SAPIENZA
UNIVERSITÀ DI ROMA

ADVANCED STATISTICS FOR FINANCE REPORT

Master's Degree course:
Financial Risk & Data Analysis

Fabio Tripodi
Matricola 1970235

A.A. 2023-2024

INTRODUCTION

For this project, I used the UN dataset available in R's carData library, which contains 213 observations on different countries and seven key variables: region (geographical area), group (OECD classification), fertility (fertility rate), ppgdp (GDP per capita), lifeExpF (life expectancy of women), pctUrban (percentage of urban population), and infantMortality (infant mortality rate).

Throughout the project, I applied several R codes to analyze the data. Each step is documented in detail, including the commands used in R, the results obtained, and a discussion of the methods implemented.

Function of Codes

```r
# Load the necessary libraries
install.packages("carData")
install.packages("dplyr")
install.packages("ggplot2")
install.packages("caret")
install.packages("splines")
```

These commands are commands in R to install additional packages needed to perform data analysis and statistical modeling. We only need to run this command once to download and install the package on R. If we have already installed the package previously, we can skip this step.

```r
library(carData)
library(dplyr)
library(ggplot2)
library(caret)
library(splines)
```

This code block loads several libraries that will be used in the data analysis process.

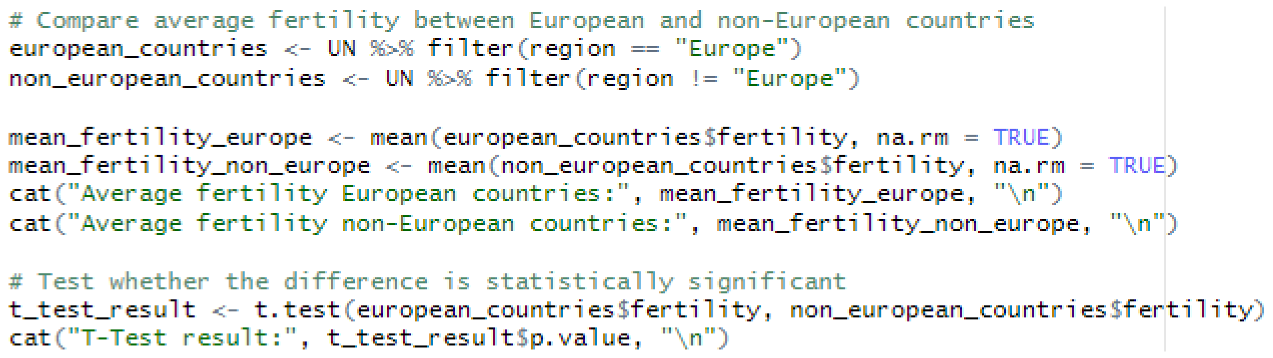
```r
# Load the UN dataset
data("UN")

# Remove rows with missing data and calculate the percentage of missing data
missing_data <- sum(is.na(UN))
total_data <- prod(dim(UN))
percent_missing <- (missing_data / total_data) * 100
cat("Percentage of missing data:", percent_missing, "%\n")

# Remove rows with missing data
UN <- na.omit(UN)
```

At first I loaded the UN dataset. This dataset is included in the "carData" library and contains socio-economic data from various countries around the world.

With the following code block I calculated the percentage of missing data in the "UN" dataset and I got a percentage of missing data of 6.036217%, which we can say is a low percentage.

Then i removed the missing data (NA) from the "UN" dataset.

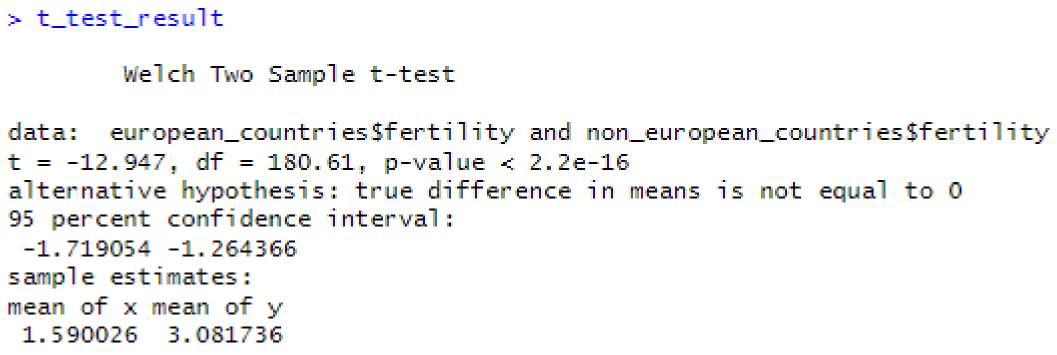
COMPARE AVERAGE FERTILITY BETWEEN EUROPEAN AND NON-EUROPEAN COUNTRIES



```r
# Compare average fertility between European and non-European countries
european_countries <- UN %>% filter(region == "Europe")
non_european_countries <- UN %>% filter(region != "Europe")

mean_fertility_europe <- mean(european_countries$fertility, na.rm = TRUE)
mean_fertility_non_europe <- mean(non_european_countries$fertility, na.rm = TRUE)
cat("Average fertility European countries:", mean_fertility_europe, "\n")
cat("Average fertility non-European countries:", mean_fertility_non_europe, "\n")

# Test whether the difference is statistically significant
t_test_result <- t.test(european_countries$fertility, non_european_countries$fertility)
cat("T-Test result:", t_test_result$p.value, "\n")
```

With the first two codes, I filtered the data for European and non-European countries; at this point I calculated the average fertility for the two subsets.

Finally, I ran the t-test and obtained several results:

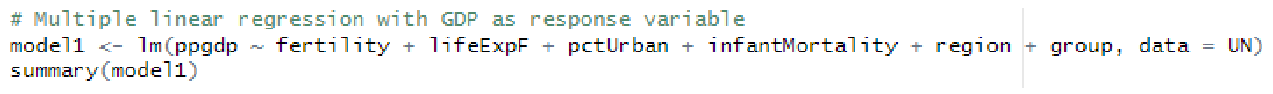


```
> t_test_result

	Welch Two Sample t-test

data:  european_countries$fertility and non_european_countries$fertility
t = -12.947, df = 180.61, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.719054 -1.264366
sample estimates:
mean of x mean of y 
 1.590026  3.081736 
```

The estimates of the means of the two groups which are 1.590026 for European countries (mean of x) and 3.081736 for non-European countries (mean of y).

The p-value is very small, < 2.2e-16 (which is virtually zero). This indicates that there is strong evidence against the null hypothesis, that is, that the averages of the two groups are equal. In other words, there is a significant difference in fertility between European and non-European countries.

MULTIPLE LINEAR REGRESSION WITH GDP AS RESPONSE VARIABLE



```
# Multiple linear regression with GDP as response variable
model1 <- lm(ppgdp ~ fertility + lifeExpF + pctUrban + infantMortality + region + group, data = UN)
summary(model1)
```

Here I created a multiple linear regression with GDP as the response variable with the following characteristics:

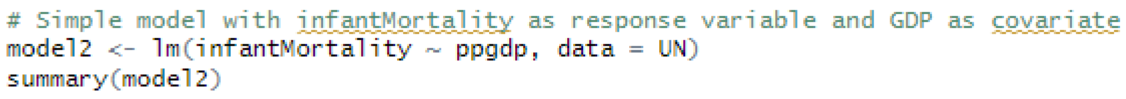
The intercept (-50278.73) which is the predicted GDP per capita when all other variables are zero (which is not very significant in this context).

