Linear regression

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

The prediction task: to predict the amount of calories in Starbucks' drinks based on the other available nutritional data.

Load and inspect the data

The data originates from the Starbucks menu data set. It has been adapted for the class.

Load the data

```
starbucks <- read.csv("data/starbucks_calories.csv")
```

Inspect the data

```
glimpse(starbucks)
```

```
## Rows: 241
## Columns: 7
## $ drink_id          <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1~
## $ beverage          <chr> "Brewed Coffee (Short)", "Brewed Coffee (Tall)", "Br~
## $ calories          <int> 3, 4, 5, 5, 70, 100, 70, 100, 150, 110, 130, 190, 15~
## $ fat_tot           <dbl> 0.1, 0.1, 0.1, 0.1, 0.1, 3.5, 2.5, 0.2, 6.0, 4.5, 0.~
## $ carbohydrates_tot <int> 5, 10, 10, 10, 75, 85, 65, 120, 135, 105, 150, 170, ~
## $ sugar             <int> 0, 0, 0, 0, 9, 9, 4, 14, 14, 6, 18, 17, 8, 23, 22, 1~
## $ proteins          <dbl> 0.3, 0.5, 1.0, 1.0, 6.0, 6.0, 5.0, 10.0, 10.0, 8.0, ~
```

```
# View(starbucks)
```

As the first step, we will do the prediction based on just one variable (simple regression) and then will expand to all potentially relevant variables (multiple regression).

Examining relationships among variables

Let's start examining the correlations among the variables. In particular, we are primarily interested in correlations between the outcome variable (calories) and other variables that may serve as predictors.

Select variables that might be used for predicting calories

```
sb_calories = starbucks[, 3:7]
```

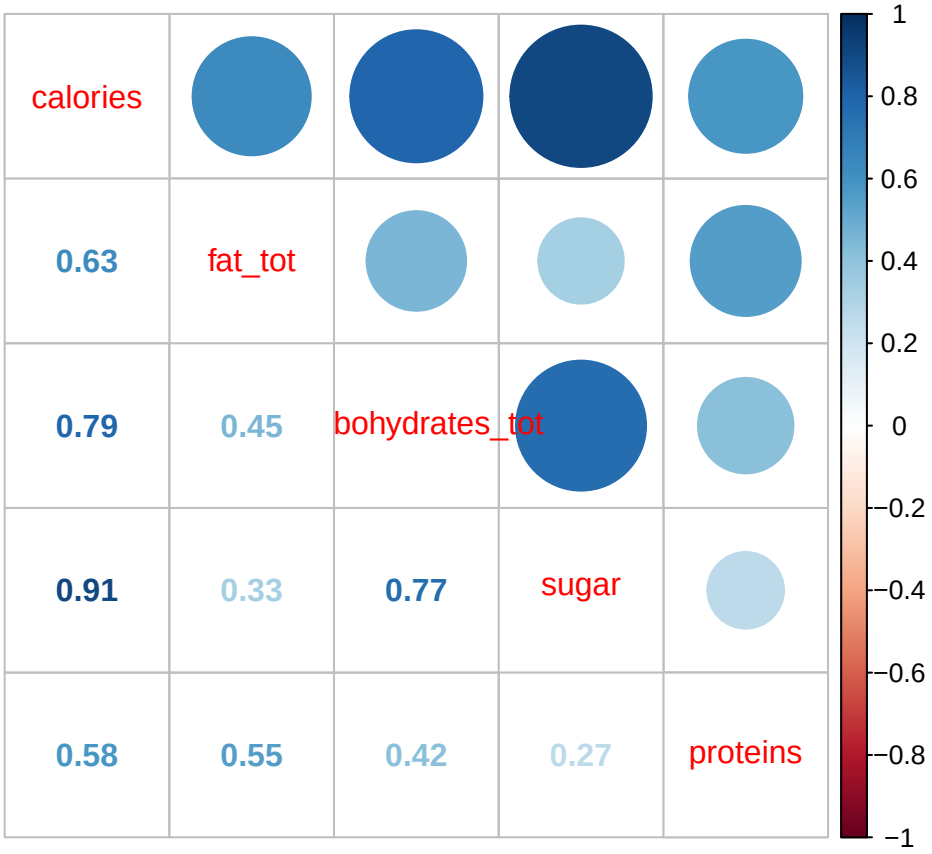
Compute correlations

```
sb_cor = cor(sb_calories)
sb_cor
```

```
##                    calories   fat_tot carbohydrates_tot     sugar  proteins
## calories          1.0000000 0.6342758         0.7935856 0.9091753 0.5825552
## fat_tot           0.6342758 1.0000000         0.4509799 0.3315062 0.5508502
## carbohydrates_tot 0.7935856 0.4509799         1.0000000 0.7691970 0.4151386
## sugar             0.9091753 0.3315062         0.7691970 1.0000000 0.2671951
## proteins          0.5825552 0.5508502         0.4151386 0.2671951 1.0000000
```

