Lab3

Tristen Tooming

2/3/2021

###Assignment 1: Analysis of mortality rates using splines

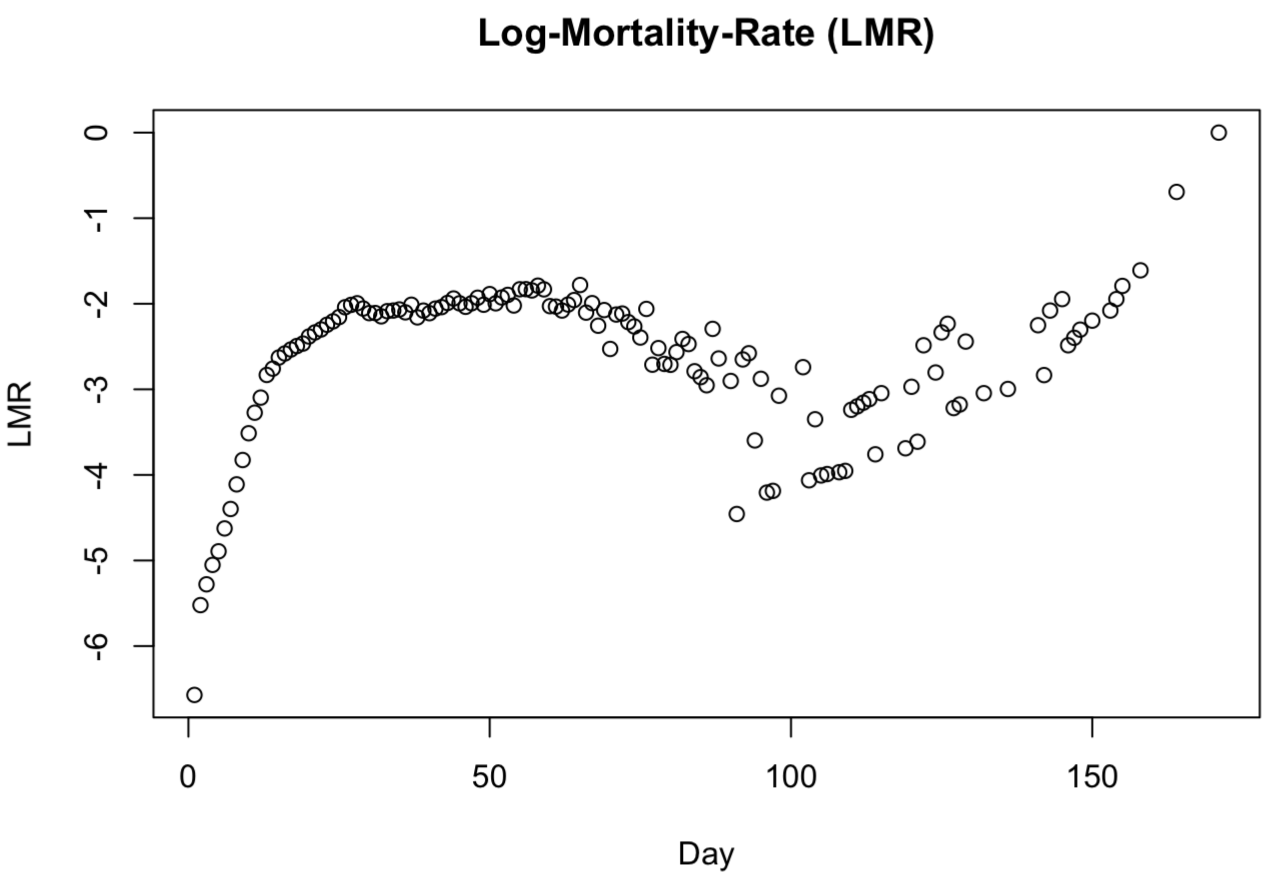
Pre

```
mortality_rate = read.csv('mortality_rate.csv', header = TRUE, sep = ';')
summary(mortality_rate$Rate)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00140 0.04940 0.09455 0.09836 0.12848 1.00000
```

1.a.