Some variables have coefficients with a p-value greater than 0.05 (such as fertility and infant mortality). This suggests that they may not have a statistically significant relationship with GDP per capita in this model. The other variables have significant effects (p-value < 0.05):

R-square: This value (0.6135) indicates that the model explains about 61% of the variation in GDP per capita based on the chosen variables.

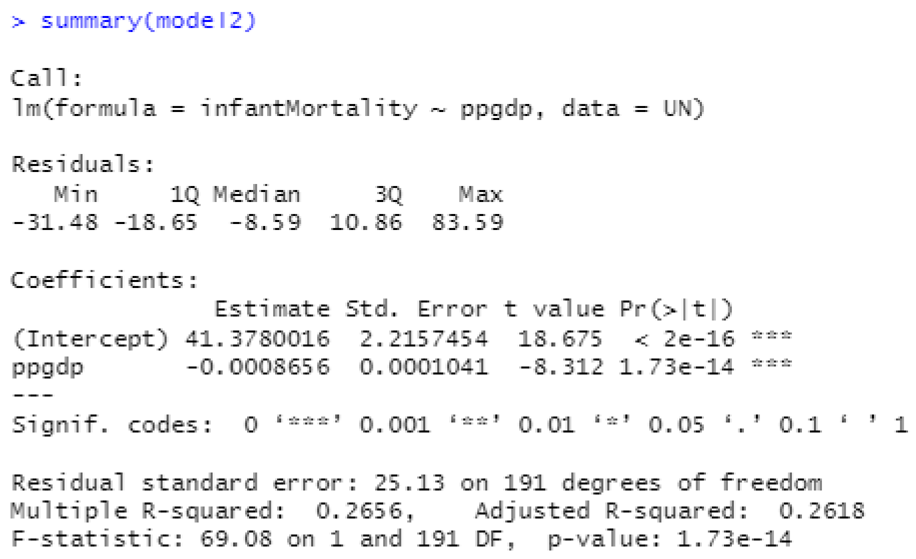
Adjusted R-square: This value (0.59) is a more accurate measure for considering the number of predictor variables.

SIMPLE MODEL WITH INFANT-MORTALITY AS RESPONSE VARIABLE AND GDP AS COVARIATE



```
# Simple model with infantMortality as response variable and GDP as covariate
model2 <- lm(infantMortality ~ ppgdp, data = UN)
summary(model2)
```

Here I ran a simple linear regression with infant mortality as the dependent variable and Gross Domestic Product per capita (ppgdp) as the independent variable.



```
> summary(model2)

Call:
lm(formula = infantMortality ~ ppgdp, data = UN)

Residuals:
   Min     1Q Median     3Q    Max 
-31.48 -18.65  -8.59  10.86  83.59 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept) 41.3780016  2.2157454  18.675  < 2e-16 ***
ppgdp       -0.0008656  0.0001041  -8.312 1.73e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 25.13 on 191 degrees of freedom
Multiple R-squared:  0.2656,	Adjusted R-squared:  0.2618 
F-statistic: 69.08 on 1 and 191 DF,  p-value: 1.73e-14
```

The intercept (41.378) represents the expected infant mortality rate when GDP per capita is zero (a hypothetical and unrealistic value).

The coefficient of ppgdp is negative (-0.0008656). This indicates that there is a negative relationship between GDP per capita and infant mortality rate.

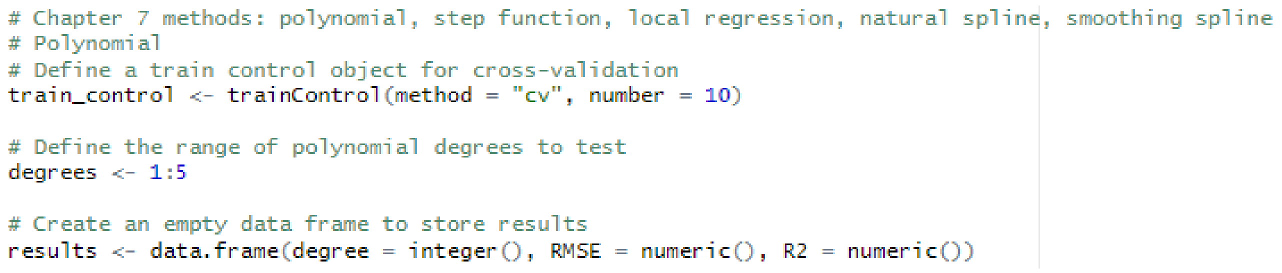
Both coefficients have a very low p-value (less than 2.2e-16) indicating a statistically significant relationship.

R-square: This value (0.2656) indicates that the model explains about 26.56 percent of the variation in the infant mortality rate based on GDP per capita. The adjusted R-square (0.2618) takes into account the number of predictor variables and is slightly lower.

p-value: 1.73e-14 indicate that the model statistically explains a significant portion of the variance in the infant mortality rate (considering GDP per capita as a predictor).

CHAPTER 7 METHODS:

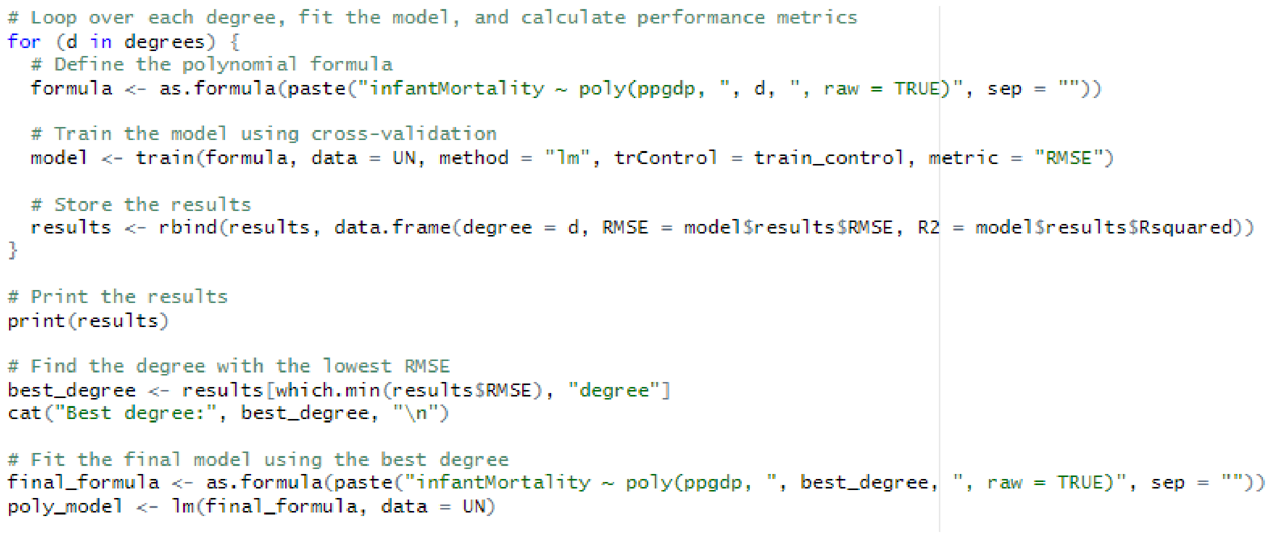
POLYNOMIAL REGRESSION MODEL



```r
# Chapter 7 methods: polynomial, step function, local regression, natural spline, smoothing spline
# Polynomial
# Define a train control object for cross-validation
train_control <- trainControl(method = "cv", number = 10)

# Define the range of polynomial degrees to test
degrees <- 1:5

# Create an empty data frame to store results
results <- data.frame(degree = integer(), RMSE = numeric(), R2 = numeric())
```

In this step, I created a cross-validation control object using the trainControl function. This object was used to specify how to perform cross-validation during the model training process: I specified that 10 folds are to be used for cross-validation. This means that the data will be divided into 10 parts and the model will be trained on 9 of these parts and tested on the remaining part. This process will be repeated 10 times, using each part once as a test set.

