Homework 1: Smoothers and Generalized Additive Models

Harvard CS 109B, Spring 2018

Jan 2018

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.

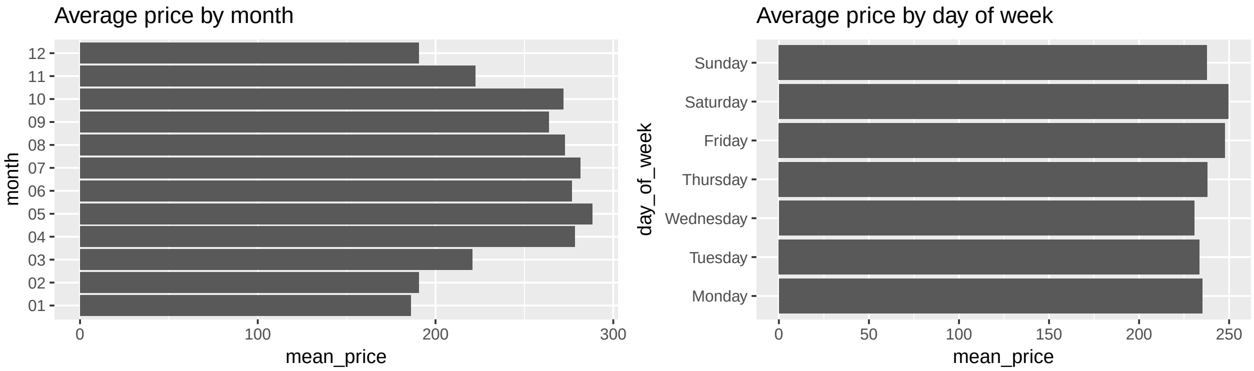
Exploratory Analysis

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()`.

```
# import libraries
library(ggplot2)
library(dplyr)
library(tidyr)
library(gridExtra)
library(splines)
```

```
# read in data
df_train <- read.csv("data/calendar_train.csv")
df_test <- read.csv("data/calendar_test.csv")

# convert dates into date object
df_train$date <- as.Date(df_train$date,"%m/%d/%y")
df_test$date <- as.Date(df_test$date, "%m/%d/%y")

# exclude observations that are missing price
df_train <- df_train[!is.na(df_train$price),]
df_test <- df_test[!is.na(df_test$price), ]

# group df_train by month or by day of week and calculate the mean price
mean_price_by_month <- df_train %>% group_by(month = factor(format(df_train$date, '%m'))) %>%
  summarise(mean_price = mean(price))
mean_price_by_day_of_week <- df_train %>% group_by(day_of_week = factor(weekdays(date))) %>%
  summarise(mean_price = mean(price))

# relevel day of week from Monday to Sunday
mean_price_by_day_of_week$day_of_week <- factor(mean_price_by_day_of_week$day_of_week,
                                   levels = c("Monday", "Tuesday", "Wednesday", "Thursday",
                                                "Friday", "Saturday", "Sunday"))

# plot
price_by_month.plot <- ggplot(data=mean_price_by_month, aes(x=month, y=mean_price)) +
  geom_bar(stat="identity") + ggtitle("Average price by month") + coord_flip()
price_by_day_of_week.plot <- ggplot(data=mean_price_by_day_of_week, aes(x=day_of_week, y=mean_price)) +
  geom_bar(stat="identity") + ggtitle("Average price by day of week") + coord_flip()
grid.arrange(price_by_month.plot, price_by_day_of_week.plot, ncol=2)
```



ANSWER (Problem 1 - EDA):
Average price by month appears to peak between April and October and falls between November and March. Since Boston winter is harsh, it makes sense that people would be less inclined to visit in winter months.

For average price by day of week, average prices on Friday and Saturday are slightly higher than those on the other days. It also makes sense because we can expect more bookings over weekends than weekdays.

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.

```
# convert the `date` column into a numerical variable
first_day <- min(df_train$date)
df_train$days_since <- as.integer(difftime(df_train$date, first_day, units = "days"))
df_test$days_since <- as.integer(difftime(df_test$date, first_day, units = "days"))

# group dfs by date
df_train_by_date <- df_train %>% group_by(days_since = df_train$days_since) %>%
  summarise(price = mean(price))
df_test_by_date <- df_test %>% group_by(days_since = df_test$days_since) %>%
  summarise(price = mean(price))
```

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.

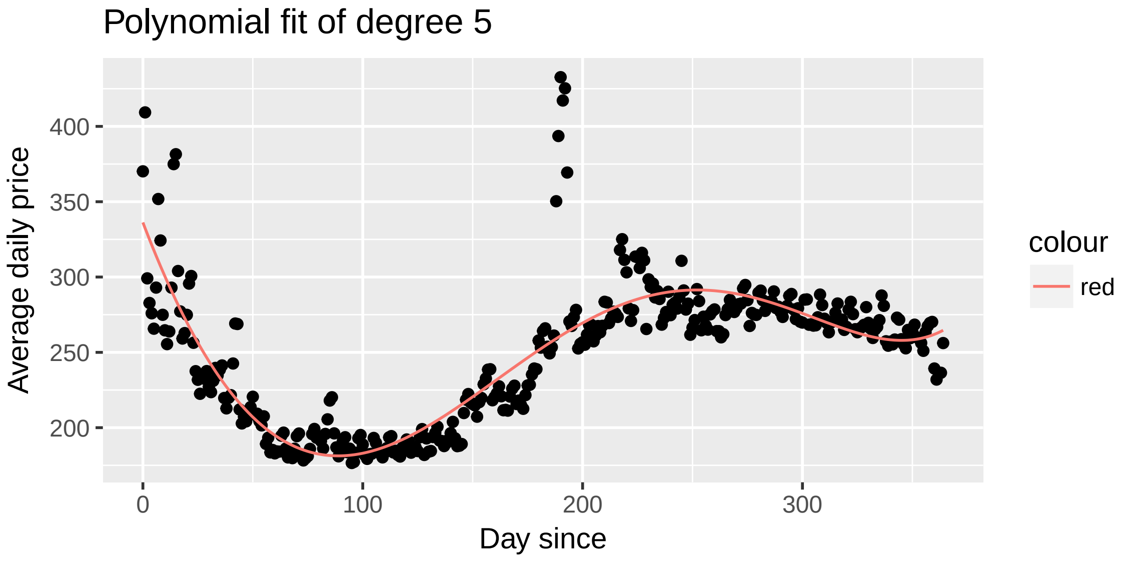
Is there a reason you would prefer one of these methods over the other?

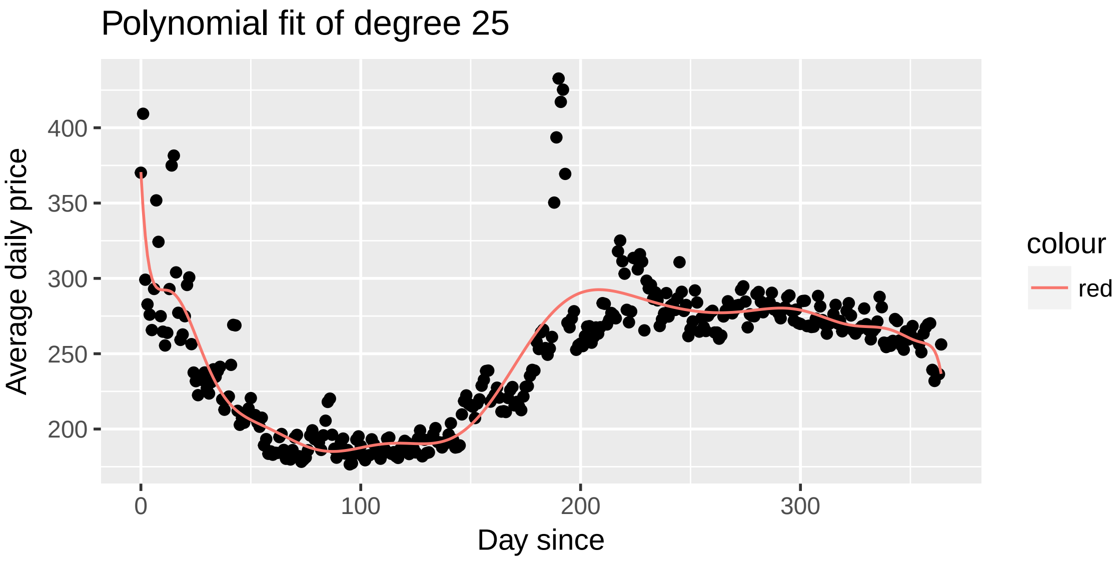
Hints: - You may use the function poly to generate polynomial basis functions (use the attribute degree to set the degree of the polynomial), the function bs for B-spline basis functions (use the attribute knots to specify the knots), and the function ns for natural cubic spline basis functions (use 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.