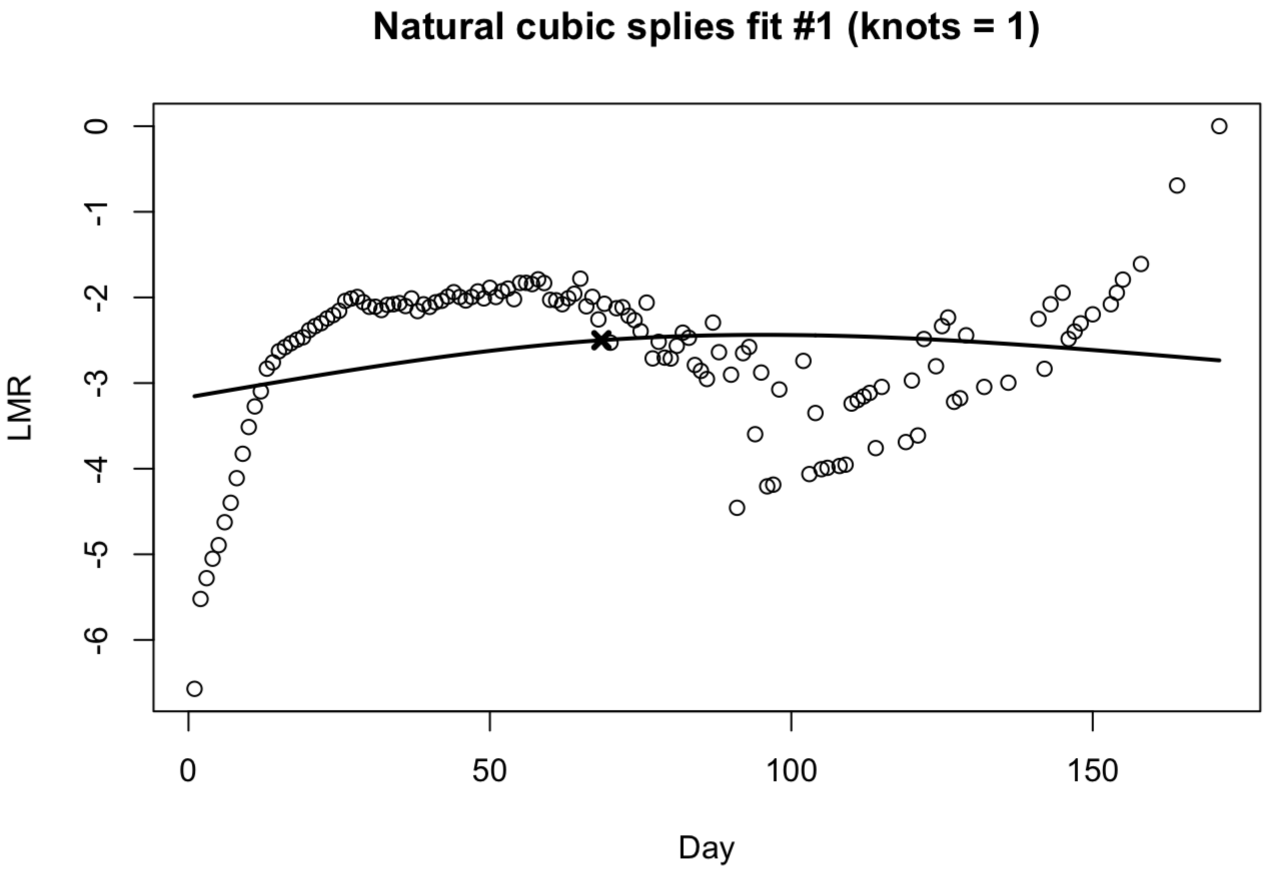
```
mortality_rate$lmr = log(mortality_rate$Rate)
plot(mortality_rate$Day, mortality_rate$lmr,
     ylab = 'LMR',
     xlab = 'Day',
     main = 'Log-Mortality-Rate (LMR)')
```

