Homework 1: Smoothers and Generalized Additive Models

Harvard CS 109B, Spring 2018

Jan 2018

Contents

Homework 1 is due Feburary 7, 2018

Problem 1: Modeling Seasonality of Airbnb Prices

In this problem, the task is to build a regression model to predict the price of an Airbnb rental for a given date. The data is provided in `calendar_train.csv` and `calendar_test.csv`, which contains availability and price data for Airbnb units in the Boston area from 2017 to 2018. Note that some of the rows in the .csv file refer to dates in the future. These refer to bookings that have been made far in advance.

Exploratory Analysis

Visualize the average price by month and day of the week (i.e. Monday, Tuesday etc.) for the training set. Point out any trends you notice and explain whether or not they make sense.

Hint: You will want to first convert the `date` column into an R Date object using `as.Date()`.

Libraries

```
library(gam)
library(ggplot2)
library(splines)
library(MASS)
library(readr)  #for reading csv files
library(scales)  #for controlling chart alpha values
```

```r
library(dplyr)
library(lubridate)
library(gridExtra)
library(ggmap)
```

Seed for Pseudo-RNG

```r
set.seed(11235813)
```

Solution:
Load Train and Test Sets

```r
#load train and test set
cal_train <- read.csv("data/calendar_train.csv")
cal_test  <- read.csv("data/calendar_test.csv")
```

Inspect Train Set—Data check

```r
#inspect
summary(cal_train)
```

```
##    listing_id            date         available      price
##  Min.   :    3781   6/13/18:  2076   f:424715   Min.   :  15.0
##  1st Qu.: 7281884   3/22/18:  2072   t:309288   1st Qu.: 115.0
##  Median :13908638   7/19/18:  2070              Median : 190.0
##  Mean   :12538637   5/5/18 :  2066              Mean   : 238.8
##  3rd Qu.:18354570   11/1/17:  2062              3rd Qu.: 299.0
##  Max.   :21228356   1/17/18:  2060              Max.   :5993.0
##                     (Other):721597              NA's   :424715
```

```r
str(cal_train)
```

```
## 'data.frame':    734003 obs. of  4 variables:
##  $ listing_id: int  20872145 20872145 20872145 20872145 20872145 20872145 20872145 20872145 20872145
##  $ date      : Factor w/ 365 levels "1/1/18","1/10/18",..: 349 346 345 344 343 342 341 340 339 337 .
##  $ available : Factor w/ 2 levels "f","t": 1 1 1 1 1 1 1 1 1 1 ...
##  $ price     : int  NA NA NA NA NA NA NA NA NA NA ...
```

```r
cat("Train data size: ", dim(cal_train), "\n")
```

```
## Train data size:  734003 4
```

```r
head(cal_train)
```

```
##   listing_id    date available price
## 1   20872145 9/21/18         f    NA
## 2   20872145 9/19/18         f    NA
## 3   20872145 9/18/18         f    NA
## 4   20872145 9/17/18         f    NA
## 5   20872145 9/16/18         f    NA
## 6   20872145 9/15/18         f    NA
```

Inspect Test Set—Data check

```
#inspect
summary(cal_test)
```

```
##    listing_id             date        available      price        
##  Min.   :    3781   12/21/17:   934   f:181472   Min.   :   15.0  
##  1st Qu.: 7281884   9/28/18 :   934   t:133100   1st Qu.:  115.0  
##  Median :13908031   11/19/17:   920              Median :  190.0  
##  Mean   :12537918   4/6/18  :   917              Mean   :  239.7  
##  3rd Qu.:18356892   5/30/18 :   917              3rd Qu.:  300.0  
##  Max.   :21228356   7/23/18 :   913              Max.   :10000.0  
##                     (Other) :309037              NA's   :181472
```

```
str(cal_test)
```

```
## 'data.frame':    314572 obs. of  4 variables:
##  $ listing_id: int  21205442 5166870 9698823 18894466 6765855 11710680 18052990 4555637 15020558 135
##  $ date      : Factor w/ 365 levels "1/1/18","1/10/18",..: 356 307 40 137 350 266 221 20 309 296 ...
##  $ available : Factor w/ 2 levels "f","t": 2 2 1 1 1 1 1 1 1 1 ...
##  $ price     : int  138 210 NA NA NA NA NA NA NA NA ...
```

```
cat("Test data size: ", dim(cal_test), "\n")
```

```
## Test data size:  314572 4
```

```
head(cal_test)
```

```
##   listing_id     date available price
## 1   21205442  9/28/18         t   138
## 2    5166870  8/11/18         t   210
## 3    9698823 10/17/17         f    NA
## 4   18894466  2/21/18         f    NA
## 5    6765855  9/22/18         f    NA
## 6   11710680   6/3/18         f    NA
```

Helper Function for month and day and filter for rows with prices

```
convert_filter = function(df) {
  #Conversion date string field into date
  df$date <- as.Date(df$date, format='%m/%d/%y')
  #Feature creation: month, day
  df$month <- month(df$date)
  df$day <- wday(df$date, label=T)

  #Filter for day availability 't'
  available <- filter(df, available=='t')
  return(available)
}
```

Clean Train and Test sets

```
avail_train <- convert_filter(cal_train)
#data check
cat("Clean Train data size: ", dim(avail_train), "\n")
```

```
## Clean Train data size:  309288 6
```

```r
head(avail_train)
```

```
##   listing_id       date available price month day
## 1   20872145 2018-04-02         t    62     4 Mon
## 2   20872145 2018-04-01         t    59     4 Sun
## 3   20872145 2018-03-31         t    75     3 Sat
## 4   20872145 2018-03-30         t    71     3 Fri
## 5   20872145 2018-03-28         t    51     3 Wed
## 6   20872145 2018-03-24         t    46     3 Sat
```

```r
avail_test <- convert_filter(cal_test)
#data check
cat("Clean Test data size: ", dim(avail_test), "\n")
```

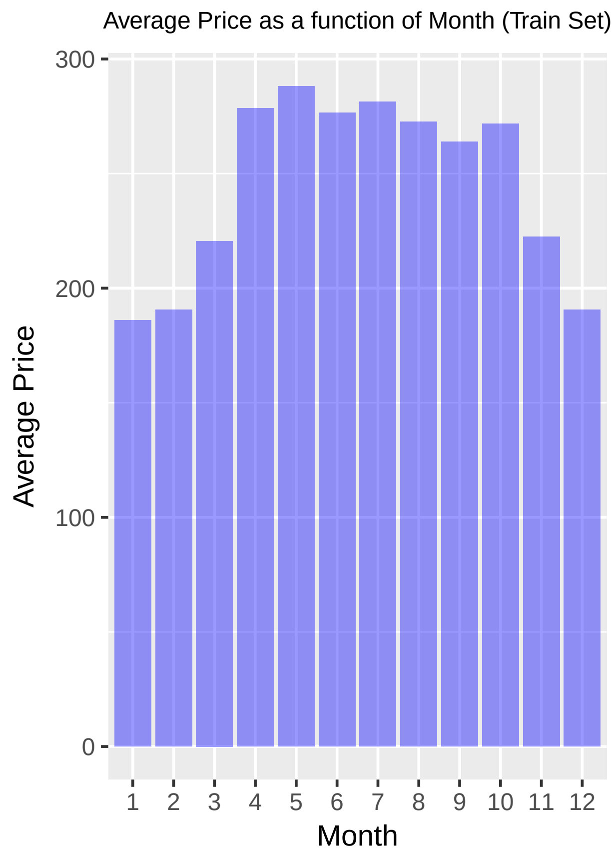
```
## Clean Test data size:  133100 6
```

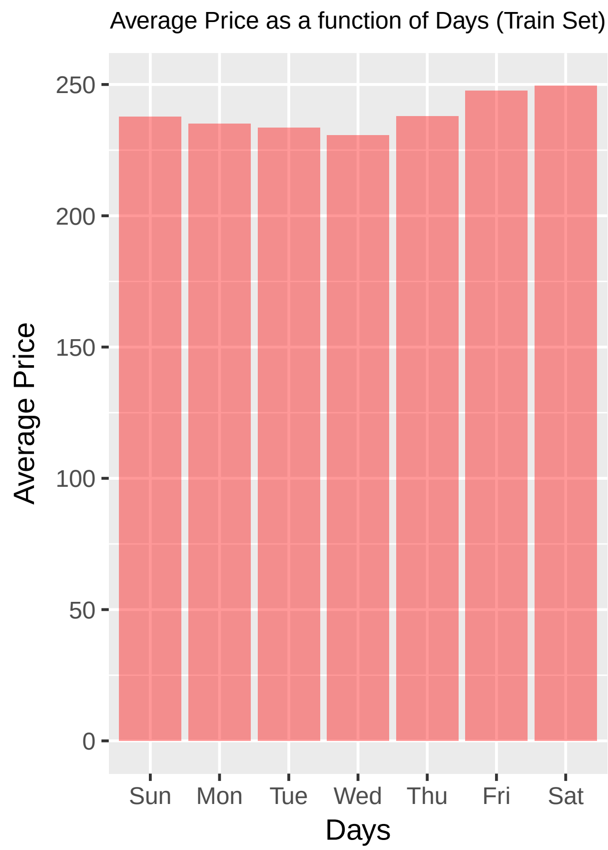
```r
head(avail_test)
```

```
##   listing_id       date available price month day
## 1   21205442 2018-09-28         t   138     9 Fri
## 2    5166870 2018-08-11         t   210     8 Sat
## 3   19455818 2018-04-13         t   869     4 Fri
## 4   20351854 2017-12-23         t   239    12 Sat
## 5   20622324 2018-02-09         t   259     2 Fri
## 6   19309434 2018-06-30         t   227     6 Sat
```

Visualizations — Train Set

```r
#Avg price as a function of the Month
a1 <- ggplot(avail_train, aes(x=as.factor(month), y=price), ) +
  stat_summary(fun.y='mean', geom='bar', fill='blue', alpha=.4) +
  labs(title='Average Price as a function of Month (Train Set)',
       x='Month', y='Average Price') +
  theme(plot.title=element_text(hjust=0.5, size=9))
#Avg price as a function of the day
a2 <- ggplot(avail_train, aes(x=as.factor(day), y=price)) +
  stat_summary(fun.y='mean', geom='bar', fill='red', alpha=.4) +
  labs(title='Average Price as a function of Days (Train Set)',
       x='Days', y='Average Price') +
  theme(plot.title=element_text(hjust=0.5, size=9))
#vizualize
grid.arrange(a1, a2, nrow=1, ncol=2)
```





Trends

- Lower prices in winter months, potentially attributed to demand contraction as they are not traditional (average) vacation months.
- Higher price Fridays and Saturdays.
- Rental pattern per day decreases on average to a week min in Wednesday increasing towards end of the week — *week is considered Sunday through Saturday.* Consistent with rationale that people on average may try to time time-off closer to the weekend.

Part 1a: Explore different regression models

Fit a regression model that uses the date as a predictor and predicts the average price of an Airbnb rental on that date. For this part of the question, you can ignore all other predictors besides the date. Fit the following models on the training set and compare the R^2 of the fitted models on the test set. Include plots of the fitted models for each method.

Hint: You may want to convert the `date` column into a numerical variable by taking the difference in days between each date and the earliest date in the column, which can be done using the `difftime()` function.

1. Regression models with different basis functions:

- Simple polynomials with degrees 5, 25, and 50
- Cubic B-splines with the knots chosen by visual inspection of the data.
- Natural cubic splines with the degree of freedom chosen by cross-validation on the training set

2. Smoothing spline model with the smoothness parameter chosen by cross-validation on the training set

3. Locally-weighted regression model with the span parameter chosen by cross-validation on the training set

In each case, analyze the effect of the relevant tuning parameters on the training and test R^2, and give explanations for what you observe.

Is there a reason you would prefer one of these methods over the other?

Hints: - You may use the function `poly` to generate polynomial basis functions (use the attribute `degree` to set the degree of the polynomial), the function `bs` for B-spline basis functions (use the attribute `knots` to specify the knots), and the function `ns` for natural cubic spline basis functions (use the attribute `df` to specify the degree of freedom). You may use the `lm` function to fit a linear regression model on the generated basis functions. You may use the function `smooth.spline` to fit a smoothing spline and the attribute `spar` to specify the smoothness parameter. You may use the function `loess` to fit a locally-weighted regression model and the attribute `span` to specify the smoothness parameter that determines the fraction of the data to be used to compute a local fit. Functions `ns` and `bs` can be found in the `splines` library.

- For smoothing splines, `R` provides an internal cross-validation feature: this can be used by leaving the `spar` attribute in `smooth.spline` unspecified; you may set the `cv` attribute to choose between leave-one-out cross-validation and generalized cross-validation. For the other models, you will have to write your own code for cross-validation. Below, we provide a sample code for k-fold cross-validation to tune the `span` parameter in `loess`:

Solution:

Helper Functions

R^2 function

```
rsq = function(y, predict){
  tss = sum((y - mean(y))^2)
  rss = sum((y - predict)^2)
  r_squared = 1 - rss/tss

  return(r_squared)
}
```

K-fold CV for Loess span parameter tuning

```
crossval_loess = function(train, param_val, k) {
  #Input:
  #   Training data frame: 'train',
  #   Vector of span paramter values: 'param_val',
  #   Number of CV folds: 'k'
  #Output:
  #   Vector of R^2 values for the provided parameters: 'cv_rsq'

  num_param = length(param_val) # Number of paramters

  # seed for pseudo rng set atop

  # Divide training set into k folds by sampling uniformly at random
  # fold[s] has the fold index for train instance 's'
  folds = sample(1:k, nrow(train), replace = TRUE)
```

```r
  cv_rsq = rep(0., num_param) # Store cross-validated R^2 for different parameter values

  #iterate over parameter values
  for(i in 1:num_param){
    # Iterate over the folds to compute R^2 for paramter
    for(j in 1:k){
      # Fit model on all folds other than 'j' with parameter value param_val[i]
      model.loess = loess(price ~ days_since, span = param_val[i],
                          data = train[folds!=j, ],
                          control = loess.control(surface = "direct"))

      # Make prediction on fold 'j'
      pred = predict(model.loess, train$days_since[folds == j])

      # Compute R^2 for predicted values
      cv_rsq[i] = cv_rsq[i] + rsq(train$price[folds == j], pred)
    }
    # Average R^2 across k folds
    cv_rsq[i] = cv_rsq[i]/k
  }
  # Return cross-validated R^2 values
  return(cv_rsq)
}
```

Plotting

```r
plot_the_fit_r2 = function(model, model_name, flag) {

  #train, test r2 calcs
  pred_train <- predict(model, newdata=train1_agg)
  pred_test <- predict(model, newdata=test1_agg)

  train_rsq <- 0
  test_rsq <- 0

  if (flag==TRUE){ # if smoothing spline
    train_rsq <- rsq(train1_agg$price, predict(model, newdata=train1_agg)$y)
    test_rsq <- rsq(test1_agg$price, predict(model, newdata=test1_agg)$y)
    } else {
      train_rsq <- rsq(train1_agg$price, pred_train)
      test_rsq <- rsq(test1_agg$price, pred_test)
    }

  title_str <- sprintf("%s: Train R^2 = %.3f, Test R^2 = %.3f",
                       model_name, train_rsq, test_rsq)

  #plot
  p <- ggplot()
  if (flag==TRUE) {
    p <- ggplot(train1_agg, aes(x=days_since, y=price)) +
    geom_point() +
    geom_line(aes(y = predict(model, newdata=data.frame(days_since))$y, colour='red')) +
    labs(x="Days Since", y="Average Price", title=title_str) +
    theme(plot.title= element_text(hjust=0.5, size=9))
```

```r
  } else {
    p <- ggplot(train1_agg, aes(x=days_since, y=price)) +
    geom_point() +
    geom_line(aes(y = predict(model, newdata=data.frame(days_since)), colour='red')) +
    labs(x="Days Since", y="Average Price", title=title_str) +
    theme(plot.title= element_text(hjust=0.5, size=9))
  }

  return(list(test_rsq=test_rsq, p=p))
}
```

1. Regreesion Models with Different Basis Functions

Simple Polynomials degree 5, 25 and 50.

