Homework 1: Smoothers and Generalized Additive Models

Harvard CS 109B, Spring 2017

Danqing Wang

2/5/2017

Problem 1

Data Visualization

We import the data files into the `train` and `test` datasets and examine their contents.

```
# Import data
train <- read.csv("./CS109b-hw1-datasets/dataset_1_train.txt", sep = ",")
test <- read.csv("./CS109b-hw1-datasets/dataset_1_test.txt", sep = ",")

# Check structure of data
str(train)
```

```
## 'data.frame':    1125 obs. of  3 variables:
##  $ TimeMin    : int  57 68 182 298 363 395 483 501 509 514 ...
##  $ DayOfWeek  : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ PickupCount: int  111 95 95 75 35 30 15 13 14 15 ...
```

```
str(test)
```

```
## 'data.frame':    1125 obs. of  3 variables:
##  $ TimeMin    : int  4 5 96 114 201 231 244 334 375 397 ...
##  $ DayOfWeek  : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ PickupCount: int  51 57 116 95 98 101 100 36 32 34 ...
```

```
# Check for any  missing values
sum(is.na(train))
```

```
## [1] 0
```

```
sum(is.na(test))
```

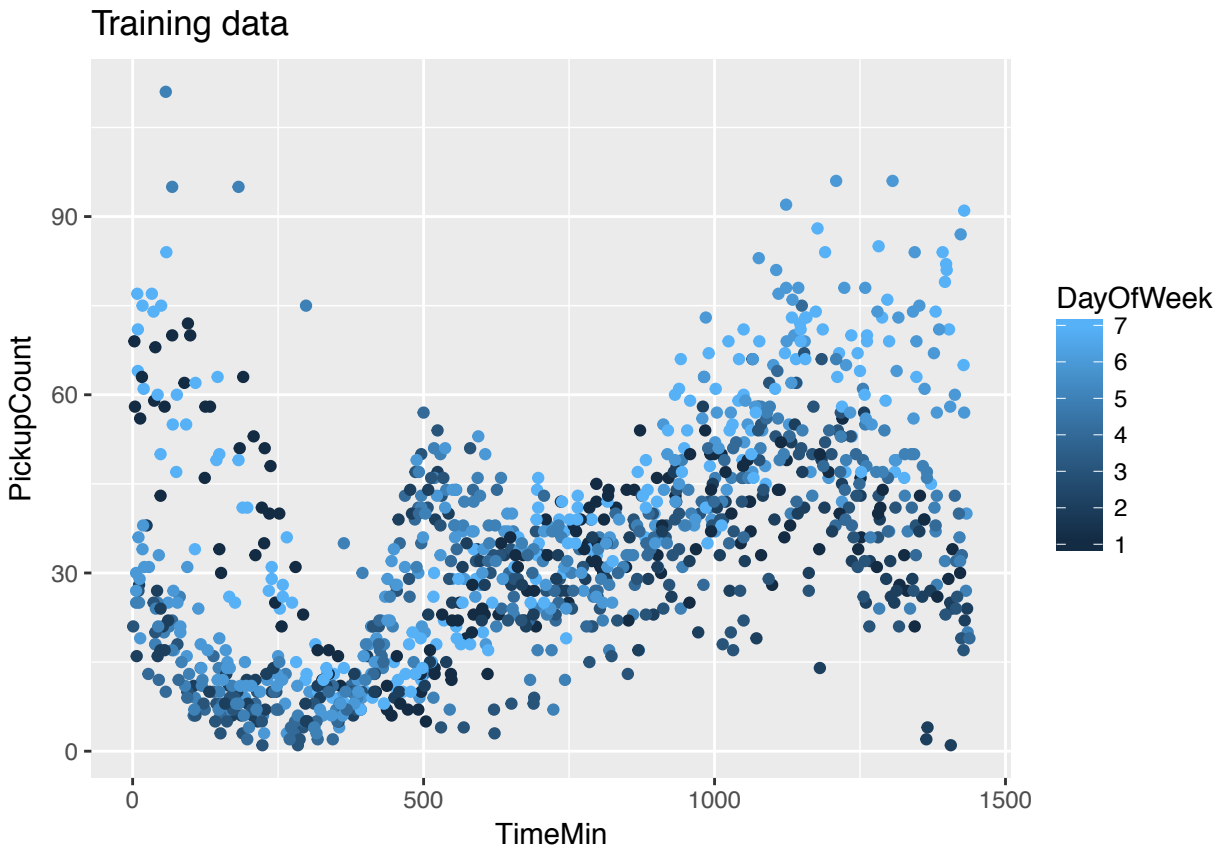
```
## [1] 0
```

We then visualize the data of the two datasets. It looks like the `PickupCount` varies by time during the day but show similar trends in the train and test datasets. The trend of `PickupCount` are also different on different days of the week.

