Monte Carlo simulation

Christian, Jonathan, Marcus, Puk & Sofia

May 5, 2025

Contents

1 Introduction

Regression is a tool in statistics used to understand the relationship between one dependent variable and one or more independent variables. For regression to be both accurate and reliable, a set of assumptions needs to be met. These assumptions are independence of errors, linearity, homoscedasticity, normality of errors, multicollinearity, and correct model specifications. If these assumptions are not met, it can introduce bias and inaccuracies in the regression model.

1.1 Independence of Errors

This assumption states that the errors from the model are not correlated with each other but are independent. This means that one error cannot be used to predict the next one.

1.2 Linearity

This assumption states that the relationship between the independent variables and the parameters, also known as the coefficients, is linear. This does not mean that the regression model itself has to be linear. For example, in polynomial regression, the relationship between the independent variables and the coefficients is still linear, but the independent variables can be transformed using powers.

1.3 Homoscedasticity

This is the assumption of constant variance across errors for all levels of the independent variables.

1.4 Normality of Errors

This assumption states that the errors between the model and the observed values, also called residuals, are normally distributed. If this is not met, it may result in a biased model and a worse model fit.

1.5 Multicollinearity

This occurs when two or more independent variables in a regression model are highly correlated with each other. This means that changes in one independent variable are associated with changes in another, making it difficult to determine the individual effect of each independent variable on the dependent variable.

1.6 Correct Model Specifications

This assumes that the provided dependent variables for the models are the correct ones. If one is missing or the model is overfitting, this may result in incorrect coefficients and introduce errors into the model.

2 Statistical theory

2.1 Probability space

The **sample space**, S, is the set of all possible outcomes.
If a sample space contains a finite number of possibilities or an unending sequence with as many elements as there are whole numbers, it is called a **discrete sample space**.
Example: When rolling a standard six-sided die form the discrete sample space, the possible outcomes are $S = 1, 2, 3, 4, 5, 6$

If a sample space contains an infinite number of possibilities equal to the number of points on a line segment, it is called a **continuous sample space**.
Example: Measuring the heights of people in a population. This is a continuous sample space, because height can take any real value within a given range.
An **event** is a subset, $A \subseteq S$, of the sample space. The event is the amount that contains all possible events. An example of a discrete event could be rolling a die and getting an uneven number, this would be the event $A = 1, 3, 5$.
For the continuous event, it could be that a person is between 160 cm and 170 cm tall.

The probability of an event A, $P(A)$, is the sum of the weights of all sample points in A. The probability of the whole sample space is 1, $P(S) = 1$ The probability of any event being between 0 and 1,$0 < P(A) < 1$ The probability of the empty set being 0, $P(\emptyset) = 0$

Probability of mutually exclusive events
If A and B are mutually exclusive, $A \cap B = \emptyset$, then
$P(A \cup B) = P(A) + P(B)$

Where A and B never occur at the same time, so their union is equal to the two events added together.
Probability of union
$P(A \cup B) = P(A) + P(B) - P(A \cap B)$
Here, the union of the two events is A added to B, but minus their common event, since it otherwise would be added twice.
Two events A and B are independent, if
$P(A|B) = P(A)$
The equivalent definition to this is:
Two events A and B are independent if and only if
$P(A \cap B) = P(A)P(B)$ This says that the probability of both event A and B happening, is equal to the product of the two events. If we are interested in certain parameters of a population distribution, we can look at a sample. From this, we can make a **point estimate**.
Examples of this are,
$\bar{x}$ is a point estimate of μ

s is a point estimate of σ

This is often supplemented with a **confidence interval**
This is an interval around the point estimate, where we are confident that the population parameter is located.

For μ, we have different ways of estimating it. We can use the sample mean $\bar{X}$, or the average X_T of the sample upper and lower quartiles. But in this case, we have to look out for **bias**. If the distribution of a population is skewed, then X_T is biased. The result of this is, that in the long run, this estimator will systematically over or under estimate the value of μ. This is written as,
$E(X_T) \neq \mu$.
It is generally preferred that the estimator is **unbiased**. In this case, $\bar{X}$ is an unbiased estimate of the population mean μ.

The standard error of $\bar{X}$ is $\frac{\sigma}{\sqrt{n}}$. Here, the standard error decreases, when the sample size increases. If an estimator has this property, it is called **consistent**. If we compare, the estimator X_T is also consistent, but has a greater variance than $\bar{X}$.
It is generally preferred that the estimator has the smallest possible variance, and in that case it is called efficient. So $\bar{X}$ is an efficient estimator.
When estimating a parameter, the symbol $\hat{}$ is used above it. For μ, $\hat{\mu} = \bar{X}$.
We can calculate $\bar{X}$ using the following formula,

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

For the variance σ, we can estimate it by using the formula for S^2,

$$S^2 = \frac{1}{n-1}\sum_{i=1}(X_i - \bar{X})$$

2.2 Probability distribution

Data can come in various distributions depending on different parameters such as degrees of freedom. The distribution is the shape of the data and it will have an effect on statistical models. Therefore it is important to have an understanding of distributions.

2.2.1 Normal distribution

In the world of statistics, the most common distribution is the normal distribution. It is constructed as a bell shape. The normal distribution is a continuous distribution, with this density function:

$$n(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)}{2\sigma^2}}$$

The distribution is dependent on the mean(μ) and the standard deviation(σ), where changes to the mean will result in a change in the positioning of the normal distribution. Whereas a change in the standard deviation will change the spread of the curve. The normal distribution also always contains an area under the curve that is equal to one. This is to ensure that the normal distribution correctly models probability.

There is a special case of the normal distribution, called the standard normal distribution, where the mean is zero and the standard deviation is one. All variations of a normal distribution can be standardized by a transformation of the distribution, using the Z-score formula.

$$Z = \frac{X - \mu}{\sigma}$$

Z in the Z-score represents the amount of standard deviations a given X value, deviates from the mean.