Feature creation: `days_since` `days_since`:= number of days since the earliest date in the training set

```r
avail_train$days_since <- as.numeric(difftime(avail_train$date, min(avail_train$date),
                                              units='days'))
avail_test$days_since <- as.numeric(difftime(avail_test$date, min(avail_train$date),
                                             units='days'))

# Create aggregated dataframe containing mean price for each time
train1_agg = aggregate(x=avail_train[, c('days_since', 'price')],
                       by=list(avail_train$days_since), FUN=mean)
head(train1_agg)
```

```
##   Group.1 days_since    price
## 1       0          0 370.1734
## 2       1          1 409.2987
## 3       2          2 299.1287
## 4       3          3 282.7689
## 5       4          4 275.9332
## 6       5          5 265.7014
```

```r
test1_agg = aggregate(x=avail_test[, c('days_since', 'price')],
                      by=list(avail_test$days_since), FUN=mean)
```

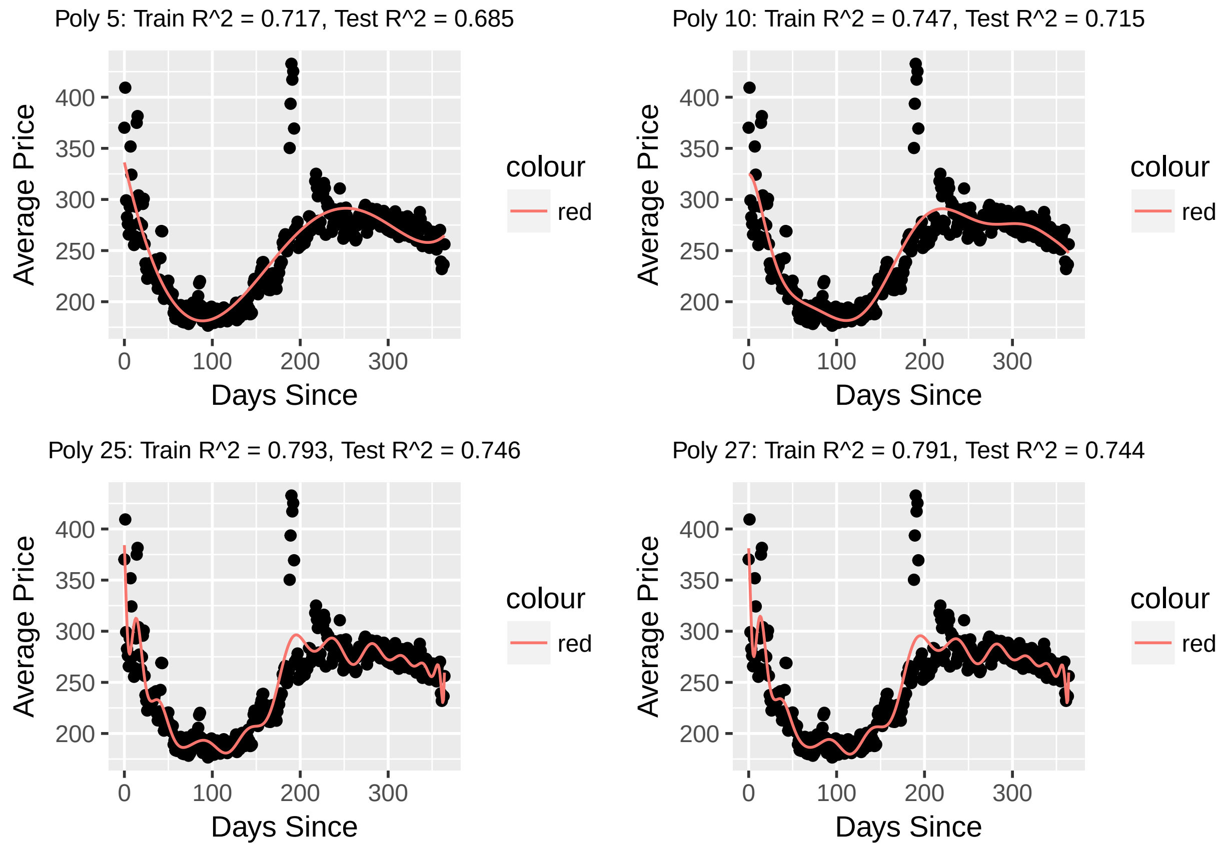
Fit and plot polys

```r
fit_plot_poly = function(degree) {

  model.poly_ <- lm(price ~ poly(days_since, degree), data=train1_agg)

  #train, test r2 calcs
  pred_train <- predict(model.poly_, newdata=train1_agg)
  pred_test <- predict(model.poly_, newdata=test1_agg)

  train_rsq <- rsq(train1_agg$price, pred_train)
  test_rsq <- rsq(test1_agg$price, pred_test)

  title_str <- sprintf("Poly %d: Train R^2 = %.3f, Test R^2 = %.3f",
                       degree, train_rsq, test_rsq)

  #plot
```

```
  p <- ggplot(train1_agg, aes(x=days_since, y=price)) +
    geom_point() +
    geom_line(aes(y = predict(model.poly_, newdata=data.frame(days_since)),
                  colour='red')) +
    labs(x="Days Since", y="Average Price", title=title_str) +
    theme(plot.title= element_text(hjust=0.5, size=9))

  return(list(test_rsq=test_rsq, p=p))
}
```

Fit and plot polys degrees, 5, 10, 25, 50

```
## poly fits
p5 <- invisible(fit_plot_poly(5)) #Poly 5
p10 <- fit_plot_poly(10) #Poly 10
p25 <- fit_plot_poly(25) #Poly 25
p27 <- fit_plot_poly(27) #Poly 27

# NOTE ###########################################################################
#p50 <- fit_plot_poly(50) #Poly 50
#cannot be performed due to data set size ... Error in poly(days_since, degree) :
#  'degree' must be less than number of unique points
##################################################################################

## If grid not desired for viz ease ---> comment below + uncomment subsequent line
grid.arrange(p5$p, p10$p, p25$p, p27$p, nrow=2, ncol=2)
```



```r
#p5$p; p10$p; p25$p; p27$p
```

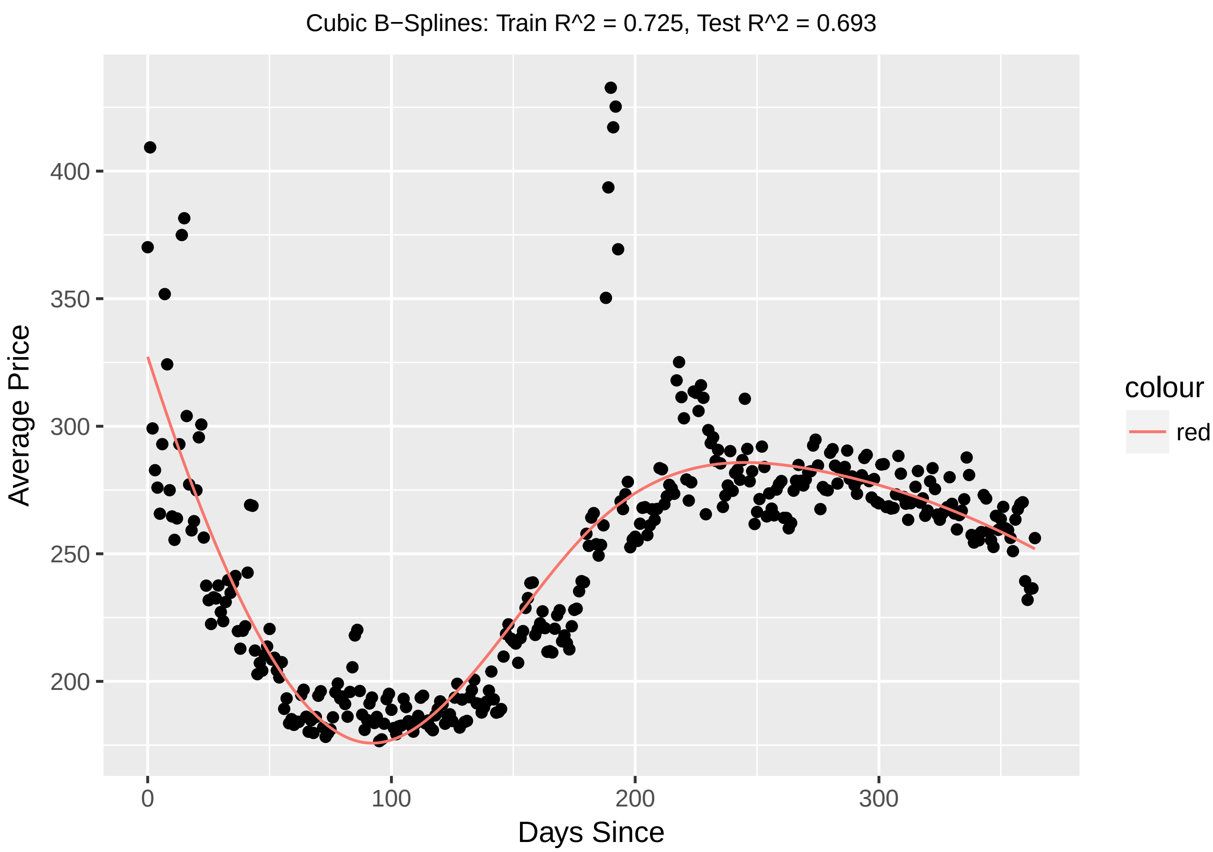
Polynomial Fit Conclusion Seen in the R^2 of the train and test sets, as the polynomial degree increases up to 25 degrees, the R^2 increases. On the training set increases due to more degrees of freedom. As seen in the plots, we risk overfitting the training set as we increase the degrees of freedom — degree 27 where we can see the test R^2 reducing.

- *important to note that the model becomes unstable to fit polynomial degrees larger than 27*

Cubic B-splines with the knots chosen by visual inspection of the data.

Knots placement After visual inspection of the data, we will proceed to a quantile related knot placement.

```r
b.spline <- lm(price ~ bs(days_since, knots=quantile(days_since, c(.25,.50,.75))),
               data=train1_agg)
p_b <- plot_the_fit_r2(b.spline, 'Cubic B-Splines', FALSE)
p_b$p
```

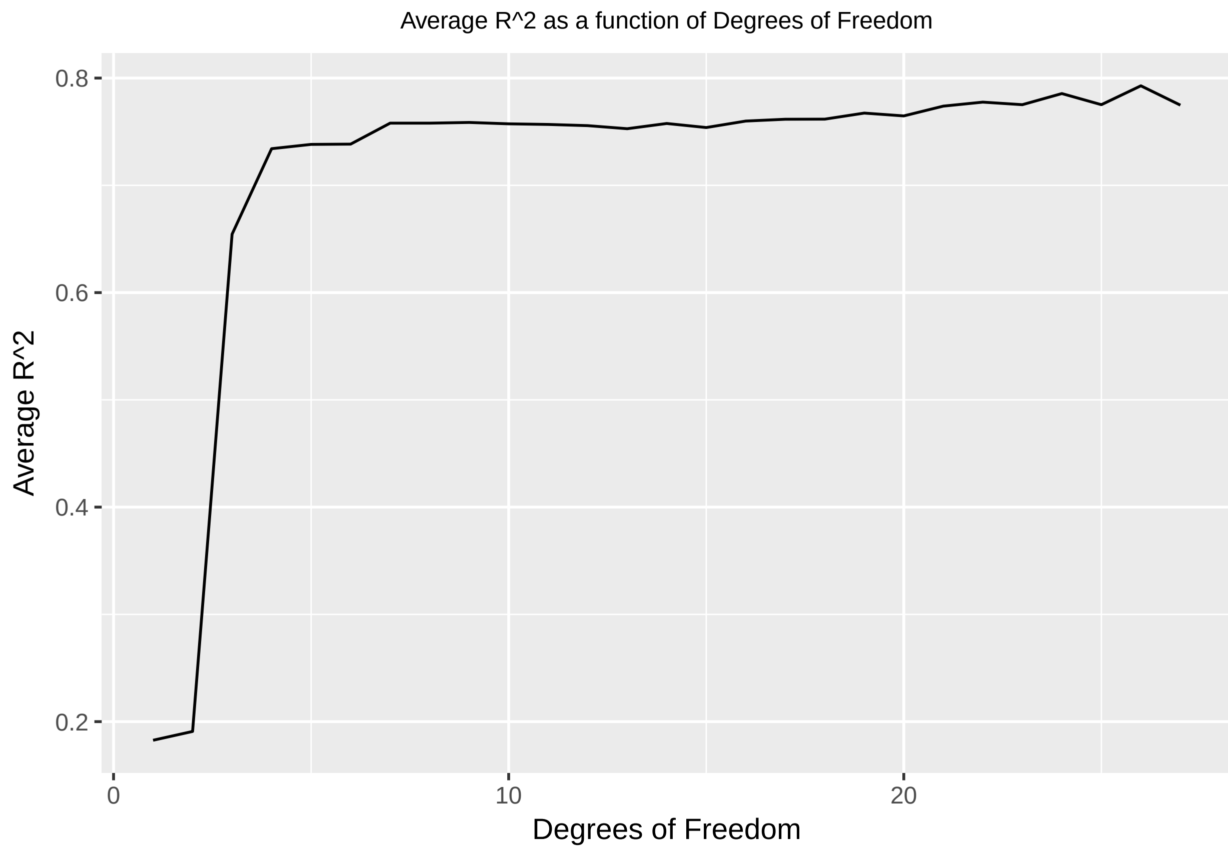


Cubic B-Spline Fit Conclusion Yields a worse R^2 on the test set than the polynomial degree fits. However we can infer that this performance is dependant on the fact of the visual inspection performed for the knot selection. Thus, we can see that Cubic B-splines are very sensitive to knots placement choice.