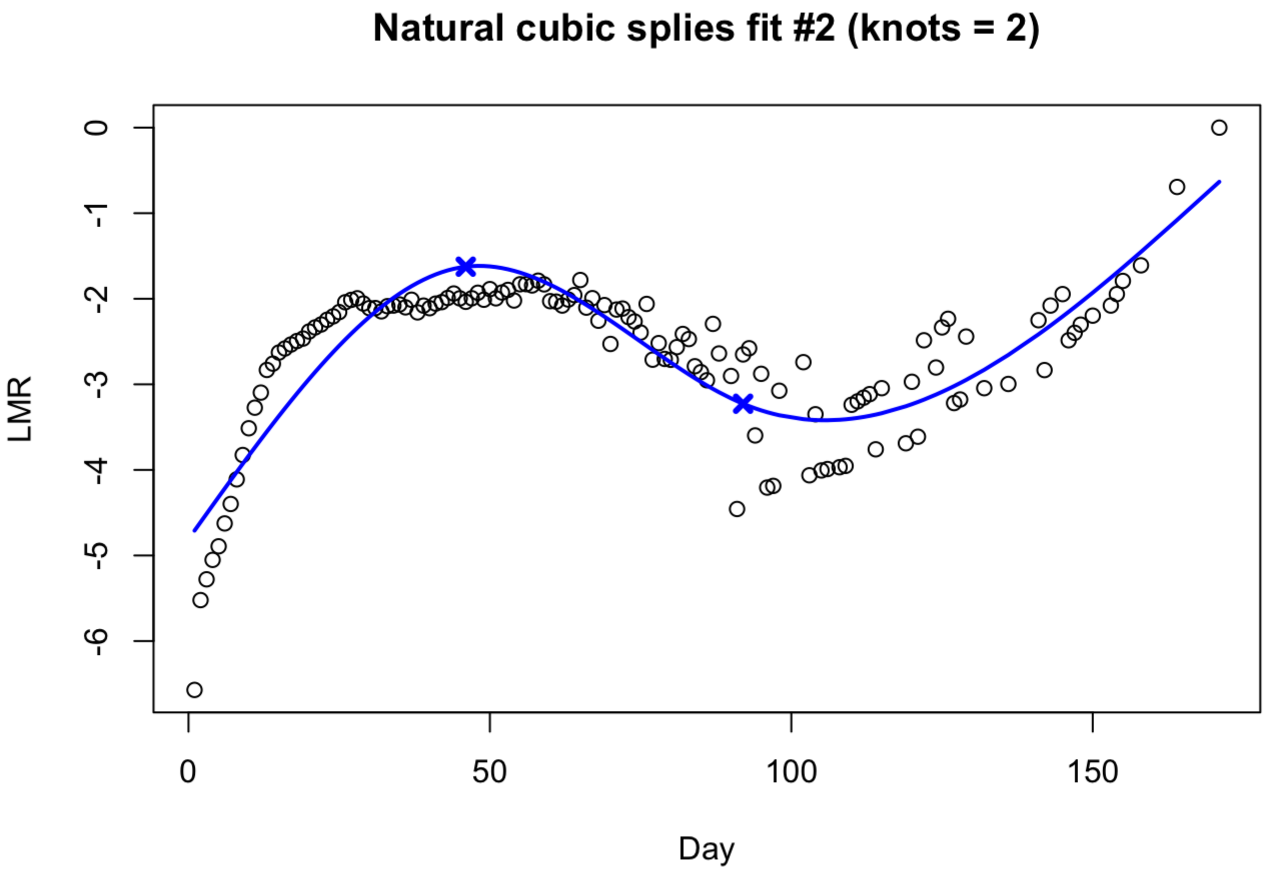


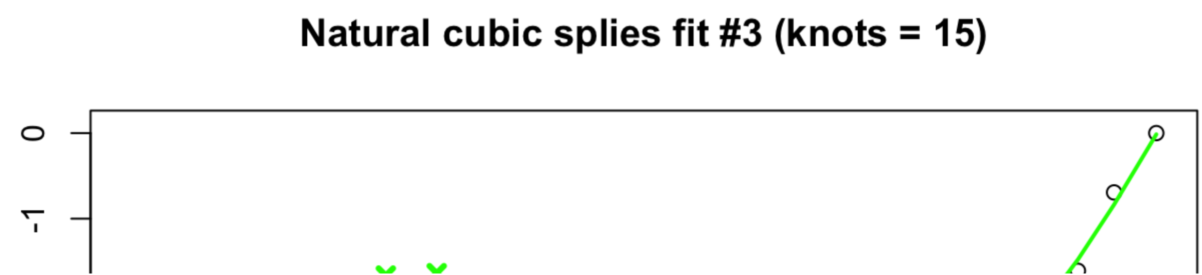
> Plot looks a little bit scrappy and you cannot see a solid 'line' to indicate that data follows an exponential mortality rate.

