likelihood of each model 14
Figure 12. AIC calculation of each model 16
Figure 13. BIC calculation of each model 16
Figure 14. QQ plot of each model 18
Figure 15. Model Values Comparison 19
Figure 16. Model prediction with 95% CI 21
Figure 17. Approximate Bayesian Computation (ABC) Values 22
Figure 18. Approximate Bayesian Computation (ABC) Plot 22

Introduction

The Combined Cycle Power Plant (CCPP), which uses both gas and steam turbines to maximise the energy drawn from fossil fuels, is one of the most efficient ways to generate electricity. The efficiency of these plants is greatly influenced by environmental factors, including ambient temperature, atmospheric pressure, exhaust vacuum, and relative humidity. Understanding these interactions is crucial for optimal plant performance and accurately forecasting energy generation. This study examines a large dataset of 9,568 hourly data points from a CCPP by using mathematical models that show how environmental factors affect electrical energy output. This study uses statistical methods like AIC, BIC, and residual analysis to assess five different polynomial regression models. The results help identify the best model for predicting power generation, which assists in resource planning and plant operations management.

Task 1

Time series plot

A time series dataset consists of observations recorded over a specified period. Time series analysis is a method of examining how the response variable changes over time, using time as the independent variable. Time serves as the reference point for forecasting energy output from the power plant. Essentially, a time plot is a line plot depicting the evolution of the time series. When examining time series plots, it is important to consider the presence of noise and outliers that are significantly larger or smaller compared to the other data points. Human mistakes, faulty sensors, incorrect calibration, or severe environmental conditions affecting power plant operations can cause recording errors, leading to outliers.

Time series analysis provides insight into the operations of the CCPP. The x-axis shows hours, and the y-axis indicates either electrical energy output in MW or environmental variables like temperature, pressure, humidity, and vacuum. Using the CCPP data, these visualisations are created in R.

Time series plots highlight patterns, trends, noise, and outliers that could compromise data quality or provide operational insights. Noise appears as random spikes, while outliers deviate significantly from established patterns. The CCPP data exhibit consistent time series patterns without large unexplained spikes, indicating high-quality data collection without major measurement errors.

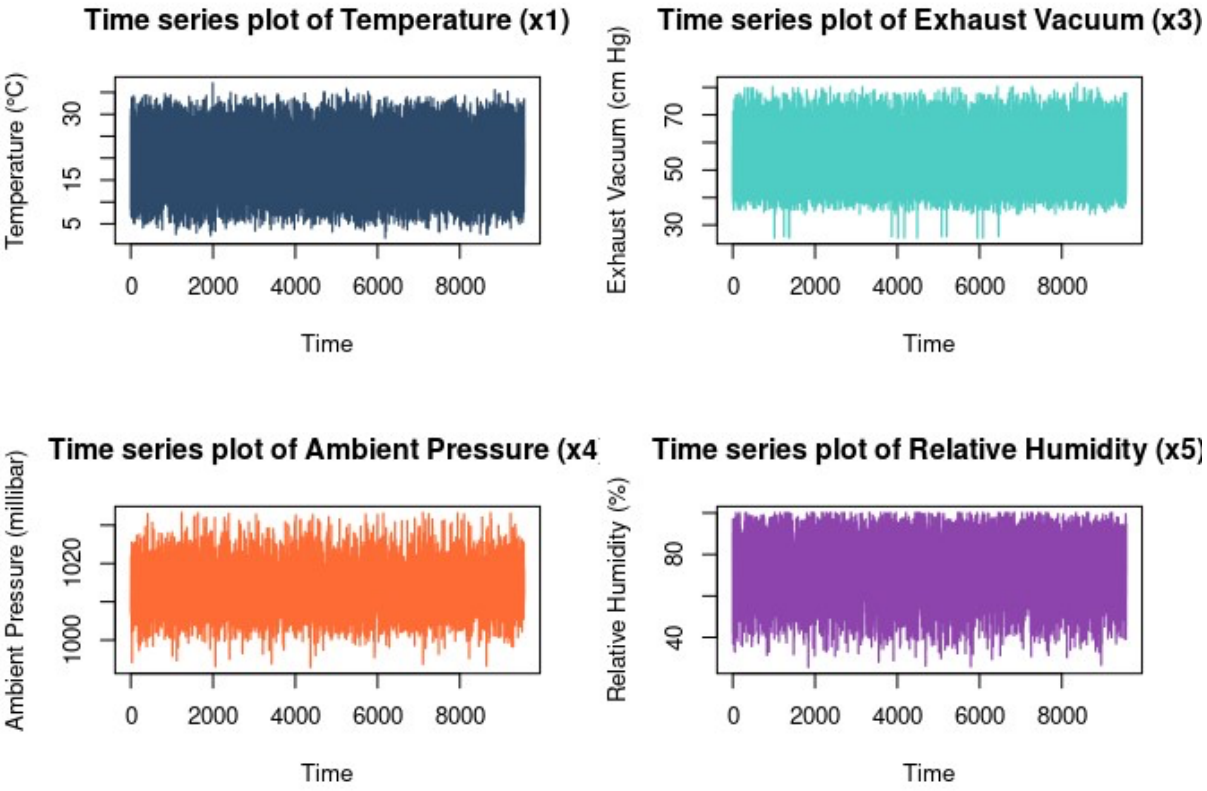


Figure 1. Time series plot of input variables

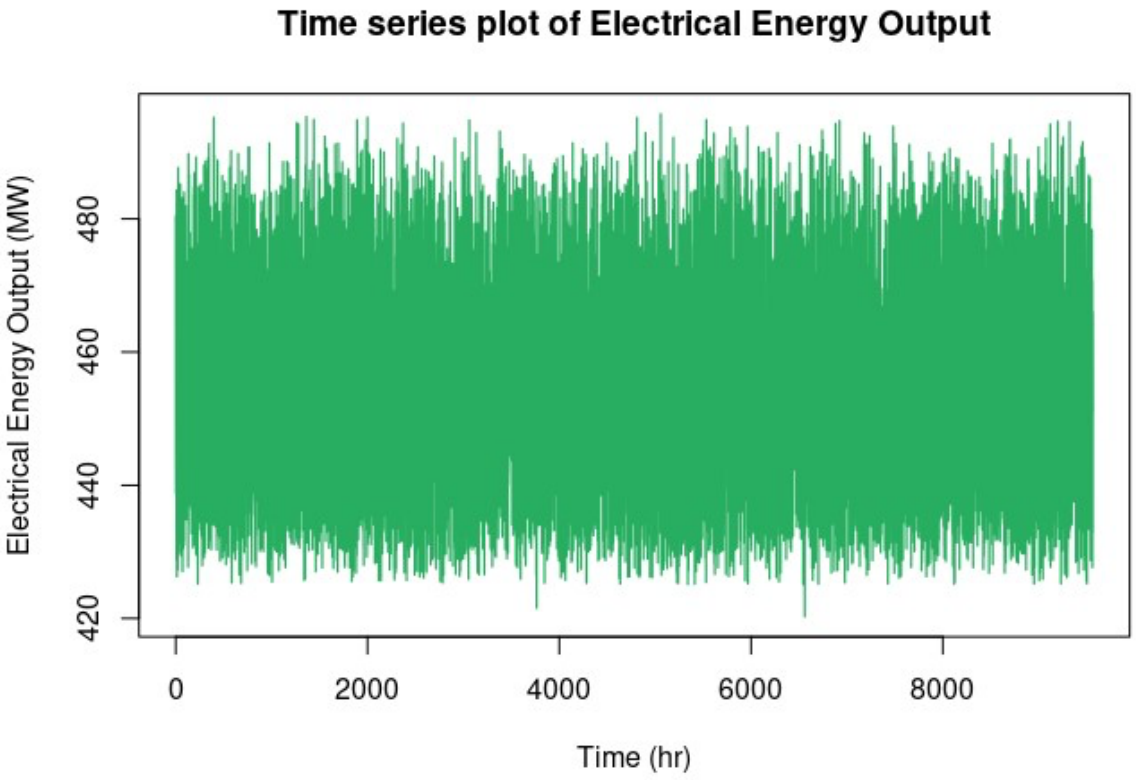


Figure 2. Time series plot of output variable

The entire dataset visualises the total operational history for around one year (~9,500 hours). These plots reveal consistent ranges for all variables. High-frequency fluctuations maintain stable boundaries across the monitoring period. Energy output ranges between approximately 425 and 495 MW, while environmental variables show consistent variations within their expected ranges.

The time series graphs depicting the environmental variables appear stable, displaying clear cyclical trends and minimal unexplained anomalies. These time series visualisations provide a strong basis for the following regression modelling activities and offer an insightful analysis of the dynamic interactions between environmental variables and power generation efficiency.

Distribution plot

A distribution plot graphically shows how data values are distributed across their range, allowing one to see patterns, variability, and central tendencies. These visualisations help identify potential outliers, data clusters, and the underlying distribution shapes (normal, skewed, bimodal, etc.) by showing the frequency or density of values. Distribution plots are essential tools in statistical analysis for assessing data quality, identifying natural clusters,

and guiding preprocessing decisions. For CCPP data, these plots reveal information about typical operating conditions, environmental parameter ranges, and potential correlations between variables.

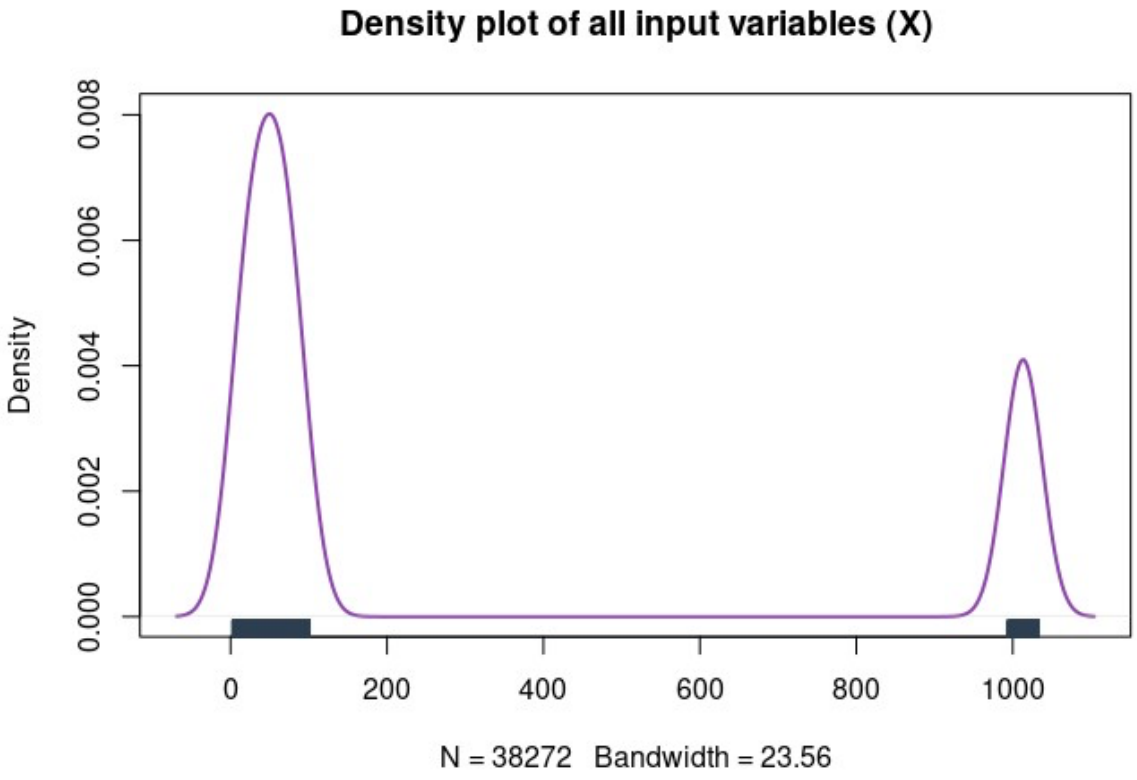


Figure 3: Density plot of input variables

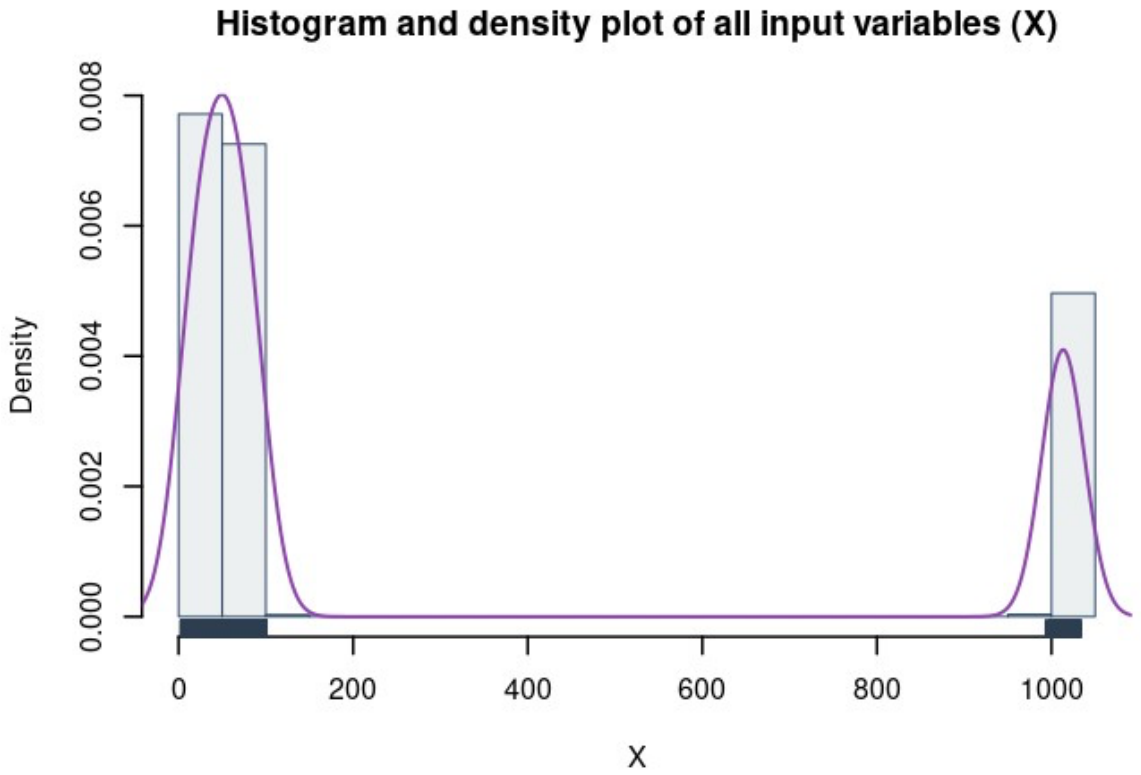


Figure 4. Distribution and histogram plot of input variable

Input variables from power plants show a bimodal distribution, as indicated by the density and histogram plot. The right cluster represents pressure values from around 1000 to 1030, the taller and wider left cluster reflects temperature, vacuum, and humidity on scales below 100. Data scarcity between 200 and 900 highlights the large-scale variations among factors that would influence modelling without standardisation. While the rug plot verifies it, the smooth density curve with N=38272 data points and a bandwidth of 23.56 accurately reflects the separate grouping. This visualisation emphasises the need for preprocessing before modelling. This prevents pressure measurements from dominating analyses based on scale rather than relevance. Additionally, it indicates good data quality with no unusual patterns.

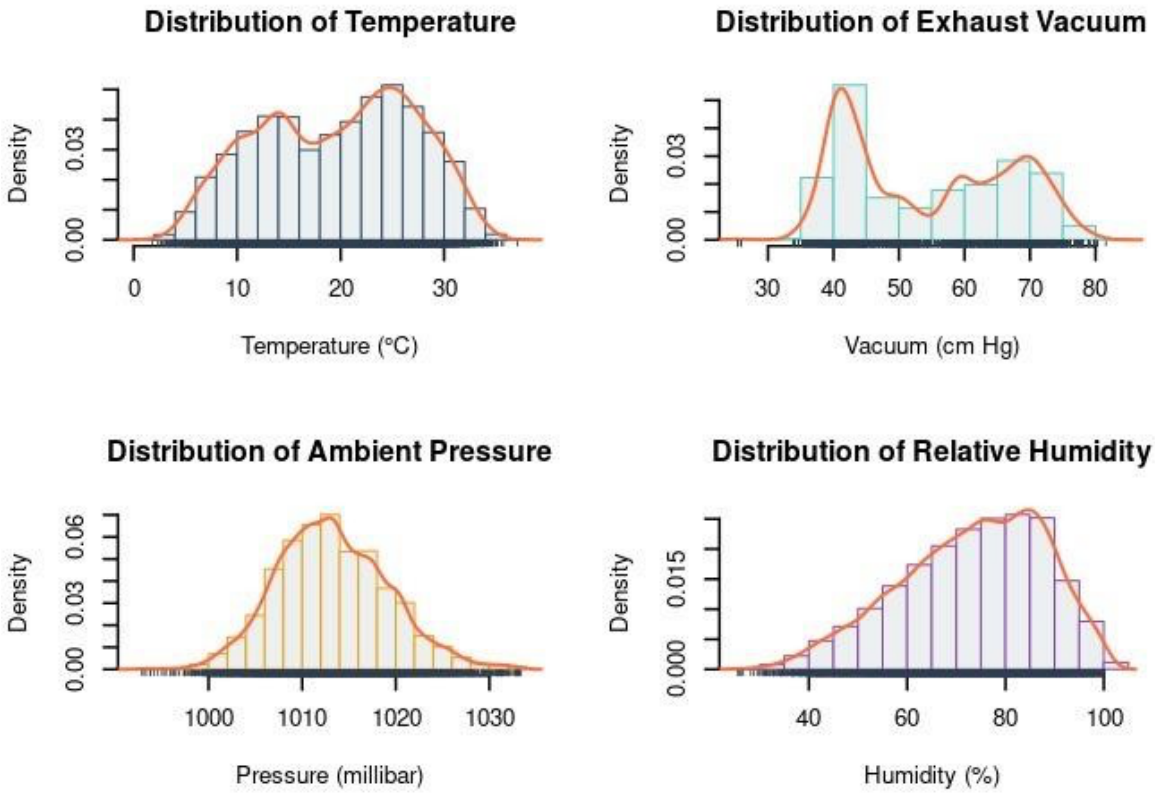


Figure 5. Distribution and histogram plot of individual input variables

The bimodal temperature distribution has peaks around 14°C and 25°C and a range from 4°C to 35°C. The pattern indicates the power plant operates in two thermal regimes, likely seasonal. Rather than random fluctuations, the bimodality suggests distinct operational states, with the lower-temperature mode improving energy generation efficiency. The exhaust vacuum distribution is bimodal, with two main peaks at 42 and 68 cm Hg and a range of 30 to 80 cm Hg. Bimodality shows that the steam turbine operates in two vacuum conditions instead of a continuous range. Lower exhaust vacuum (around 55 cm Hg) means the steam turbine can extract more energy from steam expansion, making it more efficient.