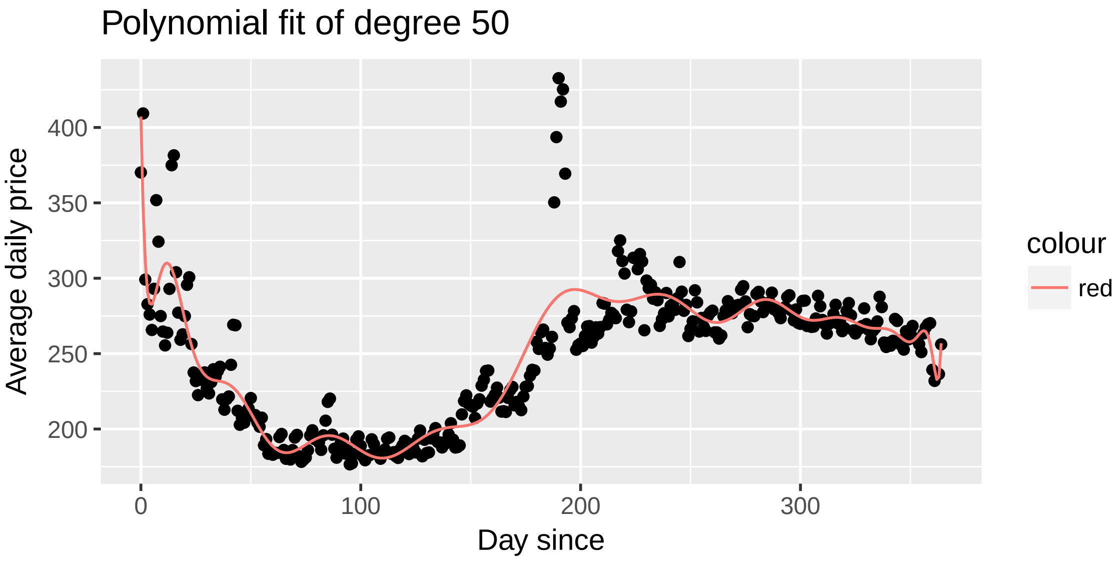
- For smoothing splines, R provides an internal cross-validation feature: this can be used by leaving the spar attribute in smooth.spline unspecified; you may set the cv attribute to choose between leave-one-out cross-validation and generalized cross-validation. For the other models, you will have to 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:

```r
# Function to compute R^2 for observed and predicted responses
rsq = function(y, predict) {
  tss = sum((y - mean(y))^2)
  rss = sum((y-predict)^2)
  r_squared = 1 - rss/tss

  return(r_squared)
}

# Function for k-fold cross-validation to tune span parameter in loess
cv_loess = function(train, param_val, k) {
  # Input:
  #   Training data frame: 'train',
  #   Vector of span 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.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)
}

# Function for k-fold cross-validation to tune df parameter in ns (natural spline)
cv_ns = function(train, param_val, k){
  # Input:
  #   Training data frame: 'train',
  #   Vector of df 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.ns = lm(price ~ ns(days_since, df = param_val[i]),
                    data = train[folds!=j, ])

      # Make prediction on fold 'j'
      pred = predict(model.ns, 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)
}
```

```
# 1. Regression with different basis functions
# * Simple polynomials with degrees 5, 25, and 50
# Function to create polynomial fits, plot the fit, and return train.r2 and test.r2
polynomial_basis = function(degree, train, test){
  # Input:
  #   Training data frame: 'train',
  #   Test data frame: 'test',
  #   Degree parameter value: 'degree',
  # Output:
  #   Print data + regression line to console
  #   Return a vector of train.r2, test.r2
  model <- lm(price ~ poly(days_since, degree = degree, raw = TRUE), data = train)
  pred.train <- predict(model, newdata = train)
  pred.test <- predict(model, newdata = test)

  print(ggplot(train, aes(x = days_since, y = price)) + geom_point() +
          geom_line(aes(y = pred.train, color="red")) +
          ggtitle(paste0("Polynomial fit of degree ", degree)) +
          xlab("Day since") + ylab("Average daily price"))

  return(c(train.r2 = rsq(train$price, pred.train),
    test.r2 = rsq(test$price, pred.test)))
}

degrees = c(5, 25, 50)
polynomial.models <- sapply(degrees, polynomial_basis,
                                  train = df_train_by_date, test = df_test_by_date, simplify = FALSE)
```





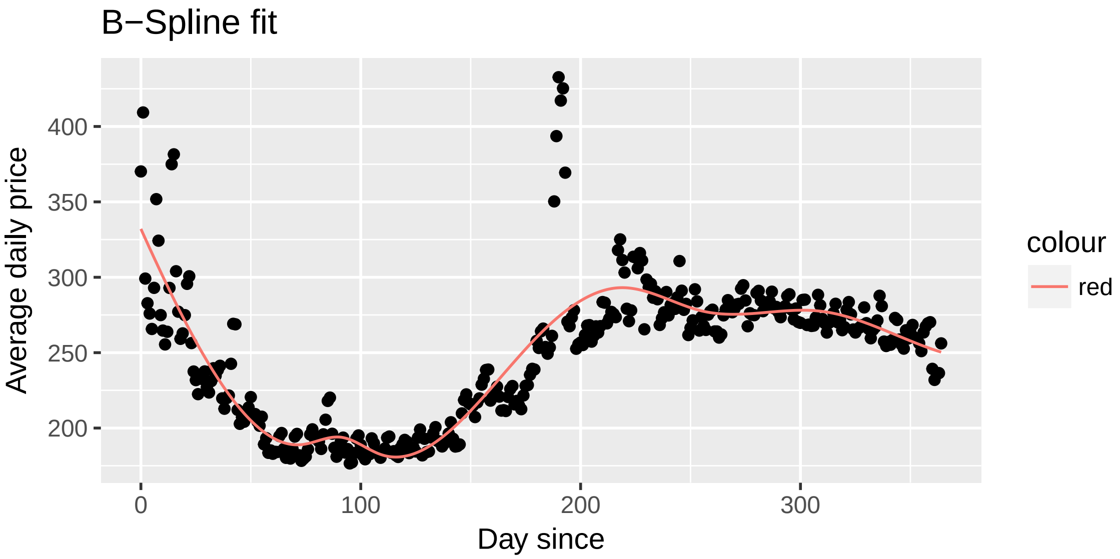


```
polynomial.performance <- as.data.frame(do.call(rbind, polynomial.models))
polynomial.performance$degrees <- degrees
polynomial.performance
```

```
##    train.r2   test.r2 degrees
## 1 0.7168986 0.6850170       5
## 2 0.7630495 0.7256176      25
## 3 0.7862399 0.7403374      50
```

```
# * Cubic B-splines with the knots chosen by visual inspection of the data.
model_bspline <- lm(price ~ bs(days_since,
                knots = quantile(days_since, c(.20, .25, .30, .50, .60, .70, .85))),
                data = df_train_by_date)

pred.bspline.train <-  predict(model_bspline)
pred.bspline.test <- predict(model_bspline, newdata = df_test_by_date)
train.r2.bspline <- rsq(df_train_by_date$price, pred.bspline.train)
test.r2.bspline <- rsq(df_test_by_date$price, pred.bspline.test)
```

```
ggplot(df_train_by_date, aes(x = days_since, y = price)) + geom_point() +
          geom_line(aes(y = pred.bspline.train, color="red")) +
          ggtitle(paste0("B-Spline fit")) +
          xlab("Day since") + ylab("Average daily price")
```



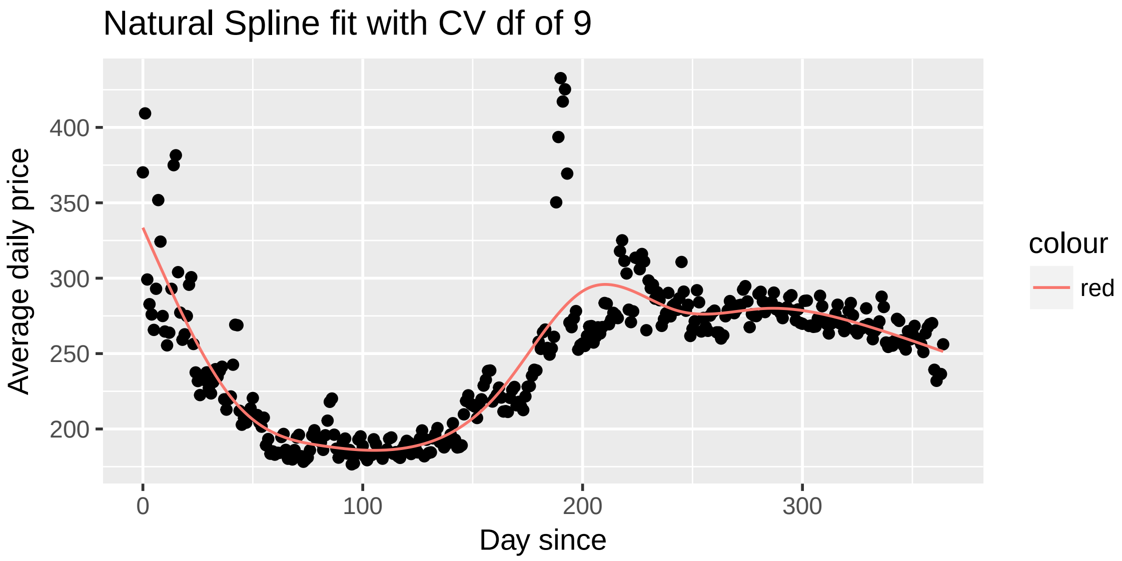
```
train.r2.bspline
```

```
## [1] 0.7516088
```

```
test.r2.bspline
```

```
## [1] 0.718528
```

```
# * Natural cubic splines with the degree of freedom chosen by cross-validation on the training set
dfs <- 2:10
cv_ns_scores <- cv_ns(df_train_by_date, dfs, 5)
opt_df <- dfs[which.max(cv_ns_scores)]

# re-fit NS with optimal degree
model.ns_opt = lm(price ~ ns(days_since, df = opt_df),
                    data = df_train_by_date)

pred.ns_opt.train <- predict(model.ns_opt, df_train_by_date)
pred.ns_opt.test <- predict(model.ns_opt, df_test_by_date)

train.r2.ns_opt <- rsq(df_train_by_date$price, pred.ns_opt.train)
test.r2.ns_opt <- rsq(df_test_by_date$price, pred.ns_opt.test)

ggplot(df_train_by_date, aes(x = days_since, y = price)) + geom_point() +
          geom_line(aes(y = pred.ns_opt.train, color="red")) +
          ggtitle(paste0("Natural Spline fit with CV df of ", opt_df)) +
          xlab("Day since") + ylab("Average daily price")
```



```
train.r2.ns_opt
```

```
## [1] 0.7553177
```