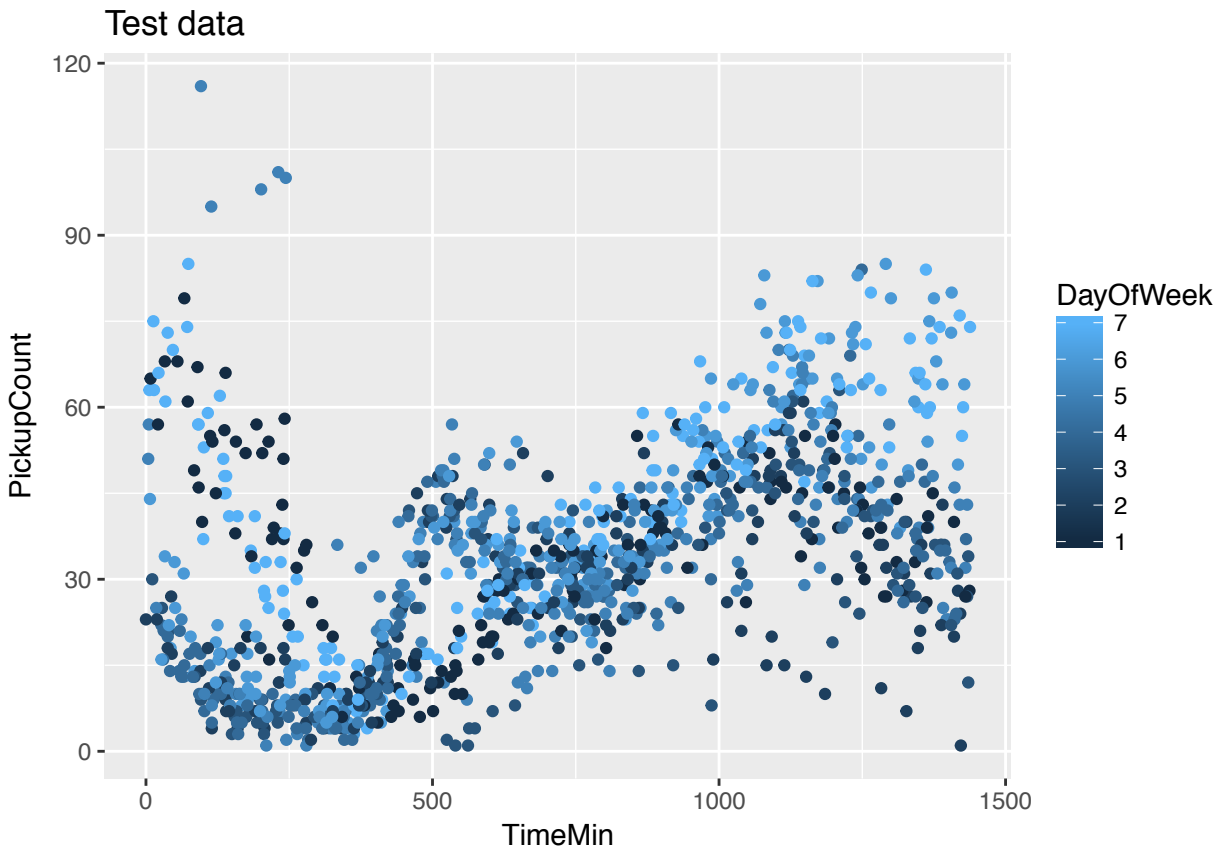
It makes intuitive sense that: - there are more taxi pickups at rush hours of the day (early morning, evening, midnight), and - the taxi pick up counts on weekdays differ from that on weekends. Since people do not go to work on the weekends, there is less pick ups during the morning rush hour, and people tend to spend their nights out on weekeds, we observe from the data sets that there are more pickups on weekend nights.

```
library(ggplot2)

ggplot(train, mapping = aes(x = TimeMin, y = PickupCount)) +
```

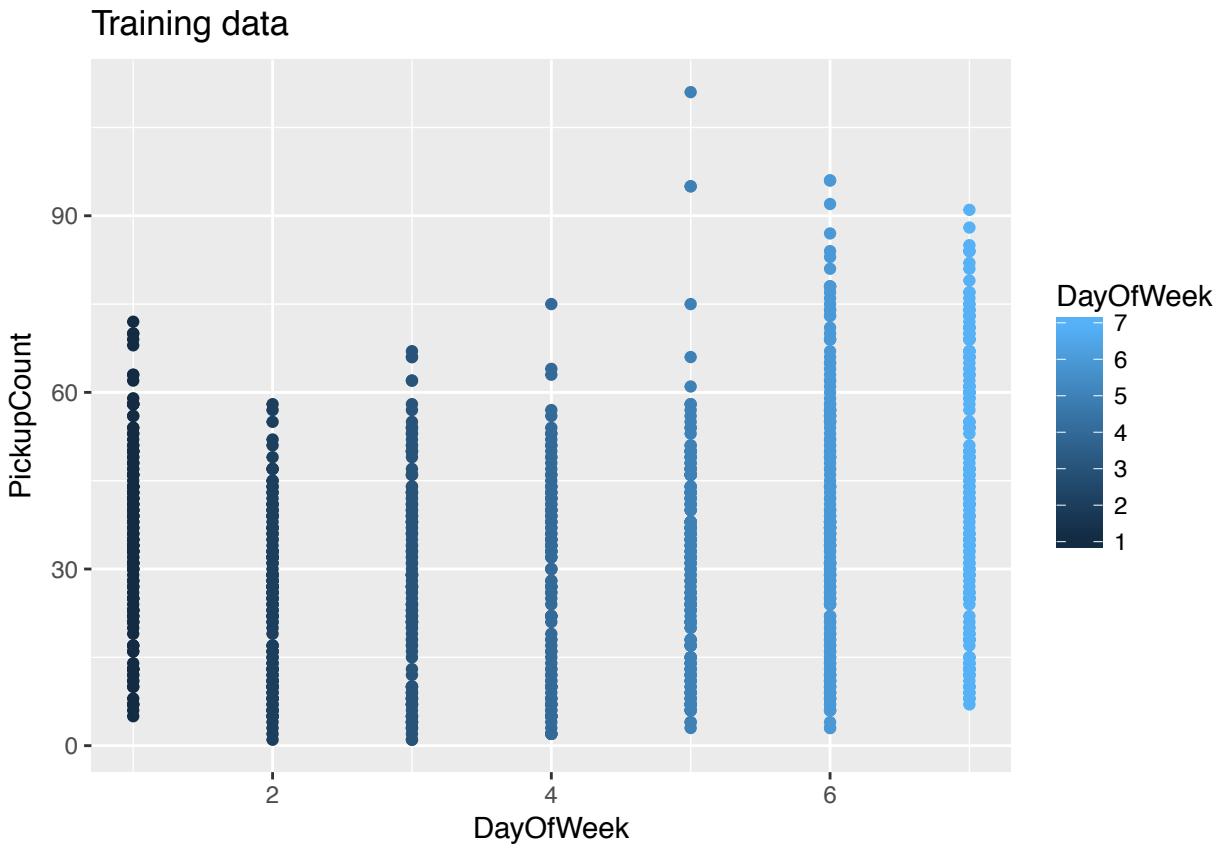
```
  geom_point(aes(color = DayOfWeek))+
  ggtitle('Training data')
```