Then, I have defined a range of degrees of polynomials to be tested and finally I have created an empty data frame which will be used to store the cross-validation results for each degree of polynomial I tested.



```r
# Loop over each degree, fit the model, and calculate performance metrics
for (d in degrees) {
  # Define the polynomial formula
  formula <- as.formula(paste("infantMortality ~ poly(ppgdp, ", d, ", raw = TRUE)", sep = ""))

  # Train the model using cross-validation
  model <- train(formula, data = UN, method = "lm", trControl = train_control, metric = "RMSE")

  # Store the results
  results <- rbind(results, data.frame(degree = d, RMSE = model$results$RMSE, R2 = model$results$Rsquared))
}

# Print the results
print(results)

# Find the degree with the lowest RMSE
best_degree <- results[which.min(results$RMSE), "degree"]
cat("Best degree:", best_degree, "\n")

# Fit the final model using the best degree
final_formula <- as.formula(paste("infantMortality ~ poly(ppgdp, ", best_degree, ", raw = TRUE)", sep = ""))
poly_model <- lm(final_formula, data = UN)
```

This code runs a cycle on several polynomial degrees, trains a regression model for each, and collects performance metrics (RMSE and R^2) to compare the models. This approach is useful for determining which polynomial degree offers the best predictive performance.

```
> print(results)
  degree     RMSE        R2
1      1 24.67230 0.3265863
2      2 23.64087 0.4071528
3      3 22.92272 0.4772489
4      4 25.37787 0.5658916
5      5 35.63811 0.4868672
```

I then extract the best degree (3), which is, in my case, the one that minimizes the value of RMSE.

The last two codes allowed me to use the best polynomial degree found to fit the final regression model.

```
> summary(poly_model)

Call:
lm(formula = final_formula, data = UN)

Residuals:
    Min      1Q  Median      3Q     Max 
-33.839 -15.659  -1.720   8.776  87.565 

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)    
(Intercept)               5.507e+01  2.483e+00  22.176  < 2e-16 ***
poly(ppgdp, 3, raw = TRUE)1 -4.452e-03  4.597e-04  -9.685  < 2e-16 ***
poly(ppgdp, 3, raw = TRUE)2  1.010e-07  1.448e-08   6.978 4.91e-11 ***
poly(ppgdp, 3, raw = TRUE)3 -6.218e-13  1.082e-13  -5.748 3.57e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 21.41 on 189 degrees of freedom
Multiple R-squared:  0.4726,	Adjusted R-squared:  0.4642 
F-statistic: 56.44 on 3 and 189 DF,  p-value: < 2.2e-16
```

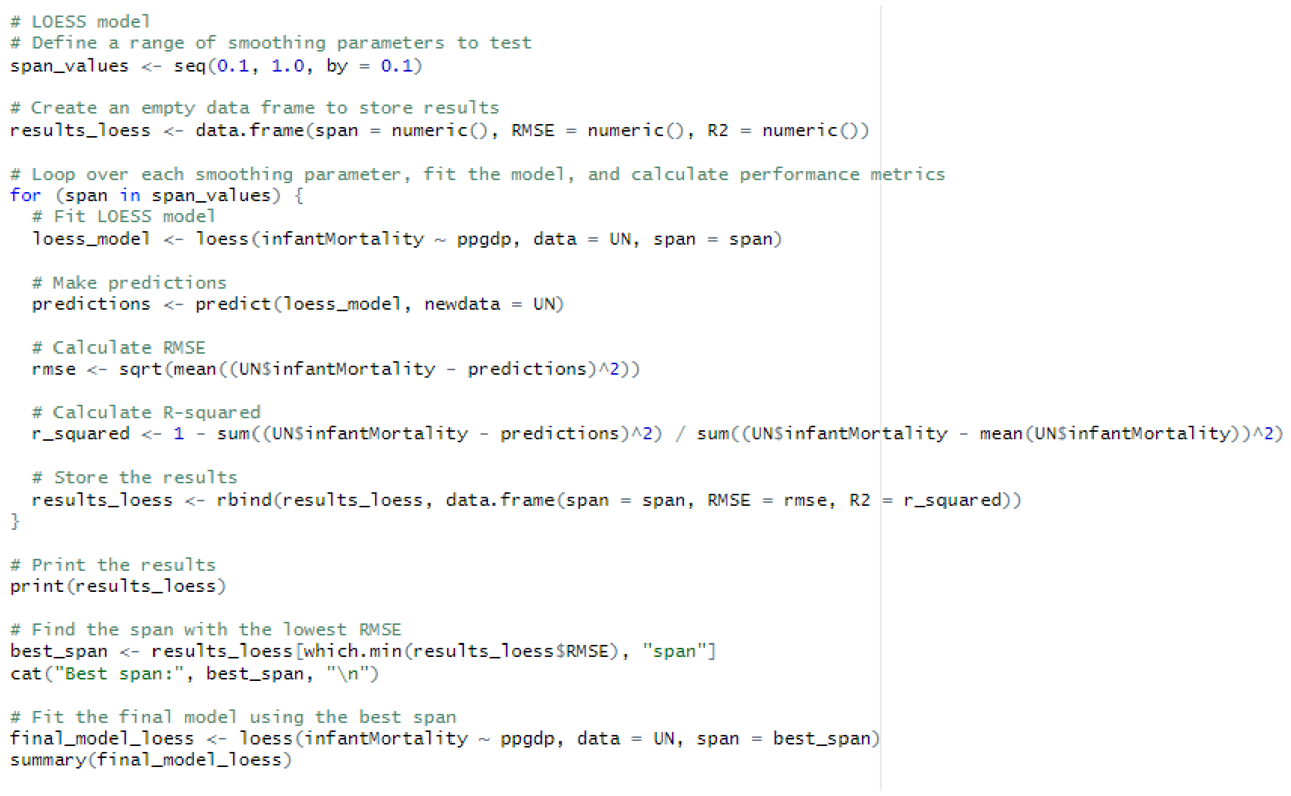
Based on the values of the coefficients, we can say that a unit increase in the independent variables produces a very small change in the dependent variable and that when all the independent variables are equal to 0 the intercept is very low.

Multiple R-squared (R^2): 0.4726. This value indicates that the model explains about 47.26% of the variability in the data.

Adjusted R-squared: 0.4642. This value is adjusted for the number of predictors in the model and provides a more accurate measure of the goodness of fit of the model.

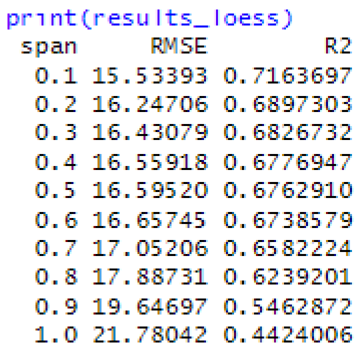
The fitted polynomial regression model is highly significant, as can be inferred from the very low p value. All terms in the polynomial (up to degree 3) are significant and contribute to the prediction of infant mortality based on gross domestic product per capita (ppgdp).