2.2.2 The central limit theorem

A very effective theorem in statistics is the central limit theorem. This theorem states that if a random sample $\overline{X}$, with the size n, is taken from a population with a mean and a finite variance, then as n goes towards infinity, the distribution will resemble a normal distribution. If used with the Z-score formula, the distribution will resemble a standard normal distribution. The formula for the Z-score, when in conjunction with the central limit theorem, looks like this:

$$Z = \frac{\overline{X} - \mu}{\sigma/\sqrt{n}}$$

Where $\overline{X}$ is a random sample of size n and μ is the mean of the true population. The standard error i represented by $\sigma/\sqrt{n}$, where σ is the standard deviation and n is the sample size. Usually the standard deviation is unknown, for these situations it's possible to use the estimator S^2. This estimates the variance of the population from the variance of the sample, by this formula:

$$s^2 = \sum_{i=1}^{n} \frac{(x_i - \overline{x})}{n - 1}$$

The square root of the variance is the standard deviation, therefore the square root of the estimator S^2 would be the estimated standard deviation. The problem with using the estimator S^2, is that with small samples the variance is small and therefore it contains a lot of bias. In this situation the t-distribution would be used instead of the normal distribution, because the t-distribution takes the bias into account the bias of the standard deviation. It does this by having thicker tails, meaning that the probability of more extreme values are higher.

2.2.3 The t-distribution

The t-distribution is shaped as the standard normal distribution, in a bell shape and symmetrical around the mean of zero, the difference is that the t-distribution is more variable. This comes from the fact that the t-distribution is dependent on the degrees of freedom. When the degrees of freedom surpasses 30, the rule of thumb is that the distribution will resemble a normal distribution. So before 30 degrees of freedom, the distribution contains more variance. The t-distribution will come to resemble the standard normal distribution, when in surpasses 30 degrees of freedom, this makes sense, since the two distributions have the same formula:

$$T = \frac{\overline{X} - \mu}{S/\sqrt{n}}$$

The only difference is the estimated standard deviation S.

2.3 Statistical methods

2.3.1 Confidence intervals

The confidence interval is a good tool to use, when trying to estimate a parameter of a population. Its used to create an interval, where the parameter has a probability to lie inside of. This probability is called the confidence level and it's a chosen value, usually the chosen confidence level is either 95% or 99%. The confidence interval will become bigger with a larger confidence level. A good confidence interval is small with a large confidence level, this will usually occur when the sample size is large. The chosen confidence level relates to an α-value, where as an example the chosen confidence level is 95%, then the α-value would be 5% or normally written as 0.05. The α-value will sometimes be needed to find the critical value, that is used to calculate the margin of error, as an example it's used when trying to find the critical value of the confidence interval, when working with a t-distribution.
To set up a confidence interval, the margin of error needs to be computed and then that will be both added and subtracted from the point estimate. This will give the values of the outer bounds of the interval. The margin of error is calculated from this formula:

$$Margin\_of\_error = critical\_value \pm standard\_error$$

The standard error will change depending on which parameter that the confidence interval is estimating, but the general formula for the standard error is:

$$\frac{\sigma}{\sqrt{n}}$$

An example of computing a confidence interval of the mean while working with a standard normal distribution, then the formula for the confidence interval would be this:

$$P(-z_{\alpha/2} < Z < z_{\alpha/2}) = 1 - \alpha$$

Where $1 - \alpha$ is the confidence level. As it's the mean that is being estimated, then instead of Z-score, then μ must be isolated and that is done by multiplying $\frac{\sigma}{\sqrt{n}}$ and subtracting $\bar{X}$ on all sides, then multiplying all side by -1 to remove the minus sign. So the formula for a confidence interval of the mean will look like this:

$$P(\bar{X} - z_{\alpha/2}\frac{\sigma}{\sqrt{n}} < \mu < \bar{X} + z_{\alpha/2}\frac{\sigma}{\sqrt{n}}) = 1 - \alpha$$

This formula will give the upper and lower bounds of the confidence interval.

The interpretation of a confidence interval
To interpret a confidence interval, it would be incorrect to interpret the confidence level of some value x, as the probability of the true parameter being inside of the interval. The reason behind this is that the computed interval is static, so either the value x is inside the interval or it's not. So the correct way of interpreting the confidence interval is by taking multiple samples and computing the confidence interval for all samples, then the value x would reside inside 95% of the confidence intervals. **Kilde for fortolkningen af kofidense intervaller:**
$http://www.drhuang.com/science/mathematics/book/probability_and_statistics_for_engineering_and_the_scienc$

2.3.2 Hypothesis testing

A hypothesis test is used to test an assumption about a population. This is done from a sample of the population, as the information about the population is usually hard to come by. A hypothesis test is set up, by having a null hypothesis and an alternate hypothesis.

$$H_0 = Null\ hypothesis$$

$$H_a = Alternate\ hypothesis$$

When working with hypothesis testing, the hypothesis H_0 is usually represented as the status quo, where as the hypothesis H_a is represented as the opposition. It is also important to note that there is only two outcomes of a hypothesis test, either H_0 is rejected in favor of H_a or H_0 is failed to be rejected. Therefore in no situation can H_0 be stated to be an absolute truth, as there might be other samples where H_0 will be rejected. Therefore in a hypothesis test H_0 needs to be the thing that can be rejected and if H_0 gets rejected, then H_a will become the new status quo until proven otherwise.
In a hypothesis test H_0 will be the assumption that a parameter for two populations is the same, where as H_a can be either one of three assumptions, depending

on the intention of the hypothesis test.

$$H_0 : \theta = \theta_0$$

1. $H_a : \theta \neq \theta_0$
2. $H_a : \theta < \theta_0$
3. $H_a : \theta > \theta_0$

When the direction of the rejection is not important and also is unknown, then (1) will be the case. This scenario sets up a two-tailed-test, where the hypothesis test is used to reject H_0 if H_a is either significantly larger or smaller than H_0, this means that the critical area is on both sides of the difference of θ and θ_0. Either (2) or (3) will set up a one-tailed-test, where depending on what is important, either the hypothesis test is used to determine if H_a is significantly bigger or smaller than H_0. This means that the critical area only spans one side of the difference between θ and θ_0.

3 Polynomial regression

input(Polinomial regression)

3.1 Assumptions

4 Pseudo random number generator

5 Pseudo Random Number Generator

To generate the dataset, we use a Pseudo-Random Number Generator (PRNG). PRNGs are algorithms that produce sequences of numbers that appear random but are actually deterministically generated from an initial seed value. While these numbers are not truly random, they are sufficiently unpredictable for many practical applications. Random numbers are widely used in fields such as statistics, game theory, cryp- tography, and simulations. These applications require numbers that behave as if they were random, yet can be reproduced when needed. This is where PRNGs come in—they allow for repeatable randomness, making them ideal for controlled experiments, testing, and security. This chapter will explore the key concepts behind PRNGs. Before going into the mechanics of these generators, it is important to first understand what 'random' means and the characteristics that define truly random numbers.