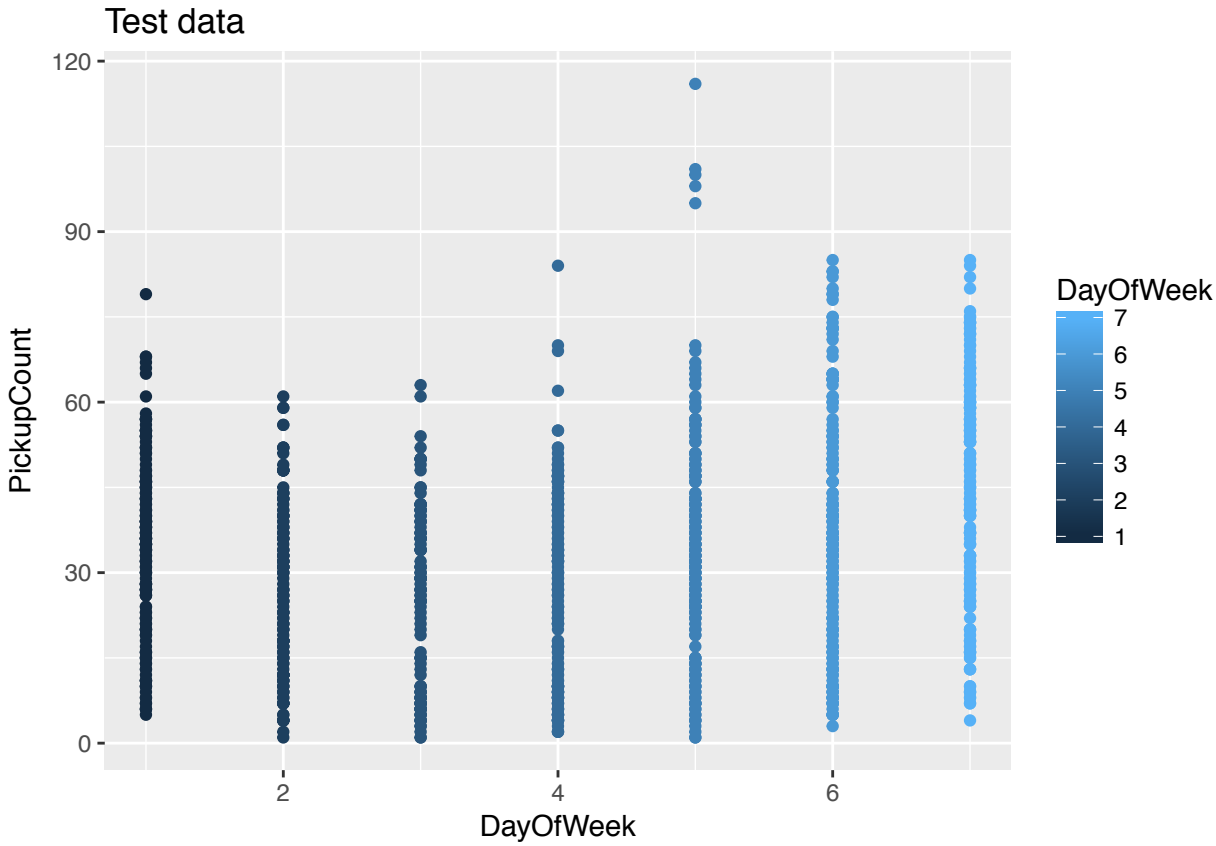

```
ggplot(test, mapping = aes(x = TimeMin, y = PickupCount)) +
  geom_point(aes(color = DayOfWeek)) +
  ggtitle('Test data')
```




```
ggplot(train, aes(x = DayOfWeek, y = PickupCount)) +
geom_point(aes(color=DayOfWeek))+
  ggtitle('Training data')
```




```
ggplot(test, aes(x = DayOfWeek, y = PickupCount)) +
geom_point(aes(color=DayOfWeek))+
  ggtitle('Test data')
```



Part 1a: Regression Model with different basis function

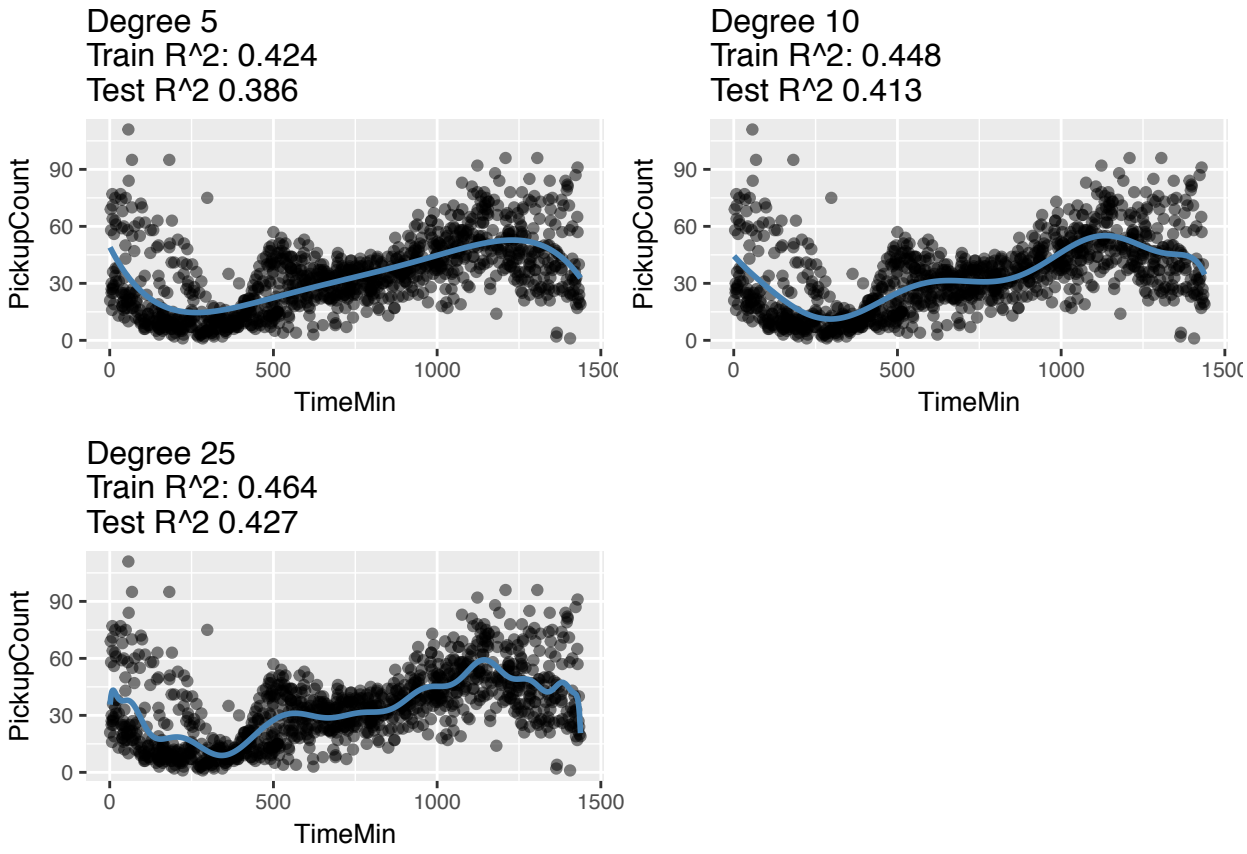
1. Regression models with different basis functions:

