CS 109B, Spring 2017, Homework 1: Smoothers and Generalized Additive Models

Jan 2017

Problem 1: Predicting Taxi Pick-ups

In this problem, the task is to build a regression model that can predict the number of taxi pickups in New York city at any given time of the day. The data set is provided in the files `dataset_1_train.txt` and `dataset_1_test.txt`, and contains details of taxi pickups in the month of Jan 2015. The first column contains the time of the day in minutes, the second column contains information about the day of the week (1 through 7, with 1 denoting Monday) and the last column contains the number of pickups observed at that time.

Visualize the data and check if the pattern of taxi pickups makes intuitive sense.

Part 1a: Explore different regression models

You are required to fit a regression model that uses the time of the day (in minutes) as a predictor and predicts the average number of taxi pick ups at that time. For this part of the question, you can ignore the `DayOfWeek` predictor. 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.

1. Regression models with different basis functions:

- Simple polynomials with degrees 5, 10, and 25
- 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.

```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
crossval_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(PickupCount ~ TimeMin, span = param_val[i],
                          data = train[folds!=j, ],
                          control = loess.control(surface="direct"))

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

      # Compute R^2 for predicted values
      cv_rsq[i] = cv_rsq[i] + rsq(train$PickupCount[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)
}
```

Part 1b: Adapting to weekends

Does the pattern of taxi pickups 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. Use the `DayOfWeek` predictor to 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:

Load train and test sets

```
#load libraries  - These libraries will be used in Problem 1 and Problem 2
library(ggplot2)
library(gridExtra)
library(gam)  #This also loads splines() package
```

```
## Loading required package: splines
```

```
## Loading required package: foreach
```

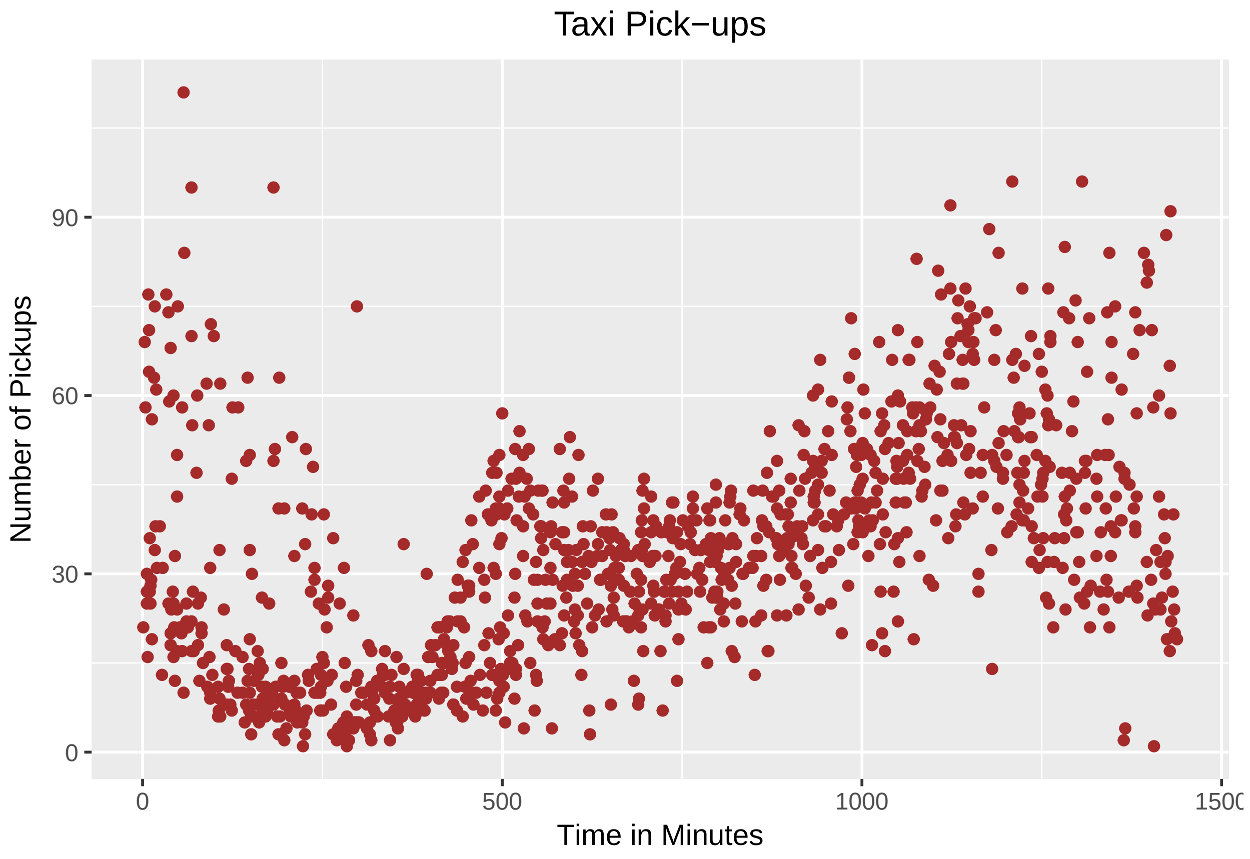
```
## Loaded gam 1.14
```

```
# load train set
train = read.csv("datasets/dataset_1_train.txt", header=TRUE)
cat("Train data size:", dim(train), "\n")
head(train)

# load test set
test = read.csv("datasets/dataset_1_test.txt", header=TRUE)
cat("\nTest data size:", dim(test) , "\n")
head(test)

# data frames of predictors
train.TimeMin <- data.frame(TimeMin = train$TimeMin)
test.TimeMin <- data.frame(TimeMin = test$TimeMin)
```

```
## Train data size: 1125 3
##   TimeMin DayOfWeek PickupCount
## 1      57         5         111
## 2      68         5          95
## 3     182         5          95
## 4     298         5          75
## 5     363         5          35
## 6     395         5          30
##
## Test data size: 1125 3
##   TimeMin DayOfWeek PickupCount
## 1       4         5          51
## 2       5         5          57
## 3      96         5         116
## 4     114         5          95
## 5     201         5          98
## 6     231          5          101
```

Visualize data

```r
# Plot training set
ggplot(train, aes(x =TimeMin, y = PickupCount)) +
    geom_point(col="brown") +
    labs(x="Time in Minutes" , y = "Number of Pickups" ,title="Taxi Pick-ups" ) +
    theme(plot.title = element_text(hjust = 0.5))
```



Part 1a: Explore different regression models

You are required to fit a regression model that uses the time of the day (in minutes) as a predictor and predicts the average number of taxi pick ups at that time. For this part of the question, you can ignore the `DayOfWeek` predictor. 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.

1. Regression models with different basis functions:

(i) Simple polynomials with degrees 5, 10, and 25

Fit regression models with polynomial basis with degrees 5, 10, and 25 to the train data, visualize, report test R^2.

But first, . . . Function to calculate R^2

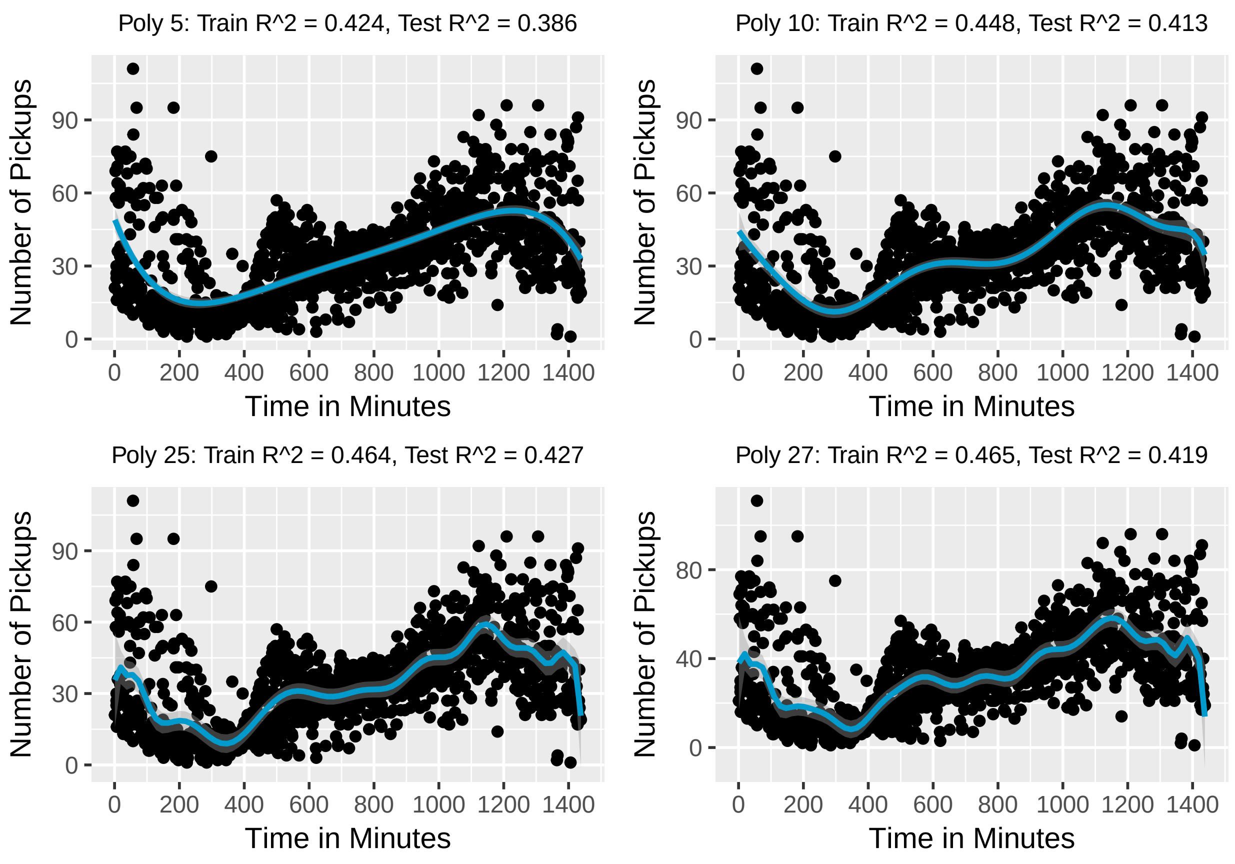
```r
# This function is used in Problem 1 as well as Problem 2
rsq = function(y, predict) {
  tss = sum((y - mean(y))^2)
  rss = sum((y-predict)^2)
  rsq_ = max(0, 1 - rss/tss)

  return(rsq_)
}
```

Function to fit and plot polynomial

```r
fit_plot_poly = function(degree) {
  # Input:
  #   Degree for polynomial: 'degree'
  # Output:
  #   A list with two objects: 'tst_rsq' - Test R^2
  #                            'p' - Plot for the polynomial

  model.poly_ <- lm(PickupCount ~ poly(TimeMin, degree), data=train)

  # Compute train, test R^2
  pred_tr <- predict(model.poly_, newdata=train.TimeMin)
  pred_ts <- predict(model.poly_, newdata=test.TimeMin)

  tr_rsq = rsq(train$PickupCount, pred_tr)
  tst_rsq = rsq(test$PickupCount, pred_ts)

  title_str = sprintf("Poly %d: Train R^2 = %.3f, Test R^2 = %.3f", degree, tr_rsq, tst_rsq)

  # Plot train and test R^2
  p = ggplot(train,aes(x=TimeMin, y=PickupCount))  +
  geom_point() +
  stat_smooth(method = "lm",formula=y ~ poly(x, degree), col = "deepskyblue3")+
  scale_x_continuous( breaks=seq(0,1500,200)) +
  labs(x="Time in Minutes" , y = "Number of Pickups" ,title=title_str ) +
  theme(plot.title = element_text(hjust = 0.5,size=9))

  #returning test R^2 and plot
  return(list(tst_rsq=tst_rsq,p=p))
}
```

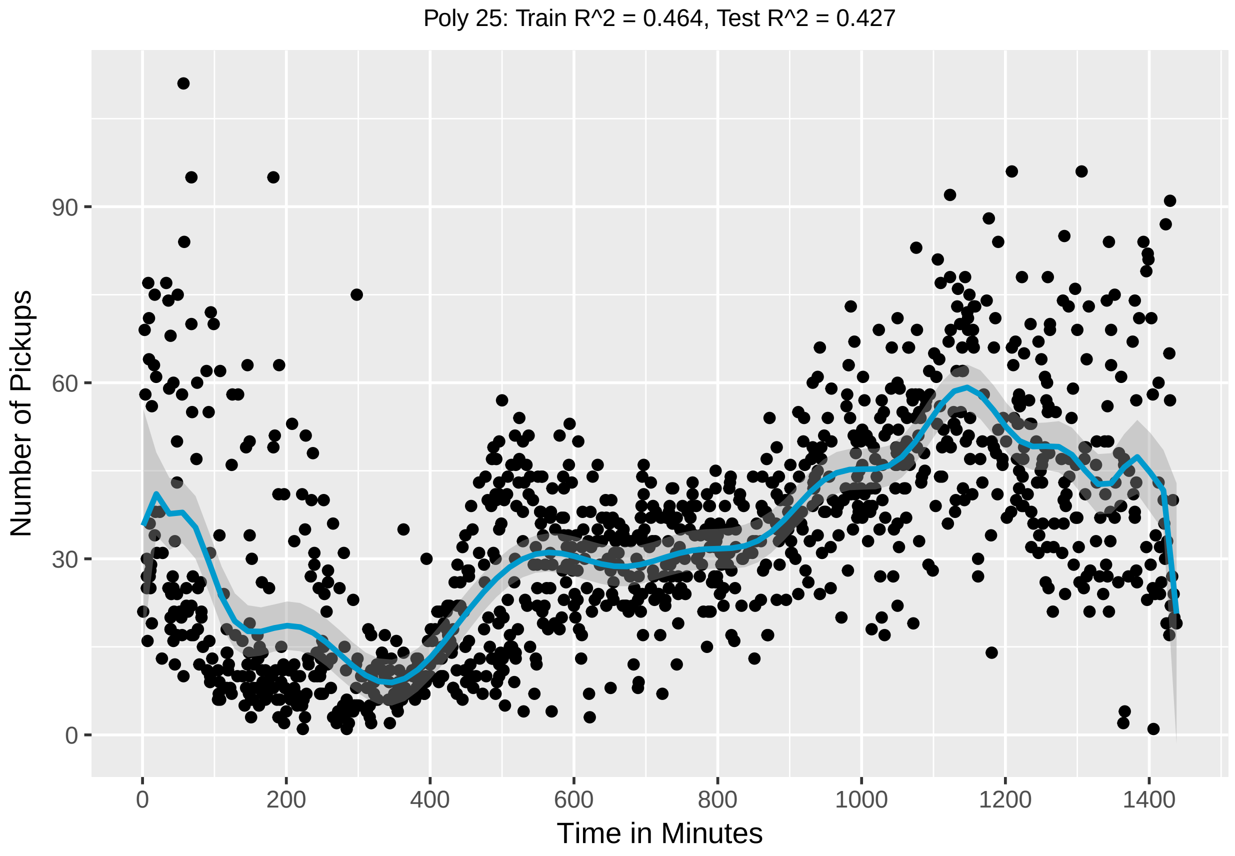
Fit polynomials of degree 5, 10 and 25

```r
# Poly 5
p5 = invisible(fit_plot_poly(5))
# Poly 10
p10 = invisible(fit_plot_poly(10))
# Poly 25
p25 = invisible(fit_plot_poly(25))
# Poly 27
p27 = invisible(fit_plot_poly(27)) #Checking what is the max degree we can set.

grid.arrange(p5$p,p10$p,p25$p,p27$p,nrow=2,ncol=2)
```