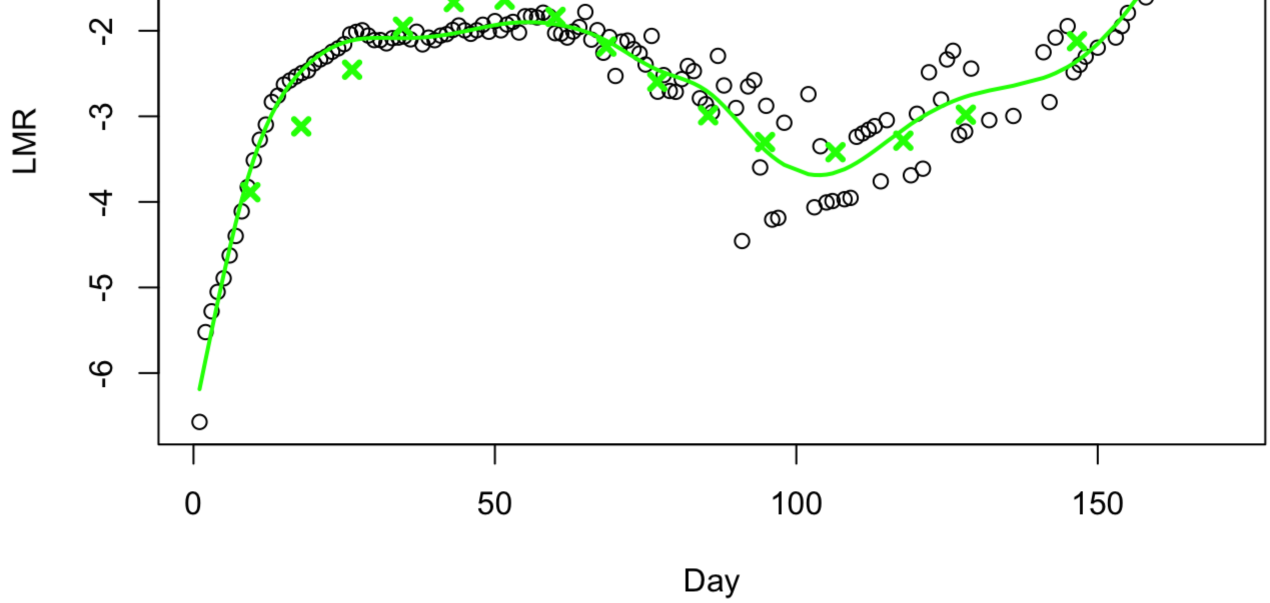
1.b

```r
library(splines)

fit_ns_1 = lm(lmr ~ ns(Day, df = 2), data = mortality_rate)
pred1 = predict(fit_ns_1, newdata = list(Day=mortality_rate$Day))

fit_ns_2 = lm(lmr ~ ns(Day, df = 3), data = mortality_rate)
pred2 = predict(fit_ns_2, newdata = list(Day=mortality_rate$Day))

fit_ns_3 = lm(lmr ~ ns(Day, df = 16), data = mortality_rate)
pred3 = predict(fit_ns_3, newdata = list(Day=mortality_rate$Day))

fit_ns_4 = lm(lmr ~ ns(Day, df = 51), data = mortality_rate)
pred4 = predict(fit_ns_4, newdata = list(Day=mortality_rate$Day))
```