We can also plot this matrix to make it easier for inspection

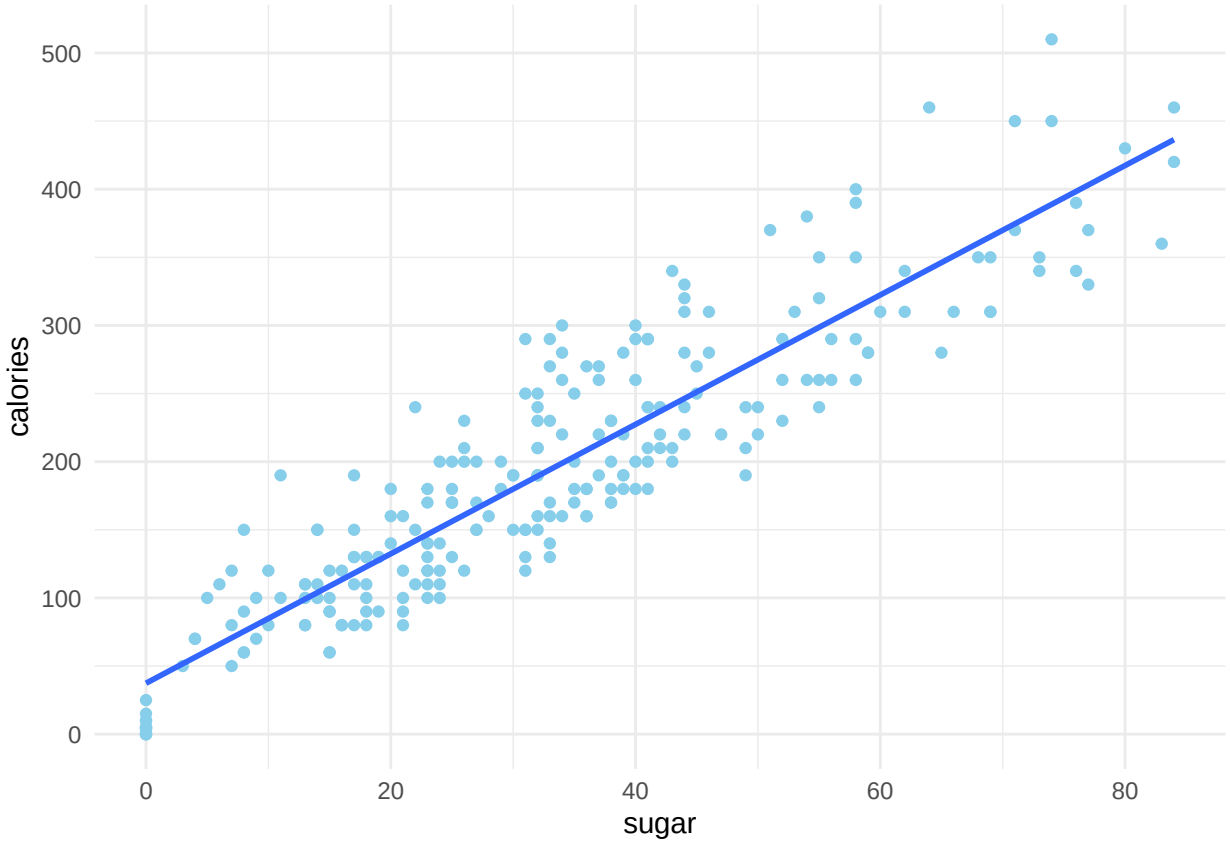
```
corrplot.mixed(sb_cor)
```



Clearly all selected variables are highly correlated with `calories`, but especially `sugar`. Let's inspect that closer.

```
ggplot(data = starbucks,
       mapping = aes(x = sugar, y = calories)) +
  geom_point(color="skyblue") +
  geom_smooth(method = "lm", se=FALSE) +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



It seems that sugar and calories have fine linear relationship, thus satisfying the primary criterion for the application of linear modeling.

Train-test split

Before going any further, we first need to split the data into training and test sets. We will build our models using training data only, while test data will be used for model evaluation.

There are 2 key requirements when splitting the data into training and test sets: 1) random selection of observations to go into training and test sets 2) ensuring the same distribution of the outcome variable in both parts (train and test) We can achieve both things by using the `createDataPartition` function from the `caret` package.

```
library(caret)
```

```
set.seed(2024)
training_indices = createDataPartition(sb_calories$calories, p = 0.75, list = FALSE)
sb_train = sb_calories[training_indices, ]
sb_test = sb_calories[-training_indices, ]
```

Check the distribution of the calories variable on the train and test sets

```
summary(sb_train$calories)
```

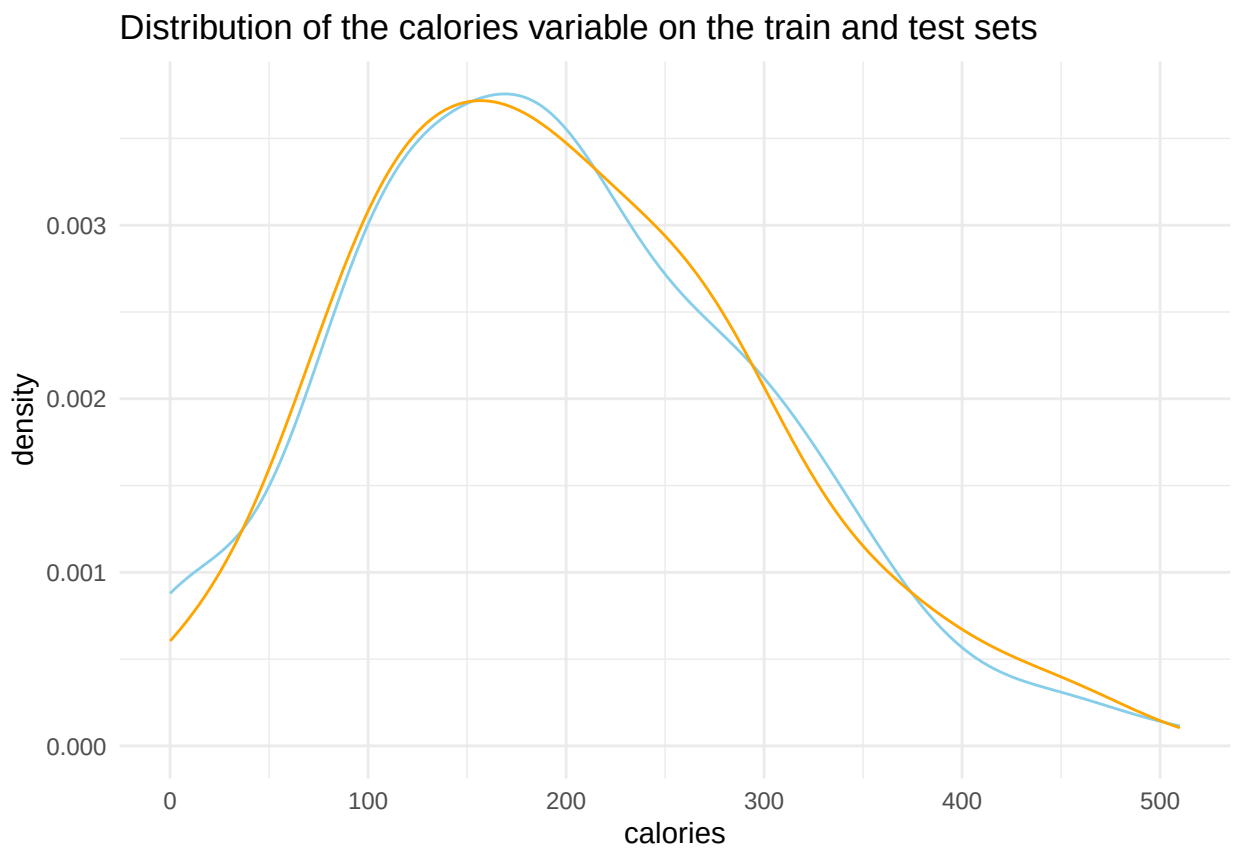
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0   120.0   180.0   192.6   260.0   510.0
```

```
summary(sb_test$calories)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0   120.0   185.0   195.7   260.0   460.0
```

We can do that visually as well

```
ggplot() +
  geom_density(data = sb_train, mapping = aes(x=calories), color = "skyblue") +
  geom_density(data = sb_test, mapping = aes(x=calories), color = "orange") +
  labs(title = "Distribution of the calories variable on the train and test sets") +
  theme_minimal()
```



Simple linear regression

Create the model

```
lm1 = lm(calories ~ sugar, data = sb_train)
```

Print the model

```r
summary(lm1)
```

```
##
## Call:
## lm(formula = calories ~ sugar, data = sb_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -78.821 -34.102  -9.812  24.042 115.751
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  34.8118     6.1384   5.671 5.52e-08 ***
## sugar         4.8573     0.1622  29.951  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 42.61 on 181 degrees of freedom
## Multiple R-squared:  0.8321, Adjusted R-squared:  0.8312
## F-statistic: 897.1 on 1 and 181 DF,  p-value: < 2.2e-16
```

Get the estimated paramter values

```r
coef(lm1)
```

```
## (Intercept)       sugar
##   34.811751    4.857267
```

and the confidence interval for the parameters...

