ASML Exam

Maher SEBAI (A18)

July 12, 2019

Contents

1 Exercise 1: Procespin dataset analysis

1.1 Introduction

In this exercice we will analyse the procespin dataset which consists of one continuous dependend variable and 10 continous independant variable. We will use our arsenal of learned modelling technique at DSTI to better assess the relationship between the response and the explanatory variables. The goal is to produce the best performing **predictive** model (in a Kaggle-style objective), in other words we won't carry explanatory modeling in our dataset. The choosen performance metric is the Root Mean Squared Errror **RMSE**. Our analysis strategy is the following:

- Dataset Statistical exploratory analysis
- Classic Multiple regression modeling with model selection
- Principal Component Analysis and Regression
- Partial Least Square Regression
- Penalized Regression: Ridge and Lasso
- Random Forest Modeling
- Randon Forest with PCA projected predictors
- Boosted Trees Modeling

- Summary of findings and Conclusion

1.2 Data Acquisition and Environment Setup

We start by loading our library toolbox

```
library(PerformanceAnalytics)      # for dataframe correlation analysis
library(olsrr)                     # Linear Regression utility library
library(tidyverse)                 # Dataframe manipulation toolbox
library(caret)                     # for machine learning easier workflow
library(pls)                       # PCA and PLS regression
library(glmnet)                    # Ridge, Lasso Penalized Regression
library(randomForest)              # Random Forest
library(e1071)                     # utility library
library(ranger)                    # random Forest wrapper package
library(xgboost)                   # boosted trees
options(digits = 3)
```

We load our prospin and supplementary prospin datasets, modeling and cross-validation will be performed on the first one, metric performance assessed on the second one.

```
procespin.df <- read.table("procespin.txt", header = TRUE)
head(procespin.df)
```

```
##      y   x1 x2 x3  x4   x5  x6  x7  x8  x9 x10
## 1 2.37 1200 22  1 4.0 14.8 1.0 1.1 5.9 1.4 1.4
## 2 1.47 1342 28  8 4.4 18.0 1.5 1.5 6.4 1.7 1.7
## 3 1.13 1231 28  5 2.4  7.8 1.3 1.6 4.3 1.5 1.4
## 4 0.85 1254 28 18 3.0  9.2 2.3 1.7 6.9 2.3 1.6
## 5 0.24 1357 32  7 3.7 10.7 1.4 1.7 6.6 1.8 1.3
## 6 1.49 1250 27  1 4.4 14.8 1.0 1.7 5.8 1.3 1.4
```

```
procespinsup.df <- read.table("procespinsup.txt", header = TRUE)
head(procespinsup.df)
```

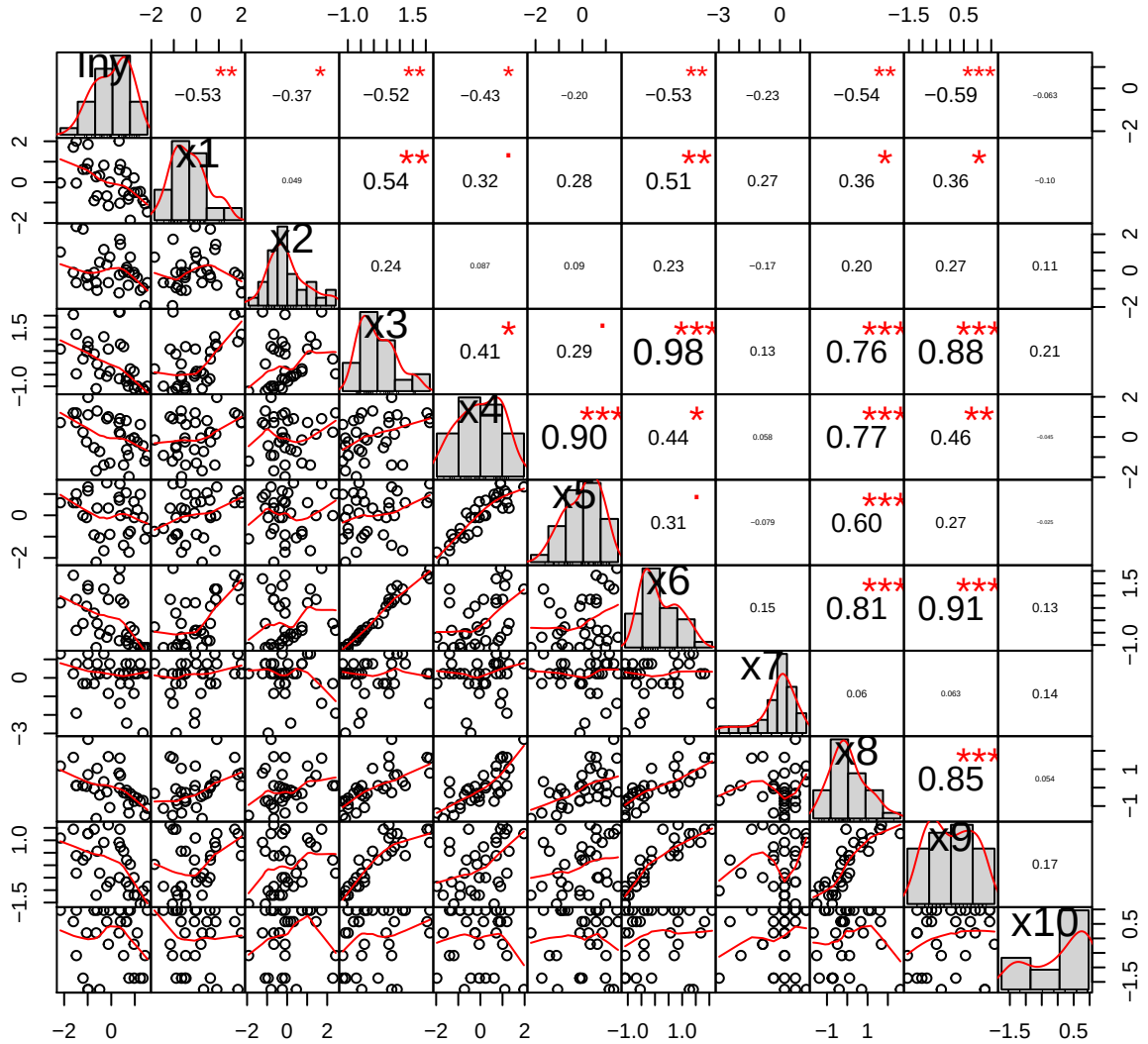
```
##    n   x1 x2 x3  x4   x5  x6  x7  x8  x9 x10  x11
## 1 34 1107 31 23 6.0 22.2 2.6 1.0 9.0 3.0 1.4 1.17
## 2 35 1116 34  6 2.5  6.6 1.3 1.8 3.9 1.2 1.5 0.67
## 3 36 1174 32 22 3.9 11.9 2.3 1.7 6.1 1.8 1.5 0.90
## 4 37 1131 30  6 4.7 15.3 1.5 1.5 6.5 1.4 1.3 2.32
## 5 38 1150 34 12 3.1  9.4 1.7 1.8 4.8 1.6 1.3 3.89
## 6 39 1132 22 18 7.0 37.0 2.5 1.5 9.0 2.0 1.5 6.00
```

The goal is to model Log(y) function of the remaining 10 explanatory variable. All variable (including response) are then standardized (centered and scaled). The supplementary procespin data are standardized using the original procespin mean and standard deviation.

```
procespin.df <-  transform(procespin.df, y = log(y))
colnames(procespin.df)[1] <- c("lny")
procespin.scaled.df <- data.frame(scale(procespin.df))
procespinsup.df <- cbind(procespinsup.df[12],procespinsup.df[-c(1,12)])
procespinsup.df <-  transform(procespinsup.df, x11 = log(x11))
colnames(procespinsup.df)[1] <- c("lny")
procespin.mean = attr(scale(procespin.df), "scaled:center")
procespin.scale = attr(scale(procespin.df), "scaled:scale")
procespinsup.scaled.df <- data.frame(scale(procespinsup.df, center = procespin.mean, scale = procespin.s
```

Let's now analyse the correlation matrix:

```
chart.Correlation(procespin.scaled.df)
```



A lot of correlation is hightlighted among the independant variables, for instance between $\mathbf{x^3}$ and $\mathbf{x^6}$ (0,98), $\mathbf{x^4}$ and $\mathbf{x^5}$ (0.9), $\mathbf{x^6}$ and $\mathbf{x^9}$ (0,91), $\mathbf{x^3}$ and $\mathbf{x^9}$ (0,88). This reveals that, besides simple linear dependency of the listed explanatory variables, we can suspect a bigger issue of **Muticolinearity** among these variables. This leads to instable results for the linear regression when solved using OLS methods. Let's first build the model and output its summary

```
model.lm.full <- lm(lny ~ ., data = procespin.scaled.df)
summary(model.lm.full)
```

```
##
## Call:
## lm(formula = lny ~ ., data = procespin.scaled.df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.3643 -0.2636  0.0035  0.2312  1.4356
```

```
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.31e-16   1.17e-01    0.00    1.000
## x1          -4.73e-01   1.62e-01   -2.92    0.008 **
## x2          -3.04e-01   1.26e-01   -2.40    0.025 *
## x3           4.43e-01   7.63e-01    0.58    0.567
## x4          -1.13e+00   4.74e-01   -2.38    0.026 *
## x5           8.56e-01   3.64e-01    2.35    0.028 *
## x6          -1.65e-01   9.06e-01   -0.18    0.857
## x7          -2.35e-02   1.51e-01   -0.15    0.878
## x8           3.12e-01   4.50e-01    0.69    0.495
## x9          -5.36e-01   3.98e-01   -1.35    0.192
## x10         -9.77e-02   1.52e-01   -0.64    0.527
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.669 on 22 degrees of freedom
## Multiple R-squared:  0.692,  Adjusted R-squared:  0.552
## F-statistic: 4.94 on 10 and 22 DF,  p-value: 0.000865
```

Only 4 predictors appear to be significant, the overall F-statistic also is significant. We build some metric lists to help us store and compare subsequent model performances.

```
# fitted values root mean square error
metric.procespin.rmse <- c(lm.full = RMSE(model.lm.full$fitted.values, procespin.scaled.df$lny))
# predicted values root mean square error
k <- predict(model.lm.full, newdata = procespinsup.scaled.df)
metric.procespin.rmsep <- c(lm.full = RMSE(k, procespinsup.scaled.df$lny))
```

To confirm the Multicolinearity issue we can use the ** Variance Inflation Factor (VIF) ** performed on a full Linear Regression Model. Let's inspect the VIF report for the full model.

```
ols_vif_tol(model.lm.full)
```

```
## # A tibble: 10 x 3
##    Variables Tolerance   VIF
##    <chr>         <dbl> <dbl>
##  1 x1           0.533   1.88
##  2 x2           0.876   1.14
##  3 x3           0.0240 41.6
##  4 x4           0.0622 16.1
##  5 x5           0.106   9.44
##  6 x6           0.0171 58.6
##  7 x7           0.611   1.64
##  8 x8           0.0693 14.4
##  9 x9           0.0884 11.3
## 10 x10          0.607   1.65
```

Values above 5 depicts variable with highly suspicious Multicolineary problems, values above 10 are really harmful. We notice that $\mathbf{x^3}$, $\mathbf{x^4}$, $\mathbf{x^6}$, $\mathbf{x^8}$ and $\mathbf{x^9}$ shows VIF values above 10. We'll try to address this by first trying to automate the model selection. Several Methods will be tried out.