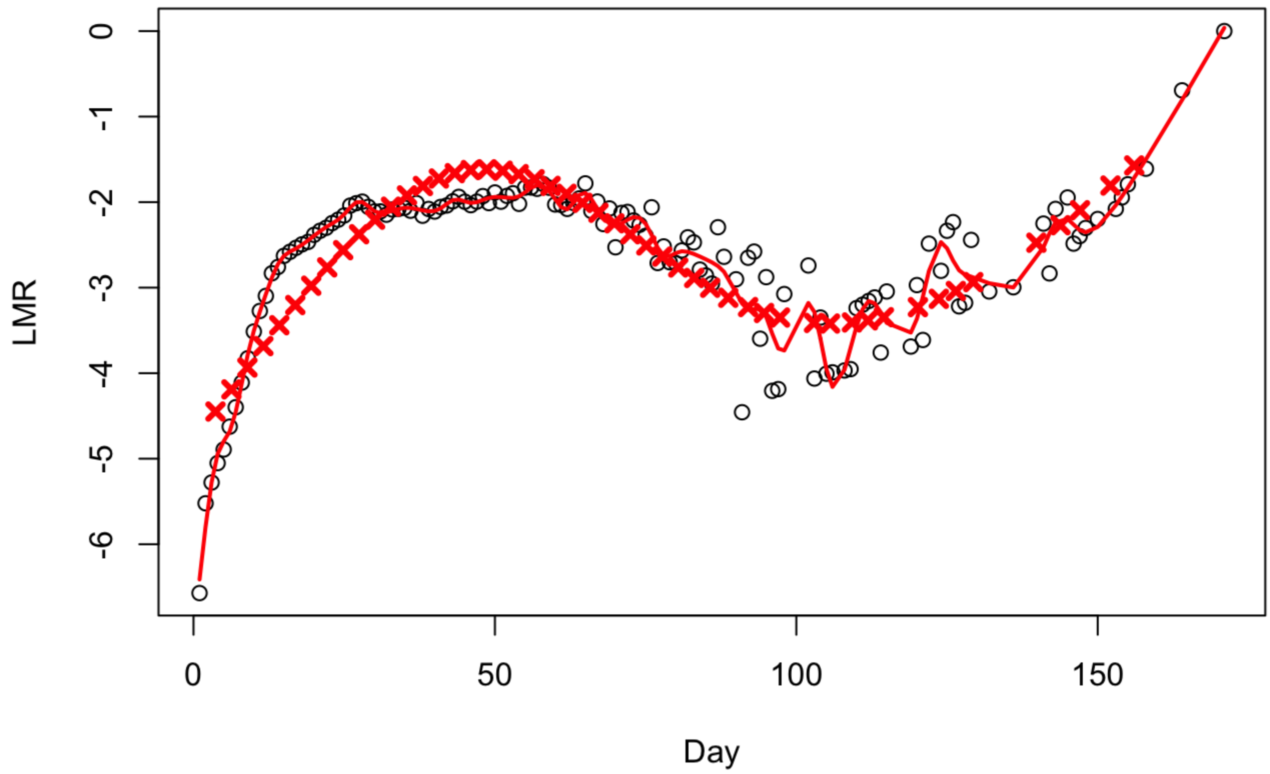

Natural cubic splies fit #1 (knots = 1)
LMR
0
-1
-2
-3
-4
-5
-6
0
50
100
150
Day


Natural cubic splies fit #2 (knots = 2)
LMR
0
-1
-2
-3
-4
-5
-6
0
50
100
150
Day


Natural cubic splies fit #3 (knots = 15)
0
-1



Natural cubic splies fit #4 (knots = 50)




```
## Model #1 MSE: 0.80
## Model #2 MSE: 0.20
## Model #3 MSE: 0.08
## Model #4 MSE: 0.06
```

> Based on MSE model number four performs best. However with that many knots it can lead to overfit of the model. Model number three or even number two could be better choices. Needs more testing/validation.