5.1 Properties of PRNGs

The quality of a PRNG is determined by several key factors that influence its use for different applications. Some of the properties of a good PRNG is properties: Independency, a large period and reproducibility The numbers produced by the PRNG should be statistically independent, en- suring that each generated value exhibits no correlation with previous numbers or other sequences. This implies that knowledge of previously generated num- bers or sequences provides no advantage in predicting the next output. A PRNG operates within a specific interval before its sequence begins to repeat. A high-quality PRNG has a long interval, delaying repetition and enhancing its unpredictability. Conversely, a PRNG with a shorter period becomes more pre- dictable and less suitable for practical use. A key feature of a PRNG is its ability to reproduce the same sequence of num- bers when given a specific seed. This property is particularly useful in testing and simulation scenarios, where it is essential to generate identical sequences multiple times for consistency and reproducibility In addition, a PRNG must be fast and efficient to prevent it from introducing performance bottlenecks within an application. The speed of number generation directly impacts computational efficiency, especially in applications requiring a large volume of random numbers. An inefficient PRNG can significantly slow down processes, undermining the overall performance of the system. Therefore, balancing randomness and efficiency is essential for practical applications

5.2 Linear Congruential Generator

Linear Congruential Sequence (LCS) is a commonly used approach to generate pseudo-random numbers. LCS generates a sequence of numbers using a linear recurrence relation LCS is expressed as:

$$X_{n+1} = (aX_n + c) \bmod m.$$

where X_0 is the seed, a is the multiplier, c is the increment, and m is the modulus.

Example: Given $a = 5$, $c = 1$, $m = 16$, and $X_0 = 7$:

$$X_1 = (5 \cdot 7 + 1) \bmod 16 = 4$$

$$X_2 = (5 \cdot 4 + 1) \bmod 16 = 5$$

$$X_3 = (5 \cdot 5 + 1) \bmod 16 = 10$$

$$X_4 = (5 \cdot 10 + 1) \bmod 16 = 3$$

This sequence has a period of 16. In an LCG, the period can be as large as m, but choosing parameters carefully is crucial to achieving long periods. Therefore

6 Monte Carlo

6.1 Assumptions

7 Problem statement

8 intro til data

9 intro til data

To determine whether the data set meets the assumption of homoscedasticity, a scatter plot between the dependent and independent variables is created. It becomes apparent that the variables displacement, weight, and acceleration are not homoscedastic.Furthermore, MPG, displacement, weight, and acceleration are continuous numeric variables, while cylinders, model year, and origin are discrete numeric variables. Additionally, a heat map of the correlation between variables is created to determine if multicollinearity is present among the independent variables. Here, it is apparent that weight, displacement, and cylinders are highly correlated, and that all three are highly correlated with MPG. It is also clear from observing the density functions of the independent variables that none of them, except acceleration, are normally distributed.

```
library(ggplot2)
library(dplyr)
library(gridExtra)
library(reshape2)
library(lmtest)
library(gridExtra)
#read data
data <- read.csv("auto-mpg.csv", na.strings = ".")

#calculet the NA-%
na_percentage <- function(column) {
  sum(is.na(column)) / length(column) * 100
}

#print the NA-%
print(sapply(data, na_percentage))

# Convert horsepower to numeric, coercing non-numeric values to NA
data$horsepower <- as.numeric(data$horsepower)



# Remove rows with any NA values
data <- data %>% na.omit()

"numeric_data <- data %>% select_if(is.numeric)

# Beregn korrelationer og smelt til format for ggplot2
data_korrelationer <- melt(cor(numeric_data, use = "complete.obs"))

"Opret heatmap baseret p korrelationer
ggplot(data_korrelationer, aes(x = Var1, y = Var2, fill = abs(value
    ))) +
  geom_tile() +
  geom_text(aes(label = round(value, 2)), color = "black", size =
    4) +
  scale_fill_gradient(low = "white", high = "red", limit = c(0, 1),
     name="|Correlation|") +
```

```
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Correlation Matrix Heatmap", x = "", y = "")

" Lav scatter plots for hver kolonne mod 'mpg'
scatter_plots <- lapply(names(numeric_data), function(col) {
  ggplot(numeric_data, aes_string(x = col, y = "mpg")) +
    geom_point(color = "blue") +
    ggtitle(paste("Scatter Plot of", col, "vs mpg")) +
    theme_minimal()
})

"bp_results <- sapply(names(numeric_data), function(col) {
  model <- lm(mpg ~ numeric_data[[col]], data = numeric_data)
  bp_test <- bptest(model)
  bp_test$p.value
})

#make df with P_Value
bp_df <- data.frame(Variable = names(bp_results), P_Value = bp_
    results)

print(bp_df)


arrange scatter plots and Breusch prage plot
do.call(grid.arrange, c(scatter_plots, ncol = 3))

"Lav histogrammer og density plots med mean og SD
plots <- lapply(names(numeric_data), function(col) {
  n_bins <- ceiling(log2(length(numeric_data[[col]])) + 1)
  mean_value <- mean(numeric_data[[col]])
  sd_value <- sd(numeric_data[[col]])

  ggplot(numeric_data, aes_string(x = col)) +
    geom_histogram(aes(y = ..density..), bins = n_bins, fill = "
    blue", color = "black") +
    geom_density(color = "red", size = 1) +
    geom_vline(aes(xintercept = mean_value), color = "green",
    linetype = "dashed", size = 1) +
    ggtitle(paste("Density, Mean and SD of", col)) +
    theme_minimal() +
    annotate("text", x = Inf, y = Inf, label = paste("Mean:", round
    (mean_value, 2), "\nSD:", round(sd_value, 2)),
             hjust = 1.1, vjust = 2, color = "black", size = 4)
})

do.call(grid.arrange, c(plots, ncol = 3))
```