1.3 All possible Regression

The first variable selection approach is a brut force one: as we have 10 predictor, we have then $2^{10} = 1024$ possible combination of linear models. We will select the best one based on the adjusted R^2 criterion and

also the AIC criterion

```
model.allpossible <- ols_step_all_possible(model.lm.full)
# model with highest Adjusted R squared
model.allpossible[which.max(model.allpossible$adjr),]
```

```
## # A tibble: 1 x 6
##   Index     N Predictors        `R-Square` `Adj. R-Square` `Mallow's Cp`
## * <int> <int> <chr>                  <dbl>           <dbl>         <dbl>
## 1   638     6 x1 x2 x4 x5 x6 x9      0.676           0.601          4.15
```

```
# model with lowest AIC
model.allpossible[which.min(model.allpossible$aic),]
```

```
## # A tibble: 1 x 6
##   Index     N Predictors  `R-Square` `Adj. R-Square` `Mallow's Cp`
## * <int> <int> <chr>            <dbl>           <dbl>         <dbl>
## 1   176     4 x1 x2 x4 x5      0.650           0.600          2.00
```

We build the two choosen models and compute their respective Performane Metrics.

```
model.allpossible.best.adjr <- lm(lny ~ x1 + x2 + x4 + x5 + x6 + x9, data = procespin.scaled.df)
model.allpossible.best.aic <- lm(lny ~ x1 + x2 + x4 + x5, data = procespin.scaled.df)
# Metric for best Adjested R-squared
# fitted values root mean square error
metric.procespin.rmse <- c(metric.procespin.rmse, allpossible.best.adjr = RMSE(model.allpossible.best.ad
# predicted values root mean square error
k <- predict(model.allpossible.best.adjr, newdata = procespinsup.scaled.df)
metric.procespin.rmsep <- c(metric.procespin.rmsep, allpossible.best.adjr = RMSE(k, procespinsup.scaled
# Metric for best AIC
# fitted values root mean square error
metric.procespin.rmse <- c(metric.procespin.rmse, allpossible.best.aic = RMSE(model.allpossible.best.ai
# predicted values root mean square error
k <- predict(model.allpossible.best.aic, newdata = procespinsup.scaled.df)
metric.procespin.rmsep <- c(metric.procespin.rmsep, allpossible.best.aic = RMSE(k, procespinsup.scaled.
```

For pure prediction performance, the best adjusted R-squared model with 6 predictor is the top ranked one so far.

```
data.frame(RMSE = metric.procespin.rmse, RMSE.Pred = metric.procespin.rmsep)
```

```
##                          RMSE RMSE.Pred
## lm.full                 0.547      1.28
## allpossible.best.adjr 0.561      1.20
## allpossible.best.aic  0.583      1.45
```

1.4 Best Subset Regression

Select the subset of predictors that do the best at meeting some well-defined objective criterion, such as having the largest R2 value or the smallest MSE, Mallow's Cp or AIC.

```
ols_step_best_subset(model.lm.full)
```

```
##            Best Subsets Regression
## -------------------------------------------
## Model Index    Predictors
## -------------------------------------------
##      1         x9
```

```
##      2         x1 x9
##      3         x1 x4 x5
##      4         x1 x2 x4 x5
##      5         x1 x2 x4 x5 x9
##      6         x1 x2 x4 x5 x6 x9
##      7         x1 x2 x3 x4 x5 x9 x10
##      8         x1 x2 x3 x4 x5 x8 x9 x10
##      9         x1 x2 x3 x4 x5 x6 x8 x9 x10
##     10         x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
## ----------------------------------------------
##
##                                                    Subsets Regression Summary
## ------------------------------------------------------------------------------------------------------------
##                         Adj.        Pred
## Model    R-Square    R-Square    R-Square     C(p)        AIC        SBIC        SBC       MSEP
## ------------------------------------------------------------------------------------------------------------
##   1        0.3528      0.3319       0.273    17.2157    84.2750    -10.6824    88.7645    0.7112    (
##   2        0.4690      0.4337       0.368    10.9160    79.7427    -14.7188    85.7288    0.6238    (
##   3        0.5368      0.4889       0.432     8.0760    77.2361    -16.4322    84.7186    0.5830    (
##   4        0.6498      0.5998        0.54     2.0049    70.0049    -20.6522    78.9839    0.4734    (
##   5        0.6605      0.5977       0.511     3.2422    70.9826    -18.5932    81.4582    0.4942    (
##   6        0.6758      0.6010       0.526     4.1498    71.4610    -16.5941    83.4331    0.5097    (
##   7        0.6840      0.5956       0.509     5.5642    72.6154    -14.0624    86.0840    0.5382    (
##   8        0.6906      0.5874       0.493     7.0957    73.9231    -11.3309    88.8882    0.5729    (
##   9        0.6916      0.5709       0.458     9.0240    75.8159     -8.3636    92.2774    0.6230    (
##  10        0.6919      0.5519       0.391    11.0000    77.7798     -5.3701    95.7379    0.6815    (
## ------------------------------------------------------------------------------------------------------------
## AIC: Akaike Information Criteria
##  SBIC: Sawa's Bayesian Information Criteria
##  SBC: Schwarz Bayesian Criteria
##  MSEP: Estimated error of prediction, assuming multivariate normality
##  FPE: Final Prediction Error
##  HSP: Hocking's Sp
##  APC: Amemiya Prediction Criteria
```

The model with the highest Adjusted R-squared (0.6010) is the model 6 (x1 x2 x4 x5 x6 x9), and the model which has the lowest AIC (70.0049) is the model 4 (x1 x2 x4 x5). Both model have been already discussed in the previous section. So this method did not provide a novel solution.

1.5 AIC stepwise Regression

Build regression model from a set of candidate predictor variables by entering and removing predictors based on AIC values, in a stepwise manner until there is no variable left to enter or remove any more.

```
ols_step_both_aic(model.lm.full)
```

```
## Stepwise Selection Method
## ------------------------
##
## Candidate Terms:
##
## 1 . x1
## 2 . x2
## 3 . x3
## 4 . x4
```

```
## 5 . x5
## 6 . x6
## 7 . x7
## 8 . x8
## 9 . x9
## 10 . x10
##
##
## Variables Entered/Removed:
##
## - x9 added
## - x1 added
## - x2 added
## - x6 added
## - x7 added
##
## No more variables to be added or removed.
```

```
##
##
##                               Stepwise Summary
## ----------------------------------------------------------------------------
## Variable     Method      AIC       RSS      Sum Sq     R-Sq      Adj. R-Sq
## ----------------------------------------------------------------------------
## x9          addition    84.275    20.710    11.290    0.35282      0.33195
## x1          addition    79.743    16.990    15.010    0.46905      0.43365
## x2          addition    78.070    15.201    16.799    0.52497      0.47583
## x6          addition    76.872    13.797    18.203    0.56884      0.50725
## x7          addition    76.380    12.793    19.207    0.60021      0.52617
## ----------------------------------------------------------------------------
```

Let's compute the metrics for This novel model:

```
model.stepwise.aic <- lm(lny ~ x1 + x2 + x6 + x7 + x9, data = procespin.scaled.df)
# Metric for best AIC stepwise regression
# fitted values root mean square error
metric.procespin.rmse <- c(metric.procespin.rmse, stepwise.aic = RMSE(model.stepwise.aic$fitted.values,
# predicted values root mean square error
k <- predict(model.stepwise.aic, newdata = procespinsup.scaled.df)
metric.procespin.rmsep <- c(metric.procespin.rmsep, stepwise.aic = RMSE(k, procespinsup.scaled.df$lny))
# Metric Performance review
data.frame(RMSE = metric.procespin.rmse, RMSE.Pred = metric.procespin.rmsep)
```

```
##                        RMSE RMSE.Pred
## lm.full               0.547      1.28
## allpossible.best.adjr 0.561      1.20
## allpossible.best.aic  0.583      1.45
## stepwise.aic          0.623      1.01
```

Although the RMSE of the stepwise AIC model is the worst compared to previous model, it achieves the best Prediction results so far. But with such highly correlated predictors, that the best we can do and novel mothods should be investigated to address the Multicolinearity issue. We will use Principal Component Regression (PCR) and Partial Least Square Regression (PLSR) as a new strategy.

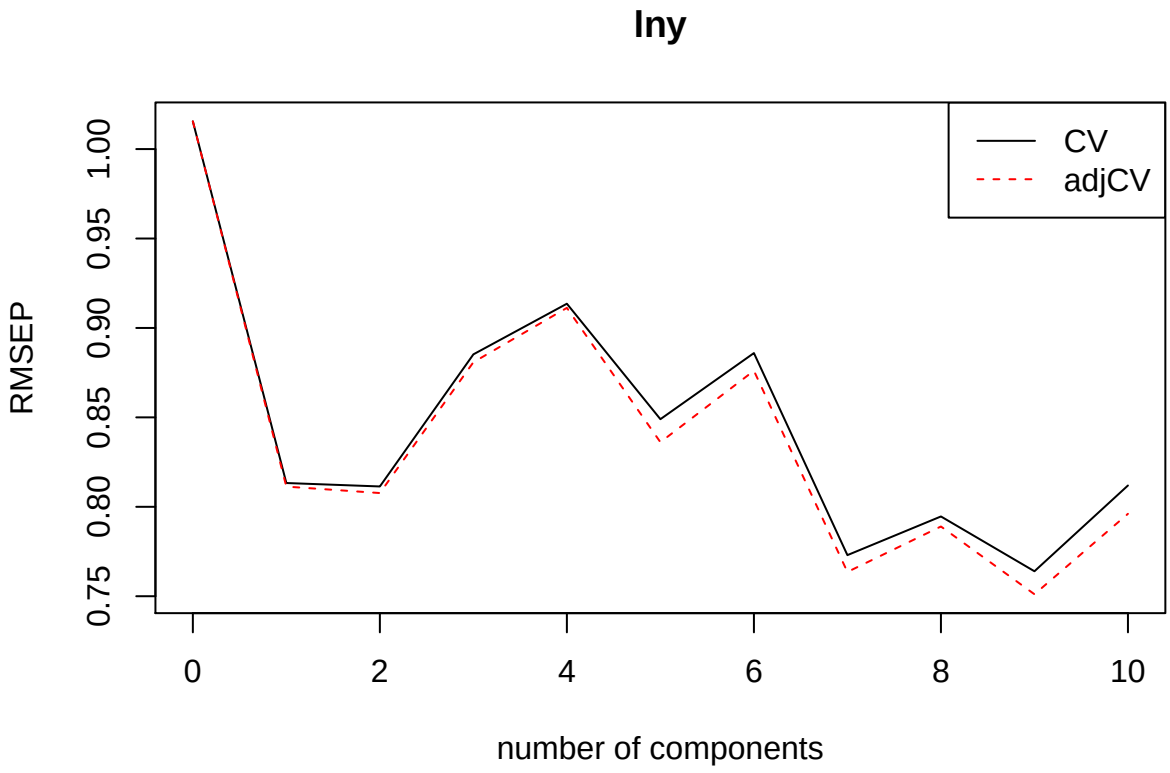
1.6 Principal Component Regression (PCR)