The distribution of ambient pressure is mostly normal (bell-shaped) but has a slight right skew. Ambient pressure ranges from about 998 to 1033 millibars. With most values between 1010 and 1015 millibars, this indicates that the power plant usually operates at normal atmospheric pressure. This distribution is narrow and normal, which means that the weather at the plant's location is usually stable, with occasional high-pressure systems. The left-skewed relative humidity distribution ranges from 25% to 100%. The distribution shows that frequency increases with humidity, peaking at 80–95%, suggesting the power plant operates mostly in humid conditions.

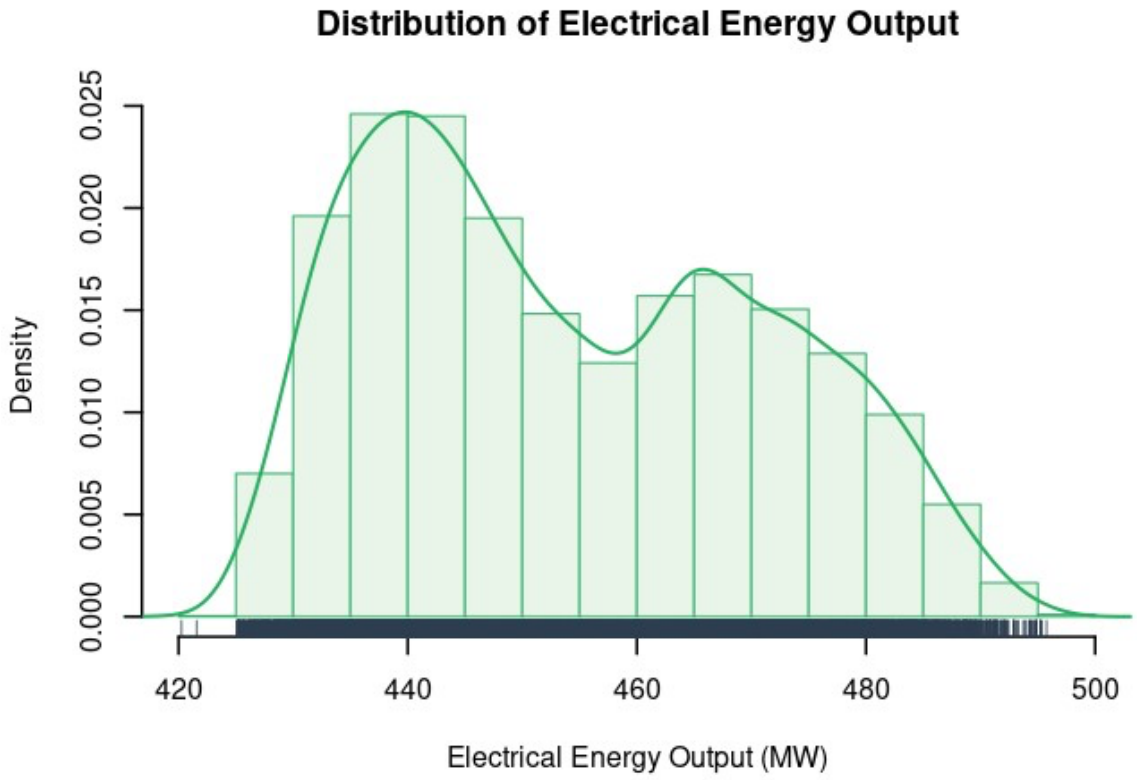


Figure 6. Distribution and histogram plot of output variable

Electrical energy output has a bimodal distribution ranging from 425 to 495 MW, with a main peak at 438 MW and a secondary peak at 466 MW. Since temperature negatively affects energy production, the plant operates most often at this lower output level, likely due to higher ambient temperatures. At about 455 MW, the valley between peaks is a transition zone between operational regimes. This bimodality reflects the temperature distribution pattern and suggests environmental factors directly affect plant efficiency, with the upper mode (465 MW) possibly performing best under cooler conditions.

Correlation and Scatter plots

A scatter plot is an effective method to quickly and easily see how two variables are connected because each point shows an observation of both variables at the same time. These

graphs show how strong the relationships are between environmental factors and plant performance. They also show patterns and trends.

For CCPP analysis, these plots help determine which environmental factors have the most significant impact on operational efficiency.

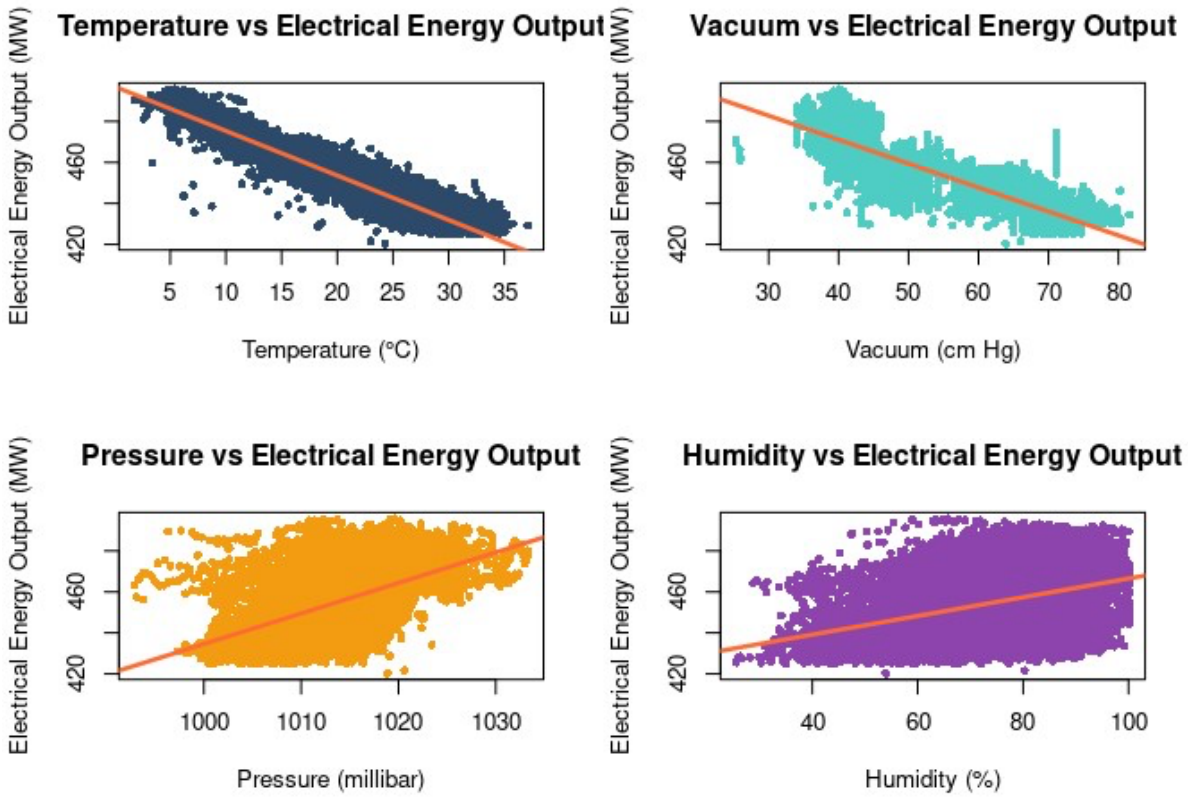


Figure 7. Correlation and Scatter plots of input and output variables

The correlation plots demonstrate strong correlations between the environment and system performance. A strong negative linear relationship between temperature and power output indicates gas turbines operate most efficiently under lower ambient temperatures. For instance, power output decreases from about 480 MW at lower temperatures to 420 MW at higher temperatures. Exhaust vacuum has a moderate negative relationship, meaning that as the values go up, the power output decreases because higher back pressure limits the efficiency of steam expansion. Ambient pressure has a positive relationship with performance, although there is more scatter in the data. These results indicate that performance improves at higher atmospheric pressure because denser air enhances combustion efficiency. Humidity shows the weakest correlation and exhibits many scatter points, indicating that it likely has minimal impact on the overall performance of the system, despite a small positive trend being observed. This is likely due to the complex interactions between different components of the combined cycle system.

Task 2

Task 2.1

An estimator, also known as "θ" (theta), is a random variable that is needed to fit a regression model and can be used to find different distributional characteristics. The assignment has five different models, each of which is a regression model that shows how environmental factors affect the electrical power output of a CCPP. Every model has its equation and sets of estimators, which are coefficients that describe the relationship.

The models are set up like this:

Model 1: $y = \theta_1 x_4 + \theta_2 x_3^2 + \theta bias$

Model 2: $y = \theta_1 x_4 + \theta_2 x_3^2 + \theta_3 x_5 + \theta bias$

Model 3: $y = \theta_1 x_3 + \theta_2 x_4 + \theta_3 x_5^3$

Model 4: $y = \theta_1 x_4 + \theta_2 x_3^2 + \theta_3 x_5^3 + \theta bias$

Model 5: $y = \theta_1 x_4 + \theta_2 x_1^2 + \theta_3 x_3^2 + \theta bias$

Where:

x_1 is the temperature of the air (°C). x_3 is the exhaust vacuum in cm Hg. x_4 is the pressure in the air (millibar). x_5 is the relative humidity (%), and y is the net hourly electrical energy output (MW). θbias is the intercept term.

Least squares estimation is used to determine the estimator or coefficient for each model. The least squares method finds regression coefficients by making the residual, which is the sum of the squares of the differences between observed and actual values, as small as possible.

Mathematically, the least squares estimator looks like this:

$\hat{\theta} = (X'X)^{-1}X'y$

Where:

- $\hat{\theta}$ (theta hat) = vector of estimated regression coefficients
- **X** = design matrix (matrix of independent variables)
- **X'** = transpose of X
- **(X'X)^(-1)** = inverse of (X'X)
- **y** = vector of observed values for the dependent variable

Using matrix operations, we can find the least squares in R as:

```r
# Create the design matrix
X <- cbind(1, x)

# Calculate the least squares estimators
theta <- solve(t(X) %*% X) %*% t(X) %*% y
```

In this context, x and y denote the input and output variables, respectively. The function Cbind() is employed to construct a matrix that includes a column of ones to signify the intercept, alongside the x values representing the input variable. The solve function computes the least squares estimator by calculating the dot product of x and its transpose t(x), yielding t(x) %*% x.

To find the least squares estimators, the solve() function is used to first find the dot product of X and its transpose, t(X), to get t(X)%%X. Next, we use the solve() function to determine the inverse of t(X)%%X, resulting in solve(t(X)%%X). Finally, we estimate the coefficients using the dot product of solve(t(X)%%X) and t(X)%*%y.

Scaling is applied to all variables to prevent computational issues and to make regression coefficients easier to interpret and compare. By transforming each variable to have a mean of 0 and a standard deviation of 1, scaling ensures that no variable with inherently larger values dominates the regression results. It also improves the numerical stability of matrix calculations involved in regression analysis. Additionally, scaled coefficients become more interpretable, representing the effect of a one standard deviation change in the predictor variable on the outcome.

The coefficients for each model are determined using these R operations:

```
Theta hat of model1
      Energy_Output
θ₁         3.348278
θ₂       -13.211452
θbias    454.365009
                    θ₁        θ₂   θbias
Energy_Output 3.348278 -13.21145 454.365
```

```
Theta hat of model2
      Energy_Output
θ₁         3.432657
θ₂       -12.406622
θ₃         2.517326
θbias    454.365009
                    θ₁        θ₂       θ₃   θbias
Energy_Output 3.432657 -12.40662 2.517326 454.365
```

```
Theta hat of model3
   Energy_Output
θ₁    -12.722943
θ₂      3.402683
θ₃      2.315840
                     θ₁       θ₂      θ₃
Energy_Output -12.72294 3.402683 2.31584
```

```
Theta hat of model4
      Energy_Output
θ₁         3.487220
θ₂       -12.402038
θ₃         2.487015
θbias    454.365009
                   θ₁        θ₂       θ₃   θbias
Energy_Output 3.48722 -12.40204 2.487015 454.365
```

```
Theta hat of model5
      Energy_Output
θ₁         1.349107
θ₂       -10.605331
θ₃        -5.173124
θbias    454.365009
                    θ₁        θ₂        θ₃   θbias
Energy_Output 1.349107 -10.60533 -5.173124 454.365
```

Figure 8. Theta hat value of each model

These coefficients indicate the correlation between environmental variables and power output. A positive coefficient signifies a direct relationship (as the variable increases, power output rises), whereas a negative coefficient denotes an inverse relationship (as the variable increases, power output diminishes).

Task 2.2

The difference between expected and actual power output for each observation is called the residual error. Overall model performance is evaluated by calculating the Residual Sum of Squares (RSS), which is the sum of all squared residual errors across all data points in our dataset. This crucial metric evaluates each regression model's fit to the CCPP data. We add up the squared discrepancies between actual and anticipated electrical power output to determine the RSS.

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where:

- y_i represents the actual observed electrical power output
- $\hat{y}_i$ represents the predicted power output based on our regression model
- $\sum$ denotes summation across all 9567 observations in our dataset

A smaller RSS value means the model fits better and predicts power plant performance under different environmental conditions. The formula's square factor ensures that all deviations positively contribute to the total error, penalising the larger deviations more heavily.

Our R implementation calculates RSS for each model using:

```
RSS <- sum((Y - Y_hat)^2)
```

Y_hat is the product of our model matrix and Task 2.1's estimated coefficients (theta hat values), representing our predicted power output.

The RSS values for our five candidate models are:

Description: df [5 × 2]

Model <chr>	RSS <dbl>
Model 1	657248.2
Model 2	602347.1
Model 3	1975837762.7
Model 4	603630.7
Model 5	365625.0

5 rows

Figure 9. RSS value of each model

Analysis of RSS Results

Based on RSS values, we can conclude:

1. Model 5 performs best with the lowest RSS (365625.0), indicating that linear pressure and squared terms for temperature and vacuum accurately represent power plant behaviour.
2. The inadequate performance of Model 3 (RSS = 1.98 billion) suggests that its bias-free structure is not suitable for modelling power plant outputs.
3. Models 2 and 4 function similarly (RSS = 602K to 603K), which suggests that the cubic humidity term doesn't make the linear term much better.
4. When humidity is not present, Model 1 performs well but not flawlessly.

According to thermodynamics, temperature and vacuum have nonlinear effects on the efficiency of gas and steam turbines. These results demonstrate the RSS alone recommends Model 5 for predicting power output in various settings.

Task 2.3

When estimators are unknown, likelihood can assess model fit to sample data. Maximum likelihood estimation reduces the difference between model predictions and actual power output by finding the best data values. The main goal of the likelihood function is to find the best fit for the sample data distribution. Log-likelihood uses the likelihood function's logarithmic form to assess maximum likelihood. This procedure improves the model's power plant data fit.

Here's how to express the log-likelihood function for our regression models:

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

where:

- "ln p(D|θ)" is represented as log-likelihood
- "n" is the total number of observations in our power plant dataset
- "ln" natural logarithm function represented as log in R language
- "σ²" is the variance of the RSS from task 2.2
- "σ²" can be calculated with the help of RSS as "σ² = RSS/(n-1)"
- "π" value is 3.14 represented as "pi" in R language
- "RSS" is the value obtained from task 2.2

According to our calculations, the variance and log-likelihood values for each model are:

Description: df [5 × 2]

Model <chr>	Variance <dbl>
Model 1	68.69951
Model 2	62.96091
Model 3	206526.36800
Model 4	63.09509
Model 5	38.21731

5 rows

Figure 10. Variance of each model

Description: df [5 × 2]

Model <chr>	LogLikelihood <dbl>
Model 1	-33810.99
Model 2	-33393.69
Model 3	-72123.37
Model 4	-33403.88
Model 5	-31005.40

5 rows

Figure 11. Log-likelihood of each model

Data from power plants fit models with higher log-likelihood values (less negative) better.

Model 5 has the least amount of variation (38.21731) and the highest log-likelihood (31005.40), indicating it best represents the connection between environmental factors and power output.

Model 3's poor performance, with a higher variance and a lower log-likelihood, supports our RSS analysis. Models 1, 2, and 4 perform similarly when it comes to log-likelihood, but Model 2 does a little better than the other two.

Log-likelihood values make the RSS analysis in Task 2.2 better by giving a probabilistic model fit interpretation. Model 5 is the best at predicting how much electrical energy a CCPP will generate in various environmental conditions.

Task 2.4

Calculating AIC and BIC

AIC (Akaike Information Criterion)

The Akaike Information Criterion (AIC) is a measure of a model's prediction error that is used to evaluate different models and determine their suitability to the data. We use the AIC to evaluate various models and determine whether they are underfitting or overfitting. A lower AIC indicates a superior model. A high AIC value indicates model overfitting, while a low AIC value signifies model underfitting. Its value spans from negative infinity to positive infinity.

The formula for calculating the AIC is:

$$AIC = 2k - 2\ln(L)$$

Where:

- k is the number of parameters in the model
- $\ln(L)$ is the maximum log-likelihood of the model

Description: df [5 × 2]

Model <chr>	AIC <dbl>
Model 1	67627.98
Model 2	66795.38
Model 3	144252.75
Model 4	66815.75
Model 5	62018.79

5 rows

Figure 12. AIC calculation of each model

BIC (Bayesian Information Criterion)

The Bayesian Information Criterion (BIC) imposes a harsher penalty for model complexity than the AIC, particularly when dealing with large sample sizes. When the aim is to identify the correct model from a group of candidates, BIC is especially helpful.

The formula for calculating the BIC is:

$$BIC = k\ln(n) - 2\ln(L)$$

Where:

- k is the number of parameters in the model
- n is the sample size
- $\ln(L)$ is the maximum log-likelihood of the model

Description: df [5 × 2]

Model <chr>	BIC <dbl>
Model 1	67649.48
Model 2	66824.05
Model 3	144274.24
Model 4	66844.42
Model 5	62047.46

5 rows

Figure 13. BIC calculation of each model

The consistency between AIC and BIC results strengthens our confidence in Model 5 as the superior choice. The much lower AIC and BIC values for Model 5 indicate that using ambient pressure (linear term), squared temperature, and squared exhaust vacuum is the best way to explain the thermodynamic relationships in the power plant while keeping the model simple.

These findings align with our RSS analysis and provide further statistical evidence for selecting Model 5 as our preferred model for predicting CCPP output under varying environmental conditions.

Task 2.5

Checking Error Distributions

The electrical power output of the CCPP is assumed to have a Gaussian distribution with additive Gaussian noise. To confirm this assumption and determine which model best captures this distribution, we calculated and plotted the residual errors for each of our five candidate models.