LOCAL REGRESSION MODEL



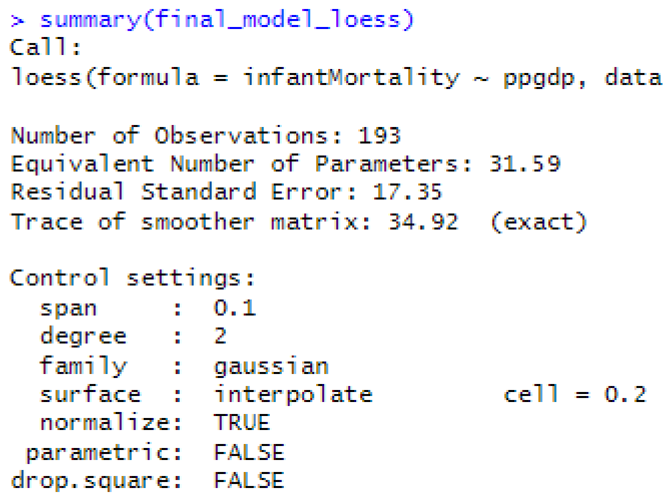
```
# LOESS model
# Define a range of smoothing parameters to test
span_values <- seq(0.1, 1.0, by = 0.1)

# Create an empty data frame to store results
results_loess <- data.frame(span = numeric(), RMSE = numeric(), R2 = numeric())

# Loop over each smoothing parameter, fit the model, and calculate performance metrics
for (span in span_values) {
  # Fit LOESS model
  loess_model <- loess(infantMortality ~ ppgdp, data = UN, span = span)

  # Make predictions
  predictions <- predict(loess_model, newdata = UN)

  # Calculate RMSE
  rmse <- sqrt(mean((UN$infantMortality - predictions)^2))

  # Calculate R-squared
  r_squared <- 1 - sum((UN$infantMortality - predictions)^2) / sum((UN$infantMortality - mean(UN$infantMortality))^2)

  # Store the results
  results_loess <- rbind(results_loess, data.frame(span = span, RMSE = rmse, R2 = r_squared))
}

# Print the results
print(results_loess)

# Find the span with the lowest RMSE
best_span <- results_loess[which.min(results_loess$RMSE), "span"]
cat("Best span:", best_span, "\n")

# Fit the final model using the best span
final_model_loess <- loess(infantMortality ~ ppgdp, data = UN, span = best_span)
summary(final_model_loess)
```

At first I defined a set of smoothing parameters to be tested, then created an empty data frame to store the results, and finally calculated and added the current model results (span values, RMSE, and R²) to the results_loess data frame.



```
print(results_loess)
 span     RMSE        R2
  0.1 15.53393 0.7163697
  0.2 16.24706 0.6897303
  0.3 16.43079 0.6826732
  0.4 16.55918 0.6776947
  0.5 16.59520 0.6762910
  0.6 16.65745 0.6738579
  0.7 17.05206 0.6582224
  0.8 17.88731 0.6239201
  0.9 19.64697 0.5462872
  1.0 21.78042 0.4424006
```

Then I obtained the results from which I deduced the best span, that is, the one that minimizes the RMSE value, which in my case is 0.1. in fact with this span value we have an RMSE equal to 15.53393 and an R-squared equal to 0.7163697



```
> summary(final_model_loess)
Call:
loess(formula = infantMortality ~ ppgdp, data = UN, span = best_span)

Number of Observations: 193
Equivalent Number of Parameters: 31.59
Residual Standard Error: 17.35
Trace of smoother matrix: 34.92  (exact)

Control settings:
  span     :  0.1
  degree   :  2
  family   :  gaussian
  surface  :  interpolate          cell = 0.2
  normalize:  TRUE
 parametric:  FALSE
drop.square:  FALSE
```

Equivalent Number of Parameters: This value represents the equivalent number of parameters in the model, taking into account the smoothing parameter used. The higher this value, the more complex the model. In this case, it is 31.59.

Trace of Smoother Matrix: This value indicates the “complexity” of the LOESS model. It is an indicator of the amount of smoothing applied to the data. Higher values indicate more smoothing.

Control settings: This section shows the control settings used during the LOESS model fitting process. It includes the smoothing parameter (span); the degree of the polynomial used to fit the model (degree); and the distribution family used (family);

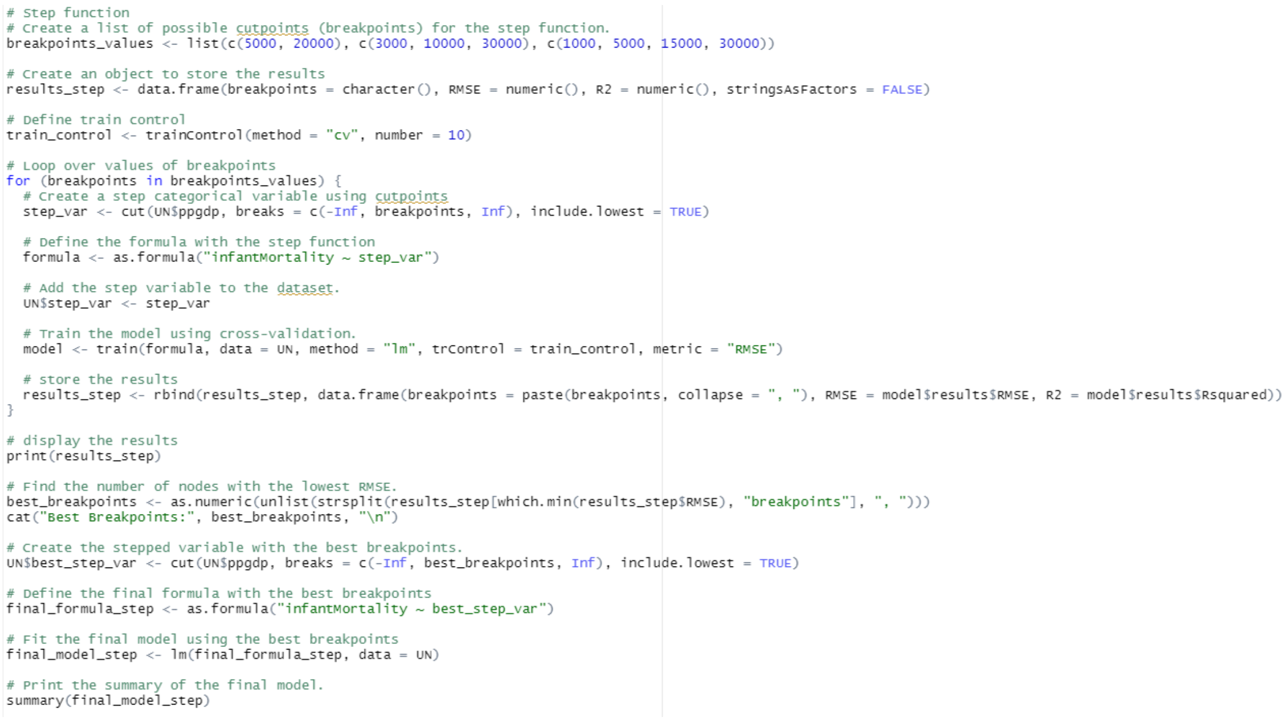
surface: interpolate, cell = 0.2: The regression surface is interpolated between data points with a cell size of 0.2;

normalize: TRUE: The input variables have been normalized;

parametric: FALSE: The LOESS model is completely nonparametric;

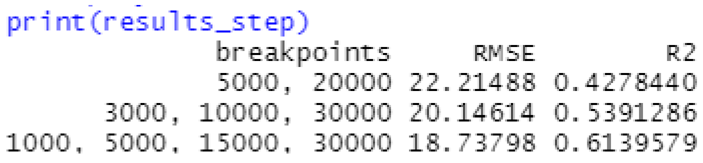
drop.square: FALSE: Quadratic terms are included in local models.