The principal component regression (PCR) first applies Principal Component Analysis on the data set to summarize the original predictor variables into few new variables also known as principal components (PCs), which are a linear combination of the original data. These PCs are then used to build the linear regression model. The number of principal components, to incorporate in the model, is chosen by cross-validation (cv). PCR is suitable when the data set contains highly correlated predictors, which is the case for our procespin dataset. In order to find the best number of of PCs to retain, we run a 10-fold cross validation procedure and comprare the RMSE of prediction on the validation segment.

```
set.seed(123)
model.pcr <- pcr(lny ~ ., data = procespin.scaled.df, validation = "CV")
summary(model.pcr)
```

```
## Data:    X dimension: 33 10
##  Y dimension: 33 1
## Fit method: svdpc
## Number of components considered: 10
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           1.016   0.8133   0.8114   0.8852   0.9135   0.8490   0.8859
## adjCV        1.016   0.8113   0.8077   0.8808   0.9112   0.8361   0.8761
##        7 comps  8 comps  9 comps  10 comps
## CV      0.7730   0.7946   0.7640    0.8119
## adjCV   0.7636   0.7890   0.7511    0.7961
##
## TRAINING: % variance explained
##      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X      46.32    61.60    74.22    84.14    91.75    97.33    98.93
## lny    38.01    39.92    40.12    41.18    49.79    49.99    62.61
##      8 comps  9 comps  10 comps
## X      99.47    99.90    100.00
## lny    62.94    68.68     69.19
```

```
validationplot(model.pcr, legendpos = "topright")
```




```
RMSEP(model.pcr, newdata = procespinsup.scaled.df)
```

```
## (Intercept)      1 comps      2 comps      3 comps      4 comps
##       1.252        1.313        1.278        1.268        1.273
##     5 comps      6 comps      7 comps      8 comps      9 comps
##       1.255        1.290        1.016        1.000        1.203
##    10 comps
##       1.283
```

Based on the prediction performances, the 8 PCs model seems to perform the best one. We notice that dimension reduction is not that significant, from 10 down to 8. Let's save the model with 8 PCs for future comparisons

```
# Metric for Principal Component regression
# fitted values root mean square error
metric.procespin.rmse <- c(metric.procespin.rmse, pcr = RMSE(predict(model.pcr, ncomp = 8), procespin.s
# predicted values root mean square error
k <- predict(model.pcr, ncomp = 8, newdata = procespinsup.scaled.df)
metric.procespin.rmsep <- c(metric.procespin.rmsep, pcr = RMSE(k, procespinsup.scaled.df$lny))
# Metric Performance review
data.frame(RMSE = metric.procespin.rmse, RMSE.Pred = metric.procespin.rmsep)
```

```
##                        RMSE RMSE.Pred
## lm.full               0.547      1.28
## allpossible.best.adjr 0.561      1.20
## allpossible.best.aic  0.583      1.45
## stepwise.aic          0.623      1.01
## pcr                   0.599      1.00
```

1.7 Partial Least Square Regression (PLSR)

A possible drawback of PCR is that we have no guarantee that the selected principal components are associated with the outcome. Here, the selection of the principal components to incorporate in the model is not supervised by the outcome variable.

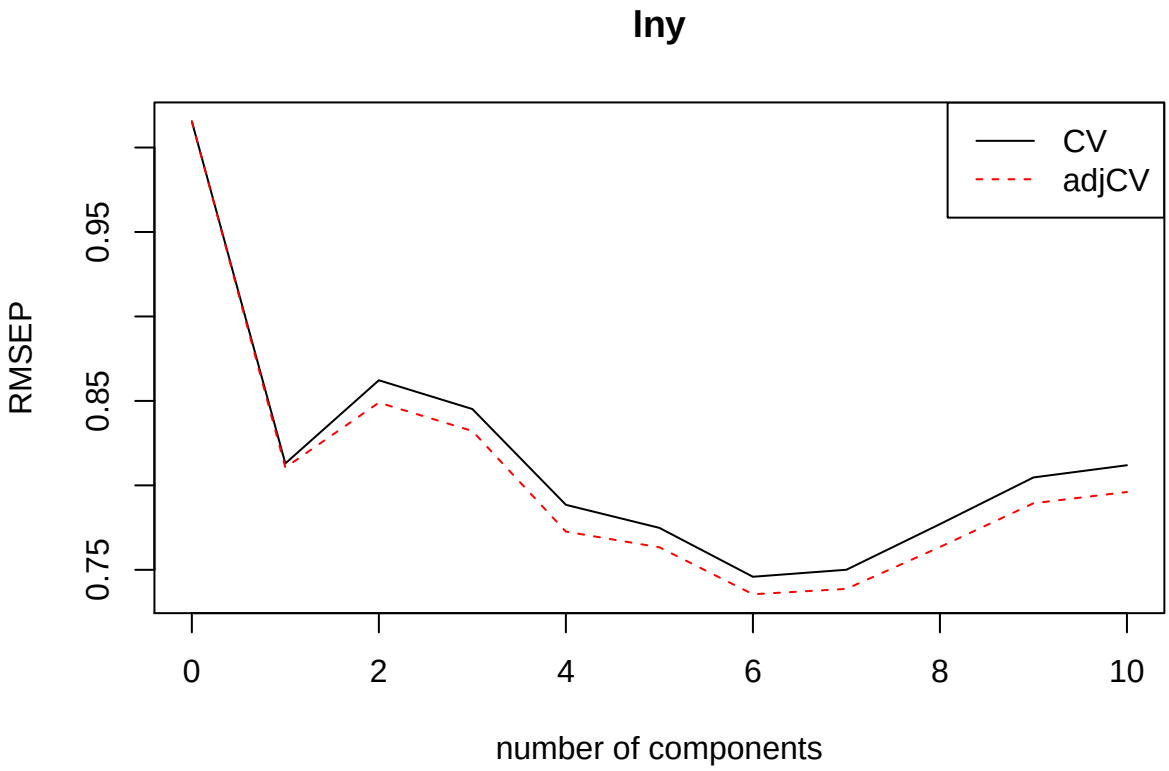
An alternative to PCR is the Partial Least Squares (PLS) regression, which identifies new principal components that not only summarizes the original predictors, but also that are related to the outcome. These components are then used to fit the regression model. So, compared to PCR, PLS uses a dimension reduction strategy that is supervised by the outcome.

Like PCR, PLS is convenient for data with highly-correlated predictors. The number of PCs used in PLS is generally chosen by cross-validation. Predictors and the outcome variables should be generally standardized, to make the variables comparable.

```
set.seed(123)
model.plsr <- plsr(lny ~ ., data = procespin.scaled.df, validation = "CV")
summary(model.plsr)
```

```
## Data:    X dimension: 33 10
##  Y dimension: 33 1
## Fit method: kernelpls
## Number of components considered: 10
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           1.016   0.8130   0.8622   0.8452   0.7885   0.7748   0.7459
## adjCV        1.016   0.8106   0.8490   0.8322   0.7726   0.7633   0.7355
##        7 comps  8 comps  9 comps  10 comps
## CV      0.7500   0.7770   0.8047    0.8119
## adjCV   0.7387   0.7635   0.7894    0.7961
##
## TRAINING: % variance explained
##      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X      45.84    56.29    66.93    71.46    81.25    87.77    93.76
## lny    43.32    55.08    59.99    65.93    66.57    67.67    68.74
##      8 comps  9 comps  10 comps
## X      99.36    99.64    100.00
## lny    68.94    69.13     69.19
```

```
validationplot(model.plsr, legendpos = "topright")
```




```
RMSEP(model.plsr, newdata = procespinsup.scaled.df)
```

```
## (Intercept)      1 comps      2 comps      3 comps      4 comps
##       1.252        1.279        1.177        1.127        1.068
##     5 comps      6 comps      7 comps      8 comps      9 comps
##       1.046        1.105        1.191        1.226        1.290
##    10 comps
##       1.283
```

Here the model with 5 component acheives the best prediction results on the test dataset.

```
# Metric for Partial Least Square Regression
# fitted values root mean square error
metric.procespin.rmse <- c(metric.procespin.rmse, plsr = RMSE(predict(model.plsr, ncomp = 5), procespin
# predicted values root mean square error
k <- predict(model.plsr, ncomp = 5, newdata = procespinsup.scaled.df)
metric.procespin.rmsep <- c(metric.procespin.rmsep, plsr = RMSE(k, procespinsup.scaled.df$lny))
# Metric Performance review
data.frame(RMSE = metric.procespin.rmse, RMSE.Pred = metric.procespin.rmsep)
```

```
##                          RMSE RMSE.Pred
## lm.full                 0.547      1.28
## allpossible.best.adjr 0.561      1.20
## allpossible.best.aic  0.583      1.45
## stepwise.aic          0.623      1.01
## pcr                   0.599      1.00
## plsr                  0.569      1.05
```

1.8 Penalized Ridge Regression

Ridge regression shrinks the regression coefficients, so that variables, with minor contribution to the outcome, have their coefficients close to zero. The shrinkage of the coefficients is achieved by penalizing the regression model with a penalty term called L2-norm, which is the sum of the squared coefficients. The amount of the penalty can be fine-tuned using a constant called lambda (λ). Selecting a good value for λ is critical.

```
# Find the best lambda using cross-validation
set.seed(123)
cv.ridge <- cv.glmnet(model.matrix(lny~., procespin.scaled.df)[,-1], procespin.scaled.df$lny, alpha = 0
# Display the best lambda value
cv.ridge$lambda.min
```

```
## [1] 0.0422
```

```
# Fit the final model on the training data
model.ridge <- glmnet(model.matrix(lny~., procespin.scaled.df)[,-1], procespin.scaled.df$lny, alpha = 0
```

We finish by computing the achieved performances

```
# Metric for Ridge Regression
# fitted values root mean square error
k <- as.vector(predict(model.ridge, newx = model.matrix(lny~., procespin.scaled.df)[,-1]))
metric.procespin.rmse <- c(metric.procespin.rmse, ridge = RMSE(k, procespin.scaled.df$lny))
# predicted values root mean square error
k <- as.vector(predict(model.ridge, newx = model.matrix(lny~., procespinsup.scaled.df)[,-1]))
metric.procespin.rmsep <- c(metric.procespin.rmsep, ridge = RMSE(k, procespinsup.scaled.df$lny))
# Metric Performance review
data.frame(RMSE = metric.procespin.rmse, RMSE.Pred = metric.procespin.rmsep)
```

```
##                          RMSE RMSE.Pred
## lm.full                 0.547      1.28
## allpossible.best.adjr 0.561      1.20
## allpossible.best.aic  0.583      1.45
## stepwise.aic            0.623      1.01
## pcr                     0.599      1.00
## plsr                    0.569      1.05
## ridge                   0.568      1.09
```

1.9 Penalized Lasso Regression

Lasso stands for Least Absolute Shrinkage and Selection Operator. It shrinks the regression coefficients toward zero by penalizing the regression model with a penalty term called L1-norm, which is the sum of the absolute coefficients. In the case of lasso regression, the penalty has the effect of forcing some of the coefficient estimates, with a minor contribution to the model, to be exactly equal to zero. This means that, lasso can be also seen as an alternative to the subset selection methods for performing variable selection in order to reduce the complexity of the model.

```
# Find the best lambda using cross-validation
set.seed(123)
cv.lasso <- cv.glmnet(model.matrix(lny~., procespin.scaled.df)[,-1], procespin.scaled.df$lny, alpha = 1
# Display the best lambda value
cv.lasso$lambda.min
```

```
## [1] 0.0154
```

```
# Fit the final model on the training data
model.lasso <- glmnet(model.matrix(lny~., procespin.scaled.df)[,-1], procespin.scaled.df$lny, alpha = 0
```