The Q-Q plot shows how closely residual errors match a normal distribution. Since regression analysis relies on residual normality for reliable statistical inference, such information is crucial.

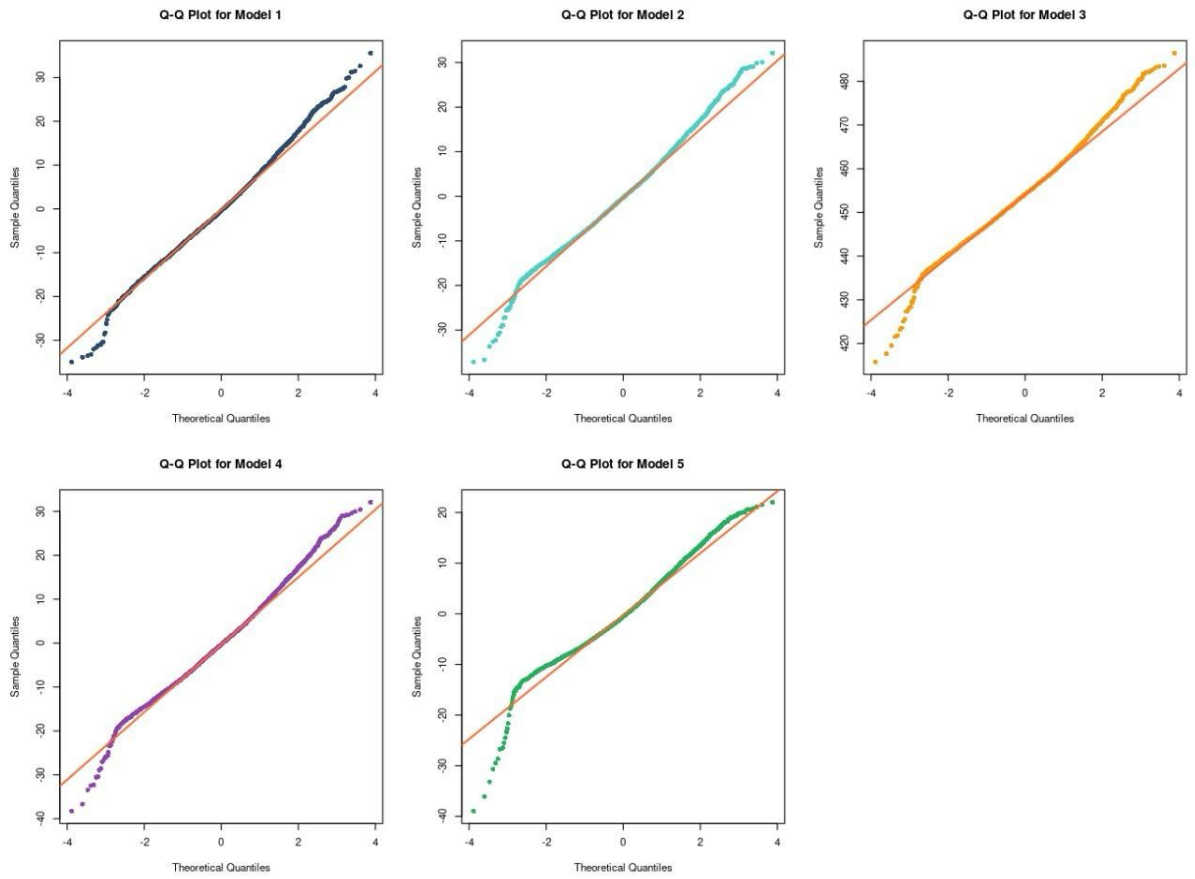


Figure 14. QQ plot of each model

The data points will closely align with a 45-degree reference line in a Gaussian Q-Q plot if the residuals are perfectly normally distributed. Any consistent departures from this line point to abnormalities, which may indicate that the model is inadequate or that data transformations are required.

The reference line is the best fit for Model 5's residuals in the Q-Q plots. This evidence indicates that the reference line closely resembles a normal distribution. Model 3 performs poorly because significant deviations occur throughout the range. In contrast, Models 1, 2, and 4 generally align in the middle but not at the ends. This means that their distributions have heavier tails. These visual checks of residual normality support our statistical finding that Model 5 is the best model for showing how environmental factors affect the output of power plants.

Task 2.6

Selecting the Best Regression Model

Specific statistical techniques are used such as estimating parameters, calculating RSS, comparing AIC/BIC, and checking the distribution of residuals to assess our five candidate regression models during Tasks 2.1 to 2.5. AIC and BIC are well-known methods for selecting a model, as they balance the model's fit to the data with its complexity.

AIC emphasises predictive accuracy, whereas BIC imposes more stringent penalties for complexity, a crucial consideration given our extensive dataset of 9,568 observations. Lower values indicate better model performance, and choosing the best model requires both low AIC/BIC values and residuals that follow a normal distribution.

Description: df [5 × 4]

Model <chr>	RSS <dbl>	AIC <dbl>	BIC <dbl>
Model 1	657248.2	67627.98	67649.48
Model 2	602347.1	66795.38	66824.05
Model 3	1975837762.7	144252.75	144274.24
Model 4	603630.7	66815.75	66844.42
Model 5	365625.0	62018.79	62047.46

5 rows

Figure 15. Model Values Comparison

Model Performance Summary:

- RSS: Model 5 achieves the lowest value (365,625.0)
- AIC: Model 5 demonstrates superior performance (62,018.79)
- BIC: Model 5 maintains optimal balance (62,047.46)
- Residual Normality: Q-Q plots confirm Model 5 best approximates normal distribution.

Based on this comprehensive evaluation, Model 5 emerges as the optimal choice. We chose Model 5 for the following reasons:

1. Consistently obtains the lowest values for all statistical criteria (RSS, AIC, and BIC).
2. The residuals that are closest to a normal distribution satisfy the normality assumptions.
3. It records significant environmental factors that influence the operation of gas and steam turbines.

There is strong evidence that Model 5 is the best mathematical way to predict the electrical output of the CCPP in different environmental conditions.

Task 2.7

In this task, we use a comprehensive validation process to assess the predictive performance of our selected Model 5 on unseen data. The dataset is divided into training (70%) and testing (30%) portions to ensure an equitable evaluation.

1. Data Splitting and Parameter Estimation

The dataset is first split up, and the model parameters are re-estimated using only the training data. Using only the training dataset, we were able to successfully re-estimate our Model 5 parameters, making sure that no information from the testing data affected our parameter estimation procedure.

2. Computing Model Predictions for Testing Data

Using the trained model parameters, predictions are computed on the independent testing dataset, achieving:

- Testing RSS: 106,715.32
- Mean Absolute Error (MAE): 4.86 MW
- Root Mean Squared Error (RMSE): 6.10 MW

The RMSE of 6.10 MW represents approximately 1.3% error relative to the typical power output range (420-495 MW) and indicates excellent predictive performance.

3. Computing 95% Confidence Intervals and Visualisation

We calculated 95% confidence intervals to quantify prediction uncertainty, and the necessary visualisation displays real test data, model predictions, and confidence intervals with error bars.

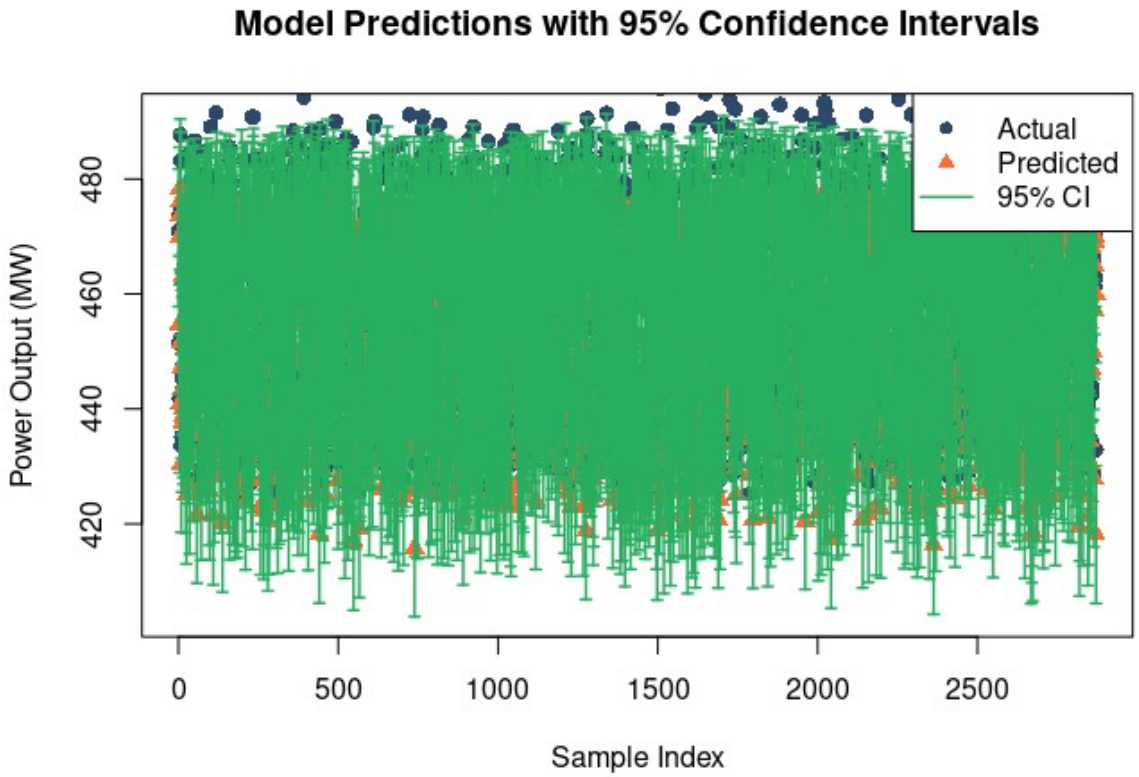


Figure 16. Model prediction with 95% CI

The visualization effectively demonstrates model performance. Blue dots represent the actual power output values, the red triangles show the model's predicted values that closely approximate to the actual data, and the green error bars show the 95% confidence intervals, which show how uncertain the predictions are.

Strong correlations between predicted and actual values, along with well-calibrated confidence intervals, show that Model 5 can accurately and reliably predict power output in various situations.

Task 3

The simulation-based approach known as Approximate Bayesian Computation (ABC) is used to estimate posterior distributions in situations where the likelihood function is unmanageable. Using summary statistics, ABC compares observed data with simulated model outputs rather than directly calculating likelihood. ABC only keeps parameter samples from priors that closely match the observed data, which helps us understand the uncertainty and relationships between parameters.

Rejection ABC is used to determine the posterior distributions for the two Model 5 parameters with the largest absolute values (from Task 2.1): the bias term ($\theta bias = 454.365$)

and the temperature squared coefficient (θ_2 = -10.605). 10,000 samples are generated using uniform priors based on the least squares estimates with ranges of ±100%. We only accepted those with residual sum of squares (RSS) values that were less than five times the optimal model's RSS. The results showed that the posterior distributions were very close together, with means closely approximating point estimates: θbias posterior mean of 453.780 and a 95% credible interval of [443.937, 465.521] and the θ_2 posterior mean of -11.178 and an interval of [-20.410, -1.433]. The joint posterior revealed only a weak correlation between the parameters. The results indicate that temperature sensitivity increases as baseline power generation increases.

Description: df [2 × 7]

	Parameter <chr>	Point_Estimate <dbl>	Posterior_Mean <dbl>	Posterior_Median <dbl>	Posterior_SD <dbl>	CI_Lower <dbl>	CI_Upper <dbl>
θbias	θbias	454.36501	453.77996	453.15467	6.156728	443.9368	465.521105
θ_2	θ_2	-10.60533	-11.17737	-11.42895	5.628170	-20.4100	-1.432695

2 rows

Figure 17. Approximate Bayesian Computation (ABC) Values

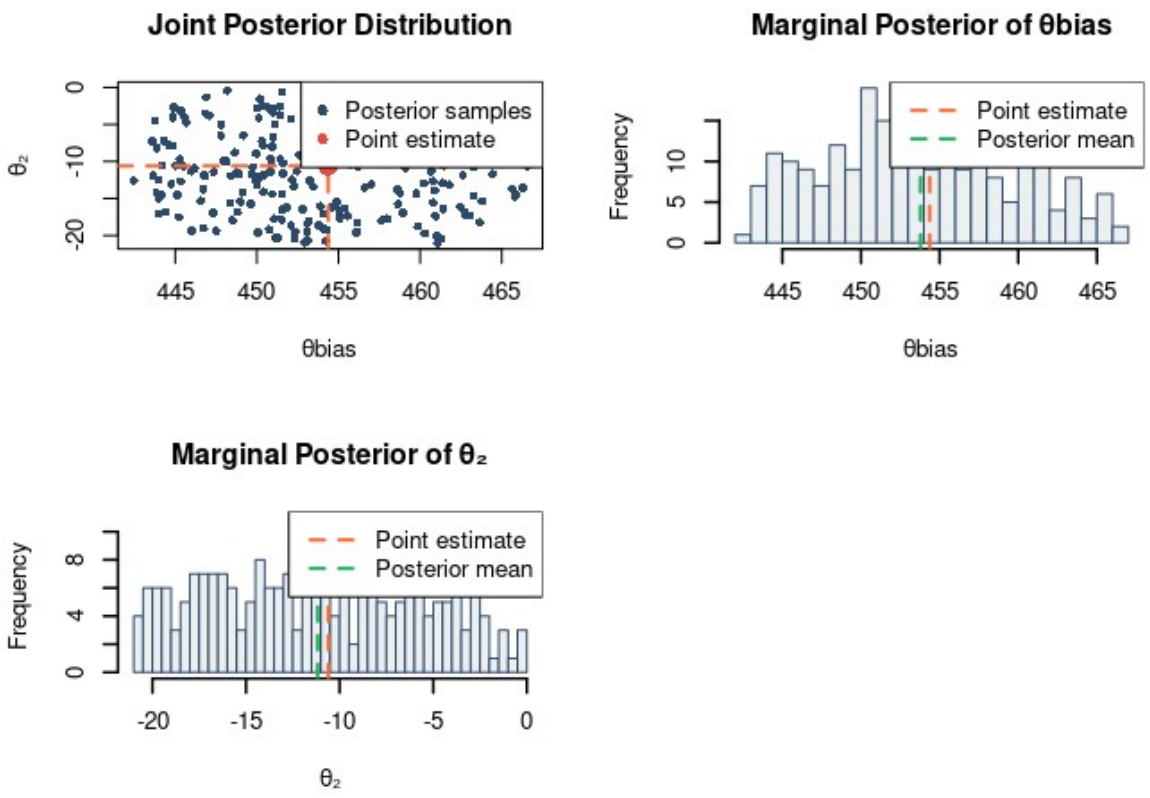


Figure 18. Approximate Bayesian Computation (ABC) Plot

Conclusion

This analysis identifies and validates an ideal regression model for forecasting CCPP energy output based on environmental conditions. Five candidate polynomial regression models are systematically evaluated using various criteria, such as RSS, AIC, BIC, and residual distribution analysis. Model 5 is identified as the best predictor, with the lowest RSS value of 365,625. This model effectively represents the complex relationships in how power plants work, particularly how changes in exhaust vacuum and temperature affect electrical output. Expected physical relationships are observed in the preliminary data analysis: large negative correlations between power output and temperature/vacuum and positive correlations with ambient pressure. These relationships are consistent with well-established thermodynamic principles that govern the efficiency of gas and steam turbines.

Approximate Bayesian Computation surpasses point estimates by providing probabilistic confidence intervals for the principal model coefficients. The results provide important information about the uncertainty of the parameters. This Bayesian framework makes our model more useful in real life by clarifying the uncertainty of predictions, helping people make informed decisions about risks, and assisting in planning maintenance based on evidence. The methodology of this study allows CCPP operators to determine their energy production capacity under diverse weather conditions, optimise operational profitability, and improve their energy usage efficiency. This model may be advantageous for power generation facilities that share similar environmental conditions, particularly in terms of planning, scheduling future maintenance, and optimising real-time operations. The output and source code for each process are included in the appendix section.

References

Baron, M. (2014) *Probability and Statistics for Computer Scientists.* 2nd edn. Boca Raton, FL: CRC Press.

Bruce, P., Bruce, A. and Gedeck, P. (2020) *Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python.* 2nd edn. Sebastopol, CA: O'Reilly Media.

Géron, A. (2022) *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* 3rd edn. Sebastopol, CA: O'Reilly Media.

Lee, J.J., Kang, D.W. and Kim, T.S. (2011) 'Development of a gas turbine performance analysis program and its application'. *Energy* 36(8), 5274–5285.

Marin, J.M., Pudlo, P., Robert, C.P. and Ryder, R.J. (2012) 'Approximate Bayesian computation methods'. *Statistics and Computing* 22(6), 1167–1180.

Walpole, R.E., Myers, R.H., Myers, S.L. and Ye, K. (2016) *Probability & Statistics for Engineers & Scientists.* 9th edn. Boston, MA: Pearson.