1.c

```r
# Splitting data function
make_sample = function(data, seed_num, train_size) {
  set.seed(seed_num)
  smp_size <- floor(train_size * nrow(mtcars))
  train_ind <- sample(seq_len(nrow(mtcars)), size = smp_size)

  return_list = vector(mode="list", length=2)
  names(return_list) = c('train', 'test')

  train <- mortality_rate[train_ind, ]
  test <- mortality_rate[-train_ind, ]

  return_list[[1]] = train
  return_list[[2]] = test

  return(return_list)
}

# Calc MSE
calc_mse = function(actual, pred) {
  return(mean(actual - pred)**2)
}



test = function(data, nknots) {
  mse_data = data.frame(matrix(nrow = 0, ncol = 4))

  # Column names based on the knots
  cnames = c()
  cnames_mse = c()
  for (x in 1:length(nknots)) {
    cnames[x] = paste(c('nkonts_', nknots[x]), collapse ='')
    cnames_mse[x]= paste(c('MSE_of_nkonts_', nknots[x]), collapse ='')
  }
  colnames(mse_data) = cnames


  # Fitting and data splitting
  for (i in 1:10) {
    model_data = make_sample(data, 123 + i, 0.7)

    train = model_data$train
    test = model_data$test

    row_name = paste(c('MSE_', i), collapse = '')

    for (j in 1:length(nknots)) {
      fit = lm(lmr ~ ns(Day, df = nknots[j] + 1), data = train) # Hardcoded
      pred = predict(fit, test)

      # Warning: prediction from a rank-deficient fit may be misleading

      # A matrix that does not have "full rank" is said to be "rank deficient".
      # A matrix is said to have full rank if its rank is either equal to its
      # number of columns or to its number of rows (or to both).

      mse_data[i, j] = calc_mse(test$lmr, pred)
    }
  }
```

```
  # Return data
  return_data = vector(mode="list", length=5)

  names(return_data) = c('MSE_data', cnames_mse)

  return_data$MSE_data = mse_data
  for (c in 1:ncol(mse_data)) {
    return_data[c + 1] = mean(mse_data[, c])
  }

  return(return_data)
}


print(test(mortality_rate, nknots=c(1, 2, 15, 50)))
```

```
## $MSE_data
##      nkonts_1  nkonts_2   nkonts_15    nkonts_50
## 1   1.7408974 0.2701516 0.254657398 5.664550e+30
## 2   0.5893208 1.2064456 0.008371057 6.671313e+28
## 3   0.7408476 1.3057002 5.682091604 3.911257e+39
## 4   0.6303391 2.8017029 0.021897559 4.607687e+27
## 5   2.8447095 2.7585691 2.252976792 3.194681e+33
## 6   2.5540288 2.0878584 2.624939018 4.284474e+29
## 7   1.4090349 1.1673164 0.022260675 1.505071e+32
## 8   2.7118099 1.3700389 0.321958189 1.644365e+38
## 9   2.0105508 2.3455332 3.143670603 2.413490e+28
## 10  1.1457163 2.6349757 1.222091743 3.434806e+34
##
## $MSE_of_nkonts_1
## [1] 1.637726
##
## $MSE_of_nkonts_2
## [1] 1.794829
##
## $MSE_of_nkonts_15
## [1] 1.555491
##
## $MSE_of_nkonts_50
## [1] 4.075731e+38
```

> Compared to 1 b MSEs are considerably higher. With this seed best MSE (1.56) is from model with fifthteen knots. Model with fifty knots gives absurd MSE.

1.d. The final task of this assignment is to fit smoothing splines using the smooth.spline() function in R. Use generalized cross-validation to find the optimal degree of smoothing on each of the training data. Provide a plot of the data points and fitted spline curve for the first training data. Also, present the average of the estimated effective degrees of freedom and the smoothing parameter λ, as well as the number of proper knots. Compare the average predicted MSE with the best MSE from 1.c.

```
data = make_sample(mortality_rate, 2707, 0.7)

train = data$train
test = data$test

smooth <- smooth.spline(x=train$Day, y=train$lmr, cv = T)
pred = stats:::predict.smooth.spline(smooth, test$Day)

cat(sprintf('MSE of the best lambda: %.2g \n', calc_mse(test$lmr, pred$y)))
```

```
## MSE of the best lambda: 0.055
```

```
cat(sprintf('Lambda: %.2g \n', smooth$lambda))  #Smoothing parameter
```

```
## Lambda: 0.00098
```

```
cat(sprintf('Number of Knots: %s \n', smooth$fit$nk))
```

```
## Number of Knots: 24
```

```
cat(sprintf('Avarage of the effective Degrees of Freedom: %.3g \n', smooth$df - 2))
```

```
## Avarage of the effective Degrees of Freedom: 3.35
```

> Smoothing splines performs effectively comprored to natural cubic splines with MSE of 0.055 and difference to best cubic models MSE is (1.555 - 0.055) 1.05.

Assignment 2: Comparison of GAMs with spline and loess basis functions

Pre

```
auto_mpg = read.table('auto-mpg.txt', header = T)

auto_mpg$year = factor(auto_mpg$year)
auto_mpg$origin = factor(auto_mpg$origin)

auto_mpg = auto_mpg[ , !(names(auto_mpg) %in% 'name')]

summary(auto_mpg)
```

```
##       mpg          cylinders      displacement     horsepower        weight    
##  Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1613  
##  1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0   1st Qu.:2225  
##  Median :22.75   Median :4.000   Median :151.0   Median : 93.5   Median :2804  
##  Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5   Mean   :2978  
##  3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0   3rd Qu.:3615  
##  Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5140  
##                                                                                
##   acceleration        year     origin 
##  Min.   : 8.00   73     : 40   1:245  
##  1st Qu.:13.78   78     : 36   2: 68  
##  Median :15.50   76     : 34   3: 79  
##  Mean   :15.54   75     : 30          
##  3rd Qu.:17.02   82     : 30          
##  Max.   :24.80   70     : 29          
##                  (Other):193          
```

```
# 2.a. Set up a lasso model (option ‘glmnet’) for all data with the train()
# function in the caret and use LOOCV to find the RMSE.
# In order to tune the alpha and lambda parameters, set the tuneLength = 50.
# Present the best model (not all) and its model fit criteria.

library(caret)

tuneGrid <- expand.grid(alpha = 1, lambda = seq(0.0001, 1, length = 50))

fit_lasso = train(mpg ~.,
                  data = auto_mpg,
                  method = "glmnet",
                  tuneGrid = tuneGrid,
                  tuneLength = 50,
                  trControl = trainControl("LOOCV", number = 1))


# Does not work
#fit_lasso$finalModel
getTrainPerf(fit_lasso)[1]
```

```
##   TrainRMSE
## 1  3.143726
```