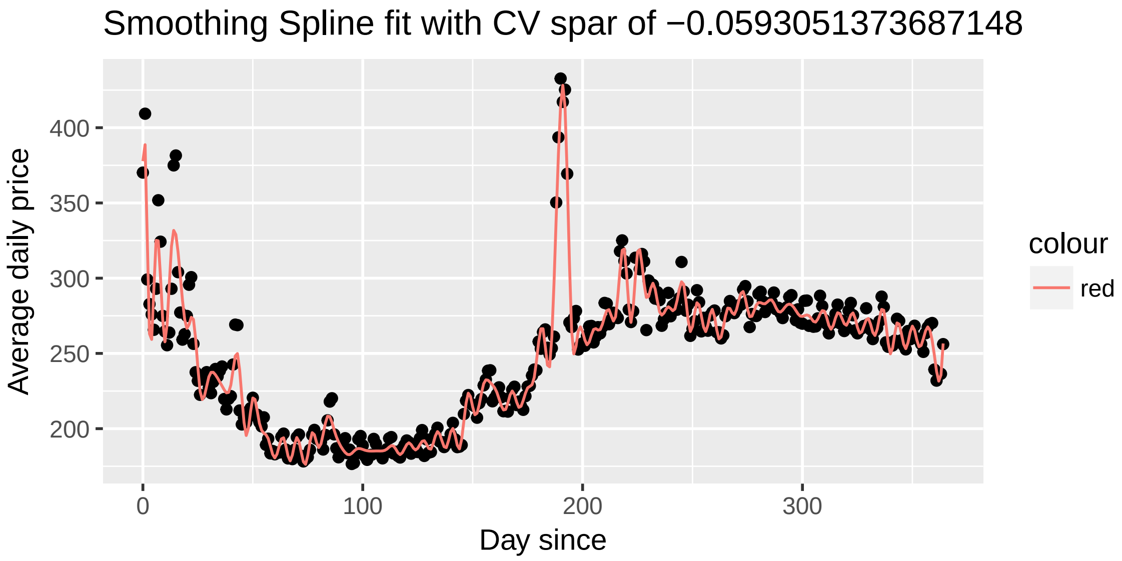
```
test.r2.ns_opt
```

```
## [1] 0.7219328
```

ANSWER (Question 1a - Part 1 Regression models with different basis functions):
Since both the flexibility and complexity of models in this part of the question increase as the tuning parameter increases, we would expect training R^2 to increase while there exists an optimal test R^2 that peaks at some value and declines after such value due to overfitting. We discuss the details of each basis as follows. In general, we observed that more complex models tend to perform better. This perhaps is because the data has many small cyclic patterns on a weekly basis as observed in our EDA, so more complex models can capture/model such local behavior better than less complex ones.

- Polynomial basis - As the polynomial degree increases from 5 to 50, the fit improves with the highest test $R^2 = 0.7403374$ at degree = 50 with a corresponding training $R^2 = 0.7862399$.
- Cubic B-splines - We chose 7 knots at the .20, .25, .30, .50, .60, .70, .85 quantiles of the predictor variable (day_since) by visual inspection based on where the mean of data changes rapidly. The training $R^2 = 0.7516088$, and the test $R^2 = 0.718528$.
- Natrual cubic splines - We used 5-fold cross-validation to choose an optimal degree of freedom (‘df‘) from 2 to 10. The result of cross-validation was an optimal degree of freedom = 9. We refitted the whole training set using df = 9 and obtained a training $R^2 = 0.7553177$ and test $R^2 = 0.7219328$.

```
# 2. Smoothing spline model with the smoothness parameter chosen by cross-validation on the training set
model.ss <- smooth.spline(x = df_train_by_date$days_since, y = df_train_by_date$price)
pred.ss.train <- predict(model.ss, df_train_by_date$days_since)$y
pred.ss.test <- predict(model.ss, df_test_by_date$days_since)$y

ggplot(df_train_by_date, aes(x = days_since, y = price)) + geom_point() +
          geom_line(aes(y = pred.ss.train, color="red")) +
          ggtitle(paste0("Smoothing Spline fit with CV spar of ", model.ss$spar)) +
          xlab("Day since") + ylab("Average daily price")
```



```
train.r2.ss <- rsq(df_train_by_date$price, pred.ss.train)
test.r2.ss <- rsq(df_test_by_date$price, pred.ss.test)
train.r2.ss
```

```
## [1] 0.9575765
```

```
test.r2.ss
```

```
## [1] 0.9156352
```

ANSWER (Question 1a - Part 2 Smoothing splines):
For smoothing splines, model flexibility and complexity increases as the tuning paramter ‘spar‘ decreases. Therefore, we would expect training R^2 to increase as ‘spar‘ decreases while there exists an optimal test R^2 that peaks at some value and declines after such value due to overfitting. Again, we observed that more complex models tend to perform better.

We used the built-in ‘generalized’ cross-validation functionality of ‘smooth.spline‘ to tune the ‘spar‘ parameter. The resulting optimal spar value = -0.0593051 with a training R^2 = 0.9575765 and a test R^2 = 0.9156352

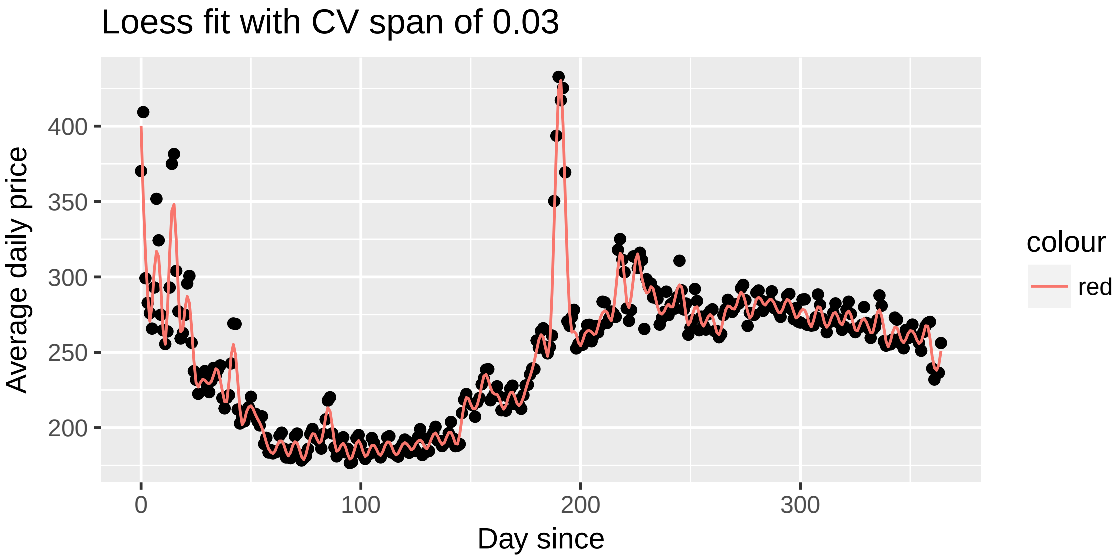
```
# 3. Locally-weighted regression model with the span parameter chosen
# by cross-validation on the training set
spans <- seq(0.01, 0.1, by = 0.01)
cv_loess_scores <- cv_loess(df_train_by_date, spans, 5)
opt_span <- spans[which.max(cv_loess_scores)]
cv_loess_scores
```

```
##  [1] -25.8422396   0.8811015   0.9037324   0.8913800   0.8607610
##  [6]   0.8376741   0.8273911   0.8228646   0.8145923   0.8058578
```

```
opt_span
```

```
## [1] 0.03
```

```
# re-fit loess with optimal span
model.loess_opt = loess(price ~ days_since, span = opt_span,
                          data = df_train_by_date,
                          control = loess.control(surface="direct"))

pred.loess_opt.train <- predict(model.loess_opt, df_train_by_date)
pred.loess_opt.test <- predict(model.loess_opt, df_test_by_date)

train.r2.loess_opt <- rsq(df_train_by_date$price, pred.loess_opt.train)
test.r2.loess_opt <- rsq(df_test_by_date$price, pred.loess_opt.test)

ggplot(df_train_by_date, aes(x = days_since, y = price)) + geom_point() +
          geom_line(aes(y = pred.loess_opt.train, color="red")) +
          ggtitle(paste0("Loess fit with CV span of ", opt_span)) +
          xlab("Day since") + ylab("Average daily price")
```



```
train.r2.loess_opt
```

```
## [1] 0.9705876
```

```
test.r2.loess_opt
```

```
## [1] 0.9264111
```

ANSWER (Question 1a - Part 3 Loess):
For loess, model flexibility and complexity increases as the tuning paramter ‘span‘ decreases. Therefore, we would expect training R^2 to increase as ‘span‘ decreases while there exists an optimal test R^2 that peaks at some value and declines after such value due to overfitting. Again, we observed that more complex models tend to perform better.

We used 5-fold cross-validation to choose an optimal span from 0.01 to 1, with which we refitted the training set. The result of cross-validation was an optimal span = 0.03, and the refitted model gave a training $R^2 = 0.9705876$ and a test $R^2 = 0.9264111$.

ANSWER (Question - Reason to prefer smoothing spline over others):
We would prefer smoothing spline to the other methods because it gives the second highest test R^2. While the lowess model has the highest test R^2, we would not prefer it because its optimal span parameter (0.03) chosen by cross validation is too small to fit the data points within the local neighborhood such that R uses a pseudoinverse, which results in warnings.