The Performance Metrics associated with

```
# Metric for Ridge Regression
# fitted values root mean square error
k <- as.vector(predict(model.lasso, newx = model.matrix(lny~., procespin.scaled.df)[,-1]))
metric.procespin.rmse <- c(metric.procespin.rmse, lasso = RMSE(k, procespin.scaled.df$lny))
# predicted values root mean square error
k <- as.vector(predict(model.lasso, newx = model.matrix(lny~., procespinsup.scaled.df)[,-1]))
metric.procespin.rmsep <- c(metric.procespin.rmsep, lasso = RMSE(k, procespinsup.scaled.df$lny))
# Metric Performance review
data.frame(RMSE = metric.procespin.rmse, RMSE.Pred = metric.procespin.rmsep)
```

```
##                          RMSE RMSE.Pred
## lm.full                 0.547      1.28
## allpossible.best.adjr 0.561      1.20
## allpossible.best.aic  0.583      1.45
## stepwise.aic            0.623      1.01
## pcr                     0.599      1.00
## plsr                    0.569      1.05
## ridge                   0.568      1.09
## lasso                   0.553      1.15
```

1.10 Random Forest Regression

Random Forest algorithm, is one of the most commonly used and the most powerful machine learning techniques. It is a special type of bagging applied to decision trees. We apply the Random Forest Algorithm without fine tuning first using the square root of number of predictor (mtry = 3)to select the split variables.

```
model.rf <- randomForest(lny ~ ., data = procespin.scaled.df, ntree = 500, mtry = 3, importance = TRUE)
model.rf
```

```
##
## Call:
##  randomForest(formula = lny ~ ., data = procespin.scaled.df, ntree = 500,      mtry = 3, importance =
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 0.735
##                     % Var explained: 24.2
```

The Performance metrics:

```
# Metric for Partial Least Square Regression
# fitted values root mean square error
k <- predict(model.rf, procespin.scaled.df)
metric.procespin.rmse <- c(metric.procespin.rmse, rf = RMSE(k, procespin.scaled.df$lny))
# predicted values root mean square error
k <- predict(model.rf, procespinsup.scaled.df)
metric.procespin.rmsep <- c(metric.procespin.rmsep, rf = RMSE(k, procespinsup.scaled.df$lny))
# Metric Performance review
data.frame(RMSE = metric.procespin.rmse, RMSE.Pred = metric.procespin.rmsep)
```

```
##                          RMSE RMSE.Pred
## lm.full                 0.547      1.28
## allpossible.best.adjr 0.561      1.20
## allpossible.best.aic  0.583      1.45
```

```
## stepwise.aic          0.623      1.01
## pcr                   0.599      1.00
## plsr                  0.569      1.05
## ridge                 0.568      1.09
## lasso                 0.553      1.15
## rf                    0.384      1.24
```

The former Random Forest model was not tuned (number of variables used at each split was fixed to be 3). We try to optimise the model by performing a grid search over the number of variables for splits (mtry), the split rule, and finally the minimun node size. Cross validation will be used to selected the best Hyper-parameters

```
# Fit the model on the training set
set.seed(123)
rf_grid <- expand.grid(mtry = seq(2:7), splitrule = c("variance", "extratrees"), min.node.size = c(1,3,5
model.rf.tuned <- train(
  lny ~., data = procespin.scaled.df, method = "ranger",
  trControl = trainControl("repeatedcv", number = 5, repeats=3),
  tuneGrid = rf_grid,
  verbose = FALSE)
# Best tuning parameter mtry
model.rf.tuned$bestTune
```

```
##    mtry splitrule min.node.size
## 15    2  variance             9
```

```
# final model
model.rf.tuned$finalModel
```

```
## Ranger result
##
## Call:
##  ranger::ranger(dependent.variable.name = ".outcome", data = x,      mtry = param$mtry, min.node.size
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      33
## Number of independent variables:  10
## Mtry:                             2
## Target node size:                 9
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.666
## R squared (OOB):                  0.334
```

```
# Make predictions on the test data
predictions <- predict(model.rf.tuned, procespinsup.scaled.df)
# Compute the average prediction error RMSE
RMSE(predictions, procespinsup.scaled.df$lny)
```

```
## [1] 1.23
```

Did we achieve better performances ?

```
# Metric for Partial Least Square Regression
# fitted values root mean square error
k <- predict(model.rf.tuned, procespin.scaled.df)
metric.procespin.rmse <- c(metric.procespin.rmse, rf.tuned = RMSE(k, procespin.scaled.df$lny))
# predicted values root mean square error
```

```
k <- predict(model.rf.tuned, procespinsup.scaled.df)
metric.procespin.rmsep <- c(metric.procespin.rmsep, rf.tuned = RMSE(k, procespinsup.scaled.df$lny))
# Metric Performance review
data.frame(RMSE = metric.procespin.rmse, RMSE.Pred = metric.procespin.rmsep)
```

```
##                          RMSE RMSE.Pred
## lm.full                 0.547      1.28
## allpossible.best.adjr 0.561      1.20
## allpossible.best.aic  0.583      1.45
## stepwise.aic          0.623      1.01
## pcr                   0.599      1.00
## plsr                  0.569      1.05
## ridge                 0.568      1.09
## lasso                 0.553      1.15
## rf                    0.384      1.24
## rf.tuned              0.476      1.23
```

1.11 Random Forest with PCA projected predictors

In this section we get a bit fancier and we try to combine PCA technique to produce a new set of uncorrelated preditors that the random forest would operated on:

```
# PCA transformation
pca.preprocess <- preProcess(procespin.df[-1], method = c("pca"), pcaComp = 10)
procespin.pca.df <- cbind(procespin.df[1], predict(pca.preprocess, procespin.df[-1]))
procespinsup.pca.df <- cbind(procespinsup.df[1], predict(pca.preprocess, procespinsup.df[-1]))

# Fit the model on the training set
set.seed(123)
rf_grid <- expand.grid(mtry = seq(2:11), splitrule = c("variance", "maxstat"), min.node.size = c(1,3,5,
model.rf.pca <- train(
  lny ~., data = procespin.pca.df, method = "ranger",
  trControl = trainControl("repeatedcv", number = 5, repeats=3),
  tuneGrid = rf_grid,
  verbose = FALSE)
# Best tuning parameter mtry
model.rf.pca$bestTune
```

```
##     mtry splitrule min.node.size
## 133   10  variance            13
```

```
# final model
model.rf.pca$finalModel
```

```
## Ranger result
##
## Call:
##  ranger::ranger(dependent.variable.name = ".outcome", data = x,      mtry = param$mtry, min.node.size
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      33
## Number of independent variables:  10
## Mtry:                             10
## Target node size:                 13
```

```
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.962
## R squared (OOB):                  0.379
```

```
# Make predictions on the test data
predictions <- predict(model.rf.pca, procespinsup.pca.df)
# Compute the average prediction error RMSE
RMSE(predictions, procespinsup.pca.df$lny)
```

```
## [1] 1.72
```

As usual the performance Metrics:

```
# Metric for Partial Least Square Regression
# fitted values root mean square error
k <- predict(model.rf.pca, procespin.pca.df)
metric.procespin.rmse <- c(metric.procespin.rmse, rf.pca = RMSE(k, procespin.pca.df$lny))
# predicted values root mean square error
k <- predict(model.rf.pca, procespinsup.pca.df)
metric.procespin.rmsep <- c(metric.procespin.rmsep, rf.pca = RMSE(k, procespinsup.pca.df$lny))
# Metric Performance review
data.frame(RMSE = metric.procespin.rmse, RMSE.Pred = metric.procespin.rmsep)
```

```
##                         RMSE RMSE.Pred
## lm.full                0.547      1.28
## allpossible.best.adjr  0.561      1.20
## allpossible.best.aic   0.583      1.45
## stepwise.aic           0.623      1.01
## pcr                    0.599      1.00
## plsr                   0.569      1.05
## ridge                  0.568      1.09
## lasso                  0.553      1.15
## rf                     0.384      1.24
## rf.tuned               0.476      1.23
## rf.pca                 0.603      1.72
```

1.12 Boosted Trees

Boosting, which is similar to the bagging method, except that the trees are grown sequentially: each successive tree is grown using information from previously grown trees, with the aim to minimize the error of the previous models.

```
# Fit the model on the training set
set.seed(123)
model.xgbtree <- train(
  lny ~., data = procespin.scaled.df, method = "xgbTree",
  trControl = trainControl("repeatedcv", number = 5, repeats=3),
  verbose = FALSE)
# Best tuning parameter mtry
model.xgbtree$bestTune
```

```
##   nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 4      50         1 0.3     0              0.6                1      0.75
```

```
# final model
model.xgbtree$finalModel
```

```
## ##### xgb.Booster
## raw: 13.1 Kb
## call:
##   xgboost::xgb.train(params = list(eta = param$eta, max_depth = param$max_depth,
##     gamma = param$gamma, colsample_bytree = param$colsample_bytree,
##     min_child_weight = param$min_child_weight, subsample = param$subsample),
##     data = x, nrounds = param$nrounds, verbose = FALSE, objective = "reg:linear")
## params (as set within xgb.train):
##   eta = "0.3", max_depth = "1", gamma = "0", colsample_bytree = "0.6", min_child_weight = "1", subsa
## xgb.attributes:
##   niter
## # of features: 10
## niter: 50
## nfeatures : 10
## xNames : x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
## problemType : Regression
## tuneValue :
##    nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 4      50         1 0.3     0              0.6                1      0.75
## obsLevels : NA
## param :
##  $verbose
## [1] FALSE
```

```
# Make predictions on the test data
predictions <- predict(model.xgbtree, procespinsup.scaled.df)
# Compute the average prediction error RMSE
RMSE(predictions, procespinsup.scaled.df$lny)
```

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

Let's inspect the performance metrics:

```
# Metric for Partial Least Square Regression
# fitted values root mean square error
k <- predict(model.xgbtree, procespin.scaled.df)
metric.procespin.rmse <- c(metric.procespin.rmse, xgbtree = RMSE(k, procespin.scaled.df$lny))
# predicted values root mean square error
k <- predict(model.xgbtree, procespinsup.scaled.df)
metric.procespin.rmsep <- c(metric.procespin.rmsep, xgbtree = RMSE(k, procespinsup.scaled.df$lny))
# Metric Performance review
data.frame(RMSE = metric.procespin.rmse, RMSE.Pred = metric.procespin.rmsep)
```

```
##                         RMSE RMSE.Pred
## lm.full                0.547      1.28
## allpossible.best.adjr  0.561      1.20
## allpossible.best.aic   0.583      1.45
## stepwise.aic           0.623      1.01
## pcr                    0.599      1.00
## plsr                   0.569      1.05
## ridge                  0.568      1.09
## lasso                  0.553      1.15
## rf                     0.384      1.24
## rf.tuned               0.476      1.23
## rf.pca                 0.603      1.72
## xgbtree                0.286      1.00
```

1.13 Conclusion

In this exercice we adopted a **predictive modeling** strategy using a training dataset and measuring predictive performance on the test dataset. This kind of strategy lead us trying a large pannel of modeling techniques wihtout digging too deep into the model assumptions of each. For pure prediction purposes, the model is the **Gradient Boosted Trees** (RMSE of Prediction = 1.00) at the same performance with the **Principal Component Regression**. But the Boosted Tree outperforms the PCR model in Cross-validated RMSE (0.286 vs 0.599). We expected that Random Forest Model woul deliver a descent Prediction performance based on the low Cross-Validated RMSE, but it turns out that The model was one of the worst performing one (1.23) sign of over-fitting. The key observation during all these model head-to-head comparison is that with such low number of observations in the validation dataset (33 observation) it is really hard to deliver a good performing model in both fronts: cross-validation RMSE and Prediction RMSE.