> How I can get the best model? fit_lasso$finalModel gives strange output

```
# 2.b. Set up a GAM model with spline basis functions for all data
# with the train() function in the caret package and use LOOCV to find the best RMSE.
# Find a reasonable interval for the tuning parameter df. Present the table
# of the evaluated models and their model fit criteria.

library(gam)
# Build the GAM model based on splines and find best model with LOOCV
fit_gam_spline <- train(
  mpg~., data = auto_mpg, method = "gamSpline",
  scale = FALSE,
  trControl = trainControl("LOOCV", number = 1),
  tuneGrid = expand.grid(df = seq(1, 9, length = 9)))
```

```
fit_gam_spline
```

```
## Generalized Additive Model using Splines
##
## 392 samples
##   7 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 391, 391, 391, 391, 391, 391, ...
## Resampling results across tuning parameters:
##
##   df  RMSE      Rsquared   MAE
##   1   3.144266  0.8373873  2.404518
##   2   2.726054  0.8777006  1.992124
##   3   2.714038  0.8787980  1.973555
##   4   2.731593  0.8772613  1.989474
##   5   2.750592  0.8755802  2.006770
##   6   2.769341  0.8739123  2.022423
##   7   2.788327  0.8722171  2.037122
##   8   2.807811  0.8704709  2.053446
##   9   2.827439  0.8687046  2.069307
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was df = 3.
```

```
#summary(fit_gam_spline$finalModel)
cat(sprintf('Best model RMSE: %.4g', min(fit_gam_spline$results$RMSE)))
```

```
## Best model RMSE: 2.714
```

2.c

```
# 2.c. Set up a GAM model with loess basis functions for all data with the train()
# function in the caret package and use LOOCV to find the RMSE. Find a reasonable
# interval for the tuning parameter span and fix degree to 1.
# Present the table of the evaluated models and their model fit criteria.

# Build the GAM model based on splines and find best model with LOOCV
fit_gam_loess <- train(
  mpg~., data = auto_mpg, method = "gamLoess",
  scale = FALSE,
  trControl = trainControl("LOOCV", number = 1),
  tuneGrid = expand.grid(span = seq(0.1, 0.5, length = 9),
                       degree = c(rep(1,9))))
```

```
fit_gam_loess
```