Listing 1: intro til data

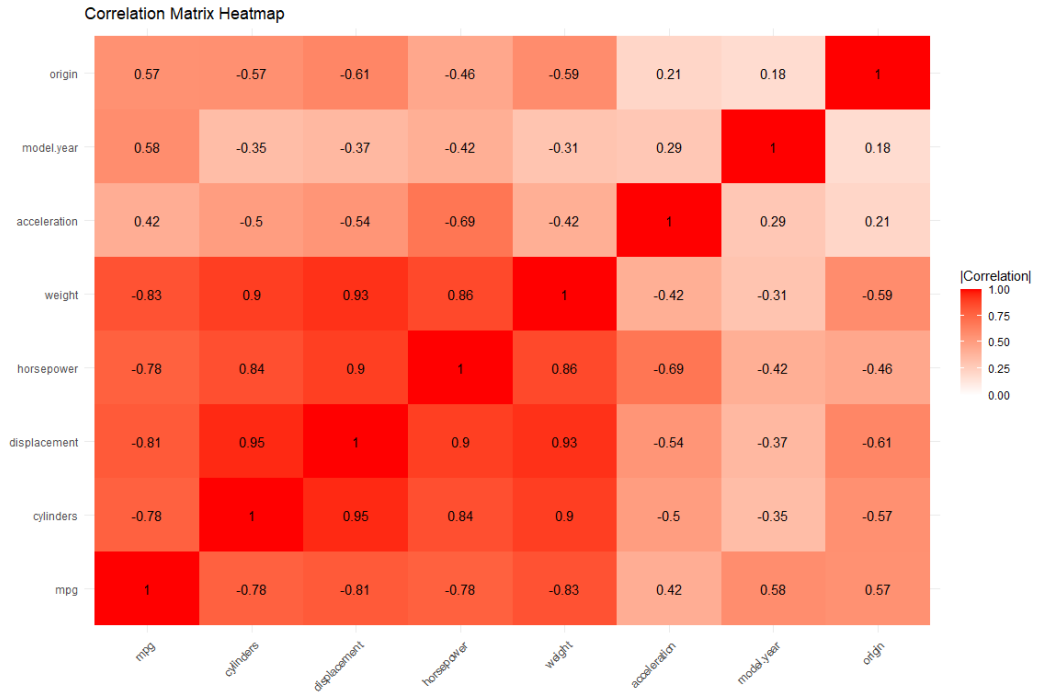

Figure 1: heatmap

10 Classical regression

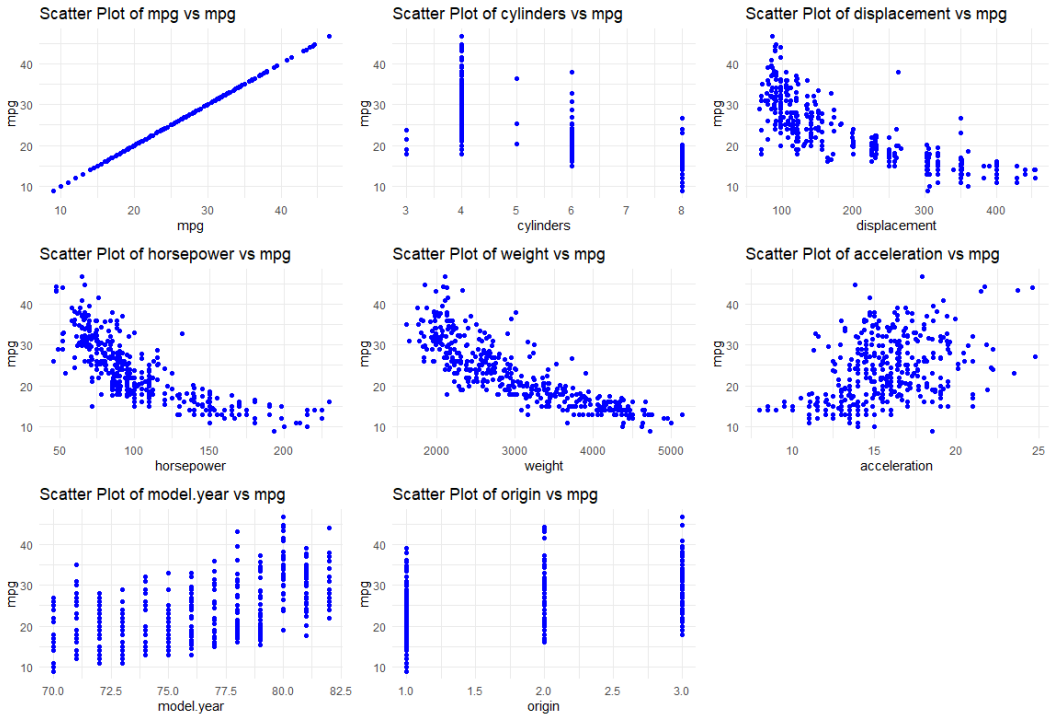

Figure 2: scatterplot

11 Monte Carlo regression

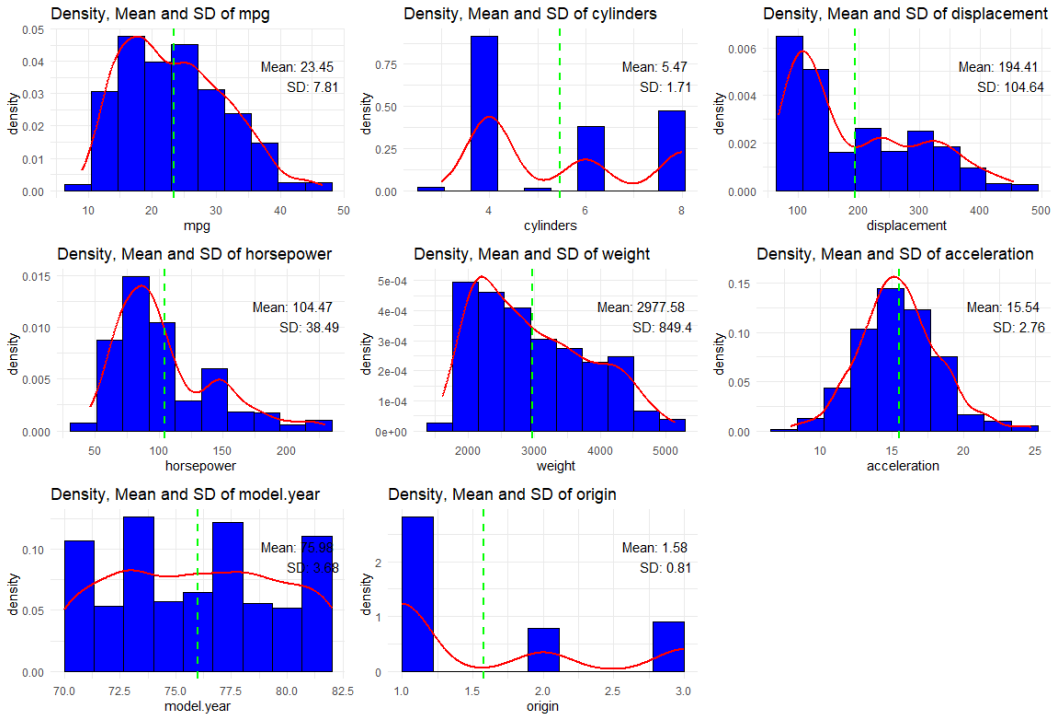

Figure 3: density,mean and sd

12 montecarlo bootstrapping

Monte Carlo bootstrapping is used to find the model for the mpg data. It starts by cleaning the data, then selecting the numeric columns, and defining the Monte Carlo bootstrapping function. After that, it defines the polynomial regression function using mpg as the dependent variable. Next, it defines the simulation function using clusters, thereby speeding up the computation time with the doParallel library. After defining the run simulation function, it runs the simulation with the set seed, simulation number, and sample size of the Monte Carlo bootstrapping method.

After that, it creates histograms of the simulation results. It calculates the mean result of the coefficients, and using these means, it calculates a new regression model and the R-squared of the new average model. Finally, it makes a scatter plot showing the final regression model results versus the results from the actual data set.

```
library(dplyr)
library(ggplot2)
library(foreach)
library(doParallel)
library(gridExtra)
```

```r

# Set working directory and read data
setwd("C:/Users/Jonathan/Documents/GitHub/P2/R kode")
data <- read.csv("auto-mpg.csv", na.strings = ".")

# Convert horsepower to numeric, coercing non-numeric values to NA
data$horsepower <- as.numeric(data$horsepower)

# Remove rows with any NA values
data <- data %>% na.omit()

# Select numeric columns
numeric_data <- data[sapply(data, is.numeric)]

# Function to perform Monte Carlo bootstrapping
monte_carlo_bootstrap <- function(data, sample_size) {
  data %>%
    sample_n(sample_size, replace = TRUE)
}

fit_polynomial_regression <- function(data) {
  formula <- as.formula(paste("mpg ~", paste(sapply(setdiff(names(
    data), "mpg"), function(var) paste("poly(", var, ", 2)")),
    collapse = " + ")))
  lm(formula, data = data)
}

run_simulations <- function(n_simulations, numeric_data, sample_
    size) {
  num_cores <- detectCores() - 1
  cl <- makeCluster(num_cores)
  registerDoParallel(cl)

  clusterExport(cl, c("numeric_data", "fit_polynomial_regression",
    "monte_carlo_bootstrap", "sample_size"))
  clusterEvalQ(cl, library(dplyr))

  results <- foreach(i = 1:n_simulations, .combine = rbind, .