Observation: We select poly 25 as the best degree. Model becomes numerically unstable for more than 27 degrees.

```r
best_deg = 25
p25 = fit_plot_poly(best_deg)
poly_test_rsq = p25$tst_rsq
print(p25$p)
```

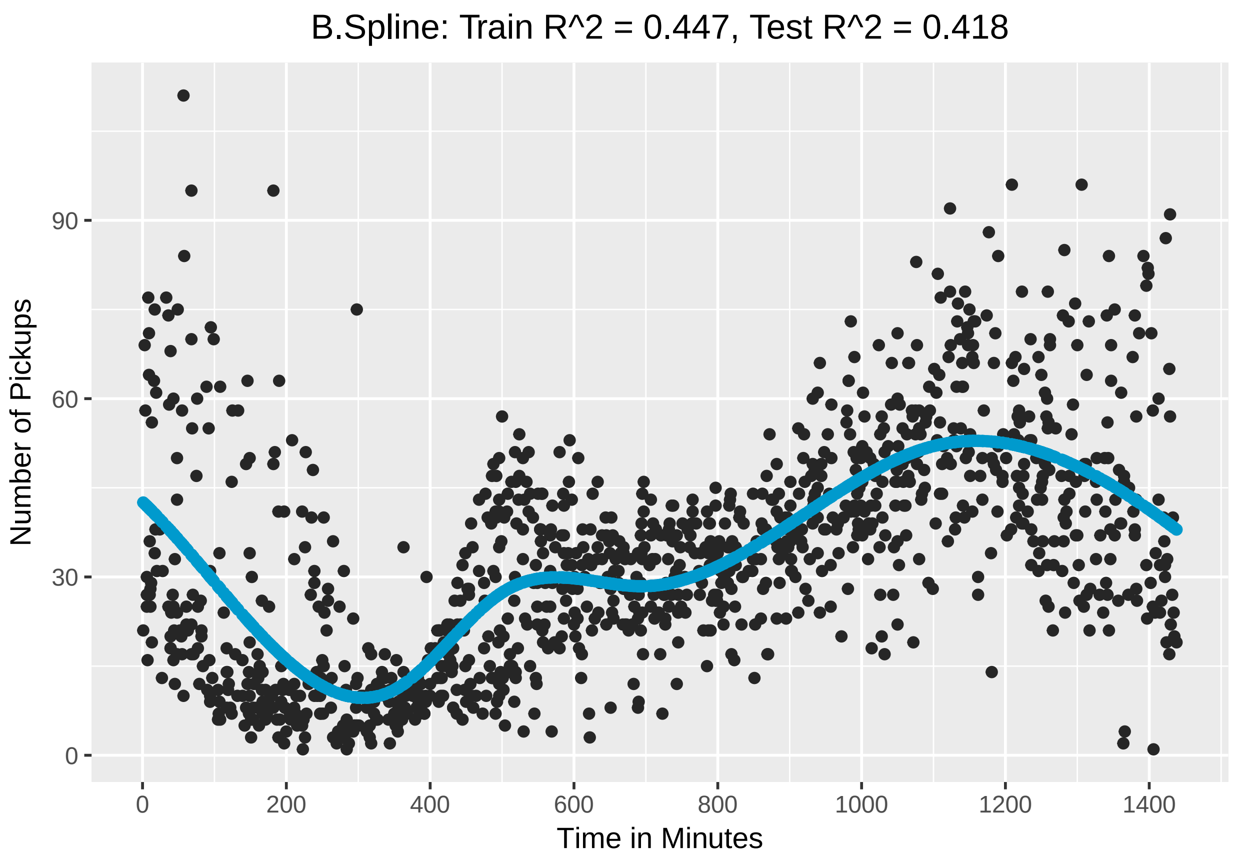


(ii) Cubic B-splines with the knots chosen by visual inspection of the data.

By visual inspection we choose knots at (350, 500, 700, 1100)

Fit regression model with B spline basis with knots at (350, 500, 700, 1100) to the train data, visualize, report test R^2.

```
# Supress warnings
options(warn=-1)

fit.bs_ = lm(PickupCount ~ bs(TimeMin, knots=c(350,500,700,1100)), data=train)

pred_train = predict(fit.bs_, newdata = train)
pred_test = predict(fit.bs_, newdata = test)

bs_train_sq <- rsq(train$PickupCount, pred_train)
bs_test_sq <- rsq(test$PickupCount, pred_test)

title_str = sprintf("B.Spline: Train R^2 = %.3f, Test R^2 = %.3f", bs_train_sq, bs_test_sq)

p = ggplot(train, aes(x =TimeMin, y = PickupCount)) +
        geom_point(col="gray15") +
        geom_point(aes(x=TimeMin, y=pred_train),col="deepskyblue3")+
        scale_x_continuous( breaks=seq(0,1500,200)) +
        labs(x="Time in Minutes" , y = "Number of Pickups" ,title=title_str ) +
        theme(plot.title = element_text(hjust = 0.5))
print(p)
```

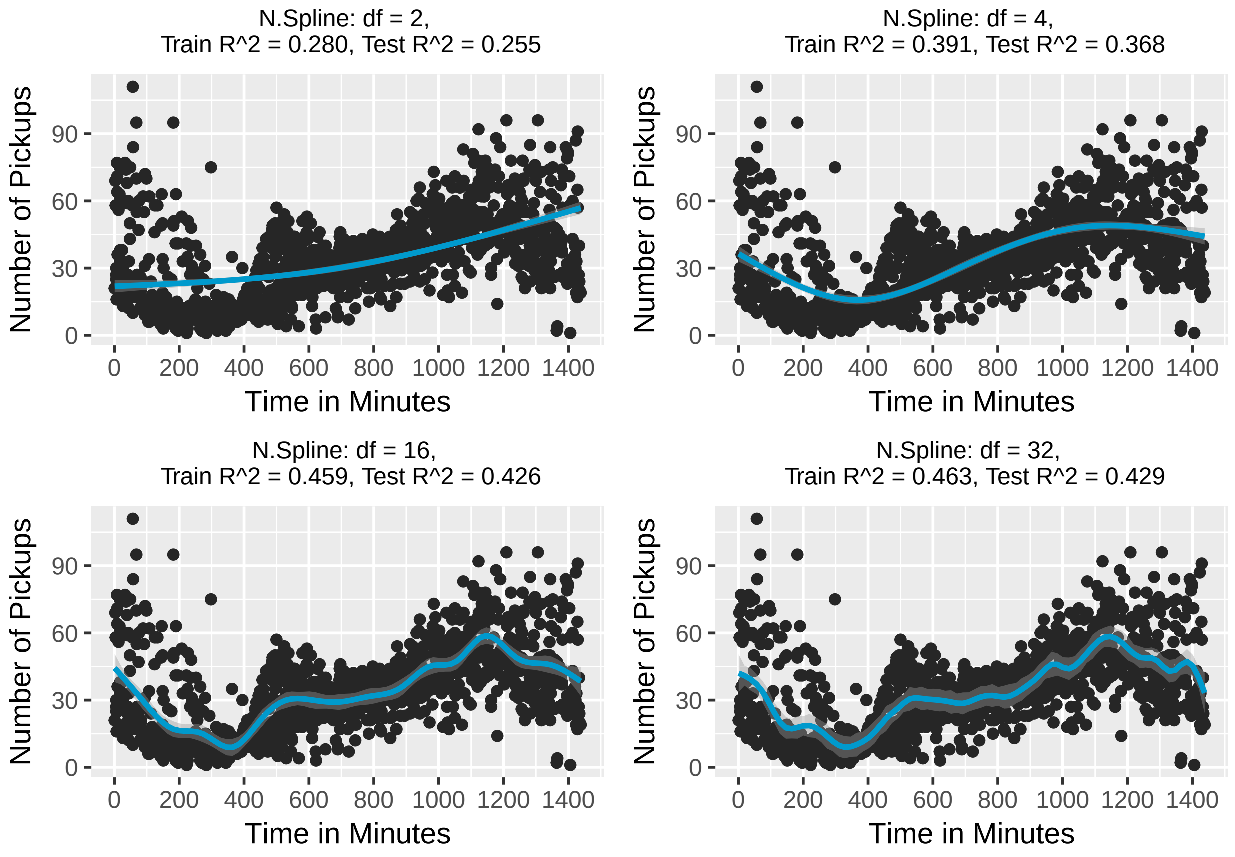


(iii) Natural cubic splines with the degree of freedom chosen by cross-validation on the training set.

```
# Function to fit spline
fit_plot_n_spline = function(df_val) {
  # Input:
  #   Degrees of freedom for Natural Splines: 'df_val'
  # Output:
  #   A list with two objects: 'tst_rsq' - Test R^2
  #                            'p' - Plot for the Natural Splines

  fit.ns_ = lm(PickupCount ~ ns(TimeMin, df=df_val), data=train)

  pred_train = predict(fit.ns_, newdata = train)
  pred_test = predict(fit.ns_, newdata = test)

  train_sq <- rsq(train$PickupCount, pred_train)
  tst_rsq <- rsq(test$PickupCount, pred_test)

  title_str = sprintf("N.Spline: df = %.0f, \n Train R^2 = %.3f, Test R^2 = %.3f", df_val,
                    train_sq, tst_rsq)

  p = ggplot(train, aes(x =TimeMin, y = PickupCount)) +
        geom_point(col="gray15") +
        stat_smooth(method = "lm",formula=y ~ ns(x, df=df_val), col = "deepskyblue3")+
        scale_x_continuous( breaks=seq(0,1500,200)) +
        labs(x="Time in Minutes" , y = "Number of Pickups" ,title=title_str ) +
        theme(plot.title = element_text(hjust = 0.5,size=9))

  #returning test R^2 and plot
  return(list(tst_rsq=tst_rsq,p=p))
}
```

Fit regression model with Natural spline basis with df 2, 4, 8, 16 on train data, visualize, report test R^2.

```
#Lets explore the effect of tuning parameter -df, before we use cross validation.
s2 = invisible(fit_plot_n_spline(2))
s4 = invisible(fit_plot_n_spline(4))
s8 = invisible(fit_plot_n_spline(16))
s16 = invisible(fit_plot_n_spline(32))

grid.arrange(s2$p,s4$p,s8$p,s16$p,nrow=2,ncol=2)
```

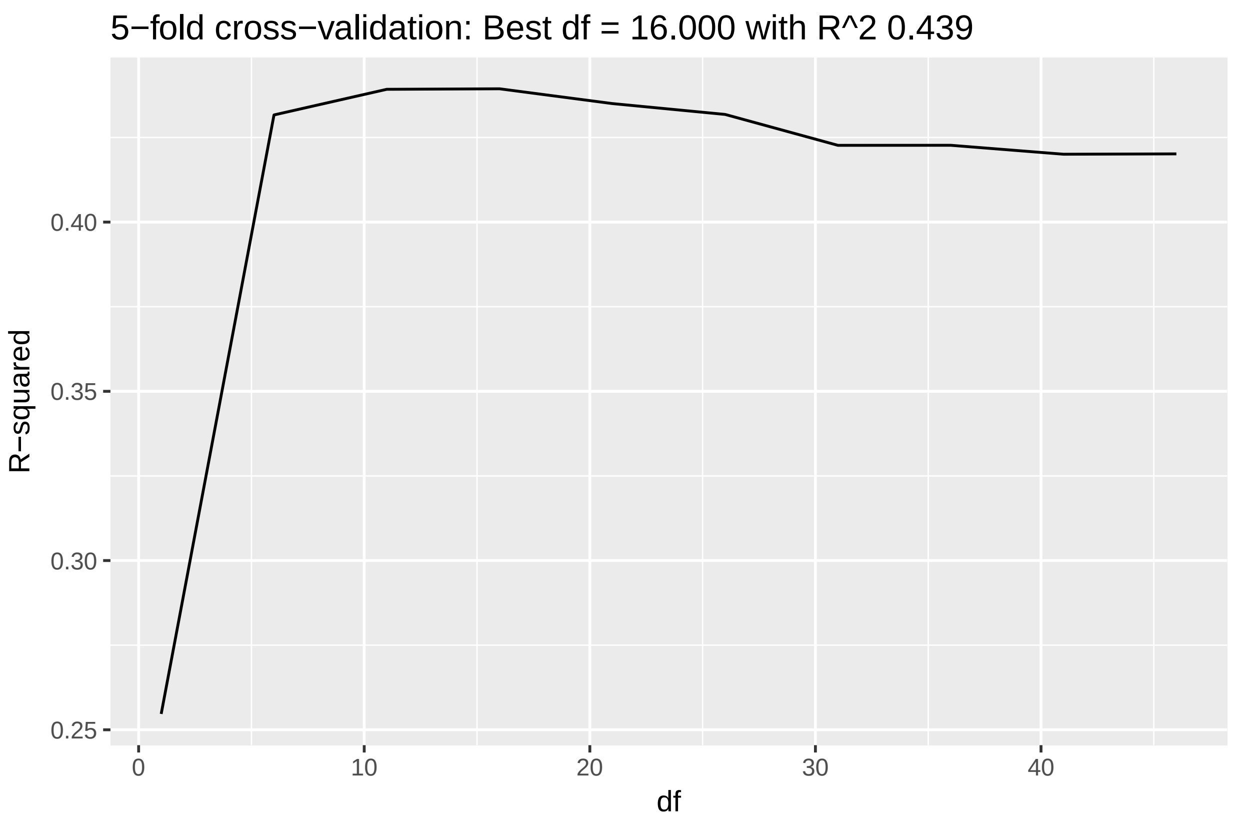


Observation: As df is increasing, the curve is getting wigglier.