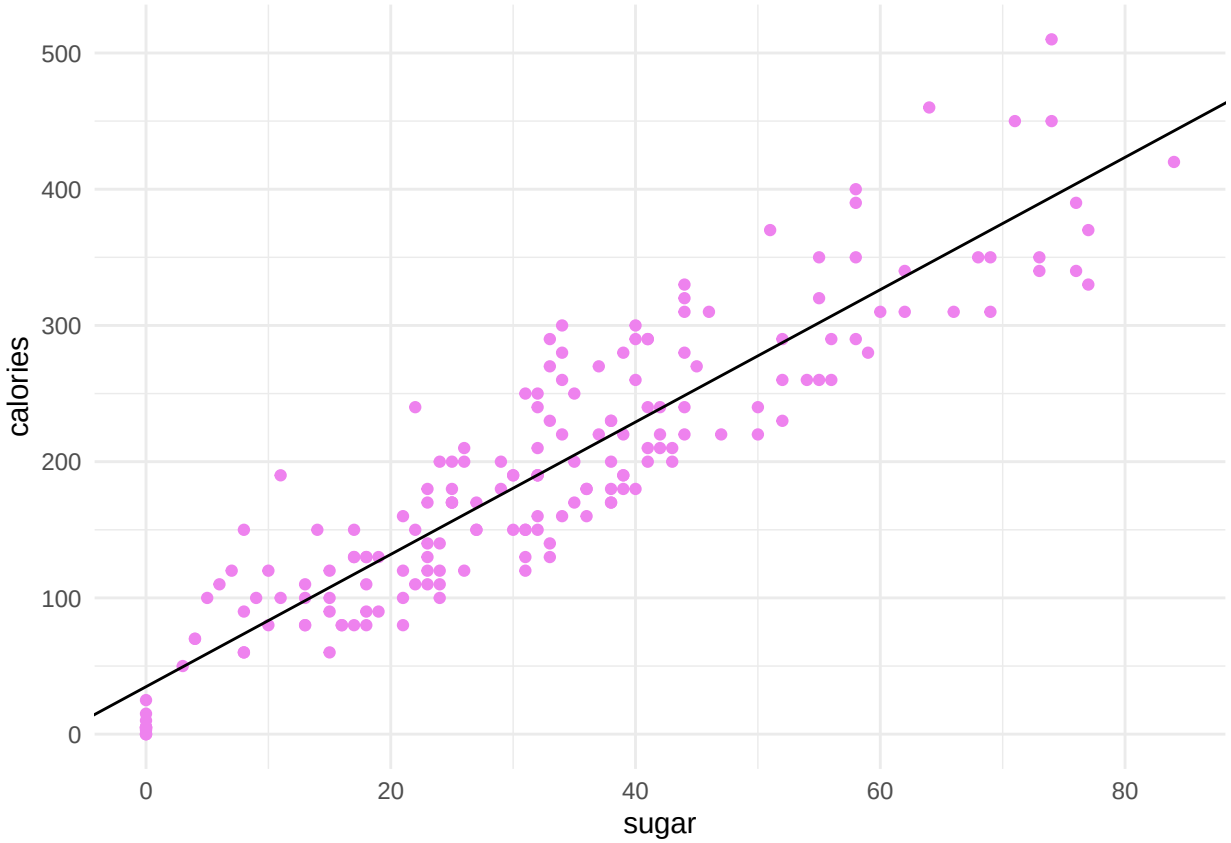
```r
confint(lm1, level = 0.95)
```

```
##                2.5 %    97.5 %
## (Intercept) 22.69981 46.923688
## sugar        4.53727  5.177263
```

We would interpret this as follows: in 95% of samples taken from the population of Starbucks drinks, the estimated value of the parameter associated with the sugar variable will be in the (4.53, 5.18) range.

Plot the regression line

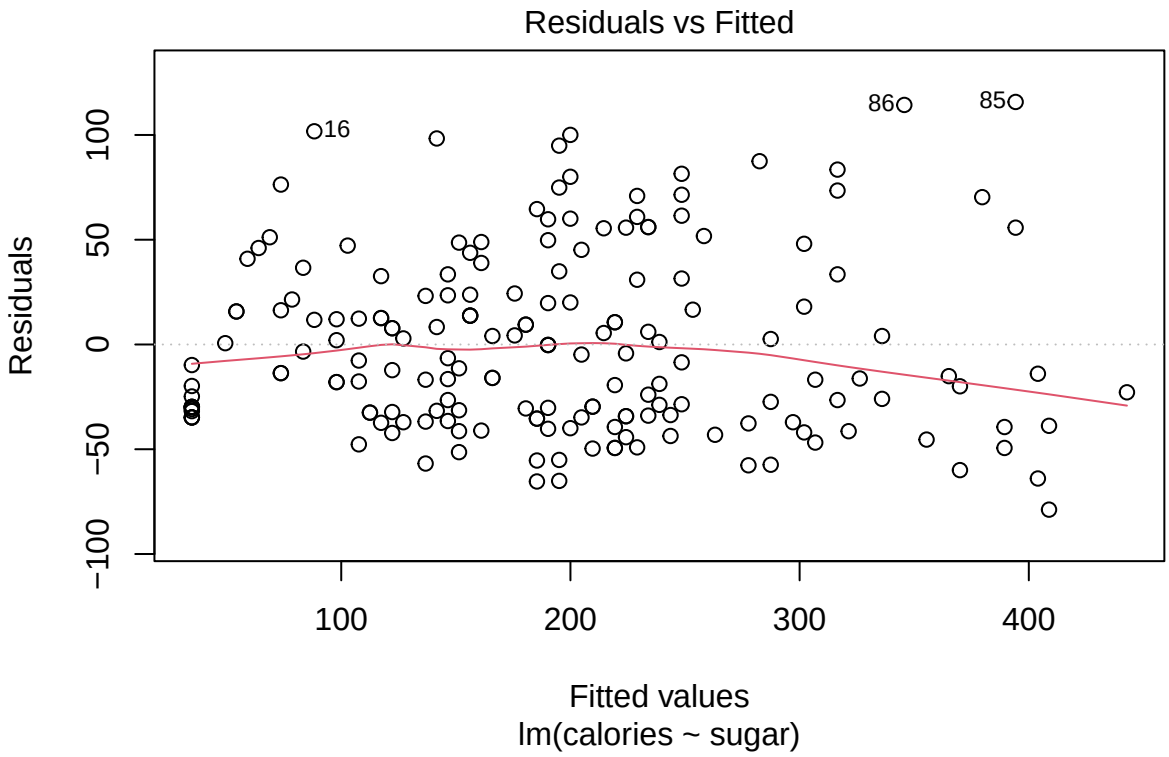
```r
b0 = as.numeric(coef(lm1)[1])
b1 = as.numeric(coef(lm1)[2])

ggplot(data = sb_train,
       mapping = aes(x = sugar, y = calories)) +
  geom_point(color = "violet") +
  geom_abline(intercept = b0, slope = b1) +
  theme_minimal()
```