2 Exercise 2: Swiss Fertility and Socioeconomic Indicators (1888) Data

2.1 Introduction

The swiss dataset (part of datasets base package) is a standardized fertility measure and socio-economic indicators for each of 47 French-speaking provinces of Switzerland at about 1888. It consists of a data frame with 47 observations on 6 variables, each of which is in percent, i.e., in [0, 100]. In this exercice, an ** Explanatory** modeling approach is considered.

2.2 Dataset Response and Predictors description

Fertility is the reponse variable. All variable are numeric (no factor) [,1] Fertility Ig, 'common standardized fertility measure' [,2] Agriculture % of males involved in agriculture as occupation [,3] Examination % draftees receiving highest mark on army examination [,4] Education % education beyond primary school for draftees. [,5] Catholic % 'catholic' (as opposed to 'protestant'). [,6] Infant.Mortality live births who live less than 1 year.

All variables but 'Fertility' give proportions of the population.

2.3 Exploratory Analysis

We Start by loading the tookit and the dataset.

```
library(PerformanceAnalytics)      # for dataframe correlation analysis
library(olsrr)                     # Linear Regression utility library
library(tidyverse)                 # Dataframe manipulation toolbox
library(caret)                     # for machine learning easier workflow
# Load the dataset
data("swiss")
```

As the predictor as all on the same percentage scale, no need to transform the data.

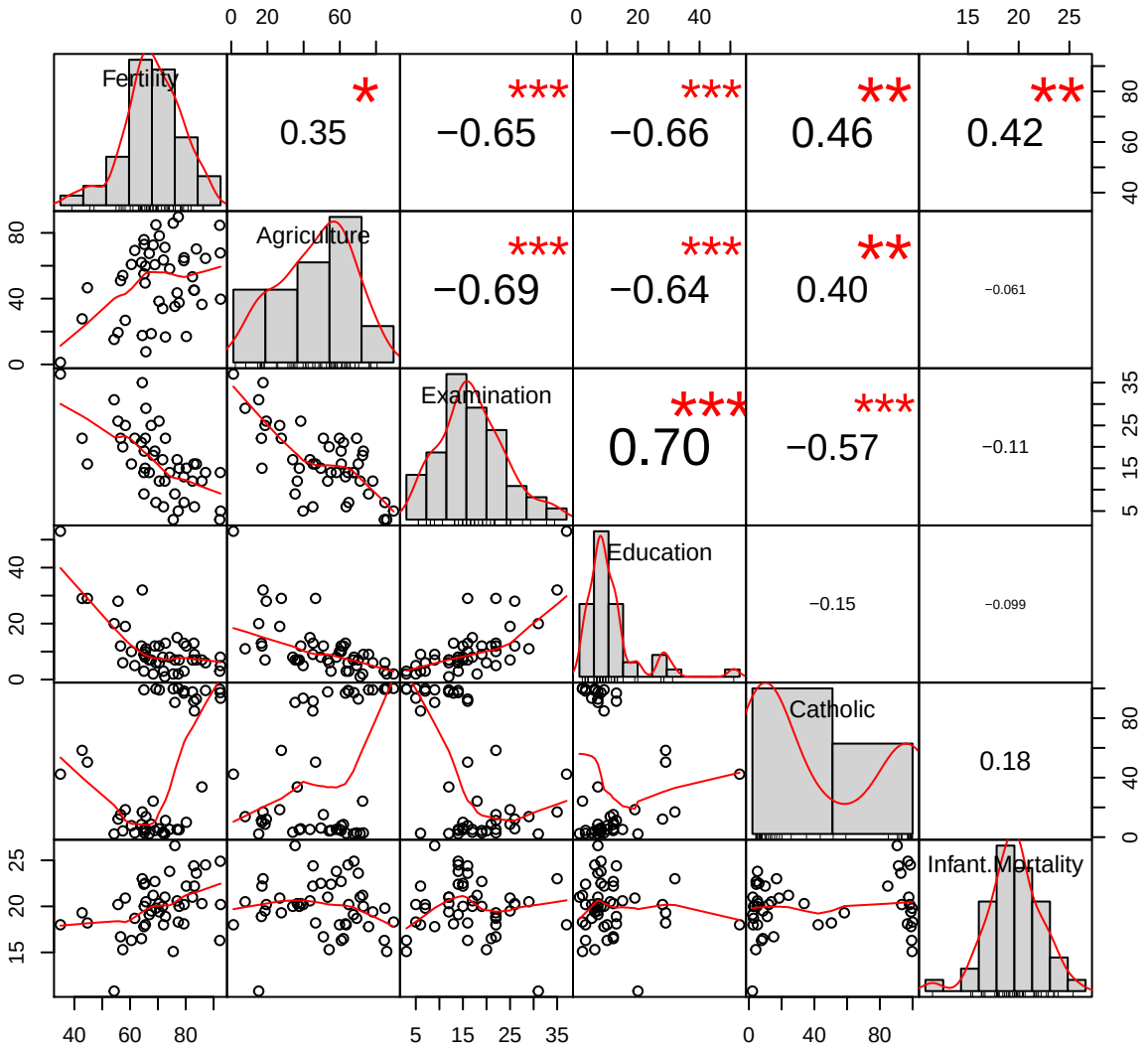
```
head(swiss)
```

```
##              Fertility Agriculture Examination Education Catholic
## Courtelary        80.2        17.0          15        12     9.96
## Delemont          83.1        45.1           6         9    84.84
## Franches-Mnt      92.5        39.7           5         5    93.40
## Moutier           85.8        36.5          12         7    33.77
## Neuveville        76.9        43.5          17        15     5.16
## Porrentruy        76.1        35.3           9         7    90.57
##              Infant.Mortality
```

```
## Courtelary              22.2
## Delemont                22.2
## Franches-Mnt            20.2
## Moutier                 20.3
## Neuveville              20.6
## Porrentruy              26.6
```

Let's have a look at the correlation matrix:

```
chart.Correlation(swiss)
```



Moderate correlation exist between **Agriculture**, **Examination** and **Education**. We will try to assess how severe is the Multicolinearity when we'll compute the **VIF** indicator.

2.4 Multiple Linear Regression: Full model

Let's compute the full model taking into account all predictors.

```
swiss.lm.full <- lm(Fertility ~ ., data = swiss)
summary(swiss.lm.full)
```

```
##
## Call:
## lm(formula = Fertility ~ ., data = swiss)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.274  -5.262   0.503   4.120  15.321
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       66.9152    10.7060    6.25  1.9e-07 ***
## Agriculture       -0.1721     0.0703   -2.45   0.0187 *
## Examination       -0.2580     0.2539   -1.02   0.3155
## Education         -0.8709     0.1830   -4.76  2.4e-05 ***
## Catholic           0.1041     0.0353    2.95   0.0052 **
## Infant.Mortality   1.0770     0.3817    2.82   0.0073 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.17 on 41 degrees of freedom
## Multiple R-squared:  0.707,  Adjusted R-squared:  0.671
## F-statistic: 19.8 on 5 and 41 DF,  p-value: 5.59e-10
```

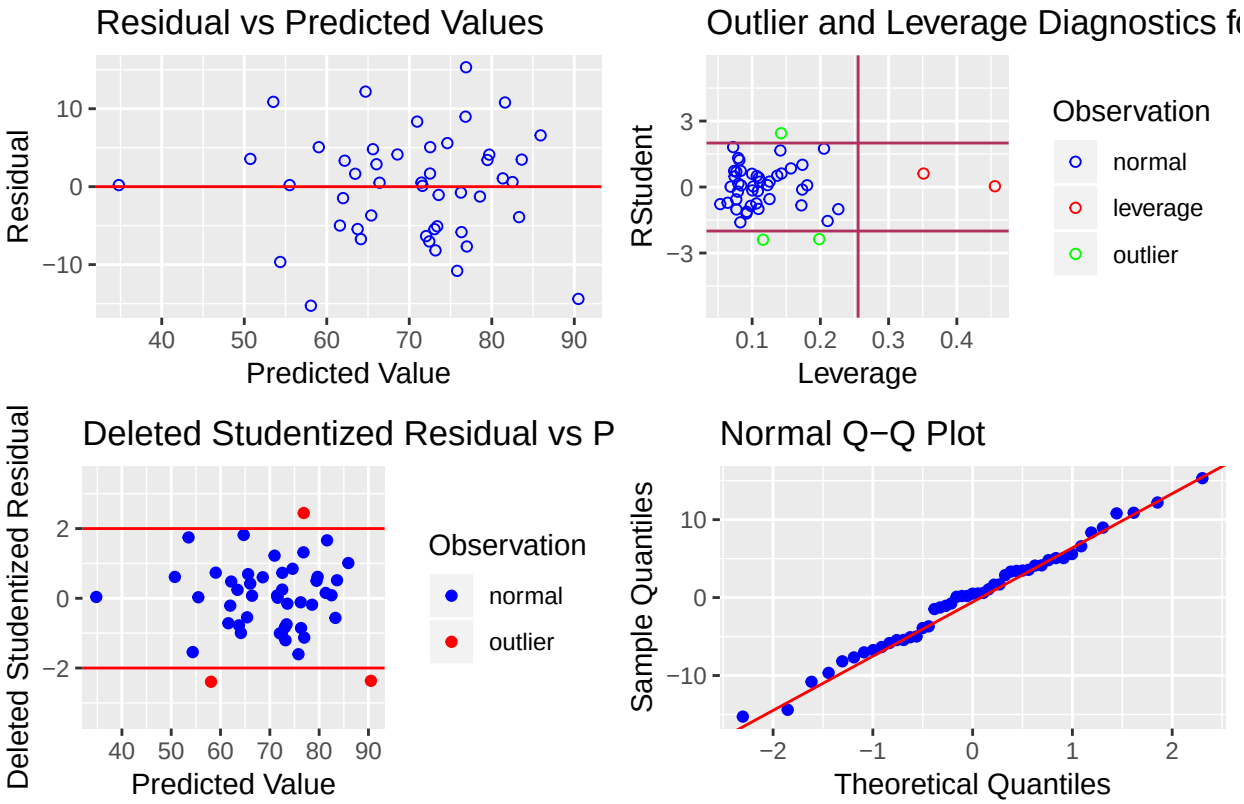
Only the Examination predictor turns out not being significant in the full model. the adjusted R-squared is relativelty good, the the F-statistic is remarkably significant. Let's have a look at the VIF indicator:

```
ols_vif_tol(swiss.lm.full)
```

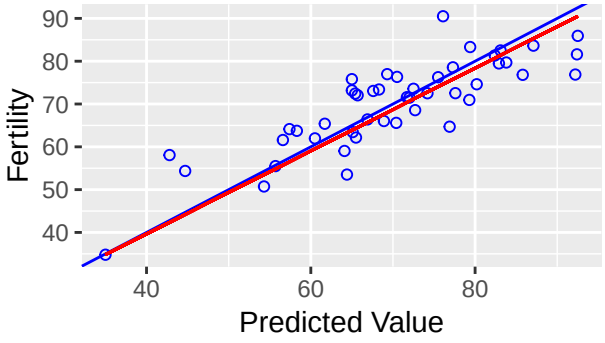
```
## # A tibble: 5 x 3
##   Variables        Tolerance   VIF
##   <chr>                <dbl> <dbl>
## 1 Agriculture          0.438  2.28
## 2 Examination          0.272  3.68
## 3 Education            0.360  2.77
## 4 Catholic             0.516  1.94
## 5 Infant.Mortality     0.903  1.11
```

No predictor exceeds the threshold of 5, meaning that the Multicolinearity is of limited effect. We produce the full diagnostic report to pay a special attention to residuals.