Natural cubic splines with the degree of freedom chosen by cross-validation on the training set

```r
# Function for k-fold cross-validation to tune degrees of freedom for natural cubic splines
crossval_ns = function(train, param_val, k) {
```

```r
  # Input:
  #   Training data frame: 'train',
  #   Vector of degree of freedom parameter values: 'param_val',
  #   Number of CV folds: 'k'
  # Output:
  #   Vector of R^2 values for the provided parameters: 'cv_rsq'

  num_param = length(param_val) # Number of parameters
  set.seed(109) # Set seed for random number generator

  # Divide training set into k folds by sampling uniformly at random
  # folds[s] has the fold index for train instance 's'
  folds = sample(1:k, nrow(train), replace = TRUE)

  cv_rsq = rep(0., num_param) # Store cross-validated R^2 for different parameter values

  # Iterate over parameter values
  for(i in 1:num_param){
    # Iterate over folds to compute R^2 for parameter
    for(j in 1:k){
      # Fit model on all folds other than 'j' with parameter value param_val[i]
      model = lm(price ~ ns(days_since, df = param_val[i]),
                 data = train[folds!=j, ])

      # Make prediction on fold 'j'
      pred = predict(model, data.frame(days_since = train$days_since[folds == j]))

      # Compute R^2 for predicted values
      cv_rsq[i] = cv_rsq[i] + rsq(train$price[folds == j], pred)
    }

    # Average R^2 across k folds
    cv_rsq[i] = cv_rsq[i] / k
  }

  # Return cross-validated R^2 values
  return(cv_rsq)
}

##degrees of freedom contingent on poly result previously shown
# Run k fold cv with k = 5
ns.r2 = crossval_ns(train = train1_agg, param_val = seq(1,27), k = 5)

# Plot R^2 v. choice of degrees of freedom
ggplot(data.frame(ns.r2=ns.r2, df=seq(1,27))) +#, aes(x = factor(df), y = ns.r2)) +
  geom_line(aes(x = seq(1,27), y = ns.r2) ) +
  labs(title='Average R^2 as a function of Degrees of Freedom',
       x="Degrees of Freedom", y="Average R^2") +
  theme(plot.title=element_text(hjust=0.5, size=9))
```



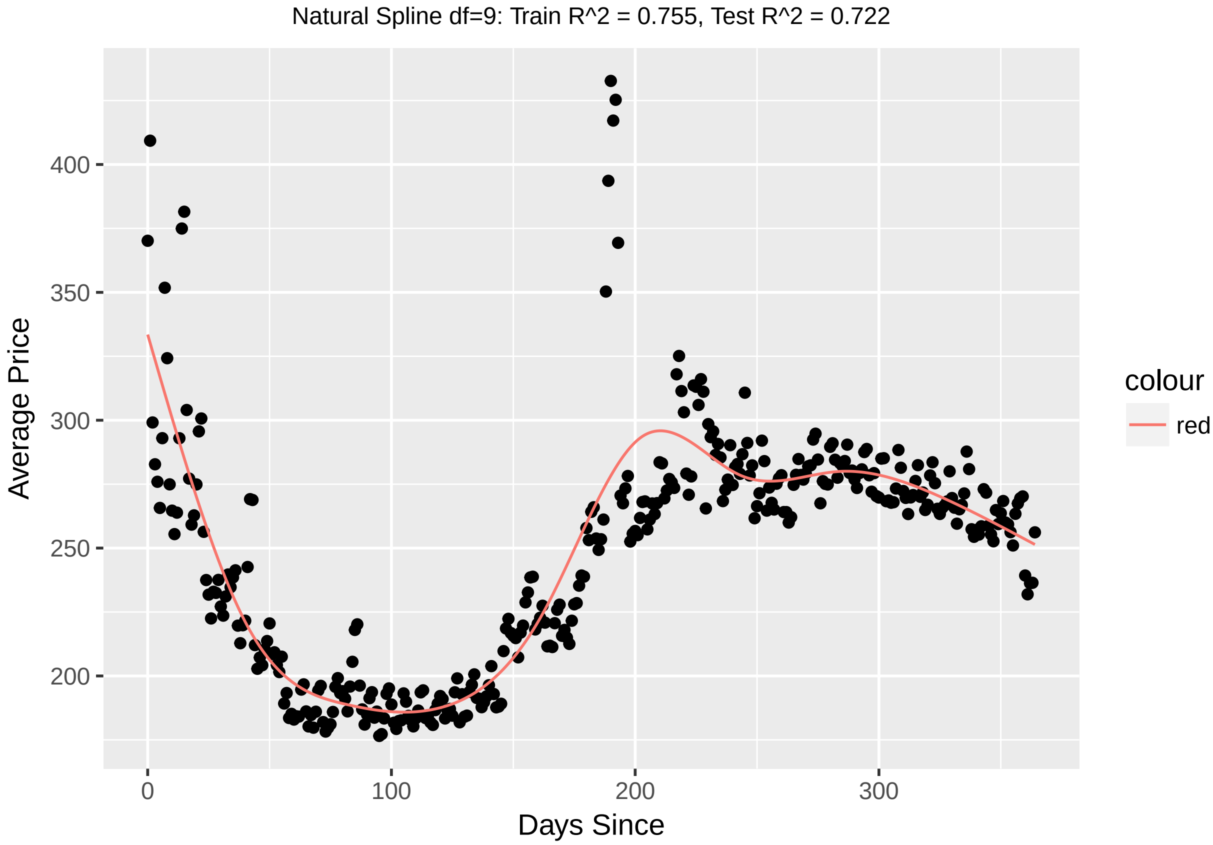

```r
setNames(seq(1,27), ns.r2)
```

```
##  0.18266036591357 0.190861388762872 0.654468449649026 0.734159356630974
##                 1                 2                 3                 4
## 0.738172451596262 0.738416854445836 0.757978835137637 0.757980368905733
##                 5                 6                 7                 8
## 0.758630850286607 0.757331050602873 0.756768956037216 0.755642191910896
##                 9                10                11                12
## 0.752787468192446 0.757667666207314 0.753897839488022 0.759927577156273
##                13                14                15                16
## 0.761616224614979 0.761726194491477 0.767350749430789 0.764736239829175
##                17                18                19                20
## 0.773856029477643 0.777590673287952  0.77518516538436 0.785565127452265
##                21                22                23                24
##  0.77518633450602  0.79280219623256 0.774769634150956
##                25                26                27
```

Per above, we can see via 5-fold cross validation that the first optimization—inflection point (lowest error, highest average R^2) at degrees of freedom $= 9$.

Fit and plot

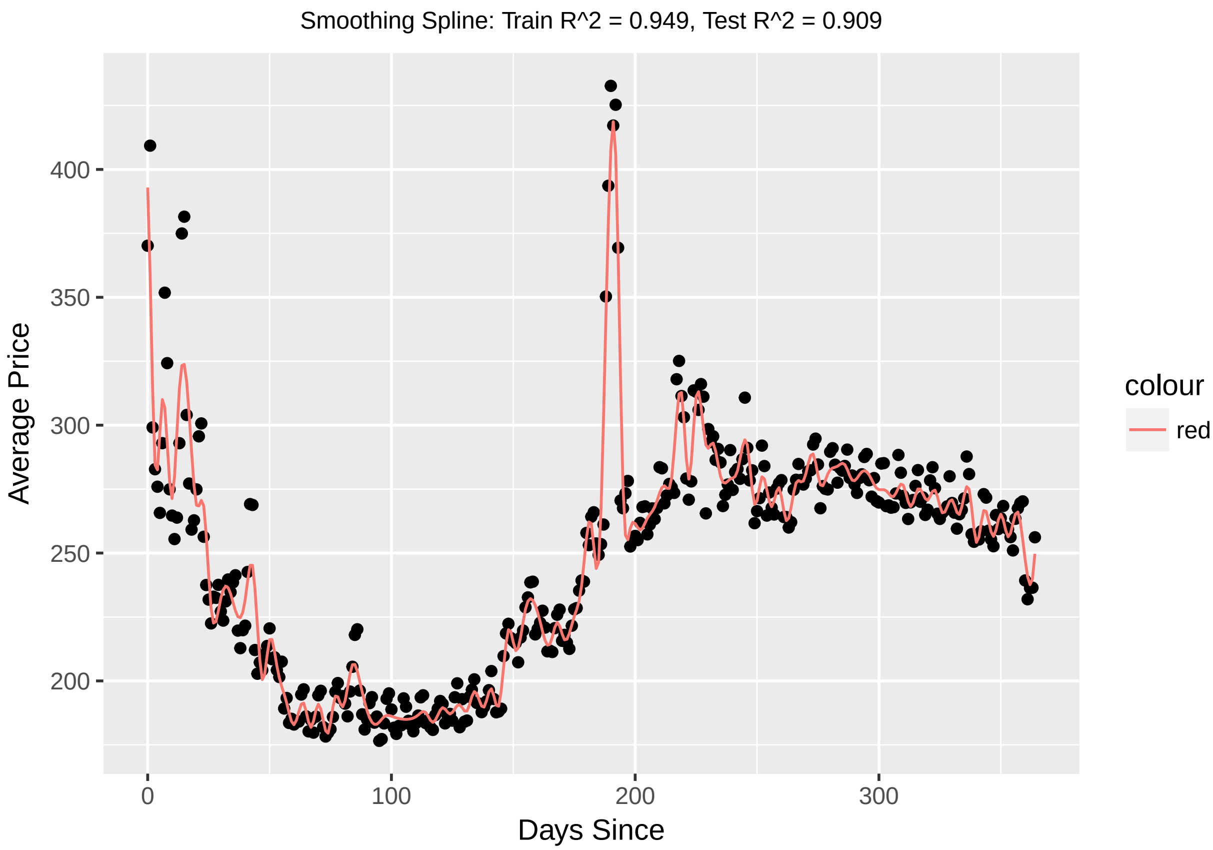
```r
n.spline <- lm(price ~ ns(days_since, df=9), data=train1_agg)
p_n <- plot_the_fit_r2(n.spline, 'Natural Spline df=9', FALSE)
p_n$p
```



2. Smoothing spline model with the smoothness parameter chosen by cross-validation on the training set

Fit and plot

```
s.spline <- smooth.spline(train1_agg$price ~ train1_agg$days_since,
                          cv=TRUE)# choosing default leave one out cv
p_s <- plot_the_fit_r2(s.spline, 'Smoothing Spline', TRUE)
p_s$p
```

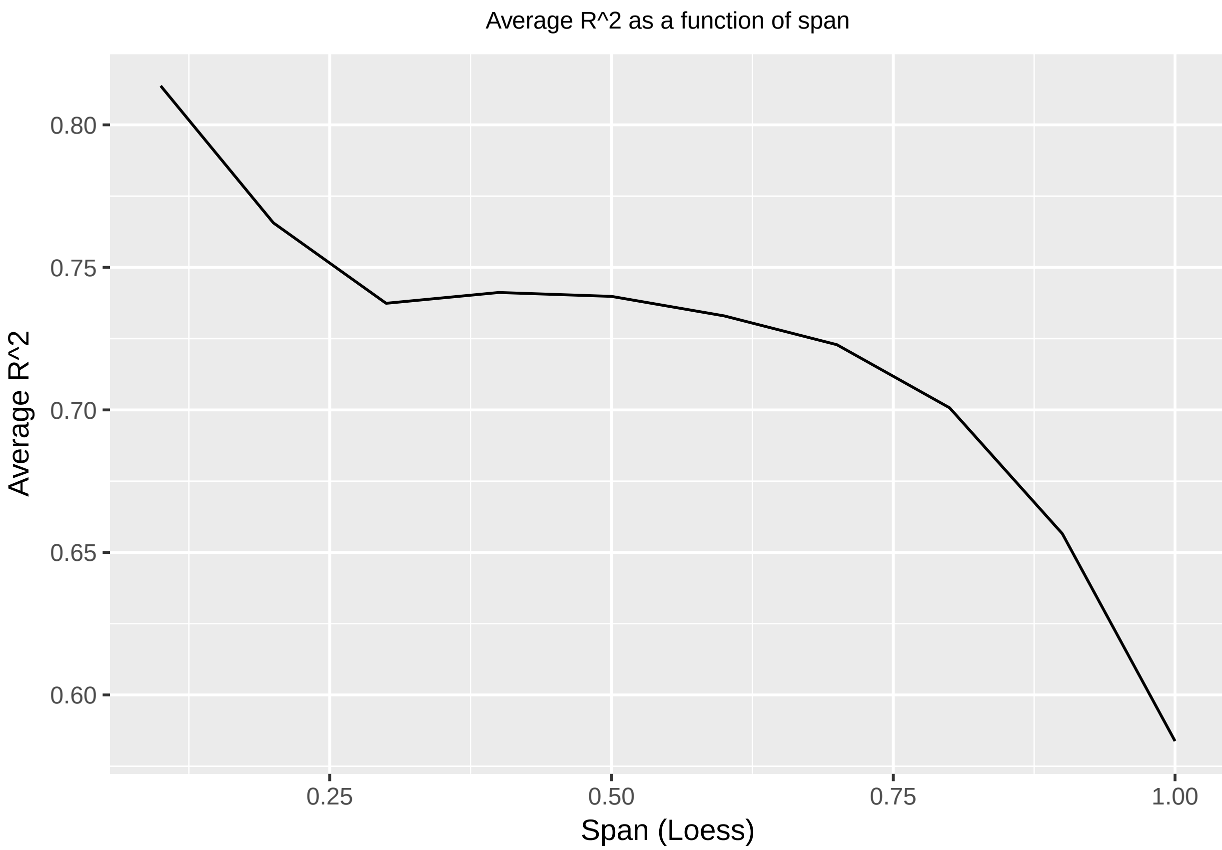


Smoothing Spline Conclusion We have achieve the highest R^2 — for training and test sets. However, we can clearly observe per the graph above, that the fitted model line is not smooth.

3. Locally-weighted regression model with the span parameter chosen by cross-validation on the training set

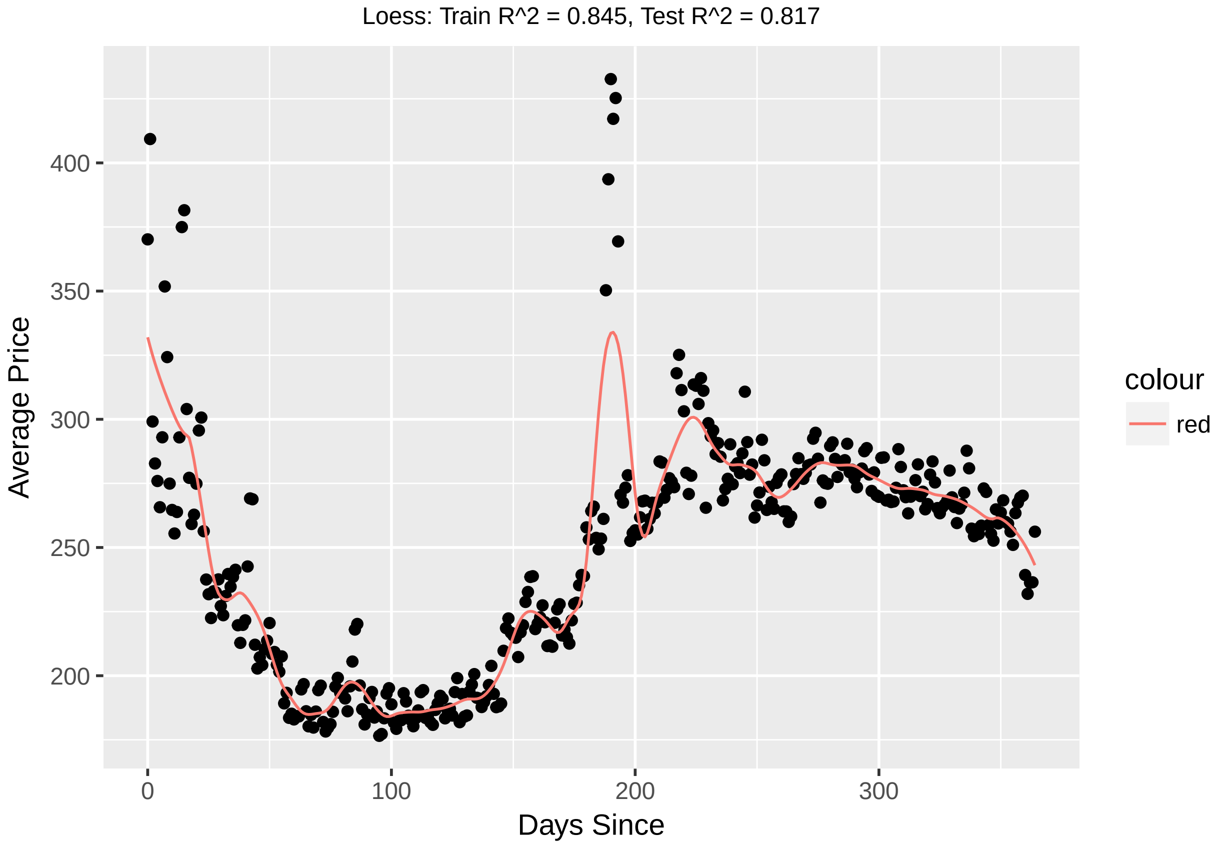
5-fol CV

```
loess.r2 <- crossval_loess(train1_agg, seq(.1, 1, by=.1), 5)

# Plot R^2 v. choice of span
ggplot(data.frame(loess.r2=loess.r2, span=seq(.1, 1, by=.1))) +
  geom_line(aes(x = seq(.1, 1, by=.1), y = loess.r2) ) +
  labs(title='Average R^2 as a function of span',
       x="Span (Loess)", y="Average R^2") +
  theme(plot.title=element_text(hjust=0.5, size=9))
```



Fit and plot

```
loess_m <- loess(price ~ days_since, span=0.1,data=train1_agg,
                 control=loess.control(surface="direct"))
p_l <- plot_the_fit_r2(loess_m, 'Loess', FALSE)
p_l$p
```



Part 1a Overall Conclusions

- As seen per the analysis above, the smoothing spline performs best out of all the models in terms of R^2 on the test set. I would prefer it over B-splines since choosing knots is difficult and would prefer it over polynomial regression since its parameters are chosen via cross-validation as opposed to just choosing arbitrarily.
- That said, the graph for the smoothing spline does does not appear particularly smooth. This suggests that though in this case the test set is perhaps fairly close to the training set (even so, the higher training R^2 than on test suggests overfitting), we should still be careful about overfitting.