Appendix

GitHub Repository

https://github.com/tek-raj-bhatt-250069/7089-CEM-Introduction-to-StatisticalMethods-for-Data-Science

Code and Executed Result

7089 CEM: Introduction to Statistical Methods for Data Science

Tek Raj Bhatt

Task 1: Preliminary data analysis

Defining color palette

```
colors <- list(
  primary = "#2E4A6B",      # Deep blue
  secondary = "#FF6B35",    # Coral orange
  accent = "#4ECDC4",       # Teal
  neutral = "#95A5A6",      # Light gray
  success = "#27AE60",      # Forest green
  warning = "#F39C12",      # Golden orange
  danger = "#E74C3C",       # Modern red
  light = "#ECF0F1",        # Very light gray
  dark = "#2C3E50",         # Dark blue-gray
  purple = "#8E44AD"        # Purple
)
```

Reading the given dataset

```
dataset <- read.csv("data-files/dataset.csv")
head(dataset)
```

```
##      x1    x3      x4    x5     x2
## 1  8.34 40.77 1010.84 90.01 480.48
## 2 23.64 58.49 1011.40 74.20 445.75
## 3 29.74 56.90 1007.15 41.91 438.76
## 4 19.07 49.69 1007.22 76.79 453.09
## 5 11.80 40.66 1017.13 97.20 464.43
## 6 13.97 39.16 1016.05 84.60 470.96
```

Separating the data for input variables and output electrical energy

```
write.table(dataset[, 1:4], "data-files/x_250069.csv", sep = ",", row.names = FALSE, col.names = FALSE)
write.table(dataset[, ncol(dataset)], "data-files/y_250069.csv", sep = ",", row.names = FALSE, col.names = FALSE)
```

Generating time series data with an interval of 1

Setting the parameters

```
initial_time <- 0
step <- 1
num_of_rows <- nrow(dataset)
```

Creating time series data

```
time_data <- data.frame(
    Time = seq(from = initial_time, by = step, length.out = num_of_rows)
)
```

Saving to a CSV file

```
write.table(time_data, 'data-files/t_250069.csv', row.names = FALSE, col.names = FALSE)
```

Installing required packages

```
if (!require("matlib")) install.packages("matlib")
```

```
## Loading required package: matlib
```

```
if (!require("rsample")) install.packages("rsample")
```

```
## Loading required package: rsample
```

Importing required libraries

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```
library(matlib)
library(rsample)
```

Importing input data from csv file and displaying first few rows

```
X=as.matrix(read.csv(file="data-files/x_250069.csv",header = F))
colnames(X)<-c("Temperature", "Vacuum", "Pressure", "Humidity")
head(X)
```

```
##      Temperature Vacuum Pressure Humidity
## [1,]        8.34  40.77  1010.84    90.01
## [2,]       23.64  58.49  1011.40    74.20
## [3,]       29.74  56.90  1007.15    41.91
## [4,]       19.07  49.69  1007.22    76.79
## [5,]       11.80  40.66  1017.13    97.20
## [6,]       13.97  39.16  1016.05    84.60
```

Importing output data from csv file and displaying first few rows

```
Y=as.matrix(read.csv(file="data-files/y_250069.csv",header = F))
colnames(Y)<-c("Energy_Output")
head(Y)
```

```
##      Energy_Output
## [1,]        480.48
## [2,]        445.75
## [3,]        438.76
## [4,]        453.09
## [5,]        464.43
## [6,]        470.96
```

Importing time series data from csv file and displaying first few rows

```
time = read.csv(file="data-files/t_250069.csv", header = F, skip = 1)
time = as.matrix(rbind(0, time))
head(time)
```

```
##      V1
## [1,]  0
## [2,]  1
## [3,]  2
## [4,]  3
## [5,]  4
## [6,]  5
```

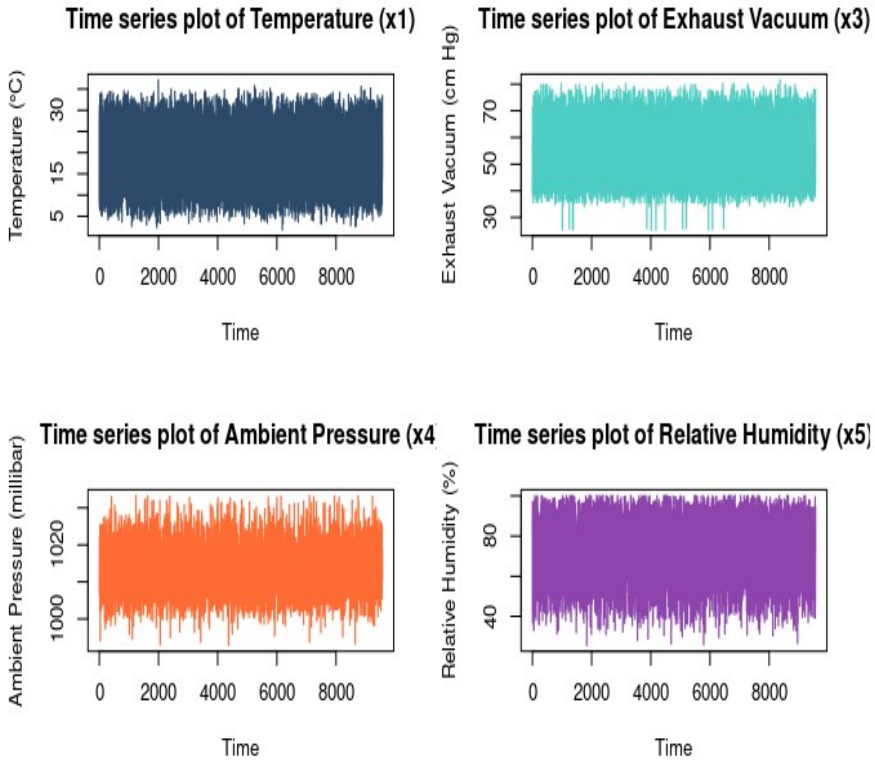
Task 1.1: Time series plots (of input and output variables)

Creating a time series object for each variable

```
X_ts <- ts(X, start = c(min(time)), frequency = 1)
Y_ts <- ts(Y, start = c(min(time)), frequency = 1)
```

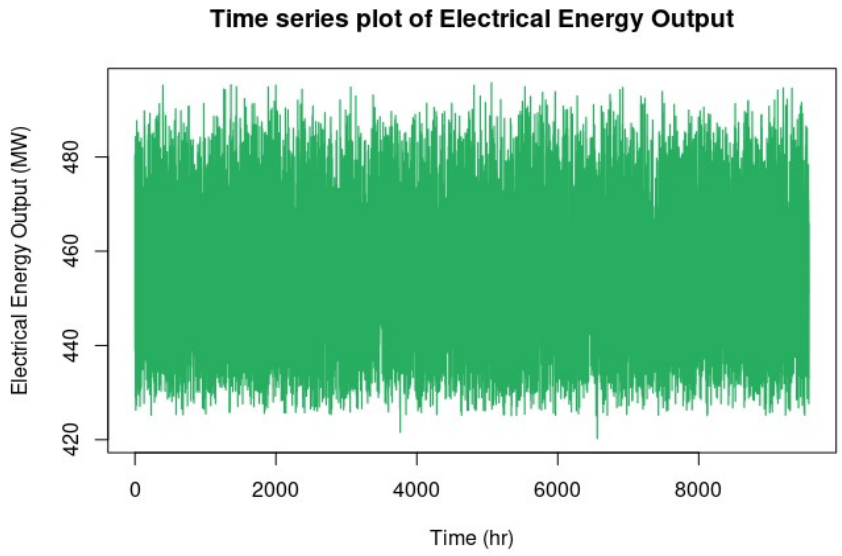
Plotting time series of input variables

```
par(mfrow=c(2,2))
plot(X_ts[,1], main = "Time series plot of Temperature (x1)",
     xlab = "Time", ylab = "Temperature (°C)", col = colors$primary, lwd = 1.2)
plot(X_ts[,2], main = "Time series plot of Exhaust Vacuum (x3)",
     xlab = "Time", ylab = "Exhaust Vacuum (cm Hg)", col = colors$accent, lwd = 1.2)
plot(X_ts[,3], main = "Time series plot of Ambient Pressure (x4)",
     xlab = "Time", ylab = "Ambient Pressure (millibar)", col = colors$secondary, lwd = 1.2)
plot(X_ts[,4], main = "Time series plot of Relative Humidity (x5)",
     xlab = "Time", ylab = "Relative Humidity (%)", col = colors$purple, lwd = 1.2)
```



Plotting time series for output variable

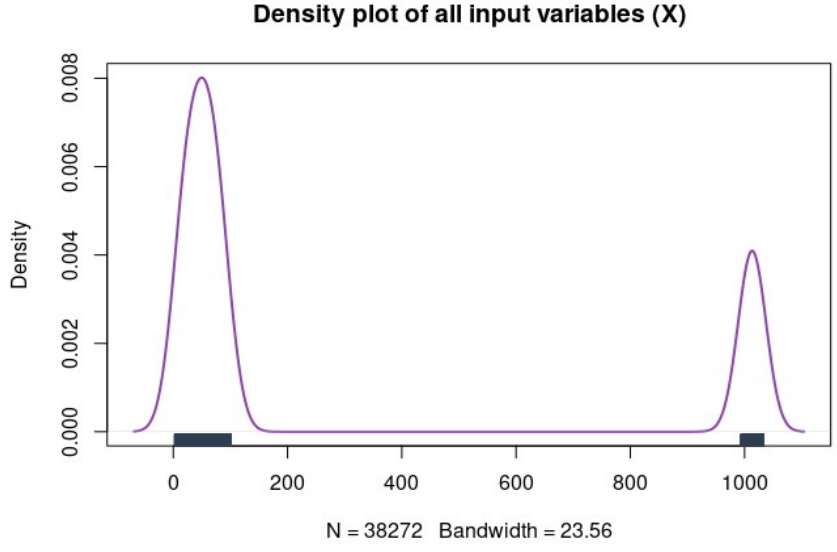
```
plot(Y_ts, main = "Time series plot of Electrical Energy Output",
    xlab = "Time (hr)", ylab = "Electrical Energy Output (MW)",
    col = colors$success, lwd = 1.2)
```



Task 1.2: Distribution plot for each variable

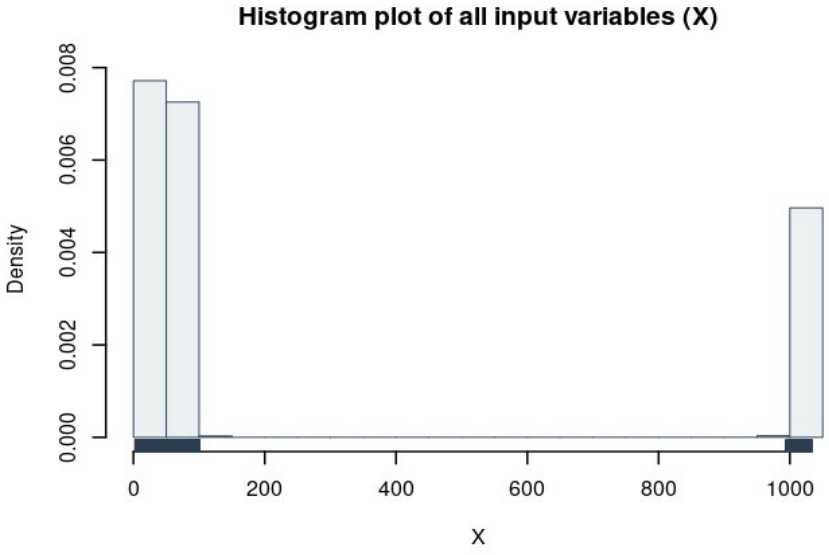
Density plot of all input variables (X)

```
density_X = density(X)
plot(density_X, main = "Density plot of all input variables (X)",
     col = colors$purple, lwd = 2)
rug(jitter(X), col = colors$dark, lwd = 0.8)
```



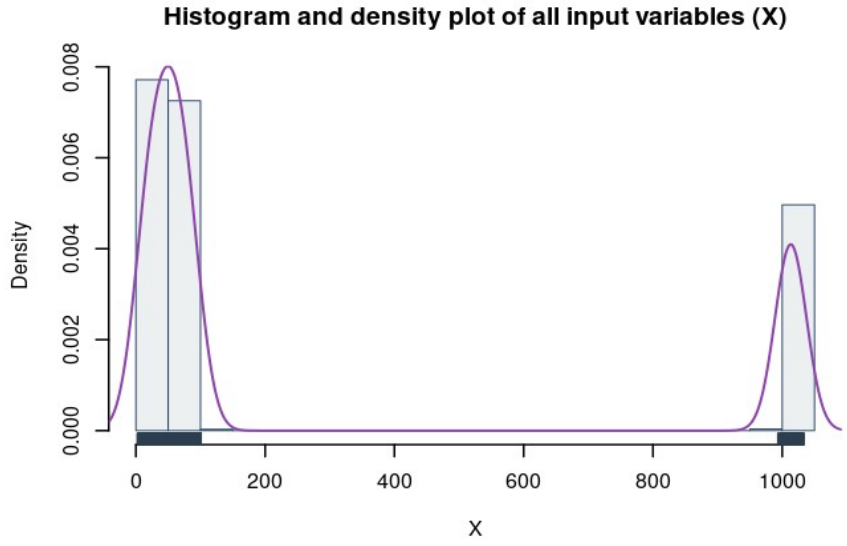
Histogram plot of all input variables (X)

```
hist(X, freq = FALSE, main = "Histogram plot of all input variables (X)",
     col = colors$light, border = colors$primary, lwd = 1.5)
rug(jitter(X), col = colors$dark, lwd = 0.8)
```



Combine histogram and density plot of variable X

```
hist(X, freq = FALSE, main = "Histogram and density plot of all input variables (X)",
     col = colors$light, border = colors$primary, lwd = 1.5)
lines(density_X, main = "Density plot of input variable X",
      col = colors$purple, lwd = 2)
rug(jitter(X), col = colors$dark, lwd = 0.8)
```

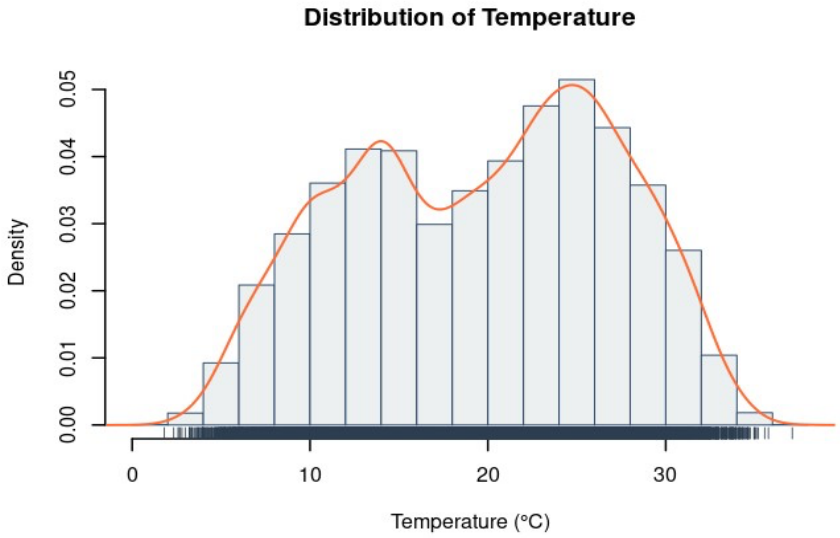


Histograms with density plots for all input variables

```
par(mfrow=c(2,2))
```

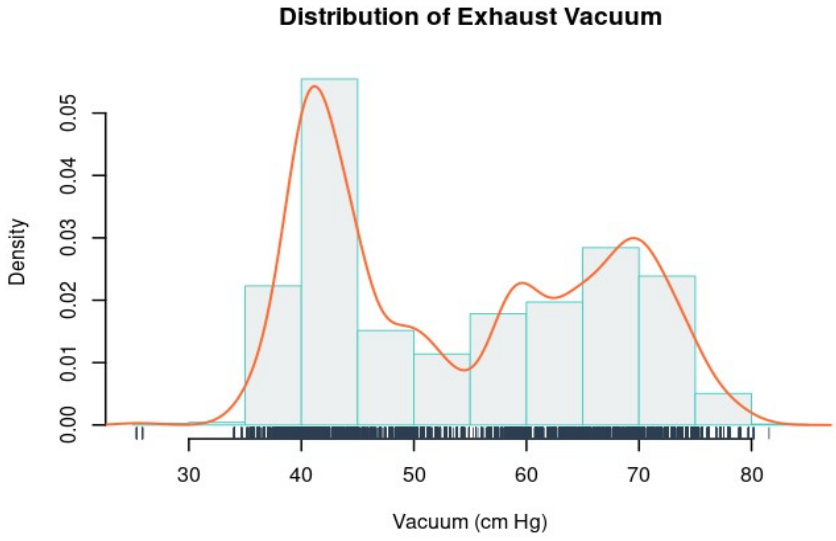
Histogram with density plot for Temperature

```
hist(X[,"Temperature"], freq = FALSE, main = "Distribution of Temperature",
     xlab = "Temperature (°C)", col = colors$light, border = colors$primary, lwd = 1.5)
lines(density(X[,"Temperature"]), lwd = 2, col = colors$secondary)
rug(jitter(X[,"Temperature"]), col = colors$dark, lwd = 0.8)
```



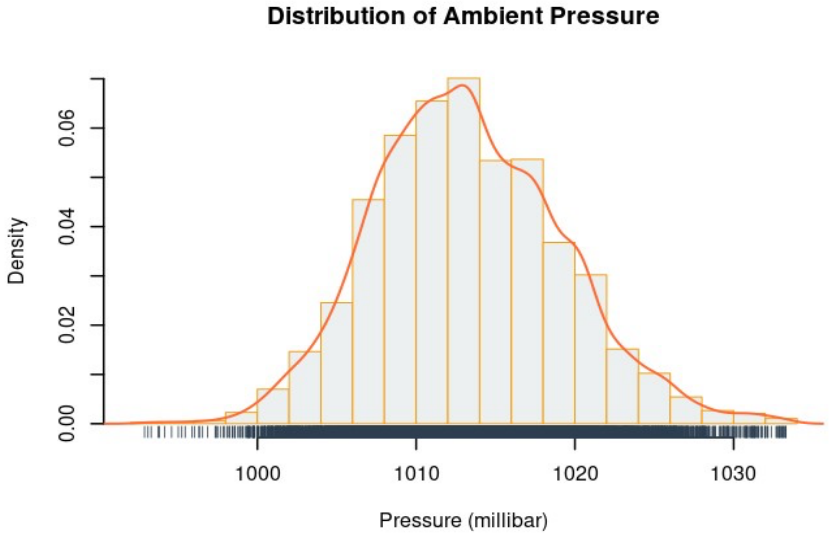
Histogram with density plot for Exhaust Vacuum

```
hist(X[,"Vacuum"], freq = FALSE, main = "Distribution of Exhaust Vacuum",
     xlab = "Vacuum (cm Hg)", col = colors$light, border = colors$accent, lwd = 1.5)
lines(density(X[,"Vacuum"]), lwd = 2, col = colors$secondary)
rug(jitter(X[,"Vacuum"]), col = colors$dark, lwd = 0.8)
```



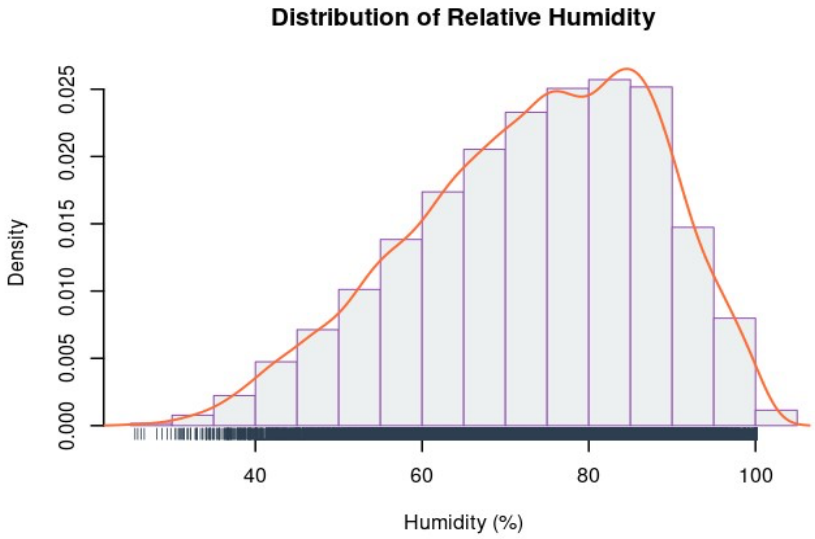
Histogram with density plot for Ambient Pressure

