Statistical Learning Final Report

Alberto Calabrese, Eleonora Mesaglio, Greta d'Amore Grelli

2024-06-13

Contents

1 Introduction

In this project, we will conduct a thorough analysis of a dataset of our choice, to gain a full understanding of our data, and we will then build models that enable us to make accurate predictions. The dataset we chose is called *Starbucks Beverage Components* and contains information about the ingredients of Starbucks' beverages.

We will go through several steps, including Data Cleaning, Exploratory Data Analysis (EDA) and Regression Analysis. Indeed, first we will prepare our data for analysis by handling missing values and ensuring that our data is correctly formatted. Once our data is clean, we will proceed to the EDA stage, where we will avail ourself of visual and quantitative methods to understand the structure of our data and the relationships

between variables. Finally, we will perform Regression Analysis to understand the relationship between our dependent and independent variables. This will allow us to make predictions about our data and understand the factors that influence our dependent variable, `"Calories"`.

2 Data

The dataset we will analyze in this project is *Starbucks Beverage Components* from Kaggle, that you can find at the following link: https://www.kaggle.com/datasets/henryshan/starbucks.

This data provides a comprehensive guide to the nutritional content of the beverages available on the Starbucks menu. We have a total of 242 samples described by 18 variables. These attributes include the name of the beverage, its categorization and preparation method, the total caloric content and the constituents of the beverage.

```
data <- read.csv("Data/starbucks.csv", header = TRUE, sep = ",")
```

2.1 Data Transformation

Note that several variables in our dataset, namely `"Vitamin.A....DV."`, `"Vitamin.C....DV."`, `"Calcium....DV."` and `"Iron....DV."`, are represented as percentages. Consequently, the percentage symbol is included in our data. However, when conducting statistical analysis using R, the presence of non-numeric characters such as the percentage symbol can cause complications, interfering with the processing and analysis of the data. Therefore, we proceed to remove it.

Similarly, as R primarily operates on numeric and categorical data, we also convert all the other numerical variables into numeric format.

These preprocessing steps ensure a smooth and efficient analysis, making it easier to explore, visualize, and understand our data.

```
# Remove percentage sign from the data
data$Vitamin.C....DV. <- as.numeric(gsub("%", "", data$Vitamin.C....DV.))
# Set the other variables as numeric
data$Calories <- as.numeric(data$Calories)
```

2.2 Data Cleaning

Another challenge we have to face is the presence of missing data. Indeed, in `"Caffeine..mg."` column there are some `NA` values. This is a common issue in data analysis and needs to be addressed appropriately to ensure the validity of our statistical results.

One way to deal with these unwanted `NA` values is to omit the samples containing them from our study. This guarantees that our analysis is conducted solely on complete and dependable data. Alternatively, we can fill them in with the average or the median of the observed values for that specific attribute. This second method helps to preserve the overall data distribution while addressing the missing data points.

In our work, we opt for the latter approach, replacing `NA` values with the median. This choice is particularly suitable for our data, which is skewed and contains outliers. Indeed, the median, being a measure of central tendency that is not affected by extreme values, provides a more robust replacement in the presence of outliers.

```
summary(data$Caffeine..mg.)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.00   50.00   75.00   89.52  142.50  410.00      23
```

```
# Replace NA values with the median
data_cleaned <- data
```

```r
data_cleaned$Caffeine..mg.[is.na(data_cleaned$Caffeine..mg.)] <- median(
  data_cleaned$Caffeine..mg., na.rm = TRUE)
# Summary of the Caffeine column after cleaning
summary(data_cleaned$Caffeine..mg.)
```