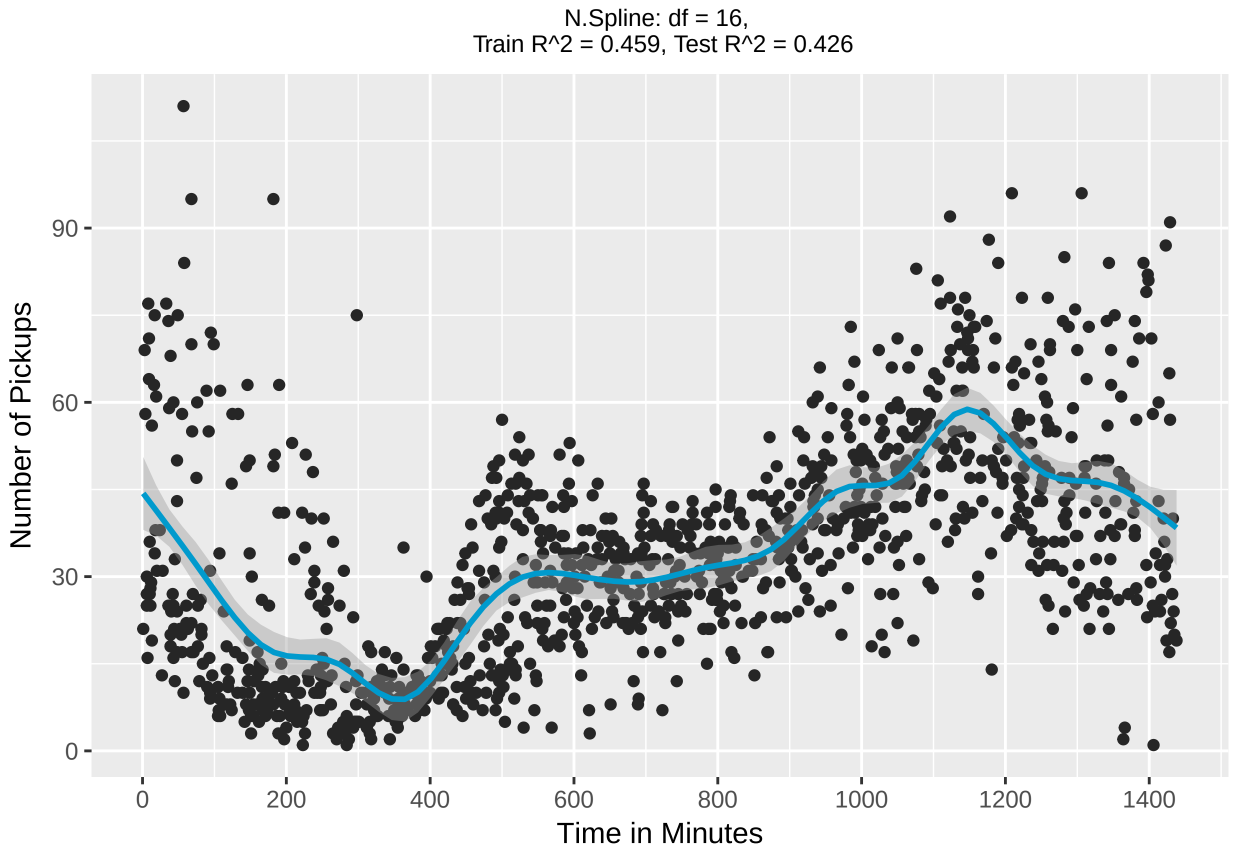
Apply 5-fold cross-validation on the train set to find the best df, and report the corresponding test R^2.

```
crossval_ns = function(df_param, k) {
  # Input:
  #   Degrees of freedom for Natural Splines: 'df_param',
  #   Number of CV folds: 'k'
  # Output:
```

```
  #   Vector of R^2 values for the provided parameters: 'rsq_res'


  #k = 5 #Folds

  # sample from 1 to k, nrow times (the number of observations in the data)
  set.seed(109)
  train$id <- sample(1:k, nrow(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(train, id %in% list[-i])
    testset <- subset(train, id %in% c(i))

    # run model
    fit.ns = lm(PickupCount ~ ns(TimeMin, df=df_param), data=trainingset)

    # calculate R^2 on test set
    rsq_res[i] = rsq(testset$PickupCount, predict(fit.ns, newdata = testset))

  }

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

dfs = seq(1,50,by=5)
res = rep(NA, length(dfs))

for (i in 1:length(dfs)) {
  res[i] = crossval_ns(dfs[i],5)  #5 fold CV
}

# Find degree with highest CV R^2
best_df = which(res==max(res))
title_str = sprintf("\n5-fold cross-validation: Best df = %.3f with R^2 %.3f", dfs[best_df],
                    res[best_df])

# Plot cross-validation R^2
ggplot() +
  geom_line(aes(x=dfs,y=res)) +
  labs(x="df" , y = "R-squared" ,title=title_str )
```



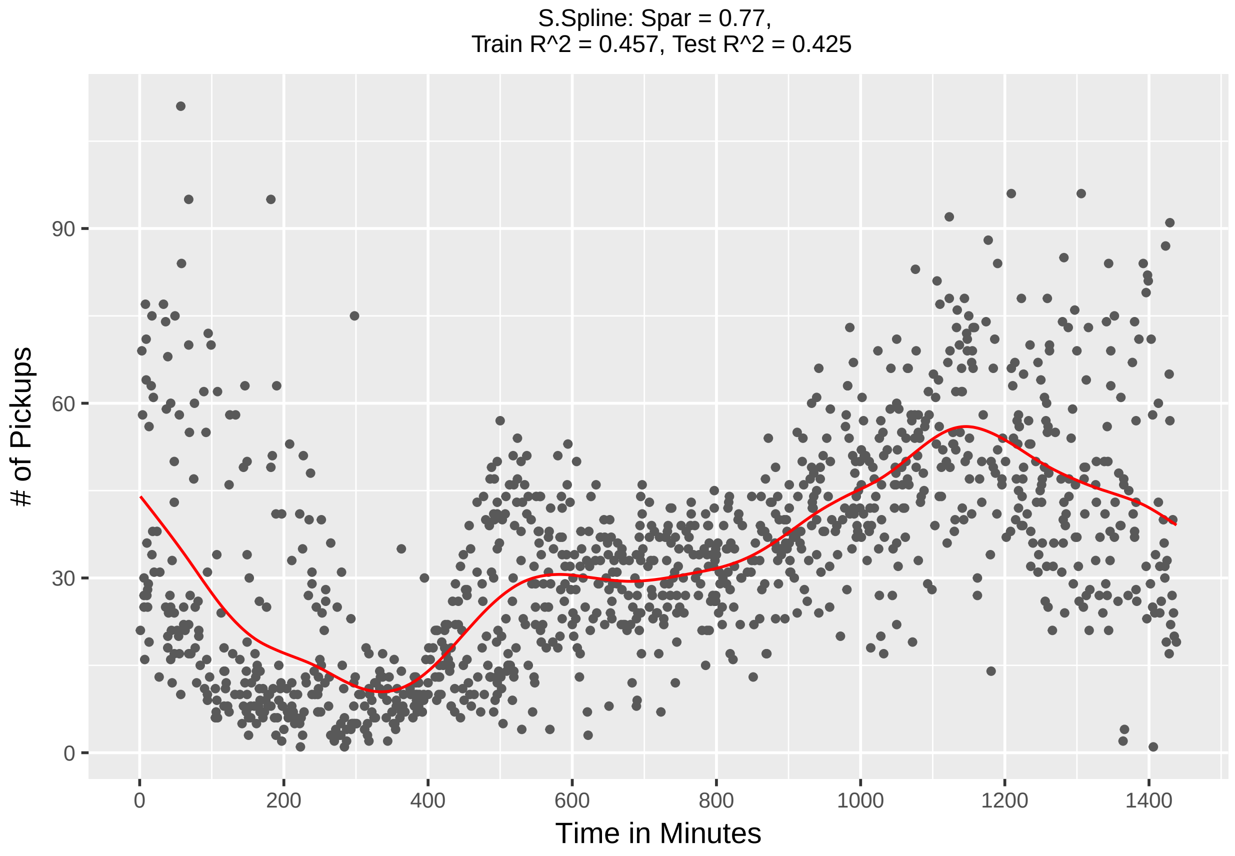
```
s_best = invisible(fit_plot_n_spline(dfs[best_df]))
ns_test_rsq = s_best$tst_rsq
print(s_best$p)
```



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

Apply cross-validation on the train set to find the best spar value, and report the corresponding test R^2.

```
fit.sp = smooth.spline(train$PickupCount~train$TimeMin, cv = TRUE) #Chosing ordinary leave one out cros
train_sq <- rsq(train$PickupCount, fitted(fit.sp))
test_sq <- rsq(test$PickupCount, predict(fit.sp,test$TimeMin)$y)
spline_test_rsq <-  test_sq
title_str = sprintf("S.Spline: Spar = %.2f, \n Train R^2 = %.3f, Test R^2 = %.3f",
                    fit.sp$spar, train_sq, test_sq)

p = ggplot() +
    geom_point(data = train, aes(x = TimeMin, y = PickupCount),col="gray35" , size=1) +
    geom_line(aes(x = fit.sp$x, y = fit.sp$y),col="red") +
    scale_x_continuous( breaks=seq(0,1500,200)) +
    labs(x="Time in Minutes" , y = "# of Pickups" ,title=title_str ) +
    theme(plot.title = element_text(hjust = 0.5,size=9),
          axis.text=element_text(size=8))
print(p)
```



Observation: We used `cv` parameter in smooth.spline to select the best smoothing parameter. If cv is set to TRUE, it uses ordinary Leave one out cross validation method. Best spar returned by model is 0.77 and test R^2 is 0.425.

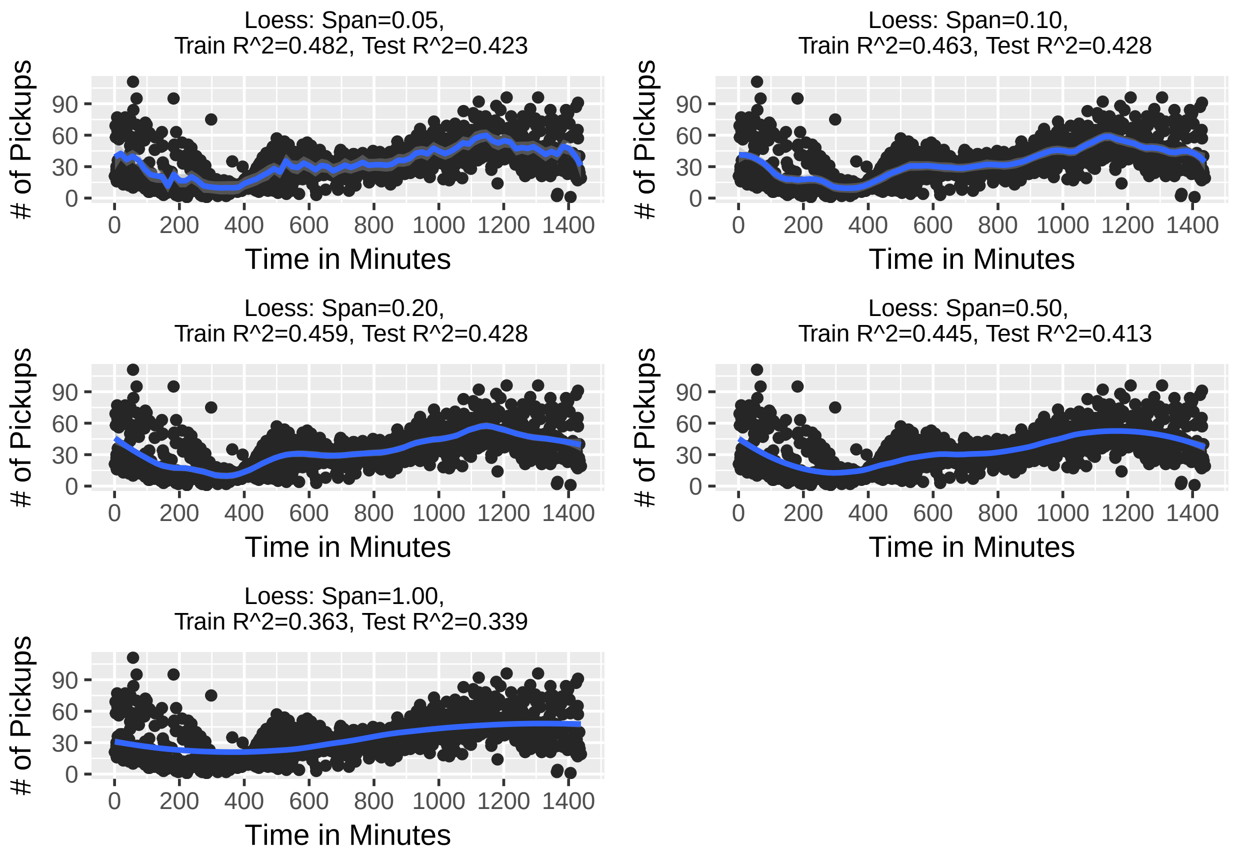
3. Locally-weighted regression model with the span parameter chosen by cross-validation on the training set