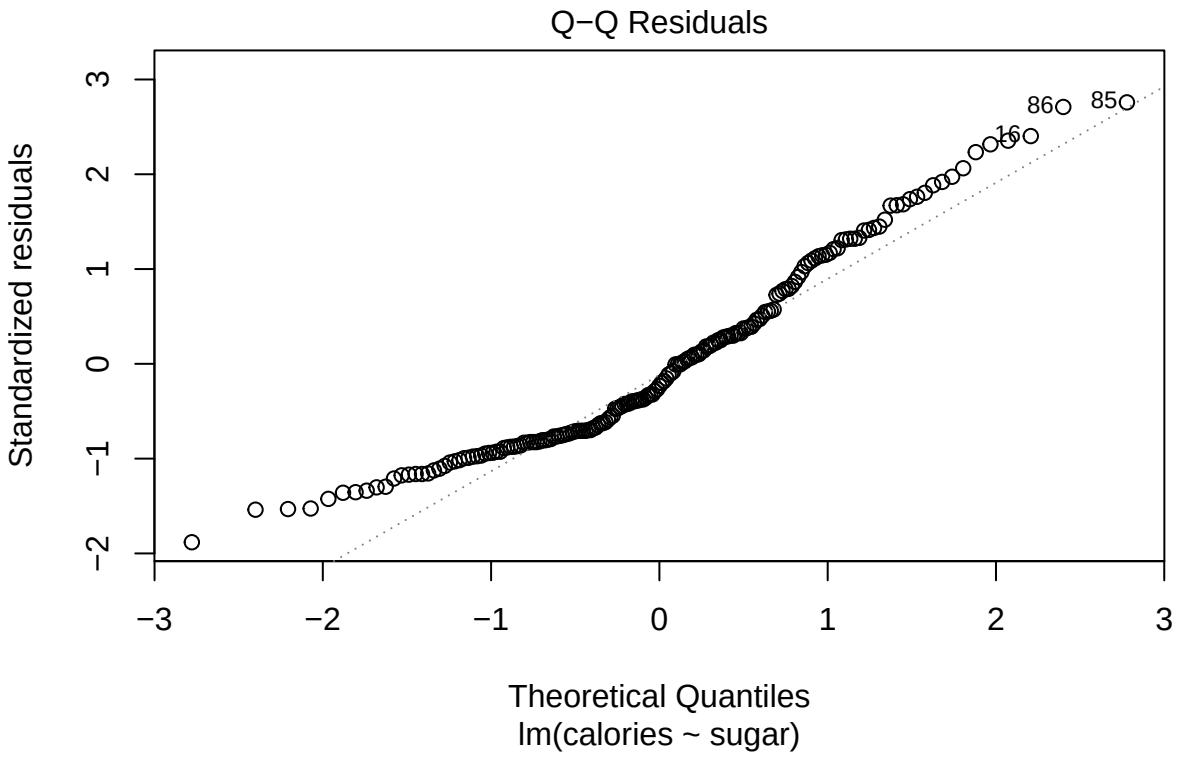


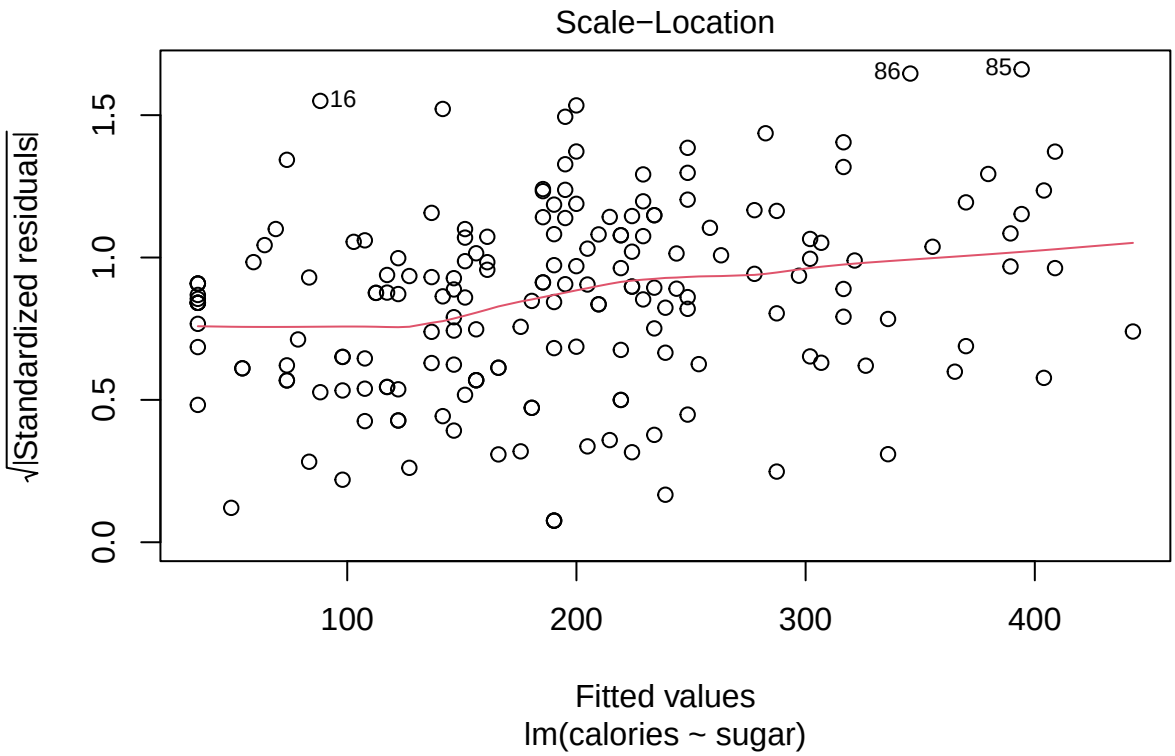
Check if the assumptions for linear models are met, using the `plot` function. The function will create 4 plots:

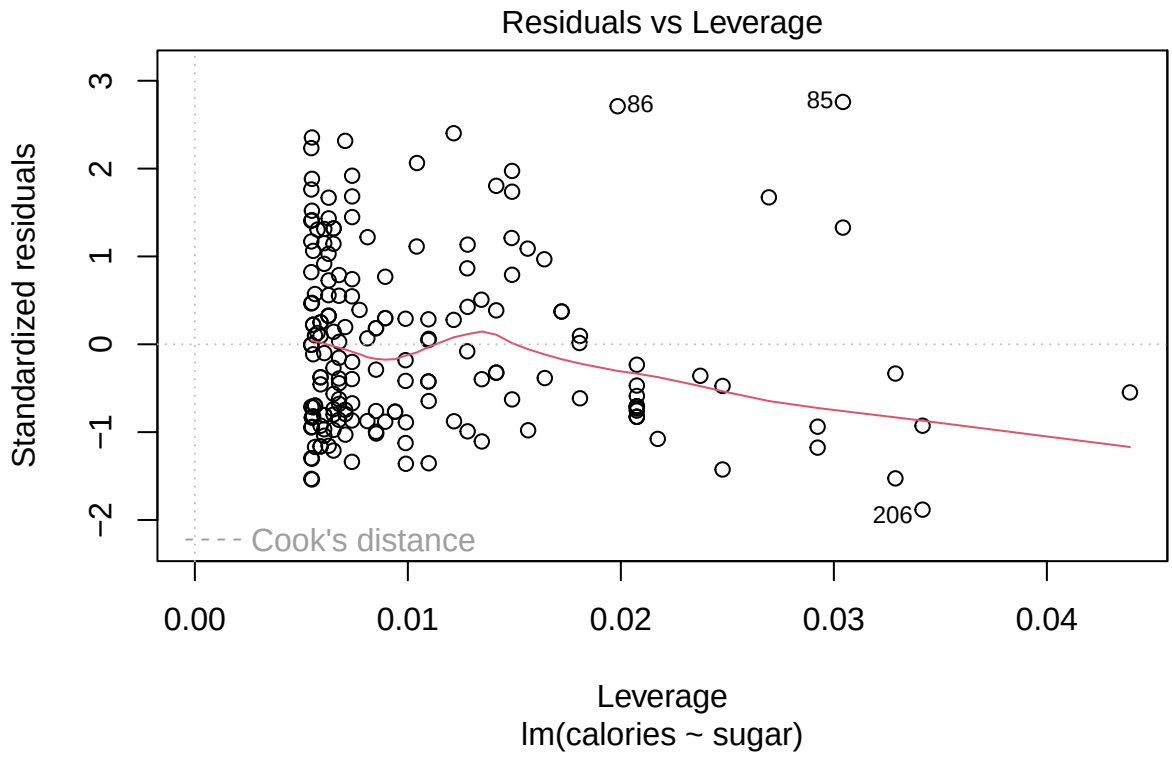
- The 1st plot, **Residual vs Fitted value**, is used for checking if the linearity assumption is satisfied.
- The 2nd plot, **Normal Q-Q plot**, tells us if residuals are normally distributed.
- The 3rd plot, **Scale-Location**, is used for checking the assumption - known as homoscedasticity - that the residuals have equal variance.
- The 4th plot, **Residuals vs Leverage**, is used for spotting the presence of highly influential observations; those are the observations that, if included in or excluded from the model, can significantly affect the values of the model parameters. Such observations are identified among those with unusually high/low output values (outliers) and/or unusually high/low predictor values (high leverage).

```
plot(lm1)
```