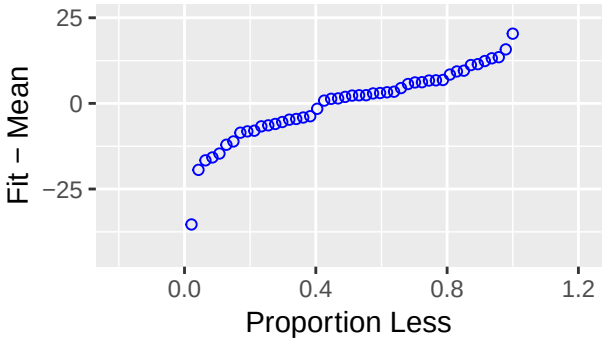
```
ols_plot_diagnostics(swiss.lm.full)
```


Residual vs Predicted Values
Residual
Predicted Value
Outlier and Leverage Diagnostics f
RStudent
Leverage
Observation
normal
leverage
outlier
Deleted Studentized Residual vs P
Deleted Studentized Residual
Predicted Value
Observation
normal
outlier
Normal Q−Q Plot
Sample Quantiles
Theoretical Quantiles

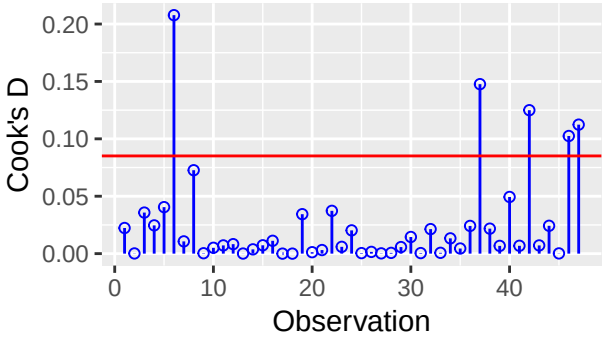
Observed by Predicted for Fertility



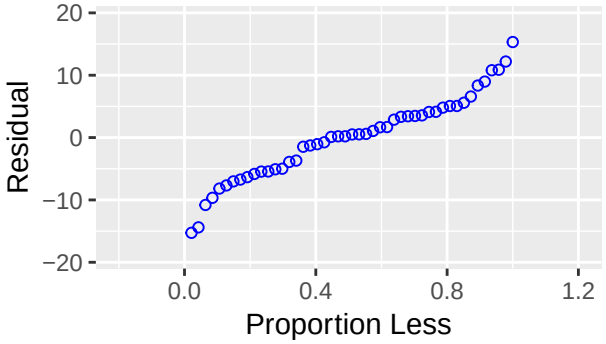
Residual Fit Spread Plot

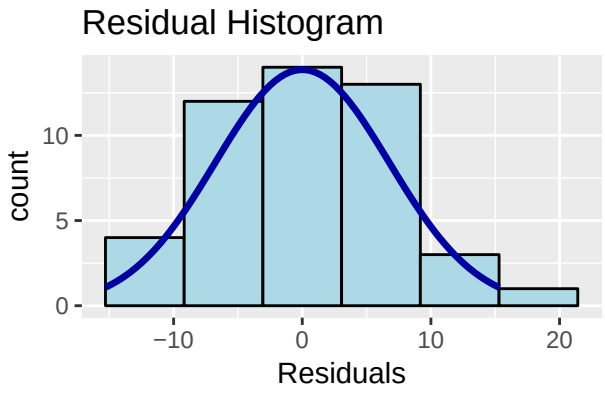


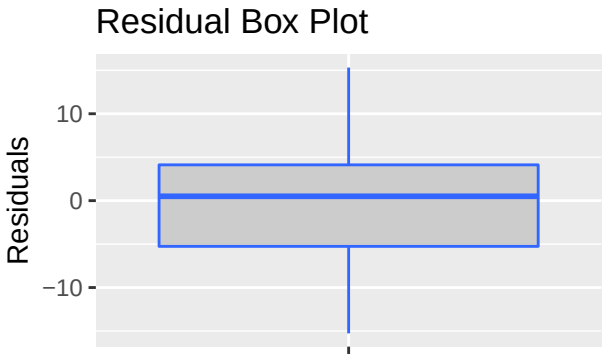
Cook's D Chart



Residual Fit Spread Plot







The Nornal Q-Q Plot hints for the normality of the residuals, we need to confirm it with a normality test. We don't see any suspicious pattern on the Residuals vs Predicted plot. the Cook's D chart highlights several outlier obsevations that we may remove to improve the modeling performance. To test the normality of the residuals we have the shapiro test (Null hypothesis, sample came from normal distribution)

```
ols_test_normality(swiss.lm.full)
```

```
## -----------------------------------------------
##        Test             Statistic       pvalue
## -----------------------------------------------
## Shapiro-Wilk              0.9889         0.9318
## Kolmogorov-Smirnov        0.0807         0.8952
## Cramer-von Mises          2.8583         0.0000
## Anderson-Darling          0.2316         0.7907
## -----------------------------------------------
```

test statistics shows that we fail to reject the null hypothesis so we can assume **NOrmality of the Residuals**. Now let's check the Homescedasticity of the Residuals thanks to a Breusch Pagan Test.

```
ols_test_breusch_pagan(swiss.lm.full)
```

```
##
##  Breusch Pagan Test for Heteroskedasticity
##  -----------------------------------------
##  Ho: the variance is constant
##  Ha: the variance is not constant
##
##                 Data
##  -----------------------------------
```

```
##  Response : Fertility
##  Variables: fitted values of Fertility
##
##       Test Summary
##  -------------------------
##  DF            =    1
##  Chi2          =    0.3077
##  Prob > Chi2   =    0.5791
```

we failt to reject the Null hypothesis so we can assume that the variance is constant.

2.5 Variable Selection using best subset

```
ols_step_best_subset(swiss.lm.full)
```

```
##                          Best Subsets Regression
## --------------------------------------------------------------------------
## Model Index    Predictors
## --------------------------------------------------------------------------
##      1         Education
##      2         Education Catholic
##      3         Education Catholic Infant.Mortality
##      4         Agriculture Education Catholic Infant.Mortality
##      5         Agriculture Examination Education Catholic Infant.Mortality
## --------------------------------------------------------------------------
##
##                                                  Subsets Regression Summary
## ----------------------------------------------------------------------------------------------------------
##                        Adj.        Pred
## Model    R-Square    R-Square    R-Square     C(p)        AIC         SBIC        SBC         MSEP
## ----------------------------------------------------------------------------------------------------------
##   1        0.4406      0.4282       0.398    35.2049    348.4223    213.1276    353.9727    93.1970
##   2        0.5745      0.5552       0.514    18.4862    337.5636    202.8359    344.9642    74.1870
##   3        0.6625      0.6390       0.596     8.1782    328.6684    195.2580    337.9192    61.6391
##   4        0.6993      0.6707        0.62     5.0328    325.2408    193.0144    336.3417    57.5954
##   5        0.7067      0.6710       0.608     6.0000    326.0716    194.4046    339.0226    58.9893
## ----------------------------------------------------------------------------------------------------------
## AIC: Akaike Information Criteria
##  SBIC: Sawa's Bayesian Information Criteria
##  SBC: Schwarz Bayesian Criteria
##  MSEP: Estimated error of prediction, assuming multivariate normality
##  FPE: Final Prediction Error
##  HSP: Hocking's Sp
##  APC: Amemiya Prediction Criteria
```

We notice that the model 4 (Agriculture Education Catholic Infant.Mortality) has a better Predicted R-Square, better AIC, SBIC and SBC. we can pick it as the best model.

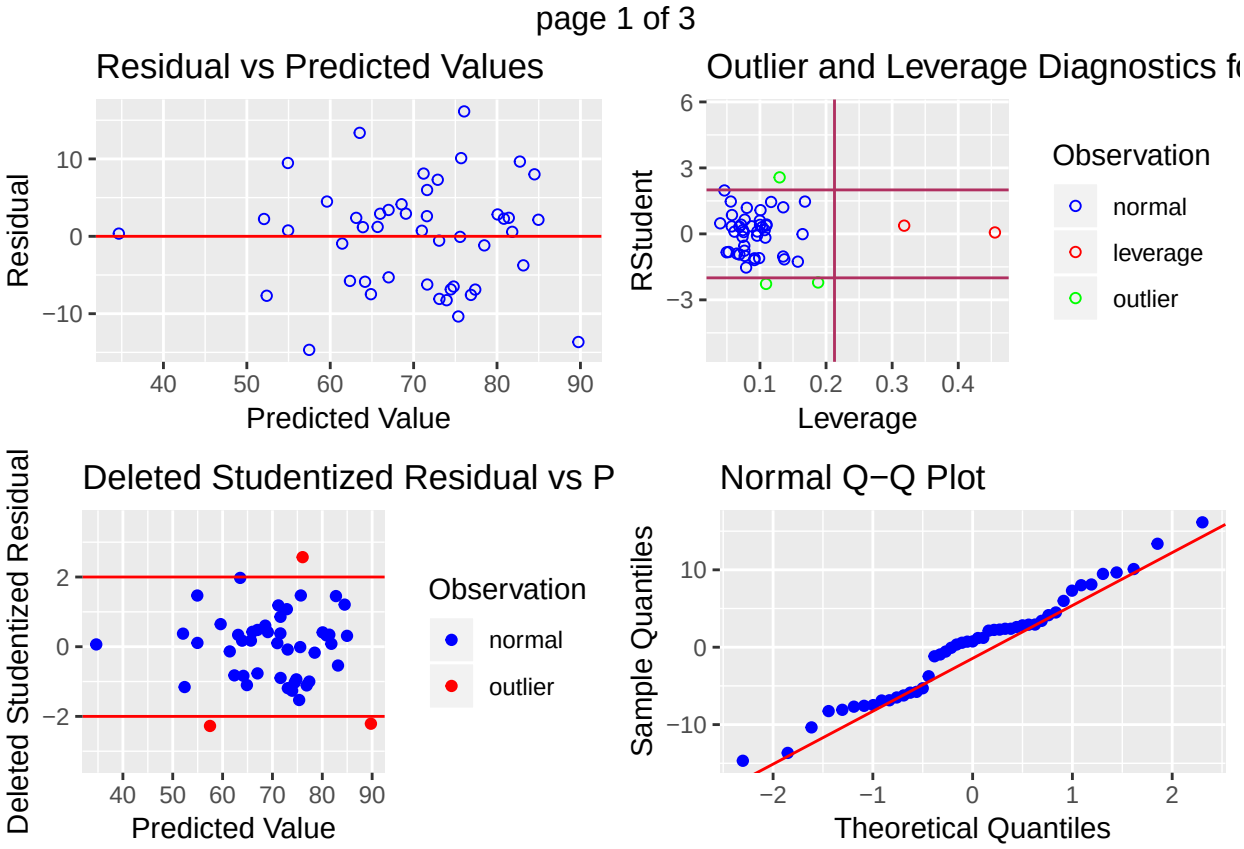
```
swiss.lm.best <- lm(Fertility ~ Agriculture + Education + Catholic + Infant.Mortality, data = swiss)
summary(swiss.lm.best)
```