Function to fit and plot LOESS

```r
fit_plot_loess = function(train,test,span_param) {
  # Input:
  #   Training dataframe: 'train',
  #   Test dataframe: 'test',
  #   Tuning parameter span for loess: 'span_param'
  # Output:
  #   A list with two objects: 'tst_rsq' - Test R^2
  #                            'p' - Plot for loess

    set.seed(109)
    model.loess = loess(PickupCount~TimeMin, span=span_param, data = train,
                        control=loess.control(surface="direct"))
    y.predict <- predict(model.loess, train$TimeMin)
    pred <- predict(model.loess, test$TimeMin)

    trainrsq = rsq(train$PickupCount, y.predict)
    tst_rsq = rsq(test$PickupCount, pred)
    title_str = sprintf("Loess: Span=%.2f, \n Train R^2=%.3f, Test R^2=%.3f", span_param,
                        trainrsq, tst_rsq)

    p = ggplot(train, aes(x =TimeMin, y = PickupCount)) +
        geom_point(col="gray15") +
        stat_smooth(method="loess",formula = y~x, span=span_param) +
        scale_x_continuous( breaks=seq(0,1500,200)) +
        labs(x="Time in Minutes" , y = "# of Pickups" ,title=title_str ) +
        theme(plot.title = element_text(hjust = 0.5,size=9))

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

Fit LOESS models with span = 0.05, 0.1, 0.2, 0.5, 1

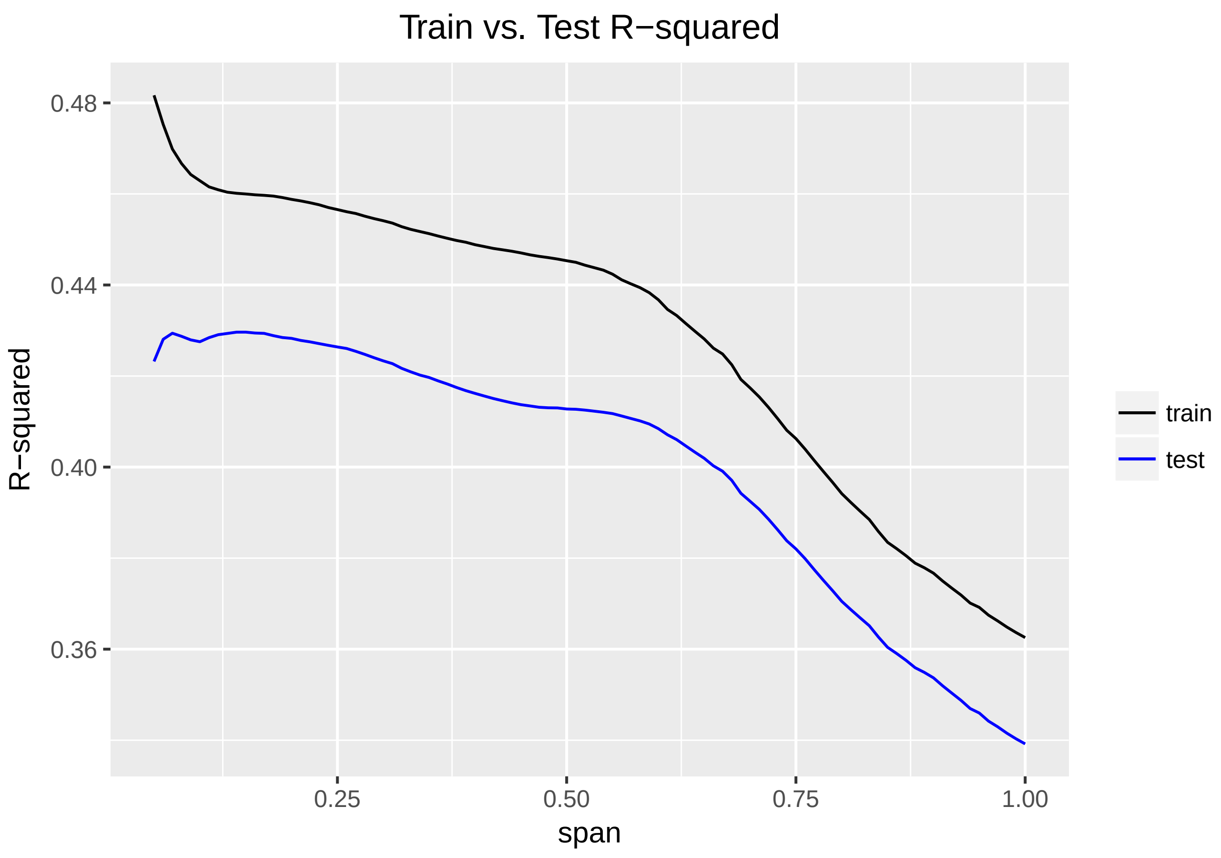
```
##Lets explore the effect of tuning parameter - span, before we use cross validation.
l0 = invisible(fit_plot_loess(train,test,0.05))
l1 = invisible(fit_plot_loess(train,test,0.1))
l2 = invisible(fit_plot_loess(train,test,0.2))
l3 = invisible(fit_plot_loess(train,test,0.5))
l4 = invisible(fit_plot_loess(train,test,1))

grid.arrange(l0$p,l1$p,l2$p,l3$p,l4$p,nrow=3,ncol=2)
```



Observation: The smaller the value of span, the more local and wiggly the fit is. A larger value of span will lead to a more global fit.

Plot train and test R^2 as a function of span values

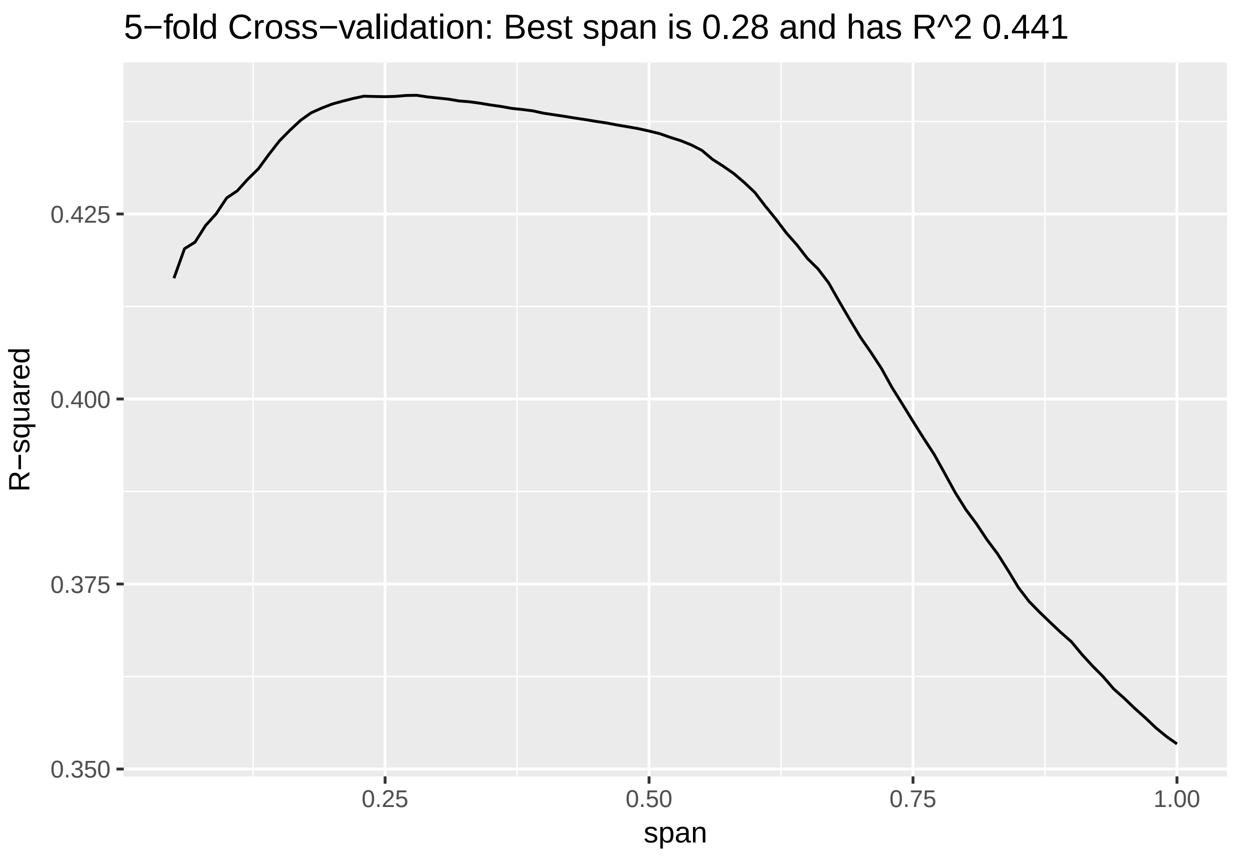
```r
#Let's visualize how R^2 is changing on train and test dataset with different span values.
spans = seq(0.05,1,by=0.01)

n = length(spans)
trainrsq = rep(NA, n)
testrsq = rep(NA, n)

for (i in 1:n) {
  set.seed(109)
  model.loess = loess(train$PickupCount~train$TimeMin, span=spans[i],
                      control=loess.control(surface="direct"))
  y.predict <- predict(model.loess, data.frame(TimeMin=train$TimeMin))
  pred <- predict(model.loess, test$TimeMin)
  trainrsq[i] = rsq(train$PickupCount, y.predict)
  testrsq[i] = rsq(test$PickupCount, pred)
}

p = ggplot() +
  geom_line(aes(x=spans,y=trainrsq, colour="train")) +
  geom_line(aes(x=spans,y=testrsq,colour="test"))  +
  scale_colour_manual("",
                      breaks = c("train", "test"),
                      values = c("blue", "black")) +
  labs(x="span" , y = "R-squared" ,title="Train vs. Test R-squared" )  +
  theme(plot.title = element_text(hjust = 0.5))

print(p)
```



Apply 5-fold cross-validation on the train set to find the best span value, and report the corresponding test R^2.

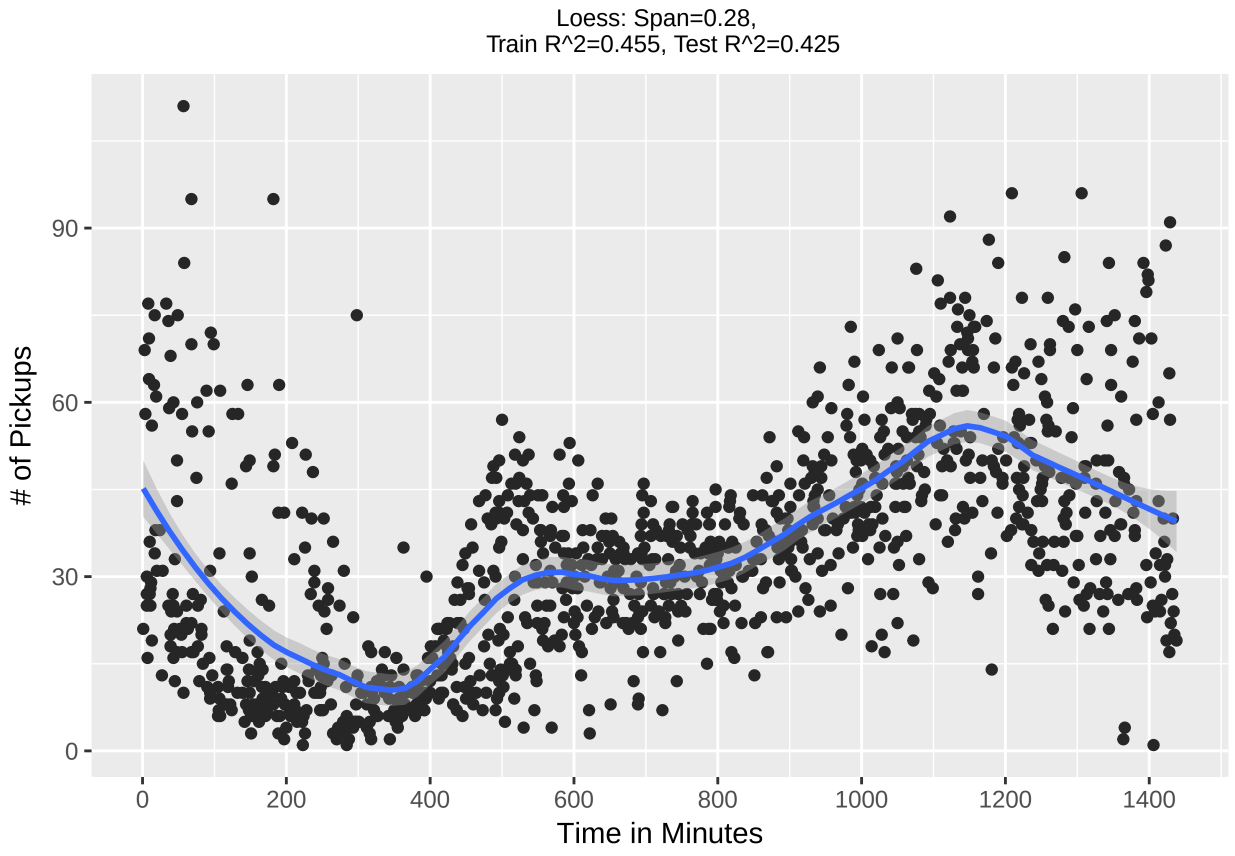
```
crossval_loess = function(train, param_val, k) {
  # Input:
  #   Training data frame: 'train',
  #   Vector of span parameter values: 'param_val',
  #   Number of CV folds: 'k'
  # Output:
  #   Vector of R^2 values for the provided parameters: 'cv_rsq'

  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(PickupCount ~ TimeMin, span = param_val[i],
```

```r
                          data = train[folds!=j, ],
                          control = loess.control(surface="direct"))

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

      # Compute R^2 for predicted values
      cv_rsq[i] = cv_rsq[i] + rsq(train$PickupCount[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)
}

spans = seq(0.05,1,by=0.01)

cv_rsq = crossval_loess(train,spans,5)
# Find best span value
best_span = which(cv_rsq==max(cv_rsq))

title_str = sprintf("5-fold Cross-validation: Best span is %.2f and has R^2 %.3f",
                    spans[best_span], cv_rsq[best_span])

# Plot
ggplot() +
   geom_line(aes(x=spans,y=cv_rsq)) +
   labs(x="span" , y = "R-squared" ,title=title_str )
```



Retrain on whole train data set with chosen span value

```
a = fit_plot_loess(train,test,spans[best_span])
loess_test_rsq = a$tst_rsq
print(a$p)
```



Summary of all models

```
cat("Poly:", poly_test_rsq, "\nB. Spline:", bs_test_sq, "\nN. Spline:", ns_test_rsq,
    "\nS.Spline:", spline_test_rsq, "\nLOESS:", loess_test_rsq)
```

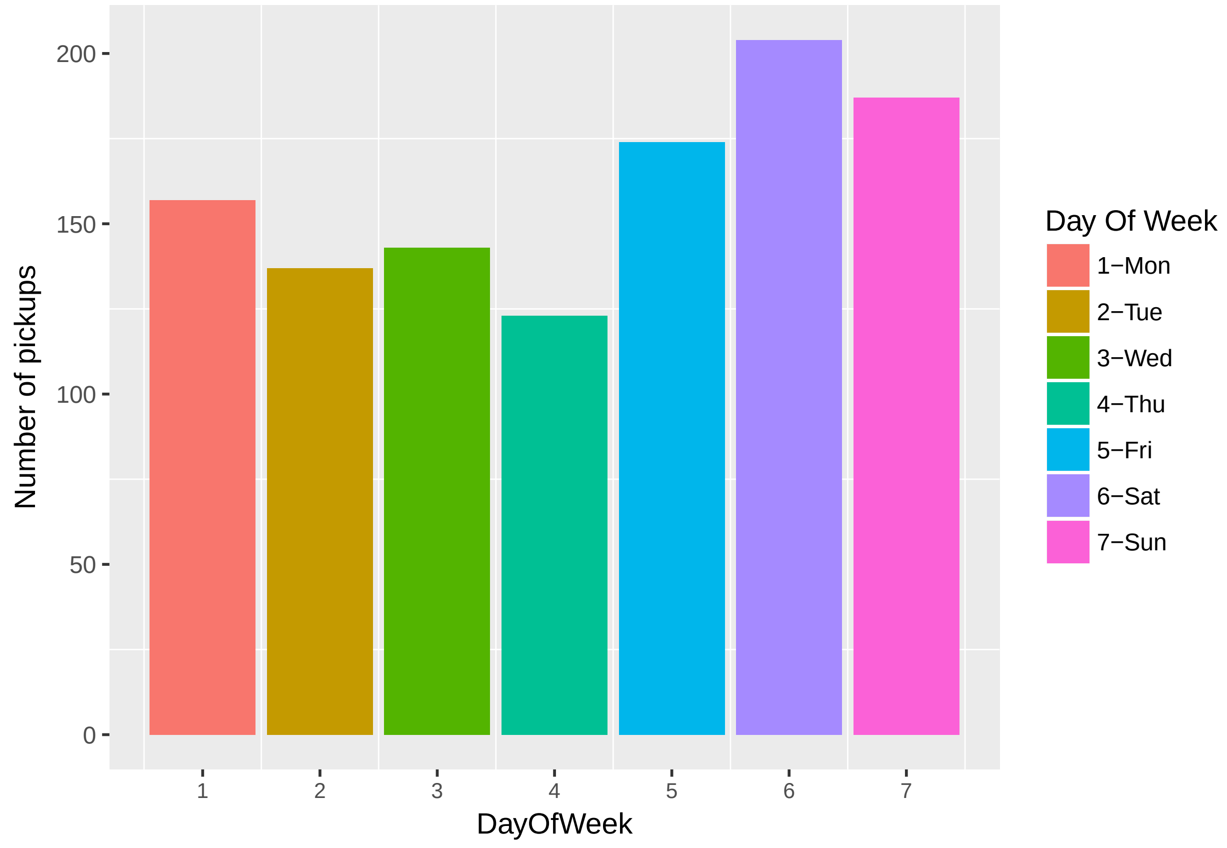
```
## Poly: 0.4266038
## B. Spline: 0.4176375
## N. Spline: 0.4261959
## S.Spline: 0.425427
## LOESS: 0.4247486
```

Observation: It appears that all methods provide similar performance. One reason regression with polynomial basis might be undesirable is because it becomes numerically unstable for high degree polynomials.

Part 1b: Adapting to weekends

Does the pattern of taxi pickups 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. Use the `DayOfWeek` predictor to 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)