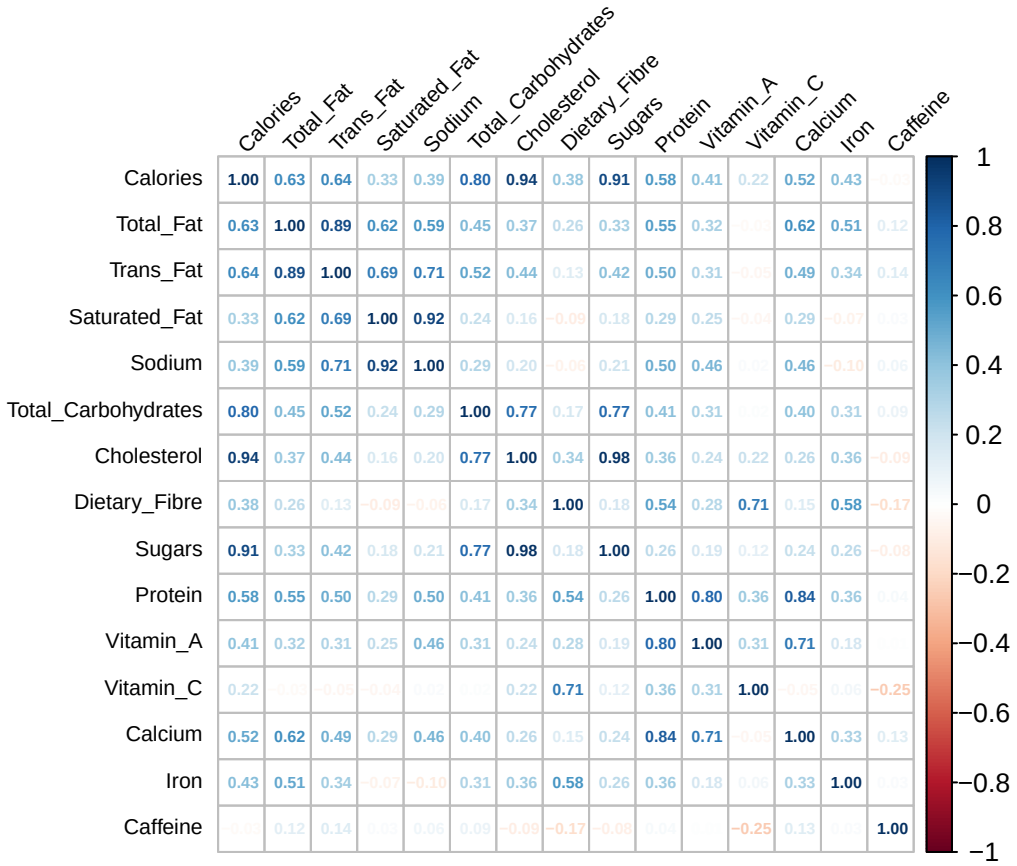
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   70.00   75.00   88.14  130.00  410.00
```

Lastly, taking in consideration our cleaned data, we renamed the columns by removing dots and units of measure, in order to obtain a more readable dataset.

3 Correlation Analysis

After completing these preliminary preprocessing steps, we calculate the correlation matrix for our dataset. This computation helps us in comprehending the interrelationships among the dataset's variables. In the correlation matrix, a value near to 1 at the ij position indicates a strong positive correlation between the i-th and j-th variables. Conversely, a value close to -1 signifies a strong negative correlation. A value near 0 suggests that the two variables do not significantly influence each other.

Observe that the first three columns of our data are categorical features, thus for these we cannot compute Pearson's correlation coefficient. In the following code lines we remove them to compute and plot such matrix.



4 Data Visualization

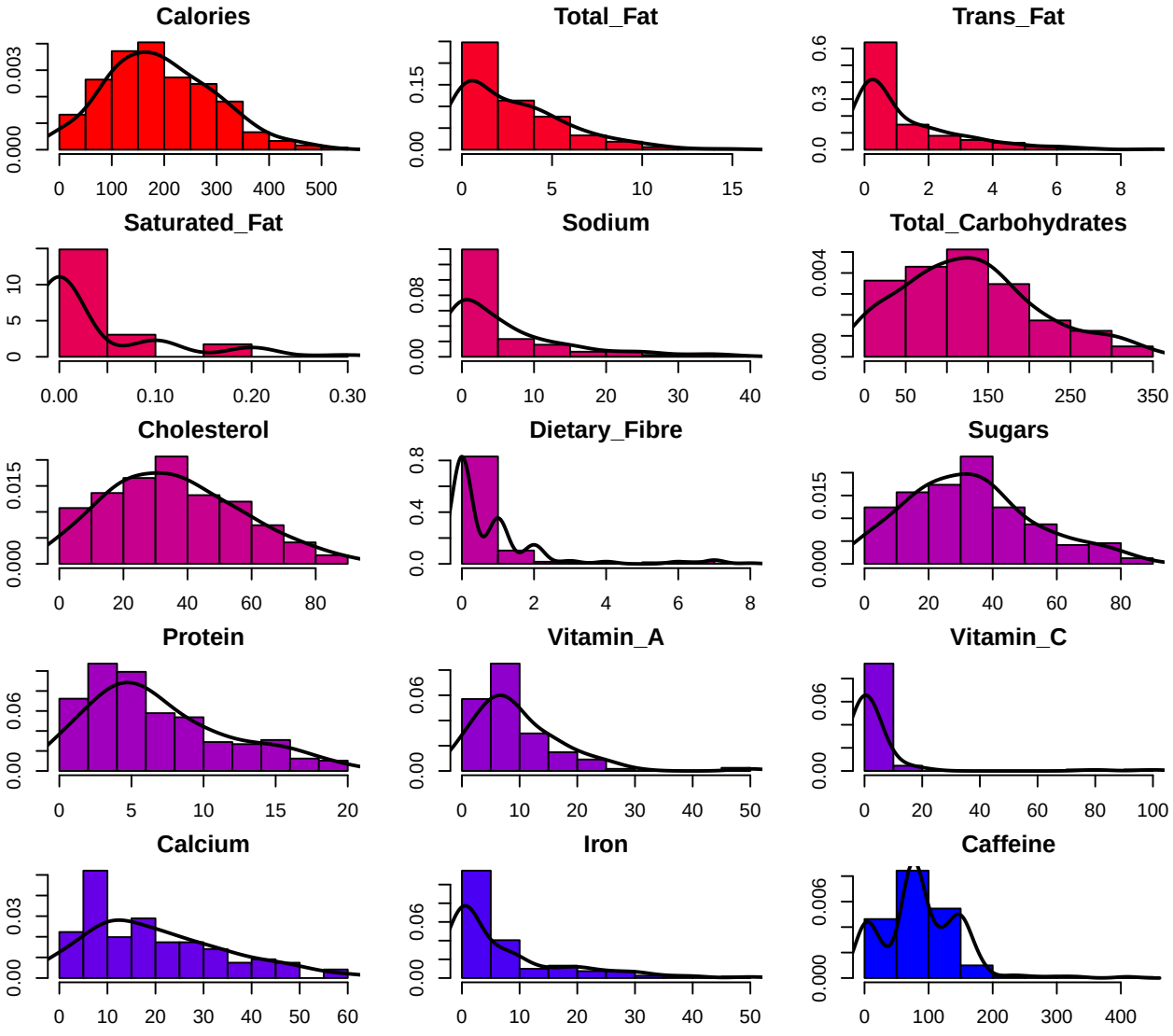
Data visualization is a powerful tool that allows us to uncover patterns, correlations and outliers in our data. It provides visual information on the dataset in our analysis, representing large amounts of data in a clear

and comprehensive way and underlining the relationships among them. This enables us to recognize patterns quickly.

So, let us transform our raw data into graphical representations, to gain a more comprehensive understanding of the information at hand.

4.1 Histograms

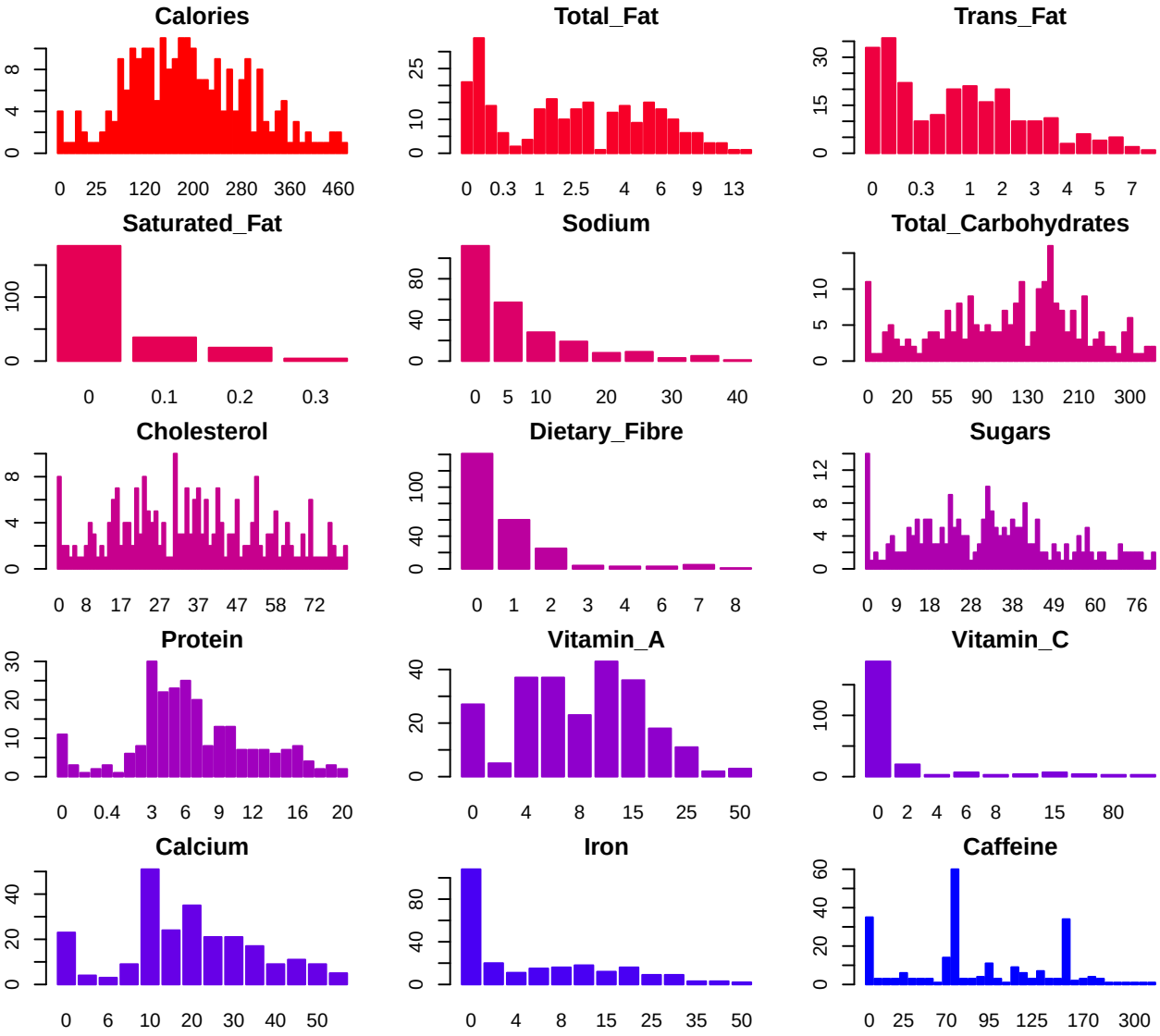
Histograms serve as a graphical interpretation of data distribution. In a histogram, each bar corresponds to the counted frequency within each bin or interval. We introduce these plots to see if our data is normally distributed, skewed, or has outlier values.



By looking at the graphs, we can notice that the variables `"Calories"`, `"Total_Carbohydrates"`, `"Cholesterol"`, and `"Sugars"` exhibit distributions that are nearly normal. Conversely, the distributions of the remaining variables display a noticeable skewness towards the left.

4.2 Barplot

We will now plot the bar plots for our dataset. The primary use of bar plots is to make comparisons between the amounts of different categories. Indeed, each bar corresponds to a category and the height of the bar represents the frequency or proportion of that category. These graphs are commonly used for categorical data, or numerical data that has been binned into categories.



We can deduce some useful information by looking at these plots.

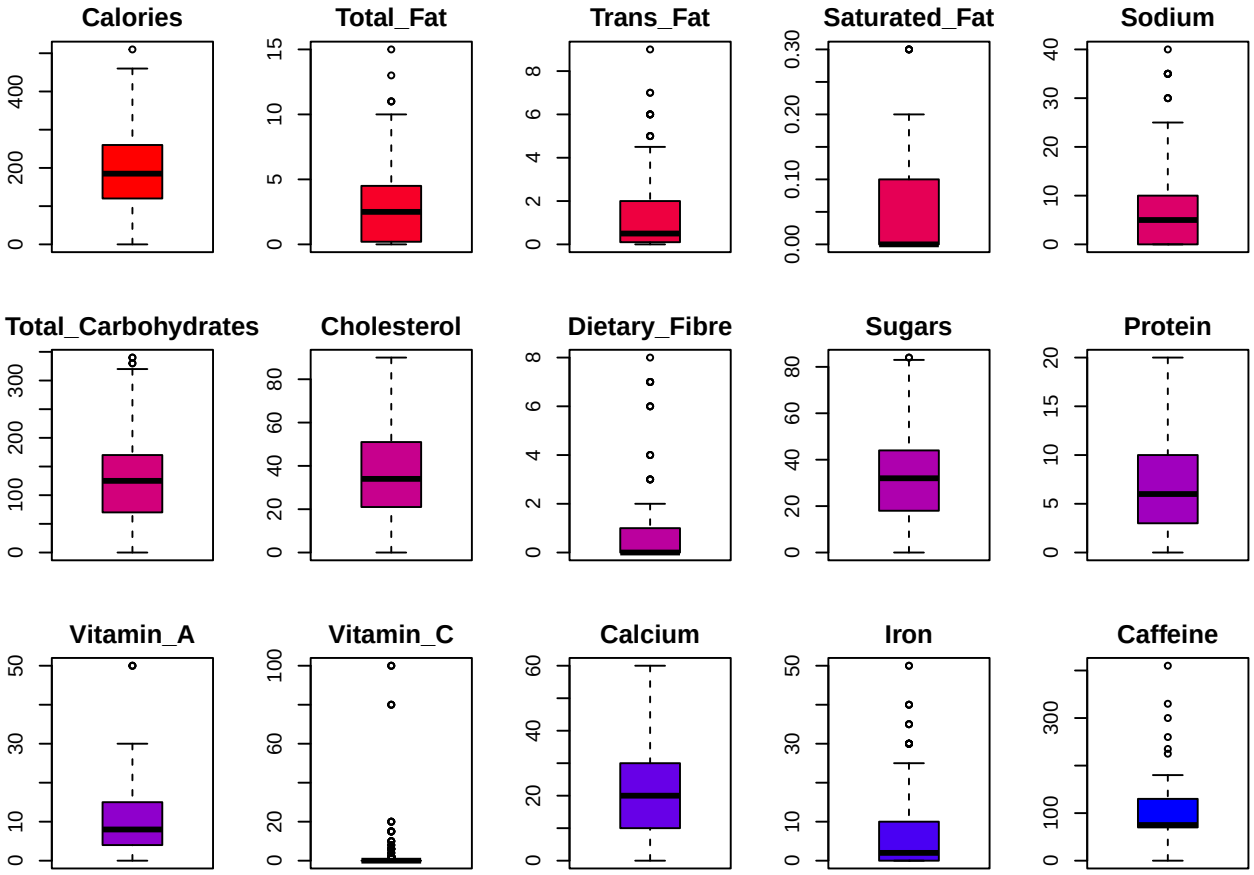
For example, we can notice that variables such as `"Saturated_Fat"`, `"Dietary_Fibre"`, `"Vitamin_C"`, and `"Iron"` are typically either absent or present in small quantities in the beverages. In particular, the frequency of these variables rapidly diminishes as their levels increase. On the other hand, the variables `"Calories"`, `"Total_Fat"`, `"Trans_Fat"`, and `"Total_Carbohydrates"` show a wide range of values across different beverage types, going from high levels in some beverages to minimal amounts in others.

We can further observe that the distribution of `"Vitamin_A"` appears to be more evenly spread among the different levels in various beverages, while instead `"Caffeine"` plot is interesting as it exhibits three distinct peaks in frequency.

4.3 Boxplot

Boxplots are a type of graphical representation used to display the distribution of a dataset. They provide a visual summary of the data, enabling us to quickly identify key statistical measures such as median, quartiles and outliers. This visualization also helps us to determine the spread and variability of the data.

In this section, we create the boxplots of our dataset.



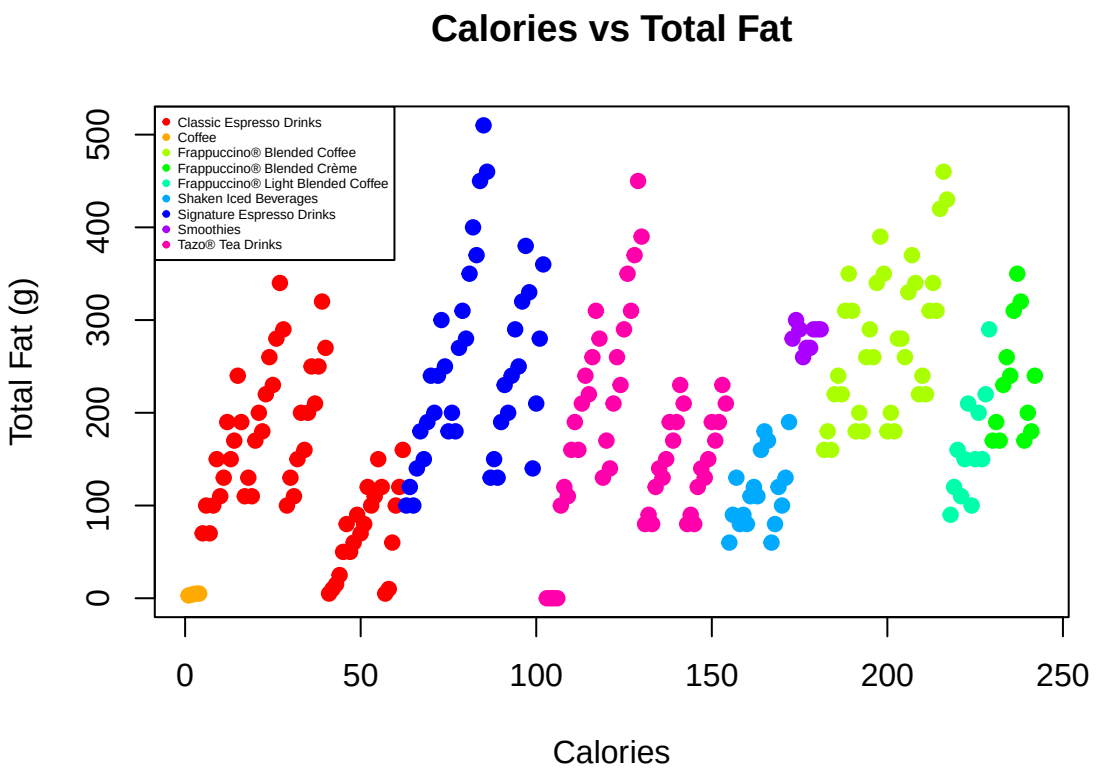
As we observed earlier, the majority of the graphs exhibits a skewness towards zero, with the exceptions being `"Calories"`, `"Total_Carbohydrates"`, `"Cholesterol"`, and `"Sugars"`.

Another aspect that has not been previously highlighted is the presence of outliers. These are notably evident in `"Dietary_Fiber"`, `"Vitamin_C"`, and `"Caffeine"` plots.

4.4 Scatterplot

A scatterplot is a type of data visualization that uses dots to represent the values obtained for two different variables - one plotted along the x-axis and the other plotted along the y-axis. Scatterplots are used to observe relationships between variables. This type of graphical representation is crucial in detecting underlying patterns and potential correlations among the variables.