- Simple polynomials with degree 5, 10 and 25

```
### Function to compute R^2 for observed and predicted responses
rsq = function(y, predict) {
  tss = sum((y - mean(y))^2, na.rm = TRUE)
  rss = sum((y-predict)^2, na.rm = TRUE)
  r_squared = 1 - rss/tss
  # return(r_squared)
  return(round(r_squared, 3))
}
```

```
train_poly <- train
test_poly <- test

# Function that trains a polynomila model based on the degree specified
# and use the model to predict on the test set, store the predicted value as a new column
# and calculates the r2 value of both the train and the test datasets
# returns a graph to plot
poly_pred = function(d){
  # Train Model
  mod.poly <- lm(PickupCount ~ poly(TimeMin, degree = d), data = train)
```

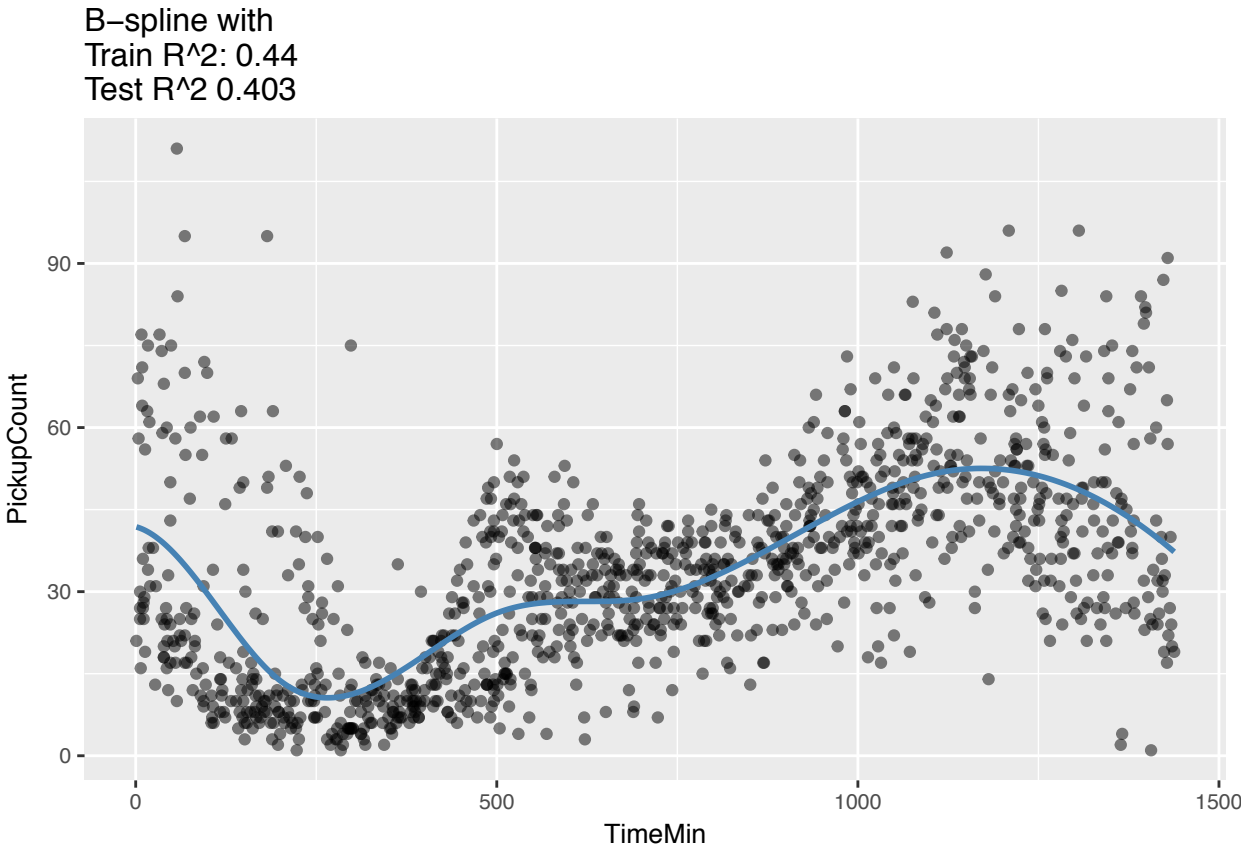
```
  train_poly <- transform(train_poly, pred.poly = predict(mod.poly))
  train.poly.r2 <- rsq(train_poly$PickupCount, train_poly$pred.poly)

  # Predict on Test dataset
  test_poly <- transform(test_poly, pred.poly = predict(mod.poly, newdata = test))
  test.poly.r2 <- rsq(test_poly$PickupCount, test_poly$pred.poly)

  # Visualization
  p = ggplot(train_poly, aes(x = TimeMin, y = PickupCount))+
    geom_point(alpha = 0.5)+
    geom_line(aes(y = pred.poly), color = 'steelblue', size = 1)+
    ggtitle(paste('Degree', d, '\nTrain R^2:', train.poly.r2, '\nTest R^2',
                  test.poly.r2))+
    theme(text = element_text(size=10))
  return(p)
}

# Visualization for Polynomial degrees 5, 10, and 25
degrees = c(5, 10, 25)

p1 <- poly_pred(degrees[1])
p2 <- poly_pred(degrees[2])
p3 <- poly_pred(degrees[3])

library(gridExtra)
grid.arrange(p1, p2, p3, ncol = 2)
```



As we increase the degree of polynomial from 5, 10, to 25, both the training and the test R^2 increases. The increase in test R^2 indicates that we have not overfitted our data. However, when we are dealing with other datasets, we need to be careful when dealing with polynomials as higher degree polynomials may overfit the training data.