```
ggplot(train, aes(x=DayOfWeek,fill=factor(DayOfWeek))) +
    geom_bar() +
    scale_x_continuous(name="DayOfWeek", breaks=seq(1,7,1)) +
    ylab("Number of pickups") +
    theme(axis.text.x = element_text(angle=0,size=8),
          panel.grid.major.x = element_blank(),
          panel.grid.major.y = element_blank() ) +
    scale_fill_discrete(name="Day Of Week",labels=c("1-Mon", "2-Tue", "3-Wed", "4-Thu",
                                                     "5-Fri", "6-Sat", "7-Sun"))
```



Observation: Yes the pattern of taxi pickups is different on weekends (DayOfWeek = 6 or 7) compared to weekdays. On weekends, there is a spike in the number of taxi-pickups in the night / early morning.

```
#We trained a loess model on complete train dataset in 1A. We got the following $R^2$
cat ("Loess R^2 on complete train dataset: ", loess_test_rsq)
```

```
## Loess R^2 on complete train dataset:  0.4247486
```

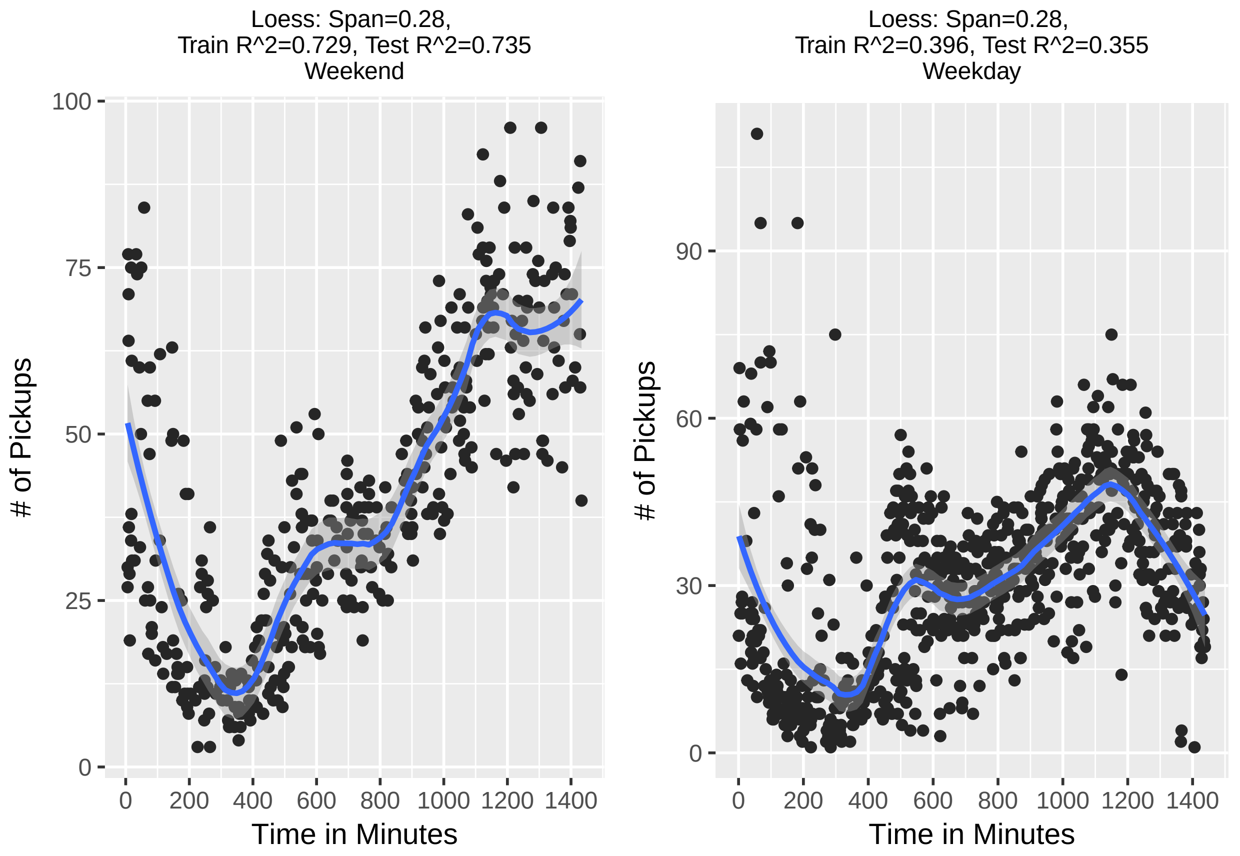
```
#Divide the dataset in Weekday and Weekend
train_weekday = train[train$DayOfWeek<=5,]
train_weekend = train[train$DayOfWeek>5,]
test_weekday = test[test$DayOfWeek<=5,]
test_weekend = test[test$DayOfWeek>5,]
```

```
# Next we are going to predict on test_weekend and test_weekday on the model created in 1A.

a = fit_plot_loess(train,test_weekend,spans[best_span])
loess_test_rsq_weekend = a$tst_rsq
```

```r
a1 = fit_plot_loess(train,test_weekday,spans[best_span])
loess_test_rsq_weekday = a1$tst_rsq

cat ("Loess R^2 on test_weekend (model created on complete train): ", loess_test_rsq_weekend)
cat ("\nLoess R^2 on test_weekday (model created on complete train): ",
     loess_test_rsq_weekday)
```

```
## Loess R^2 on test_weekend (model created on complete train):  0.5405613
## Loess R^2 on test_weekday (model created on complete train):  0.2841138
```

```r
# Now we will train the model on train_weekend and predict on test_weekend.
# We will also train the model on train_weekday and predict on test_weekday.

a = fit_plot_loess(train_weekend,test_weekend,spans[best_span])
loess_test_rsq_weekend1B = a$tst_rsq
a$p$labels$title = paste0(a$p$labels$title, " \nWeekend")

a1 = fit_plot_loess(train_weekday,test_weekday,spans[best_span])
loess_test_rsq_weekday1B = a1$tst_rsq
a1$p$labels$title = paste0(a1$p$labels$title, " \nWeekday")

grid.arrange(a$p,a1$p,nrow=1,ncol=2)
```



```r
cat ("Loess R^2 on test_weekend (model created on train_weekend): ", loess_test_rsq_weekend1B)
cat ("\nLoess R^2 on test_weekday (model created on train_weekday): ",
     loess_test_rsq_weekday1B)
```

```
## Loess R^2 on test_weekend (model created on train_weekend):  0.7354261
## Loess R^2 on test_weekday (model created on train_weekday):  0.3546994
```

Observation: It was indeed beneficial to have separate models for weekdays and weekends.

Summary of all models

```r
cat("Poly:", poly_test_rsq)
cat("\nB. Spline:", bs_test_sq)
cat("\nN. Spline:", ns_test_rsq)
cat("\nS.Spline:", spline_test_rsq)
cat("\nLOESS:", loess_test_rsq)

cat ("\nLoess: on test_weekend (model created on complete train): ", loess_test_rsq_weekend)
cat ("\nLoess: on test_weekday (model created on complete train):", loess_test_rsq_weekday)
cat ("\nLoess: on test_weekend (model created on train_weeked): ", loess_test_rsq_weekend1B)
cat("\nLoess: on test_weekday (model created on train_weekday): ", loess_test_rsq_weekday1B)
```

```
## Poly: 0.4266038
## B. Spline: 0.4176375
## N. Spline: 0.4261959
## S.Spline: 0.425427
## LOESS: 0.4247486
## Loess: on test_weekend (model created on complete train):  0.5405613
## Loess: on test_weekday (model created on complete train): 0.2841138
## Loess: on test_weekend (model created on train_weeked):  0.7354261
## Loess: on test_weekday (model created on train_weekday):  0.3546994
```

Problem 2: Predicting Crime in the City

In this problem, the task is to build a model that can predict the per-capita crime rate in a given region of the US. The data set is provided in the files `dataset_2_train.txt` and `dataset_2_test.txt`. Each row corresponds to a region in the US: the first column contains the number of violent crimes per 100K population, and the remaining columns contain 8 attributes about the region. All numeric predictors are normalized into the range 0.00-1.00, and retain their distribution and skew (e.g. the population predictor has a mean value of 0.06 because most communities are small)

Examine the relationship between the crime rate and the individual predictors visually. Do some of the predictors have a non-linear relationship with the response variable, warranting the use of a non-linear regression model?

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 2 (i.e. basis functions x, x^2 for each predictor x)
- Regression with polynomial basis functions of degree 3 (i.e. basis functions x, x^2, x^3 for each predictor x)
- Regression with B-splines basis function on each predictor with three degrees of freedom

Part 2b: Generalized Additive Model (GAM)

Do you see any advantage in fitting an additive regression model to this 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 'PrecentageAsian' and 'PercentageUrban'. 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'`).

Part 2c: Including interaction terms

Re-fit the GAM with the following interaction terms included:

- A local regression basis function involving attributes 'Population', 'PercentageUrban' and 'MedIncome'
- A local regression basis function involving a race-related attribute and 'MedIncome'

You can retain the smoothing parameter chosen earlier for the smoothing spline basis functions, and only tune the smoothing parameter for the interaction terms. Do the interaction terms yield an improvement in the test R^2? Use a likelihood ratio test to check if the fit of these models is better than the previous GAM without interaction terms.

Hint: You may use the function `lo` for local regression basis functions.

Solution:

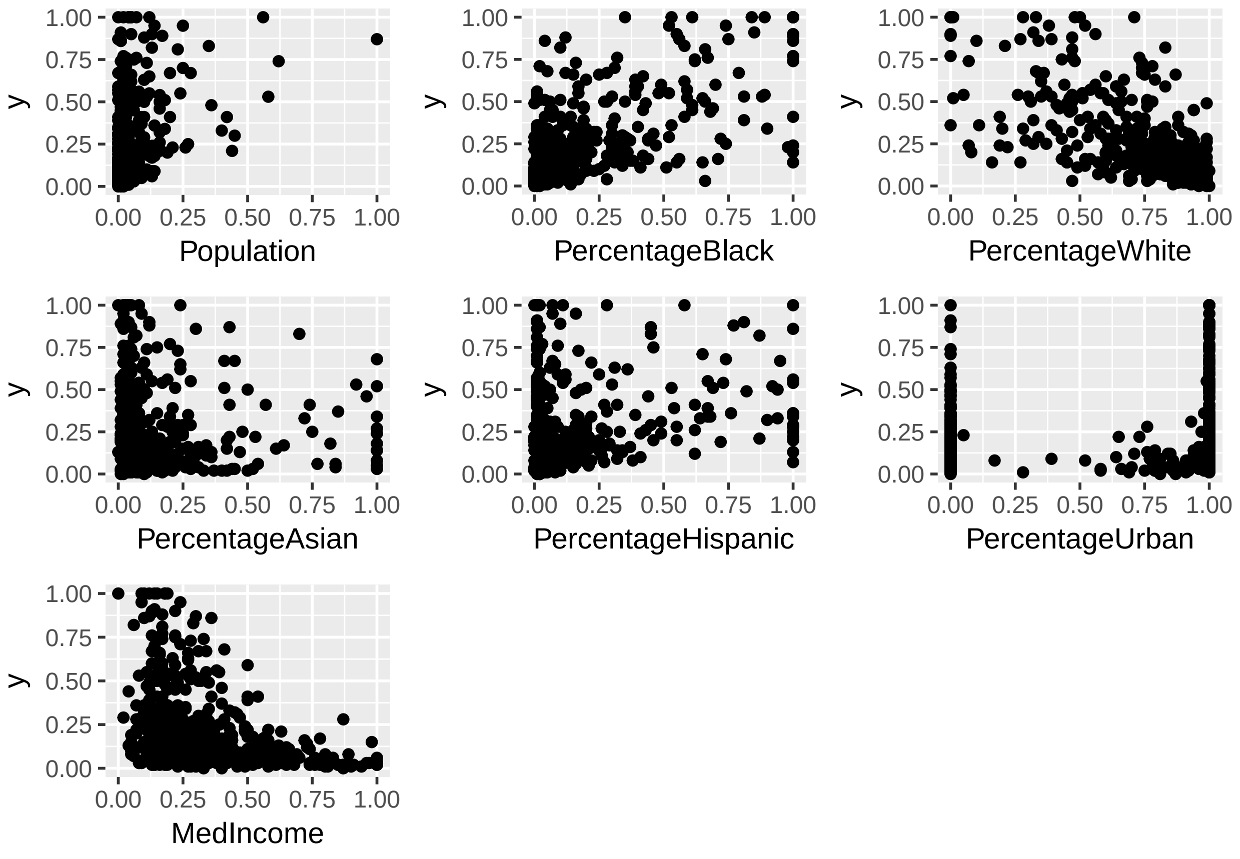
Load train and test

```r
# Load train set
train = read.csv("datasets/dataset_2_train.txt", header=TRUE, sep="")
cat("Size of train set:", dim(train), "\n")
head(train)

# Load train set
test = read.csv("datasets/dataset_2_test.txt", header=TRUE, sep="")
cat("\nSize of test set:", dim(test), "\n")
head(test)