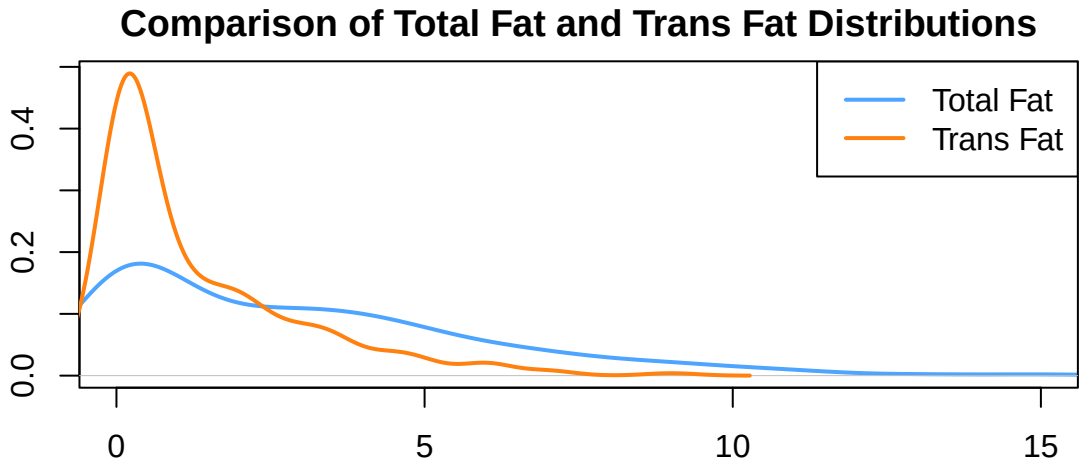
In particular, we place the calorie content and fat levels of various beverage categories side by side for comparison. To make the visualization more intuitive, we assign distinct colors to each beverage category and create a legend to identify each category.



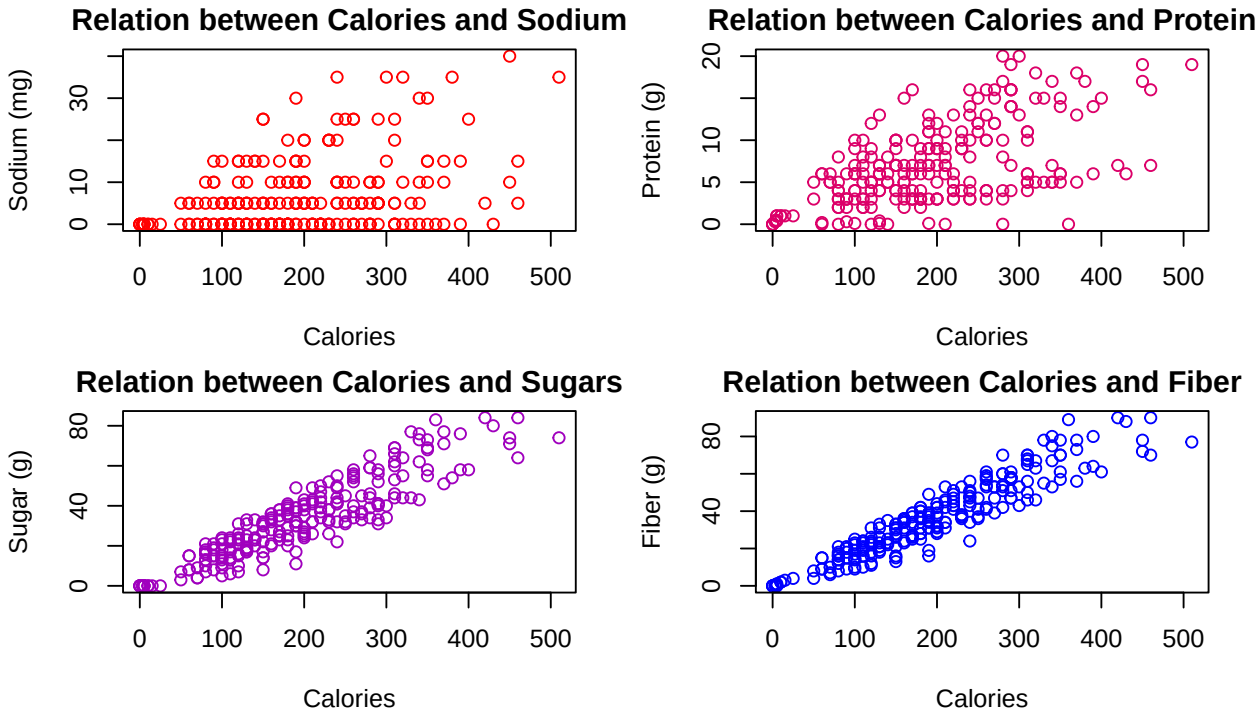
Let us look for any overall pattern or trend in the data points. For all the categories the two variables seem to be related following an almost linear trend, with a positive correlation - as one variable increases, so does the other. However, it is important to note that a very high `"Total_Fat"` value does not necessarily equate to a very high `"Calories"` value - see `"Espresso Signature Drinks"` category.

Additionally we can observe that, given a specific category, it is possible that we find two clusters of data points that follow distinct distibutions. This phenomenon is observed in the `"Classic Espresso Drinks"`, `"Espresso Signature Drinks"` and `"Tazo Tea Drinks"` categories and it is likely attributable to the diverse methods of drink preparation.

In the next plot we can also see a comparison between `"Total_Fat"` and `"Trans_Fat"` distributions:



Finally, we create some scatterplots to look into relantionship between `"Calories"` and other variables. In particular we focus on `"Sodium"`, `"Protein"`, `"Sugars"` and `"Dietary_Fiber"`.



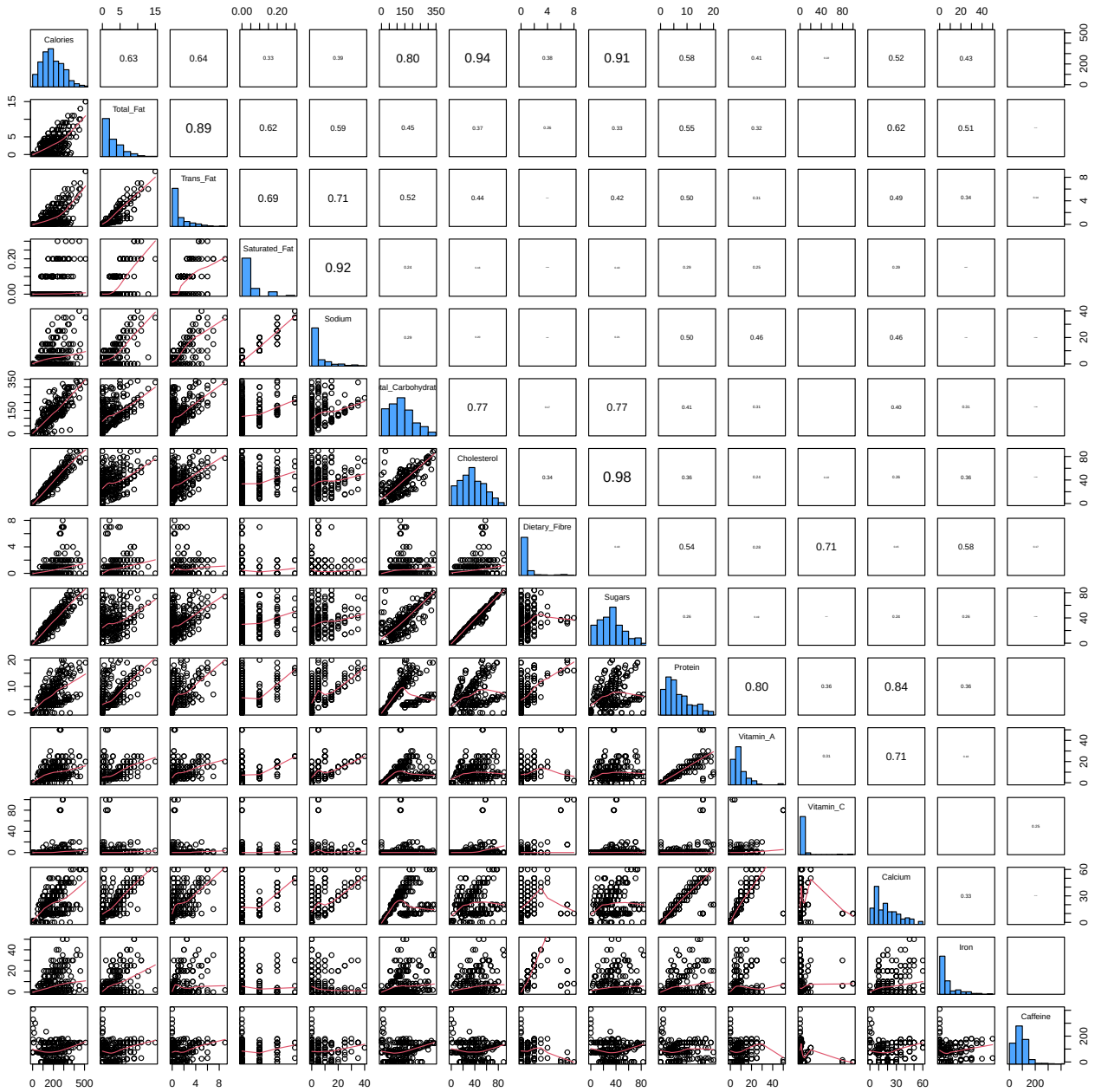
With an increase in calories, we observe a corresponding rise in all features. However, the rate of increase varies among different features. For instance, `"Sugars"` and `"Dietary_Fiber"` show a steep ascent, indicating a rapid increase with calorie count. On the other hand, `"Sodium"` and `"Protein"` exhibit a more gradual growth, suggesting a slower rate of increase despite the rising calorie content. These observations are further substantiated by the correlation coefficients, which provide a quantitative measure of these relationships. This highlights the complex interplay between calories and various nutritional components in our beverages.

4.5 Pairplot

A pairplot, as the name suggests, is a plot that enables us to visualize the pairwise relationships between different variables in a dataset. It is essentially a matrix of scatterplots, where each scatterplot shows the relationship between a pair of variables. This type of visualization is particularly useful for exploring potential relationships and correlations between all the variables. As previously noted, by examining the scatterplots we can identify patterns, trends, and outliers in the data.

Before we can create a pairplot, we need to define a some functions for it. This functions will generate the scatterplots, as well as additional elements such as histograms, correlation coefficients, and a smooth line to help visualize the distribution and correlation of the data.

Finally, we create the pairplot using the defined functions.



ADD COMMENTS ON THE GRAPH (I don't know how to comment it)

5 Regression Analysis

In this section we will conduct a comprehensive regression analysis on our dataset, exploring our data, constructing different models and lastly performing model selection and validation. Our goal is to build a model that accurately represents the relationships within our data and can provide meaningful predictions. In particular, the variable we want to predict is `"Calories"`. To achieve this, we will first fit a linear regression model to predict the amount of calories based on the amount of the other variables. We will then compare different models, evaluate their performance, and select the best model for our data.

5.1 Linear Regression

The simplest form of regression analysis is linear regression, where we predict an outcome variable based on one or more predictor variables.

Linear regression model to predict the amount of calories based on the amount of the other variables We use the `lm()` function to fit a linear regression model to predict the amount of calories based on the amount of the other variables in the dataset. We then evaluate the model using various metrics such as AIC, BIC, R-squared, and adjusted R-squared.

5.1.1 Simple Linear Regression

Fit linear simple regression with just one variable on `data_cleaned`, looking at correlation plot we choose `Sugars` due to high correlation.

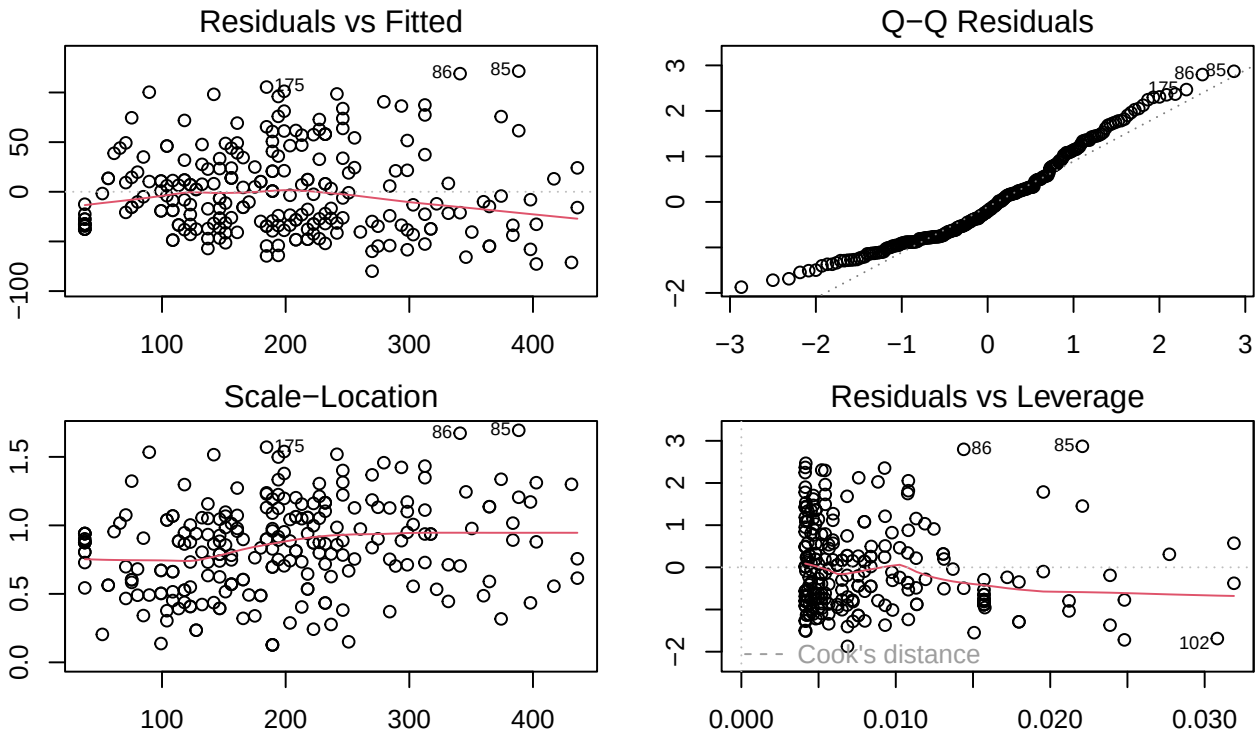
This code will fit a simple linear regression model predicting `"Calories"` using `"Sugars"` as the predictor variable and provide a summary of the model.