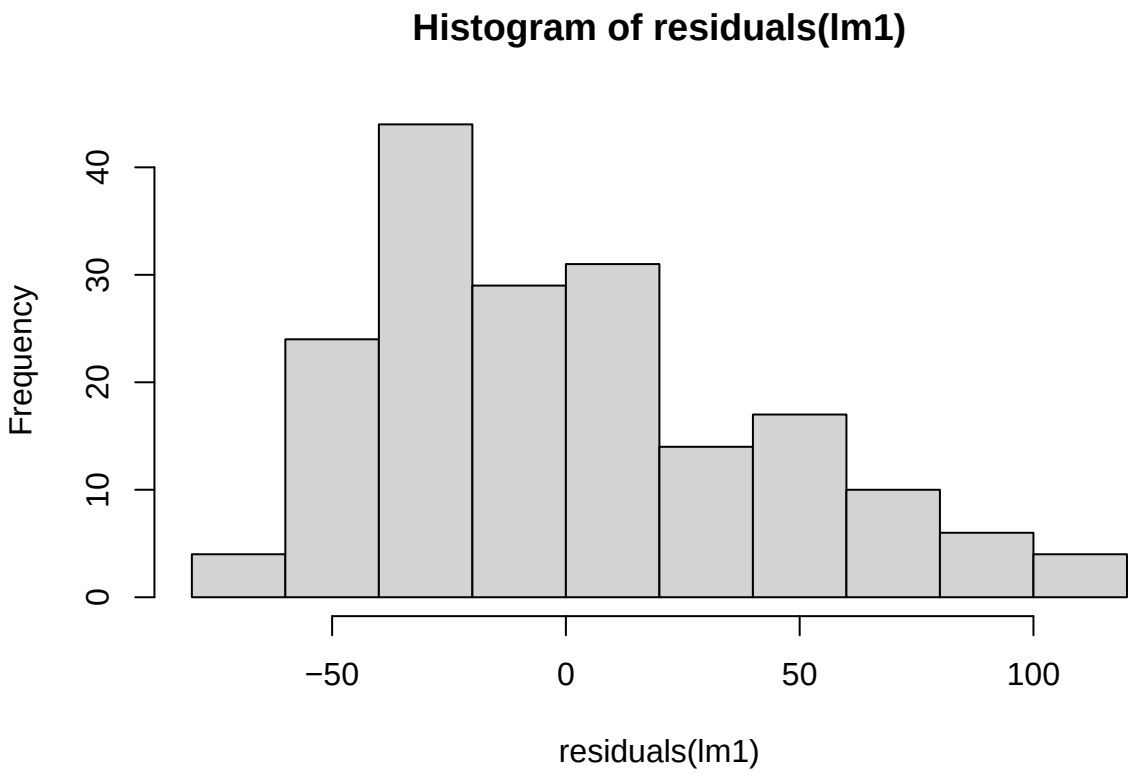

Residuals vs Fitted
Residuals
Fitted values
lm(calories ~ sugar)
16
86
85
100
50
0
−50
−100
100
200
300
400


Q−Q Residuals
Standardized residuals
Theoretical Quantiles
lm(calories ~ sugar)
86
85
16


Scale−Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
16
86
85
100
200
300
400
Fitted values
lm(calories ~ sugar)




```
hist(residuals(lm1))
```



So, out of the four assumptions, one is not satisfied - normality of residuals. This may affect the prediction intervals (see below), that is, lead to incorrect confidence and prediction intervals, since those intervals are calculated based on the assumption that the residuals are normally distributed.

Make predictions and evaluate the model on the test data

```
lm1_predicted = predict(lm1, newdata = sb_test)
```

```
sb_test |> select(sugar, calories) |> mutate(calories_pred = lm1_predicted) |> head(10)
```

```
##    sugar calories calories_pred
## 5      9       70      78.52715
## 8     14      100     102.81348
## 9     14      150     102.81348
## 12    17      190     117.38528
## 17    17      110     117.38528
## 22    20      180     131.95708
## 25    26      230     161.10068
## 27    43      340     243.67421
## 29    18      100     122.24255
## 31    13      110      97.95622
```

We can also get **confidence** and **prediction** intervals for our predictions

```r
lm1_pred_ci = predict(lm1, newdata = sb_test, interval = "confidence", level = 0.95)
sb_test |> select(sugar, calories) |> cbind(lm1_pred_ci) |> head()
```

```
##    sugar calories       fit       lwr       upr
## 5      9       70  78.52715  68.77442  88.27988
## 8     14      100 102.81348  94.23291 111.39405
## 9     14      150 102.81348  94.23291 111.39405
## 12    17      190 117.38528 109.43607 125.33450
## 17    17      110 117.38528 109.43607 125.33450
## 22    20      180 131.95708 124.56810 139.34606
```

Confidence intervals (CI) reflects the uncertainty around the mean predictions. This uncertainty is due to the fact that we can only *estimate* values of the model parameters, but cannot determine the "true" or "ideal" values of those parameters. Consequently, the predictions made by our model will deviate from the predictions that the "ideal" linear model would make; this error is captured by the CI. For example, if we consider a CI of 95%, we can say that in 95% of drinks with 9g of sugar (see the 1st row in the table above), the expected (mean) number of calories will range between 68.77 and 88.28.

```r
lm1_pred_pi = predict(lm1, newdata = sb_test, interval = "prediction", level = 0.95)
sb_test |> select(sugar, calories) |> cbind(lm1_pred_pi) |> head()
```

```
##    sugar calories       fit       lwr      upr
## 5      9       70  78.52715 -6.118175 163.1725
## 8     14      100 102.81348 18.295192 187.3318
## 9     14      150 102.81348 18.295192 187.3318
## 12    17      190 117.38528 32.928753 201.8418
## 17    17      110 117.38528 32.928753 201.8418
## 22    20      180 131.95708 47.551440 216.3627
```

Prediction interval (PI) reflects uncertainty related to a particular instance. This uncertainty is due to not only the fact that we can only *estimate* values of the model parameters (as in case of CI), but also due to the fact that even the best linear model is just an approximation of the true relationship between the input and output variables. Hence, PI captures both the reducible and irreducible error, and thus is always wider than CI.
For example, if we consider a PI of 95%, we can say that in 95% of cases, for a particular drink with 9g of sugar, the number of calories will range between -6.12 and 123.18. However, note that due to the lack of normality of residuals (the assumption that was violated), the prediction intervals for this particular model can be considered trustworthy.