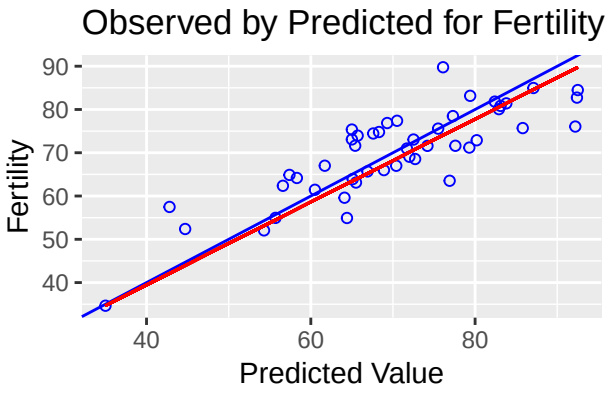
```
##
## Call:
## lm(formula = Fertility ~ Agriculture + Education + Catholic +
##     Infant.Mortality, data = swiss)
##
```

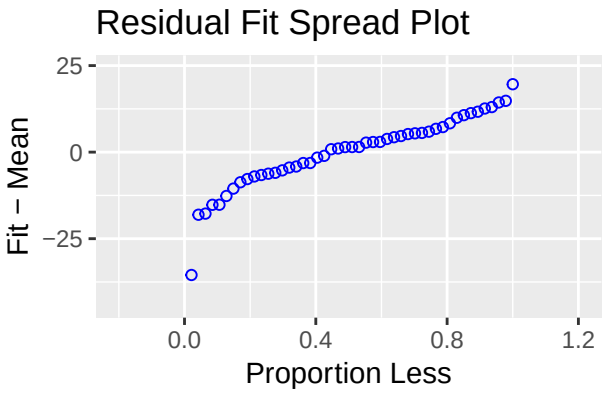
```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -14.676  -6.052   0.751   3.166  16.142
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       62.1013     9.6049    6.47  8.5e-08 ***
## Agriculture       -0.1546     0.0682   -2.27   0.0286 *
## Education         -0.9803     0.1481   -6.62  5.1e-08 ***
## Catholic           0.1247     0.0289    4.31  9.5e-05 ***
## Infant.Mortality   1.0784     0.3819    2.82   0.0072 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.17 on 42 degrees of freedom
## Multiple R-squared:  0.699,  Adjusted R-squared:  0.671
## F-statistic: 24.4 on 4 and 42 DF,  p-value: 1.72e-10
```

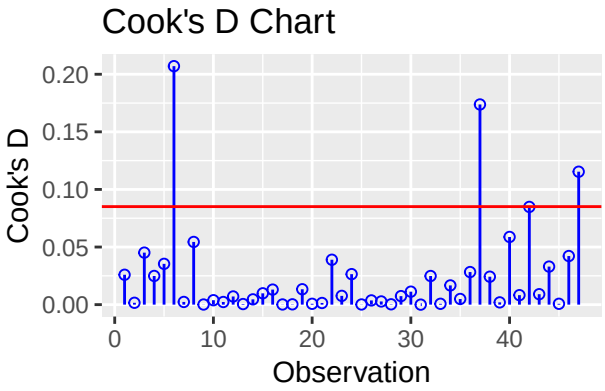
All the predictor's coefficients are significant. the adjusted R-square is 0.671. The diagnostic report:

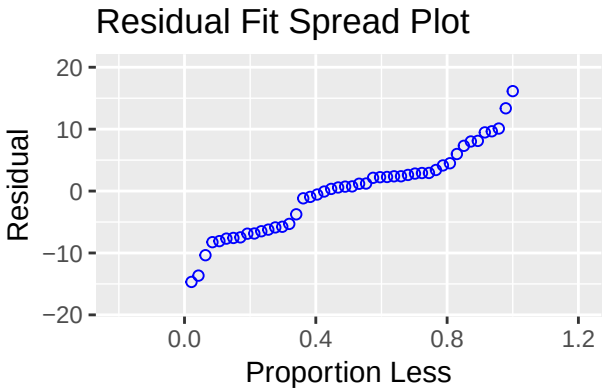
```
ols_plot_diagnostics(swiss.lm.best)
```

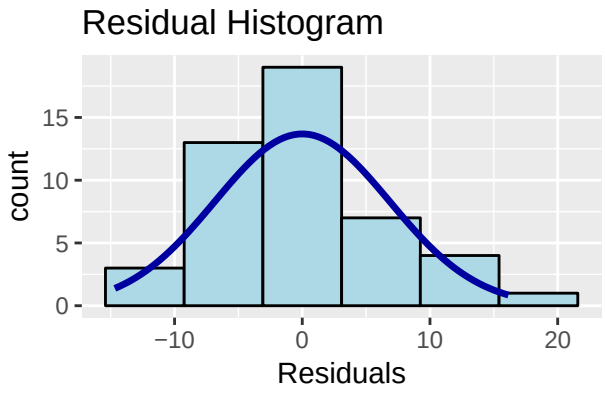


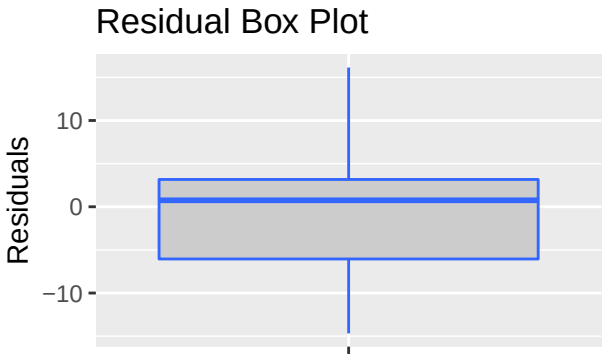

Observed by Predicted for Fertility
Fertility
Predicted Value
90
80
70
60
50
40
40
60
80


Residual Fit Spread Plot
Fit – Mean
Proportion Less
25
0
−25
0.0
0.4
0.8
1.2


Cook's D Chart
Cook's D
Observation
0.20
0.15
0.10
0.05
0.00
0
10
20
30
40


Residual Fit Spread Plot
Residual
Proportion Less
20
10
0
−10
−20
0.0
0.4
0.8
1.2





the Normal Q-Q Plot seems a bit altered compared to the previous full model. the residual Box Plot shows a slightly skewed residuals. let's tests the residuals for normality

```
ols_test_normality(swiss.lm.best)
```

```
## -----------------------------------------------
##        Test             Statistic       pvalue
## -----------------------------------------------
## Shapiro-Wilk              0.9766         0.4590
## Kolmogorov-Smirnov        0.0998         0.7003
## Cramer-von Mises          3.1859         0.0000
## Anderson-Darling          0.5536         0.1453
## -----------------------------------------------
```

Test statistics shows that we fail to reject the Null hypothesis which assumes the normality of the residuals distribution. Let's test the homoscedasticity of the residuals.

```
ols_test_breusch_pagan(swiss.lm.best)
```

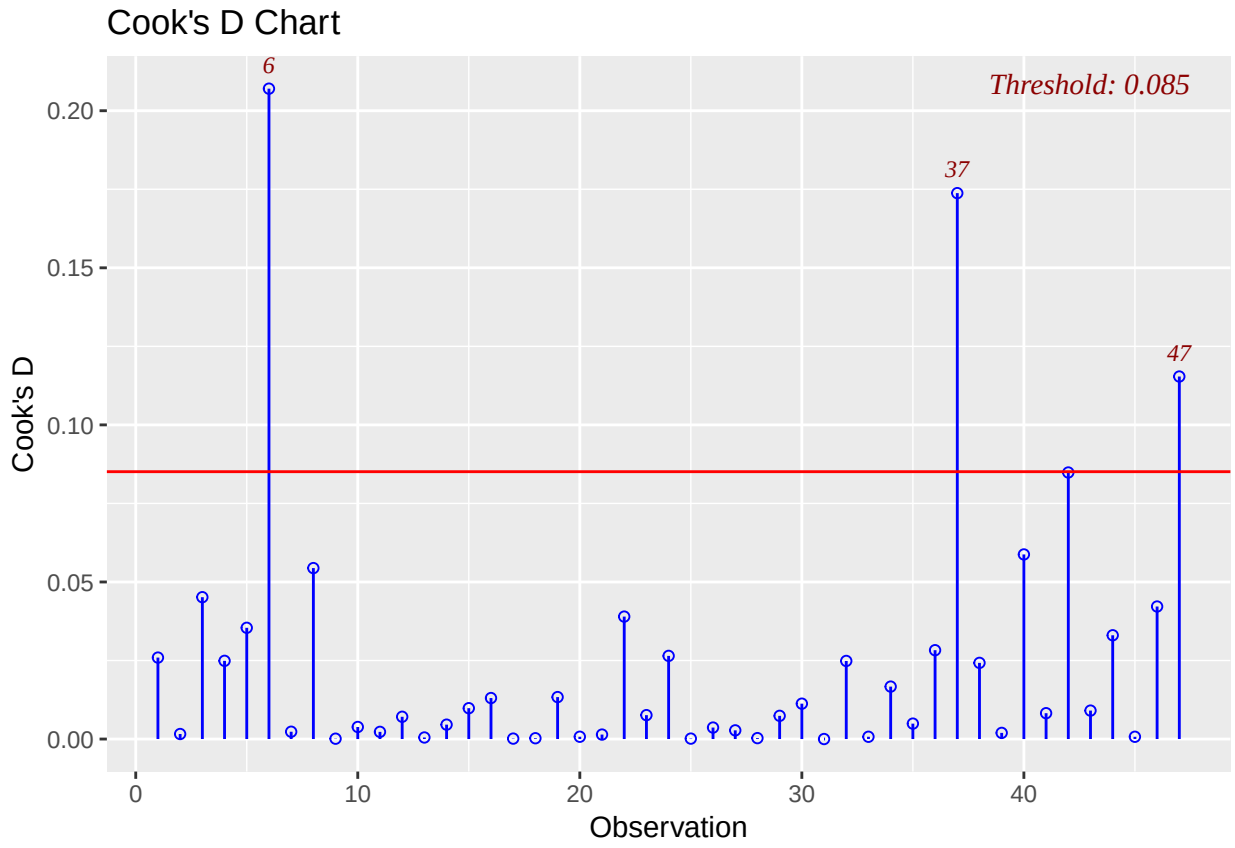
```
##
##  Breusch Pagan Test for Heteroskedasticity
##  -----------------------------------------
##  Ho: the variance is constant
##  Ha: the variance is not constant
##
##                 Data
##  -----------------------------------
##  Response : Fertility
##  Variables: fitted values of Fertility
```

```
##
##       Test Summary
##  -------------------------
##  DF            =    1
##  Chi2          =    0.4687
##  Prob > Chi2   =    0.4936
```

Again, we fail to reject the null hypothesis so we can assume that the variance is constance.