```
hist(X[,"Pressure"], freq = FALSE, main = "Distribution of Ambient Pressure",
     xlab = "Pressure (millibar)", col = colors$light, border = colors$warning, lwd = 1.5)
lines(density(X[,"Pressure"]), lwd = 2, col = colors$secondary)
rug(jitter(X[,"Pressure"]), col = colors$dark, lwd = 0.8)
```



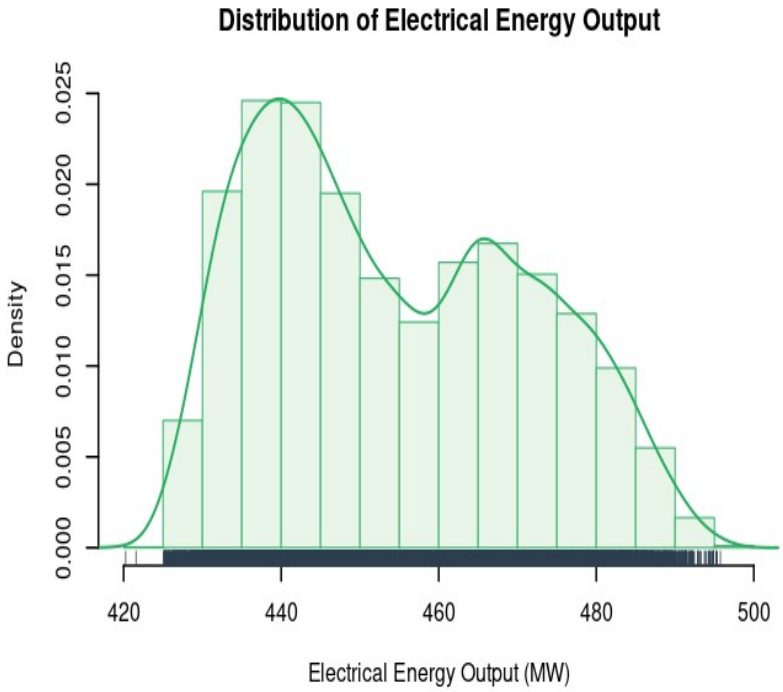
Histogram with density plot for Relative Humidity

```
hist(X[,"Humidity"], freq = FALSE, main = "Distribution of Relative Humidity",
     xlab = "Humidity (%)", col = colors$light, border = colors$purple, lwd = 1.5)
lines(density(X[,"Humidity"]), lwd = 2, col = colors$secondary)
rug(jitter(X[,"Humidity"]), col = colors$dark, lwd = 0.8)
```



Histogram with density plot for output variable

```
par(mfrow=c(1,1))
hist(Y, freq = FALSE, main = "Distribution of Electrical Energy Output",
     xlab = "Electrical Energy Output (MW)",
     col = "#E8F5E8", border = colors$success, lwd = 1.5)
lines(density(Y), lwd = 2, col = colors$success)
rug(jitter(Y), col = colors$dark, lwd = 0.8)
```



Task 1.3: Correlation and scatter plots

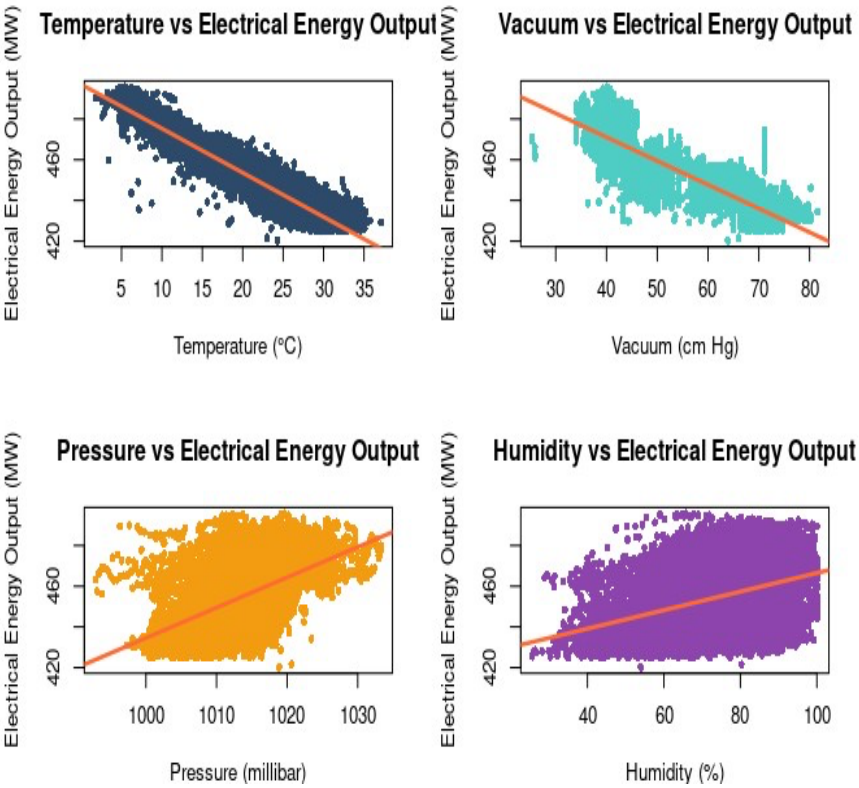
Plotting the scatter plots between input variables and output

```
par(mfrow=c(2,2))
plot(X[,"Temperature"], Y, main = "Temperature vs Electrical Energy Output",
     xlab = "Temperature (°C)", ylab = "Electrical Energy Output (MW)",
     pch = 16, cex = 0.8, col = colors$primary)
abline(lm(Y ~ X[,"Temperature"]), col = colors$secondary, lwd = 2.5)

plot(X[,"Vacuum"], Y, main = "Vacuum vs Electrical Energy Output",
     xlab = "Vacuum (cm Hg)", ylab = "Electrical Energy Output (MW)",
     pch = 16, cex = 0.8, col = colors$accent)
abline(lm(Y ~ X[,"Vacuum"]), col = colors$secondary, lwd = 2.5)

plot(X[,"Pressure"], Y, main = "Pressure vs Electrical Energy Output",
     xlab = "Pressure (millibar)", ylab = "Electrical Energy Output (MW)",
     pch = 16, cex = 0.8, col = colors$warning)
abline(lm(Y ~ X[,"Pressure"]), col = colors$secondary, lwd = 2.5)

plot(X[,"Humidity"], Y, main = "Humidity vs Electrical Energy Output",
     xlab = "Humidity (%)", ylab = "Electrical Energy Output (MW)",
     pch = 16, cex = 0.8, col = colors$purple)
abline(lm(Y ~ X[,"Humidity"]), col = colors$secondary, lwd = 2.5)
```



Task 2: Regression – modelling the relationship between electrical energy output

Creating a column of ones for the bias term

```
ones = matrix(1, nrow(X), 1)
head(ones)
```

```
##      [,1]
## [1,]    1
## [2,]    1
## [3,]    1
## [4,]    1
## [5,]    1
## [6,]    1
```

Task 2.1: Estimate model parameters for each candidate model

Model 1: $y = \theta_1 x_4 + \theta_2 x_3^2 + \theta bias$

scaling the variables and binding data from equation of model 1

```
X_model1 <- cbind(scale(X[,"Pressure"]), scale(X[,"Vacuum"]^2), ones)
head(X_model1)
```

```
##           [,1]        [,2] [,3]
## [1,] -0.4073356 -1.02213385    1
## [2,] -0.3130402  0.21910945    1
## [3,] -1.0286750  0.08963496    1
## [4,] -1.0168880 -0.45270382    1
## [5,]  0.6518038 -1.02845500    1
## [6,]  0.4699484 -1.11294823    1
```

```
Model1_thetahat <- solve(t(X_model1) %*% X_model1) %*% t(X_model1) %*% Y
rownames(Model1_thetahat) <- c("θ₁", "θ₂", "θbias")
```

Printing values for Model 1

```
print(Model1_thetahat)
```

```
##       Energy_Output
## θ₁         3.348278
## θ₂       -13.211452
## θbias    454.365009
```

```
t(Model1_thetahat)
```

```
##                    θ₁        θ₂   θbias
## Energy_Output 3.348278 -13.21145 454.365
```

Model 2: $y = \theta_1 x_4 + \theta_2 x_3^2 + \theta_3 x_5 + \theta bias$

scaling the variables and binding data from equation of model 2

```
X_model2 <- cbind(scale(X[,"Pressure"]), scale(X[,"Vacuum"]^2), scale(X[,"Humidity"]), ones)
Model2_thetahat <- solve(t(X_model2) %*% X_model2) %*% t(X_model2) %*% Y
rownames(Model2_thetahat) <- c("θ₁", "θ₂", "θ₃", "θbias")
```

Printing values for Model 2

```
print(Model2_thetahat)
```

```
##       Energy_Output
## θ₁         3.432657
## θ₂       -12.406622
## θ₃         2.517326
## θbias    454.365009
```

```
t(Model2_thetahat)
```

```
##                    θ₁        θ₂       θ₃   θbias
## Energy_Output 3.432657 -12.40662 2.517326 454.365
```

Model 3: $y = \theta_1 x_3 + \theta_2 x_4 + \theta_3 x_5^3$

scaling the variables and binding data from equation of model 3

```
X_model3 <- cbind(scale(X[,"Vacuum"]), scale(X[,"Pressure"]), scale(X[,"Humidity"]^3))
Model3_thetahat <- solve(t(X_model3) %*% X_model3) %*% t(X_model3) %*% Y
rownames(Model3_thetahat) <- c("θ₁", "θ₂", "θ₃")
```

Printing values for Model 3

```
print(Model3_thetahat)
```

```
##    Energy_Output
## θ₁    -12.722943
## θ₂      3.402683
## θ₃      2.315840
```

```
t(Model3_thetahat)
```

```
##                     θ₁       θ₂      θ₃
## Energy_Output -12.72294 3.402683 2.31584
```

Model 4: $y = \theta_1 x_4 + \theta_2 x_3^2 + \theta_3 x_5^3 + \theta bias$

scaling the variables and binding data from equation of model 4

```
X_model4 <- cbind(scale(X[,"Pressure"]), scale(X[,"Vacuum"]^2), scale(X[,"Humidity"]^3), ones)
```

Calculating theta hat for Model 4

```
Model4_thetahat <- solve(t(X_model4) %*% X_model4) %*% t(X_model4) %*% Y
rownames(Model4_thetahat) <- c("θ₁", "θ₂", "θ₃", "θbias")
```

Printing values for Model 4

```
print(Model4_thetahat)
```

```
##       Energy_Output
## θ₁         3.487220
## θ₂       -12.402038
## θ₃         2.487015
## θbias    454.365009
```

```
t(Model4_thetahat)
```

```
##                    θ₁        θ₂       θ₃   θbias
## Energy_Output 3.48722 -12.40204 2.487015 454.365
```

Model 5: $y = \theta_1 x_4 + \theta_2 x_1^2 + \theta_3 x_3^2 + \theta bias$

scaling the variables and binding data from equation of model 5

```
X_model5 <- cbind(scale(X[,"Pressure"]), scale(X[,"Temperature"]^2), scale(X[,"Vacuum"]^2), ones)
Model5_thetahat <- solve(t(X_model5) %*% X_model5) %*% t(X_model5) %*% Y
rownames(Model5_thetahat) <- c("θ₁", "θ₂", "θ₃", "θbias")
```

Printing values for Model 5

```
print(Model5_thetahat)
```

```
##       Energy_Output
## θ₁         1.349107
## θ₂       -10.605331
## θ₃        -5.173124
## θbias    454.365009
```

```
t(Model5_thetahat)
```

```
##                     θ₁        θ₂        θ₃   θbias
## Energy_Output 1.349107 -10.60533 -5.173124 454.365
```

Task 2.2: Calculating the RSS value for each candidate model

Calculating predicted values and RSS for Model 1

```
Y_hat_model1 <- X_model1 %*% Model1_thetahat
RSS_Model_1 <- sum((Y - Y_hat_model1)^2)
```

Calculating predicted values and RSS for Model 2

```
Y_hat_model2 <- X_model2 %*% Model2_thetahat
RSS_Model_2 <- sum((Y - Y_hat_model2)^2)
```

Calculating predicted values and RSS for Model 3

```
Y_hat_model3 <- X_model3 %*% Model3_thetahat
RSS_Model_3 <- sum((Y - Y_hat_model3)^2)
```

Calculating predicted values and RSS for Model 4

```
Y_hat_model4 <- X_model4 %*% Model4_thetahat
RSS_Model_4 <- sum((Y - Y_hat_model4)^2)
```

Calculating predicted values and RSS for Model 5

```
Y_hat_model5 <- X_model5 %*% Model5_thetahat
RSS_Model_5 <- sum((Y - Y_hat_model5)^2)
```

Creating a data frame of RSS values for comparison

```
RSS_values <- data.frame(
  Model = c("Model 1", "Model 2", "Model 3", "Model 4", "Model 5"),
  RSS = c(RSS_Model_1, RSS_Model_2, RSS_Model_3, RSS_Model_4, RSS_Model_5)
)
```

Printing Residual Sum of Squares (RSS) for All Models

```
print(RSS_values)
```

```
##     Model          RSS
## 1 Model 1     657248.2
## 2 Model 2     602347.1
## 3 Model 3 1975837762.7
## 4 Model 4     603630.7
## 5 Model 5     365625.0
```

Task 2.3: Calculating log-likelihood and variance

Total number of observations

```
N <- nrow(Y)
```

Calculating variance and log-likelihood for Model 1

```
Variance_model1 <- RSS_Model_1 / (N - 1)
likelihood_Model_1 <- -(N/2) * log(2*pi) - (N/2) * log(Variance_model1) -
                      (1/(2*Variance_model1)) * RSS_Model_1
```

Calculating variance and log-likelihood for Model 2

```
Variance_model2 <- RSS_Model_2 / (N - 1)
likelihood_Model_2 <- -(N/2) * log(2*pi) - (N/2) * log(Variance_model2) -
                      (1/(2*Variance_model2)) * RSS_Model_2
```

Calculating variance and log-likelihood for Model 3

```
Variance_model3 <- RSS_Model_3 / (N - 1)
likelihood_Model_3 <- -(N/2) * log(2*pi) - (N/2) * log(Variance_model3) -
                      (1/(2*Variance_model3)) * RSS_Model_3
```

Calculating variance and log-likelihood for Model 4

```
Variance_model4 <- RSS_Model_4 / (N - 1)
likelihood_Model_4 <- -(N/2) * log(2*pi) - (N/2) * log(Variance_model4) -
                      (1/(2*Variance_model4)) * RSS_Model_4
```

Calculating variance and log-likelihood for Model 5

```
Variance_model5 <- RSS_Model_5 / (N - 1)
likelihood_Model_5 <- -(N/2) * log(2*pi) - (N/2) * log(Variance_model5) -
                      (1/(2*Variance_model5)) * RSS_Model_5
```

Creating data frames for variance and likelihood values

```
variance_values <- data.frame(
  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_values <- data.frame(
  Model = c("Model 1", "Model 2", "Model 3", "Model 4", "Model 5"),
  LogLikelihood = c(likelihood_Model_1, likelihood_Model_2, likelihood_Model_3,
                    likelihood_Model_4, likelihood_Model_5)
)
```

Printing Variance Values for All Models

```
print(variance_values)
```

```
##     Model     Variance
## 1 Model 1     68.69951
## 2 Model 2     62.96091
## 3 Model 3 206526.36800
## 4 Model 4     63.09509
## 5 Model 5     38.21731
```

Printing Log-Likelihood Values for All Models

```
print(likelihood_values)
```

```
##     Model LogLikelihood
## 1 Model 1     -33810.99
## 2 Model 2     -33393.69
## 3 Model 3     -72123.37
## 4 Model 4     -33403.88
## 5 Model 5     -31005.40
```

Task 2.4: Compute AIC and BIC for every candidate model

Calculating the number of parameters (K) for each model

```
K_model1 <- length(Model1_thetahat)
K_model2 <- length(Model2_thetahat)
K_model3 <- length(Model3_thetahat)
K_model4 <- length(Model4_thetahat)
K_model5 <- length(Model5_thetahat)
```

Calculating AIC for each model

```
AIC_model1 <- 2 * K_model1 - 2 * likelihood_Model_1
AIC_model2 <- 2 * K_model2 - 2 * likelihood_Model_2
AIC_model3 <- 2 * K_model3 - 2 * likelihood_Model_3
AIC_model4 <- 2 * K_model4 - 2 * likelihood_Model_4
AIC_model5 <- 2 * K_model5 - 2 * likelihood_Model_5
```

Calculating BIC for each model

```
BIC_model1 <- K_model1 * log(N) - 2 * likelihood_Model_1
BIC_model2 <- K_model2 * log(N) - 2 * likelihood_Model_2
BIC_model3 <- K_model3 * log(N) - 2 * likelihood_Model_3
BIC_model4 <- K_model4 * log(N) - 2 * likelihood_Model_4
BIC_model5 <- K_model5 * log(N) - 2 * likelihood_Model_5
```

Creating data frames for K, AIC, and BIC values

```
parameter_counts <- data.frame(
  Model = c("Model 1", "Model 2", "Model 3", "Model 4", "Model 5"),
  K = c(K_model1, K_model2, K_model3, K_model4, K_model5)
)

AIC_values <- data.frame(
  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_values <- data.frame(
  Model = c("Model 1", "Model 2", "Model 3", "Model 4", "Model 5"),
  BIC = c(BIC_model1, BIC_model2, BIC_model3, BIC_model4, BIC_model5)
)
```

Printing Number of Parameters (K) for Each Model

```
print(parameter_counts)
```

```
##     Model K
## 1 Model 1 3
## 2 Model 2 4
## 3 Model 3 3
## 4 Model 4 4
## 5 Model 5 4
```

Printing AIC Values for All Models

```
print(AIC_values)
```

```
##     Model       AIC
## 1 Model 1  67627.98
## 2 Model 2  66795.38
## 3 Model 3 144252.75
## 4 Model 4  66815.75
## 5 Model 5  62018.79
```

Printing BIC Values for All Models