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

```
train_bs <- train
test_bs <- test

library(splines)
# Train model
mod.bs <- lm(PickupCount ~ bs(TimeMin, knots = c(200, 500, 700, 1100)), data = train)

# Train dataset
train_bs <- transform(train_bs, pred.bs = predict(mod.bs))
train.bs.r2 <- rsq(train_bs$PickupCount, train_bs$pred.bs)

# Test dataset
test_bs <- transform(test_bs, pred.bs = predict(mod.bs, newdata = test))
test.bs.r2 <- rsq(test_bs$PickupCount, test_bs$pred.bs)

# Visualization
ggplot(train_bs, aes(x = TimeMin, y = PickupCount))+
  geom_point(alpha = 0.5)+
  geom_line(aes(y = pred.bs), color = 'steelblue', size = 1)+
  ggtitle(paste('B-spline with \nTrain R^2:',
                train.bs.r2, '\nTest R^2', test.bs.r2))+
  theme(text = element_text(size=10))
```



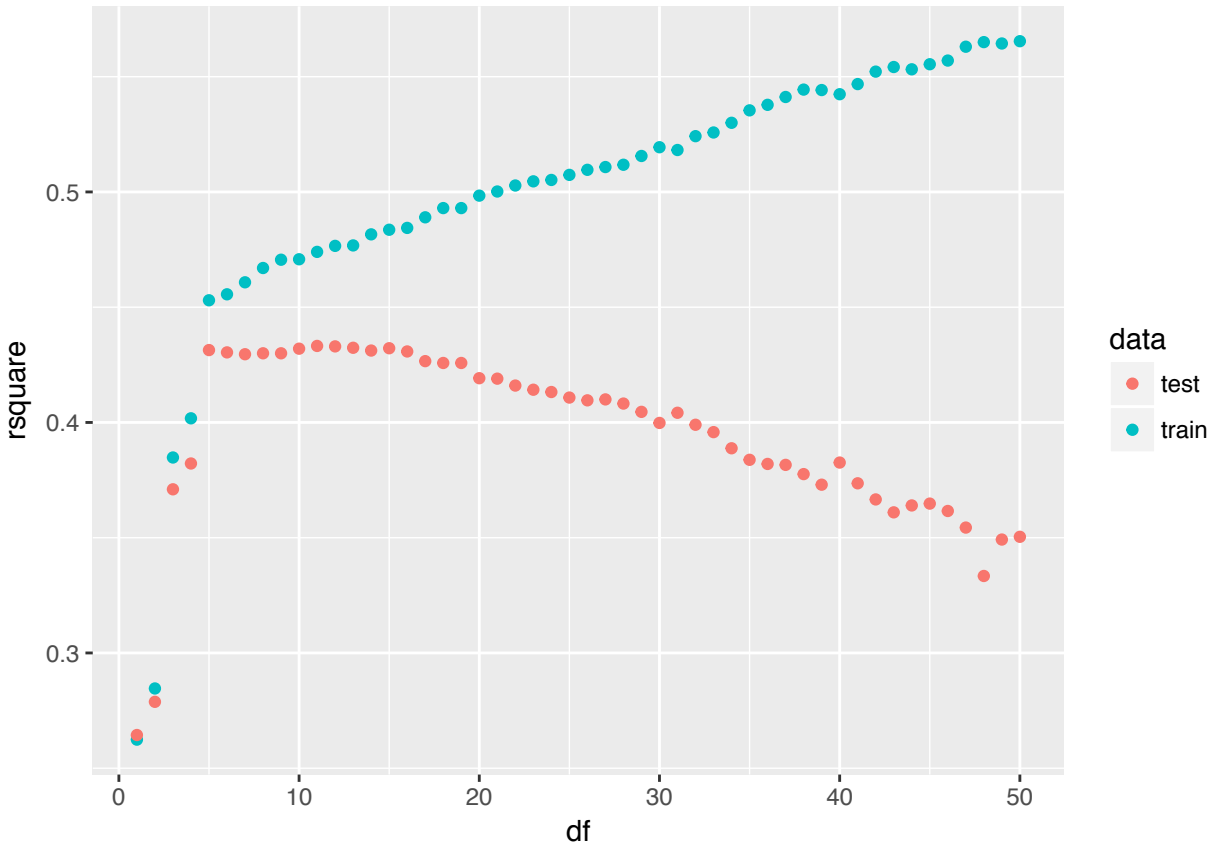
The number of knots defines the internal breakpoints that define the spline. Here by visual inspection, we notice that there seem to be a turning point at `TimeMin` of 200, 500, 700, and 1100, and inbetween these points, there seem to be a well-defined relationship. We input these numbers as the knots.

- Nature cubic spline

```
# Function to calculate R^2 of test set
rsq_pred = function(model, data, y) {
  y <- data[[y]]
  predict <- predict(model, newdata = data)
  tss = sum((y - mean(y))^2, na.rm = TRUE)
  rss = sum((y-predict)^2, na.rm = TRUE)
  rsq_ = max(0, 1 - rss/tss)
  return(round(rsq_, 3))
}
```