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)

```r
# indicate whether a day is a weekday or a weekend
df_train$is_weekend <- format(df_train$date, '%u') %in% c(6, 7)
df_test$is_weekend <- format(df_test$date, '%u') %in% c(6,7)

# split training and testing datasets into weekend or weekday
df_train_weekend <- df_train[df_train$is_weekend, ]
df_train_weekday <- df_train[!df_train$is_weekend, ]

df_test_weekend <- df_test[df_test$is_weekend, ]
df_test_weekday <- df_test[!df_test$is_weekend, ]

# group by each date
df_train_weekend_by_date <- df_train_weekend %>%
  group_by(days_since = df_train_weekend$days_since) %>%
  summarise(price = mean(price))
df_train_weekday_by_date <- df_train_weekday %>%
  group_by(days_since = df_train_weekday$days_since) %>%
  summarise(price = mean(price))
df_train_by_date_combined <- rbind(df_train_weekend_by_date, df_train_weekday_by_date)

df_test_weekend_by_date <- df_test_weekend %>%
  group_by(days_since = df_test_weekend$days_since) %>%
  summarise(price = mean(price))
df_test_weekday_by_date <- df_test_weekday %>%
  group_by(days_since = df_test_weekday$days_since) %>%
  summarise(price = mean(price))
df_test_by_date_combined <- rbind(df_test_weekend_by_date, df_test_weekday_by_date)


# fit loess model with opt_span = 0.03 from 1a
model.loess_weekend <- loess(price ~ days_since, span = opt_span,
                          data = df_train_weekend_by_date,
                          control = loess.control(surface="direct"))
model.loess_weekday <- loess(price ~ days_since, span = opt_span,
                          data = df_train_weekday_by_date,
                          control = loess.control(surface="direct"))

# predict with split model
pred.loess_weekend.train <- predict(model.loess_weekend)
pred.loess_weekend.test <- predict(model.loess_weekend, df_test_weekend_by_date)

pred.loess_weekday.train <- predict(model.loess_weekday)
pred.loess_weekday.test <- predict(model.loess_weekday, df_test_weekday_by_date)
```

```
# rbind test and train
pred.loess_combined.train <- c(pred.loess_weekend.train, pred.loess_weekday.train)
pred.loess_combined.test <- c(pred.loess_weekend.test, pred.loess_weekday.test)

dataset <- c("train", "test")
train.r2.separate <- rsq(df_train_by_date_combined$price, pred.loess_combined.train)
test.r2.separate <- rsq(df_test_by_date_combined$price, pred.loess_combined.test)
separate.model <- c(train.r2.separate, test.r2.separate)
overall.model <- c(train.r2.loess_opt, test.r2.loess_opt)


compare.performance <- data.frame(r2=dataset, separate_model=separate.model, overall_model=overall.model)
compare.performance
```

```
##      r2 separate_model overall_model
## 1 train      0.9912611     0.9705876
## 2  test      0.9294641     0.9264111
```

ANSWER (Question 1b):
Based on our EDA, it appears that avervage prices on weekends (Friday and Saturday) are higher than that on weekdays. Thus, we indeed might benefit from using a different regression model for weekdays and weekends.

As expected, using 2 models to predict prices on weekdays and over weekends seperately is better than the previous lowess model. The training R^2 improves from 0.9705876 to 0.9912611, and the test R^2 improves from 0.9264111 to 0.9294641.

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?

```
# order training data in decreasing order of average price
# top 3 expensive days since
top_3_days_since <- df_train_by_date[order(df_train_by_date$price, decreasing = TRUE), ][1:3, 1]
top_3_dates = c()
for (day_since in top_3_days_since){
  top_3_dates= unique(df_train$date[df_train$days_since == day_since])
}
top_3_dates
```

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

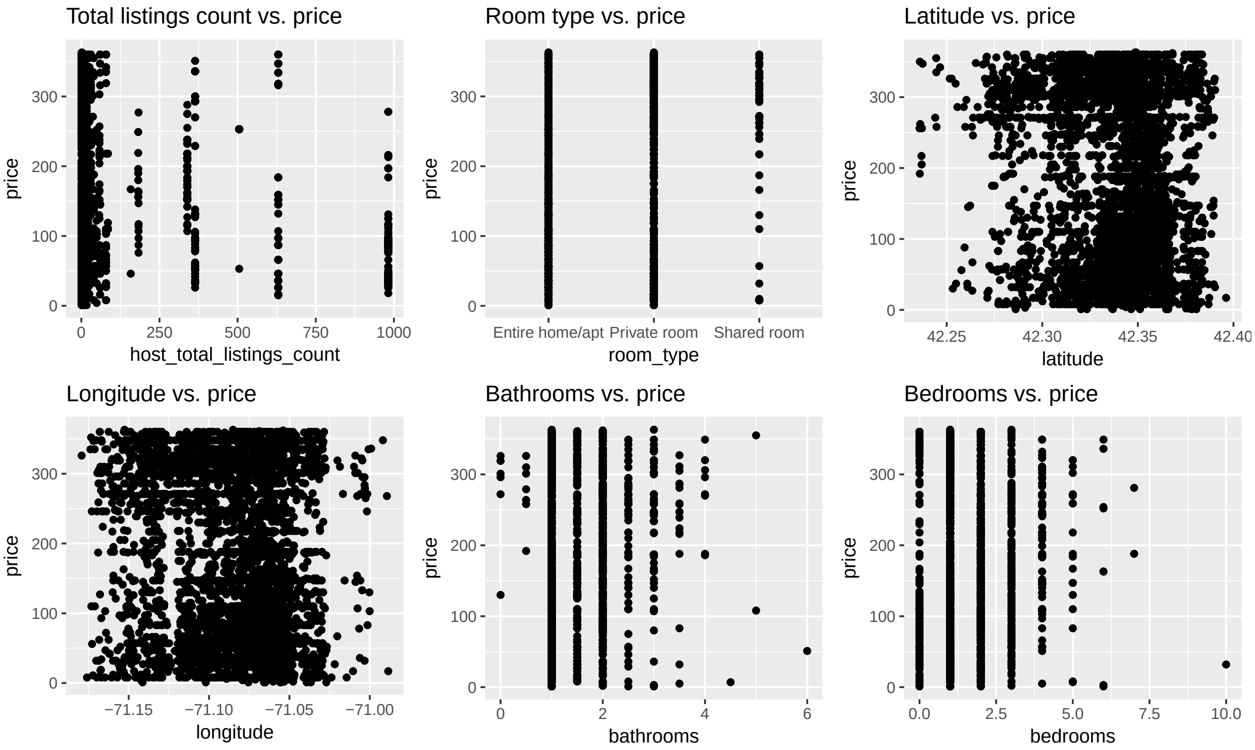
ANSWER (Problem 1c):
The top most expensive dates are 2018-04-14, 2018-04-16, 2018-04-15. Boston Marathon, Red Sox season opening and Patriot's Day events all occur around mid-April, so the demands are usually very high around these dates.

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`.

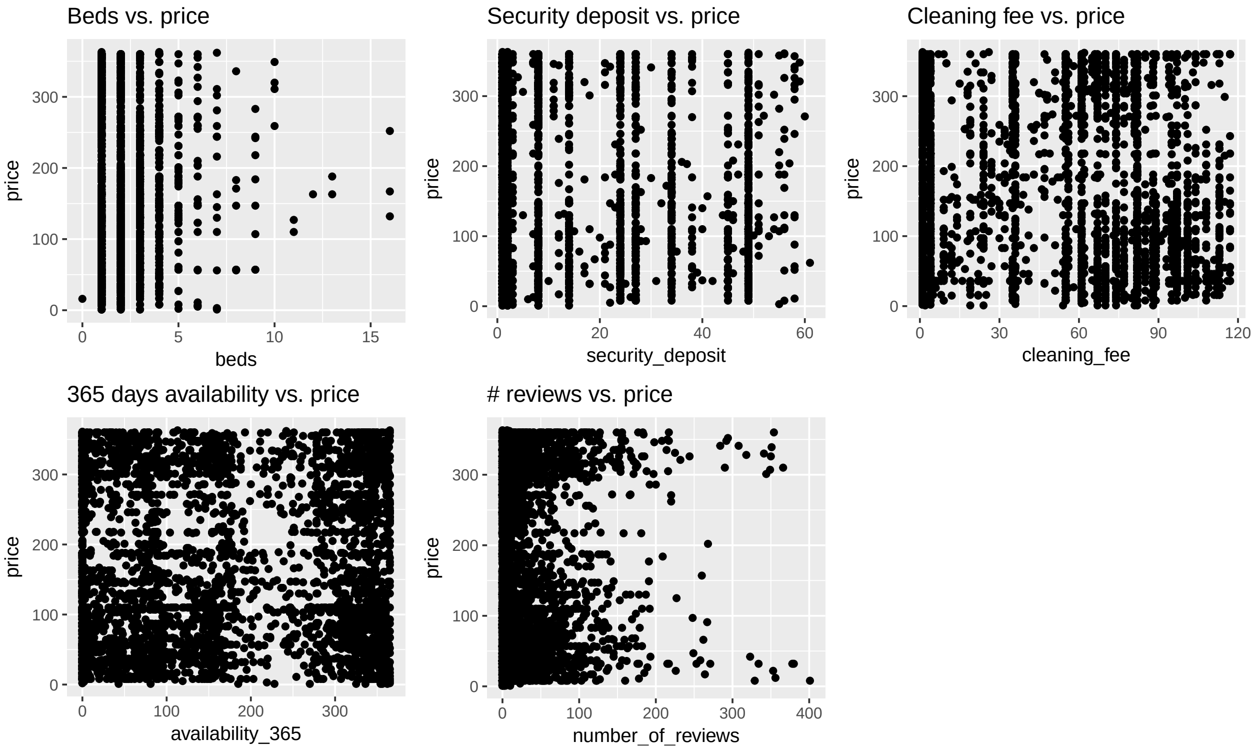
First, visualize the relationship between each of the predictors and the response variable. Does it appear that some of the predictors have a nonlinear relationship with the response variable?

```
df_train2 <- read.csv("data/listings_train.csv")
df_test2 <- read.csv("data/listings_test.csv")

# visualize predictor/response relationship
plot1 <- ggplot(df_train2, aes(x=host_total_listings_count, y=price)) + geom_point() +
  ggtitle("Total listings count vs. price")
plot2 <- ggplot(df_train2, aes(x=room_type, y=price)) + geom_point() +
  ggtitle("Room type vs. price")
plot3 <- ggplot(df_train2, aes(x=latitude, y=price)) + geom_point() +
  ggtitle("Latitude vs. price")

plot4 <- ggplot(df_train2, aes(x=longitude, y=price)) + geom_point() +
  ggtitle("Longitude vs. price")
plot5 <- ggplot(df_train2, aes(x=bathrooms, y=price)) + geom_point() +
  ggtitle("Bathrooms vs. price")
plot6 <- ggplot(df_train2, aes(x=bedrooms, y=price)) + geom_point() +
  ggtitle("Bedrooms vs. price")

grid.arrange(plot1, plot2, plot3, plot4, plot5, plot6, ncol=3)
```



```
plot8 <- ggplot(df_train2, aes(x=beds, y=price)) + geom_point() +
  ggtitle("Beds vs. price")
```

```
plot9 <- ggplot(df_train2, aes(x=security_deposit, y=price)) + geom_point() +
  ggtitle("Security deposit vs. price”)
plot10 <- ggplot(df_train2, aes(x=cleaning_fee, y=price)) + geom_point() +
  ggtitle("Cleaning fee vs. price”)

plot11 <- ggplot(df_train2, aes(x=availability_365, y=price)) + geom_point() +
  ggtitle("365 days availability vs. price”)
plot12 <- ggplot(df_train2, aes(x=number_of_reviews, y=price)) + geom_point() +
  ggtitle("# reviews vs. price”)

grid.arrange(plot8, plot9, plot10, plot11, plot12, ncol=3)
```