# Header names
myvars = colnames(train)
```

```
## Size of train set: 498 8
##   ViolentCrimesPerPop Population PercentageBlack PercentageWhite
## 1                0.15       0.03            0.13            0.73
## 2                0.07       0.07            0.06            0.88
## 3                0.12       0.00            0.05            0.52
## 4                0.22       0.00            0.02            0.97
## 5                0.11       0.01            0.01            0.99
## 6                0.63       0.02            0.20            0.79
##   PercentageAsian PercentageHispanic PercentageUrban MedIncome
## 1            0.61               0.07            1.00      0.10
## 2            0.12               0.03            0.98      0.43
## 3            0.02               0.03            0.00      0.05
## 4            0.05               0.00            0.73      0.26
## 5            0.02               0.01            1.00      0.53
## 6            0.09               0.31            0.00      0.27
##
## Size of test set: 1496 8
##   ViolentCrimesPerPop Population PercentageBlack PercentageWhite
## 1                0.43       0.00            0.49            0.56
## 2                0.12       0.04            1.00            0.08
## 3                0.03       0.01            0.02            0.95
## 4                0.14       0.02            0.06            0.54
## 5                0.03       0.01            0.00            0.98
## 6                0.55       0.01            0.03            0.46
##   PercentageAsian PercentageHispanic PercentageUrban MedIncome
## 1            0.17               0.04            0.00      0.30
## 2            0.12               0.10            1.00      0.58
## 3            0.09               0.05            0.90      0.50
## 4            1.00               0.25            1.00      0.52
## 5            0.06               0.02            0.81      0.42
## 6            0.20               1.00            0.00      0.16
```

Plots of crime rate as a function of individual predictors

```
p = list()
j = 1
for (i in myvars[-1]){
  p[[j]] = ggplot() + geom_point(data=train,aes_string(i,y='ViolentCrimesPerPop')) + ylab("y")
  j = j + 1
}
require(gridExtra)
grid.arrange(p[[1]],p[[2]],p[[3]],p[[4]],p[[5]],p[[6]],p[[7]],nrow=3,ncol=3)
```



Observation: Almost all predictors have a non-linear relationship with the response variable. The race and income related attributes seem to have a strong correlation with crime rate, while surprisingly the predictor 'PercentageUrban' does not appear very informative.

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

```
# Linear regression model
# Formula
model.lin = lm(ViolentCrimesPerPop ~ ., data=train)

# Predict on train set, compute R^2
pred_train = predict(model.lin, newdata=train)
lin_trainrsq = rsq(train$ViolentCrimesPerPop, pred_train)

# Predict on test set, compute R^2
pred_test = predict(model.lin, newdata=test)
lin_testrsq = rsq(test$ViolentCrimesPerPop, pred_test)

cat(sprintf("Linear regression: Train R^2: %.3f, Test R^2: %.3f", lin_trainrsq, lin_testrsq))
```

```
## Linear regression: Train R^2: 0.618, Test R^2: 0.555
```

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

```r
# Polynomial regression model - degree 2
# Formula
formula_poly = as.formula(paste0("ViolentCrimesPerPop ~ ", paste0(myvars[2:8], collapse="+"),
                                 "+", paste0("poly(", myvars[2:8], "^2)",collapse="+")))

model.poly = lm(formula_poly, data=train)

# Predict on train set, compute R^2
pred_train = predict(model.poly, newdata=train)
poly2_trainrsq = rsq(train$ViolentCrimesPerPop, pred_train)

# Predict on test set, compute R^2
pred_test = predict(model.poly, newdata=test)
poly2_testrsq = rsq(test$ViolentCrimesPerPop, pred_test)

cat(sprintf("2-Poly regression: Train R^2: %.3f, Test R^2: %.3f", poly2_trainrsq, poly2_testrsq))
```

```
## 2-Poly regression: Train R^2: 0.633, Test R^2: 0.575
```

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

```r
# Polynomial regression model - degree 3
# Formula
formula_poly = as.formula(paste0("ViolentCrimesPerPop ~ ", paste0(myvars[2:8], collapse="+"),
                                 "+", paste0("poly(", myvars[2:8], "^2)",collapse="+"), "+",
                                 paste0("poly(", myvars[2:8], "^3)",collapse="+")))

model.poly = lm(formula_poly, data=train)

# Predict on train set, compute R^2
pred_train = predict(model.poly, newdata=train)
poly3_trainrsq = rsq(train$ViolentCrimesPerPop, pred_train)

# Predict on test set, compute R^2
pred_test = predict(model.poly, newdata=test)
poly3_testrsq = rsq(test$ViolentCrimesPerPop, pred_test)

cat(sprintf("3-Poly regression: Train R^2: %.3f, Test R^2: %.3f", poly3_trainrsq,
            poly3_testrsq))
```

```
## 3-Poly regression: Train R^2: 0.644, Test R^2: 0.573
```

Regression with B-splines basis function on each predictor with three degrees of freedom

```r
#Regression with B-splines
# Formula
formula_poly = as.formula(paste0("ViolentCrimesPerPop ~ ", paste0("bs(", myvars[2:8], ",
                                                                   df = 3)",collapse="+")))

model.bs = lm(formula_poly, data=train)
```

```
# Predict on train set, compute R^2
pred_train = predict(model.bs, newdata=train)
bs_trainrsq = rsq(train$ViolentCrimesPerPop, pred_train)

# Predict on test set, compute R^2
pred_test = predict(model.bs, newdata=test)
bs_testrsq = rsq(test$ViolentCrimesPerPop, pred_test)

cat(sprintf("Regression with BS basis functions: Train R^2: %.3f, Test R^2: %.3f",
            bs_trainrsq, bs_testrsq))
```

```
## Regression with BS basis functions: Train R^2: 0.644, Test R^2: 0.573
```

Summary of all models so far

```
cat("Test R^2:\n")
cat(sprintf("Linear regression: %.3f\n", lin_testrsq))
cat(sprintf("2-poly regression: %.3f\n", poly2_testrsq))
cat(sprintf("2-poly regression: %.3f\n", poly3_testrsq))
cat(sprintf("Regression with BS basis functions: %.3f", bs_testrsq))
```

```
## Test R^2:
## Linear regression: 0.555
## 2-poly regression: 0.575
## 2-poly regression: 0.573
## Regression with BS basis functions: 0.573
```

Part 2b: Generalized Additive Model (GAM)

Do you see any advantage in fitting an additive regression model to this 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.

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

```
fit_gam_s = function(spar_val, train, test, disp) {
  # Input:
  #   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("ViolentCrimesPerPop ~ ", paste0("s(",
                                                                    myvars[2:length(myvars)],
                                                                     ", spar = ", spar_val, ")",
                                                                     collapse="+")))
  model.gam <- gam(gam_formula, data=train)

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

  gam_trainrsq = rsq(train$ViolentCrimesPerPop, fitted(model.gam))
  gam_testrsq = rsq(test$ViolentCrimesPerPop, 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)
  }
}
```

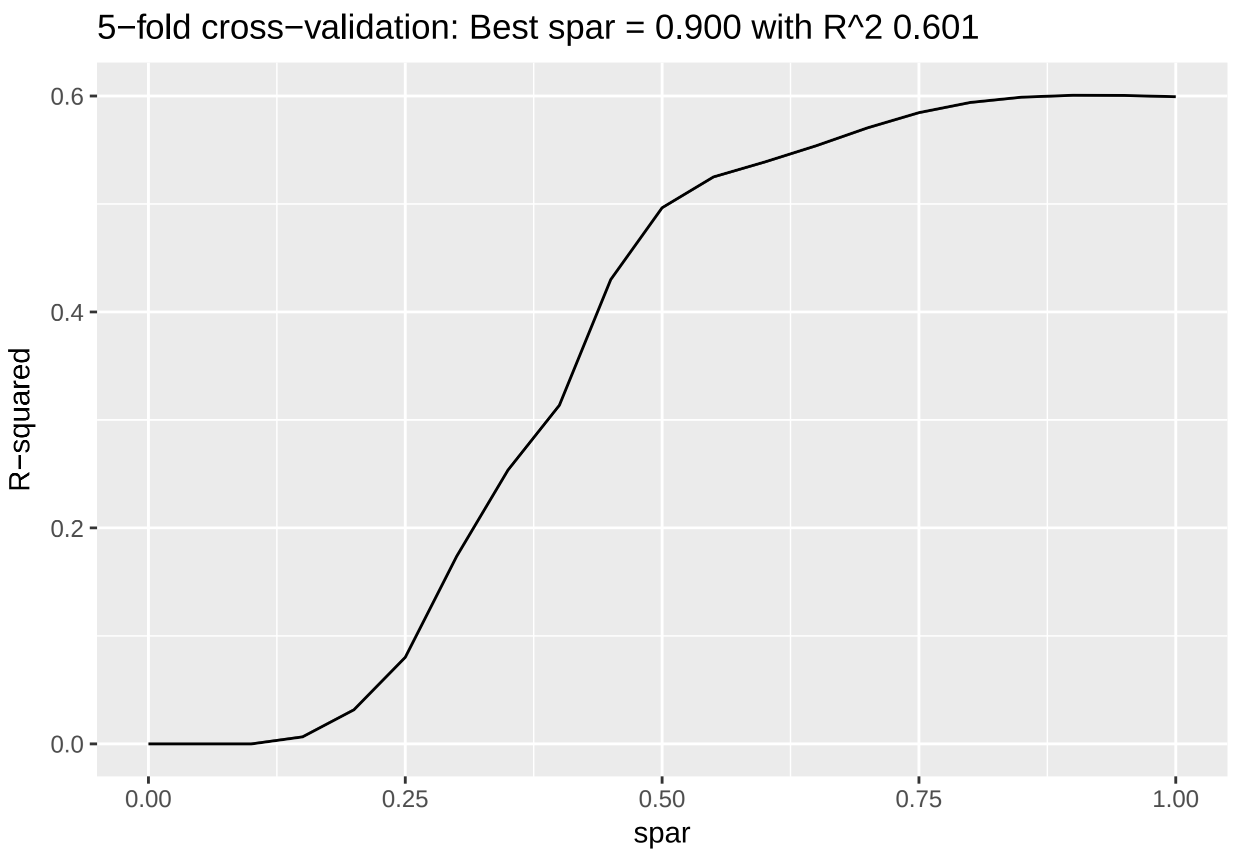
Explore models with different spar values

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

```
## GAM with smoothing spline (spar = 0.10): Train R^2: 0.898, Test R^2: 0.000
## GAM with smoothing spline (spar = 0.25): Train R^2: 0.838, Test R^2: 0.302
## GAM with smoothing spline (spar = 0.50): Train R^2: 0.726, Test R^2: 0.514
## GAM with smoothing spline (spar = 0.75): Train R^2: 0.665, Test R^2: 0.570
## GAM with smoothing spline (spar = 1.00): Train R^2: 0.630, Test R^2: 0.569
```

5-fold cross-validation to find optimal spar value

```
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)
  train$id <- sample(1:k, nrow(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(train, id %in% list[-i])
    testset <- subset(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

```r
best_spar_val = spars[best_spar]
gam_testrsq = invisible(fit_gam_s(best_spar_val, train, test, FALSE))
model.gam = invisible(fit_gam_s(best_spar_val, train, test, TRUE))
```

```
## GAM with smoothing spline (spar = 0.90): Train R^2: 0.642, Test R^2: 0.577
```

Observation: GAM yields similar test R^2 compared to the previous regression models without interaction terms, but yields a more interpretable model.

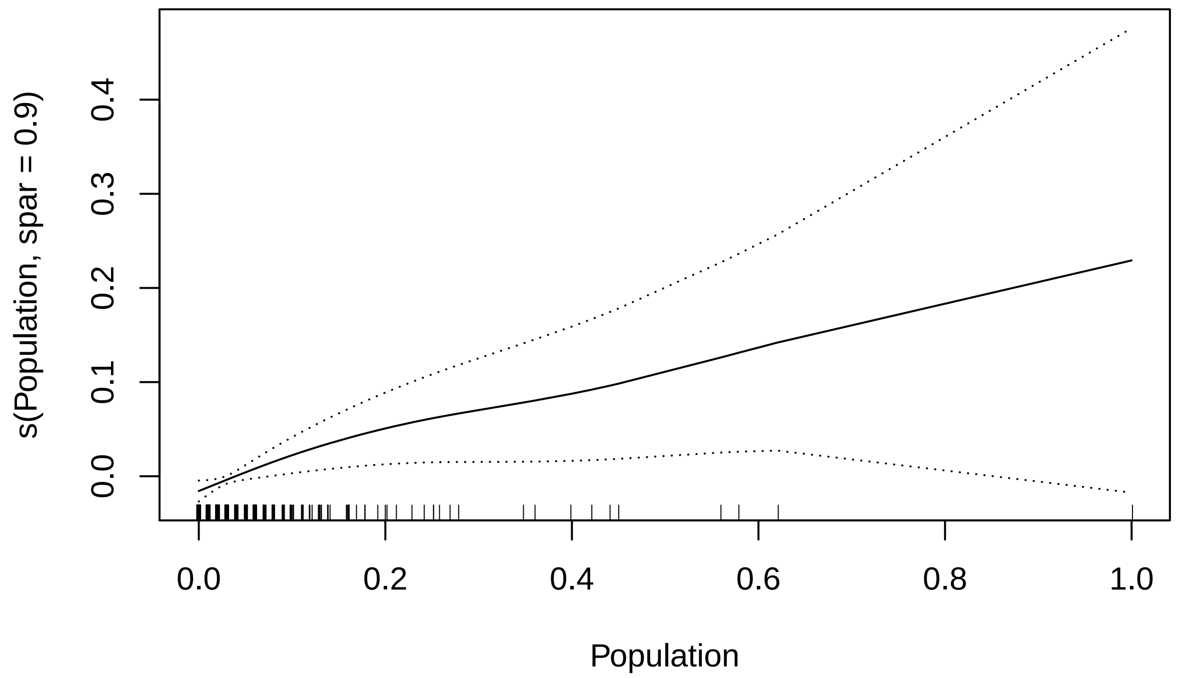
2. Plot and examine the smooth of each predictor for the fitted GAM, along with plots of upper and lower standard errors on the predictions. What are some useful insights conveyed by these plots, and by the coefficients assigned to each local model?