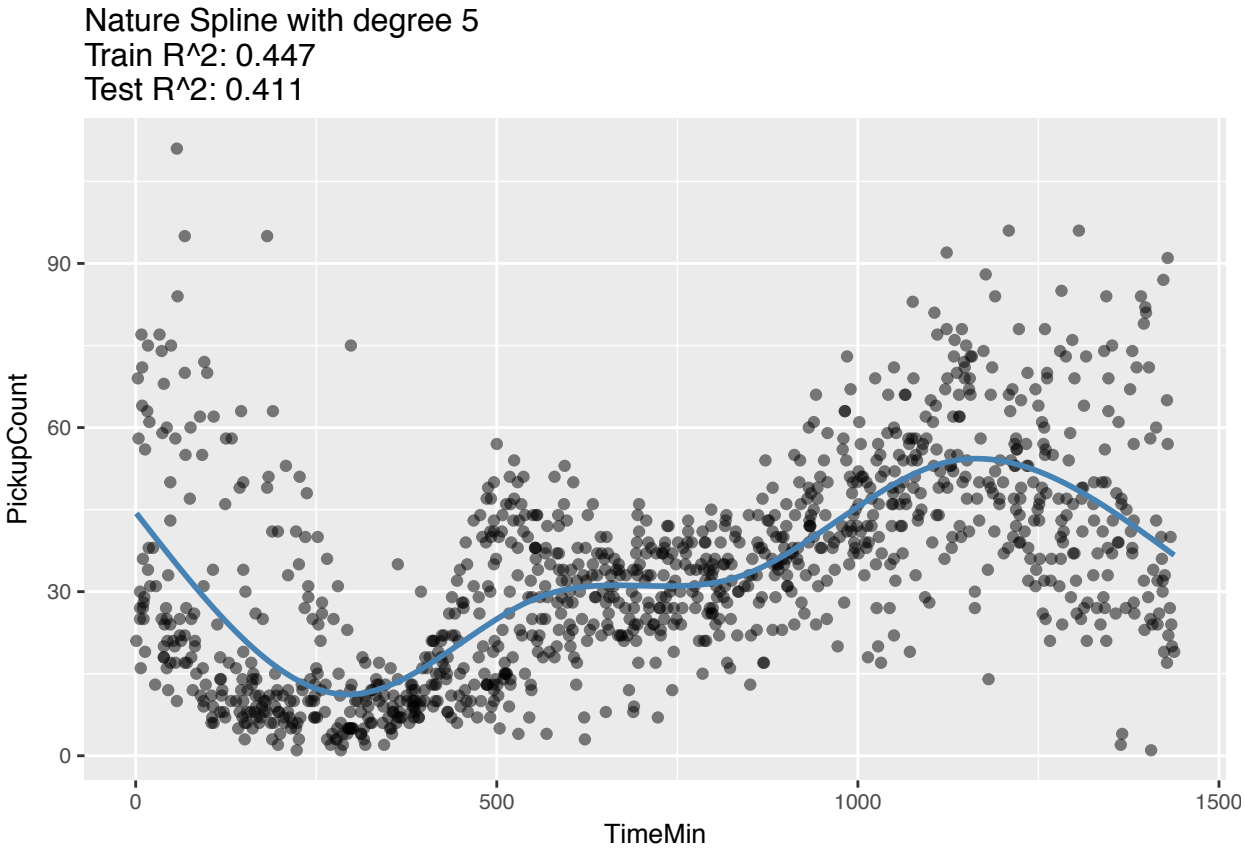
```
## create 5 partitions
train$splits <- cut(sample(1:nrow(train),
                           nrow(train)),
                    breaks = 5,
                    labels = FALSE)

#Next, define a function to fit a model with given df and calculate R-square.

model.performance <- function(df, train, test) {
  mod <- lm(PickupCount ~ ns(TimeMin, df = df), data = train)
```

```
  c(train.r2 = rsq_pred(mod, train, "PickupCount"),
    test.r2 = rsq_pred(mod, test, "PickupCount"))
}

## iterate over the splits, holding each one out as the test set.
dfs <- 1:50

perform.5fold <- lapply(unique(train$splits), function(split) {
  train_select <- train$split == split
  ns.train <- train[train_select, ]
  ns.test <- train[-train_select, ]
  data.frame(t(sapply(dfs, model.performance,
                      train = ns.train,
                      test = ns.test)),
             df = dfs)
}
)

## collect the k sets of model statistics in a data.frame
perform.5fold <- do.call(rbind, perform.5fold)

## aggregate across the k sets, averaging model statistics for each df
perform.5fold <- lapply(split(perform.5fold, perform.5fold$df),
                        function(x) {
                          data.frame(rsquare = c(mean(x$train.r2),
                                                 mean(x$test.r2)),
                                     data = c("train", "test"),
                                     df = unique(x$df))
                        })

## collect the results
perform.5fold <- do.call(rbind, perform.5fold)

## plot the results
ggplot(perform.5fold, aes(x = df, y = rsquare, color = data)) +
  geom_point()
```



From the plot, it looks like df = 5 gives the maximum R^2. We predict on the test datset with df = 5 in the following:

```
train_ns <- train
test_ns <- test

# Training model with df = 5
mod.ns5 <- lm(PickupCount ~ ns(TimeMin, df = 5), data = train)

# Train
train_ns5 <- transform(train_ns, pred.ns5 = predict(mod.ns5))
train.ns5.r2 <- rsq(train_ns5$PickupCount, train_ns5$pred.ns5)

# Test
test_ns5 <- transform(test_ns, pred.ns5 = predict(mod.ns5, newdata = test))
test.ns5.r2 <- rsq(test_ns5$PickupCount, test_ns5$pred.ns5)

ggplot(train_ns5, aes(x = TimeMin, y = PickupCount))+
  geom_point(alpha = 0.5)+
  geom_line(aes(y = pred.ns5), color = 'steelblue', size = 1)+
  ggtitle(paste('Nature Spline with degree 5 \nTrain R^2:',
                train.ns5.r2, '\nTest R^2:', test.ns5.r2))+
  theme(text = element_text(size=10))
```