ANSWER (Problem 2 - EDA):
According to the visualization, most predictors do not have an apparent linear relationship with the response variable. For instance, the non-linear relationship is most apparent between price and host total listings count. On the other hand, there seems to be a positive linear relationship between price and beds for number of beds over 7.

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

```
model.linear <- lm(price ~., data = df_train2)
train.r2.linear <- rsq(df_train2$price, predict(model.linear, df_train2))
```

```
test.r2.linear <- rsq(df_test2$price, predict(model.linear, df_test2))
train.r2.linear
```

```
## [1] 0.249415
```

```
test.r2.linear
```

```
## [1] 0.1847913
```

```
model.poly3 <- lm(price ~ poly(host_total_listings_count, degree = 3, raw = TRUE) +
                    poly(latitude, degree = 3, raw = TRUE) +
                    poly(longitude, degree = 3, raw = TRUE) +
                    poly(bathrooms, degree = 3, raw = TRUE) +
                    poly(bedrooms, degree = 3, raw = TRUE) +
                    poly(beds, degree = 3, raw = TRUE) +
                    poly(security_deposit, degree = 3, raw = TRUE) +
                    poly(cleaning_fee, degree = 3, raw = TRUE) +
                    poly(availability_365, degree = 3, raw = TRUE) +
                    poly(number_of_reviews, degree = 3, raw = TRUE) +
                    room_type, data = df_train2)
train.r2.poly3 <- rsq(df_train2$price, predict(model.poly3, df_train2))
test.r2.poly3 <- rsq(df_test2$price, predict(model.poly3, df_test2))
train.r2.poly3
```

```
## [1] 0.2706478
```

```
test.r2.poly3
```

```
## [1] 0.2378415
```

ANSWER (Problem 2a):
Regression with 3rd degree polynomial basis ($R^2_{test} = 0.2378415$) performed better on the test set than linear regression ($R^2_{test} = 0.1847913$).

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'`).

```r
library(gam)
# Problrm 2b part 1.
cv_gam = function(train, param_val, k) {
  # Input:
  #   Training data frame: 'train',
  #   Vector of spar 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.formula = as.formula(paste("price ~ ",
                                "s(host_total_listings_count, spar = ", param_val[i],") +",
                                "s(latitude, spar = ", param_val[i],") + ",
                                "s(longitude, spar = ", param_val[i],") + ",
                                "s(bathrooms, spar = ", param_val[i],") + ",
                                "s(bedrooms, spar = ", param_val[i],") + ",
                                "s(beds, spar = ", param_val[i],") + ",
                                "s(security_deposit, spar = ", param_val[i],") + ",
                                "s(cleaning_fee, spar = ", param_val[i],") + ",
                                "s(availability_365, spar = ", param_val[i],") + ",
                                "s(number_of_reviews, spar = ", param_val[i],") + room_type"))
      model.gam = gam(formula = model.formula, data = train[folds!=j, ])

      # Make prediction on fold 'j'
      j_fold_data = train[folds == j,]
      pred = predict(model.gam,
                     data.frame(
                        host_total_listings_count = j_fold_data$host_total_listings_count,
                        latitude = j_fold_data$latitude,
                        longitude = j_fold_data$longitude,
                        bathrooms = j_fold_data$bathrooms,
                        bedrooms = j_fold_data$bedrooms,
                        beds = j_fold_data$beds,
                        security_deposit = j_fold_data$security_deposit,
                        cleaning_fee = j_fold_data$cleaning_fee,
                        availability_365 = j_fold_data$availability_365,
                        number_of_reviews = j_fold_data$number_of_reviews,
                        room_type = j_fold_data$room_type))
      # Compute R^2 for predicted values
      cv_rsq[i] = cv_rsq[i] + rsq(train$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)
}
```

```
spars <- seq(0.1, 1, by = 0.1)
cv_gam_scores <- cv_gam(df_train2, spars, 5)
opt_spar <- spars[which.max(cv_gam_scores)]
cv_gam_scores
```

```
##  [1] 0.0301070 0.1530389 0.2125958 0.2346815 0.2444000 0.2493247 0.2532193
##  [8] 0.2575068 0.2600987 0.2600581
```

```
opt_spar
```

```
## [1] 0.9
```

```
# Refit on best spar value
model.formula.gam_opt <- as.formula(paste("price ~ ",
                                "s(host_total_listings_count, spar = ", opt_spar,") +",
                                "s(latitude, spar = ", opt_spar,") + ",
                                "s(longitude, spar = ", opt_spar,") + ",
                                "s(bathrooms, spar = ", opt_spar,") + ",
                                "s(bedrooms, spar = ", opt_spar,") + ",
                                "s(beds, spar = ", opt_spar,") + ",
                                "s(security_deposit, spar = ", opt_spar,") + ",
                                "s(cleaning_fee, spar = ", opt_spar,") + ",
                                "s(availability_365, spar = ", opt_spar,") + ",
                                "s(number_of_reviews, spar = ", opt_spar,") + room_type"))
model.gam_opt <- gam(formula = model.formula.gam_opt, data = df_train2)

pred.gam_opt.train <- predict(model.gam_opt)
pred.gam_opt.test <- predict(model.gam_opt, df_test2)

train.r2.gam_opt <- rsq(df_train2$price, pred.gam_opt.train)
test.r2.gam_opt <- rsq(df_test2$price, pred.gam_opt.test)
train.r2.gam_opt
```

```
## [1] 0.2769666
```

```
test.r2.gam_opt
```

```
## [1] 0.2304497
```

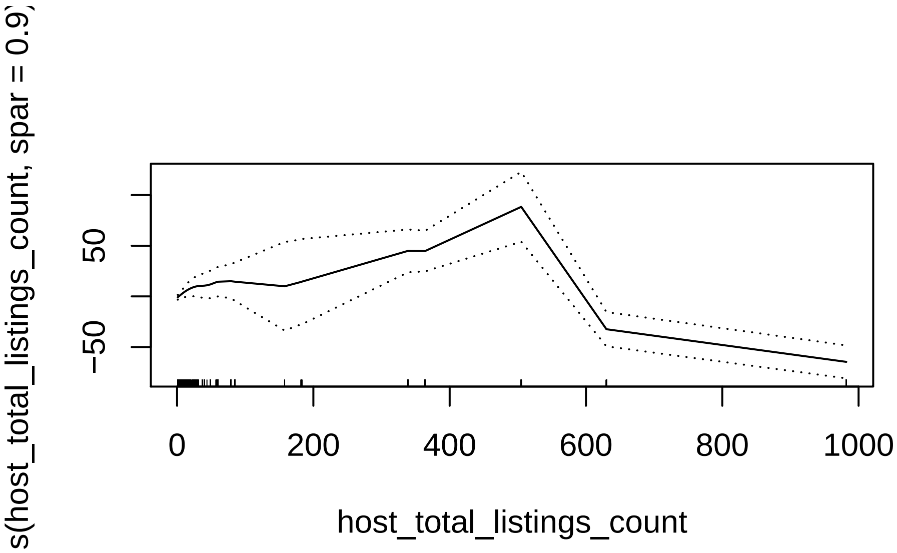
ANSWER (Problem 2 - Advantage of fitting a GAM model):
Since there isn’t an apparent 3-rd degree polynomial relationship between the response variable and any predictor, fitting a GAM model would not impose a certain model form that may cause bias.