    packages = "dplyr") %dopar% {
    resampled_data <- monte_carlo_bootstrap(numeric_data, sample_
    size)
    model <- fit_polynomial_regression(resampled_data)
    c(coef(model), summary(model)$r.squared)
  }

  stopCluster(cl)

  results_df <- as.data.frame(results)
  colnames(results_df) <- c(names(coef(fit_polynomial_regression(
    numeric_data))), "r_squared")

  return(results_df)
}

set.seed(200)
n_simulations <- 10000
sample_size <- 300
```

```r
results_df <- run_simulations(n_simulations, numeric_data, sample_size)

# Clean column names to remove special characters
clean_colnames <- function(df) {
  colnames(df) <- make.names(colnames(df), unique = TRUE)
  return(df)
}

results_df <- clean_colnames(results_df)

create_histograms <- function(df) {
  plots <- lapply(names(df), function(col) {
    mean_value <- mean(df[[col]])
    sd_value <- sd(df[[col]])

    ggplot(df, aes_string(x = col)) +
      geom_histogram(aes(y = ..density..), bins = 30, fill = "blue", color = "black", alpha = 0.7) +
      geom_density(color = "red", size = 1) +
      geom_vline(aes(xintercept = mean_value), color = "green", linetype = "dashed", size = 1) +
      ggtitle(paste("Density, Mean and SD of", col)) +
      theme_minimal() +
      annotate("text", x = Inf, y = Inf, label = paste("Mean:", round(mean_value, 2), "\nSD:", round(sd_value, 2)),
               hjust = 1.1, vjust = 2, color = "black", size = 4)
  })

  grid.arrange(grobs = plots, ncol = 2)
}

# Generate histograms for the simulation results
create_histograms(results_df)

# Calculate the mean of the coefficients from the Monte Carlo simulation
best_coefficients <- colMeans(results_df)

print(best_coefficients)

# Fit the final regression model using the best coefficients
final_model <- lm(mpg ~ ., data = numeric_data)

# Predict using the final model
y_final_pred <- predict(final_model, newdata = numeric_data)

# Calculate R-squared
actual_values <- numeric_data$mpg
ss_total <- sum((actual_values - mean(actual_values))^2)
ss_residual <- sum((actual_values - y_final_pred)^2)
r_squared <- 1 - (ss_residual / ss_total)

# Print R-squared value
cat("R-squared:", r_squared, "\n")

# Plot the final regression model
```

```r
ggplot(data.frame(Actual = actual_values, Predicted = y_final_pred)
    , aes(x = Actual, y = Predicted)) +
  geom_point(alpha = 0.7) +
  labs(title = paste("Final Regression Model (R-squared:", round(r_
    squared, 2), ")"),
       x = "Actual Values", y = "Predicted Values") +
  theme_minimal()
```