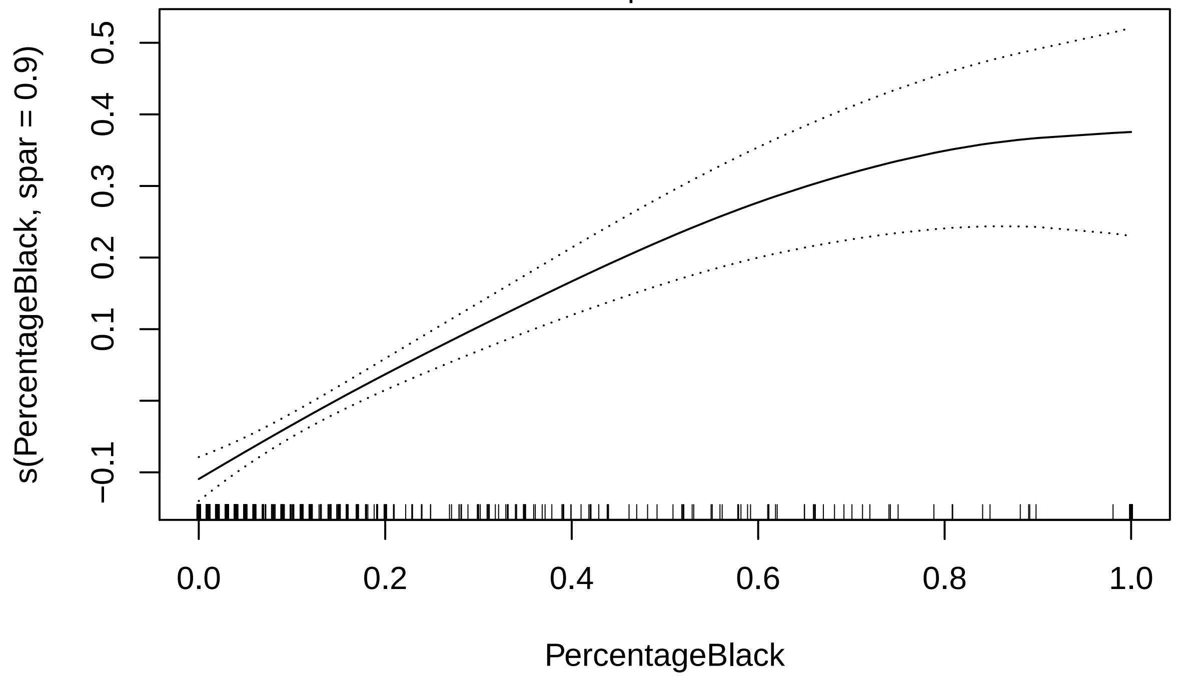
Print coefficients, visualize individual spline models

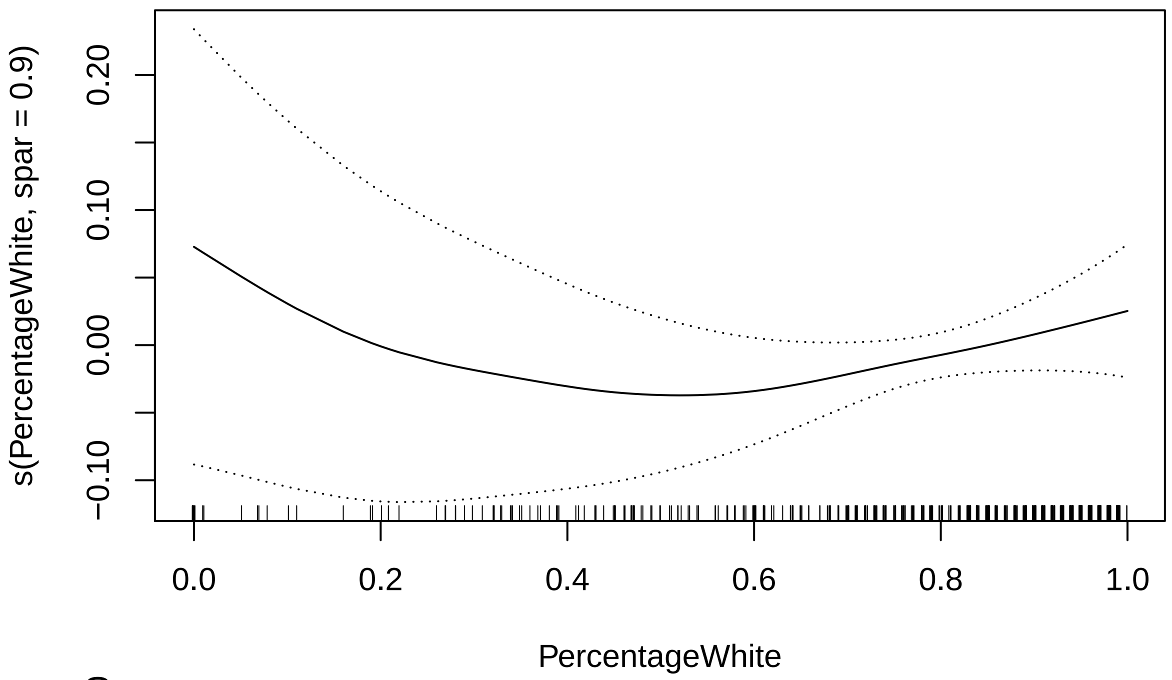
```r
coef(model.gam)
```

```
##                        (Intercept)         s(Population, spar = 0.9)
##                         0.08127651                        0.27268721
##     s(PercentageBlack, spar = 0.9)    s(PercentageWhite, spar = 0.9)
##                         0.56089584                        0.03312177
##     s(PercentageAsian, spar = 0.9) s(PercentageHispanic, spar = 0.9)
##                         0.01756880                        0.31044073
##     s(PercentageUrban, spar = 0.9)          s(MedIncome, spar = 0.9)
##                         0.07099263                       -0.22576077
```

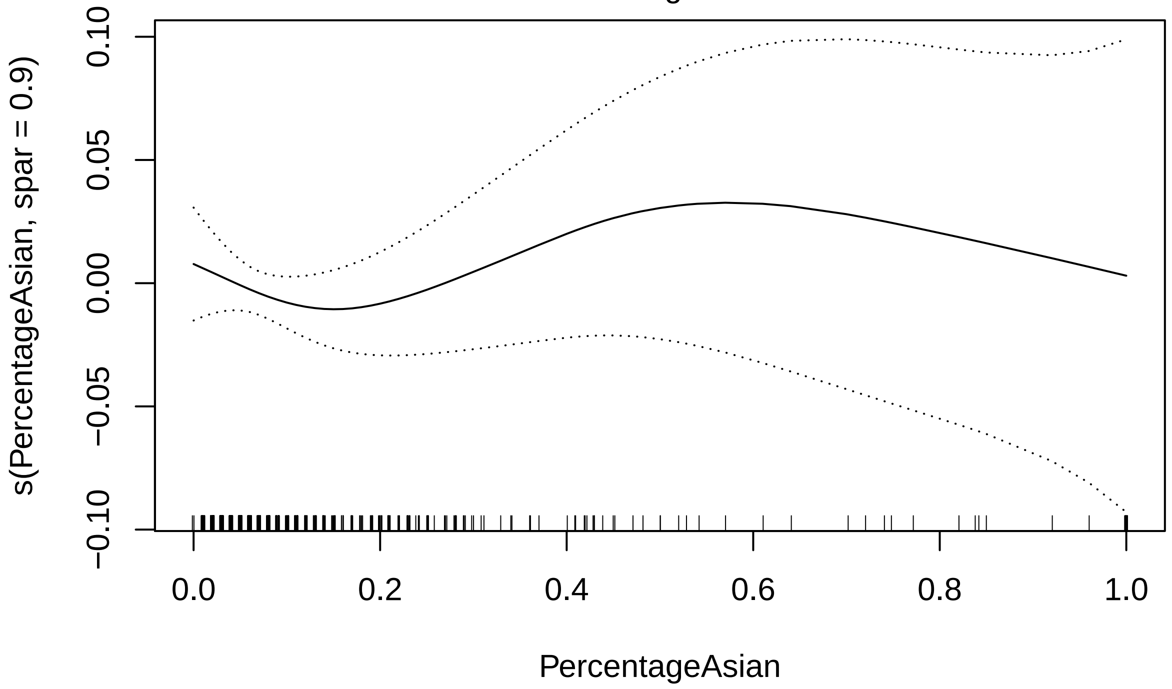
```
plot(model.gam, se=TRUE)
```



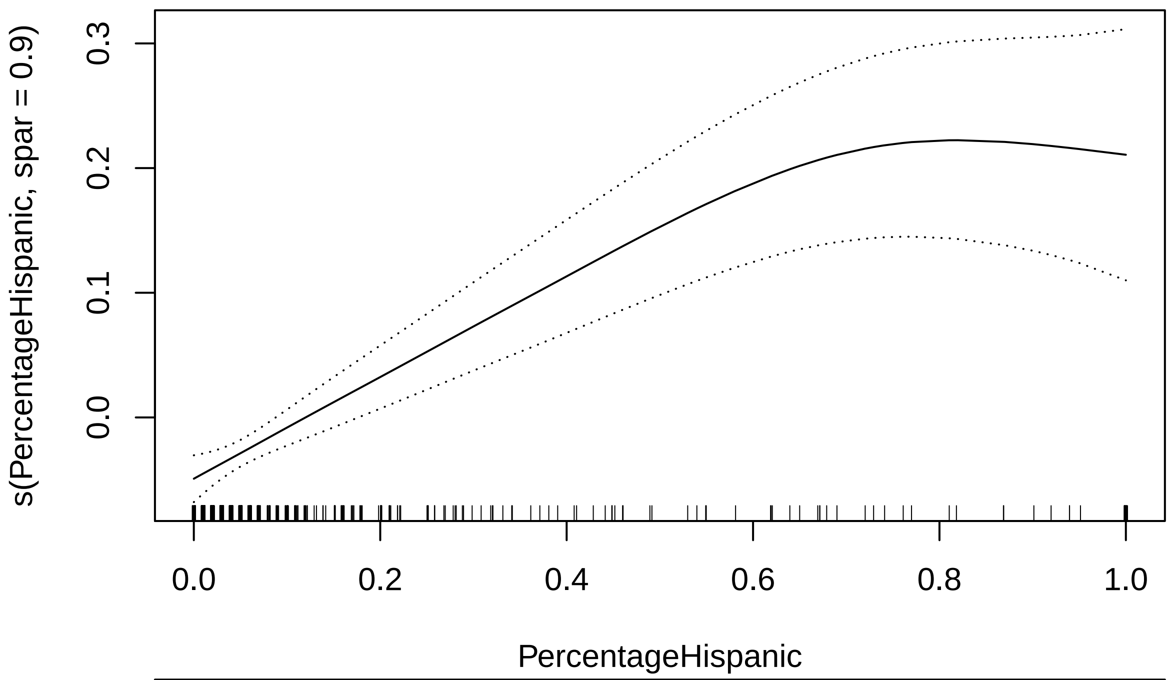
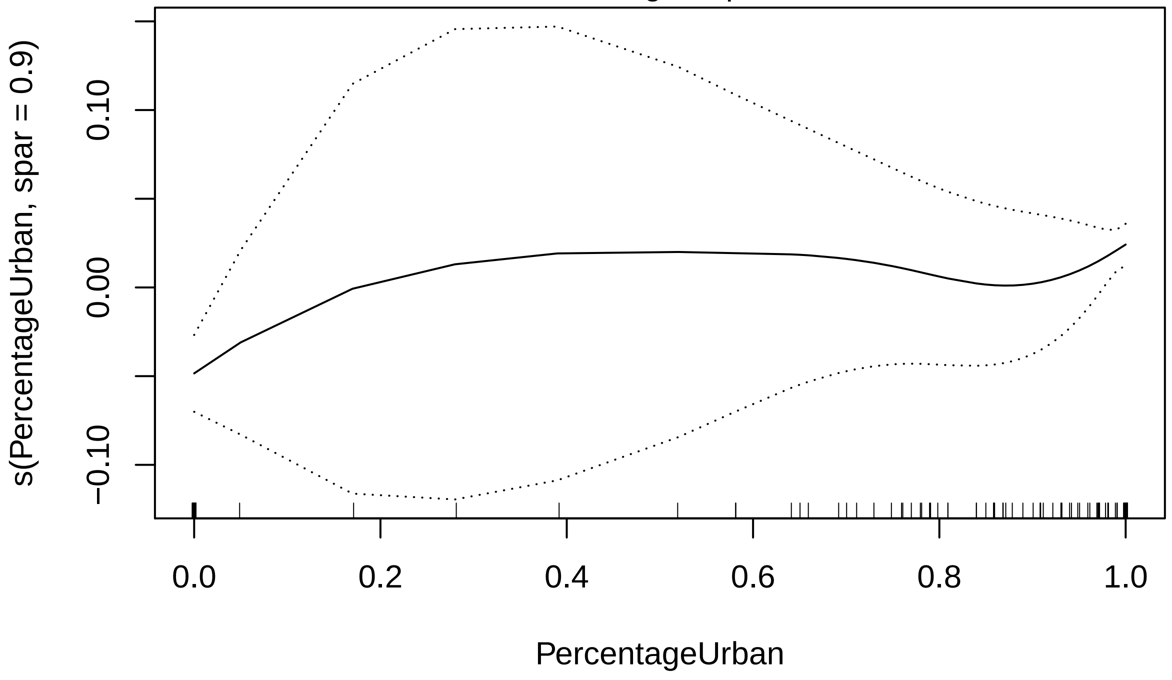


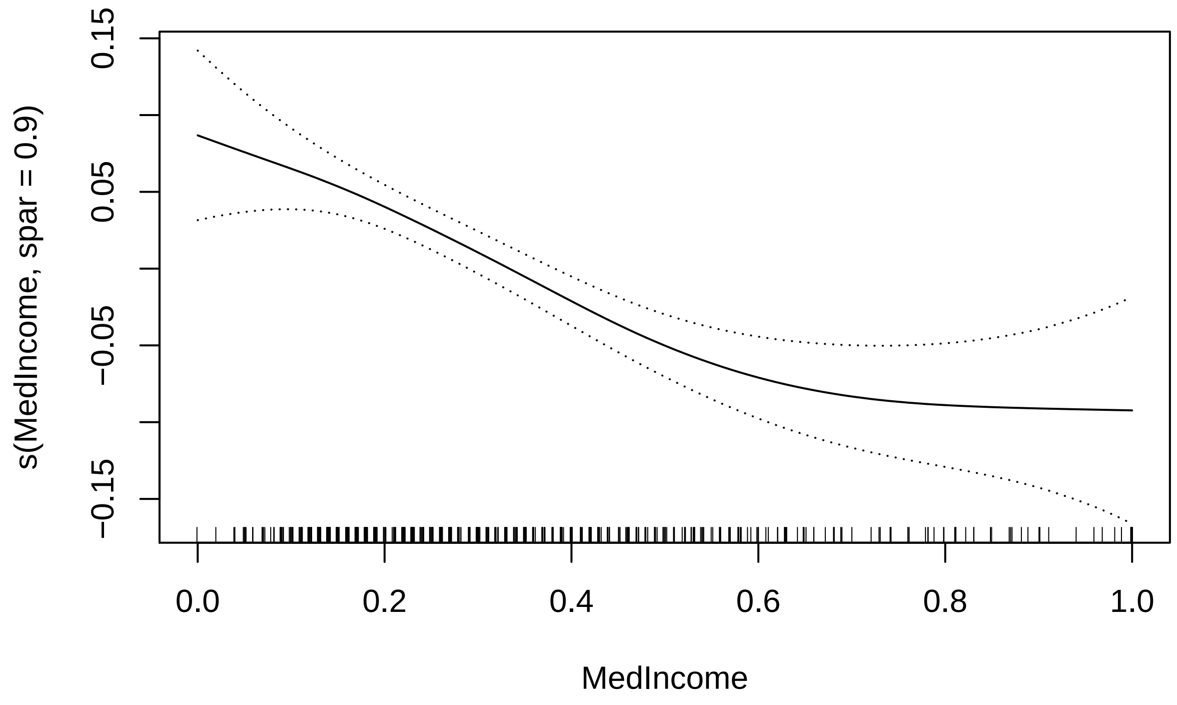


s(PercentageWhite, spar = 0.9)
−0.10
0.00
0.10
0.20
0.0
0.2
0.4
0.6
0.8
1.0
PercentageWhite



s(PercentageAsian, spar = 0.9)
−0.10
−0.05
0.00
0.05
0.10
0.0
0.2
0.4
0.6
0.8
1.0
PercentageAsian


s(PercentageHispanic, spar = 0.9)
0.0
0.1
0.2
0.3
0.0
0.2
0.4
0.6
0.8
1.0
PercentageHispanic

s(PercentageUrban, spar = 0.9)
−0.10
0.00
0.10
0.0
0.2
0.4
0.6
0.8
1.0
PercentageUrban



Observation: - As seen from initial analysis, the 'PercentageUrban' predictor is not informative. This is evident from both the large standard errors for predictions, as well as from the coeffcient for this predictor being small. - The coefficients and plots also suggest that the population predictor has a positive correlation with crime rate, while income has a negative correlation, both of which match our intuition. - Of the race-related predictors, the percentage of Asians and PercentageWhite does not have much correlation with the crime rates, while the remaining race proportions have strong correlations.