STEP FUNCTION



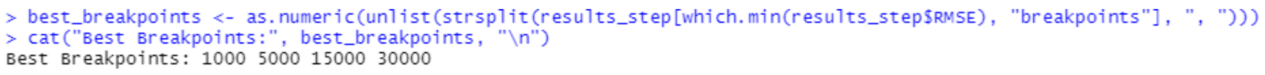
```
# Step function
# Create a list of possible cutpoints (breakpoints) for the step function.
breakpoints_values <- list(c(5000, 20000), c(3000, 10000, 30000), c(1000, 5000, 15000, 30000))

# Create an object to store the results
results_step <- data.frame(breakpoints = character(), RMSE = numeric(), R2 = numeric(), stringsAsFactors = FALSE)

# Define train control
train_control <- trainControl(method = "cv", number = 10)

# Loop over values of breakpoints
for (breakpoints in breakpoints_values) {
  # Create a step categorical variable using cutpoints
  step_var <- cut(UN$ppgdp, breaks = c(-Inf, breakpoints, Inf), include.lowest = TRUE)

  # Define the formula with the step function
  formula <- as.formula("infantMortality ~ step_var")

  # Add the step variable to the dataset.
  UN$step_var <- step_var

  # Train the model using cross-validation.
  model <- train(formula, data = UN, method = "lm", trControl = train_control, metric = "RMSE")

  # Store the results
  results_step <- rbind(results_step, data.frame(breakpoints = paste(breakpoints, collapse = ", "), RMSE = model$results$RMSE, R2 = model$results$Rsquared))
}

# display the results
print(results_step)

# Find the number of nodes with the lowest RMSE.
best_breakpoints <- as.numeric(unlist(strsplit(results_step[which.min(results_step$RMSE), "breakpoints"], ", ")))
cat("Best Breakpoints:", best_breakpoints, "\n")

# Create the stepped variable with the best breakpoints.
UN$best_step_var <- cut(UN$ppgdp, breaks = c(-Inf, best_breakpoints, Inf), include.lowest = TRUE)

# Define the final formula with the best breakpoints
final_formula_step <- as.formula("infantMortality ~ best_step_var")

# Fit the final model using the best breakpoints
final_model_step <- lm(final_formula_step, data = UN)

# Print the summary of the final model.
summary(final_model_step)
```

Here I created a list of possible breakpoints for the step function and an object to store the results; then I defined the train control. At this point, I calculated and inserted the optimized performance metrics into the previously created object.



```
print(results_step)
                breakpoints     RMSE        R2
               5000, 20000 22.21488 0.4278440
        3000, 10000, 30000 20.14614 0.5391286
1000, 5000, 15000, 30000 18.73798 0.6139579
```

Then I printed out the results from which I obtained the best breakpoints, that is, those that, in my case, minimize the RMSE:



```
> best_breakpoints <- as.numeric(unlist(strsplit(results_step[which.min(results_step$RMSE), "breakpoints"], ", ")))
> cat("Best Breakpoints:", best_breakpoints, "\n")
Best Breakpoints: 1000 5000 15000 30000
```

At this point, by dividing the range of the independent variable into segments, the model assumes that the continuous variable takes a constant value within each segment, so I transformed the continuous variable into a categorical one with the best breakpoints found previously.

Then I constructed a linear regression formula using this variable and trained the final model. The final goal is to determine the effect of different ranges of per capita income (ppgdp) on infant mortality (infantMortality) using the optimal breakpoints.

```
> summary(final_model_step)

Call:
lm(formula = final_formula_step, data = UN)

Residuals:
    Min      1Q  Median      3Q     Max 
-45.322 -10.876  -2.113   4.527  80.578 

Coefficients:
                             Estimate Std. Error t value Pr(>|t|)    
(Intercept)                    70.374      3.080   22.85   <2e-16 ***
best_step_var(1e+03,5e+03]    -35.837      3.899   -9.19   <2e-16 ***
best_step_var(5e+03,1.5e+04]  -53.927      4.204  -12.83   <2e-16 ***
best_step_var(1.5e+04,3e+04]  -57.637      5.334  -10.80   <2e-16 ***
best_step_var(3e+04, Inf]     -66.346      4.681  -14.17   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 18.98 on 188 degrees of freedom
Multiple R-squared:  0.5873,	Adjusted R-squared:  0.5786 
F-statistic: 66.89 on 4 and 188 DF,  p-value: < 2.2e-16
```

All coefficients have very low p-values (<2e-16), indicating that they are highly significant and all negatively impact the dependent variable. The standard error of the residuals, which measures the mean deviation of the residuals from the regression line is 18.98. Multiple R-squared: The proportion of variability in the dependent variable explained by the model (0.5873, or 58.73%).

Adjusted R-squared: The adjusted version of R-squared that takes into account the number of predictors in the model (0.5786, or 57.86%).

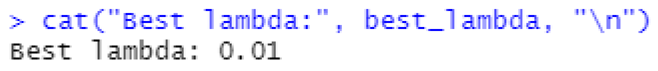
The p-value associated with the F-statistic, indicates that the model is highly significant (<2.2e-16).

SMOOTHING SPLINES

```r
# Smoothing splines
library(mgcv)
# Define a train control object for cross-validation
train_control <- trainControl(method = "cv", number = 10)

# Define the range of smoothing parameters (lambda values) to test
lambda_values <- 10^seq(-2, 2, length.out = 100)

# Create an empty data frame to store results
results_spline <- data.frame(lambda = numeric(), RMSE = numeric(), R2 = numeric())

# Loop over each lambda value, fit the model, and calculate performance metrics
for (lambda in lambda_values) {
  # Fit GAM model
  gam_model <- gam(infantMortality ~ s(ppgdp, bs = "cr", k = 10), data = UN, method = "GCV.Cp", select = TRUE)

  # Make predictions
  predictions <- predict(gam_model, type = "response")

  # Calculate RMSE
  rmse <- sqrt(mean((UN$infantMortality - predictions)^2))

  # Calculate R-squared
  r_squared <- summary(gam_model)$r.sq

  # Store the results
  results_spline <- rbind(results_spline, data.frame(lambda = lambda, RMSE = rmse, R2 = r_squared))
}

# Print the results
print(results_spline)

# Find the lambda with the lowest RMSE
best_lambda <- results_spline[which.min(results_spline$RMSE), "lambda"]
cat("Best lambda:", best_lambda, "\n")

final_model_spline <- gam(infantMortality ~ s(ppgdp, bs = "cr", k = 10), data = UN, method = "GCV.Cp", select = TRUE)
summary(final_model_spline)
```