Listing 2: montecarlo bootstrapping

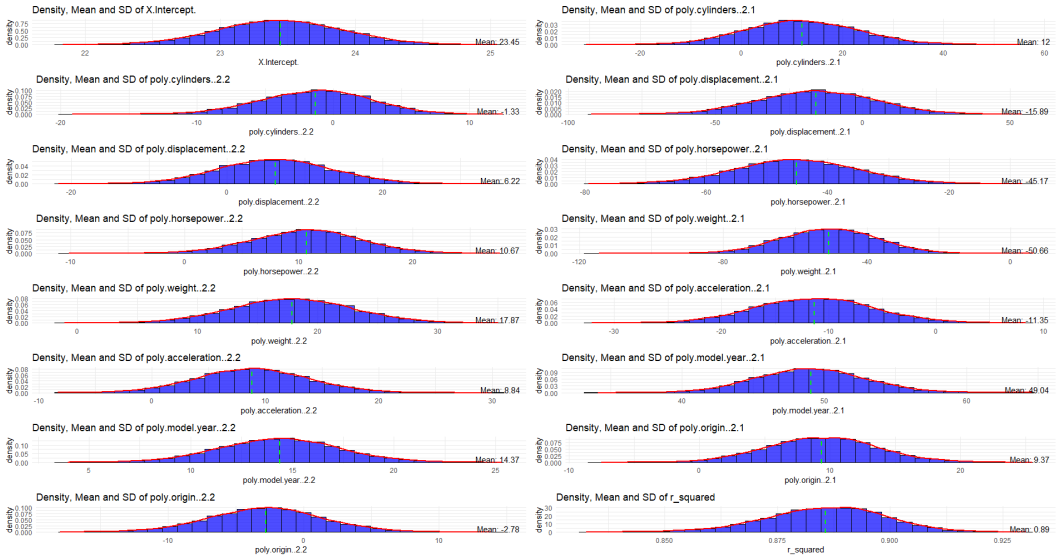

Figure 4: AQI gennem årene.

13 super syntetisk

14 super syntetisk

To show the consequences of violating the assumption of homoscedasticity, a dataset is generated with and without homoscedasticity. It starts with generating four normally distributed independent variables using a random number generator. Then, the dependent variable is generated as a function of the four independent variables. A regression model is fitted, and the model is tested. After this, homoscedasticity is violated by adding an error term that scales with the dependent variable. The model is tested again, and it becomes apparent what the consequences of violating the assumption of homoscedasticity are.

```r
library(ggplot2)
library(gridExtra)
library(lmtest)
```

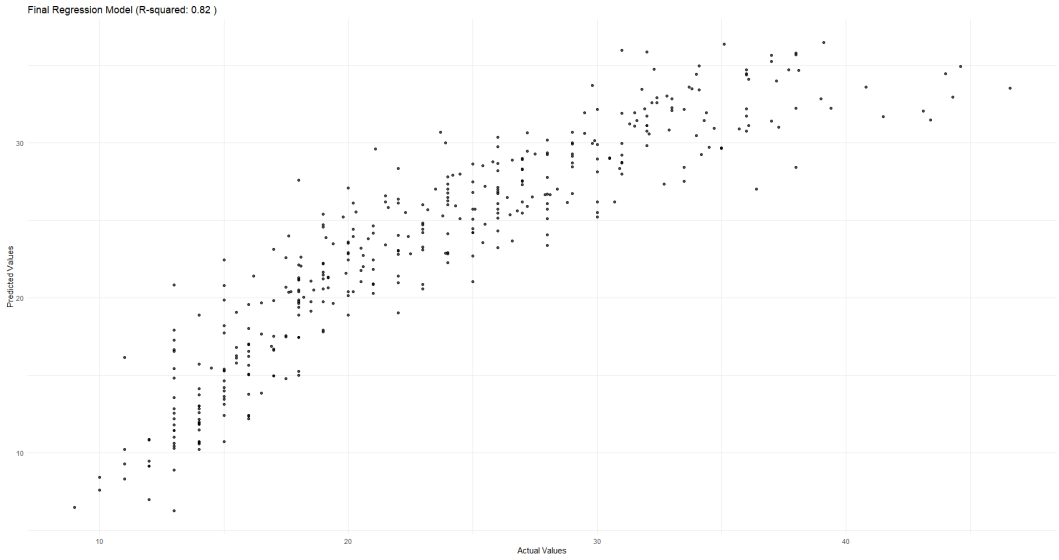

Figure 5: AQI gennem årene.

```r

# Set seed for reproducibility
set.seed(223)

# Generate independent variables
n <- 250
x1 <- rnorm(n, mean = 5, sd = 1)
x2 <- rnorm(n, mean = 10, sd = 4)
x3 <- rnorm(n, mean = 15, sd = 3)
x4 <- rnorm(n, mean = 20, sd = 5)

# Generate dependent variable with a polynomial relationship
y <- 3 + 2*x1 + 5*x1^2 + 1.5*x2 + 3*x3^2 + 2*x4 + 0.001*rnorm(n,
    mean = 0, sd = 1)

# Create a data frame
data <- data.frame(y, x1, x2, x3, x4)

# Fit a polynomial regression model
model <- lm(y ~ poly(x1, 2) + x2 + poly(x3, 2) + x4, data = data)

# Summary of the model
summary(model)

# Add an error term to x1 that scales with the corresponding y
    value
x3_new <- x3 + rnorm(n, mean = 0, sd = 0.00254 * abs(y))

# Generate new dependent variable with the same polynomial
    relationship
```

```r
y_new <- 3 + 2*x1 + 5*x1^2 + 1.5*x2 + 3*x3_new^2 + 2*x4 + 0.001*rnorm(n, mean = 0, sd = 1)

# Create a new data frame
data_new <- data.frame(y_new, x1 = x1, x2, x3_new, x4)

# Fit a new polynomial regression model
model_new <- lm(y_new ~ poly(x1, 2) + x2 + poly(x3, 2) + x4, data = data_new)

# Summary of the new model
summary(model_new)

# Diagnostic plots for the original model
par(mfrow = c(2, 2))
plot(model)

# Diagnostic plots for the new model
par(mfrow = c(2, 2))
plot(model_new)

# Function to create scatter plots for each numeric column against 'y'
scatter_plots <- function(data, color, title_prefix) {
  lapply(names(data)[-1], function(col) {
    ggplot(data, aes_string(x = "y", y = col)) +
      geom_point(color = color) +
      ggtitle(paste(title_prefix, col, "vs y")) +
      theme_minimal()
  })
}

# Scatter plots for normal data
scatter_plots_normal <- scatter_plots(data, "blue", "Normal Data: Scatter Plot of")
# Scatter plots for non-homoscedastic data
scatter_plots_non_homoscedastic <- scatter_plots(data_new, "red", "Non-Homoscedastic Data: Scatter Plot of")

# Arrange scatter plots in grids
grid.arrange(grobs = scatter_plots_normal, ncol = 2, top = "Scatter Plots for Normal Data")
grid.arrange(grobs = scatter_plots_non_homoscedastic, ncol = 2, top = "Scatter Plots for Non-Homoscedastic Data")

# Function to create histograms for each numeric column
create_histograms <- function(data, color, title_prefix) {
  lapply(names(data), function(col) {
    n_bins <- ceiling(log2(length(data[[col]])) + 1)
    ggplot(data, aes_string(x = col)) +
      geom_histogram(bins = n_bins, fill = color, color = "black") +
      ggtitle(paste(title_prefix, col)) +
      theme_minimal()
  })
}