```
lm_simple <- lm(Calories ~ Sugars, data = data_cleaned)
kable(data.frame(AIC = AIC(lm_simple), BIC = BIC(lm_simple),
                 R_squared = summary(lm_simple)$r.squared,
                 adj_R_squared = summary(lm_simple)$adj.r.squared),
      caption = "Model evaluation metrics for the simple linear regression model")
```

Table 1: Model evaluation metrics for the simple linear regression model

AIC	BIC	R_squared	adj_R_squared
2509.036	2519.503	0.8275094	0.8267907

```
par(mfrow = c(2, 2), mar = c(2, 2, 2, 2))
plot(lm_simple)
```

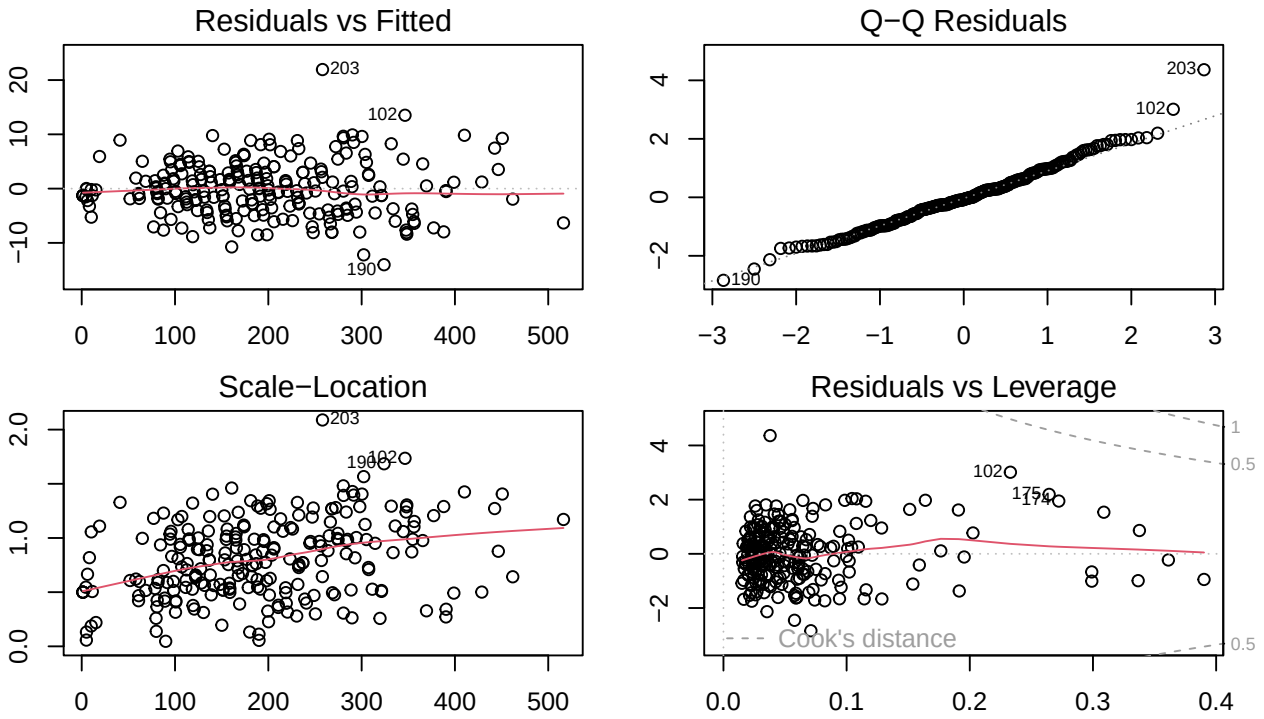


The coefficient for "Sugars" (4.7426) indicates that, on average, for every one-unit increase in "Sugars", the predicted "Calories" increases by approximately 4.7426 units. Both the intercept and the coefficient for "Sugars" are statistically significant ($p < 0.001$), indicating a strong linear relationship between "Sugars" and "Calories". The F-statistic is highly significant ($p < 2.2e-16$), indicating that the overall regression model is statistically significant in explaining the variance in "Calories". Model Fit:The adjusted R-squared value (0.8268) indicates that approximately 82.68% of the variance in "Calories" can be explained by the predictor variable "Sugars". Overall, this output suggests that the simple linear regression model provides a statistically significant relationship between "Sugars" and "Calories", with "Sugars" being a strong predictor of "Calories". However, the AIC and BIC values suggest that there might be other models that provide a better fit for the data.

Summarizing the model is too simple so it doesn't capture the complexity of the data, so we try to fit a multiple linear regression model.

5.1.2 Multiple Linear Regression

```
lm_model <- lm(y ~ ., data = data_num_)
par(mfrow = c(2, 2), mar = c(2, 2, 2, 2))
plot(lm_model)
```




```
kable(data.frame(AIC = AIC(lm_model), BIC = BIC(lm_model),
                 R_squared = summary(lm_model)$r.squared,
                 adj_R_squared = summary(lm_model)$adj.r.squared),
      caption = "Model evaluation metrics for the linear regression model")
```

Table 2: Model evaluation metrics for the linear regression model

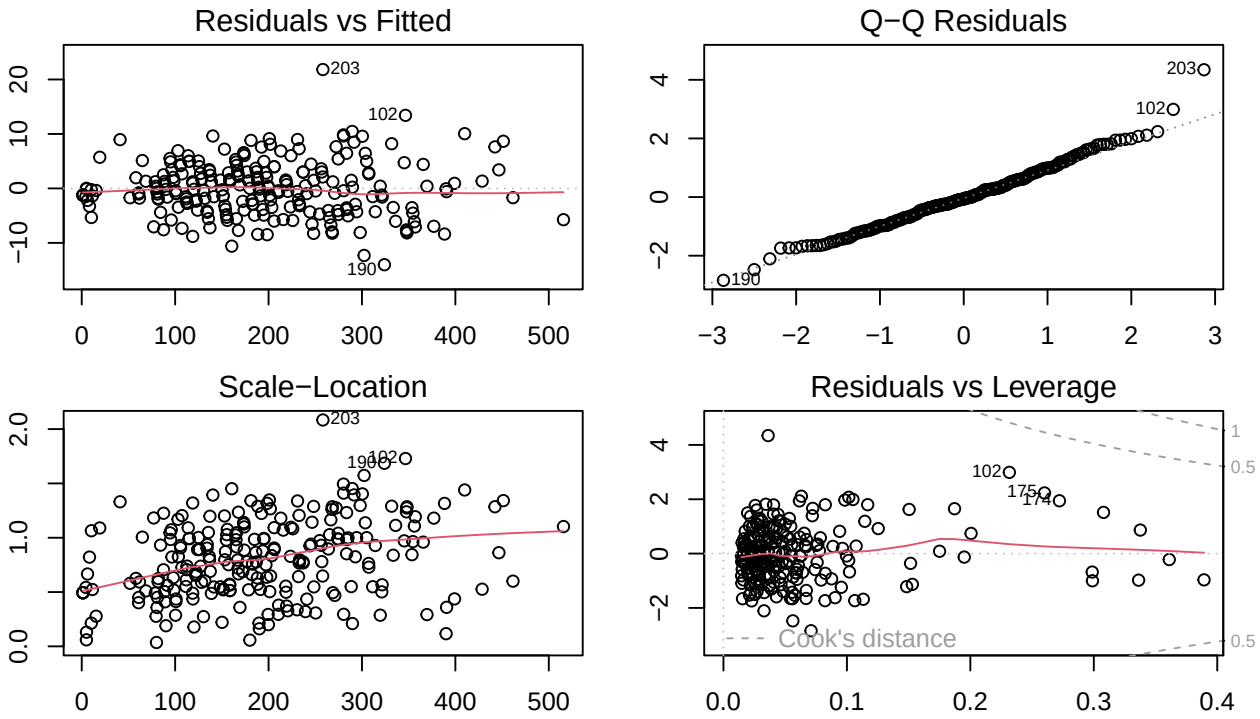
AIC	BIC	R_squared	adj_R_squared
1494.304	1550.127	0.9976608	0.9975166

The model has a low AIC and BIC values, the R-squared value is 0.997 so the model is a good fit for the data.

5.1.3 Backward Elimination

Now we apply the selection of the predictors with the backward elimination method. This method starts with all the predictors in the model and then removes the least significant predictor one at a time until all remaining predictors are significant.