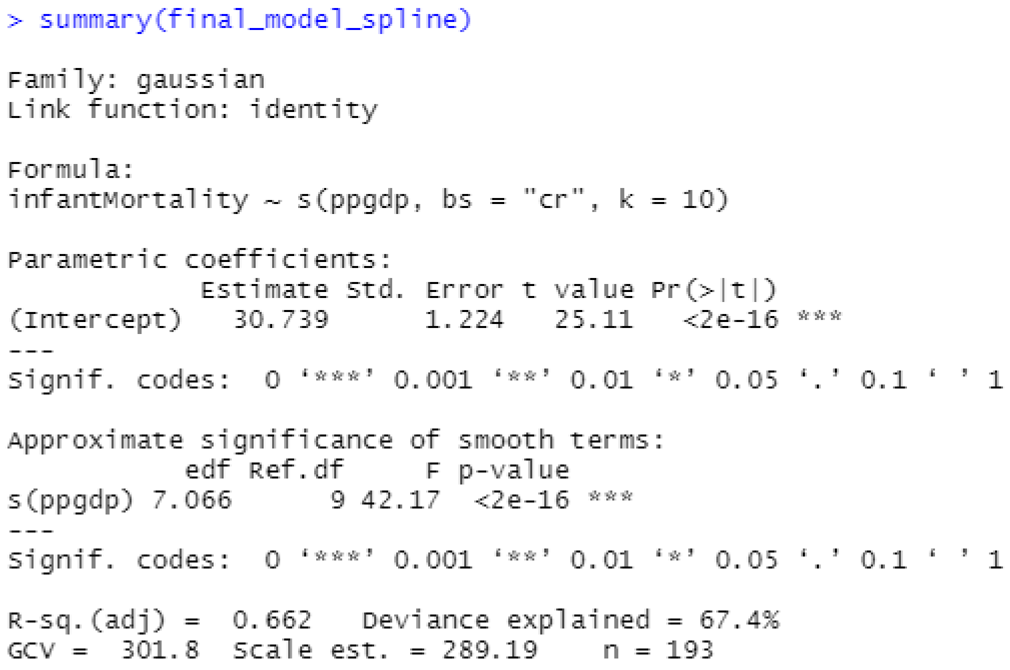
In this step, I created a cross-validation control object using the trainControl function, then I generated a sequence of 100 values logarithmically distributed between 10^{-2} and 10^{2} power and created an empty data frame that will be used to store the cross-validation results for each value of lambda.

I ran a loop on each value of lambda. Inside the loop, I fitted a GAM model using the gam() function with the variable ppgdp as the predictor variable and infantMortality as the response variable. I used the specification s(ppgdp, bs = "cr", k = 10) to specify the use of a regular cubic basis function with 10 internal nodes for the ppgdp variable and used the GCV.Cp method to automatically select the best model during fitting. I then made predictions using the fitted model and calculated the RMSE and R^2 to evaluate the model's performance.



```
> cat("Best lambda:", best_lambda, "\n")
Best lambda: 0.01
```

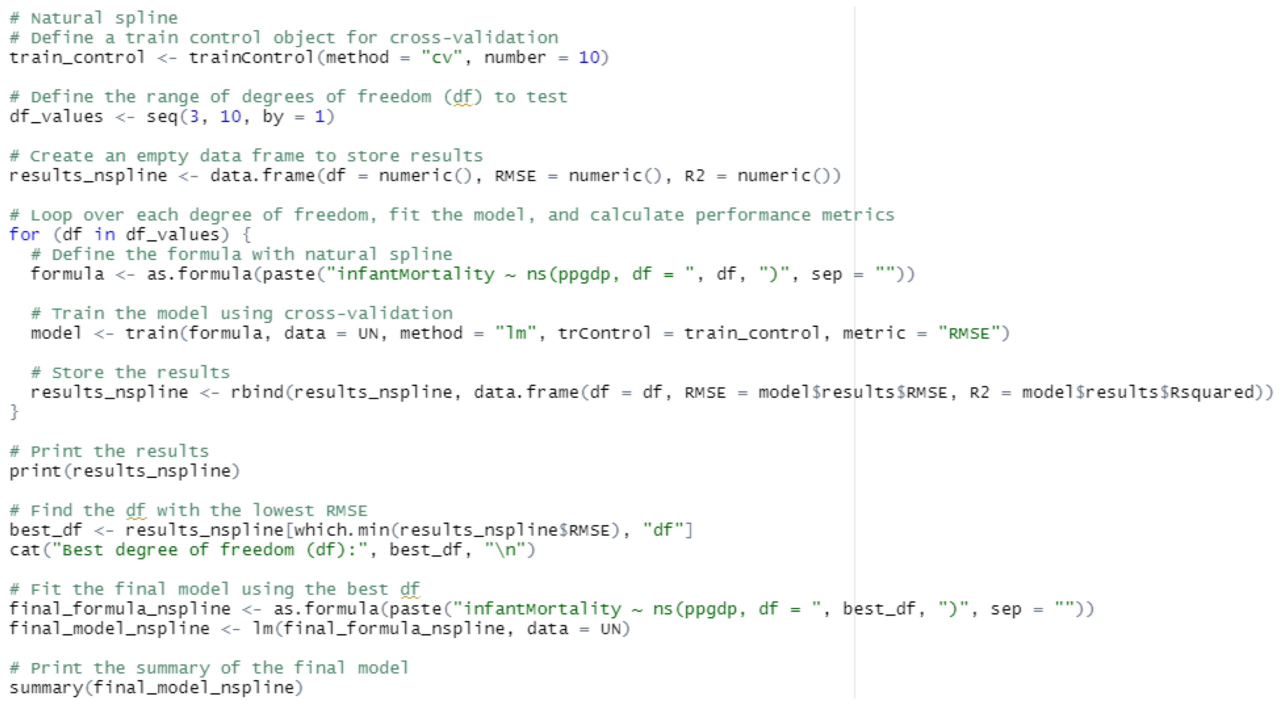
In this case the best smooth parameter is 0.1, which is also the smallest among those generated, indicating that the model may be analyzing the data too closely, which could lead to overfitting.



```
> summary(final_model_spline)

Family: gaussian
Link function: identity

Formula:
infantMortality ~ s(ppgdp, bs = "cr", k = 10)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   30.739      1.224   25.11   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:
           edf Ref.df     F p-value    
s(ppgdp) 7.066      9 42.17  <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) =  0.662   Deviance explained = 67.4%
GCV =  301.8  Scale est. = 289.19    n = 193
```

The intercept has an estimated value of 30.739; in this case, the smoothing term for ppgdp has a very small p-value (<2e-16), which indicates that the smoothing term is highly significant. This value (R^2) represents the coefficient of determination adjusted for the number of predictors in the model, which indicates the proportion of the total variation in the dependent variable explained by the model. In this case, the adjusted R^2 is 0.662, which indicates that the model explains approximately 66.2% of the variation in the dependent variable. GCV (Generalized Cross Validation) is an indication of the goodness of fit of the model and in this case it is 301.8; in general, lower values of GCV indicate a better model, as they indicate a lower variance of the model compared to the data.

NATURAL SPLINES

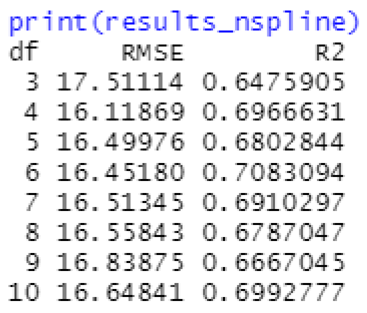


```
# Natural spline
# Define a train control object for cross-validation
train_control <- trainControl(method = "cv", number = 10)

# Define the range of degrees of freedom (df) to test
df_values <- seq(3, 10, by = 1)

# Create an empty data frame to store results
results_nspline <- data.frame(df = numeric(), RMSE = numeric(), R2 = numeric())

# Loop over each degree of freedom, fit the model, and calculate performance metrics
for (df in df_values) {
  # Define the formula with natural spline
  formula <- as.formula(paste("infantMortality ~ ns(ppgdp, df = ", df, ")", sep = ""))

  # Train the model using cross-validation
  model <- train(formula, data = UN, method = "lm", trControl = train_control, metric = "RMSE")

  # Store the results
  results_nspline <- rbind(results_nspline, data.frame(df = df, RMSE = model$results$RMSE, R2 = model$results$Rsquared))
}

# Print the results
print(results_nspline)

# Find the df with the lowest RMSE
best_df <- results_nspline[which.min(results_nspline$RMSE), "df"]
cat("Best degree of freedom (df):", best_df, "\n")

# Fit the final model using the best df
final_formula_nspline <- as.formula(paste("infantMortality ~ ns(ppgdp, df = ", best_df, ")", sep = ""))
final_model_nspline <- lm(final_formula_nspline, data = UN)

# Print the summary of the final model
summary(final_model_nspline)
```

Here, I created a cross-validation control object using the trainControl function, then I have defined a range of values for the degrees of freedom that will be tested and created an empty data frame that will be used to store the cross-validation results for each value of lambda.