To evaluate the model, we will compute - on the test set - two evaluation measures: R2 and RMSE

R2 = (TSS - RSS)/TSS RMSE = sqrt(RSS/n)

```r
compute_eval_measures <- function(y_train, y_test, y_pred) {
  tss = sum((y_test - mean(y_train))^2)
  rss = sum((y_test - y_pred)^2)
  r2 = (tss - rss)/tss

  n = length(y_test)
  rmse = sqrt(rss/n)

  return(c(R2=r2, RMSE=rmse))
}
```

```
lm1_eval = compute_eval_measures(y_train = sb_train$calories,
                                 y_test = sb_test$calories,
                                 y_pred = lm1_predicted)
```

```
round(lm1_eval, 4)
```

```
##      R2    RMSE
##  0.8057 43.9478
```

To get a perspective of how large the error is, we'll compare it with the mean value of the response variable on the test set.

```
lm1_rmse = as.numeric(lm1_eval['RMSE'])
lm1_rmse/mean(sb_test$calories)
```

```
## [1] 0.2245791
```

It's a fairly large error - about 22% of the average value.

Multiple linear regression

We've seen that in addition to `sugar`, the other three variables are also highly correlated with `calories` (as the outcome variable). Let's see if by adding them to the model we can improve the prediction performance.

```
lm2 = lm(calories ~ fat_tot + sugar + proteins + carbohydrates_tot, data = sb_train)
summary(lm2)
```

```
##
## Call:
## lm(formula = calories ~ fat_tot + sugar + proteins + carbohydrates_tot,
##     data = sb_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -24.610  -4.832  -1.140   3.079  36.794
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        2.15915    1.40170   1.540    0.125
## fat_tot            8.88152    0.28259  31.429   <2e-16 ***
## sugar              3.88674    0.06095  63.765   <2e-16 ***
## proteins           5.06721    0.17064  29.696   <2e-16 ***
## carbohydrates_tot  0.01387    0.01515   0.916    0.361
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.729 on 178 degrees of freedom
## Multiple R-squared:  0.9931, Adjusted R-squared:  0.9929
## F-statistic:  6378 on 4 and 178 DF,  p-value: < 2.2e-16
```

Note that in the presence of `sugar`, `carbohydrates_tot` are not relevant for the prediction of calories. This is intuitively clear, but might also be explained by the high correlation of `sugar` and `carbohydrates_tot` variables (see the correlations plot above). This points to a potential problem of **multicollinearity** that is often present in case of multiple linear regression and should be avoided as it tends to make the resulting model unstable and/or plainly wrong.

Let's check if we have the case of multicollinearity in our data

```
library(performance)
```

One way to check for multicollinearity is to compute the **Variance Inflation Factor (VIF)**:

```
lm2_collinearity = check_collinearity(lm2)
lm2_collinearity
```

```
## # Check for Multicollinearity
##
## Low Correlation
##
##               Term  VIF   VIF 95% CI Increased SE Tolerance Tolerance 95% CI
##            fat_tot 1.66 [1.41, 2.08]         1.29      0.60     [0.48, 0.71]
##              sugar 3.37 [2.69, 4.31]         1.83      0.30     [0.23, 0.37]
##           proteins 1.61 [1.37, 2.01]         1.27      0.62     [0.50, 0.73]
##  carbohydrates_tot 3.76 [2.99, 4.83]         1.94      0.27     [0.21, 0.33]
```

Variance Inflation Factor (VIF) is an often used way to check for multicollinearity. If the VIF value of a variable is high, it means the information in that variable is already explained by other predictors present in the given model, which means that the variable is redundant. If interested in how VIF is computed, this article offers a nice and simple explanation.

As a rule of thumb, variables having $\sqrt{VIF} > 2$ are problematic. Here, it is not the case. When it is the case, we remove the variable with the high VIF value from the data set and build a new model without it. If there are several such variables, we remove them in a step-wise manner, starting from the one with the highest VIF value.

Even though the presence of the `carbohydrates_tot` variable is not causing the problem of multicollinearity and from that perspective can be kept in the model, it proved to be irrelevant for the prediction of the outcome variable and thus it should be better removed from the model. As a general rule, if two (or more) models are of similar performance, it is better to keep the simpler model, since such a model is less prone to overfitting.

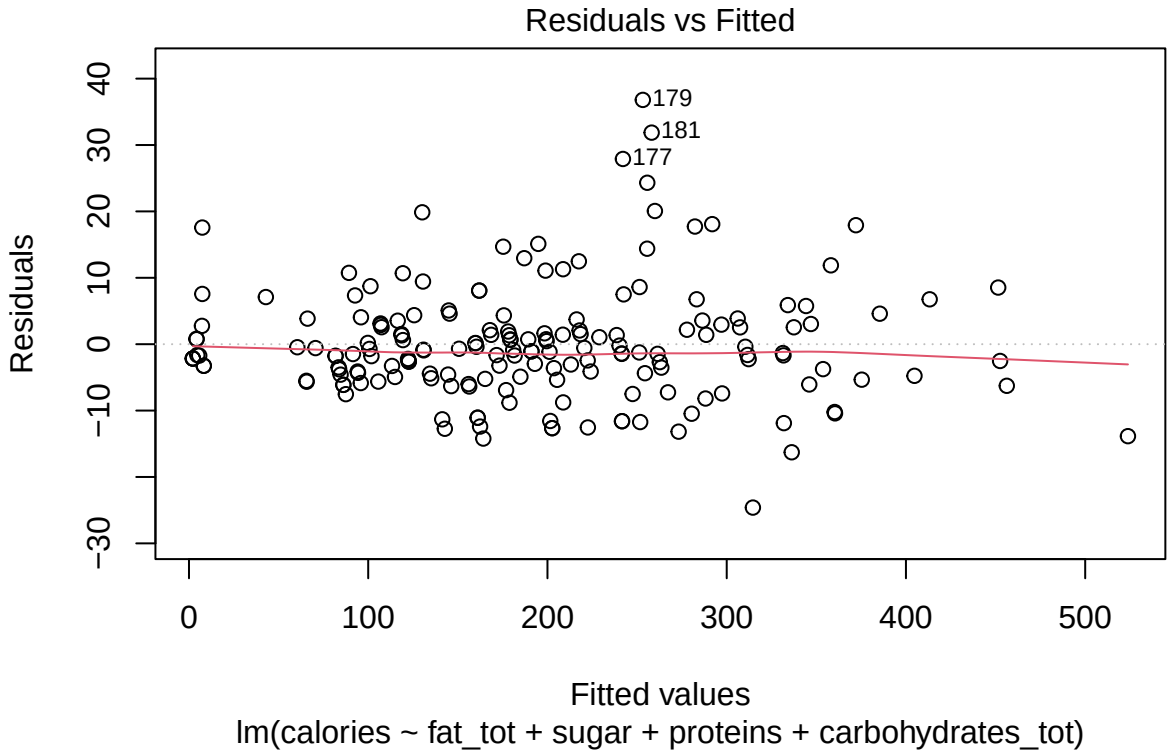
```
lm3 = lm(calories ~ fat_tot + sugar + proteins, data = sb_train)
summary(lm3)
```