```
par(mfrow = c(2, 2), mar = c(2, 2, 2, 2))
plot(backward_model)
```




```
kable(data.frame(AIC = AIC(backward_model), BIC = BIC(backward_model),
                 R_squared = summary(backward_model)$r.squared,
                 adj_R_squared = summary(backward_model)$adj.r.squared),
      caption = "Model evaluation metrics for the linear regression model
      with backward elimination")
```

Table 3: Model evaluation metrics for the linear regression model with backward elimination

AIC	BIC	R_squared	adj_R_squared
1492.616	1544.95	0.9976578	0.9975243

The backward selection drops only the variable `"Saturated_Fat"` since it's not considered significant in explaining the amount of calories mantaining the other variables.

Comarison between the models:

Table 4: Model comparison

Model	AIC	BIC	R_squared	adj_R_squared
Simple Linear Regression	2509.036	2519.503	0.8275094	0.8267907
Multiple Linear Regression	1494.304	1550.127	0.9976608	0.9975166
Multiple Linear Regression with Backward Elimination	1492.616	1544.950	0.9976578	0.9975243

The multiple linear regression model with backward elimination has the lowest AIC and BIC values, the highest R-squared value, and the highest adjusted R-squared value, indicating that it is the best model for predicting the amount of calories based on the amount of the other variables.

Coefficients: Both models have very similar coefficients for the variables that were retained. The removal of `"Saturated_Fat"` in the backward model did not significantly affect the estimates of the other coefficients.

Significance of Variables: In the full model, `"Saturated_Fat"` had a high p-value (0.589), indicating it was not a significant variable. In the backward model, `"Saturated_Fat"` was removed, slightly improving the AIC while keeping all other variables significant.

Overall Performance: Both models perform very similarly in terms of R-squared and residual standard error. The backward model is preferable because it has a slightly lower AIC, suggesting it is a more parsimonious model without sacrificing the quality of the fit.

5.1.4 Anova

Anova comparison between the models

```
anova_results <- anova(lm_model, backward_model)
anova_results
```

```
## Analysis of Variance Table
##
## Model 1: y ~ Total_Fat + Trans_Fat + Saturated_Fat + Sodium + Total_Carbohydrates +
##     Cholesterol + Dietary_Fibre + Sugars + Protein + Vitamin_A +
##     Vitamin_C + Calcium + Iron + Caffeine
## Model 2: y ~ Total_Fat + Trans_Fat + Sodium + Total_Carbohydrates + Cholesterol +
##     Dietary_Fibre + Sugars + Protein + Vitamin_A + Vitamin_C +
##     Calcium + Iron + Caffeine
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    227 5964.8
## 2    228 5972.5 -1   -7.6917 0.2927  0.589
```

Degrees of Freedom (Res.Df): The full model has 227 degrees of freedom, while the backward model has 228. This is because we removed one variable from the full model.

Residual Sum of Squares (RSS): The full model has an RSS of 5964.83, while the backward model has an RSS of 5972.55. This indicates that the difference between the two models in terms of residual error is very small.

Sum of Squares (Sum of Sq): The difference between the two models in terms of sum of squares is -7.7235, indicating that the removed variable (`"Saturated_Fat"`) does not significantly contribute to explaining the variability in calories.

F-statistic (F): The F value is 0.2937 with a p-value of 0.588. This high p-value indicates that there is no significant difference between the two models. In other words, the reduced model is not significantly worse than the full model.

Conclusion: The ANOVA shows that the removal of the `"Saturated_Fat"` variable does not have a significant impact on the model. This confirms that the model obtained through backward selection is more parsimonious without compromising the quality of the fit. Therefore, the backward model is preferable to the full model.

5.1.5 Multicollinearity

To check for multicollinearity, we calculate the Variance Inflation Factors (VIF) for the variables in the multiple linear regression model, it measures how much the variance of the estimated coefficients is increased due to multicollinearity.Usually a VIF value greater than 10 indicates a problematic amount of multicollinearity.

Table 5: VIF values for the linear regression model

	VIF
Total_Fat	17.863697
Trans_Fat	14.667324

	VIF
Sodium	4.448925
Total_Carbohydrates	3.419094
Cholesterol	442.886703
Dietary_Fibre	16.896773
Sugars	417.769822
Protein	56.706156
Vitamin_A	4.205667
Vitamin_C	4.288442
Calcium	37.105615
Iron	5.027804
Caffeine	1.176323

However as we can see from the *Table X* e have a problem with multicollinearity, the VIF values are high for some variables, so we have to act on the data to solve this problem

The high values of the VIF could be due to:

- High correlation between variables,means that variable contribuite in the same way to predict and explain calories
- Same information
- Data unbalanced
- Different Measurement Scales: If the variables in the model have significantly different measurement scales such as g and mg this could affect the VIF values.
- Non-linear Relationships: If the relationships between the variables are non-linear, this could also affect the VIF values.

In this case, normalizing the variables might help reduce multicollinearity.

5.1.6 Standardize the data

We have tried different kind of standardization to reduce the multicollinearity First at all we tried the scale by standard normalization The negative value of the AIC (-748.2623) indicates that the standardized linear regression model provides a better compromise between data fit and model complexity compared to the reference model. However, the VIF values are still high, indicating that multicollinearity is still present in the model.

To reduce the problem of high VIF in linear regression, it is generally preferable to use the transformation that includes both log transformation and standardization of the data. This is because standardization helps to put all variables on the same scale, reducing the likelihood of multicollinearity. Log transformation: Reduces the variance of the variables, making the distribution more normal and reducing the impact of outliers. Standardization: Puts all variables on a common scale, with mean 0 and standard deviation 1, further reducing multicollinearity.

```
std_data_log <- scale(log(data_num + 1)) # Standardize the data
std_data_log_df <- as.data.frame(std_data_log) # Set as dataframe
mod_log_tr <- lm(Calories ~ ., data = std_data_log_df)
kable(data.frame(AIC = AIC(mod_log_tr), BIC = BIC(mod_log_tr),
                 R_squared = summary(mod_log_tr)$r.squared,
                 adj_R_squared = summary(mod_log_tr)$adj.r.squared),
      caption = "Model evaluation metrics for the log transformed data")
```

Table 6: Model evaluation metrics for the log transformed data

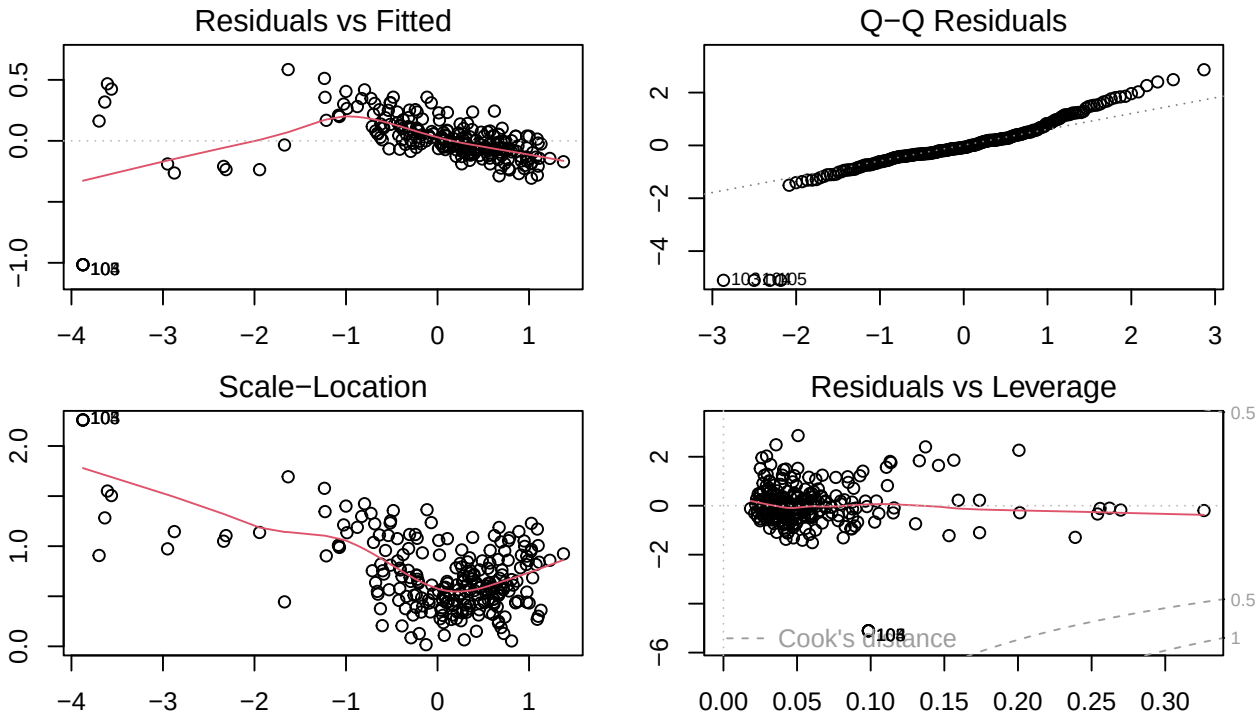
AIC	BIC	R_squared	adj_R_squared
-53.42411	2.398897	0.9586932	0.9561457

```
kable(data.frame(VIF = vif(mod_log_tr)),
      caption = "VIF values for the log transformed data")
```

Table 7: VIF values for the log transformed data

	VIF
Total_Fat	12.049669
Trans_Fat	10.577306
Saturated_Fat	4.528080
Sodium	5.817088
Total_Carbohydrates	4.363628
Cholesterol	39.988684
Dietary_Fibre	7.115085
Sugars	38.415586
Protein	31.007121
Vitamin_A	13.647581
Vitamin_C	2.196674
Calcium	25.873742
Iron	4.582594
Caffeine	1.310005

```
par(mfrow = c(2, 2), mar = c(2, 2, 2, 2))
plot(mod_log_tr)
```



The model has a low AIC and BIC values, the R-squared value is 0.95 so the model is a good fit for the data. However we have still collinearity, so we try to use backward elimination to check if this method will removes the variables that are not significant in the model.

```
kable(data.frame(AIC = AIC(backward_model_log), BIC = BIC(backward_model_log),
                 R_squared = summary(backward_model_log)$r.squared,
                 adj_R_squared = summary(backward_model_log)$adj.r.squared),
      caption = "Model evaluation metrics for the log transformed data
      with backward elimination")
```

Table 8: Model evaluation metrics for the log transformed data with backward elimination

AIC	BIC	R_squared	adj_R_squared
-63.78109	-32.38065	0.9580667	0.9568123

```
kable(data.frame(VIF = vif(backward_model_log)),
      caption = "VIF values for the log transformed data with backward elimination")
```

Table 9: VIF values for the log transformed data with backward elimination

	VIF
Total_Fat	5.823786
Trans_Fat	5.161252
Total_Carbohydrates	3.390622
Cholesterol	36.628698
Dietary_Fibre	1.851534
Sugars	33.948827
Protein	2.976444

AIC slightly worst and the VIF values are still high, indicating that multicollinearity is still present in the model. So we try to remove manually the variables that has high VIF values.