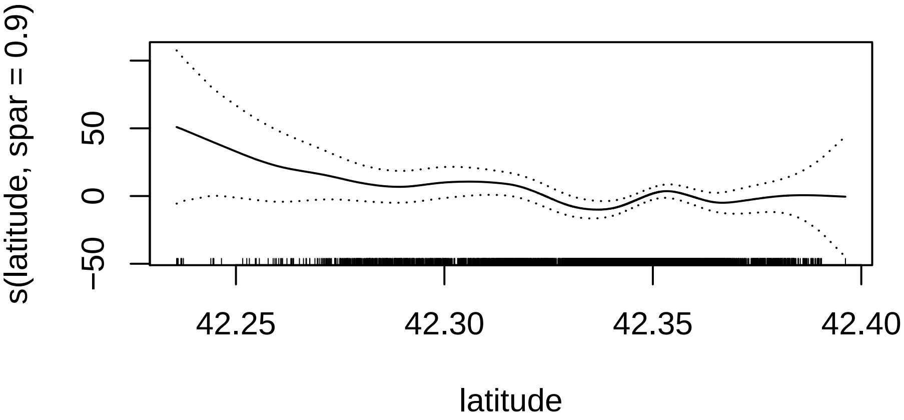
ANSWER (Problem 2b - Part 1):
Using 5-fold cross-validation, we found that the optimal valud of tuning parameter ‘spar‘ = 0.9. We then

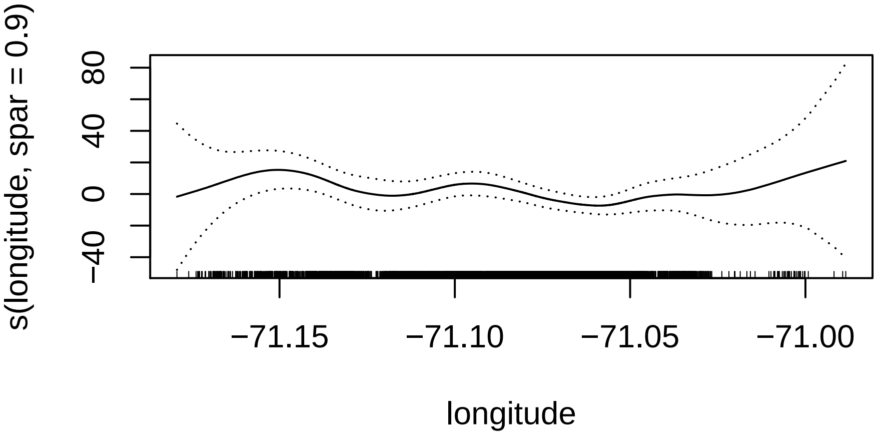
re-fitted a GAM with optimized spar value on the training set.
Comparison of linear regression, 3rd degree polynomial regression and GAM on their test R^2's:

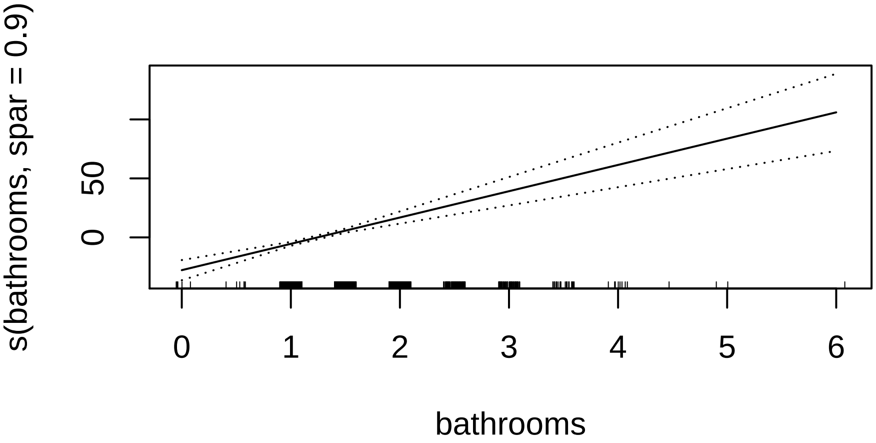
- Linear regression: 0.1847913
- 3rd degree polynomial: 0.2378415
- GAM with optimized spar: 0.2304497

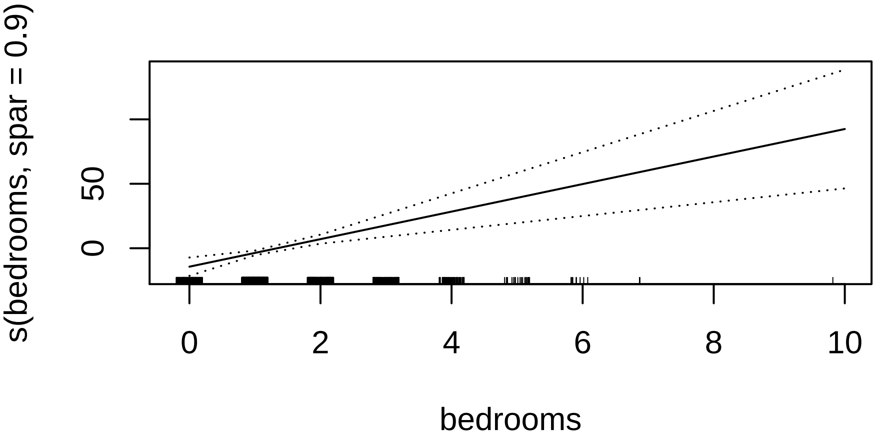
```r
# Problem 2b part 2
plot(model.gam_opt, se = TRUE)
```