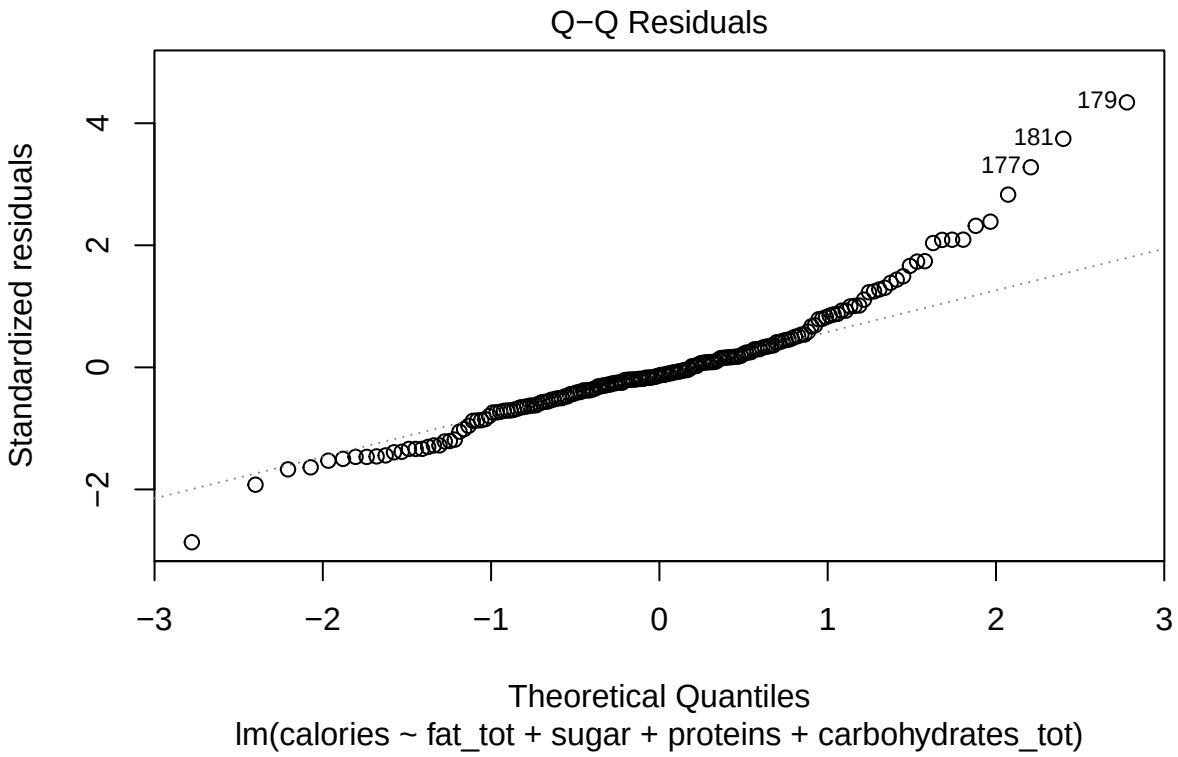
```
##
## Call:
## lm(formula = calories ~ fat_tot + sugar + proteins, data = sb_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -25.030  -4.795  -1.190   3.233  36.190
##
## Coefficients:
```

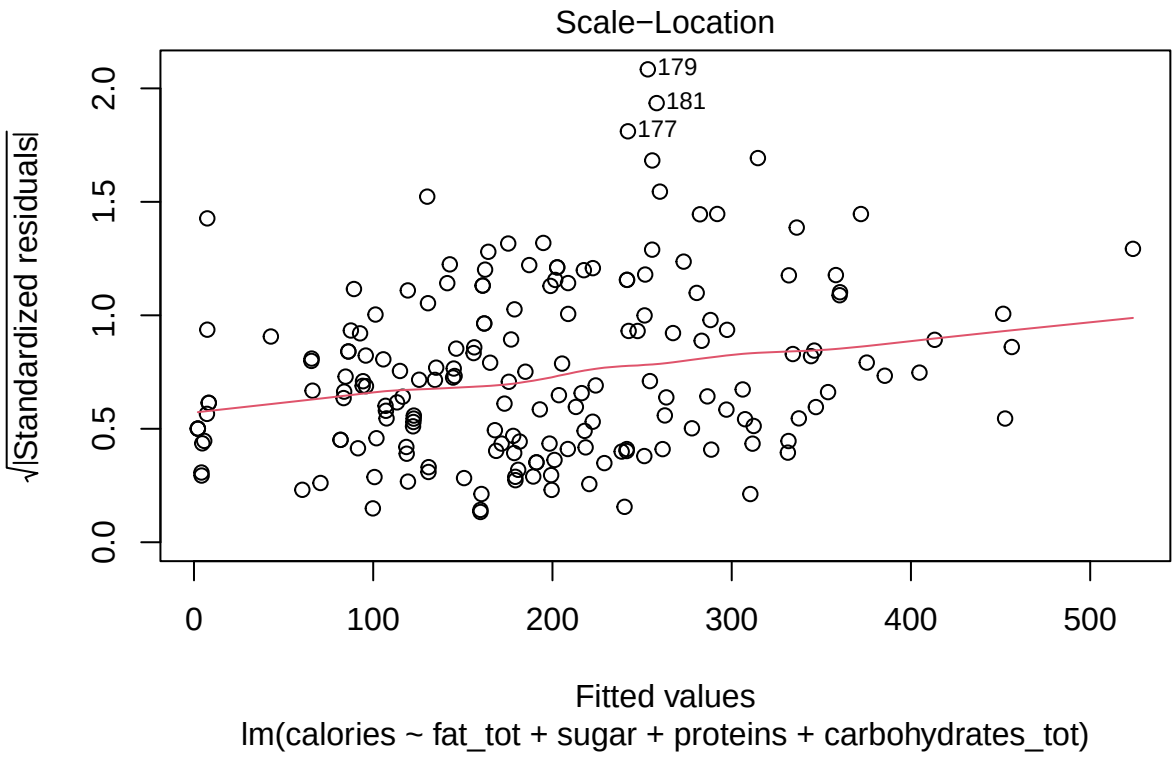
```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.21598    1.39969   1.583    0.115
## fat_tot      8.92549    0.27835  32.065   <2e-16 ***
## sugar        3.93163    0.03621 108.568   <2e-16 ***
## proteins     5.09198    0.16840  30.237   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.725 on 179 degrees of freedom
## Multiple R-squared:  0.993,  Adjusted R-squared:  0.9929
## F-statistic:  8511 on 3 and 179 DF,  p-value: < 2.2e-16
```

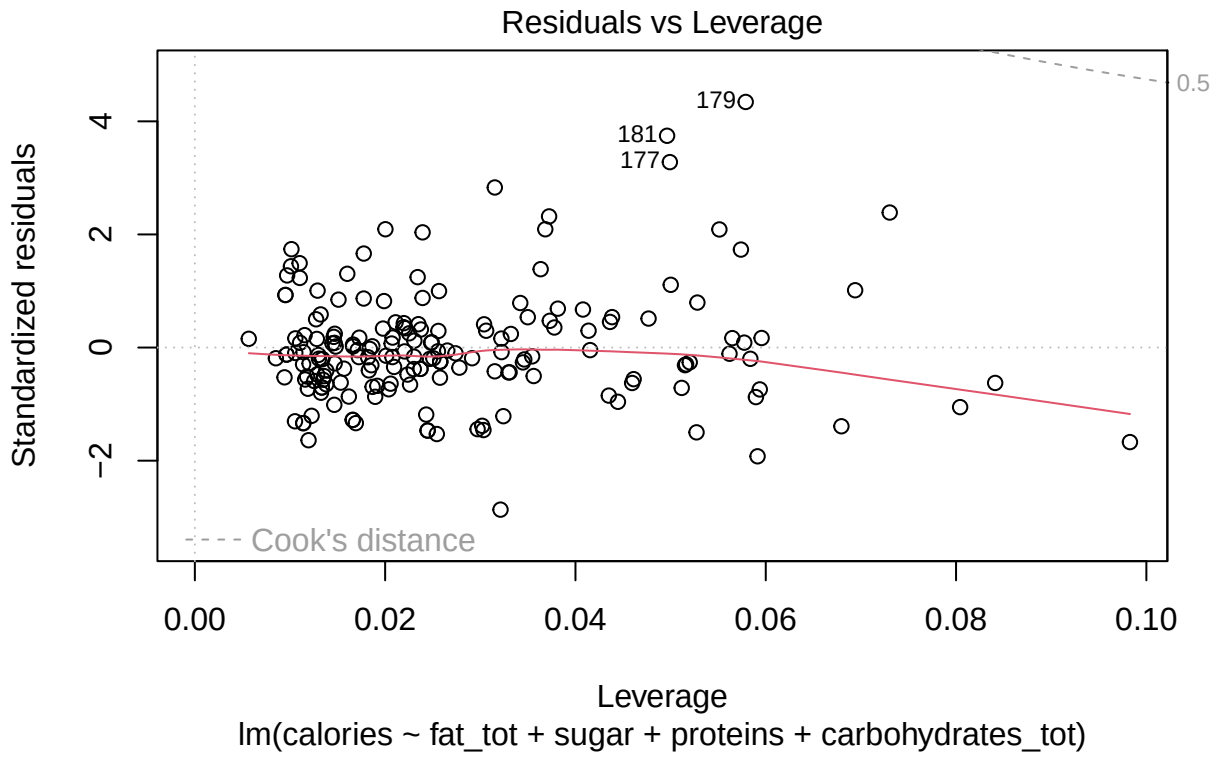
Check the four assumptions of linear models:

```
plot(lm2)
```