Part 1b: Adapting to weekends

Does the pattern of Airbnb pricing differ over the days of the week? Are the patterns on weekends different from those on weekdays? If so, we might benefit from using a different regression model for weekdays and weekends. Split the training and test sets into two parts, one for weekdays and one for weekends, and fit a separate model for each training subset using locally-weighted regression. Do the models yield a higher R^2 on the corresponding test subsets compared to the (loess) model fitted previously? (You may use the loess model fitted in 1A (with the span parameter chosen by CV) to make predictions on both the weekday and weekend test sets, and compute its R^2 on each set separately, you may also use the same best_span calculated in 1A)

> **Solution** As previously stated, there definitely are different patterns on weekends, namely the fact that weekends (Fridays and Saturdays) tend to be more expensive. With that in mind, we will proceed to fit separate models for weekends and weekdays.

```r
day_0 = min(avail_train$date)

# Create index for weekends
train_weekend_index = wday(train1_agg$days_since + day_0, label=T)  %in% c('Sat', 'Sun')
test_weekend_index = wday(test1_agg$days_since + day_0, label=T)  %in% c('Sat', 'Sun')

# Split train and test into weekends and weekdays
train.1.weekends = train1_agg[train_weekend_index,]
train.1.weekdays = train1_agg[!train_weekend_index,]

test.1.weekends = test1_agg[test_weekend_index,]
test.1.weekdays = test1_agg[!test_weekend_index,]

# Fit separate loess models for weekdays and the weekend
loess.weekend = loess(price ~ days_since, span = 0.1,
                          data = train.1.weekends,
                          control = loess.control(surface="direct"))

loess.weekday = loess(price ~ days_since, span = 0.1,
                          data = train.1.weekdays,
                          control = loess.control(surface="direct"))

# Generate predictions for weekends and weekdays separately, concatenating them
we.wd.pred = c(sapply(test.1.weekends$days_since, function(x) predict(loess.weekend, x)),
               sapply(test.1.weekdays$days_since, function(x) predict(loess.weekday, x)))
# Calculate R^2 on test dataset
rsq(c(test.1.weekends$price, test.1.weekdays$price), we.wd.pred)
```

```
## [1] 0.8515486
```

Indeeed, this approach does yield an improved R^2 on the test set, as expected.

Part 1c: Going the Distance

You may have noticed from your scatterplots of average price versus day on the training set that there are a few days with abnormally high average prices.

Sort the training data in decreasing order of average price, extracting the 3 most expensive dates. Given what you know about Boston, how might you explain why these 3 days happen to be so expensive?

```r
# Sort training set in decreasing order of price, extracting 3 most expensive days
# and calculating their date
train1_agg[order(-train1_agg$price),]$days_since[1:3] + min(avail_test$date)
```

```
## [1] "2018-04-14" "2018-04-16" "2018-04-15"
```

Extracting the dates corresponding to the 3 highest average prices, we see that the 3 most expensive days happen to be Marathon Monday and the weekend leading up to it.

Problem 2: Predicting Airbnb Rental Price Through Listing Features

In this problem, we'll continue our exploration of Airbnb data by predicting price based on listing features. The data can be found in `listings_train.csv` and `listings_test.csv`.

First, visualize the relationship between each of the predictors and the response variable. Does it appear that some of the predictors have a nonlinear relationship with the response variable?

```r
listings_train = read.csv('data/listings_train.csv')
listings_test = read.csv('data/listings_test.csv')
dim(listings_train)
```

```
## [1] 4370   12
```

```r
dim(listings_test)
```

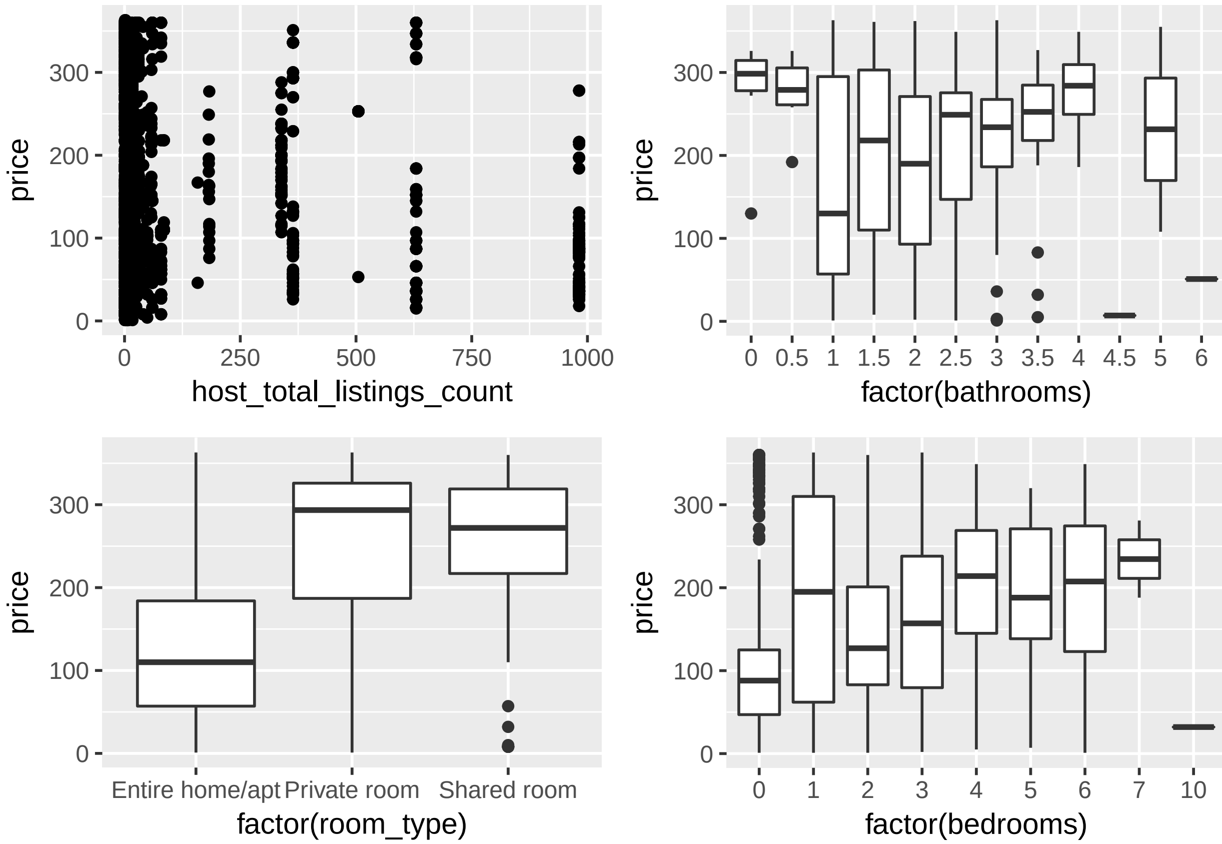
```
## [1] 487  12
```

```r
head(listings_train)
```

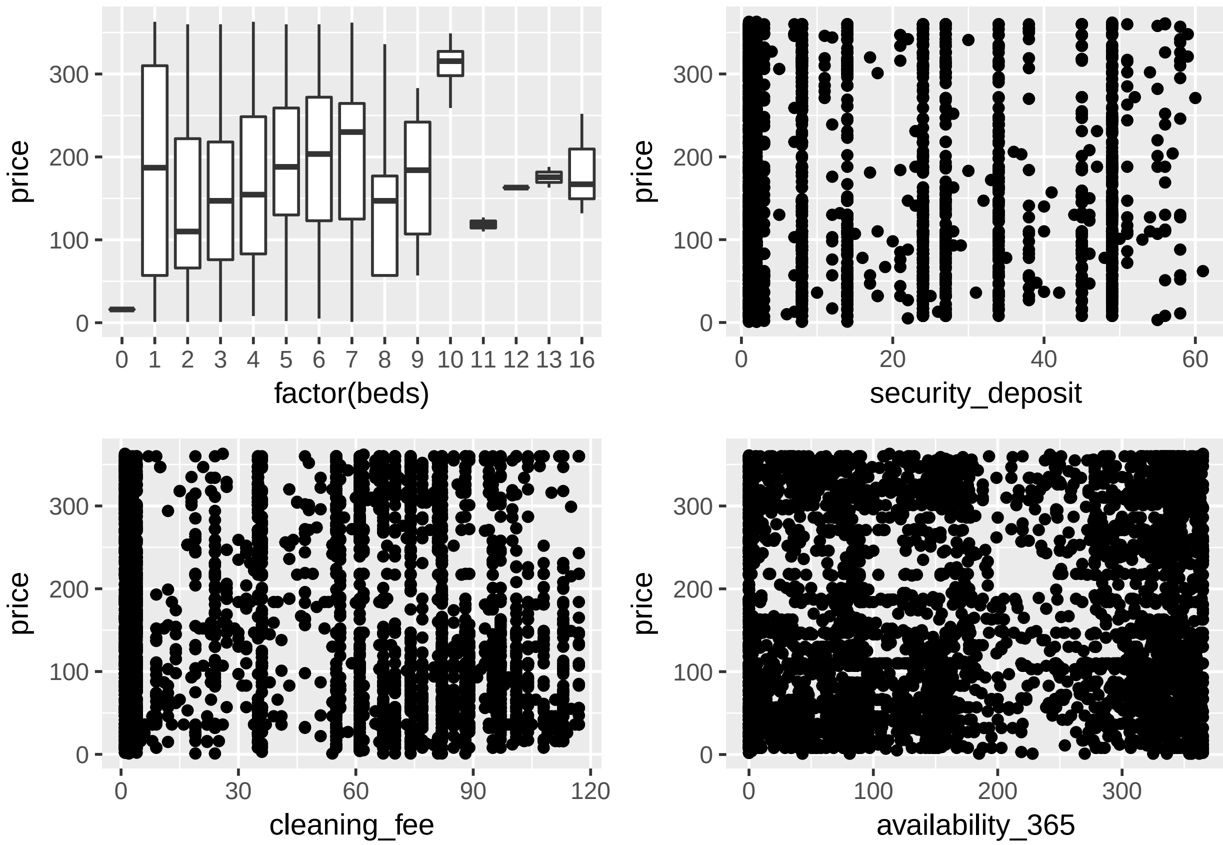
```
##   host_total_listings_count       room_type latitude longitude bathrooms
## 1                         1    Private room 42.34796 -71.15520       1.0
## 2                        85 Entire home/apt 42.34930 -71.08347       1.0
## 3                         6 Entire home/apt 42.34190 -71.07379       1.0
## 4                         1 Entire home/apt 42.31923 -71.10502       2.0
## 5                         1 Entire home/apt 42.34645 -71.13490       1.0
## 6                         1 Entire home/apt 42.34253 -71.05386       1.5
##   bedrooms beds price security_deposit cleaning_fee availability_365
## 1        1    1    52                1           65              365
## 2        0    1   110                1          104              107
## 3        1    1    67               45           56              322
## 4        2    2   103                8          113              341
## 5        0    1     8               24           82               41
## 6        1    1    83                2           67                9
##   number_of_reviews
## 1                26
## 2                38
## 3                 9
## 4                49
## 5                13
## 6                 2
```

```r
p1 = ggplot(listings_train, aes(x=host_total_listings_count, y=price)) + geom_point()
p2 = ggplot(listings_train, aes(x=factor(bathrooms), y=price)) + geom_boxplot()
p3 = ggplot(listings_train, aes(x=factor(room_type), y=price)) + geom_boxplot()
p4 = ggplot(listings_train, aes(x=factor(bedrooms), y=price)) + geom_boxplot()
p5 = ggplot(listings_train, aes(x=factor(beds), y=price)) + geom_boxplot()
p6 = ggplot(listings_train, aes(x=security_deposit, y=price)) + geom_point()
p7= ggplot(listings_train, aes(x=cleaning_fee, y=price)) + geom_point()
p8 = ggplot(listings_train, aes(x=availability_365, y=price)) + geom_point()
p9 = ggplot(listings_train, aes(x=number_of_reviews, y=price)) + geom_point()

m <- get_map("Boston",zoom=13,maptype="toner-labels",source="stamen")

p10 = ggmap(m) +
```