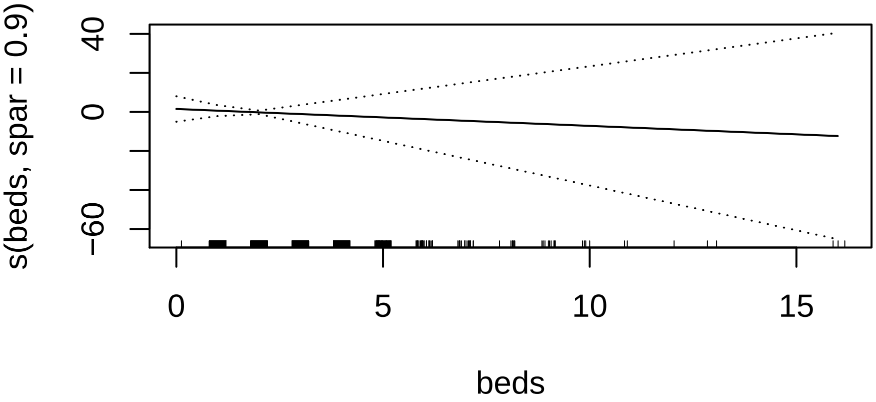


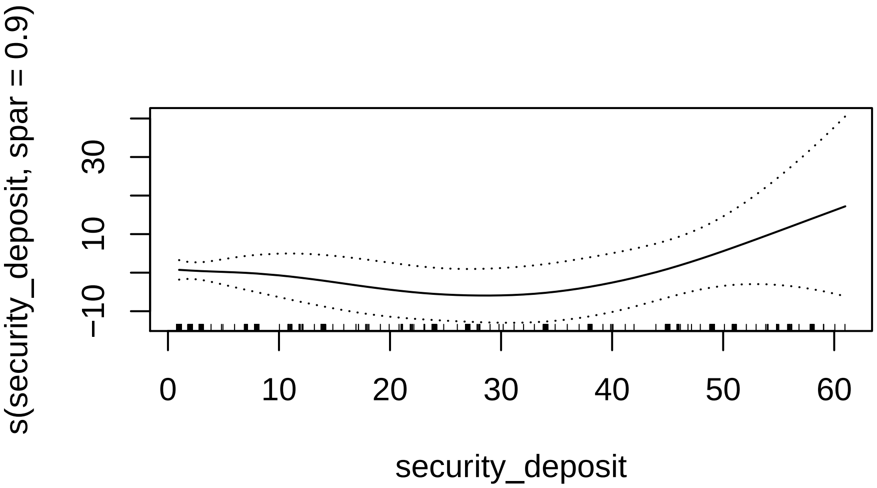


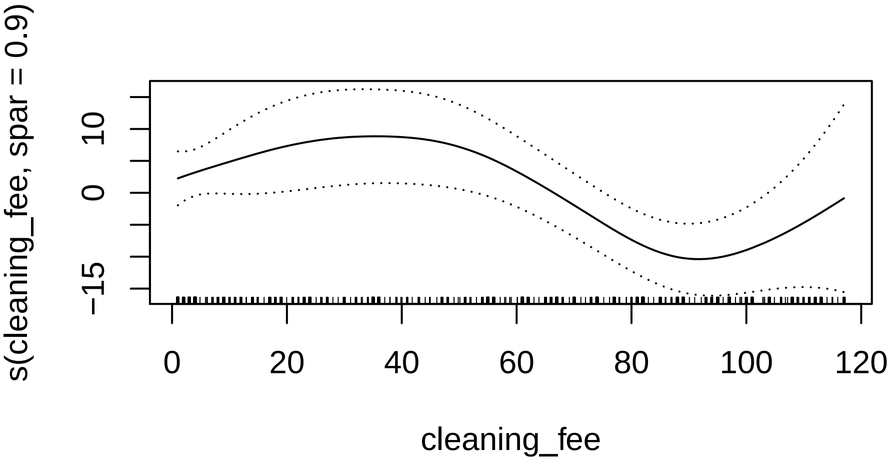


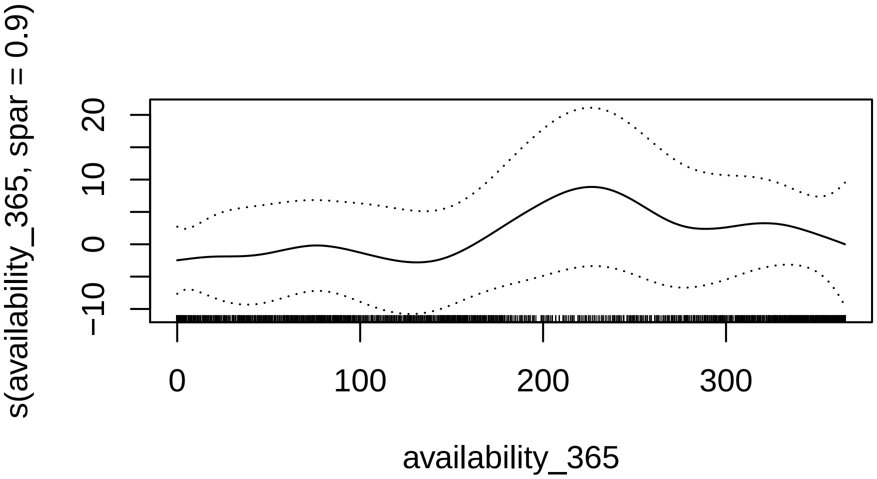

s(bathrooms, spar = 0.9)
0
50
0
1
2
3
4
5
6
bathrooms

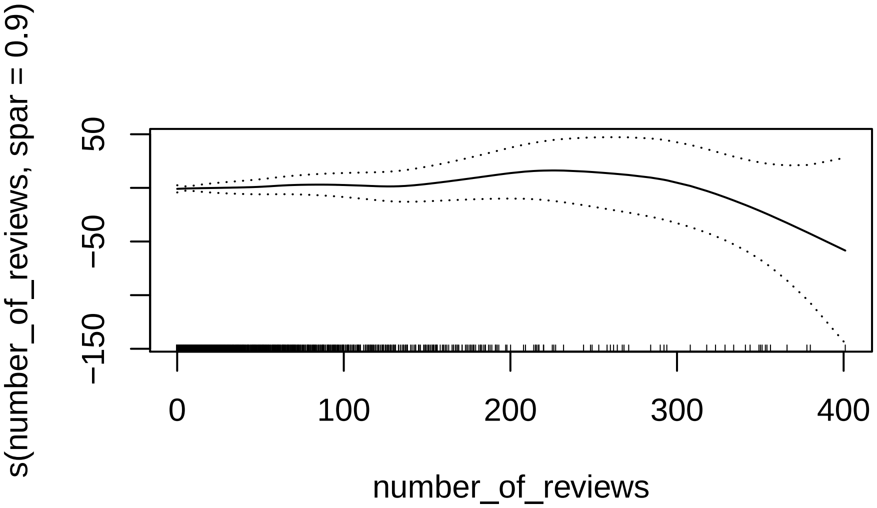

s(bedrooms, spar = 0.9)
0
50
0
2
4
6
8
10
bedrooms

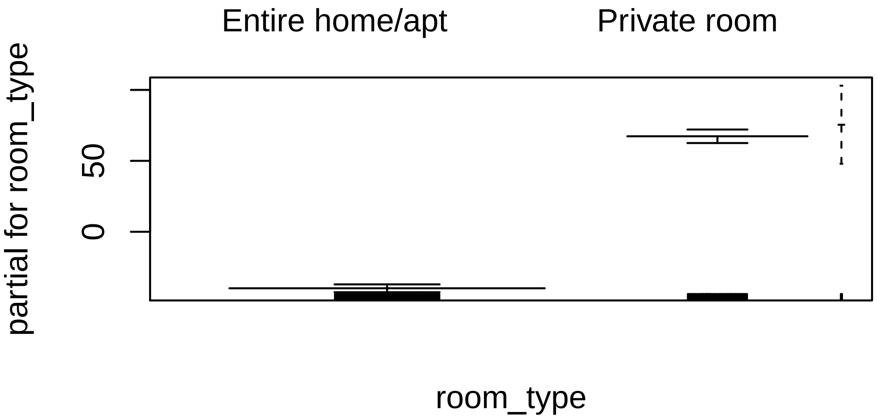

s(beds, spar = 0.9)
40
0
−60
0
5
10
15
beds


s(security_deposit, spar = 0.9)
−10
10
30
0
10
20
30
40
50
60
security_deposit


s(cleaning_fee, spar = 0.9)
−15
0
10
0
20
40
60
80
100
120
cleaning_fee


s(availability_365, spar = 0.9)
−10
0
10
20
0
100
200
300
availability_365





```
model.gam_opt$coefficients
```

```
##                                (Intercept) 
##                              -2.471408e+03 
## s(host_total_listings_count, spar = 0.9) 
##                              -4.373102e-02 
##                   s(latitude, spar = 0.9) 
##                              -1.534795e+02 
##                  s(longitude, spar = 0.9) 
##                              -1.275550e+02 
##                  s(bathrooms, spar = 0.9) 
##                               2.230609e+01 
##                   s(bedrooms, spar = 0.9) 
##                               1.068416e+01 
##                       s(beds, spar = 0.9) 
##                              -8.671794e-01 
##           s(security_deposit, spar = 0.9) 
##                               5.707157e-03 
##               s(cleaning_fee, spar = 0.9) 
##                              -1.074640e-01 
##           s(availability_365, spar = 0.9) 
##                               1.355803e-02 
##          s(number_of_reviews, spar = 0.9) 
##                               1.947156e-02
##                     room_typePrivate room
##                             1.071458e+02
##                      room_typeShared room
##                             1.152748e+02
```

```
summary(model.gam_opt)
```

```
## 
## Call: gam(formula = model.formula.gam_opt, data = df_train2)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max 
## -287.01  -58.54   -1.43   65.08  274.84 
## 
## (Dispersion Parameter for gaussian family taken to be 9657.713)
## 
##     Null Deviance: 57745390 on 4369 degrees of freedom
## Residual Deviance: 41751846 on 4323.161 degrees of freedom
## AIC: 52547.1 
## 
## Number of Local Scoring Iterations: 2 
## 
## Anova for Parametric Effects
##                                              Df   Sum Sq Mean Sq  F value
## s(host_total_listings_count, spar = 0.9)    1.0  1810852 1810852 187.5032
## s(latitude, spar = 0.9)                     1.0  1443182 1443182 149.4331
## s(longitude, spar = 0.9)                    1.0   353429  353429  36.5956
## s(bathrooms, spar = 0.9)                    1.0   855352  855352  88.5667
## s(bedrooms, spar = 0.9)                     1.0   288985  288985  29.9227
## s(beds, spar = 0.9)                         1.0   785601  785601  81.3444
## s(security_deposit, spar = 0.9)             1.0   241679  241679  25.0245
## s(cleaning_fee, spar = 0.9)                 1.0   337580  337580  34.9545
## s(availability_365, spar = 0.9)             1.0    44538   44538   4.6116
## s(number_of_reviews, spar = 0.9)            1.0    41959   41959   4.3446
## room_type                                   2.0  8093528 4046764 419.0189
## Residuals                                4323.2 41751846    9658         
##                                             Pr(>F)    
## s(host_total_listings_count, spar = 0.9) < 2.2e-16 ***
## s(latitude, spar = 0.9)                  < 2.2e-16 ***
## s(longitude, spar = 0.9)                 1.577e-09 ***
## s(bathrooms, spar = 0.9)                 < 2.2e-16 ***
## s(bedrooms, spar = 0.9)                  4.750e-08 ***
## s(beds, spar = 0.9)                      < 2.2e-16 ***
## s(security_deposit, spar = 0.9)          5.885e-07 ***
## s(cleaning_fee, spar = 0.9)              3.635e-09 ***
## s(availability_365, spar = 0.9)            0.03181 *  
## s(number_of_reviews, spar = 0.9)           0.03719 *  
## room_type                                < 2.2e-16 ***
## Residuals                                             
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Anova for Nonparametric Effects
##                                          Npar Df  Npar F     Pr(F)    
## (Intercept)                                                           
## s(host_total_listings_count, spar = 0.9)     6.9 15.1672 < 2.2e-16 ***
## s(latitude, spar = 0.9)                      6.9  3.6095 0.0007611 ***
## s(longitude, spar = 0.9)                     7.2  1.6738 0.1078008
## s(bathrooms, spar = 0.9)                     0.0 13.6510 0.0020406 **
## s(bedrooms, spar = 0.9)                      0.0  3.2313 0.0022280 **
## s(beds, spar = 0.9)                          0.0  0.3873 0.0172462 *
## s(security_deposit, spar = 0.9)              1.6  3.3699 0.0437713 *
## s(cleaning_fee, spar = 0.9)                  2.4  5.2756 0.0028027 **
## s(availability_365, spar = 0.9)              4.7  0.9109 0.4678090
## s(number_of_reviews, spar = 0.9)             4.1  1.1049 0.3527259
## room_type
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

ANSWER (Problem 2b - Part 2):
Based on the nonparametric anova output from our model summary, significant predictors are ‘host_total_listings_count‘, ‘latitude‘, ‘bathrooms‘, ‘bedrooms‘, ‘beds‘, ‘security_deposit‘, and ‘cleaning_fee‘ with varying degrees of significance.

Based on the plots of the smooths with standard errors, we know that a predictor has a significant effect on predicting the response if it has an interval on the x-coordinate where the smooth's confidence interval of that predictor does not contain 0. The sign of the smooth within such significant interval indicates negative/positive effect of the predictor on the response. For instance, we observed that ‘host_total_listings_count‘ has a positive effect on price for ‘host_total_listings_count‘ < 500 but a negative effect on price for ‘host_total_listings_count‘ > 500. Another example is ‘bedrooms‘ has a positive effect on price for ‘bedrooms‘ > 4.