```
print(BIC_values)
```

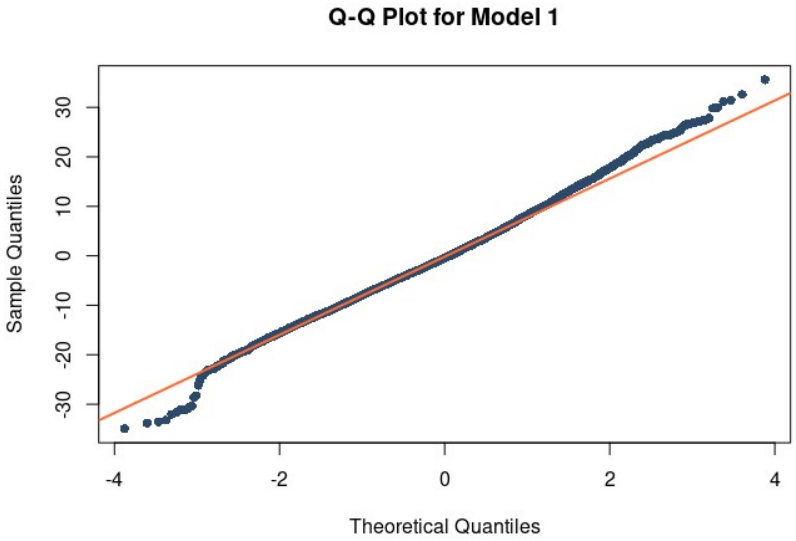
```
##     Model       BIC
## 1 Model 1  67649.48
## 2 Model 2  66824.05
## 3 Model 3 144274.24
## 4 Model 4  66844.42
## 5 Model 5  62047.46
```

Task 2.5: Checking the distribution of model prediction errors (residuals)

```
par(mfrow=c(2,3))
```

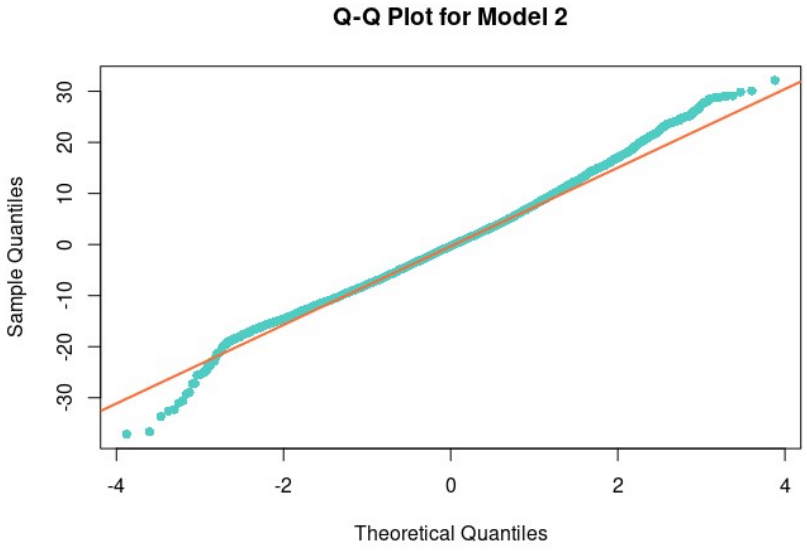
Calculating and plotting residuals for Model 1

```
model1_error <- Y - Y_hat_model1
qqnorm(model1_error, main = "Q-Q Plot for Model 1", col = colors$primary, pch = 16)
qqline(model1_error, col = colors$secondary, lwd = 2)
```



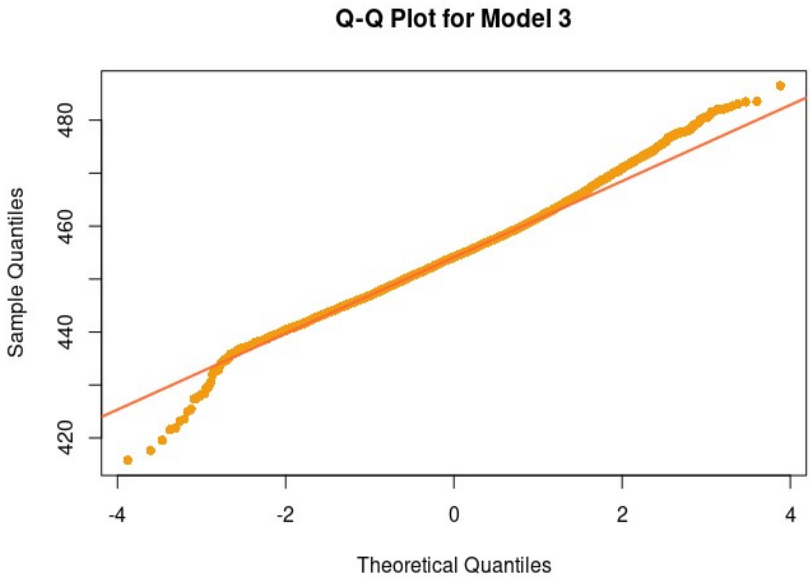
Calculating and plotting residuals for Model 2

```
model2_error <- Y - Y_hat_model2
qqnorm(model2_error, main = "Q-Q Plot for Model 2", col = colors$accent, pch = 16)
qqline(model2_error, col = colors$secondary, lwd = 2)
```



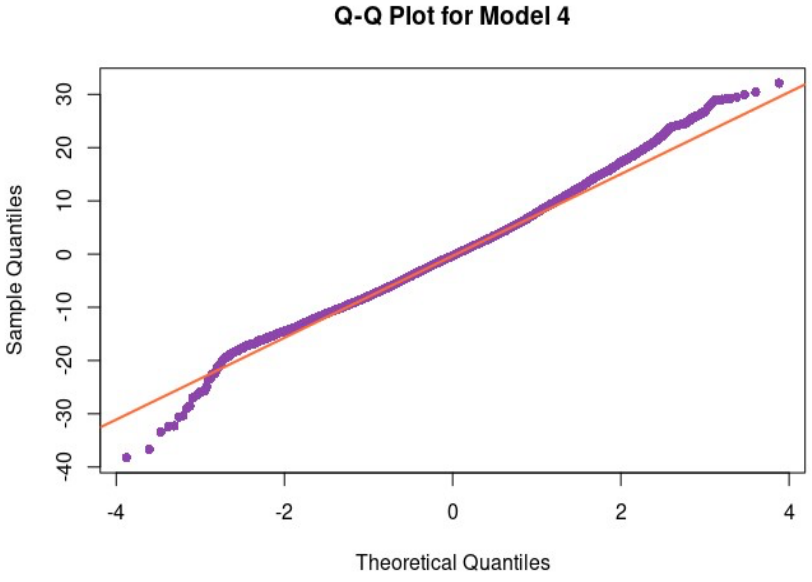
Calculating and plotting residuals for Model 3

```
model3_error <- Y - Y_hat_model3
qqnorm(model3_error, main = "Q-Q Plot for Model 3", col = colors$warning, pch = 16)
qqline(model3_error, col = colors$secondary, lwd = 2)
```



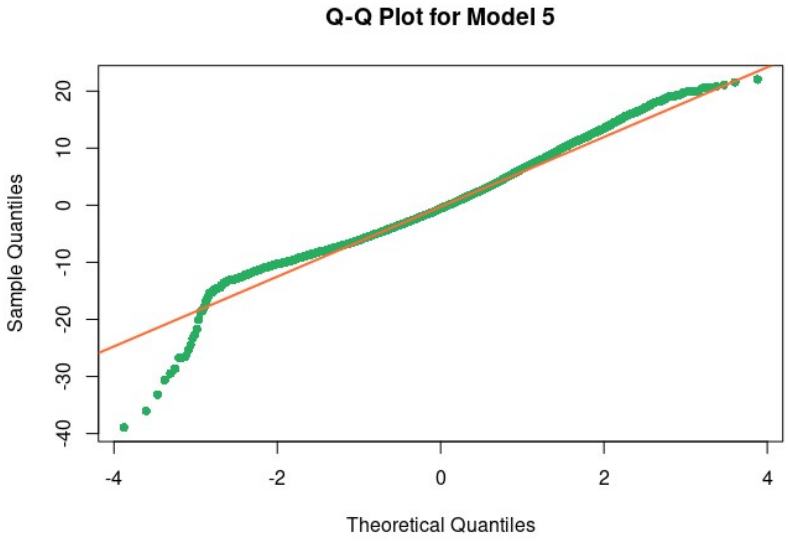
Calculating and plotting residuals for Model 4

```
model4_error <- Y - Y_hat_model4
qqnorm(model4_error, main = "Q-Q Plot for Model 4", col = colors$purple, pch = 16)
qqline(model4_error, col = colors$secondary, lwd = 2)
```



Calculating and plotting residuals for Model 5

```
model5_error <- Y - Y_hat_model5
qqnorm(model5_error, main = "Q-Q Plot for Model 5", col = colors$success, pch = 16)
qqline(model5_error, col = colors$secondary, lwd = 2)
```



Task 2.6: Selecting the best regression model

Resetting plot layout

```
par(mfrow=c(1,1))
```

Combining all evaluation metrics for comparison

```
model_comparison <- data.frame(
  Model = c("Model 1", "Model 2", "Model 3", "Model 4", "Model 5"),
  RSS = c(RSS_Model_1, RSS_Model_2, RSS_Model_3, RSS_Model_4, RSS_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)
)
```

Finding the best model based on each criterion

```
best_RSS_model <- model_comparison$Model[which.min(model_comparison$RSS)]
best_AIC_model <- model_comparison$Model[which.min(model_comparison$AIC)]
best_BIC_model <- model_comparison$Model[which.min(model_comparison$BIC)]
```

Printing the comparison values

```
print(model_comparison)
```

```
##     Model          RSS       AIC       BIC
## 1 Model 1     657248.2  67627.98  67649.48
## 2 Model 2     602347.1  66795.38  66824.05
## 3 Model 3 1975837762.7 144252.75 144274.24
## 4 Model 4     603630.7  66815.75  66844.42
## 5 Model 5     365625.0  62018.79  62047.46
```

Task 2.7: Training/Testing Split and Confidence Intervals

Splitting data together to maintain correspondence

```
set.seed(123)
combined_data <- data.frame(X, Y)
split_indices <- initial_split(combined_data, prop = 0.7)
training_set <- training(split_indices)
testing_set <- testing(split_indices)
```

Extracting training and testing data

```
X_train <- as.matrix(training_set[, c("Temperature", "Vacuum", "Pressure", "Humidity")])
Y_train <- as.matrix(training_set[, "Energy_Output"])
X_test <- as.matrix(testing_set[, c("Temperature", "Vacuum", "Pressure", "Humidity")])
Y_test <- as.matrix(testing_set[, "Energy_Output"])
```

Model 5: $y = \theta_1 x_4 + \theta_2 x_1^2 + \theta_3 x_3^2 + \theta bias$ (Pressure + Temperature² + Vacuum²)

```
ones_train <- matrix(1, nrow(X_train), 1)
X_train_model <- cbind(scale(X_train[, "Pressure"]),
                       scale(X_train[, "Temperature"]^2),
                       scale(X_train[, "Vacuum"]^2),
                       ones_train)
```

Estimating parameters

```
train_thetahat <- solve(t(X_train_model) %*% X_train_model) %*% t(X_train_model) %*% Y_train
```

Preparing test data

```
ones_test <- matrix(1, nrow(X_test), 1)
X_test_model <- cbind(scale(X_test[, "Pressure"]),
                      scale(X_test[, "Temperature"]^2),
                      scale(X_test[, "Vacuum"]^2),
                      ones_test)
```

Making predictions

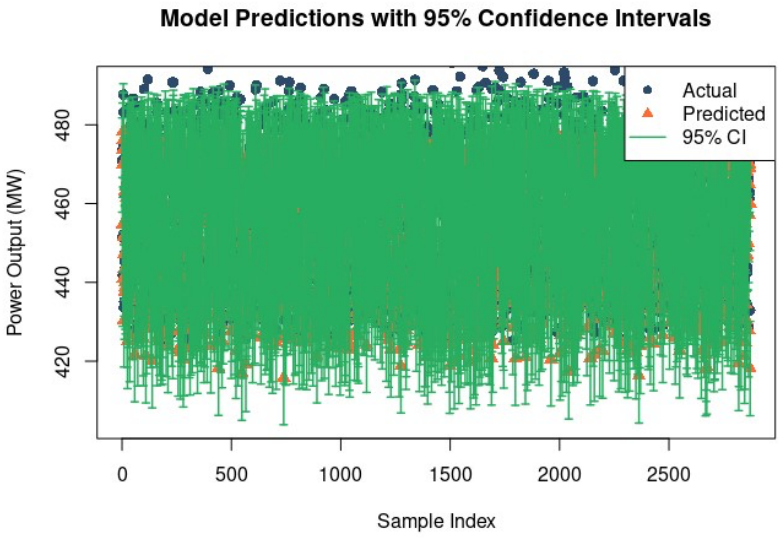
```
Y_test_pred <- X_test_model %*% train_thetahat
```

Proper confidence intervals

```
residuals <- Y_test - Y_test_pred
std_error <- sqrt(sum(residuals^2) / (nrow(Y_test) - ncol(X_test_model)))
t_value <- qt(0.975, df = nrow(Y_test) - ncol(X_test_model))
lower_CI <- Y_test_pred - t_value * std_error
upper_CI <- Y_test_pred + t_value * std_error
```

Plotting with error bars

```
plot(1:length(Y_test), Y_test, pch = 16, col = colors$primary,
     ylim = c(min(lower_CI), max(upper_CI)), cex = 1.2,
     xlab = "Sample Index", ylab = "Power Output (MW)",
     main = "Model Predictions with 95% Confidence Intervals")
points(1:length(Y_test), Y_test_pred, pch = 17, col = colors$secondary, cex = 1.2)
arrows(1:length(Y_test), lower_CI, 1:length(Y_test), upper_CI,
       length = 0.03, angle = 90, code = 3, col = colors$success, lwd = 1.5)
legend("topright",
       legend = c("Actual", "Predicted", "95% CI"),
       col = c(colors$primary, colors$secondary, colors$success),
       pch = c(16, 17, NA), lty = c(NA, NA, 1), lwd = c(NA, NA, 1.5))
```




```
# Performance metrics
print(paste("RSS:", sum(residuals^2)))
```

```
## [1] "RSS: 106715.324826487"
```

```
print(paste("MAE:", mean(abs(residuals))))
```

```
## [1] "MAE: 4.858553634188"
```

```
print(paste("RMSE:", sqrt(mean(residuals^2))))
```

```
## [1] "RMSE: 6.09672770909037"
```

Task 3: Approximate Bayesian Computation (ABC)

From Task 2.6, we determined Model 5 is the best performing model

Model 5: $y = \theta_1 x_4 + \theta_2 x_1^2 + \theta_3 x_3^2 + \theta bias$

Identifying the parameters with largest absolute values

```
print("Model 5 Parameter Estimates:")
```

```
## [1] "Model 5 Parameter Estimates:"
```

```
print(Model5_thetahat)
```

```
##       Energy_Output
## θ₁         1.349107
## θ₂       -10.605331
## θ₃        -5.173124
## θbias    454.365009
```

Calculating absolute values of all parameters

```
param_abs_values <- abs(Model5_thetahat)
print("Absolute values of parameters:")
```

```
## [1] "Absolute values of parameters:"
```

```
print(param_abs_values)
```

```
##       Energy_Output
## θ₁         1.349107
## θ₂        10.605331
## θ₃         5.173124
## θbias    454.365009
```

Finding the two largest absolute values

```
sorted_indices <- order(param_abs_values, decreasing = TRUE)
top_two_indices <- sorted_indices[1:2]
top_two_params <- Model5_thetahat[top_two_indices]
param_names <- rownames(Model5_thetahat)[top_two_indices]

print("Two parameters with largest absolute values:")
```

```
## [1] "Two parameters with largest absolute values:"
```

```
for(i in 1:2) {
  print(paste(param_names[i], "=", top_two_params[i]))
}
```

```
## [1] "θbias = 454.365009406354"
## [1] "θ₂ = -10.6053309550988"
```

Setting up uniform priors around the estimated parameter values

We'll use a range of ±100% of the absolute value of each parameter

```
prior_ranges <- matrix(0, 2, 2)
scaling_factor <- 1  # 100% of the parameter value

for(i in 1:2) {
  param_value <- top_two_params[i]
  range_width <- scaling_factor * abs(param_value)
  prior_ranges[i, 1] <- param_value - range_width  # Lower bound
  prior_ranges[i, 2] <- param_value + range_width  # Upper bound
}

rownames(prior_ranges) <- param_names
colnames(prior_ranges) <- c("Lower_Bound", "Upper_Bound")

print("Prior Distribution Ranges:")
```

```
## [1] "Prior Distribution Ranges:"
```

```
print(prior_ranges)
```

```
##       Lower_Bound Upper_Bound
## θbias     0.00000      908.73
## θ₂      -21.21066        0.00
```

Function to generate samples from uniform prior distributions

```
generate_prior_sample <- function() {
  c(
    runif(1, prior_ranges[1, 1], prior_ranges[1, 2]),
    runif(1, prior_ranges[2, 1], prior_ranges[2, 2])
  )
}
```

Creating fixed parameters (the ones we're not varying)

```
fixed_params <- Model5_thetahat
fixed_indices <- setdiff(1:length(Model5_thetahat), top_two_indices)

print("Fixed parameters:")
```

```
## [1] "Fixed parameters:"
```

```
print(fixed_params[fixed_indices])
```

```
## [1]  1.349107 -5.173124
```

Calculating RSS for a given set of parameters

```
calculate_RSS <- function(params) {
  # Creating a complete parameter set
  full_params <- fixed_params
  full_params[top_two_indices] <- params

  # Calculate predictions using Model 5 structure
  X_model5_abc <- cbind(scale(X[,"Pressure"]),
                        scale(X[,"Temperature"]^2),
                        scale(X[,"Vacuum"]^2),
                        ones)
  Y_pred <- X_model5_abc %*% full_params

  # Calculate RSS
  sum((Y - Y_pred)^2)
}
```

Getting the RSS of our best model as reference

```
best_model_RSS <- RSS_Model_5
print(paste("Best model RSS:", best_model_RSS))
```

```
## [1] "Best model RSS: 365624.974504425"
```

Getting the RSS of our best model as reference

```
best_model_RSS <- RSS_Model_5
print(paste("Best model RSS:", best_model_RSS))
```

```
## [1] "Best model RSS: 365624.974504425"
```

Setting up rejection ABC parameters

```
num_samples <- 10000  # Number of samples to generate
accepted_samples <- matrix(0, 0, 2)  # Matrix to store accepted samples
colnames(accepted_samples) <- param_names
```

Setting threshold as 5 times the best model RSS (500% tolerance)

```
threshold <- best_model_RSS * 5.0
print(paste("ABC threshold:", threshold))
```

```
## [1] "ABC threshold: 1828124.87252213"
```

Performing rejection ABC with progress monitoring