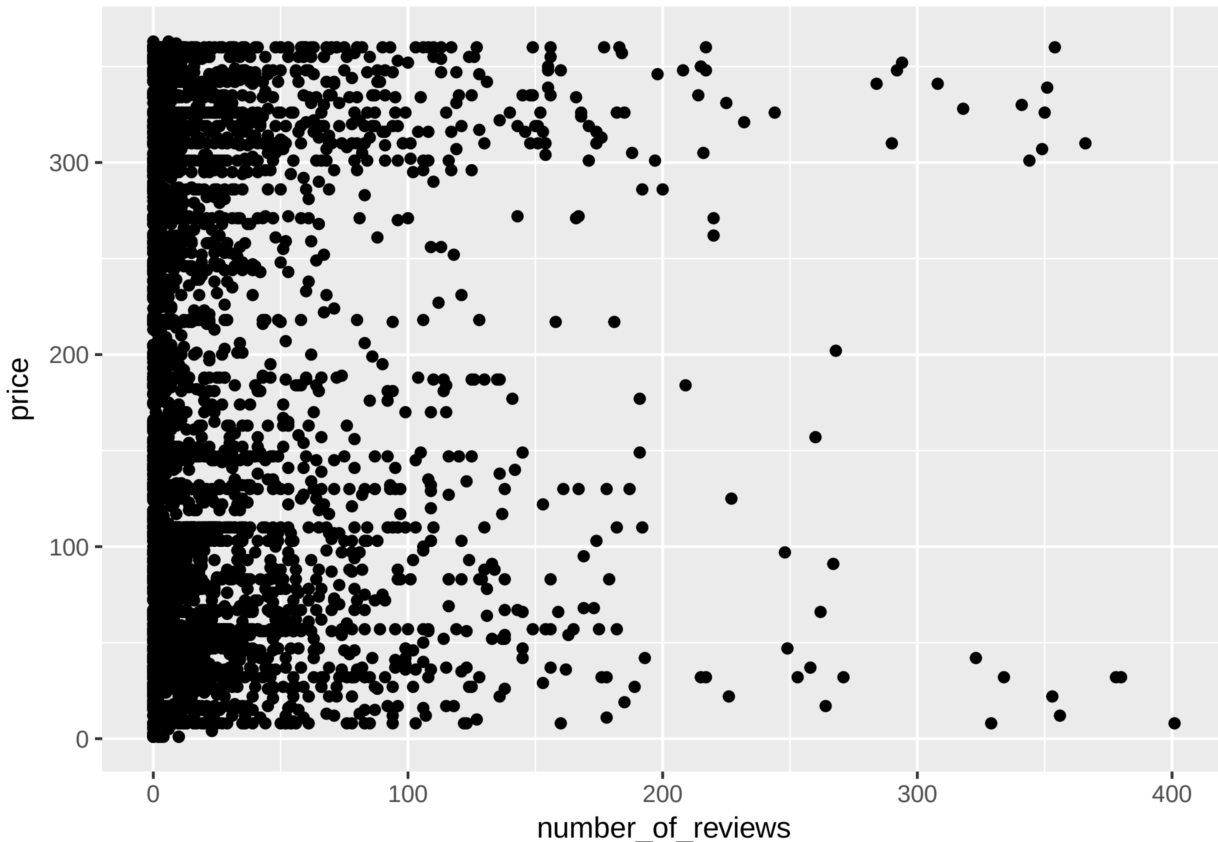
```
    geom_point(aes(x=longitude,y=latitude,color=price) ,data=listings_train) +
    geom_point(size=3,alpha=0.3) +
    xlab("Longitude") +
    ylab("Latitude")

grid.arrange(p1, p2, p3, p4, nrow=2, ncol=2)
```




```
grid.arrange(p5, p6, p7, p8, nrow=2, ncol=2)
```

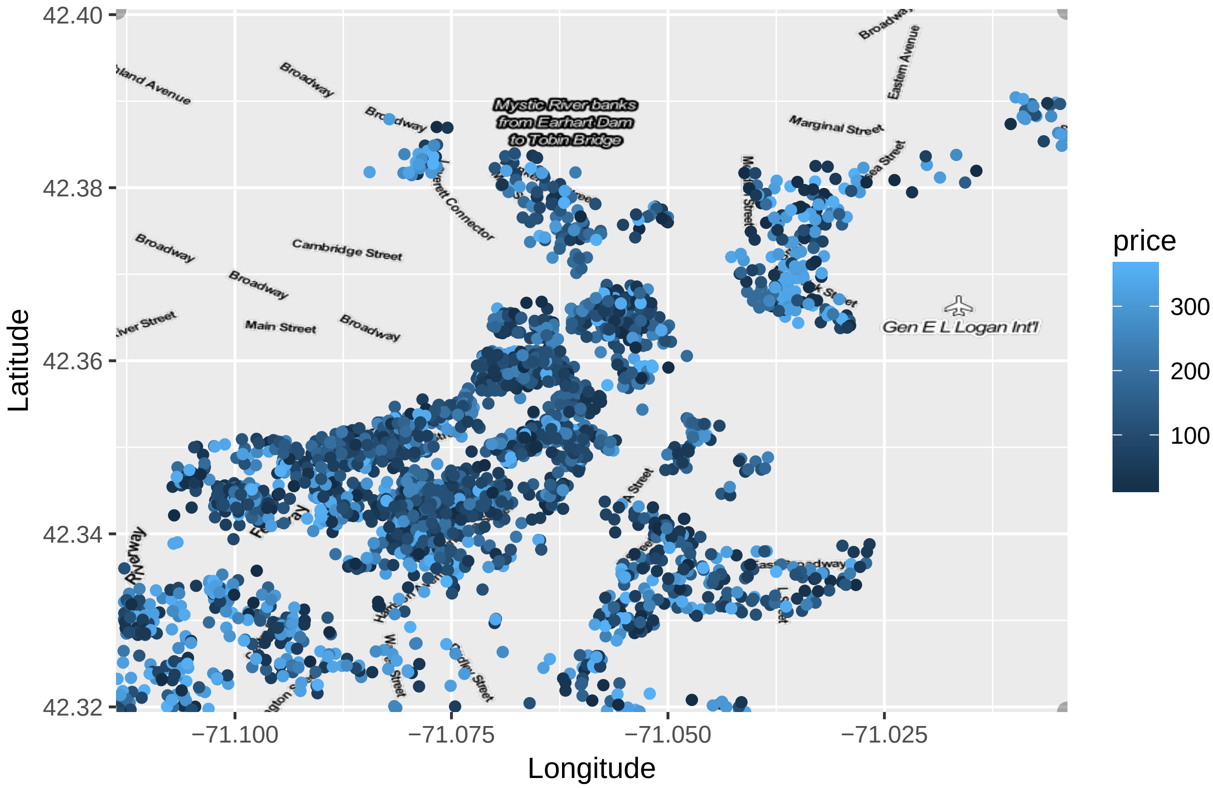



```
grid.arrange(p9, nrow=1,ncol=1)
```




```
p10 + coord_fixed()
```

```
## Warning: Removed 1417 rows containing missing values (geom_point).
```



Observation: Predictors doesn't seem to have linear relationship with response variable (price).

Part 2a: Polynomial Regression

Fit the following models on the training set and compare the R^2 score of the fitted models on the test set:

- Linear regression
- Regression with polynomial basis functions of degree 3 (i.e. basis functions x, x^2, x^3 for each predictor x) for quantitative predictors

- Linear Regression

```
linear.fit = lm(price ~ ., data=listings_train)
summary(linear.fit)

# Calculate R^2 on the test data
rsq(listings_test$price, predict(linear.fit, newdata=listings_test))
```

```
##
## Call:
## lm(formula = price ~ ., data = listings_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -314.13  -59.83    0.55   68.71  283.93
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -5.378e+03  5.118e+03  -1.051  0.29348
```

```
## host_total_listings_count -5.393e-02  7.702e-03  -7.002 2.92e-12 ***
## room_typePrivate room      1.063e+02  3.797e+00  27.996  < 2e-16 ***
## room_typeShared room       1.103e+02  1.425e+01   7.739 1.23e-14 ***
## latitude                  -6.900e+01  6.656e+01  -1.037  0.29993
## longitude                 -1.181e+02  4.953e+01  -2.384  0.01719 *
## bathrooms                  2.516e+01  3.479e+00   7.232 5.58e-13 ***
## bedrooms                   1.067e+01  2.698e+00   3.956 7.73e-05 ***
## beds                       2.076e-01  1.877e+00   0.111  0.91194
## security_deposit          -6.083e-02  9.773e-02  -0.622  0.53369
## cleaning_fee              -1.466e-01  4.183e-02  -3.506  0.00046 ***
## availability_365           3.165e-02  1.161e-02   2.725  0.00645 **
## number_of_reviews         -2.727e-03  3.590e-02  -0.076  0.93946
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 99.74 on 4357 degrees of freedom
## Multiple R-squared:  0.2494, Adjusted R-squared:  0.2473
## F-statistic: 120.7 on 12 and 4357 DF,  p-value: < 2.2e-16
##
## [1] 0.1847913
```

- Regression with polynomial basis functions of degree 3 (i.e. basis functions x, x^2, x^3 for each predictor x)

```r
myvars = colnames(listings_train)

formula_poly = as.formula(paste0("price ~ room_type + ", paste0("poly(",
                                 myvars[c(-2,-8)], ",3)",collapse="+")))

model.poly = lm(formula_poly, data=listings_train)
summary(model.poly)

# Calculate R^2 on the test data
cat(sprintf("Test R^2: %.5f", rsq(listings_test$price, predict(model.poly,
                                               newdata=listings_test)) ))
```

```
##
## Call:
## lm(formula = formula_poly, data = listings_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -278.511  -59.189   -2.221   66.298  276.425
##
## Coefficients:
##                                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)                            132.542      2.151  61.621  < 2e-16
## room_typePrivate room                  106.894      4.211  25.384  < 2e-16
## room_typeShared room                   113.616     14.238   7.980 1.86e-15
## poly(host_total_listings_count, 3)1 -748.699    124.815  -5.998 2.15e-09
## poly(host_total_listings_count, 3)2 -806.043    105.970  -7.606 3.44e-14
## poly(host_total_listings_count, 3)3  352.369    109.636   3.214  0.00132
## poly(latitude, 3)1                    -186.787    115.457  -1.618  0.10578
## poly(latitude, 3)2                     179.749    110.110   1.632  0.10266
## poly(latitude, 3)3                      -5.748    110.811  -0.052  0.95863
```

```
## poly(longitude, 3)1               -308.410    112.815  -2.734  0.00629
## poly(longitude, 3)2                 53.932    114.037   0.473  0.63628
## poly(longitude, 3)3                139.008    103.937   1.337  0.18115
## poly(bathrooms, 3)1                779.976    118.042   6.608 4.38e-11
## poly(bathrooms, 3)2               -307.506    102.056  -3.013  0.00260
## poly(bathrooms, 3)3               -185.190    100.230  -1.848  0.06472
## poly(bedrooms, 3)1                 637.138    171.744   3.710  0.00021
## poly(bedrooms, 3)2                -202.476    112.922  -1.793  0.07303
## poly(bedrooms, 3)3                -116.878    111.770  -1.046  0.29576
## poly(beds, 3)1                    -134.628    169.734  -0.793  0.42772
## poly(beds, 3)2                     108.705    114.428   0.950  0.34217
## poly(beds, 3)3                    -100.963    106.967  -0.944  0.34529
## poly(security_deposit, 3)1        -138.262    113.351  -1.220  0.22262
## poly(security_deposit, 3)2         159.086    104.220   1.526  0.12697
## poly(security_deposit, 3)3         192.427    103.593   1.858  0.06330
## poly(cleaning_fee, 3)1            -320.202    110.843  -2.889  0.00389
## poly(cleaning_fee, 3)2            -135.466    102.740  -1.319  0.18740
## poly(cleaning_fee, 3)3             326.748    111.470   2.931  0.00339
## poly(availability_365, 3)1         211.047    110.210   1.915  0.05556
## poly(availability_365, 3)2         -24.438    105.506  -0.232  0.81684
## poly(availability_365, 3)3        -114.426    105.285  -1.087  0.27718
## poly(number_of_reviews, 3)1         39.982    104.871   0.381  0.70304
## poly(number_of_reviews, 3)2        -84.440    101.734  -0.830  0.40658
## poly(number_of_reviews, 3)3       -135.737    100.295  -1.353  0.17601
##
## (Intercept)                       ***
## room_typePrivate room             ***
## room_typeShared room              ***
## poly(host_total_listings_count, 3)1 ***
## poly(host_total_listings_count, 3)2 ***
## poly(host_total_listings_count, 3)3 **
## poly(latitude, 3)1
## poly(latitude, 3)2
## poly(latitude, 3)3
## poly(longitude, 3)1               **
## poly(longitude, 3)2
## poly(longitude, 3)3
## poly(bathrooms, 3)1               ***
## poly(bathrooms, 3)2               **
## poly(bathrooms, 3)3               .
## poly(bedrooms, 3)1                ***
## poly(bedrooms, 3)2                .
## poly(bedrooms, 3)3
## poly(beds, 3)1
## poly(beds, 3)2
## poly(beds, 3)3
## poly(security_deposit, 3)1
## poly(security_deposit, 3)2
## poly(security_deposit, 3)3        .
## poly(cleaning_fee, 3)1            **
## poly(cleaning_fee, 3)2
## poly(cleaning_fee, 3)3            **
## poly(availability_365, 3)1        .
## poly(availability_365, 3)2
```

```
## poly(availability_365, 3)3
## poly(number_of_reviews, 3)1
## poly(number_of_reviews, 3)2
## poly(number_of_reviews, 3)3
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 98.52 on 4337 degrees of freedom
## Multiple R-squared:  0.271,  Adjusted R-squared:  0.2656
## F-statistic: 50.37 on 32 and 4337 DF,  p-value: < 2.2e-16
##
## Test R^2: 0.23866
```

Allowing for nonlinearity through the addition of cubic terms appears to increase R^2 on the test set compared to linear regression.

Part 2b: Generalized Additive Model (GAM)

Do you see any advantage in fitting an additive regression model to these data compared to the above models?

1. Fit a GAM to the training set, and compare the test R^2 of the fitted model to the above models. You may use a smoothing spline basis function on each predictor, with the same smoothing parameter for each basis function, tuned using cross-validation on the training set.
2. Plot and examine the smooth of each predictor for the fitted GAM, along with plots of upper and lower standard errors on the predictions. What are some useful insights conveyed by these plots, and by the coefficients assigned to each local model?
3. Use a likelihood ratio test to compare GAM with the linear regression model fitted previously. Re-fit a GAM leaving out the predictors `availability_365` and `number_of_reviews`. Using a likelihood ratio test, comment if the new model is preferred to a GAM with all predictors.

Hint: You may use the `gam` function for fitting a GAM, and the function `s` for smoothing spline basis functions. These functions are available in the `gam` library. For k-fold cross-validation, you may adapt the sample code provided in the previous question. The `plot` function can be used to visualize the smooth of each predictor for the fitted GAM (set the attribute `se` to `TRUE` to obtain standard error curves). You may use the `anova` function to compare two models using a likelihood ratio test (with attribute `test='Chi'`).

An advantage in fitting an additive regression model is the ability to tune smoothness, which is not possible with the cubic basis. Compared to the simple linear model, an additive regression model can capture nonlinearities which a simple linear model cannot do.

The main advantage to fitting a GAM is the ability to model the individual contributions of each of the predictors additively, which is more interpretable.

1. Fit a GAM to the training set, and compare the test R^2 of the fitted model to the above models. You may use a smoothing spline basis function on each predictor, with the same smoothing parameter for each basis function, tuned using cross-validation on the training set.

Observation: Rather than fitting a single complex global polynomial model, GAM seeks to fit local models to each predictor. The advantage of this approach is that it is more interpretable as it allows us to examine the effect of each predictor on the response variable.