```r
# Problem 2b part 3

# Compare GAM and linear regression model
anova(model.linear, model.gam_opt)
```

```
## 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 ~ s(host_total_listings_count, spar = 0.9) + s(latitude,
##     spar = 0.9) + s(longitude, spar = 0.9) + s(bathrooms, spar = 0.9) +
##     s(bedrooms, spar = 0.9) + s(beds, spar = 0.9) + s(security_deposit,
##     spar = 0.9) + s(cleaning_fee, spar = 0.9) + s(availability_365,
##     spar = 0.9) + s(number_of_reviews, spar = 0.9) + room_type
##   Res.Df      RSS     Df Sum of Sq      F    Pr(>F)
## 1 4357.0 43342821
## 2 4323.2 41751846 33.839   1590975 4.8682 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
cv_gam2 = function(train, param_val, k) {
  # Input:
  #   Training data frame: 'train',
  #   Vector of spar 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.formula = as.formula(paste("price ~ ",
                                "s(host_total_listings_count, spar = ", param_val[i],") +",
                                "s(latitude, spar = ", param_val[i],") + ",
                                "s(longitude, spar = ", param_val[i],") + ",
                                "s(bathrooms, spar = ", param_val[i],") + ",
                                "s(bedrooms, spar = ", param_val[i],") + ",
                                "s(beds, spar = ", param_val[i],") + ",
                                "s(security_deposit, spar = ", param_val[i],") + ",
                                "s(cleaning_fee, spar = ", param_val[i],") + room_type"))
      model.gam = gam(formula = model.formula, data = train[folds!=j, ])

      # Make prediction on fold 'j'
      j_fold_data = train[folds == j,]
      pred = predict(model.gam,
                     data.frame(
                       host_total_listings_count = j_fold_data$host_total_listings_count,
                       latitude = j_fold_data$latitude,
                       longitude = j_fold_data$longitude,
                       bathrooms = j_fold_data$bathrooms,
                       bedrooms = j_fold_data$bedrooms,
                       beds = j_fold_data$beds,
                       security_deposit = j_fold_data$security_deposit,
                       cleaning_fee = j_fold_data$cleaning_fee,
                       room_type = j_fold_data$room_type))

      # 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)
}

cv_gam_scores2 <- cv_gam(df_train2, spars, 5)
opt_spar2 <- spars[which.max(cv_gam_scores2)]
cv_gam_scores2
```

```
##  [1] 0.0301070 0.1530389 0.2125958 0.2346815 0.2444000 0.2493247 0.2532193
##  [8] 0.2575068 0.2600987 0.2600581
```

```
opt_spar2
```

```
## [1] 0.9
```

```
# Refit on best spar value
model.formula.gam_opt2 <- as.formula(paste("price ~ ",
                                "s(host_total_listings_count, spar = ", opt_spar2,") +",
                                "s(latitude, spar = ", opt_spar2,") + ",
                                "s(longitude, spar = ", opt_spar2,") + ",
                                "s(bathrooms, spar = ", opt_spar2,") + ",
                                "s(bedrooms, spar = ", opt_spar2,") + ",
                                "s(beds, spar = ", opt_spar2,") + ",
                                "s(security_deposit, spar = ", opt_spar2,") + ",
                                "s(cleaning_fee, spar = ", opt_spar2,") + room_type"))
model.gam_opt2 <- gam(formula = model.formula.gam_opt2, data = df_train2)

pred.gam_opt.train2 <- predict(model.gam_opt2)
pred.gam_opt.test2 <- predict(model.gam_opt2, df_test2)

train.r2.gam_opt2 <- rsq(df_train2$price, pred.gam_opt.train2)
test.r2.gam_opt2 <- rsq(df_test2$price, pred.gam_opt.test2)
train.r2.gam_opt2
```

```
## [1] 0.2751717
```

```
test.r2.gam_opt2
```

```
## [1] 0.2312019
```

```
anova(model.gam_opt2, model.gam_opt)
```

```
## Analysis of Deviance Table
##
## Model 1: price ~ s(host_total_listings_count, spar = 0.9) + s(latitude,
##     spar = 0.9) + s(longitude, spar = 0.9) + s(bathrooms, spar = 0.9) +
##     s(bedrooms, spar = 0.9) + s(beds, spar = 0.9) + s(security_deposit,
##     spar = 0.9) + s(cleaning_fee, spar = 0.9) + room_type
## Model 2: price ~ s(host_total_listings_count, spar = 0.9) + s(latitude,
##     spar = 0.9) + s(longitude, spar = 0.9) + s(bathrooms, spar = 0.9) +
##     s(bedrooms, spar = 0.9) + s(beds, spar = 0.9) + s(security_deposit,
##     spar = 0.9) + s(cleaning_fee, spar = 0.9) + s(availability_365,
##     spar = 0.9) + s(number_of_reviews, spar = 0.9) + room_type
##   Resid. Df Resid. Dev     Df Deviance Pr(>Chi)
## 1    4333.9   41855494
## 2    4323.2   41751846 10.767   103648   0.4456
```

ANSWER (Problem 2b - Part 3):
Previously, we showed the GAM model has better test R^2 compared to the linear regression model. Based on the anova output, the GAM model indeed improves model performance over linear regression in a statisitically significant manner with a p-value of essentially 0 (< 2.2e-16).

On the other hand, we found that the GAM model with all predictors did not significantly improve model performance over the GAM model with 2 fewer predictors with an anova test p value of 0.4456.

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? **ANSWER (Problem 2c):**
Based on results of EDA and price prediction models, we found that the room prices are higher over weekends than during weekdays, higher in summer than in winter, and reach the highest during April.14 to April.16. Therefore, we would advice a frugal visitor to Boston to book rooms during weekdays, around winter seasons, and avoid the dates between April.14 and April.16 to save money.

In addition, based on the plots of individual smooths, frugal visitors should book rooms with hosts who have ‘host_total_listings_count‘ > 500, or rooms with ‘cleaning_fee‘ between 60 to 80.

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?

```
features <- c('host_total_listings_count', 'latitude', 'longitude',
              'bathrooms', 'bedrooms', 'beds', 'security_deposit',
              'cleaning_fee', 'availability_365', 'number_of_reviews', 'room_type')
n_features <- ncol(df_train2) - 1

f_all <- list() # the list of basis spline functions for each predictor
beta <- mean(df_train2$price) # initialize beta to be the mean of y's

# the matrix that records the predictions of each basis function
f_matrix <- matrix(rep(0, n_features*nrow(df_train2)), ncol = n_features)
for (i in 1:8) {
  for (j in 1:n_features) {
    other_pred <- rowSums(f_matrix[, -j])
    if (j == 11) {
      f_all[[j]] <- lm(response ~ predictor,
                       data = data.frame(predictor = df_train2$room_type,
                                         response = as.matrix(df_train2$price - beta - other_pred)))
      pred_j <- as.matrix(predict(f_all[[j]]))
    }
    else {
      f_all[[j]] = smooth.spline(x = as.matrix(df_train2[features[j]]),
                                 y = as.matrix(df_train2$price - beta - other_pred),
                                    tol=1e-10, spar = opt_spar)
      pred_j <- as.matrix(predict(f_all[[j]], df_train2[features[j]])$y)
    }
    f_matrix[,j] <- pred_j - mean(pred_j)
  }
}

# evaluate R^2 and compare with GAM
pred.backfit.ss.train <- rep(beta, nrow(df_train2))
```

```r
pred.backfit.ss.test <- rep(beta, nrow(df_test2))
for (j in 1:n_features) {
  if (j == 11) {
    pred.backfit.ss.train <- pred.backfit.ss.train +
      as.matrix(predict(f_all[[j]], data.frame(predictor = df_train2$room_type)))
    pred.backfit.ss.test <- pred.backfit.ss.test +
      as.matrix(predict(f_all[[j]], data.frame(predictor = df_test2$room_type)))
  }
  else {
    pred.backfit.ss.train <- pred.backfit.ss.train +
      as.matrix(predict(f_all[[j]], df_train2[features[j]])$y)
    pred.backfit.ss.test <- pred.backfit.ss.test +
      as.matrix(predict(f_all[[j]], df_test2[features[j]])$y)
  }
}

pred.backfit.ss.train.r2 <- rsq(df_train2$price, pred.backfit.ss.train)
pred.backfit.ss.test.r2 <- rsq(df_test2$price, pred.backfit.ss.test)
print(paste0("Backfit train R2 = ", pred.backfit.ss.train.r2))
```

```
## [1] "Backfit train R2 = 0.284062922551028"
```

```r
print(paste0("Backfit test R2 = ", pred.backfit.ss.test.r2))
```

```
## [1] "Backfit test R2 = 0.235727731920683"
```

```r
print(paste0("GAM train R2 = ", train.r2.gam_opt))
```

```
## [1] "GAM train R2 = 0.276966596370425"
```

```r
print(paste0("GAM test R2 = ", test.r2.gam_opt))
```

```
## [1] "GAM test R2 = 0.23044970001143"
```

```r
print(paste0("rsq(pred.backfit.ss.train, pred.gam_opt.train) = ",
             rsq(pred.backfit.ss.train, pred.gam_opt.train)))
```

```
## [1] "rsq(pred.backfit.ss.train, pred.gam_opt.train) = 0.989406698573286"
```

```r
print(paste0("rsq(pred.backfit.ss.test, pred.gam_opt.test) = ",
             rsq(pred.backfit.ss.test, pred.gam_opt.test)))
```

```
## [1] "rsq(pred.backfit.ss.test, pred.gam_opt.test) = 0.988276451695023"
```

ANSWER (Problem 3 - Backfitting):
We implemented the backfitting algorithm by storing a list of basis functions of each predictor X_j. For the basis functions, we used 'smooth.spline' basis function with the same spar parameter of the part 2b GAM for numerical predictors and 'lm' model for the categorical predictor (room_type). Over iterations, each basis function is refitted with the predictor X_j and the response $price - \beta - \Sigma_{k \neq j} s(X_k)$. The predictions of each basis function are stored as columns of a matrix 'f_matrix'. The backfitting model gives a train $R^2 = 0.2840629$ and test $R^2 = 0.2357277$. This is slightly better than the part 2b GAM, which gives a train $R^2 = 0.2769666$ and test $R^2 = 0.2304497$.

The training R^2 score between the predicted values of the backfitting model and GAM is 0.9894067. The test R^2 score is 0.9882765. This shows that the fitted response values are similar to the GAM using the same tuning parameter.