```
mod_log_tr_updated <- lm(Calories ~ . - Cholesterol - Sugars,
                         data = std_data_log_df)
kable(data.frame(VIF = vif(mod_log_tr_updated)),
      caption = "VIF values for the log transformed data with
      manual removal of variables")
```

Table 10: VIF values for the log transformed data with manual removal of variables

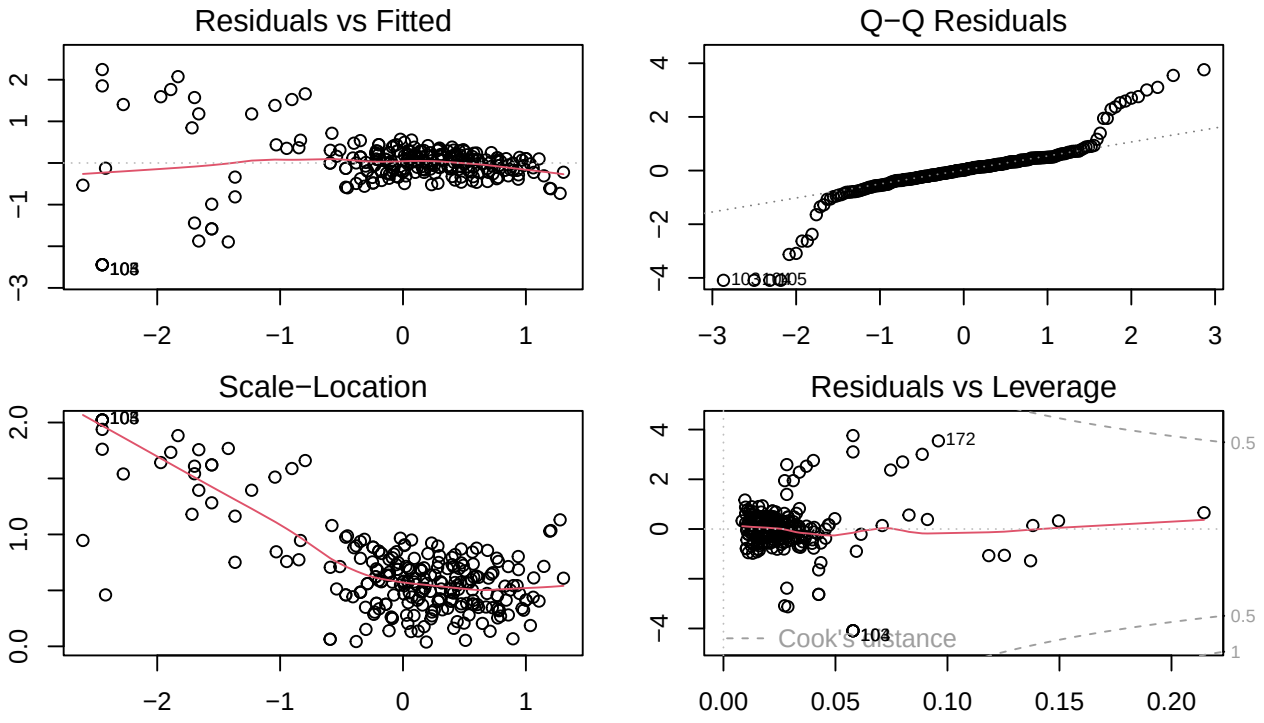
	VIF
Total_Fat	11.902918
Trans_Fat	10.114112
Saturated_Fat	4.466975
Sodium	5.782843
Total_Carbohydrates	3.375194
Dietary_Fibre	7.080360
Protein	26.902392
Vitamin_A	12.396739
Vitamin_C	1.985799
Calcium	25.519022
Iron	4.521552
Caffeine	1.295121

```
kable(data.frame(AIC = AIC(mod_log_tr_backward_2), BIC = BIC(mod_log_tr_backward_2),
                 R_squared = summary(mod_log_tr_backward_2)$r.squared,
                 adj_R_squared = summary(mod_log_tr_backward_2)$adj.r.squared),
      caption = "Model evaluation metrics for the log transformed data with
      backward elimination and manual removal of variables")
```

Table 11: Model evaluation metrics for the log transformed data with backward elimination and manual removal of variables

AIC	BIC	R_squared	adj_R_squared
460.5536	488.4651	0.6309185	0.6214952

```
par(mfrow = c(2, 2), mar = c(2, 2, 2, 2))
plot(mod_log_tr_backward_2)
```




```
kable(data.frame(VIF = vif(mod_log_tr_backward_2)),
      caption = "VIF values for the log transformed data with backward
      elimination and manual removal of variables")
```

Table 12: VIF values for the log transformed data with backward elimination and manual removal of variables

	VIF
Trans_Fat	1.491986
Total_Carbohydrates	2.649737
Protein	10.478245
Vitamin_A	8.772514
Vitamin_C	1.241962
Iron	1.304571

The VIF values are now below 10, indicating that multicollinearity has been reduced in the model. The R-squared value is decreased but it is still good.

Model Diagnostics: Non-normal residuals suggest that some assumptions of linear regression might be violated. Specifically, the assumption of normality of the residuals is not met, this can affect the validity of hypothesis tests on the coefficients and predictions.

```
shapiro.test((residuals(mod_log_tr_backward_2)))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  (residuals(mod_log_tr_backward_2))
## W = 0.82171, p-value = 5.816e-16
```

Given the p-value is significantly smaller than 0.05, we reject the null hypothesis. This indicates that the residuals of the model `mod_log_tr_backward_2` do not follow a normal distribution. In this case, W is quite a

bit lower than 1, suggesting the residuals deviate from normality. Sol Robust Methods: Use robust regression methods that do not assume normality of errors.

We have tried other trasformation like min-max scaling and robust scaling but not satisfactory due to VIF still to high. Regularization: Using regularization methods such as ridge regression or lasso regression penalizes the coefficients of variables, helping to reduce multicollinearity.

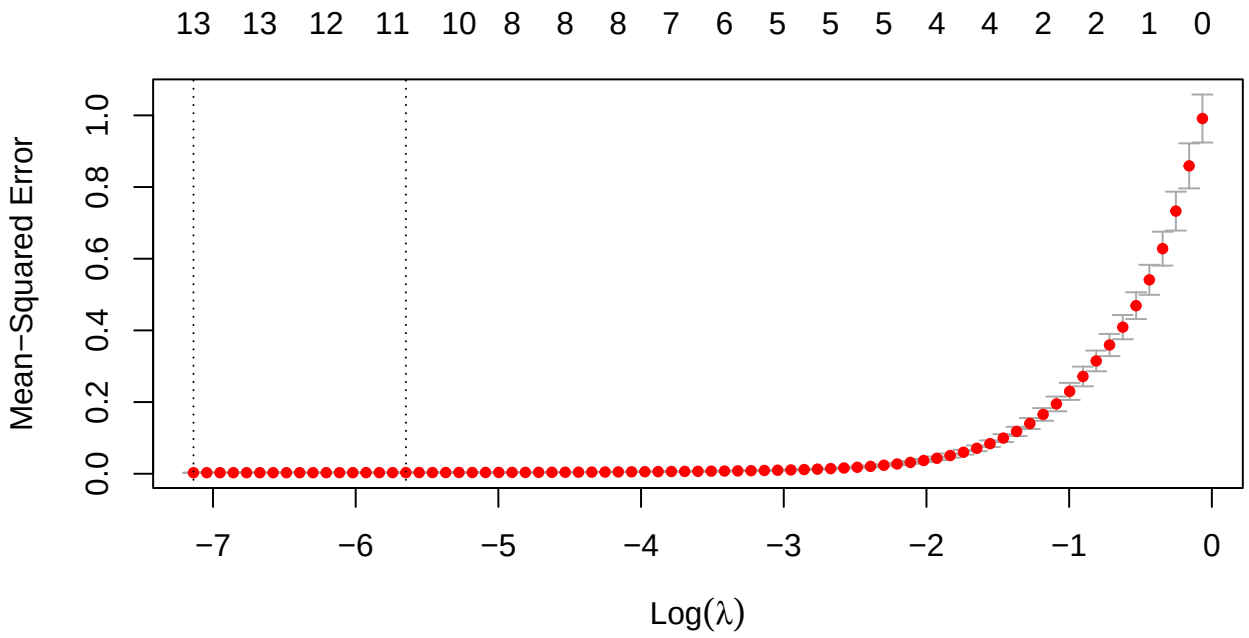
5.2 Lasso Regression

We use the `glmnet` package to fit a lasso regression model. Lasso regression, a type of linear regression that employs L1 regularization, penalizes the model's coefficients. This approach helps prevent overfitting and identifies the most significant features in the data.

First, we standardize the data and then fit the lasso regression model using the `cv.glmnet` function. Cross-validation is employed to select the optimal lambda value for the model. The lambda value that minimizes the mean squared error (MSE) is chosen as the optimal value, which is then used to fit the final lasso regression model.

Lasso regression tends to shrink the coefficients of less important variables towards zero, effectively performing variable selection. By eliminating irrelevant variables, it reduces the number of predictors and, consequently, multicollinearity. Lasso regression often produces sparse solutions by driving many coefficients to exactly zero. This reduction in variables decreases multicollinearity among predictors, resulting in lower Variance Inflation Factor (VIF) values. The automatic feature selection inherent in lasso regression removes redundant variables and reduces multicollinearity in the model.

```
std_data <- as.data.frame(scale(data_num)) # Standardize the data
mod_lasso <- cv.glmnet(x = as.matrix(std_data[, -1]),
                       y = std_data$Calories, alpha = 1, standardize = FALSE)
par(mfrow = c(1, 1))
plot(mod_lasso, xvar = "lambda", label = TRUE)
```



By setting some coefficients to zero (such as `"Saturated_Fat"`), lasso regression aids in feature selection, thereby reducing the model's complexity. The remaining non-zero coefficients indicate the variables that significantly contribute to predicting calories. The signs and magnitudes of these coefficients illustrate the direction and strength of their relationships with the target variable (`"calories"`). For example, a one-unit

increase in sodium, holding all other variables constant, is associated with a decrease of approximately 0.021 calories.

The `cv.glmnet` function performs cross-validation to determine the lambda value that minimizes the prediction error, identified as `lambda.min`. In our case, we found that the optimal lambda value is equal to 1.

To further evaluate the model's performance, metrics such as R-squared and Mean Squared Error (MSE) should be considered. These metrics help in understanding how well the model explains the variance in the data and the average error of the predictions, respectively. Additionally, the lambda value that minimizes the MSE is selected as the optimal lambda value.

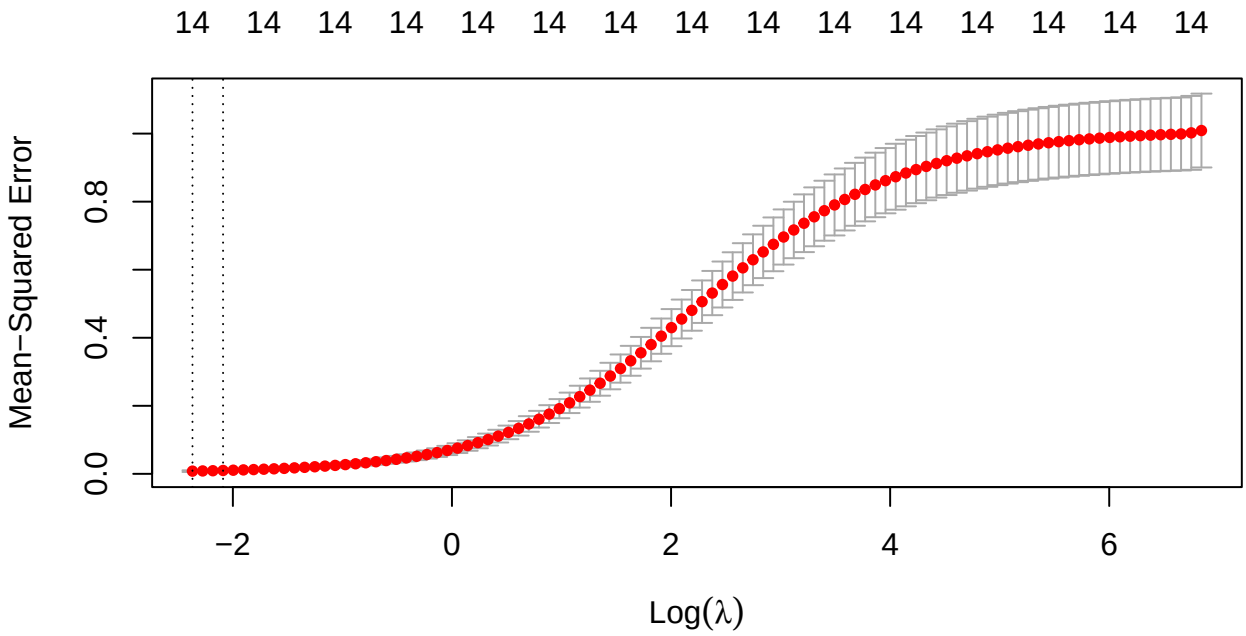
5.3 Ridge Regression

We also tried ridge regression to reduce multicollinearity. Like lasso regression, ridge regression is a regularization technique that introduces a penalty term to shrink coefficients towards zero without eliminating them entirely, meaning that all variables can remain in the model. Unlike lasso, which can zero out some coefficients and thus perform variable selection, ridge regression retains all variables, making it suitable for reducing multicollinearity without excluding any variables. Lasso may be preferred when selecting a subset of the most relevant variables is desired.

We use the `glmnet` package to fit a ridge regression model. Ridge regression, a type of linear regression, uses L2 regularization to penalize the coefficients of the model. This approach helps prevent overfitting and reduces the impact of collinearity in the data.

The ridge regression model is fit using the `cv.glmnet()` function, which incorporates cross-validation to determine the optimal lambda value. Similar to lasso, the best lambda value, which minimizes the error, was found to be equal to 1. The interpretation of the coefficients is the same as in lasso regression.