```

```r
# Histograms for normal data
histograms_normal <- create_histograms(data, "blue", "Histogram of")
# Histograms for non-homoscedastic data
histograms_non_homoscedastic <- create_histograms(data_new, "red", "Histogram of")

# Arrange histograms in grids
do.call(grid.arrange, c(histograms_normal, ncol = 3, top = "Histograms for Normal Data"))
do.call(grid.arrange, c(histograms_non_homoscedastic, ncol = 3, top = "Histograms for Non-Homoscedastic Data"))

# Breusch-Pagan test for the original model
bp_test <- bptest(model)
print(bp_test)
bp_test <- bptest(model_new)
print(bp_test)
```

Listing 3: super syntetisk data

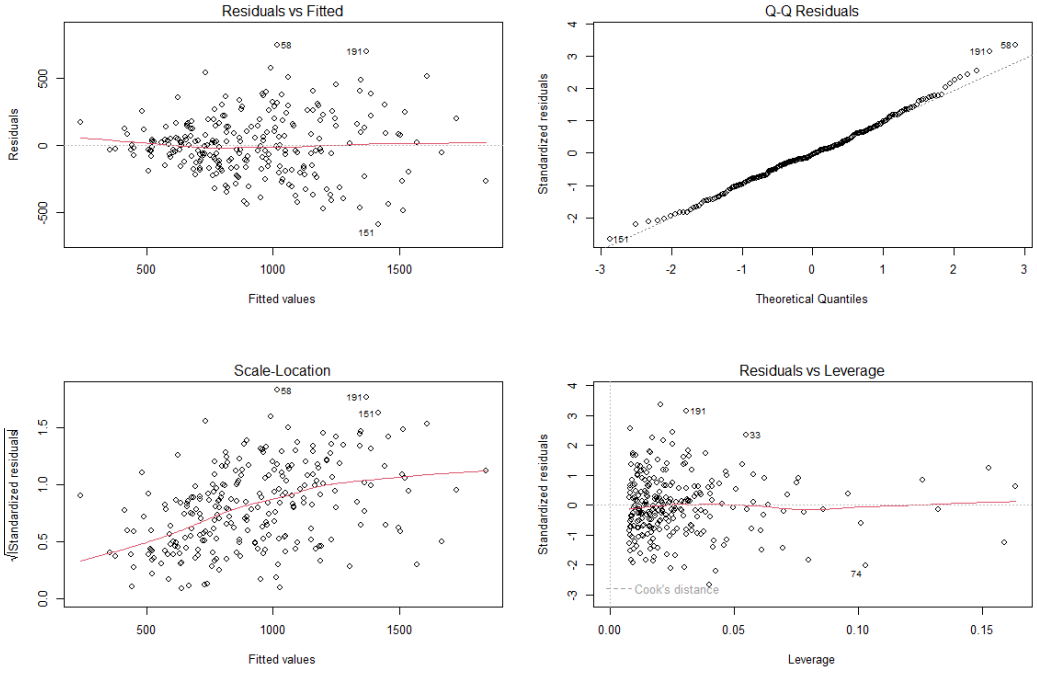

Figure 6: residualplot

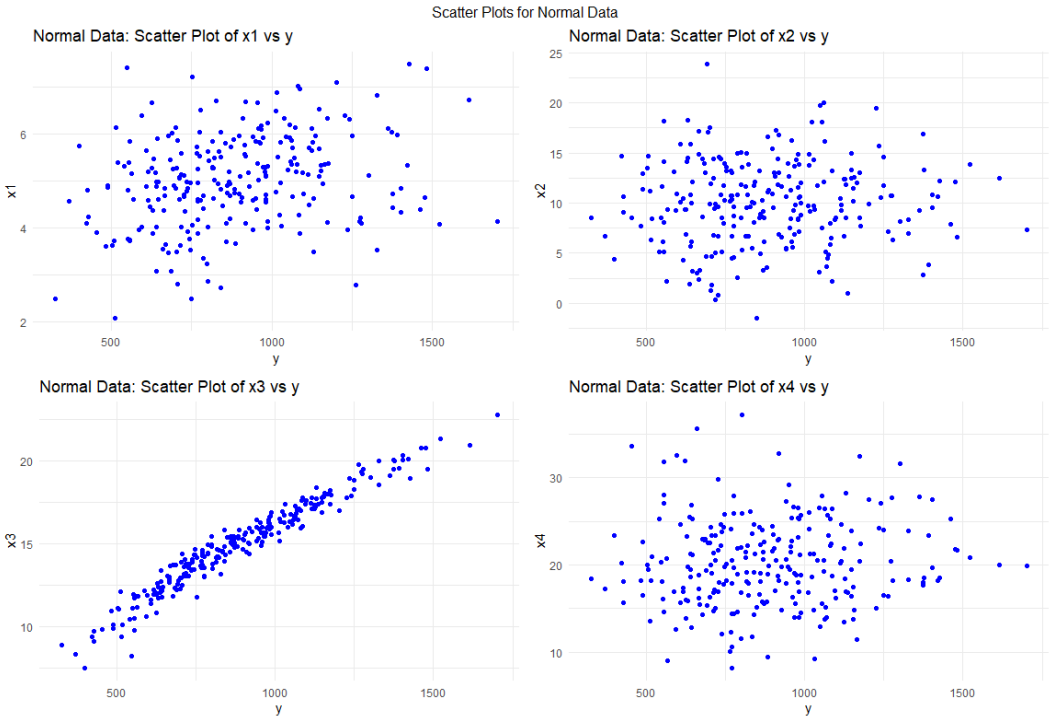

Figure 7: scatterplot.homo

15 Comparison between regressions

Figure 8: scatterplot.hetro

16 Discussion

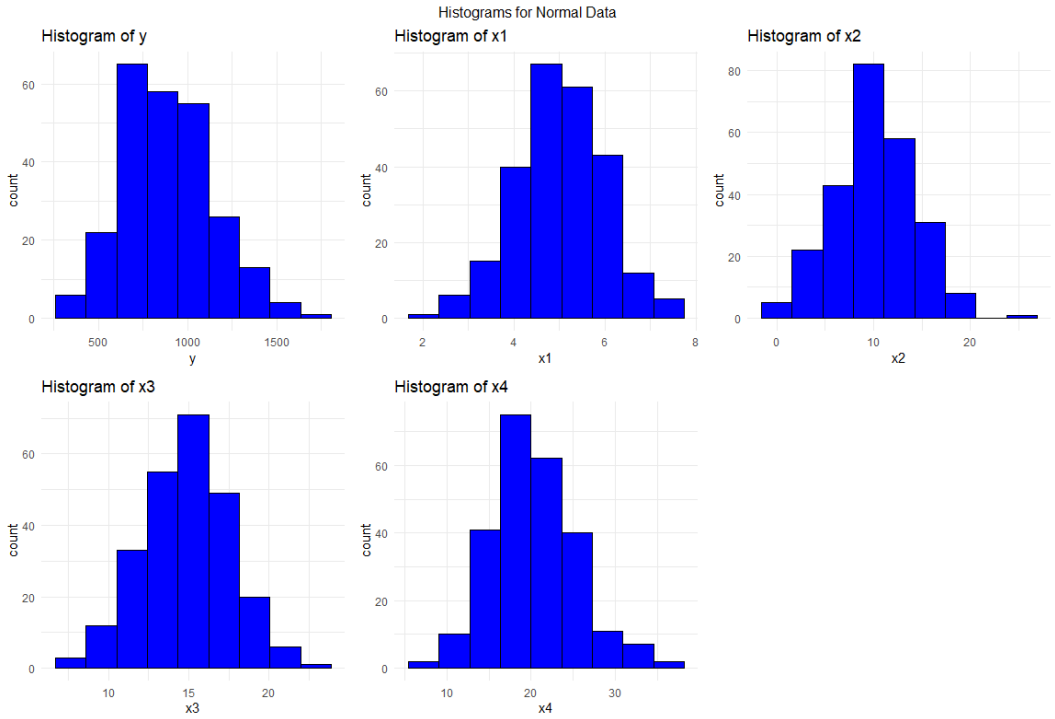

Figure 9: his.homo

17 Conclusion

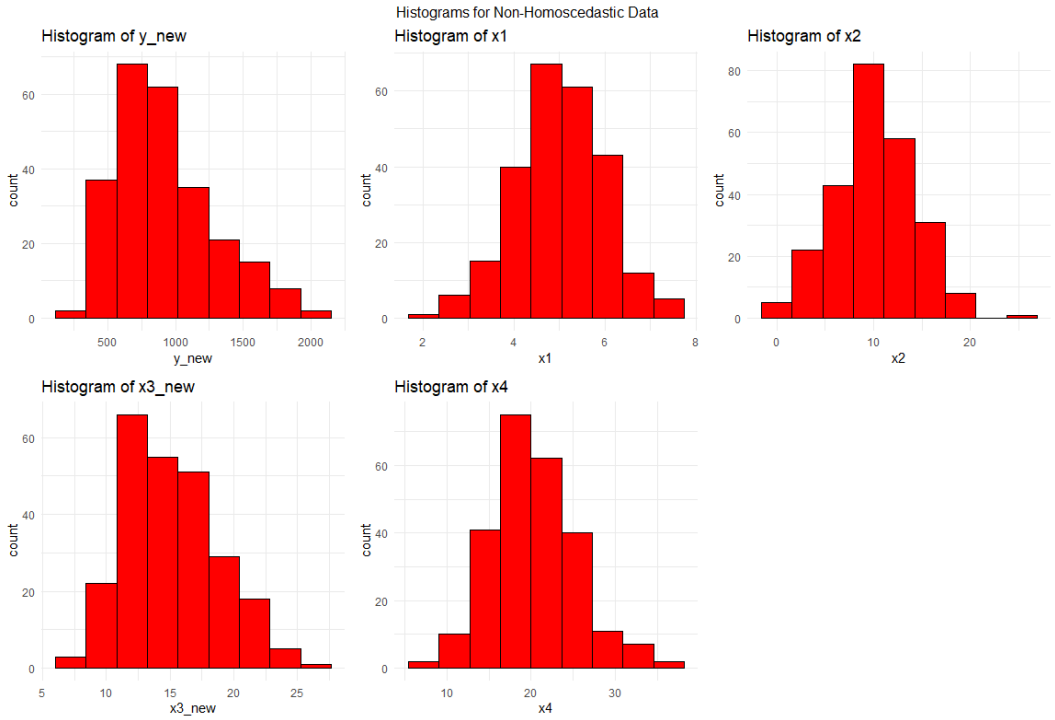

Figure 10: his.hetro.

18 Litteratur

```
        studentized Breusch-Pagan test

data:  model
BP = 4.2058, df = 6, p-value = 0.6488
```

Figure 11: model 1

```
        studentized Breusch-Pagan test

data:  model_new
BP = 36.704, df = 6, p-value = 2.011e-06
```

Figure 12: model 2

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  8.316e+02  2.860e-04 2907330   <2e-16 ***