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

Likelihood ratio test to compare against linear regression

```r
anova(model.lin, model.gam, test="Chi")
```

```
## Analysis of Variance Table
##
## Model 1: ViolentCrimesPerPop ~ Population + PercentageBlack + PercentageWhite +
##     PercentageAsian + PercentageHispanic + PercentageUrban +
##     MedIncome
## Model 2: ViolentCrimesPerPop ~ s(Population, spar = 0.9) + s(PercentageBlack,
##     spar = 0.9) + s(PercentageWhite, spar = 0.9) + s(PercentageAsian,
##     spar = 0.9) + s(PercentageHispanic, spar = 0.9) + s(PercentageUrban,
##     spar = 0.9) + s(MedIncome, spar = 0.9)
##   Res.Df     RSS     Df Sum of Sq Pr(>Chi)
## 1 490.00 10.1458
## 2 476.59  9.5165 13.408    0.6293 0.003495 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Yes, GAM is better than linear regression at a significance level of 0.01.

Fit with only subset of attributes

```r
formulagam = as.formula(paste0("ViolentCrimesPerPop ~ s(Population, spar = ",best_spar_val,
                               ") + s(PercentageBlack, spar = ",best_spar_val,
                               ") + s(PercentageWhite, spar = ",best_spar_val,
                               ")  + s(PercentageHispanic, spar = ",best_spar_val,
                               ") +  s(MedIncome, spar =",best_spar_val,")"))

model.gam_ = gam(formulagam, data=train)

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

gam_trainrsq = rsq(train$ViolentCrimesPerPop, fitted(model.gam_))
gam_testrsq = rsq(test$ViolentCrimesPerPop, preds)

cat(sprintf("GAM with subset of attributes: Train R^2: %.3f, Test R^2: %.3f\n", gam_trainrsq,
            gam_testrsq))
```

```
## GAM with subset of attributes: Train R^2: 0.625, Test R^2: 0.573
```

```r
anova(model.gam_, model.gam, test="Chi")
```

```
## Analysis of Deviance Table
##
## Model 1: ViolentCrimesPerPop ~ s(Population, spar = 0.9) + s(PercentageBlack,
##     spar = 0.9) + s(PercentageWhite, spar = 0.9) + s(PercentageHispanic,
##     spar = 0.9) + s(MedIncome, spar = 0.9)
## Model 2: ViolentCrimesPerPop ~ s(Population, spar = 0.9) + s(PercentageBlack,
##     spar = 0.9) + s(PercentageWhite, spar = 0.9) + s(PercentageAsian,
##     spar = 0.9) + s(PercentageHispanic, spar = 0.9) + s(PercentageUrban,
##     spar = 0.9) + s(MedIncome, spar = 0.9)
##   Resid. Df Resid. Dev     Df Deviance  Pr(>Chi)
## 1    482.85     9.9815
## 2    476.59     9.5165 6.2595  0.46496 0.0008693 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

GAM on all predictors is better at a significance level of 0.001

Part 2C: Including interaction terms

Re-fit the GAM with the following interaction terms included:

- A local regression basis function involving attributes 'Population', 'PercentageUrban' and 'MedIncome'
- A local regression basis function involving a race-related attribute and 'MedIncome'

You can retain the smoothing parameter chosen earlier for the smoothing spline basis functions, and only tune the smoothing parameter for the interaction terms. Do the interaction terms yield an improvement in the test R^2? Use a likelihood ratio test to check if the fit of these models is better than the previous GAM without interaction terms.

Include interaction term: LOESS on Population, PercentageUrban, MedIncome (we use same spar value chosen earlier, we use CV to find best span parameter)

```r
# Supress warnings
options(warn=-1)
```

```r
#Find best span value using cross validation
crossval_gam_lo = function(span_param,k ) {
  # Input:
  #   Tuning parameter for loess in GAM: 'span_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)
  train$id <- sample(1:k, nrow(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)

  for (j 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(train, id %in% list[-j])
    testset <- subset(train, id %in% c(j))
    #run model
    str = paste0("ViolentCrimesPerPop ~ ",
                 paste0("s(", myvars[2:length(myvars)],
                        ", spar = ", best_spar_val, ")", collapse="+"),
                 "+ lo(Population, PercentageUrban, MedIncome,span=" , span_param ,  ")" )

    gam_formula = as.formula(str)
    model.gam.loess = gam(gam_formula, data=trainingset)

    pred <- predict(model.gam.loess, testset)

    # calculate R^2 on test set
    rsq_res[j] = rsq(testset$ViolentCrimesPerPop, pred)
  }

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

#spans = seq(0.1,1,by=0.1)
spans = seq(0.4,0.9,by=0.1)  ## This will speed up markdown creation

res = rep(NA, length(spans))

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

# Find best span value
best_span = which(res==max(res))
```

```
sprintf("5-fold Cross-validation: Best span is %.2f and has R^2 %.3f", spans[best_span],
        res[best_span])
```

```
## [1] "5-fold Cross-validation: Best span is 0.50 and has R^2 0.607"
```

Include interaction term: LOESS on Population, PercentageUrban, MedIncome

```
str = paste0("ViolentCrimesPerPop ~ ",
             paste0("s(", myvars[2:length(myvars)], ", spar = ", best_spar_val, ")",
                    collapse="+"), "+ lo(Population, PercentageUrban, MedIncome,span=" ,
             spans[best_span] ,  ")" )

gam_formula = as.formula(str)

model.gam1 <- gam(gam_formula, data=train)

preds = predict(model.gam1, newdata=test)

gam2_trainrsq = rsq(train$ViolentCrimesPerPop, fitted(model.gam1))
gam2_testrsq = rsq(test$ViolentCrimesPerPop, preds)

cat(sprintf("GAM with LOESS(Population, PercentageUrban, MedIncome): Train R^2: %.3f, Test
            R^2: %.3f\n", gam2_trainrsq, gam2_testrsq))
```

```
## GAM with LOESS(Population, PercentageUrban, MedIncome): Train R^2: 0.655, Test
##             R^2: 0.581
```

Include interaction term: LOESS on PercentageBlack, MedIncome (we use same spar value chosen earlier)

```
str = paste0("ViolentCrimesPerPop ~ ",
             paste0("s(", myvars[2:length(myvars)], ", spar = ", best_spar_val, ")",
                    collapse="+"), "+ lo(PercentageBlack, MedIncome,span=" , spans[best_span]
             ,  ")" )
gam_formula = as.formula(str)

model.gam2 <- gam(gam_formula, data=train)

preds = predict(model.gam2, newdata=test)

gam3_trainrsq = rsq(train$ViolentCrimesPerPop, fitted(model.gam2))
gam3_testrsq = rsq(test$ViolentCrimesPerPop, preds)

cat(sprintf("GAM with LOESS(PercentageBlack, MedIncome): Train R^2: %.3f, Test R^2: %.3f\n",
            gam3_trainrsq, gam3_testrsq))
```

```
## GAM with LOESS(PercentageBlack, MedIncome): Train R^2: 0.657, Test R^2: 0.592
```

Include both interaction terms

```
str = paste0("ViolentCrimesPerPop ~ ",
             paste0("s(", myvars[2:length(myvars)], ", spar = ", best_spar_val, ")",
                    collapse="+"),
             "+ lo(Population, PercentageUrban, MedIncome,span=", spans[best_span] , ")",
             "+ lo(PercentageBlack, MedIncome,span=" , spans[best_span] ,  ")" )

gam_formula = as.formula(str)

model.gam3 <- gam(gam_formula, data=train)

preds = predict(model.gam3, newdata=test)

gam4_trainrsq = rsq(train$ViolentCrimesPerPop, fitted(model.gam3))
gam4_testrsq = rsq(test$ViolentCrimesPerPop, preds)

cat(sprintf("GAM with LOESS(Population, PercentageUrban, MedIncome) + LOESS(PercentageBlack,
            MedIncome): Train R^2: %.3f, Test R^2: %.3f\n",
            gam4_trainrsq, gam4_testrsq))
```

```
## GAM with LOESS(Population, PercentageUrban, MedIncome) + LOESS(PercentageBlack,
##             MedIncome): Train R^2: 0.668, Test R^2: 0.594
```

Summary of models

```
cat("Summary: Test R^2:\n")
cat(sprintf("Linear regression: %.3f\n", lin_testrsq))
cat(sprintf("2-poly regression: %.3f\n", poly2_testrsq))
cat(sprintf("2-poly regression: %.3f\n", poly3_testrsq))
cat(sprintf("Regression with BS basis functions: %.3f\n", bs_testrsq))
cat(sprintf("GAM with spline basis functions: %.3f\n", gam_testrsq))
cat(sprintf("GAM with interaction terms I: %.3f\n", gam2_testrsq))
cat(sprintf("GAM with interaction terms II: %.3f\n", gam3_testrsq))
cat(sprintf("GAM with both interaction terms: %.3f", gam4_testrsq))
```

```
## Summary: Test R^2:
## Linear regression: 0.555
## 2-poly regression: 0.575
## 2-poly regression: 0.573
## Regression with BS basis functions: 0.573
## GAM with spline basis functions: 0.573
## GAM with interaction terms I: 0.581
## GAM with interaction terms II: 0.592
## GAM with both interaction terms: 0.594
```

Observation: With both interaction terms, GAM performs slightly better than all previous models.

Likelihood test to compare against previous GAM model

```
anova(model.gam, model.gam1, test="Chi")
```

```
## Analysis of Deviance Table
##
## Model 1: ViolentCrimesPerPop ~ s(Population, spar = 0.9) + s(PercentageBlack,
##     spar = 0.9) + s(PercentageWhite, spar = 0.9) + s(PercentageAsian,
##     spar = 0.9) + s(PercentageHispanic, spar = 0.9) + s(PercentageUrban,
##     spar = 0.9) + s(MedIncome, spar = 0.9)
## Model 2: ViolentCrimesPerPop ~ s(Population, spar = 0.9) + s(PercentageBlack,
##     spar = 0.9) + s(PercentageWhite, spar = 0.9) + s(PercentageAsian,
##     spar = 0.9) + s(PercentageHispanic, spar = 0.9) + s(PercentageUrban,
##     spar = 0.9) + s(MedIncome, spar = 0.9) + lo(Population, PercentageUrban,
##     MedIncome, span = 0.5)
##   Resid. Df Resid. Dev     Df Deviance Pr(>Chi)
## 1    476.59     9.5165
## 2    469.82     9.1617 6.7753  0.35488 0.009652 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(model.gam, model.gam2, test="Chi")
```

```
## Analysis of Deviance Table
##
## Model 1: ViolentCrimesPerPop ~ s(Population, spar = 0.9) + s(PercentageBlack,
##     spar = 0.9) + s(PercentageWhite, spar = 0.9) + s(PercentageAsian,
##     spar = 0.9) + s(PercentageHispanic, spar = 0.9) + s(PercentageUrban,
##     spar = 0.9) + s(MedIncome, spar = 0.9)
## Model 2: ViolentCrimesPerPop ~ s(Population, spar = 0.9) + s(PercentageBlack,
##     spar = 0.9) + s(PercentageWhite, spar = 0.9) + s(PercentageAsian,
##     spar = 0.9) + s(PercentageHispanic, spar = 0.9) + s(PercentageUrban,
##     spar = 0.9) + s(MedIncome, spar = 0.9) + lo(PercentageBlack,
##     MedIncome, span = 0.5)
##   Resid. Df Resid. Dev     Df Deviance Pr(>Chi)
## 1    476.59     9.5165
## 2    470.15     9.1222 6.4464  0.39436 0.003358 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(model.gam, model.gam3, test="Chi")
```

```
## Analysis of Deviance Table
##
## Model 1: ViolentCrimesPerPop ~ s(Population, spar = 0.9) + s(PercentageBlack,
##     spar = 0.9) + s(PercentageWhite, spar = 0.9) + s(PercentageAsian,
##     spar = 0.9) + s(PercentageHispanic, spar = 0.9) + s(PercentageUrban,
##     spar = 0.9) + s(MedIncome, spar = 0.9)
## Model 2: ViolentCrimesPerPop ~ s(Population, spar = 0.9) + s(PercentageBlack,
##     spar = 0.9) + s(PercentageWhite, spar = 0.9) + s(PercentageAsian,
##     spar = 0.9) + s(PercentageHispanic, spar = 0.9) + s(PercentageUrban,
##     spar = 0.9) + s(MedIncome, spar = 0.9) + lo(Population, PercentageUrban,
##     MedIncome, span = 0.5) + lo(PercentageBlack, MedIncome, span = 0.5)
##   Resid. Df Resid. Dev     Df Deviance  Pr(>Chi)
## 1    476.59     9.5165
## 2    463.37     8.8235 13.222    0.693 0.0005868 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Observation: Interaction term I & II are better at a significance level of 0.01. Both interaction terms together is better at a significance level of 0.001.