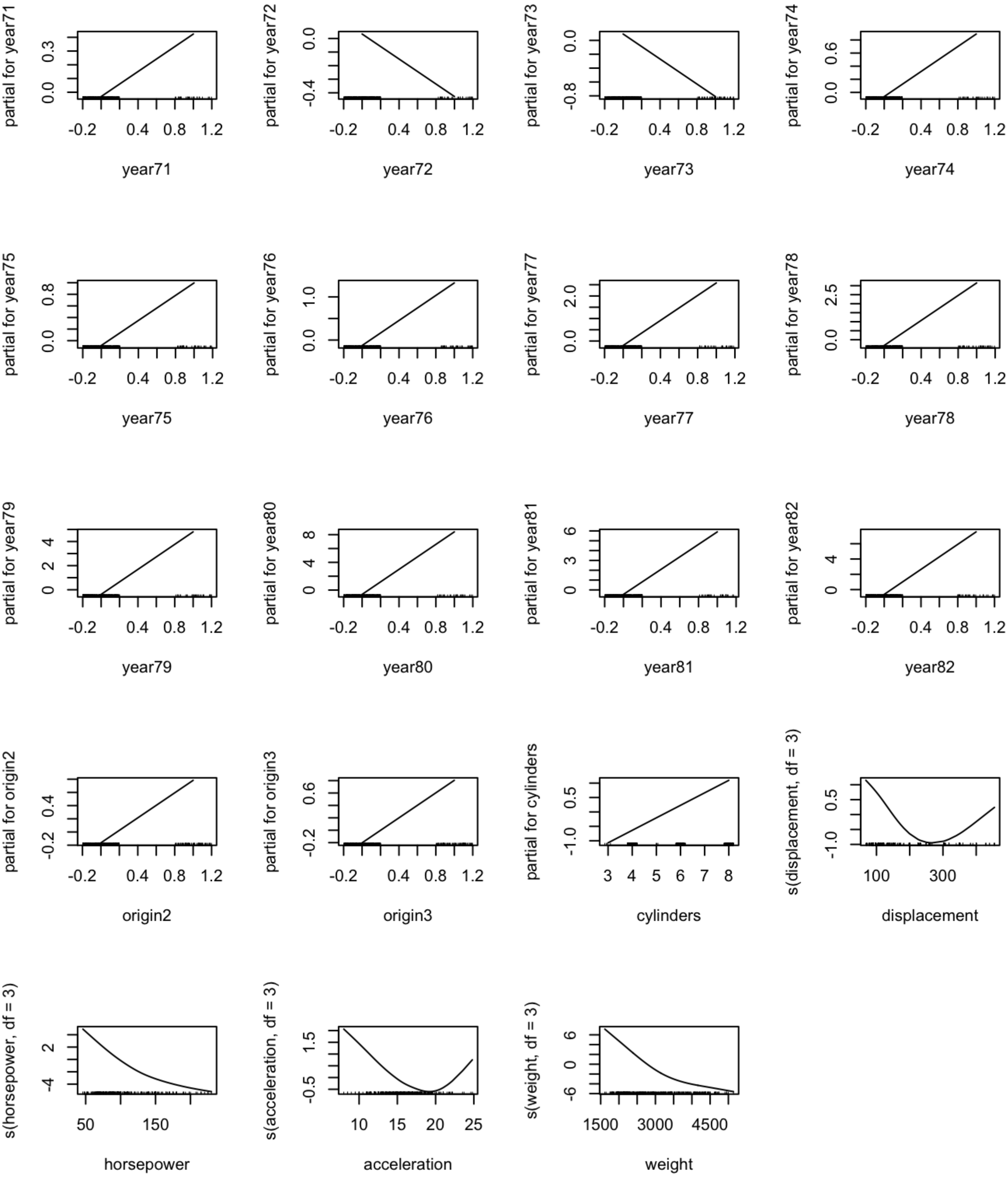
```
## Generalized Additive Model using LOESS
##
## 392 samples
##   7 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 391, 391, 391, 391, 391, 391, ...
## Resampling results across tuning parameters:
##
##   span  RMSE      Rsquared   MAE
##   0.10  2.981868  0.8548833  2.182441
##   0.15  2.847199  0.8669722  2.069900
##   0.20  2.792291  0.8718846  2.039434
##   0.25  2.784916  0.8725021  2.039142
##   0.30  2.785448  0.8724393  2.031661
##   0.35  2.767810  0.8740268  2.015848
##   0.40  2.755734  0.8751059  2.005235
##   0.45  2.745060  0.8760588  1.995748
##   0.50  2.734399  0.8769991  1.986050
##
## Tuning parameter 'degree' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were span = 0.5 and degree = 1.
```

```
cat(sprintf('RMSE: %.4g', min(fit_gam_loess$results$RMSE)))
```

```
## RMSE: 2.734
```

2.d. Best Model

```
# Present a plot of the regression coefficient/curves for the method that performed best.

# Best model is Generalized Additive Model using Splines

par(mfrow = c(3,4))
plot(fit_gam_spline$finalModel)
```



partial for year71
year71
partial for year72
year72
partial for year73
year73
partial for year74
year74
partial for year75
year75
partial for year76
year76
partial for year77
year77
partial for year78
year78
partial for year79
year79
partial for year80
year80
partial for year81
year81
partial for year82
year82
partial for origin2
origin2
partial for origin3
origin3
partial for cylinders
cylinders
s(displacement, df = 3)
displacement
s(horsepower, df = 3)
horsepower
s(acceleration, df = 3)
acceleration
s(weight, df = 3)
weight