Fit GAM with spar values 0.1, 0.25, 0.5, 0.75

```r
fit_gam_s = function(spar_val, train, test, disp) {
  # Input:
```

```r
  #   Tuning parameter spar: 'spar_val'
  #   Training dataframe: 'train',
  #   Test dataframe: 'test',
  #   Boolean value to decide what will be return value: 'disp'
  # Output:
  #   if 'disp' is true function returns GAM model else function returns GAM test R^2

  gam_formula = as.formula(paste0("price ~ room_type + ", paste0("s(",
                   myvars[c(-2,-8)], ", spar=",spar_val,")",collapse="+")))

  model.gam <- gam(gam_formula, data=train)

  preds = predict(model.gam, newdata=test)

  gam_trainrsq = rsq(train$price, fitted(model.gam))
  gam_testrsq = rsq(test$price, preds)

  if(disp==TRUE){
    cat(sprintf("GAM with smoothing spline (spar = %.2f): Train R^2: %.3f, Test R^2: %.3f\n",
                spar_val, gam_trainrsq, gam_testrsq))
    return(model.gam)
  }
  else{
    return(gam_testrsq)
  }
}
```

Explore models with different spar values

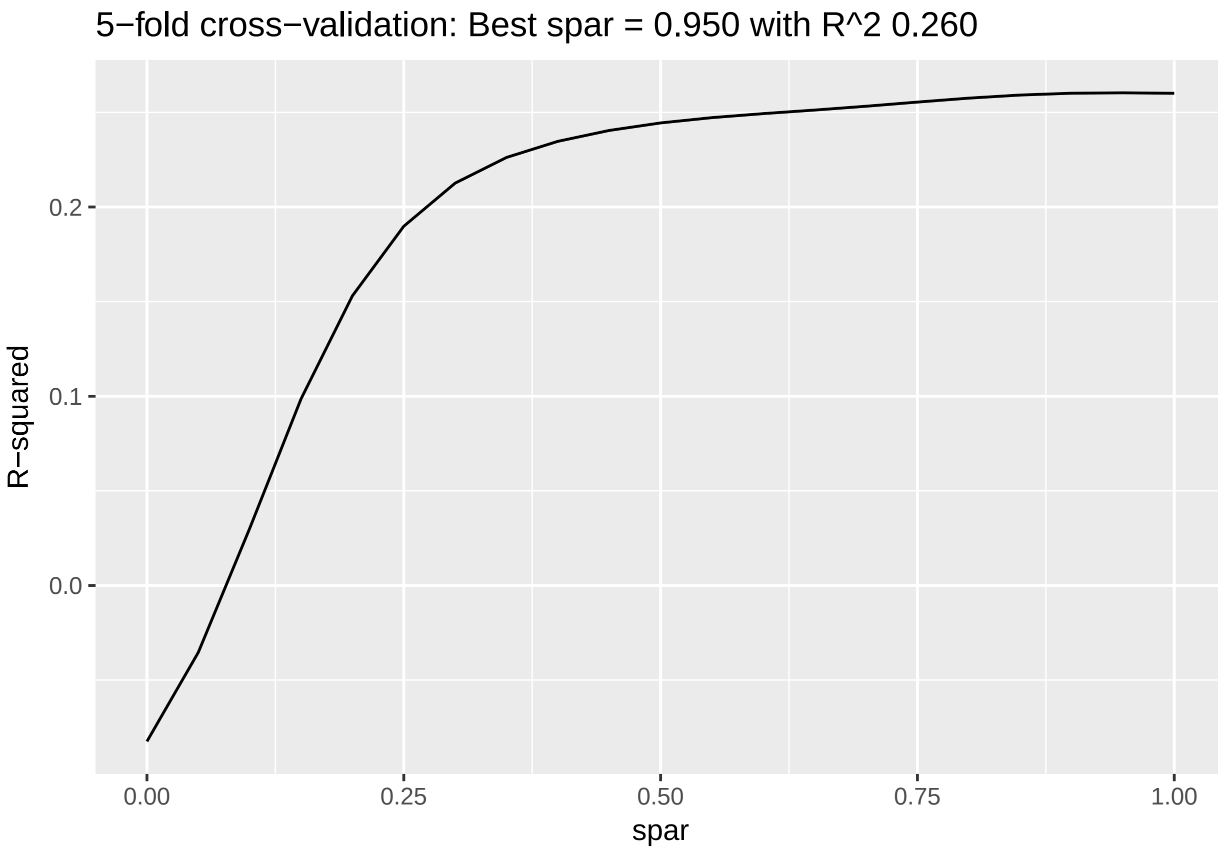
```r
#Lets explore the effect of different spar values, before using cross validation.
invisible(fit_gam_s(0.1, listings_train, listings_test, TRUE))
invisible(fit_gam_s(0.25, listings_train, listings_test, TRUE))
invisible(fit_gam_s(0.5, listings_train, listings_test, TRUE))
invisible(fit_gam_s(0.75, listings_train, listings_test, TRUE))
invisible(fit_gam_s(1, listings_train, listings_test, TRUE))
```

```
## GAM with smoothing spline (spar = 0.10): Train R^2: 0.400, Test R^2: 0.181
## GAM with smoothing spline (spar = 0.25): Train R^2: 0.375, Test R^2: 0.222
## GAM with smoothing spline (spar = 0.50): Train R^2: 0.318, Test R^2: 0.243
## GAM with smoothing spline (spar = 0.75): Train R^2: 0.285, Test R^2: 0.238
## GAM with smoothing spline (spar = 1.00): Train R^2: 0.272, Test R^2: 0.226
```

5-fold cross-validation to find optimal spar value

```r
crossval_gam_s = function(spars_param,k) {
  # Input:
  #   Tuning parameter for smoothing spline in GAM: 'spars_param',
  #   Number of CV folds: 'k'
  # Output:
  #   Average R^2 value: 'rsq_res'

  # sample from 1 to k, nrow times (the number of observations in the data)
  set.seed(109)
```

```r
  listings_train$id <- sample(1:k, nrow(listings_train), replace = TRUE)
  list <- 1:k

  # prediction and testset data frames that we add to with each iteration over
  # the folds
  rsq_res = rep(NA, k)
  dfresult = rep(NA, k)

  for (i in 1:k){
    # remove rows with id i from dataframe to create training set
    # select rows with id i to create test set
    trainingset <- subset(listings_train, id %in% list[-i])
    testset <- subset(listings_train, id %in% c(i))

   # calculate R^2 on test set
    rsq_res[i] = fit_gam_s(spars_param, trainingset, testset, FALSE)
  }

  # Get average R^2
  return(mean(rsq_res))
}

spars = seq(0, 1, 0.05)
res = rep(NA, length(spars))

for (i in 1:length(spars)) {
  res[i] = crossval_gam_s(spars[i],5)  #5-fold cross validation
}

# Find spar with highest CV R^2
best_spar = which(res==max(res))
title_str = sprintf("5-fold cross-validation: Best spar = %.3f with R^2 %.3f",
                    spars[best_spar], res[best_spar])

ggplot() +
  geom_line(aes(x=spars,y=res)) +
  labs(x="spar" , y = "R-squared" ,title=title_str )
```



Re-fit with chosen spar value

```
best_spar_val = spars[best_spar]
gam_testrsq = invisible(fit_gam_s(best_spar_val, listings_train, listings_test, FALSE))
model.gam = invisible(fit_gam_s(best_spar_val, listings_train, listings_test, TRUE))
```

```
## GAM with smoothing spline (spar = 0.95): Train R^2: 0.275, Test R^2: 0.228
```

Observation: GAM yields slightly lower test R^2 compared to the previous regression models with polynomial degree 3, but yields a more interpretable model.

2. Plot and examine the smooth of each predictor for the fitted GAM, along with plots of upper and lower standard errors on the predictions. What are some useful insights conveyed by these plots, and by the coefficients assigned to each local model?

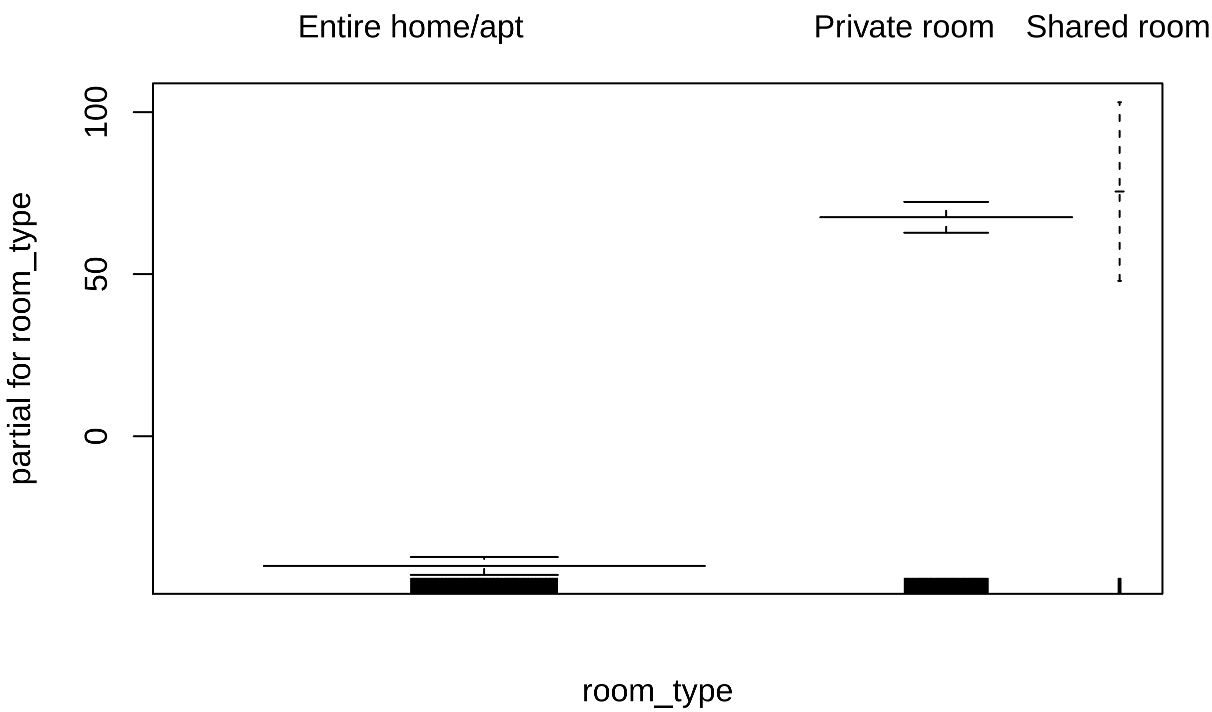
Print coefficients, visualize individual spline models

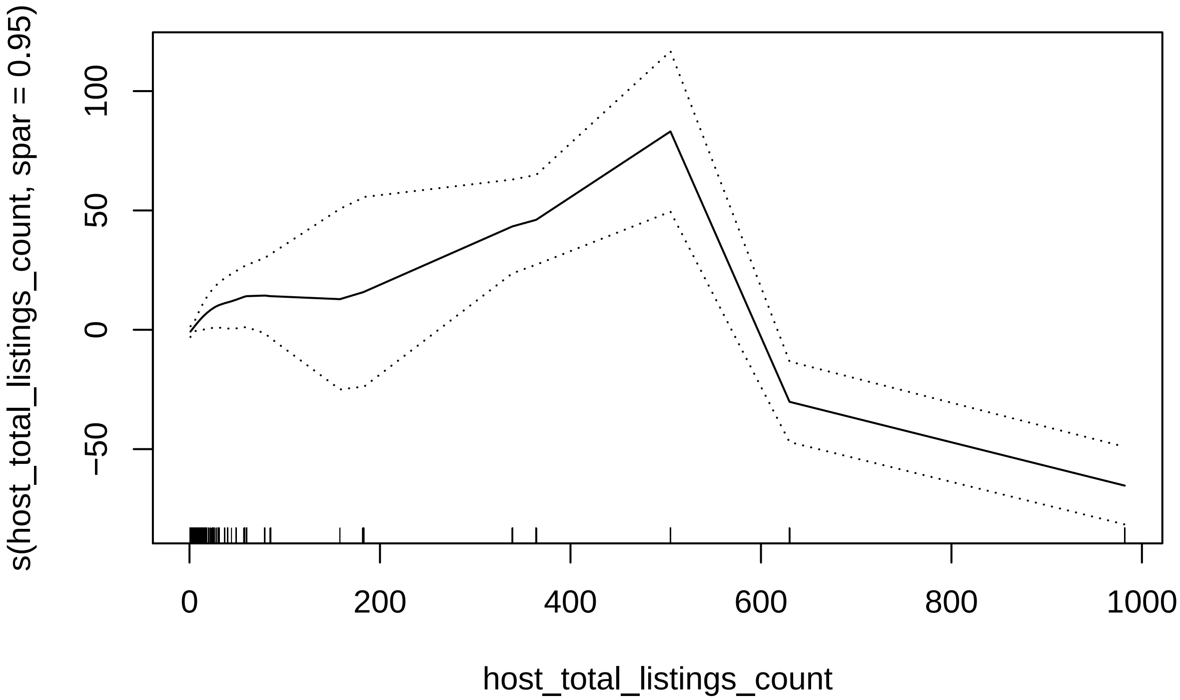
```
coef(model.gam)
```

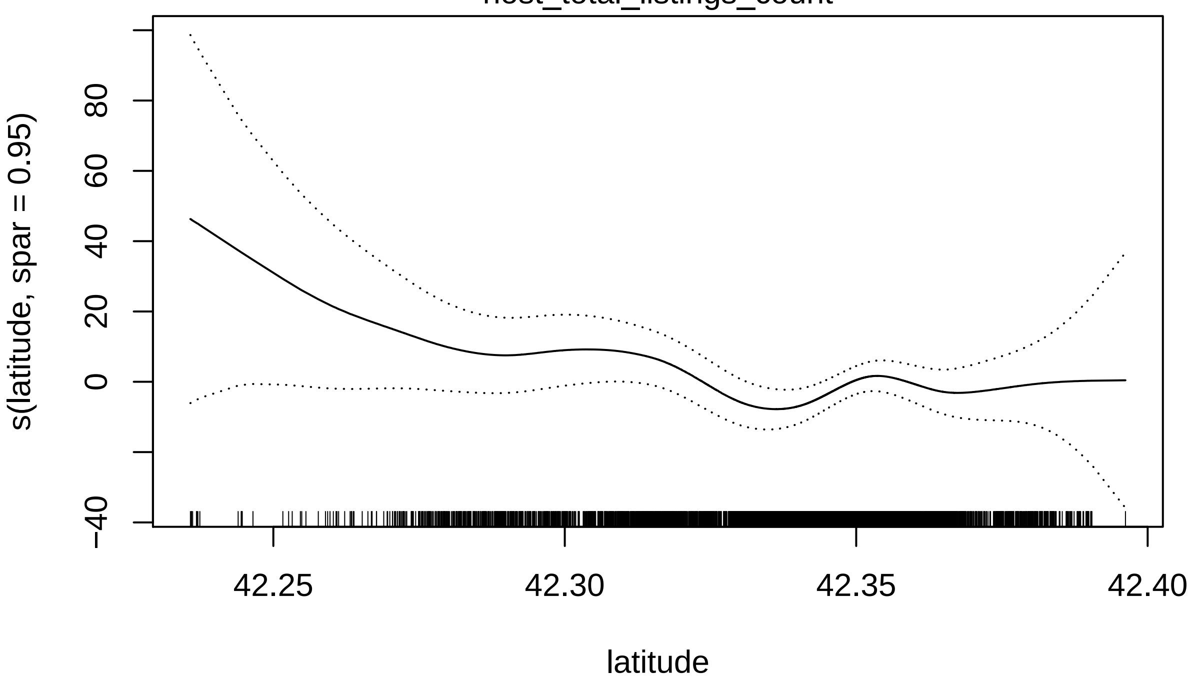
```
##                               (Intercept)
##                             -3.108937e+03
##                     room_typePrivate room
##                              1.075741e+02
##                      room_typeShared room
##                              1.155061e+02
## s(host_total_listings_count, spar = 0.95)
##                             -4.375009e-02
```