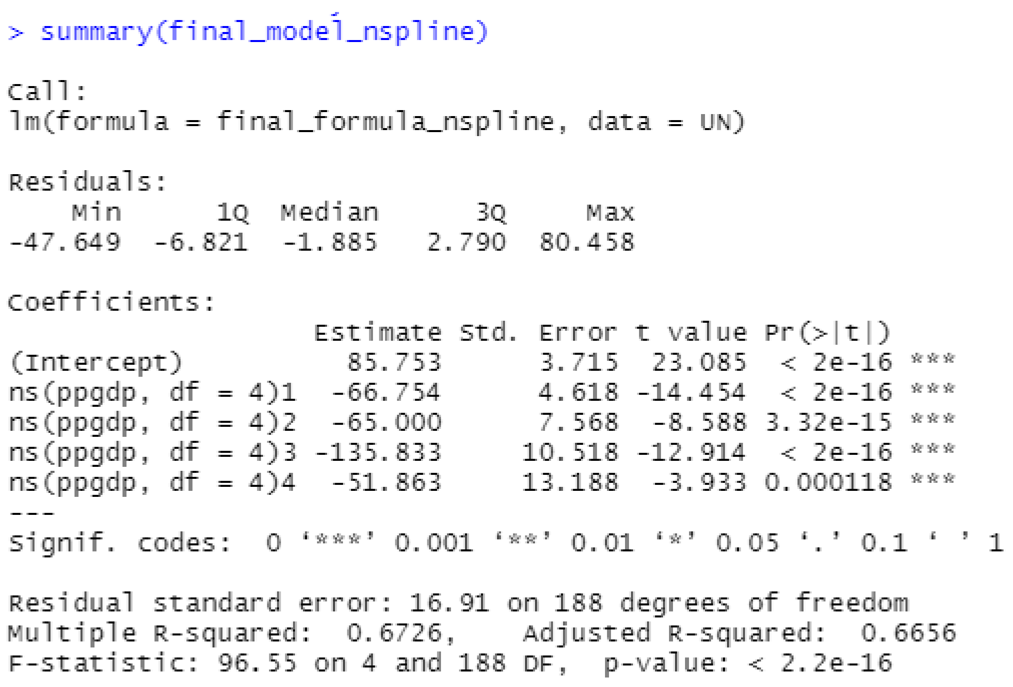
At this point, I created a for loop that iterates over each df value in the df_values vector. This allows me to test the model for different degrees of freedom. I then built the model formula using a natural spline and trained the model. I then used the train function that applies the cross-validation defined in train_control. Finally, I stored the results of the current model in the results_nspline data frame.

Here I have printed the results for the different degrees of freedom:



```
print(results_nspline)
df       RMSE        R2
 3 17.51114 0.6475905
 4 16.11869 0.6966631
 5 16.49976 0.6802844
 6 16.45180 0.7083094
 7 16.51345 0.6910297
 8 16.55843 0.6787047
 9 16.83875 0.6667045
10 16.64841 0.6992777
```

From here I understand that the best number of degrees of freedom (the one that minimizes RMSE) is 4, so I print the final results:



```
> summary(final_model_nspline)

Call:
lm(formula = final_formula_nspline, data = UN)

Residuals:
    Min      1Q  Median      3Q     Max 
-47.649  -6.821  -1.885   2.790  80.458 

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)    
(Intercept)            85.753      3.715  23.085  < 2e-16 ***
ns(ppgdp, df = 4)1    -66.754      4.618 -14.454  < 2e-16 ***
ns(ppgdp, df = 4)2    -65.000      7.568  -8.588 3.32e-15 ***
ns(ppgdp, df = 4)3   -135.833     10.518 -12.914  < 2e-16 ***
ns(ppgdp, df = 4)4    -51.863     13.188  -3.933 0.000118 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16.91 on 188 degrees of freedom
Multiple R-squared:  0.6726,	Adjusted R-squared:  0.6656 
F-statistic: 96.55 on 4 and 188 DF,  p-value: < 2.2e-16
```

The intercept of the model is 85.753, while all coefficient values are negative, implying that they negatively affect the dependent variable. We also infer from the very low p-values for each coefficient, that they are all highly significant.

Multiple R-squared: 0.6726. This indicates that approximately 67.26% of the variability in the dependent variable (infantMortality) is explained by the model.

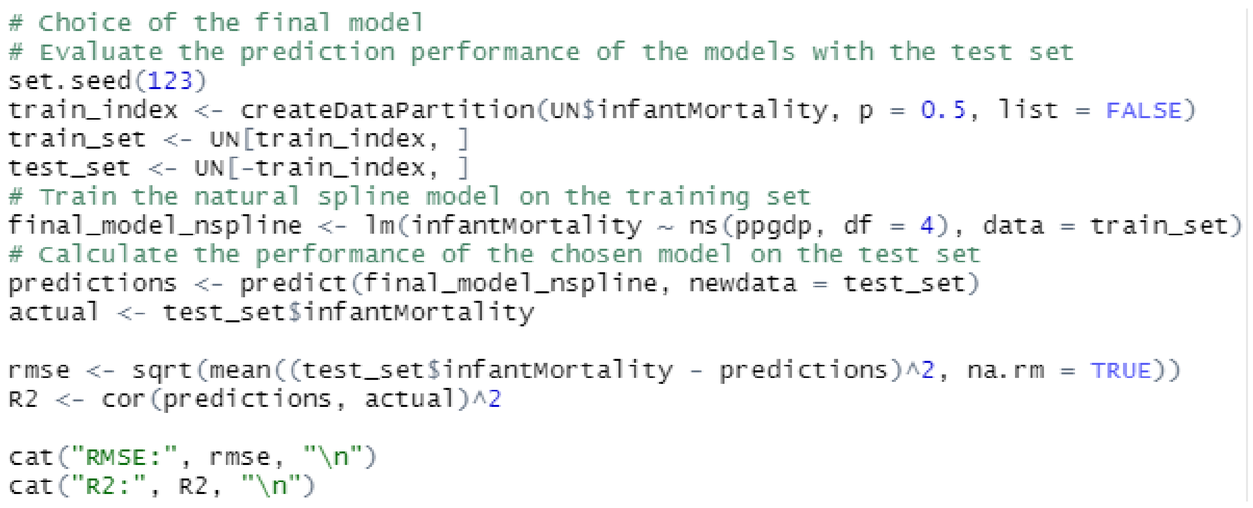
Adjusted R-squared: 0.6656. This value is slightly lower than the R-squared and takes into account the number of predictors in the model, providing a more accurate measure of the goodness of the model.

A very low p-value (< 2.2e-16) indicates that at least one of the model coefficients is significantly different from zero, suggesting that the model is significant.

FINAL CHOICE: NATURAL SPLINES

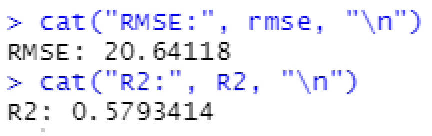
Finally, I split the UN dataset into two parts: 50% for training (train_set) and 50% for testing (test_set). I trained the final model (with 4 degrees of freedom) on the training set, then used the trained model to make predictions on the test set and evaluated the model performance using RMSE and R².