```
cat("Performing rejection ABC sampling...\n")
```

```
## Performing rejection ABC sampling...
```

```
for(i in 1:num_samples) {
  # Generating sample from prior
  sample_params <- generate_prior_sample()

  # Calculating RSS for this parameter combination
  sample_RSS <- calculate_RSS(sample_params)

  # Accept or reject based on threshold
  if(sample_RSS <= threshold) {
    accepted_samples <- rbind(accepted_samples, sample_params)
  }

  # Progress indicator every 2000 samples
  if(i %% 2000 == 0) {
    current_rate <- round(nrow(accepted_samples)/i * 100, 2)
    cat("Processed", i, "samples, accepted", nrow(accepted_samples),
        "- Rate:", current_rate, "%\n")
  }
}
```

```
## Processed 2000 samples, accepted 44 - Rate: 2.2 %
## Processed 4000 samples, accepted 88 - Rate: 2.2 %
## Processed 6000 samples, accepted 140 - Rate: 2.33 %
## Processed 8000 samples, accepted 181 - Rate: 2.26 %
## Processed 10000 samples, accepted 228 - Rate: 2.28 %
```

```
print(paste("Total accepted samples:", nrow(accepted_samples)))
```

```
## [1] "Total accepted samples: 228"
```

```
print(paste("Acceptance rate:", round(nrow(accepted_samples)/num_samples * 100, 2), "%"))
```

```
## [1] "Acceptance rate: 2.28 %"
```

Plotting joint and marginal posterior distributions with error handling

```
if(nrow(accepted_samples) >= 10) {

  # Clear any existing plot device issues
  tryCatch({
    # Set up plotting area
    par(mfrow=c(2,2))

    # Joint posterior distribution
    plot(accepted_samples[, 1], accepted_samples[, 2],
         pch = 16, cex = 0.8, col = colors$primary,
         xlab = param_names[1], ylab = param_names[2],
         main = "Joint Posterior Distribution")

    # Adding lines showing the point estimates
    abline(v = top_two_params[1], col = colors$secondary, lty = 2, lwd = 2)
    abline(h = top_two_params[2], col = colors$secondary, lty = 2, lwd = 2)

    # Adding point estimate
    points(top_two_params[1], top_two_params[2],
           col = colors$danger, pch = 16, cex = 1.8)

    legend("topright",
           legend = c("Posterior samples", "Point estimate"),
           col = c(colors$primary, colors$danger),
           pch = c(16, 16))

    # Plotting marginal posterior distribution for first parameter
    hist(accepted_samples[, 1], breaks = 30,
         col = colors$light, border = colors$primary, lwd = 1.5,
         main = paste("Marginal Posterior of", param_names[1]),
         xlab = param_names[1], ylab = "Frequency")
    abline(v = top_two_params[1], col = colors$secondary, lty = 2, lwd = 2)
    abline(v = mean(accepted_samples[, 1]), col = colors$success, lty = 2, lwd = 2)

    legend("topright",
           legend = c("Point estimate", "Posterior mean"),
           col = c(colors$secondary, colors$success),
           lty = c(2, 2), lwd = c(2, 2))
```

```
    # Marginal posterior distribution for second parameter
    hist(accepted_samples[, 2], breaks = 30,
         col = colors$light, border = colors$primary, lwd = 1.5,
         main = paste("Marginal Posterior of", param_names[2]),
         xlab = param_names[2], ylab = "Frequency")
    abline(v = top_two_params[2], col = colors$secondary, lty = 2, lwd = 2)
    abline(v = mean(accepted_samples[, 2]), col = colors$success, lty = 2, lwd = 2)

    legend("topright",
           legend = c("Point estimate", "Posterior mean"),
           col = c(colors$secondary, colors$success),
           lty = c(2, 2), lwd = c(2, 2))

  }, error = function(e) {
    cat("Plotting error occurred:", e$message, "\n")
    cat("Continuing with summary statistics...\n")
  })

  # Reset plot layout
  par(mfrow=c(1,1))

  # Calculating summary statistics for posterior distributions
  posterior_means <- colMeans(accepted_samples)
  posterior_sds <- apply(accepted_samples, 2, sd)
  posterior_medians <- apply(accepted_samples, 2, median)

  # Calculating credible intervals (Bayesian equivalent of confidence intervals)
  credible_intervals <- apply(accepted_samples, 2, function(x) {
    quantile(x, c(0.025, 0.975))
  })

  # Creating summary table
  summary_stats <- data.frame(
    Parameter = param_names,
    Point_Estimate = top_two_params,
    Posterior_Mean = posterior_means,
    Posterior_Median = posterior_medians,
    Posterior_SD = posterior_sds,
    CI_Lower = credible_intervals[1, ],
    CI_Upper = credible_intervals[2, ]
  )

  # Printing posterior distribution summary
  print(summary_stats)
} else {
  cat("Error: Too few accepted samples for meaningful analysis.\n")
  cat("Suggestions:\n")
  cat("1. Increase the threshold (try threshold <- best_model_RSS * 10.0)\n")
  cat("2. Increase the number of samples (try num_samples <- 50000)\n")
  cat("3. Widen the prior ranges (try scaling_factor <- 2.0)\n")
}
```

Code

````
---
title: '7089 CEM: Introduction to Statistical Methods for Data Science'
author:
  name: Tek Raj Bhatt
  id: '250069'
output:
  word_document: default
  html_document:
    keep_md: yes
  pdf_document: default
---
################################################################################
# Task 1: Preliminary data analysis
################################################################################

## Defining color palette
```{r}
colors <- list(
  primary = "#2E4A6B",      # Deep blue
  secondary = "#FF6B35",    # Coral orange
  accent = "#4ECDC4",       # Teal
  neutral = "#95A5A6",      # Light gray
  success = "#27AE60",      # Forest green
  warning = "#F39C12",      # Golden orange
  danger = "#E74C3C",       # Modern red
  light = "#ECF0F1",        # Very light gray
  dark = "#2C3E50",         # Dark blue-gray
  purple = "#8E44AD"        # Purple
)
```

## Reading the given dataset
```{r}
dataset <- read.csv("data-files/dataset.csv")
head(dataset)
```

## Separating the data for input variables and output electrical energy
```{r}
write.table(dataset[, 1:4], "data-files/x_250069.csv", sep = ",", row.names = FALSE, col.names = FALSE)
write.table(dataset[, ncol(dataset)], "data-files/y_250069.csv", sep = ",", row.names = FALSE, col.names = FALSE)
```

## Generating time series data with an interval of 1
### Setting the parameters
```{r}
initial_time <- 0
step <- 1
num_of_rows <- nrow(dataset)
```

### Creating time series data
```{r}
time_data <- data.frame(
    Time = seq(from = initial_time, by = step, length.out = num_of_rows)
)
```

### Saving to a CSV file
```{r}
write.table(time_data, 'data-files/t_250069.csv', row.names = FALSE, col.names = FALSE)
```
````

Installing required packages
```{r}
if (!require("matlib")) install.packages("matlib")
if (!require("rsample")) install.packages("rsample")
```

Importing required libraries
```{r}
library(ggplot2)
library(dplyr)
library(corrplot)
library(matlib)
library(rsample)
```

Importing input data from csv file and displaying first few rows
```{r}
X=as.matrix(read.csv(file="data-files/x_250069.csv",header = F))
colnames(X)<-c("Temperature", "Vacuum", "Pressure", "Humidity")
head(X)
```

Importing output data from csv file and displaying first few rows
```{r}
Y=as.matrix(read.csv(file="data-files/y_250069.csv",header = F))
colnames(Y)<-c("Energy_Output")
head(Y)
```

Importing time series data from csv file and displaying first few rows
```{r}
time = read.csv(file="data-files/t_250069.csv", header = F, skip = 1)
time = as.matrix(rbind(0, time))
head(time)
```

```
#############################################################################
# Task 1.1: Time series plots (of input and output variables)
#############################################################################
```

Creating a time series object for each variable
```{r}
X_ts <- ts(X, start = c(min(time)), frequency = 1)
Y_ts <- ts(Y, start = c(min(time)), frequency = 1)
```

Plotting time series of input variables
```{r}
par(mfrow=c(2,2))
plot(X_ts[,1], main = "Time series plot of Temperature (x1)",
     xlab = "Time", ylab = "Temperature (°C)", col = colors$primary, lwd = 1.2)
plot(X_ts[,2], main = "Time series plot of Exhaust Vacuum (x3)",
     xlab = "Time", ylab = "Exhaust Vacuum (cm Hg)", col = colors$accent, lwd = 1.2)
plot(X_ts[,3], main = "Time series plot of Ambient Pressure (x4)",
     xlab = "Time", ylab = "Ambient Pressure (millibar)", col = colors$secondary, lwd = 1.2)
plot(X_ts[,4], main = "Time series plot of Relative Humidity (x5)",
     xlab = "Time", ylab = "Relative Humidity (%)", col = colors$purple, lwd = 1.2)
```

Plotting time series for output variable
```{r}
plot(Y_ts, main = "Time series plot of Electrical Energy Output",
     xlab = "Time (hr)", ylab = "Electrical Energy Output (MW)",
     col = colors$success, lwd = 1.2)
```

```
################################################################################
# Task 1.2: Distribution plot for each variable
################################################################################
```

Density plot of all input variables (X)
```{r}
density_X = density(X)
plot(density_X, main = "Density plot of all input variables (X)",
     col = colors$purple, lwd = 2)
rug(jitter(X), col = colors$dark, lwd = 0.8)
```

Histogram plot of all input variables (X)
```{r}
hist(X, freq = FALSE, main = "Histogram plot of all input variables (X)",
     col = colors$light, border = colors$primary, lwd = 1.5)
rug(jitter(X), col = colors$dark, lwd = 0.8)
```

Combine histogram and density plot of variable X
```{r}
hist(X, freq = FALSE, main = "Histogram and density plot of all input variables (X)",
     col = colors$light, border = colors$primary, lwd = 1.5)
lines(density_X, main = "Density plot of input variable X",
      col = colors$purple, lwd = 2)
rug(jitter(X), col = colors$dark, lwd = 0.8)
```

Histograms with density plots for all input variables
```{r}
par(mfrow=c(2,2))
```

Histogram with density plot for Temperature
```{r}
hist(X[,"Temperature"], freq = FALSE, main = "Distribution of Temperature",
     xlab = "Temperature (°C)", col = colors$light, border = colors$primary, lwd = 1.5)
lines(density(X[,"Temperature"]), lwd = 2, col = colors$secondary)
rug(jitter(X[,"Temperature"]), col = colors$dark, lwd = 0.8)
```

Histogram with density plot for Exhaust Vacuum
```{r}
hist(X[,"Vacuum"], freq = FALSE, main = "Distribution of Exhaust Vacuum",
     xlab = "Vacuum (cm Hg)", col = colors$light, border = colors$accent, lwd = 1.5)
lines(density(X[,"Vacuum"]), lwd = 2, col = colors$secondary)
rug(jitter(X[,"Vacuum"]), col = colors$dark, lwd = 0.8)
```

Histogram with density plot for Ambient Pressure
```{r}
hist(X[,"Pressure"], freq = FALSE, main = "Distribution of Ambient Pressure",
     xlab = "Pressure (millibar)", col = colors$light, border = colors$warning, lwd = 1.5)
lines(density(X[,"Pressure"]), lwd = 2, col = colors$secondary)
rug(jitter(X[,"Pressure"]), col = colors$dark, lwd = 0.8)
```

Histogram with density plot for Relative Humidity
```{r}
hist(X[,"Humidity"], freq = FALSE, main = "Distribution of Relative Humidity",
     xlab = "Humidity (%)", col = colors$light, border = colors$purple, lwd = 1.5)
lines(density(X[,"Humidity"]), lwd = 2, col = colors$secondary)
rug(jitter(X[,"Humidity"]), col = colors$dark, lwd = 0.8)
```

Histogram with density plot for output variable
```{r}
par(mfrow=c(1,1))
hist(Y, freq = FALSE, main = "Distribution of Electrical Energy Output",
     xlab = "Electrical Energy Output (MW)",
     col = "#E8F5E8", border = colors$success, lwd = 1.5)
lines(density(Y), lwd = 2, col = colors$success)
rug(jitter(Y), col = colors$dark, lwd = 0.8)
```
```{r}
## Combined histogram plots for all four input variables in a 2x2 grid
par(mfrow=c(2,2))

# Histogram with density plot for Temperature
hist(X[,"Temperature"], freq = FALSE, main = "Distribution of Temperature",
     xlab = "Temperature (°C)", col = colors$light, border = colors$primary, lwd = 1.5)
lines(density(X[,"Temperature"]), lwd = 2, col = colors$secondary)
rug(jitter(X[,"Temperature"]), col = colors$dark, lwd = 0.8)

# Histogram with density plot for Exhaust Vacuum
hist(X[,"Vacuum"], freq = FALSE, main = "Distribution of Exhaust Vacuum",
     xlab = "Vacuum (cm Hg)", col = colors$light, border = colors$accent, lwd = 1.5)
lines(density(X[,"Vacuum"]), lwd = 2, col = colors$secondary)
rug(jitter(X[,"Vacuum"]), col = colors$dark, lwd = 0.8)

# Histogram with density plot for Ambient Pressure
hist(X[,"Pressure"], freq = FALSE, main = "Distribution of Ambient Pressure",
     xlab = "Pressure (millibar)", col = colors$light, border = colors$warning, lwd = 1.5)
lines(density(X[,"Pressure"]), lwd = 2, col = colors$secondary)
rug(jitter(X[,"Pressure"]), col = colors$dark, lwd = 0.8)

# Histogram with density plot for Relative Humidity
hist(X[,"Humidity"], freq = FALSE, main = "Distribution of Relative Humidity",
     xlab = "Humidity (%)", col = colors$light, border = colors$purple, lwd = 1.5)
lines(density(X[,"Humidity"]), lwd = 2, col = colors$secondary)
rug(jitter(X[,"Humidity"]), col = colors$dark, lwd = 0.8)
```

```
################################################################################
# Task 1.3: Correlation and scatter plots
################################################################################

## Plotting the scatter plots between input variables and output
```{r}
par(mfrow=c(2,2))
plot(X[,"Temperature"], Y, main = "Temperature vs Electrical Energy Output",
     xlab = "Temperature (°C)", ylab = "Electrical Energy Output (MW)",
     pch = 16, cex = 0.8, col = colors$primary)
abline(lm(Y ~ X[,"Temperature"]), col = colors$secondary, lwd = 2.5)

plot(X[,"Vacuum"], Y, main = "Vacuum vs Electrical Energy Output",
     xlab = "Vacuum (cm Hg)", ylab = "Electrical Energy Output (MW)",
     pch = 16, cex = 0.8, col = colors$accent)
abline(lm(Y ~ X[,"Vacuum"]), col = colors$secondary, lwd = 2.5)

plot(X[,"Pressure"], Y, main = "Pressure vs Electrical Energy Output",
     xlab = "Pressure (millibar)", ylab = "Electrical Energy Output (MW)",
     pch = 16, cex = 0.8, col = colors$warning)
abline(lm(Y ~ X[,"Pressure"]), col = colors$secondary, lwd = 2.5)

plot(X[,"Humidity"], Y, main = "Humidity vs Electrical Energy Output",
     xlab = "Humidity (%)", ylab = "Electrical Energy Output (MW)",
     pch = 16, cex = 0.8, col = colors$purple)
abline(lm(Y ~ X[,"Humidity"]), col = colors$secondary, lwd = 2.5)
```

##
Task 2: Regression – modelling the relationship between electrical energy output
##

Creating a column of ones for the bias term
```{r}
ones = matrix(1, nrow(X), 1)
head(ones)
```

##
Task 2.1: Estimate model parameters for each candidate model
##

Model 1: y = θ₁x₄ + θ₂x₃² + θbias
scaling the variables and binding data from equation of model 1
```{r}
X_model1 <- cbind(scale(X[,"Pressure"]), scale(X[,"Vacuum"]^2), ones)
head(X_model1)
Model1_thetahat <- solve(t(X_model1) %*% X_model1) %*% t(X_model1) %*% Y
rownames(Model1_thetahat) <- c("θ₁", "θ₂", "θbias")
```

Printing values for Model 1
```{r}
print(Model1_thetahat)
t(Model1_thetahat)
```
```

Model 2: y = θ1x4 + θ2x3² + θ3x5 + θbias
scaling the variables and binding data from equation of model 2
```{r}
X_model2 <- cbind(scale(X[,"Pressure"]), scale(X[,"Vacuum"]^2), scale(X[,"Humidity"]), ones)
Model2_thetahat <- solve(t(X_model2) %*% X_model2) %*% t(X_model2) %*% Y
rownames(Model2_thetahat) <- c("θ1", "θ2", "θ3", "θbias")
```
Printing values for Model 2
```{r}
print(Model2_thetahat)
t(Model2_thetahat)
```

Model 3: y = θ1x3 + θ2x4 + θ3x5³
scaling the variables and binding data from equation of model 3
```{r}
X_model3 <- cbind(scale(X[,"Vacuum"]), scale(X[,"Pressure"]), scale(X[,"Humidity"]^3))
Model3_thetahat <- solve(t(X_model3) %*% X_model3) %*% t(X_model3) %*% Y
rownames(Model3_thetahat) <- c("θ1", "θ2", "θ3")
```
Printing values for Model 3
```{r}
print(Model3_thetahat)
t(Model3_thetahat)
```

Model 4: y = θ1x4 + θ2x3² + θ3x5³ + θbias
scaling the variables and binding data from equation of model 4
```{r}
X_model4 <- cbind(scale(X[,"Pressure"]), scale(X[,"Vacuum"]^2), scale(X[,"Humidity"]^3), ones)
```
Calculating theta hat for Model 4
```{r}
Model4_thetahat <- solve(t(X_model4) %*% X_model4) %*% t(X_model4) %*% Y
rownames(Model4_thetahat) <- c("θ1", "θ2", "θ3", "θbias")
```
Printing values for Model 4
```{r}
print(Model4_thetahat)
t(Model4_thetahat)
```
Model 5: y = θ1x4 + θ2x1² + θ3x3² + θbias
scaling the variables and binding data from equation of model 5
```{r}
X_model5 <- cbind(scale(X[,"Pressure"]), scale(X[,"Temperature"]^2), scale(X[,"Vacuum"]^2), ones)
Model5_thetahat <- solve(t(X_model5) %*% X_model5) %*% t(X_model5) %*% Y
rownames(Model5_thetahat) <- c("θ1", "θ2", "θ3", "θbias")
```
Printing values for Model 5
```{r}
print(Model5_thetahat)
t(Model5_thetahat)
```

```
################################################################################
# Task 2.2: Calculating the RSS value for each candidate model
################################################################################
```

Calculating predicted values and RSS for Model 1