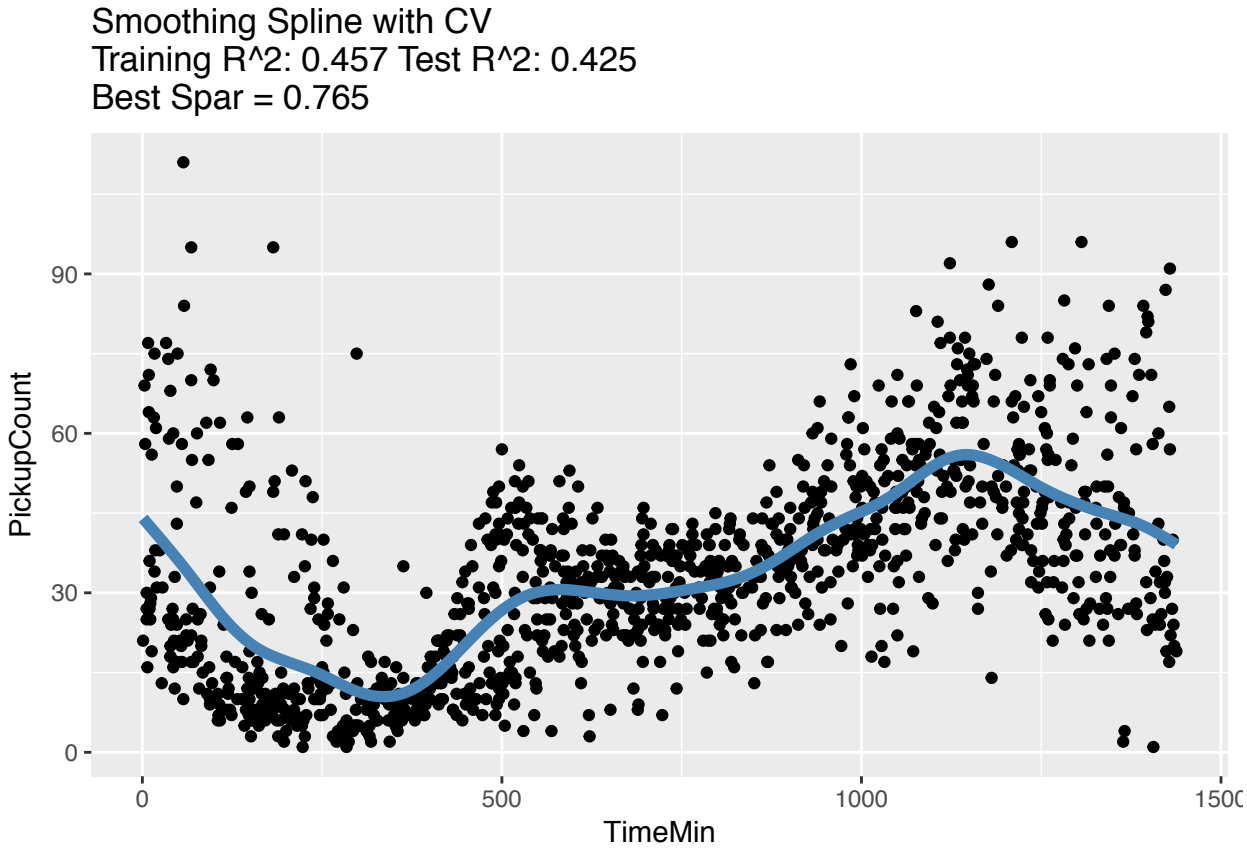


The degree of freedom defines the flexibility of the spline, it is equivalent to knots + 1. With a higher degree of freedom, the model predicts the training set very accurately but predicts the test set with decreasing rate of accuracy. This is beause we are overfitting the data to the training set. With a df as high as the number of data points, we would literally be fitting for a model that joins up each data point, which is definitely an overfit. By doing 5-fold cross-validation, we find out what is the optimal df to use here.

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

```
train_sp <- train
test_sp <- test

# Build model
fit.sp <- smooth.spline(train$PickupCount ~ train$TimeMin, cv = TRUE) #what's ordinary (TRUE) or 'gener

train_sp <- transform(train_sp, pred.sp = fitted(fit.sp))
test_sp <- transform(test_sp, pred.sp = predict(fit.sp, test$TimeMin)$y)

# Calculate R^2
train_sp.r2 <- rsq(train$PickupCount, fitted(fit.sp))
test_sp.r2 <- rsq(test$PickupCount, predict(fit.sp, test$TimeMin)$y)

ggplot(train_sp, aes(x = TimeMin, y = PickupCount))+
  geom_point()+
  geom_line(aes(y = pred.sp), color = 'steelblue', size = 2)+
  ggtitle(paste('Smoothing Spline with CV \nTraining R^2:', train_sp.r2,
```

```
              'Test R^2:', test_sp.r2,
              '\nBest Spar =', round(fit.sp$spar,3)))
```



It looks likes the optimal `spar = 0.765`. `Spar` is a smoothing parameter that takes on a value between 0 and 1. It indicates a window of values of TimeMin that we take each time in computing the mean of PickupCount within that window. The larger the neighborhood over which averages of PickupCount are taken, the smoother the estimated function; the smaller the neighborhood, the more jagged the estimated function.

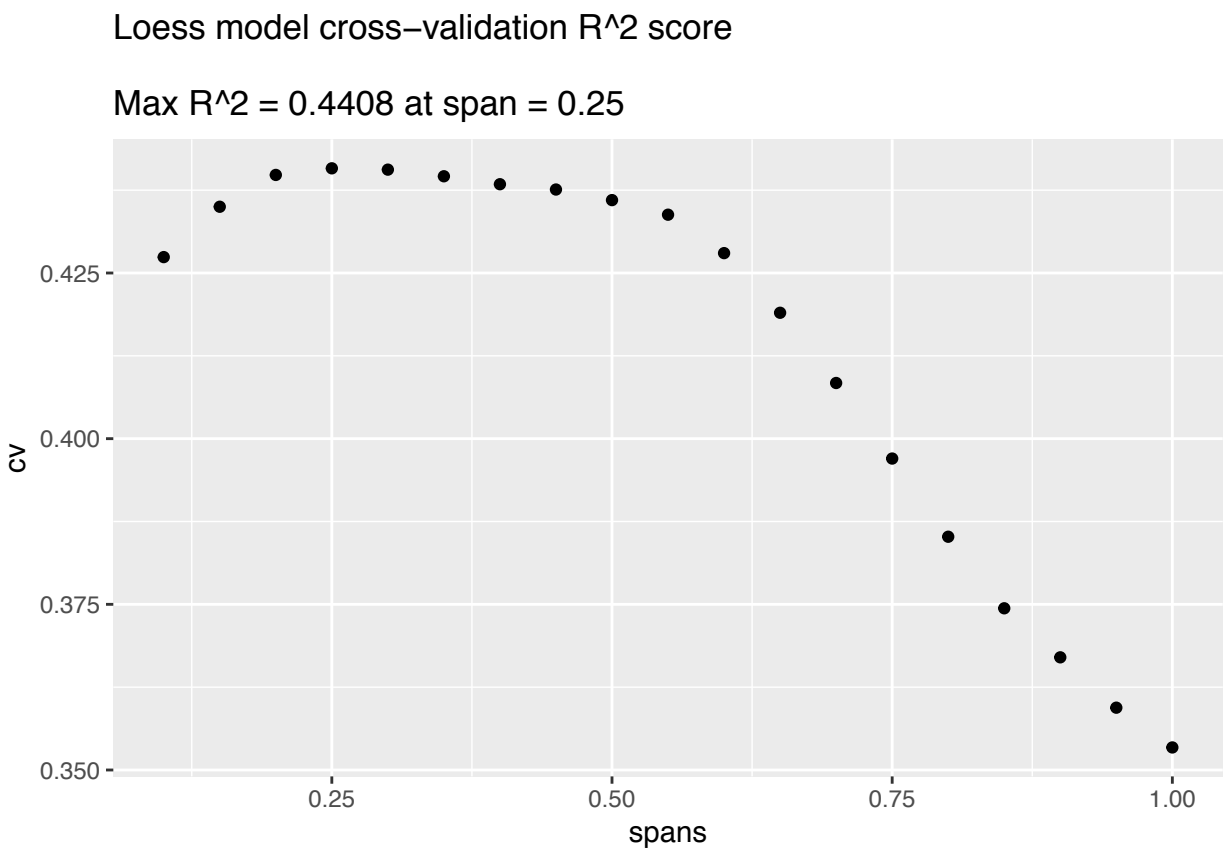
3. Locally-weighted regression model with the span parameter chosen by cross-validation on the training

```
### 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)
}
```