poly(x1, 2)1 7.972e+02  9.132e-04  872931   <2e-16 ***
poly(x1, 2)2 1.075e+02  9.156e-04  117390   <2e-16 ***
x2           1.500e+00  1.389e-05  107964   <2e-16 ***
poly(x3, 2)1 3.830e+03  9.111e-04 4203108   <2e-16 ***
poly(x3, 2)2 4.874e+02  9.167e-04  531678   <2e-16 ***
x4           2.000e+00  1.161e-05  172241   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0009082 on 243 degrees of freedom
Multiple R-squared:      1,     Adjusted R-squared:      1
F-statistic: 3.229e+12 on 6 and 243 DF,  p-value: < 2.2e-16
```

Figure 13: model opsumering 1

```
Residuals:
    Min      1Q  Median      3Q     Max
-589.70 -156.03   -9.31  139.96  752.05

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   794.644     71.380  11.133  < 2e-16 ***
poly(x1, 2)1  864.038    227.890   3.791 0.000189 ***
poly(x1, 2)2 -174.610    228.488  -0.764 0.445490
x2              7.574      3.467   2.184 0.029889 *
poly(x3, 2)1 4247.971    227.373  18.683  < 2e-16 ***
poly(x3, 2)2  632.462    228.767   2.765 0.006135 **
x4              2.414      2.898   0.833 0.405550
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 226.6 on 243 degrees of freedom
Multiple R-squared:  0.6169,    Adjusted R-squared:  0.6074
F-statistic: 65.22 on 6 and 243 DF,  p-value: < 2.2e-16
```

Figure 14: model opsumering 2