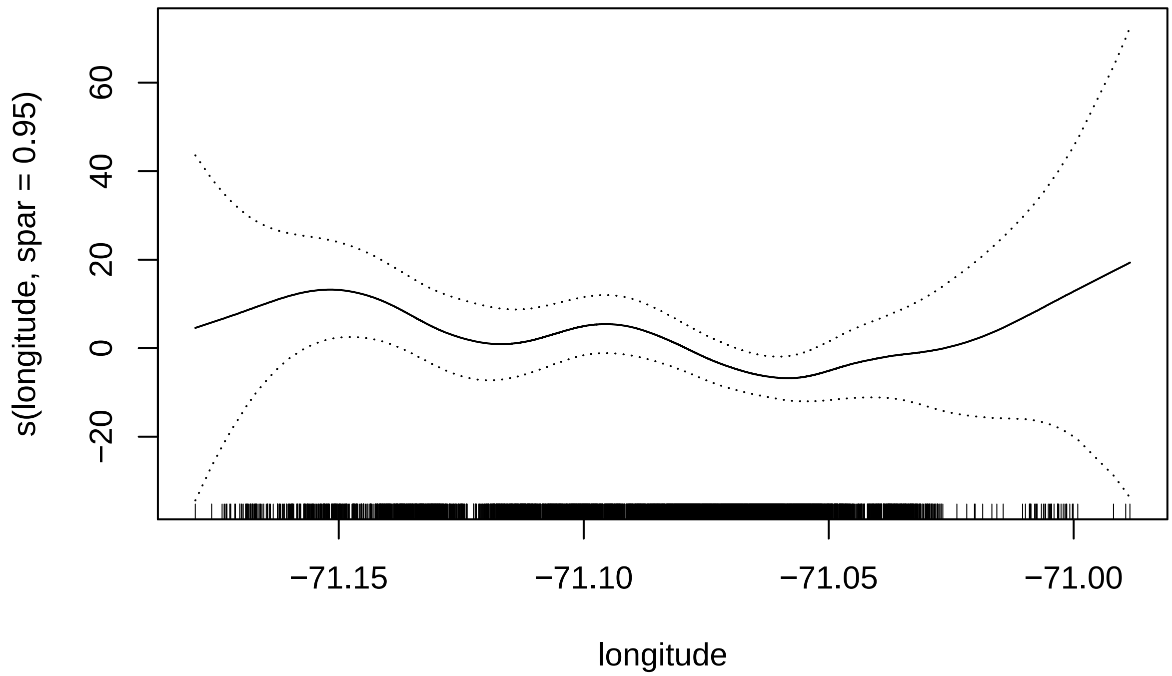
```
##                 s(latitude, spar = 0.95)
##                            -1.444620e+02
##                s(longitude, spar = 0.95)
##                            -1.311419e+02
##                s(bathrooms, spar = 0.95)
##                             2.251665e+01
##                 s(bedrooms, spar = 0.95)
##                             1.077745e+01
##                     s(beds, spar = 0.95)
##                            -7.189615e-01
##         s(security_deposit, spar = 0.95)
##                            -3.821953e-04
##             s(cleaning_fee, spar = 0.95)
##                            -1.097908e-01
##         s(availability_365, spar = 0.95)
##                             1.437819e-02
##        s(number_of_reviews, spar = 0.95)
##                             1.934077e-02
```

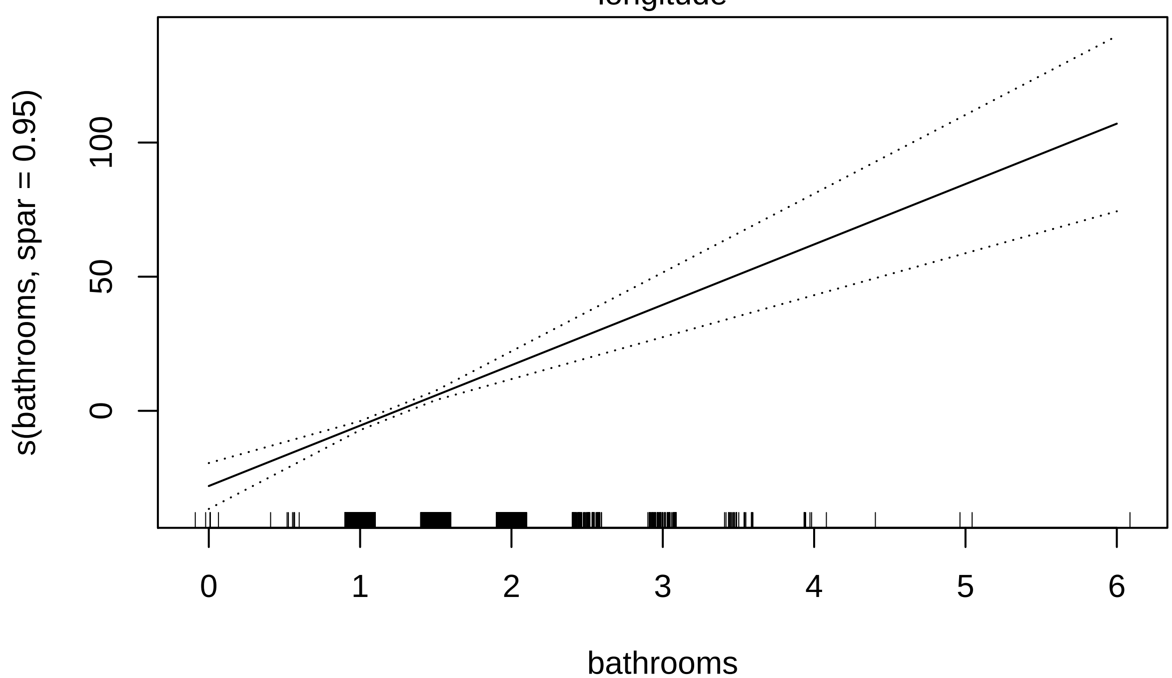
```
plot(model.gam, se=TRUE)
```

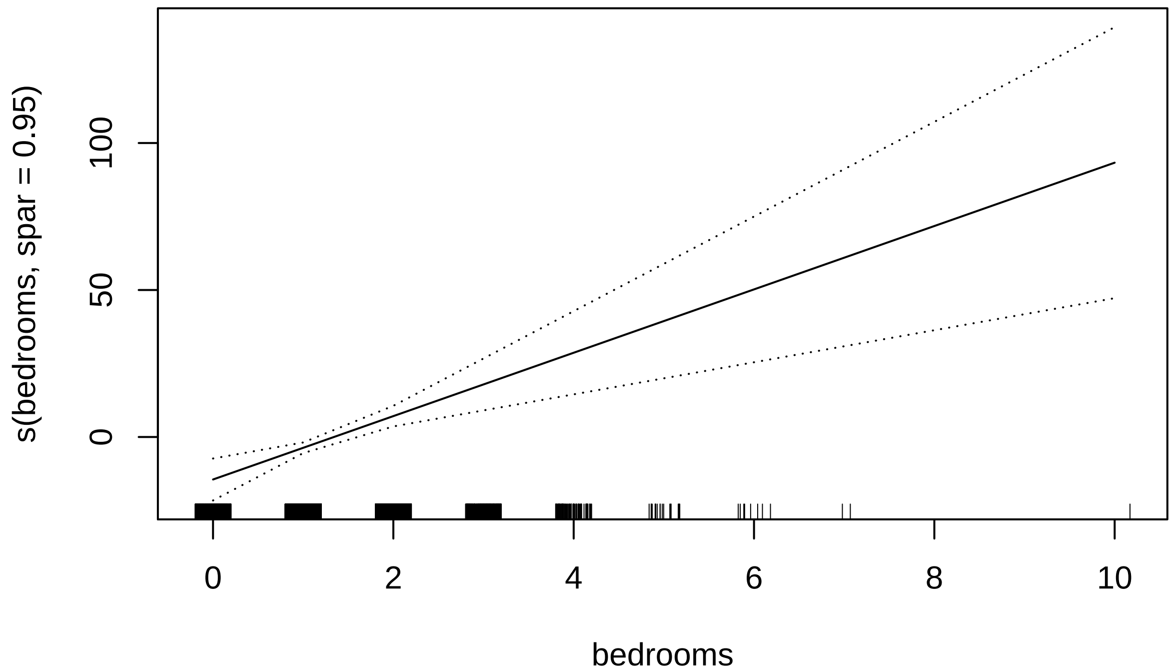


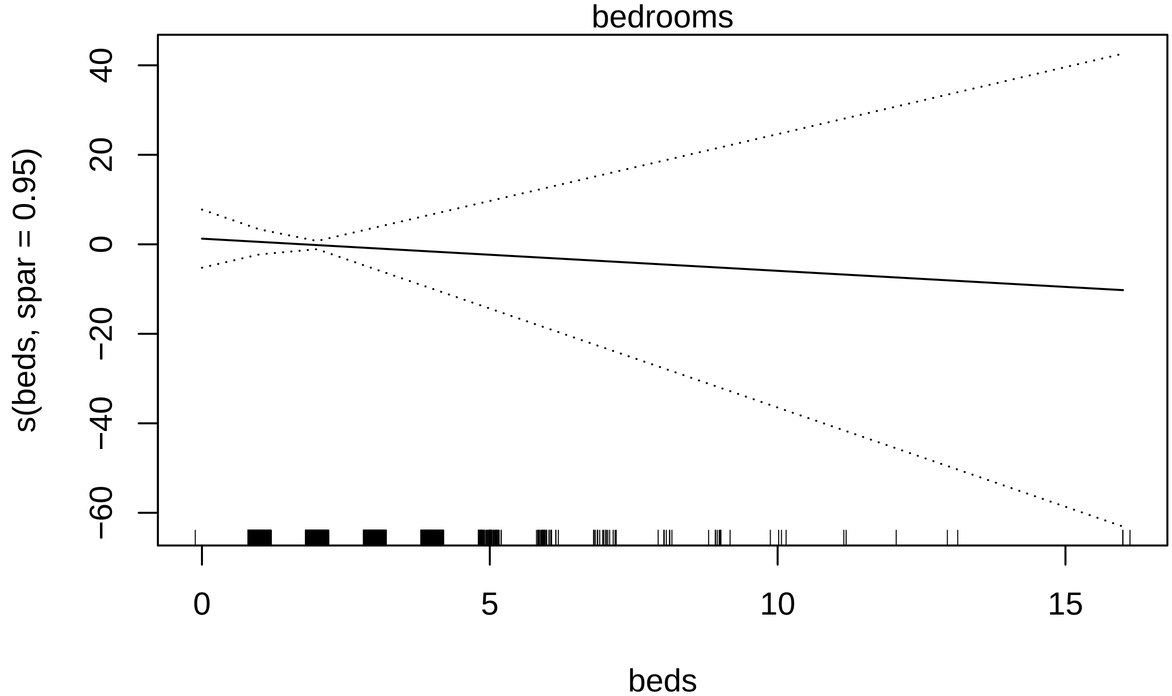

s(host_total_listings_count, spar = 0.95)
host_total_listings_count
−50
0
50
100
0
200
400
600
800
1000