2.6 Study of Outliers removal

```
ols_plot_cooksd_chart(swiss.lm.best)
```



The observations 6, 37 and 47 clearly affect the model. We try to remove these outliers and refit the best model.

```
swiss.reduced <- swiss[-c(6,37,47),]
swiss.reduced.lm.best <- lm(Fertility ~ Agriculture + Education + Catholic + Infant.Mortality, data = sw
summary(swiss.reduced.lm.best)
```

```
##
## Call:
## lm(formula = Fertility ~ Agriculture + Education + Catholic +
##     Infant.Mortality, data = swiss.reduced)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -10.976  -4.736   0.706   3.710  12.538
```

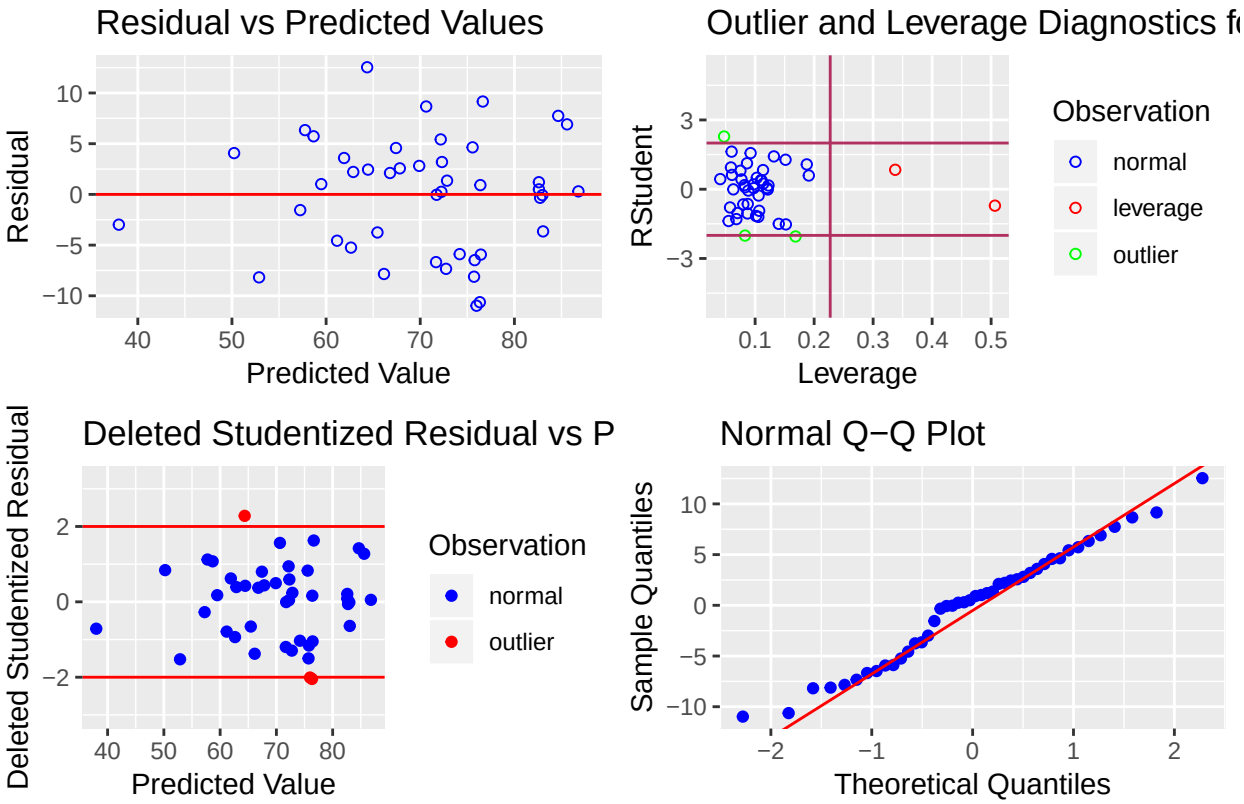
```
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       55.6776     8.0963    6.88  3.2e-08 ***
## Agriculture       -0.2012     0.0579   -3.48   0.0013 **
## Education         -0.9443     0.1268   -7.45  5.3e-09 ***
## Catholic           0.1317     0.0249    5.28  5.1e-06 ***
## Infant.Mortality   1.5010     0.3356    4.47  6.5e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.93 on 39 degrees of freedom
## Multiple R-squared:  0.768,  Adjusted R-squared:  0.745
## F-statistic: 32.3 on 4 and 39 DF,  p-value: 6.62e-12
```

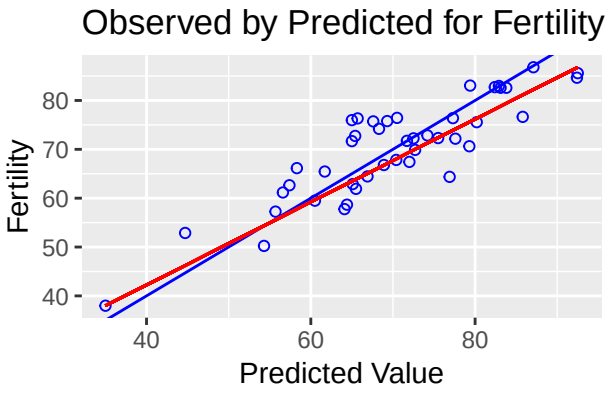
we notice a ** huge** improvement of the adjusted R-square (from 0.671 to 0.755), lower Residual standart error (from 7.17 down to 5.93). The coefficient and their significance level is also improved as summarized by this table:

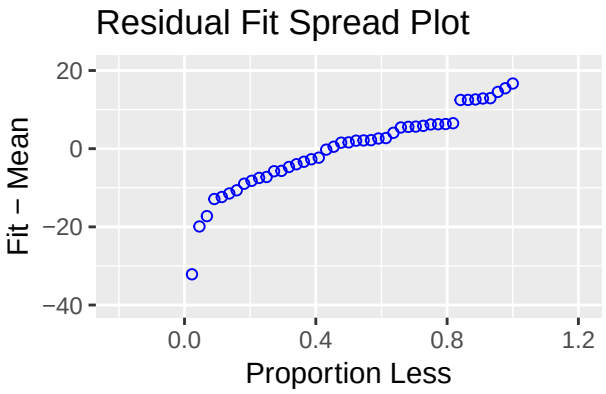
```
data.frame(lm.best = coef(swiss.lm.best) , reduced.lm.best = coef(swiss.reduced.lm.best), p.lm.best = su
```

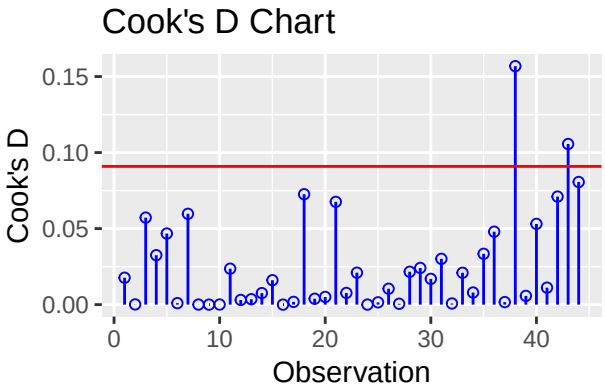
```
##                   lm.best reduced.lm.best p.lm.best p.reduced.lm.best
## (Intercept)        62.101          55.678  8.49e-08          3.16e-08
## Agriculture        -0.155          -0.201  2.86e-02          1.26e-03
## Education          -0.980          -0.944  5.14e-08          5.25e-09
## Catholic            0.125           0.132  9.50e-05          5.11e-06
## Infant.Mortality    1.078           1.501  7.22e-03          6.52e-05
```

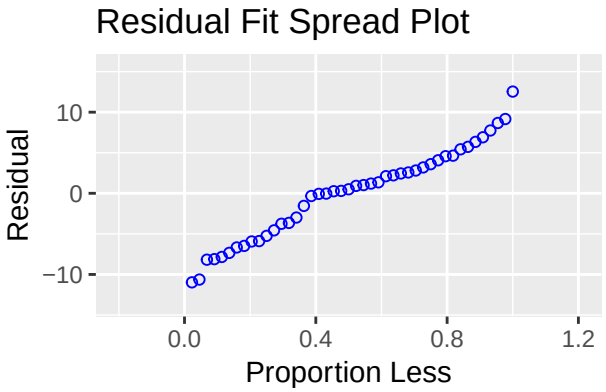
```
ols_plot_diagnostics(swiss.reduced.lm.best)
```

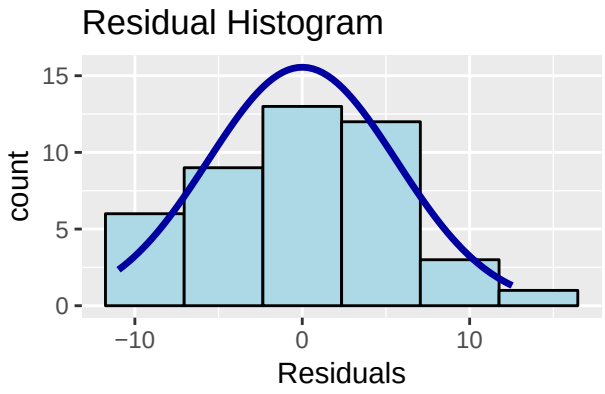

Residual vs Predicted Values
Residual
Predicted Value
Outlier and Leverage Diagnostics f
RStudent
Leverage
Observation
normal
leverage
outlier
Deleted Studentized Residual vs P
Deleted Studentized Residual
Predicted Value
Observation
normal
outlier
Normal Q−Q Plot
Sample Quantiles
Theoretical Quantiles

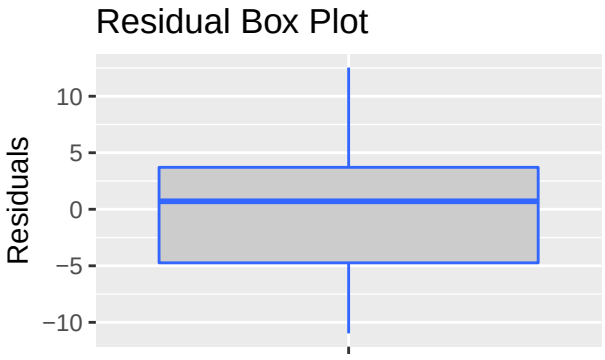

Observed by Predicted for Fertility
Fertility
80
70
60
50
40
40
60
80
Predicted Value


Residual Fit Spread Plot
Fit – Mean
20
0
−20
−40
0.0
0.4
0.8
1.2
Proportion Less


Cook's D Chart
Cook's D
0.15
0.10
0.05
0.00
0
10
20
30
40
Observation


Residual Fit Spread Plot
Residual
10
0
−10
0.0
0.4
0.8
1.2
Proportion Less





To assess the Residuals Normality:

```
ols_test_normality(swiss.reduced.lm.best)
```

```
## -----------------------------------------------
##        Test             Statistic       pvalue
## -----------------------------------------------
## Shapiro-Wilk              0.9781         0.5605
## Kolmogorov-Smirnov        0.1129         0.5892
## Cramer-von Mises          3.0025         0.0000
## Anderson-Darling          0.3868         0.3748
## -----------------------------------------------
```

Normality can be assumed as test statistics fails to reject null hypothesis. and to finish let's check the homoscedasticity of the residuals

```
ols_test_breusch_pagan(swiss.reduced.lm.best)
```

```
##
##  Breusch Pagan Test for Heteroskedasticity
##  -----------------------------------------
##  Ho: the variance is constant
##  Ha: the variance is not constant
##
##                 Data
##  -----------------------------------
##  Response : Fertility
##  Variables: fitted values of Fertility
##
```

```
##       Test Summary
##  -------------------------
##  DF            =    1
##  Chi2          =    0.0442
##  Prob > Chi2   =    0.8335
```

Homoscedasticity can be assumed as we fail to reject the Null Hypthesis

2.7 Conclusion

We used Multiple Linear Regression Modelin framework to model the swiss dataset with 47 observations on 6 variables. We were able to compute a best fitting model (in terms of adjusted R-square) using best-subset variable selection technique. we also assessed the model validity by checking the normality and the homoscedasticity of the residuals along all the **three** produced model. We finally identified 3 outlier observation thanks to Cook's D chart that helped us produced a more reliable model (in terms of better adjusted R-square, better coefficient p-value and a lower Residual standart error) where all the residual assumptions still holds true (normality and homoscedasticity)

3 Exercice 3: Summary of MTGAUE paper

To be done