```
mod_ridge <- cv.glmnet(x = as.matrix(std_data[, -1]),
                       y = std_data$Calories, alpha = 0, standardize = FALSE)
plot(mod_ridge, xvar = "lambda", label = TRUE)
```



5.4 Model Comparison

We compare the linear regression, lasso regression, and ridge regression models to select the best model for predicting the amount of calories based on the amount of the other variables. We evaluate the models using the R-squared value, and the Mean Squared Error (MSE) for each model.

The R-squared value is a measure of how well the model fits the data, it ranges from 0 to 1, with higher values indicating a better fit

```
lasso_pred <- predict(mod_lasso, s = "lambda.min", newx = as.matrix(std_data[, -1]))
lasso_r_squared <- cor(lasso_pred, std_data$Calories)^2
ridge_pred <- predict(mod_ridge, s = "lambda.min",  newx = as.matrix(std_data[, -1]))
ridge_r_squared <- cor(ridge_pred, std_data$Calories)^2
kable(data.frame(Model = c("Linear Regression", "Lasso Regression","Ridge Regression"),
                 R_squared = c(summary(lm_model)$r.squared,
                               lasso_r_squared, ridge_r_squared)),
      caption = "R-squared values for the models")
```

Table 13: R-squared values for the models

Model	R_squared
Linear Regression	0.9976608
Lasso Regression	0.9975756
Ridge Regression	0.9941815

Comment on Lasso: The R-squared value of approximately 0.998 indicates that the Lasso regression model explains about 99.76% of the variance in the Calories variable. This suggests a very strong fit, as the model is capturing almost all the variability in the target variable.

Comment on Ridge: Similar to LASSO, very high

5.5 Model Evaluation

We evaluate the performance of the linear regression, lasso regression, and ridge regression models using the mean squared error (MSE). The MSE is a measure of the average squared difference between the predicted and actual values. Lower values of the MSE indicate better performance of the model.

```
linear_pred <- predict(lm_model, newdata = data_num)
linear_mse <- mean((linear_pred - data_num$Calories)^2)
lasso_mse <- mean((lasso_pred - std_data$Calories)^2)
ridge_mse <- mean((ridge_pred - std_data$Calories)^2)
kable(data.frame(Model = c("Linear Regression", "Lasso Regression", "Ridge Regression"),
                 MSE = c(linear_mse, lasso_mse, ridge_mse)),
      caption = "MSE values for the models")
```

Table 14: MSE values for the models

Model	MSE
Linear Regression	24.6481166
Lasso Regression	0.0024158
Ridge Regression	0.0066477

We choose the model with the highest R-squared value and the lowest MSE as the best model for predicting the amount of calories based on the amount of the other variables. The best model is the lasso because it has

the lowest value for R^2 and MSE and it is the most robust model.

Comment on Lasso: The Mean Squared Error (MSE) of approximately 0.0024 indicates a very low average squared difference between the observed actual outcomes and the outcomes predicted by the model. This suggests that the model's predictions are very close to the actual values, indicating high accuracy. The optimal lambda value is used to fit the final lasso regression model

?????????????????????? we want to check is Lasso regression has effectvily reduced the multicollinearity, so we calculated the VIF on predictors resulted by fitted Lasso, by looking at coefficients and correlation matrix

Calcola i VIF TENERE??????????????????????????? The variables `"Total_Fat"`, `"Trans_Fat"`, `"Sodium"`, `"Total_Carbohydrates"`, `"Cholesterol"`, `"Dietary_Fibre"`, `"Sugars"`, `"Protein"`, `"Vitamin_A"`, `"Vitamin_C"`, `"Calcium"`, `"Iron"`, and `"Caffeine"` have coefficients of significant magnitudes, suggesting that these variables are important for predicting calories. Despite the regularization of the Lasso model, some variables have coefficients of significant magnitudes, which could suggest that these variables are not strongly correlated with each other, thus reducing the impact of multicollinearity. Overall, the absence of coefficients with very large magnitudes and the presence of coefficients close to zero for some variables suggest that the Lasso model may have helped mitigate multicollinearity and select only the most important variables for predicting calories.

5.6 Cross Validation

Cross validation is a technique used to evaluate the performance of a model. It involves splitting the data into training and testing sets, fitting the model using the training set, and evaluating the model using the testing set. We decided to split the data with 80% of examples for training and 20% for testing.

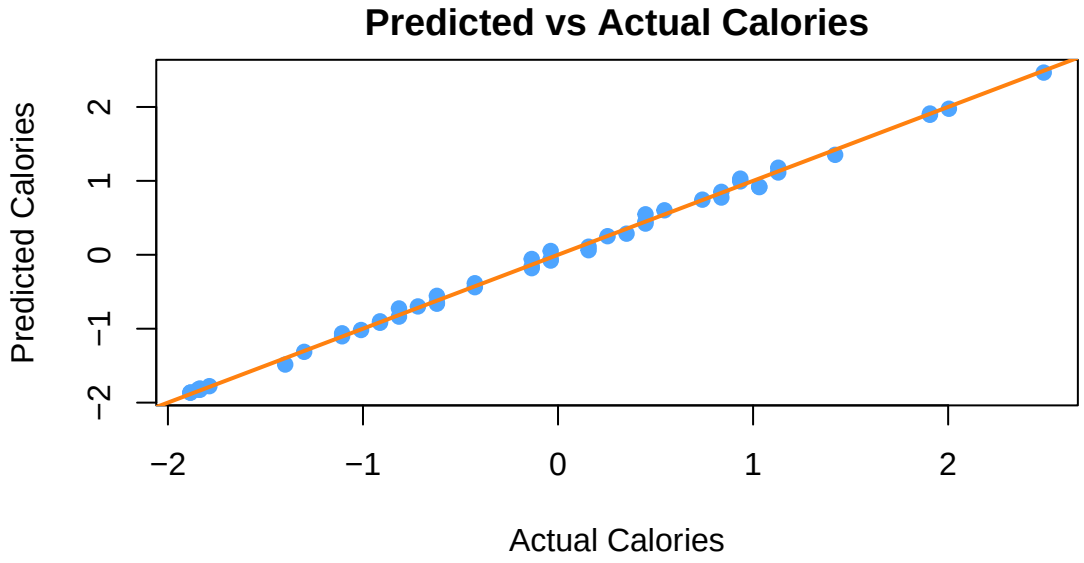
```
set.seed(123)
train_index <- sample(1:nrow(std_data), 0.8 * nrow(std_data))
train_data <- std_data[train_index, ]
test_data <- std_data[-train_index, ]
mod_lasso_train <- cv.glmnet(x = as.matrix(train_data[, -1]), y = train_data$Calories,
                             alpha = 1, standardize = FALSE)
```

We evaluate the model using the testing set. We make predictions using the testing set and calculate the mean squared error and the root mean squared error to assess the model's accuracy.

```
lasso_pred_test <- predict(mod_lasso_train, s = "lambda.min",
                           newx = as.matrix(test_data[, -1]))
lasso_r_squared_test <- cor(lasso_pred_test, test_data$Calories)^2
lasso_mse_test <- mean((lasso_pred_test - test_data$Calories)^2)
```

The R-squared value and MSE are used to evaluate the performance of the model on the test data. Overall, the high R-squared value and low MSE on the test data suggest that the lasso regression model has learned effectively from the training data and generalizes well to unseen examples.

```
accuracy_lm <- 1 - (lasso_mse_test / var(test_data$Calories))
par(mfrow = c(1, 1), mar = c(4, 4, 2, 2))
plot(test_data$Calories, lasso_pred_test, xlab = "Actual Calories", col = "#4ea5ff",
     ylab = "Predicted Calories", main = "Predicted vs Actual Calories", pch = 19)
abline(0, 1, col = "#ff810f", lwd = 2)
```




```
kable(data.frame(Accuracy = accuracy_lm, MSE = lasso_mse_test,
                 R_squared = lasso_r_squared_test),
      caption = "Model evaluation metrics on the test data")
```

Table 15: Model evaluation metrics on the test data

	Accuracy	MSE	R_squared
lambda.min	0.9979473	0.0026283	0.9979454

```
lasso_mse_test
```

```
## [1] 0.002628338
```

As we can see from *Table X* the R-squared value is 0.997, indicating that the model explains 99% of the variance in the data and the MSE is 0.002628338, indicating that the model has a low error rate. The accuracy of the model is 0.9979473, indicating that the model is able to predict the amount of calories with high accuracy. The plot shows the predicted values against the actual values on the test data, as we can see the points are close to the diagonal line, indicating that the model is making accurate predictions.

5.7 Logistic Regression

Logistic regression is a statistical model used to predict the outcome of a binary categorical dependent variable based on one or more independent variables. Since logistic regression is tipycally used for classification task and our variable Calories ,that we want to predict is continous random variable, we have to traspose the problem into a classification one by making the variable binary. In order to do that we classify foods into two categories: “low calorie” and “high calories” defining a threshold to distinguish between the two classes.

let’s take a look into the structure of the variable using numeric dataset only

We’ve tried with normal data, standardize data, and log trasformation since again it helps to reduce multicollinearity and we find out that the best is with log trasformed data looking at the summary and the plot of the variable, we notice that calories follow a semi-gaussian distribution both Median and mean are reasonable approach to use, since they are close to each other. However we choose median as treshold is less sensitive to outliers and skewness in the data. It ensures that half the data points are classified as `"low calories"` and the other half as `"high calories"`, providing balanced classes.