This process allows me to evaluate how well the model generalizes to new and unseen data.



```r
# Choice of the final model
# Evaluate the prediction performance of the models with the test set
set.seed(123)
train_index <- createDataPartition(UN$infantMortality, p = 0.5, list = FALSE)
train_set <- UN[train_index, ]
test_set <- UN[-train_index, ]
# Train the natural spline model on the training set
final_model_nspline <- lm(infantMortality ~ ns(ppgdp, df = 4), data = train_set)
# Calculate the performance of the chosen model on the test set
predictions <- predict(final_model_nspline, newdata = test_set)
actual <- test_set$infantMortality

rmse <- sqrt(mean((test_set$infantMortality - predictions)^2, na.rm = TRUE))
R2 <- cor(predictions, actual)^2

cat("RMSE:", rmse, "\n")
cat("R2:", R2, "\n")
```

Finally I got the following results:



```
> cat("RMSE:", rmse, "\n")
RMSE: 20.64118
> cat("R2:", R2, "\n")
R2: 0.5793414
```

REASONS FOR THE CHOICE:

At the end of obtaining the results of all the models used I decided to choose the natural spline as the final model for several reasons:

- Natural splines offer greater flexibility than simple polynomial models. They can adapt to non-linear relationships in the data without introducing the complexity and risk of overfitting associated with high-degree polynomials. This allows to model the relationship between GDP (ppgdp) and infant mortality more accurately, capturing subtle variations in the data that other models might not detect.
- During the evaluation of the models, the natural spline showed lower RMSE (Root Mean Squared Error) values than the other models tested. Furthermore, the R² value was significantly high.
- Natural splines impose the condition that the function is linear outside the most external nodes, reducing the risk of irregular behaviors at the edges of the dataset, unlike the step function for example that could show irregular behaviors.
- This ensures that the model does not become unstable or unrealistic at the extremes of the data range, improving the robustness of the predictions.
- Natural splines, while flexible, are relatively easy to interpret compared to more complex models such as LOESS.
- The intercept and coefficients of the spline are highly significant, as indicated by the very low p-values.

- The Adjusted R-squared of 0.6656 suggests that the model explains a good deal of the variability in the data.
- The F-statistic and its p-value indicate that the model overall is highly significant.

CHARACTERISTICS OF THE METHODS USED:

POLYNOMIAL REGRESSION MODEL

The polynomial regression model is an extension of the linear regression model that allows you to capture nonlinear relationships between the independent variable and the dependent variable. Instead of assuming that the relationship between the variables is a straight line (as in simple linear regression), the polynomial model allows you to use polynomials (e.g., squares, cubes, etc.) to model more complex relationships. Increasing the degree of the polynomial increases the flexibility of the model. However, too much flexibility can lead to overfitting, where the model fits the training data too well and does not generalize well to new data. Cross-validation can be useful for this model to estimate the best degree for the polynomial.

LOCAL REGRESSION MODEL

The local regression (LOESS) model is a method of fitting a curve or surface to the observed data, considering only a subset of the data around each point. Rather than trying to fit a single function to all the data, the LOESS model fits simpler local models (e.g., straight lines or polynomials) to small windows of data near each point in the dataset, allowing it to capture complex, nonlinear relationships (so it proves more flexible).

In the context of the LOESS model, cross-validation is used to select the optimal smoothing parameter.

Main Steps: divide the dataset into several subsets (fold); fit the LOESS model on one subset of the data (train set) and evaluate the performance on another subset (validation set); repeat the process for all subsets, calculating the performance of the model on each validation set; finally select the value of the smoothing parameter that gives the best average performance on all validation sets.

When the smoothing parameter is smaller, the window is narrow and considers only very close points. This can make the model more sensitive to local details, but it can also lead to overfitting (overfitting).

When the smoothing parameter is larger, the window is wider and considers more points. This tends to produce a smoother model that is less sensitive to local details, but may risk not capturing local features well (underfitting).

STEP FUNCTION

A step function is a function that changes value abruptly at certain points in the domain. Between these change points, the value of the function remains constant. It is used to model situations where changes occur discretely and non-continuously.

Using cross-validation for a step function helps evaluate the performance of the function and determine the best discontinuity points (breakpoints) to fit the data.

SMOOTHING SPLINES

A smoothing spline is a type of model that is used to fit a smooth curve to a set of data. The main goal is to find a balance between the curve's flexibility (fitting the data well) and its smoothness (avoiding following noise or random fluctuations in the data too closely). The smoothing spline tries to draw a curve that passes close to these data points. However, it does not try to pass through each point exactly as an interpolation curve would, but it tries to balance between passing close to the points and keeping the curve smooth since if the curve follows each point too closely, overfitting results, i.e., the curve also follows random noises and fluctuations in the data, resulting in an irregular curve; conversely, if the curve is too smooth, it may not fit the actual data well and miss important variations in the data.

The smoothing spline model uses a smoothing parameter to find the right balance. This parameter decides how smooth the curve should be. If this parameter is high, the curve will be very smooth. If it is low, the curve will follow the data more closely. This model is very useful for capturing linear relationships between variables.

Cross-validation for this model helps to identify the right smoothing parameter for the spline smoothing model. A smoothing parameter that is too low, as mentioned earlier, can lead to overfitting (the curve follows the training data too closely), while a parameter that is too high can lead to underfitting (the curve is too smooth and does not capture variations in the data).

It allows the best smoothing parameter to be selected optimally. During cross-validation, one can test different values of the smoothing parameter and choose the one that minimizes the RMSE. You repeat the whole process for different smoothing parameter values and finally select the optimal smoothing parameter value that minimizes the RMSE.

NATURAL SPLINES

A natural spline is a type of spline that is used to model nonlinear relationships between variables. Splines are piece-by-piece functions defined by polynomials. A spline allows you to draw a smooth curve through a set of data points, ensuring that the curve passes close to the data points smoothly.

The curve is divided into segments, each of which is defined by a polynomial. These segments are called "pieces" of the spline.

Splines ensure that the pieces join smoothly, without corners or discontinuities, up to a certain order of derivative.

A natural spline has additional constraints at the endpoints (i.e., the lowest and highest values of the independent variable). These constraints cause the spline to be linear outside the data range, which reduces oscillation and improves the stability of the model at the endpoints by preventing unwanted oscillations at the edges of the data. Therefore, at the edges of the data range, the natural spline behaves like a straight line.

Knots are specific points along the data range where the polynomial segments join. These knots are chosen based on the available data. Between each pair of knots, a polynomial is defined. These polynomials are chosen so that the overall curve is smooth and continuous.

When applying cross-validation to a natural spline model, the choice of knots is important for the behavior of the model.

Similar to smoothing spline models, the smoothing parameter in natural spline models must be chosen carefully. Cross-validation can help choose the best smoothing parameter by testing different values and selecting the one that provides the best performance.

Cross-validation is also extremely useful for choosing the optimal number of degrees of freedom (df) for a natural spline model. The degrees of freedom determine the flexibility of the spline: more degrees of freedom means a more flexible curve, while fewer degrees of freedom means a stiffer curve. Too many degrees of freedom can cause overfitting, where the model fits the training data too well and does not generalize well to new data; conversely, too few degrees of freedom can cause underfitting, where the model does not sufficiently capture the complexity of the relationship in the data.

Cross-validation allows us to evaluate different numbers of degrees of freedom and choose the one that provides the best overall performance.