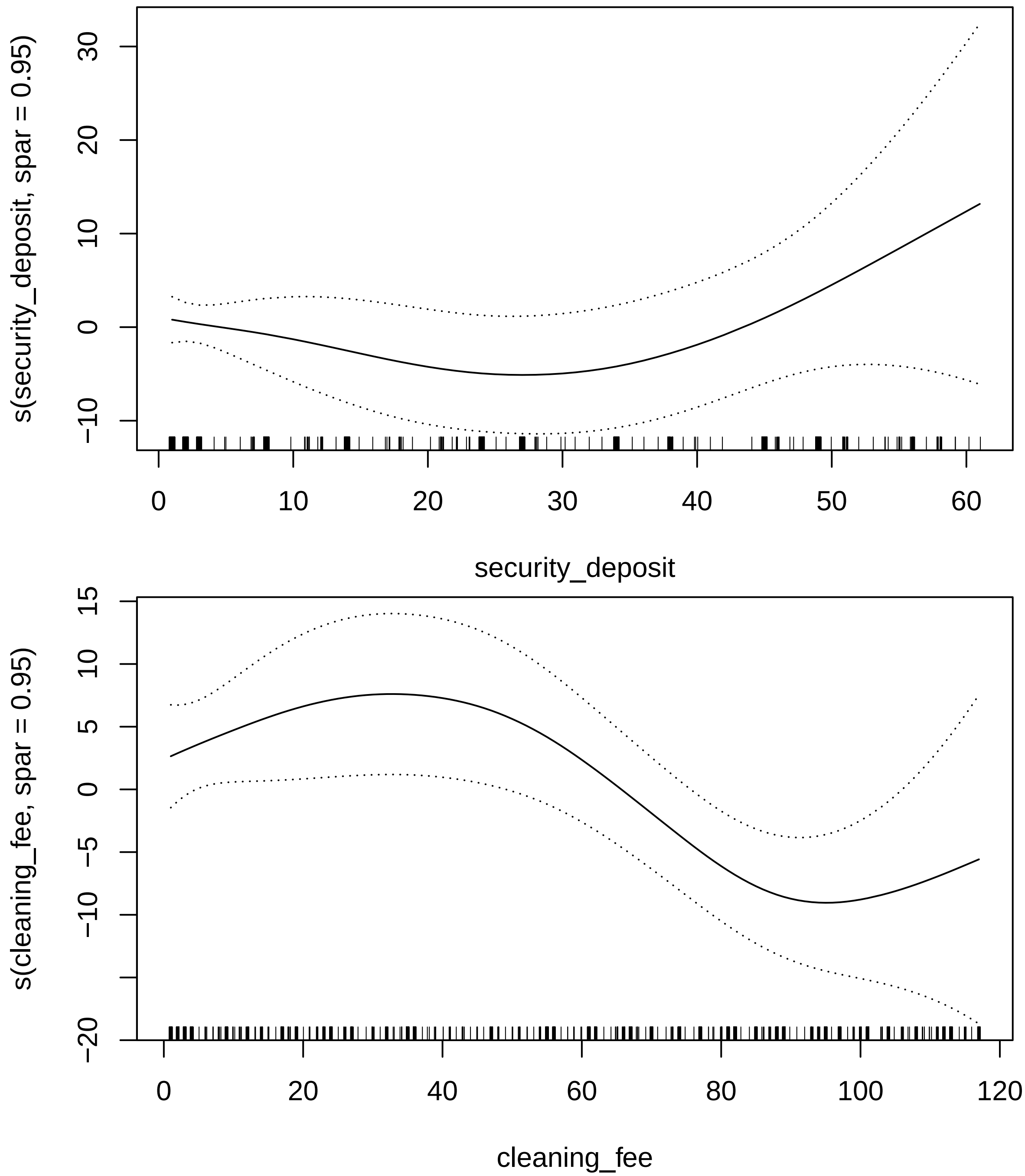

s(latitude, spar = 0.95)
latitude
−40
0
20
40
60
80
42.25
42.30
42.35
42.40


s(longitude, spar = 0.95)
60
40
20
0
−20
−71.15
−71.10
−71.05
−71.00
longitude

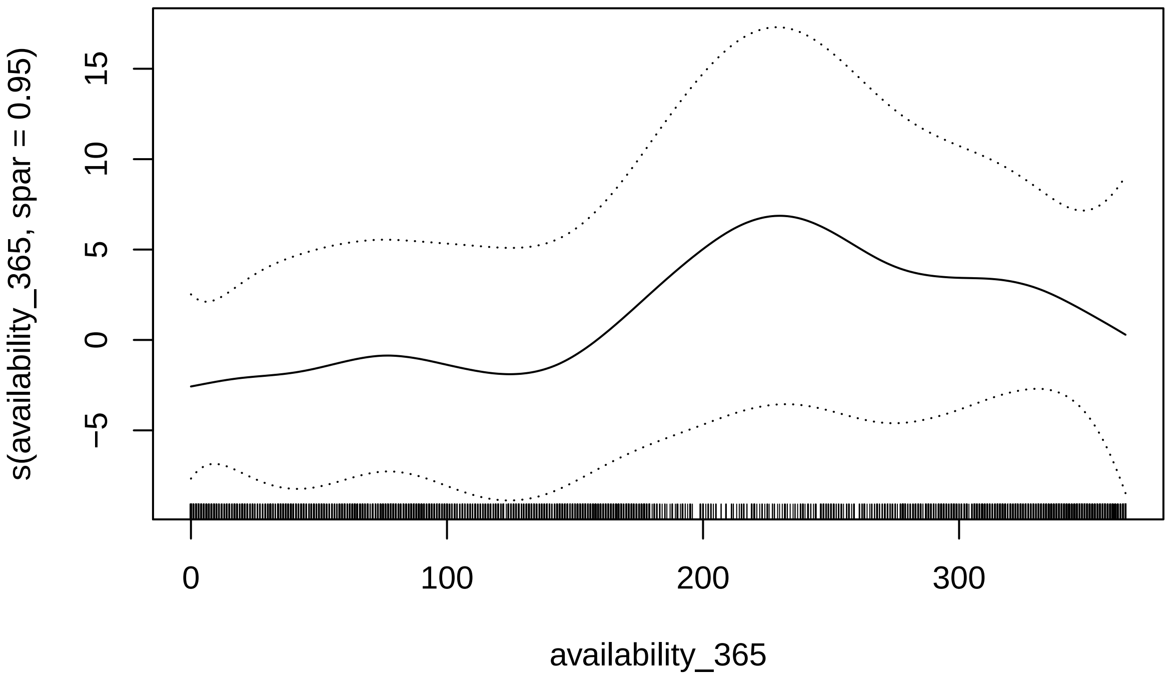

s(bathrooms, spar = 0.95)
100
50
0
0
1
2
3
4
5
6
bathrooms

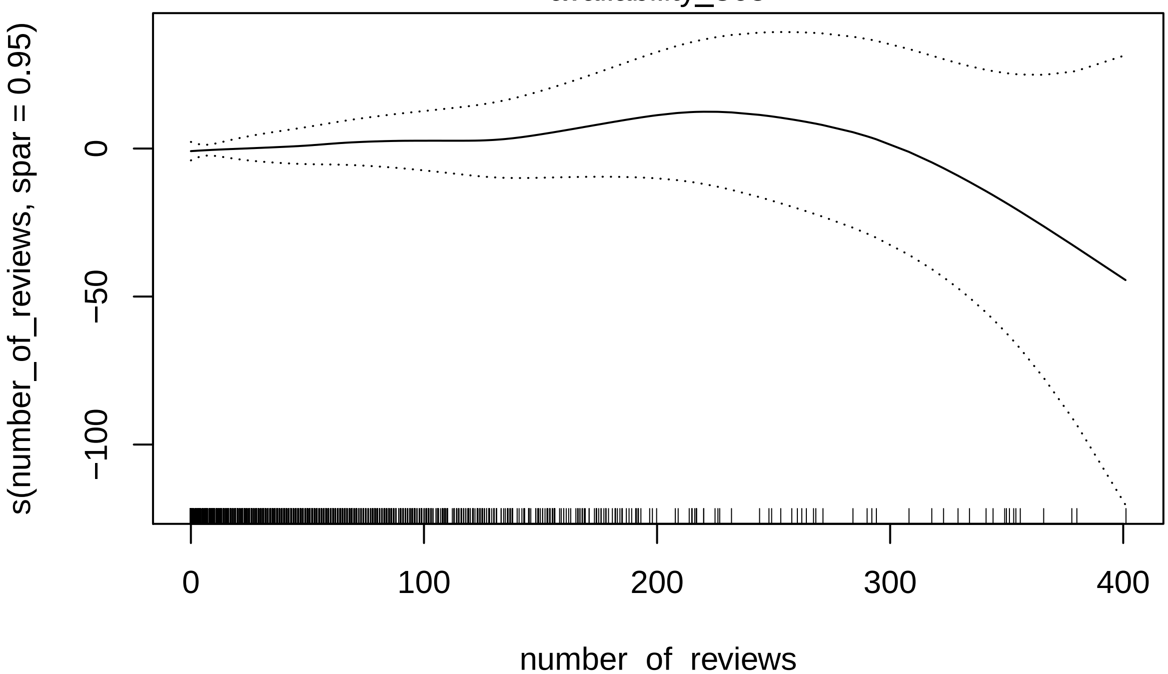

s(bedrooms, spar = 0.95)
0
50
100
0
2
4
6
8
10
bedrooms


s(beds, spar = 0.95)
−60
−40
−20
0
20
40
0
5
10
15
beds



s(security_deposit, spar = 0.95)
30
20
10
0
−10
0
10
20
30
40
50
60
security_deposit
s(cleaning_fee, spar = 0.95)
15
10
5
0
−5
−10
−20
0
20
40
60
80
100
120
cleaning_fee





We see that for some variables such as host_total_listings_count, longitude and number_of_reviews, the fitted GAM captures nonlinearities whereas for some of the other predictors such as bathrooms, the relationship appears to be more linear.

According to the coefficients for the fitted model, it appears that more bedrooms, more bedrooms and shared room or private room (compared to the baseline of entire home/apt) are positively associated with price. security_deposit and cleaning_fee appear negatively associated with price according to the coefficient estimates, but the plots reveal that this relationship is nonlinear; for each of these predictors, after a certain point they are positively associated with price, as one might intuitively expect.

3. Use a likelihood ratio test to compare GAM with the linear regression model fitted previously. Re-fit a GAM leaving out the predictors `availability_365` and `number_of_reviews`. Using a likelihood ratio test, comment if the new model is preferred to a GAM with all predictors.

```r
# Run LRT to compare fitted GAM to linear model
anova(linear.fit, model.gam, test="Chi")
```

```
## Analysis of Variance Table
##
## Model 1: price ~ host_total_listings_count + room_type + latitude + longitude +
##     bathrooms + bedrooms + beds + security_deposit + cleaning_fee +
##     availability_365 + number_of_reviews
## Model 2: price ~ room_type + s(host_total_listings_count, spar = 0.95) +
##     s(latitude, spar = 0.95) + s(longitude, spar = 0.95) + s(bathrooms,
##     spar = 0.95) + s(bedrooms, spar = 0.95) + s(beds, spar = 0.95) +
##     s(security_deposit, spar = 0.95) + s(cleaning_fee, spar = 0.95) +
##     s(availability_365, spar = 0.95) + s(number_of_reviews, spar = 0.95)
##   Res.Df      RSS     Df Sum of Sq  Pr(>Chi)
## 1 4357.0 43342821
## 2 4330.1 41888922 26.896   1453899 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see that our fitted GAM model is a significant improvement over the basic linear regression model.

```r
# Refit GAM, leaving out `availability_365` and `number_of_reviews`

gam_formula = as.formula(paste0("price ~ room_type + ", paste0("s(",
                myvars[c(-2,-8,-11,-12)], ", spar=",best_spar_val,")",collapse="+")))

best.gam.refitted = gam(gam_formula, data = listings_train)
best.gam.train.rsq = rsq(listings_train$price, fitted(best.gam.refitted))

# Run LRT to compare refitted GAM to full GAM
anova(best.gam.refitted, model.gam, test="Chi")
```

```
## Analysis of Deviance Table
##
## Model 1: price ~ room_type + s(host_total_listings_count, spar = 0.95) +
##     s(latitude, spar = 0.95) + s(longitude, spar = 0.95) + s(bathrooms,
##     spar = 0.95) + s(bedrooms, spar = 0.95) + s(beds, spar = 0.95) +
##     s(security_deposit, spar = 0.95) + s(cleaning_fee, spar = 0.95)
## Model 2: price ~ room_type + s(host_total_listings_count, spar = 0.95) +
##     s(latitude, spar = 0.95) + s(longitude, spar = 0.95) + s(bathrooms,
##     spar = 0.95) + s(bedrooms, spar = 0.95) + s(beds, spar = 0.95) +
##     s(security_deposit, spar = 0.95) + s(cleaning_fee, spar = 0.95) +
##     s(availability_365, spar = 0.95) + s(number_of_reviews, spar = 0.95)
##   Resid. Df Resid. Dev     Df Deviance Pr(>Chi)
## 1    4338.9   41968306
## 2    4330.1   41888922 8.8001    79385   0.4937
```

We see that the new model with those 2 predictors removed is indeed preferred.

Part 2c: Putting it All Together

Based on your analysis for problems 1 and 2, what advice would you give a frugal visitor to Boston looking to save some money on an Airbnb rental?

Based on problem 1, it appears that visiting on a weekday and braving one of the colder winter months is the best way to save money. Definitely try to avoid the days around Marathon Monday as much as possible.

Based on problem 2, as one would naturally guess, minimizing the number of rooms in the rental is the way to go. For example, more bedrooms seem to be associated with increased cost more than number of beds, so one way to save with multiple people may be to look for rentals with multiple beds in the same room.

Part 3: Backfitting [AC209b students only]

For the model in Part 2b, rather than using the `gam` function to estimate the component smooths write an iterative function to perform the backfitting algorithm. The backfitting algorithm for fitting a generalized additive model is described on page 25 of the lecture notes.

- Rerun the model in Part 2b using your code, and using the smoothing spline smoother (*Hint*: Use the `smooth.spline` function). Do you obtain the same fitted response values using the same tuning parameter?

Backfitting Algorithm

```
Y = listings_train$price
X1 = listings_train$room_type
X2 = listings_train$host_total_listings_count
X3 = listings_train$latitude
X4 = listings_train$longitude
X5 = listings_train$bathrooms
X6 = listings_train$bedrooms
X7 = listings_train$beds
X8 = listings_train$security_deposit
X9 = listings_train$cleaning_fee

beta1 = 10; beta2 = 10; beta3 = 10; beta4 = 10;
beta5 = 10; beta6 = 10; beta7 = 10; beta8 = 10;
beta9 = 10; beta0 = mean(Y)

f1_X1 = beta1 * as.numeric(X1)
f2_X2 = beta2 * X2
f3_X3 = beta3 * X3
f4_X4 = beta4 * X4
f5_X5 = beta5 * X5
f6_X6 = beta6 * X6
f7_X7 = beta7 * X7
f8_X8 = beta8 * X8
f9_X9 = beta9 * X9

for (i in 1:1000) {
  #keep f1,f2,f3,f4,f5,f6,f7,f8 fixed, fit model for f9
  a = Y - beta0 - f1_X1 - f2_X2 - f3_X3 - f4_X4 - f5_X5 - f6_X6 - f7_X7 - f8_X8
  f9_X9 = fitted(smooth.spline(X9,a,spar=0.95))
  f9_X9[is.na(f9_X9)] = 0
```

```r
  #keep f1,f2,f3,f4,f5,f6,f7,f9 fixed, fit model for f8
  a = Y - beta0 - f1_X1 - f2_X2 - f3_X3 - f4_X4 - f5_X5 - f6_X6 - f7_X7 - f9_X9
  f8_X8 = fitted(smooth.spline(X8,a,spar=0.95))
  f8_X8[is.na(f8_X8)] = 0

  #keep f1,f2,f3,f4,f5,f6,f8,f9 fixed, fit model for f7
  a = Y - beta0 - f1_X1 - f2_X2 - f3_X3 - f4_X4 - f5_X5 - f6_X6 - f8_X8 - f9_X9
  f7_X7 = fitted(smooth.spline(X7,a,spar=0.95))
  f7_X7[is.na(f7_X7)] = 0

  #keep f1,f2,f3,f4,f5,f7,f8,f9 fixed, fit model for f6
  a = Y - beta0 - f1_X1 - f2_X2 - f3_X3 - f4_X4 - f5_X5  - f7_X7 - f8_X8 - f9_X9
  f6_X6 = fitted(smooth.spline(X6,a,spar=0.95))
  f6_X6[is.na(f6_X6)] = 0

  #keep f1,f2,f3,f4,f6,f7,f8,f9 fixed, fit model for f5
  a = Y - beta0 - f1_X1 - f2_X2 - f3_X3 - f4_X4 - f6_X6  - f7_X7 - f8_X8 - f9_X9
  f5_X5 = fitted(smooth.spline(X5, a, spar = 0.95, tol = 0.0001))
  f5_X5[is.na(f5_X5)] = 0

  #keep f1,f2,f3,f5,f6,f7,f8,f9 fixed, fit model for f4
  a = Y - beta0 - f1_X1 - f2_X2 - f3_X3 - f5_X5 - f6_X6  - f7_X7 - f8_X8 - f9_X9
  f4_X4 = fitted(smooth.spline(X4,a,spar=0.95))
  f4_X4[is.na(f4_X4)] = 0

  #keep f1,f2,f4,f5,f6,f7,f8,f9 fixed, fit model for f3
  a = Y - beta0 - f1_X1 - f2_X2 - f4_X4 - f5_X5 - f6_X6  - f7_X7 - f8_X8 - f9_X9
  f3_X3 = fitted(smooth.spline(X3,a,spar=0.95))
  f3_X3[is.na(f3_X3)] = 0

  #keep f1,f3,f4,f5,f6,f7,f8,f9 fixed, fit model for f2
  a = Y - beta0 - f1_X1 - f3_X3 - f4_X4 - f5_X5 - f6_X6  - f7_X7 - f8_X8 - f9_X9
  f2_X2 = fitted(smooth.spline(X2,a,spar=0.95))
  f2_X2[is.na(f2_X2)] = 0

  #keep f2,f3,f4,f5,f6,f7,f8,f9 fixed, fit model for f1
  a = Y - beta0 - f2_X2 - f3_X3 - f4_X4 - f5_X5 - f6_X6  - f7_X7 - f8_X8 - f9_X9
  f1_X1 = fitted(lm(a ~ X1))

  beta0 = mean(Y - f1_X1 - f2_X2 - f3_X3 - f4_X4 - f5_X5 - f6_X6 - f7_X7 - f8_X8 - f9_X9)
}

tmp = beta0 + f1_X1 + f2_X2 + f3_X3 + f4_X4 + f5_X5 + f6_X6 + f7_X7 + f8_X8 +
  f9_X9

cat("Best GAM training R^2: ", best.gam.train.rsq, "\n")
```

```
## Best GAM training R^2:  0.2732181
```

```r
cat("Backfitting GAM training R^2: ", rsq(listings_train$price,tmp), "\n")
```

```
## Backfitting GAM training R^2:  0.2736757
```

The R^2 from Problem 2b and the backfitting procedure results are quite close.