```r
y <- std_data_log_df$Calories
calories_median <- median(y)
# Create a new binary target variable based on the median
std_data_log_df$Calorie_Class <- ifelse(std_data_log_df$Calories > calories_median, 1, 0)
table(std_data_log_df$Calorie_Class)
```

```
##
##   0   1
## 121 121
```

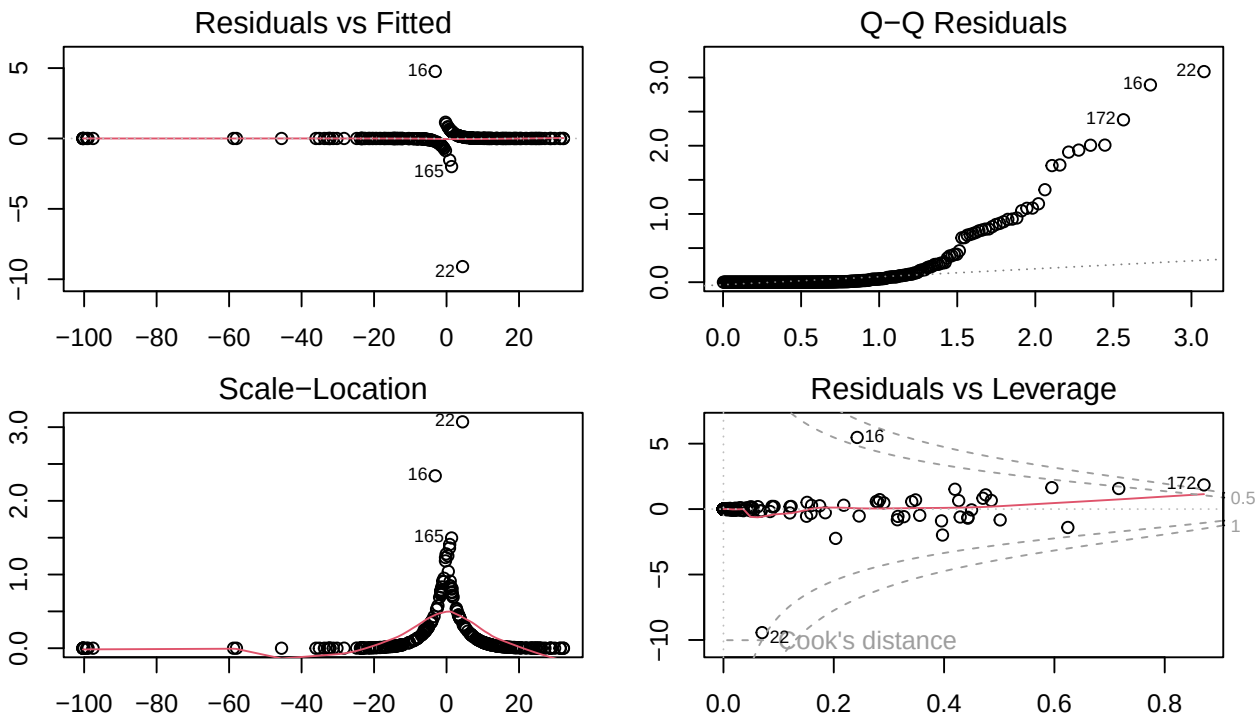
```r
# Creates a new binary variable (Calorie_Class) where 1 indicates high calorie
# (above median) and 0 indicates low calorie (below or equal to median).

# Fit a logistic regression model on the complete dataset obatines with
# log tasformation and by removing first column
logistic_model <- glm(Calorie_Class ~ ., data = std_data_log_df[,-1], family = binomial)
kable(data.frame(AIC = AIC(logistic_model), BIC = BIC(logistic_model),
                 Residual_deviance = logistic_model$deviance,
                 Null_deviance = logistic_model$null.deviance,
                 R_squared = 1 - logistic_model$deviance / logistic_model$null.deviance),
      caption = "Model evaluation metrics for the logistic regression model")
```

Table 16: Model evaluation metrics for the logistic regression model

AIC	BIC	Residual_deviance	Null_deviance	R_squared
69.42364	121.7577	39.42364	335.4832	0.882487

```r
par(mfrow = c(2, 2), mar = c(2, 2, 2, 2))
plot(logistic_model)
```



The statistically significant coefficient of the model are "Cholesterol" at the 0.01 level and "Total_Fat" at the 0.05 level. The residual deviance is much smaller than the null deviance, indicating that the model with predictors explains more variability than the null model. The AIC is 69.424, suggesting that the model has reasonable fit. Number of Fisher Scoring Iterations: Indicates the number of iterations performed by the Fisher scoring algorithm during model fitting. In this case, it took 11 iterations.

Now we use the model to make predictions on the data and evaluate its performance using the confusion matrix, accuracy, precision, recall, and F1-score.

```
conf_matrix <- table(Predicted = predicted_classes, Actual = new_test_data$Calorie_Class)
kable(conf_matrix, caption = "Confusion matrix for the logistic regression model")
```

Table 17: Confusion matrix for the logistic regression model

	0	1
0	22	2
1	2	23

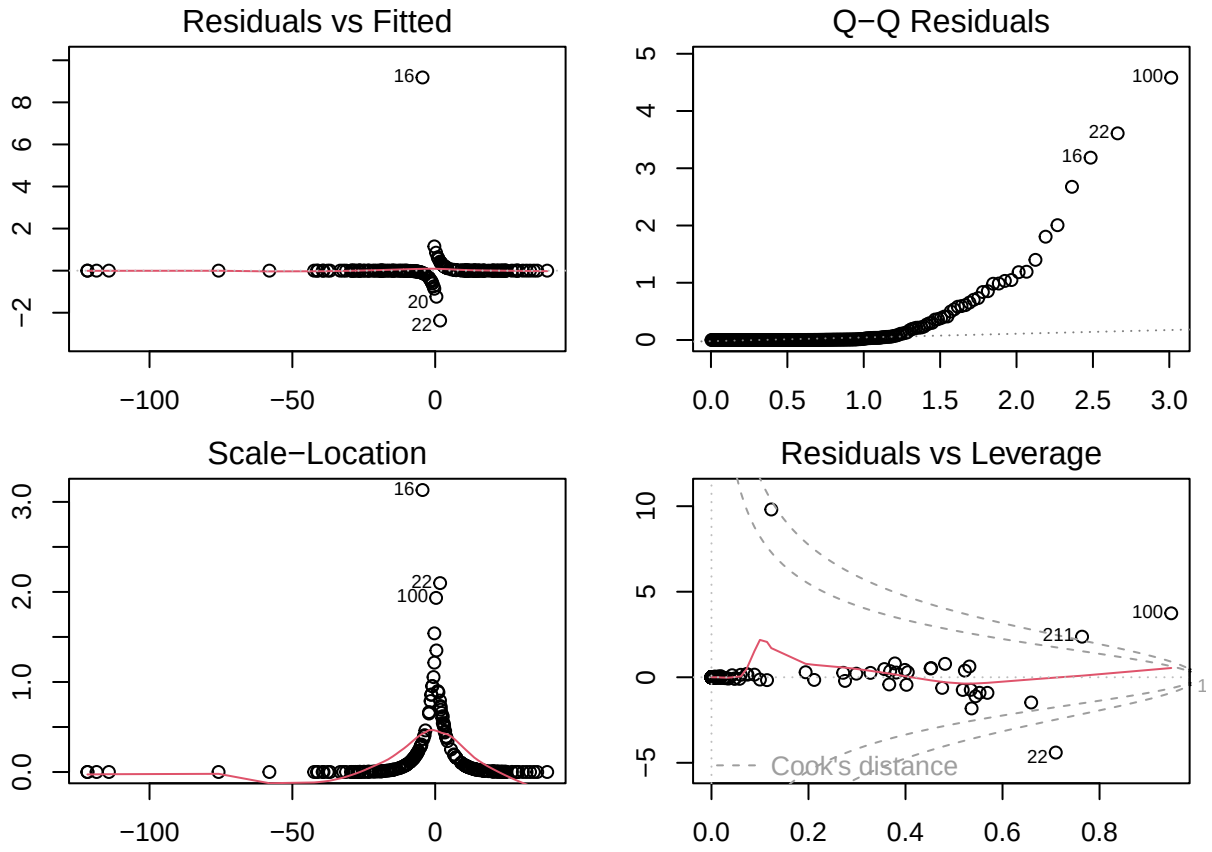
```
accuracy <- (conf_matrix[1, 1] + conf_matrix[2, 2]) / sum(conf_matrix)
precision <- conf_matrix[2, 2] / sum(conf_matrix[, 2])
recall <- conf_matrix[2, 2] / sum(conf_matrix[2, ])
f1_score <- 2 * (precision * recall) / (precision + recall)
kable(data.frame(Accuracy = accuracy, Precision = precision, Recall = recall,
                 F1_Score = f1_score), caption = "Model evaluation metrics")
```

Table 18: Model evaluation metrics

Accuracy	Precision	Recall	F1_Score
0.9183673	0.92	0.92	0.92

The model's accuracy of approximately 91.8% indicates it is performing well overall in classifying the calorie content correctly. Generalize well on the test set

```
par(mfrow = c(2,2), mar = c(2, 2, 2, 2))
plot(logistic_model_train)
```



Residuals vs Fitted:

This plot shows the Pearson residuals against the fitted values. Ideally, there should be no clear pattern, indicating that the model is well-fitted. However, the presence of data points at extreme values (far from 0) suggests potential issues with model fit or outliers.

Normal Q-Q Plot:

The Q-Q plot compares the standardized deviance residuals to a theoretical normal distribution. Significant deviations from the straight line suggest that the residuals are not normally distributed, which can indicate potential problems with the model. In this case, the data points deviate from the line, particularly at the higher quantiles, indicating that the residuals are not perfectly normally distributed.

Scale-Location Plot (Spread-Location Plot): This plot shows the square root of the standardized residuals against the fitted values. The red line helps to identify trends. Ideally, the points should be randomly scattered without a clear pattern. Here, we see some clustering and trends at extreme fitted values, suggesting heteroscedasticity or non-constant variance.

Residuals vs Leverage: This plot shows standardized residuals against leverage, highlighting influential data points. The dashed lines represent Cook's distance. Points outside these lines indicate influential observations that have a significant impact on the model. In this plot, several points, especially at higher leverage values, fall outside the dashed lines, indicating they are influential.

Interpretation

Potential Issues with Model Fit: The presence of extreme residuals in the Residuals vs Fitted and Scale-Location plots suggests that the model might not fit well across all observations. This can be due to outliers or the model not capturing the underlying data structure adequately.

Non-Normal Residuals: The Q-Q plot indicates that the residuals are not perfectly normally distributed, which is expected in logistic regression but still worth noting.

Influential Points: The Residuals vs Leverage plot shows several influential points, suggesting that some observations have a disproportionate impact on the model. These points should be investigated further to understand their nature and whether they are legitimate data points or outliers.

Next Steps

Investigate Influential Points: Check the data points identified as influential in the Residuals vs Leverage plot to understand why they have high leverage and residuals.

Consider Model Refinement: If certain variables consistently show poor performance, it might be necessary to transform them, add interaction terms, or consider alternative modeling techniques.

Check for Multicollinearity: Ensure that multicollinearity is not affecting the model by calculating Variance Inflation Factors (VIFs) for the predictors.

Evaluate Model with Additional Metrics: Use additional performance metrics such as ROC AUC, Precision-Recall curves, and confusion matrix to evaluate the model's predictive performance comprehensively.