```{r}
Y_hat_model1 <- X_model1 %*% Model1_thetahat
RSS_Model_1 <- sum((Y - Y_hat_model1)^2)
```

Calculating predicted values and RSS for Model 2

```{r}
Y_hat_model2 <- X_model2 %*% Model2_thetahat
RSS_Model_2 <- sum((Y - Y_hat_model2)^2)
```

Calculating predicted values and RSS for Model 3

```{r}
Y_hat_model3 <- X_model3 %*% Model3_thetahat
RSS_Model_3 <- sum((Y - Y_hat_model3)^2)
```

Calculating predicted values and RSS for Model 4

```{r}
Y_hat_model4 <- X_model4 %*% Model4_thetahat
RSS_Model_4 <- sum((Y - Y_hat_model4)^2)
```

Calculating predicted values and RSS for Model 5

```{r}
Y_hat_model5 <- X_model5 %*% Model5_thetahat
RSS_Model_5 <- sum((Y - Y_hat_model5)^2)
```

Creating a data frame of RSS values for comparison

```{r}
RSS_values <- data.frame(
  Model = c("Model 1", "Model 2", "Model 3", "Model 4", "Model 5"),
  RSS = c(RSS_Model_1, RSS_Model_2, RSS_Model_3, RSS_Model_4, RSS_Model_5)
)
```

Printing Residual Sum of Squares (RSS) for All Models

```{r}
print(RSS_values)
```

```
################################################################################
# Task 2.3: Calculating log-likelihood and variance
################################################################################

## Total number of observations
```{r}
N <- nrow(Y)
```

Calculating variance and log-likelihood for Model 1
```{r}
Variance_model1 <- RSS_Model_1 / (N - 1)
likelihood_Model_1 <- -(N/2) * log(2*pi) - (N/2) * log(Variance_model1) -
                      (1/(2*Variance_model1)) * RSS_Model_1
```

Calculating variance and log-likelihood for Model 2
```{r}
Variance_model2 <- RSS_Model_2 / (N - 1)
likelihood_Model_2 <- -(N/2) * log(2*pi) - (N/2) * log(Variance_model2) -
                      (1/(2*Variance_model2)) * RSS_Model_2
```

Calculating variance and log-likelihood for Model 3
```{r}
Variance_model3 <- RSS_Model_3 / (N - 1)
likelihood_Model_3 <- -(N/2) * log(2*pi) - (N/2) * log(Variance_model3) -
                      (1/(2*Variance_model3)) * RSS_Model_3
```

Calculating variance and log-likelihood for Model 4
```{r}
Variance_model4 <- RSS_Model_4 / (N - 1)
likelihood_Model_4 <- -(N/2) * log(2*pi) - (N/2) * log(Variance_model4) -
                      (1/(2*Variance_model4)) * RSS_Model_4
```

Calculating variance and log-likelihood for Model 5
```{r}
Variance_model5 <- RSS_Model_5 / (N - 1)
likelihood_Model_5 <- -(N/2) * log(2*pi) - (N/2) * log(Variance_model5) -
                      (1/(2*Variance_model5)) * RSS_Model_5
```

Creating data frames for variance and likelihood values
```{r}
variance_values <- data.frame(
  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_values <- data.frame(
  Model = c("Model 1", "Model 2", "Model 3", "Model 4", "Model 5"),
  LogLikelihood = c(likelihood_Model_1, likelihood_Model_2, likelihood_Model_3,
                    likelihood_Model_4, likelihood_Model_5)
)
```
```

Printing Variance Values for All Models
```{r}
print(variance_values)
```

Printing Log-Likelihood Values for All Models
```{r}
print(likelihood_values)
```

```
################################################################################
# Task 2.4: Compute AIC and BIC for every candidate model
################################################################################
```

Calculating the number of parameters (K) for each model
```{r}
K_model1 <- length(Model1_thetahat)
K_model2 <- length(Model2_thetahat)
K_model3 <- length(Model3_thetahat)
K_model4 <- length(Model4_thetahat)
K_model5 <- length(Model5_thetahat)
```

Calculating AIC for each model
```{r}
AIC_model1 <- 2 * K_model1 - 2 * likelihood_Model_1
AIC_model2 <- 2 * K_model2 - 2 * likelihood_Model_2
AIC_model3 <- 2 * K_model3 - 2 * likelihood_Model_3
AIC_model4 <- 2 * K_model4 - 2 * likelihood_Model_4
AIC_model5 <- 2 * K_model5 - 2 * likelihood_Model_5
```

Calculating BIC for each model
```{r}
BIC_model1 <- K_model1 * log(N) - 2 * likelihood_Model_1
BIC_model2 <- K_model2 * log(N) - 2 * likelihood_Model_2
BIC_model3 <- K_model3 * log(N) - 2 * likelihood_Model_3
BIC_model4 <- K_model4 * log(N) - 2 * likelihood_Model_4
BIC_model5 <- K_model5 * log(N) - 2 * likelihood_Model_5
```

Creating data frames for K, AIC, and BIC values
```{r}
parameter_counts <- data.frame(
  Model = c("Model 1", "Model 2", "Model 3", "Model 4", "Model 5"),
  K = c(K_model1, K_model2, K_model3, K_model4, K_model5)
)

AIC_values <- data.frame(
  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_values <- data.frame(
  Model = c("Model 1", "Model 2", "Model 3", "Model 4", "Model 5"),
  BIC = c(BIC_model1, BIC_model2, BIC_model3, BIC_model4, BIC_model5)
)
```

Printing Number of Parameters (K) for Each Model
```{r}
print(parameter_counts)
```

Printing AIC Values for All Models
```{r}
print(AIC_values)
```

Printing BIC Values for All Models
```{r}
print(BIC_values)
```

```
################################################################################
# Task 2.5: Checking the distribution of model prediction errors (residuals)
################################################################################
```

```{r}
par(mfrow=c(2,3))
```

Calculating and plotting residuals for Model 1
```{r}
model1_error <- Y - Y_hat_model1
qqnorm(model1_error, main = "Q-Q Plot for Model 1", col = colors$primary, pch = 16)
qqline(model1_error, col = colors$secondary, lwd = 2)
```

Calculating and plotting residuals for Model 2
```{r}
model2_error <- Y - Y_hat_model2
qqnorm(model2_error, main = "Q-Q Plot for Model 2", col = colors$accent, pch = 16)
qqline(model2_error, col = colors$secondary, lwd = 2)
```

Calculating and plotting residuals for Model 3
```{r}
model3_error <- Y - Y_hat_model3
qqnorm(model3_error, main = "Q-Q Plot for Model 3", col = colors$warning, pch = 16)
qqline(model3_error, col = colors$secondary, lwd = 2)
```

Calculating and plotting residuals for Model 4
```{r}
model4_error <- Y - Y_hat_model4
qqnorm(model4_error, main = "Q-Q Plot for Model 4", col = colors$purple, pch = 16)
qqline(model4_error, col = colors$secondary, lwd = 2)
```

Calculating and plotting residuals for Model 5
```{r}
model5_error <- Y - Y_hat_model5
qqnorm(model5_error, main = "Q-Q Plot for Model 5", col = colors$success, pch = 16)
qqline(model5_error, col = colors$secondary, lwd = 2)
```

```
################################################################################
# Task 2.6: Selecting the best regression model
################################################################################

## Resetting plot layout
```{r}
par(mfrow=c(1,1))
```

Combining all evaluation metrics for comparison
```{r}
model_comparison <- data.frame(
  Model = c("Model 1", "Model 2", "Model 3", "Model 4", "Model 5"),
  RSS = c(RSS_Model_1, RSS_Model_2, RSS_Model_3, RSS_Model_4, RSS_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)
)
```

Finding the best model based on each criterion
```{r}
best_RSS_model <- model_comparison$Model[which.min(model_comparison$RSS)]
best_AIC_model <- model_comparison$Model[which.min(model_comparison$AIC)]
best_BIC_model <- model_comparison$Model[which.min(model_comparison$BIC)]
```

Printing the comparison values
```{r}
print(model_comparison)
```

##
Task 2.7: Training/Testing Split and Confidence Intervals
##

Splitting data together to maintain correspondence
```{r}
set.seed(123)
combined_data <- data.frame(X, Y)
split_indices <- initial_split(combined_data, prop = 0.7)
training_set <- training(split_indices)
testing_set <- testing(split_indices)
```

Extracting training and testing data
```{r}
X_train <- as.matrix(training_set[, c("Temperature", "Vacuum", "Pressure", "Humidity")])
Y_train <- as.matrix(training_set[, "Energy_Output"])
X_test <- as.matrix(testing_set[, c("Temperature", "Vacuum", "Pressure", "Humidity")])
Y_test <- as.matrix(testing_set[, "Energy_Output"])
```
```

Model 5: y = θ₁x₄ + θ₂x₁² + θ₃x₃² + θbias (Pressure + Temperature² + Vacuum²)
```{r}
ones_train <- matrix(1, nrow(X_train), 1)
X_train_model <- cbind(scale(X_train[, "Pressure"]),
                       scale(X_train[, "Temperature"]^2),
                       scale(X_train[, "Vacuum"]^2),
                       ones_train)
```

Estimating parameters
```{r}
train_thetahat <- solve(t(X_train_model) %*% X_train_model) %*% t(X_train_model) %*% Y_train
```

Preparing test data
```{r}
ones_test <- matrix(1, nrow(X_test), 1)
X_test_model <- cbind(scale(X_test[, "Pressure"]),
                      scale(X_test[, "Temperature"]^2),
                      scale(X_test[, "Vacuum"]^2),
                      ones_test)
```

Making predictions
```{r}
Y_test_pred <- X_test_model %*% train_thetahat
```

Proper confidence intervals
```{r}
residuals <- Y_test - Y_test_pred
std_error <- sqrt(sum(residuals^2) / (nrow(Y_test) - ncol(X_test_model)))
t_value <- qt(0.975, df = nrow(Y_test) - ncol(X_test_model))
lower_CI <- Y_test_pred - t_value * std_error
upper_CI <- Y_test_pred + t_value * std_error
```

Plotting with error bars
```{r}
plot(1:length(Y_test), Y_test, pch = 16, col = colors$primary,
     ylim = c(min(lower_CI), max(upper_CI)), cex = 1.2,
     xlab = "Sample Index", ylab = "Power Output (MW)",
     main = "Model Predictions with 95% Confidence Intervals")
points(1:length(Y_test), Y_test_pred, pch = 17, col = colors$secondary, cex = 1.2)
arrows(1:length(Y_test), lower_CI, 1:length(Y_test), upper_CI,
       length = 0.03, angle = 90, code = 3, col = colors$success, lwd = 1.5)
legend("topright",
       legend = c("Actual", "Predicted", "95% CI"),
       col = c(colors$primary, colors$secondary, colors$success),
       pch = c(16, 17, NA), lty = c(NA, NA, 1), lwd = c(NA, NA, 1.5))

# Performance metrics
print(paste("RSS:", sum(residuals^2)))
print(paste("MAE:", mean(abs(residuals))))
print(paste("RMSE:", sqrt(mean(residuals^2))))
```

```
################################################################################
# Task 3: Approximate Bayesian Computation (ABC)
################################################################################

## From Task 2.6, we determined Model 5 is the best performing model
## Model 5: y = θ₁x₄ + θ₂x₁² + θ₃x₃² + θbias
## Identifying the parameters with largest absolute values
```{r}
print("Model 5 Parameter Estimates:")
print(Model5_thetahat)
```

Calculating absolute values of all parameters
```{r}
param_abs_values <- abs(Model5_thetahat)
print("Absolute values of parameters:")
print(param_abs_values)
```

Finding the two largest absolute values
```{r}
sorted_indices <- order(param_abs_values, decreasing = TRUE)
top_two_indices <- sorted_indices[1:2]
top_two_params <- Model5_thetahat[top_two_indices]
param_names <- rownames(Model5_thetahat)[top_two_indices]

print("Two parameters with largest absolute values:")
for(i in 1:2) {
  print(paste(param_names[i], "=", top_two_params[i]))
}
```

Setting up uniform priors around the estimated parameter values
We'll use a range of ±100% of the absolute value of each parameter
```{r}
prior_ranges <- matrix(0, 2, 2)
scaling_factor <- 1  # 100% of the parameter value

for(i in 1:2) {
  param_value <- top_two_params[i]
  range_width <- scaling_factor * abs(param_value)
  prior_ranges[i, 1] <- param_value - range_width  # Lower bound
  prior_ranges[i, 2] <- param_value + range_width  # Upper bound
}

rownames(prior_ranges) <- param_names
colnames(prior_ranges) <- c("Lower_Bound", "Upper_Bound")

print("Prior Distribution Ranges:")
print(prior_ranges)
```

Function to generate samples from uniform prior distributions
```{r}
generate_prior_sample <- function() {
  c(
    runif(1, prior_ranges[1, 1], prior_ranges[1, 2]),
    runif(1, prior_ranges[2, 1], prior_ranges[2, 2])
  )
}
```
```

Creating fixed parameters (the ones we're not varying)
```{r}
fixed_params <- Model5_thetahat
fixed_indices <- setdiff(1:length(Model5_thetahat), top_two_indices)

print("Fixed parameters:")
print(fixed_params[fixed_indices])
```

Calculating RSS for a given set of parameters
```{r}
calculate_RSS <- function(params) {
  # Creating a complete parameter set
  full_params <- fixed_params
  full_params[top_two_indices] <- params

  # Calculate predictions using Model 5 structure
  X_model5_abc <- cbind(scale(X[,"Pressure"]),
                        scale(X[,"Temperature"]^2),
                        scale(X[,"Vacuum"]^2),
                        ones)
  Y_pred <- X_model5_abc %*% full_params

  # Calculate RSS
  sum((Y - Y_pred)^2)
}
```

Getting the RSS of our best model as reference
```{r}
best_model_RSS <- RSS_Model_5
print(paste("Best model RSS:", best_model_RSS))
```

Setting up rejection ABC parameters
```{r}
num_samples <- 10000  # Number of samples to generate
accepted_samples <- matrix(0, 0, 2)  # Matrix to store accepted samples
colnames(accepted_samples) <- param_names
```

Setting threshold as 5 times the best model RSS (500% tolerance)
```{r}
threshold <- best_model_RSS * 5.0
print(paste("ABC threshold:", threshold))
```

Performing rejection ABC with progress monitoring
```{r}
cat("Performing rejection ABC sampling...\n")
for(i in 1:num_samples) {
  # Generating sample from prior
  sample_params <- generate_prior_sample()

  # Calculating RSS for this parameter combination
  sample_RSS <- calculate_RSS(sample_params)

  # Accept or reject based on threshold
  if(sample_RSS <= threshold) {
    accepted_samples <- rbind(accepted_samples, sample_params)
  }
```

```
for(i in 1:num_samples) {

  # Progress indicator every 2000 samples
  if(i %% 2000 == 0) {
    current_rate <- round(nrow(accepted_samples)/i * 100, 2)
    cat("Processed", i, "samples, accepted", nrow(accepted_samples),
        "- Rate:", current_rate, "%\n")
  }
}

print(paste("Total accepted samples:", nrow(accepted_samples)))
print(paste("Acceptance rate:", round(nrow(accepted_samples)/num_samples * 100, 2), "%"))
```

Plotting joint and marginal posterior distributions with error handling
```{r}
if(nrow(accepted_samples) >= 10) {

  # Clear any existing plot device issues
  tryCatch({
    # Set up plotting area
    par(mfrow=c(2,2))

    # Joint posterior distribution
    plot(accepted_samples[, 1], accepted_samples[, 2],
         pch = 16, cex = 0.8, col = colors$primary,
         xlab = param_names[1], ylab = param_names[2],
         main = "Joint Posterior Distribution")

    # Adding lines showing the point estimates
    abline(v = top_two_params[1], col = colors$secondary, lty = 2, lwd = 2)
    abline(h = top_two_params[2], col = colors$secondary, lty = 2, lwd = 2)

    # Adding point estimate
    points(top_two_params[1], top_two_params[2],
           col = colors$danger, pch = 16, cex = 1.8)

    legend("topright",
           legend = c("Posterior samples", "Point estimate"),
           col = c(colors$primary, colors$danger),
           pch = c(16, 16))

    # Plotting marginal posterior distribution for first parameter
    hist(accepted_samples[, 1], breaks = 30,
         col = colors$light, border = colors$primary, lwd = 1.5,
         main = paste("Marginal Posterior of", param_names[1]),
         xlab = param_names[1], ylab = "Frequency")
    abline(v = top_two_params[1], col = colors$secondary, lty = 2, lwd = 2)
    abline(v = mean(accepted_samples[, 1]), col = colors$success, lty = 2, lwd = 2)

    legend("topright",
           legend = c("Point estimate", "Posterior mean"),
           col = c(colors$secondary, colors$success),
           lty = c(2, 2), lwd = c(2, 2))
```

```
    # Marginal posterior distribution for second parameter
    hist(accepted_samples[, 2], breaks = 30,
         col = colors$light, border = colors$primary, lwd = 1.5,
         main = paste("Marginal Posterior of", param_names[2]),
         xlab = param_names[2], ylab = "Frequency")
    abline(v = top_two_params[2], col = colors$secondary, lty = 2, lwd = 2)
    abline(v = mean(accepted_samples[, 2]), col = colors$success, lty = 2, lwd = 2)

    legend("topright",
           legend = c("Point estimate", "Posterior mean"),
           col = c(colors$secondary, colors$success),
           lty = c(2, 2), lwd = c(2, 2))

  }, error = function(e) {
    cat("Plotting error occurred:", e$message, "\n")
    cat("Continuing with summary statistics...\n")
  })

  # Reset plot layout
  par(mfrow=c(1,1))

  # Calculating summary statistics for posterior distributions
  posterior_means <- colMeans(accepted_samples)
  posterior_sds <- apply(accepted_samples, 2, sd)
  posterior_medians <- apply(accepted_samples, 2, median)

  # Calculating credible intervals (Bayesian equivalent of confidence intervals)
  credible_intervals <- apply(accepted_samples, 2, function(x) {
    quantile(x, c(0.025, 0.975))
  })

  # Creating summary table
  summary_stats <- data.frame(
    Parameter = param_names,
    Point_Estimate = top_two_params,
    Posterior_Mean = posterior_means,
    Posterior_Median = posterior_medians,
    Posterior_SD = posterior_sds,
    CI_Lower = credible_intervals[1, ],
    CI_Upper = credible_intervals[2, ]
  )

  # Printing posterior distribution summary
  print(summary_stats)
} else {
  cat("Error: Too few accepted samples for meaningful analysis.\n")
  cat("Suggestions:\n")
  cat("1. Increase the threshold (try threshold <- best_model_RSS * 10.0)\n")
  cat("2. Increase the number of samples (try num_samples <- 50000)\n")
  cat("3. Widen the prior ranges (try scaling_factor <- 2.0)\n")
}
```