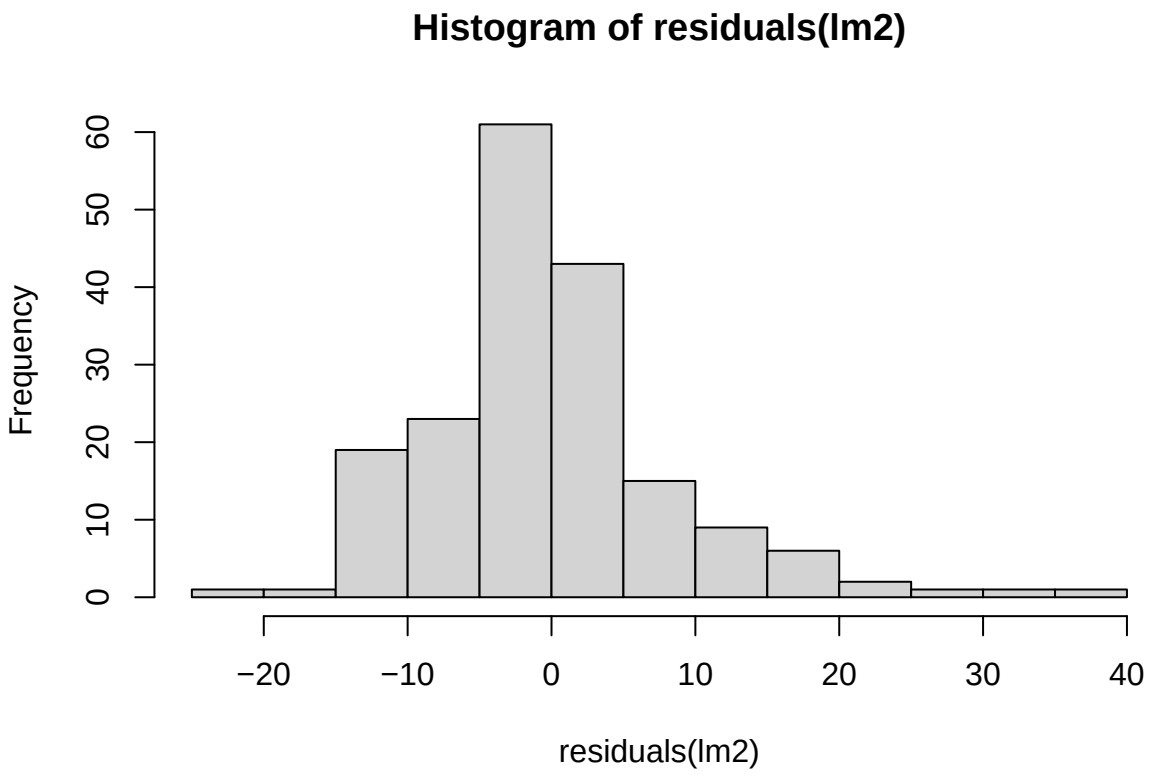



Q−Q Residuals
Standardized residuals
Theoretical Quantiles
lm(calories ~ fat_tot + sugar + proteins + carbohydrates_tot)
179
181
177
−3
−2
−1
0
1
2
3
−2
0
2
4


Scale−Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
2.0
179
181
177
0
100
200
300
400
500
Fitted values
lm(calories ~ fat_tot + sugar + proteins + carbohydrates_tot)




```
hist(residuals(lm2))
```



This time, all assumptions hold

Let's check **confidence** and **prediction** intervals for predictions based on this new model; this time, the intervals should be trustworthy since the all the assumptions are satisfied

```
lm3_pred_ci = predict(lm3, newdata = sb_test, interval = "confidence", level = 0.95)
sb_test |> select(fat_tot, sugar, proteins, calories) |> cbind(lm3_pred_ci) |> head()
```

```
##    fat_tot sugar proteins calories       fit       lwr       upr
## 5      0.1     9        6       70  69.04508  66.77282  71.31735
## 8      0.2    14       10      100 109.96371 107.15340 112.77401
## 9      6.0    14       10      150 161.73157 159.14509 164.31806
## 12     7.0    17       12      190 192.63592 189.77384 195.49800
## 17     1.5    17        7      110 118.08579 116.31730 119.85428
## 22     5.0    20        9      180 171.30387 169.33108 173.27666
```

```
lm3_pred_pi = predict(lm3, newdata = sb_test, interval = "predict", level = 0.95)
sb_test |> select(fat_tot, sugar, proteins, sugar, calories) |> cbind(lm3_pred_pi) |> head()
```

```
##    fat_tot sugar proteins calories       fit       lwr       upr
## 5      0.1     9        6       70  69.04508  51.67778  86.41239
## 8      0.2    14       10      100 109.96371  92.51785 127.40956
## 9      6.0    14       10      150 161.73157 144.32037 179.14278
## 12     7.0    17       12      190 192.63592 175.18165 210.09019
## 17     1.5    17        7      110 118.08579 100.77719 135.39439
## 22     5.0    20        9      180 171.30387 153.97320 188.63454
```

We will now make predictions on the test set and compute evaluation measures

```
lm3_predicted = predict(lm3, newdata = sb_test)

lm3_eval = compute_eval_measures(y_train = sb_train$calories,
                                 y_test = sb_test$calories,
                                 y_pred = lm3_predicted)
round(lm3_eval, 4)
```

```
##      R2    RMSE
##  0.9856 11.9769
```

```
lm3_rmse = as.numeric(lm3_eval['RMSE'])
lm3_rmse/mean(sb_test$calories)
```

```
## [1] 0.06120335
```

Compare the performance of the latest model with the initial model

```
rbind(lm1_eval, lm3_eval) |> round(4)
```

```
##              R2    RMSE
## lm1_eval 0.8057 43.9478
## lm3_eval 0.9856 11.9769
```

Obviously the latest model is much better than the initial one.

Note: linear regression is not restricted to numerical variables; that is, a linear model can be built with categorical variables, as well, or a mix of numerical and categorical variables. However, model interpretation in that case is more difficult than when only numerical variables are used and thus is out of the scope of this course.