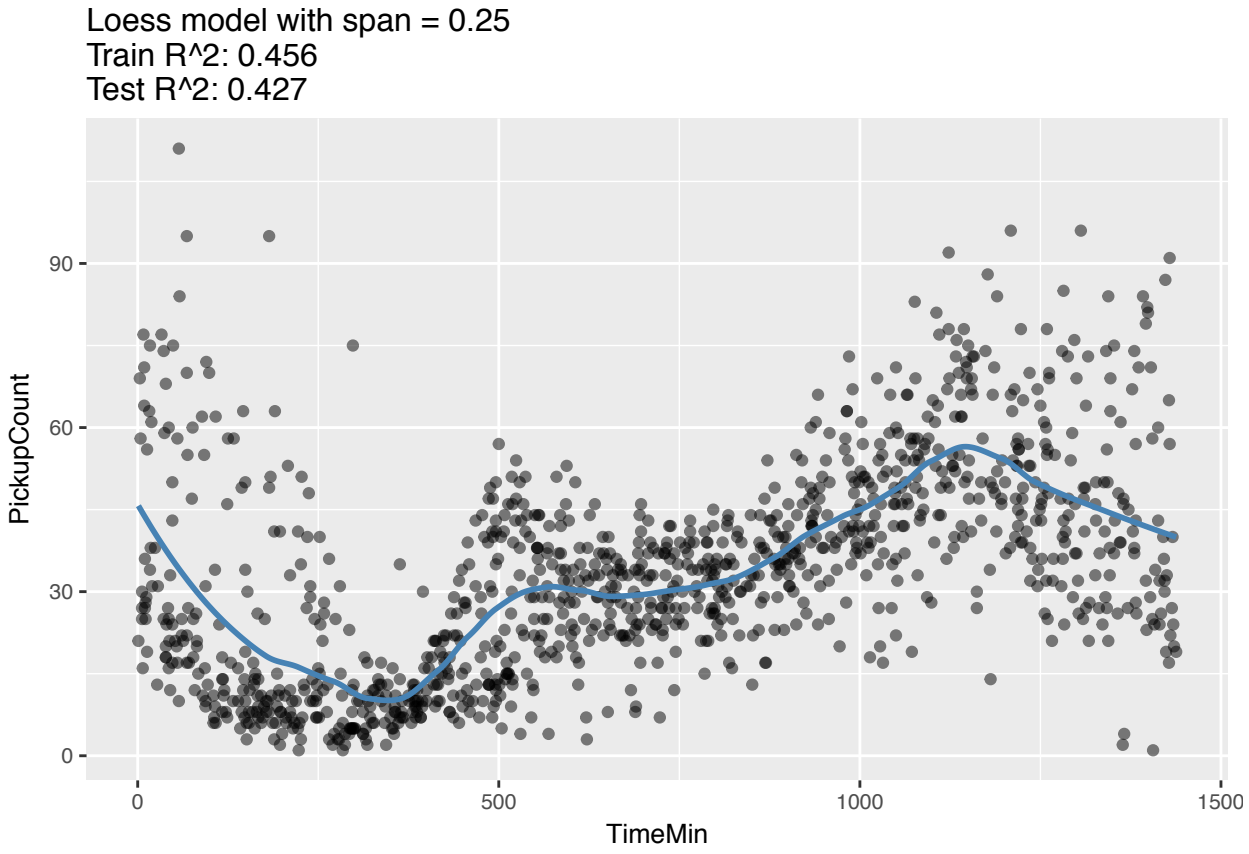
Perform 5-fold cross-validation:

```
# Perform 5-fold cross validation using the loess model
spans <- seq(0.1, 1, by=0.05)
cv_rsq = crossval_loess(train, spans, 5)

# Combine as a dataframe
loess_cv <- data.frame(param_val = spans, cv = cv_rsq)

# Visualization
ggplot(loess_cv, aes(x = spans, y = cv))+
  geom_point()+
  ggtitle(paste('Loess model cross-validation R^2 score
                \nMax R^2 =', max(loess_cv$cv), 'at span =',
                loess_cv$param_val[loess_cv$cv == max(loess_cv$cv)]))
```



We choose `span = 0.25` which gives the highest cross validation R^2 value of 0.4408 in our loess model to predict on the test dataset. Again, `span` indicates a window of values of TimeMin that we take each time in computing the weighted least-squares regression at each location within that window. The larger the `span`, the smoother the estimated function; the smaller the `span`, the more jagged the estimated function.

```
train_loess <- train
test_loess <- test

# Training model with df = 5
mod.loess <- loess(PickupCount ~ TimeMin, span = 0.25, data = train)

# Train
train_loess <- transform(train_loess, pred.loess = predict(mod.loess))
train.loess.r2 <- rsq(train_loess$PickupCount, train_loess$pred.loess)

# Test
test_loess <- transform(test_loess, pred.loess = predict(mod.loess, newdata = test))
test.loess.r2 <- rsq(test_loess$PickupCount, test_loess$pred.loess)

ggplot(train_loess, aes(x = TimeMin, y = PickupCount))+
  geom_point(alpha = 0.5)+
  geom_line(aes(y = pred.loess), color = 'steelblue', size = 1)+
  ggtitle(paste('Loess model with span = 0.25 \nTrain R^2:', train.loess.r2, '\nTest R^2:', test.loess.
  theme(text = element_text(size=10))
```



Comparison between the different R^2s

```
data.frame('Models' = c('Poly 5', 'Poly 10', 'Poly 25',
                        'B-Spline', 'N-Spline', 'Smooth.Spline', 'Loess'),
           'Train R2' = c(0.424, 0.448, 0.464, 0.440, 0.447, 0.425, 0.456),
           'Test R2' = c(0.386, 0.413, 0.427, 0.403, 0.411, 0.425, 0.427))
```

```
##          Models Train.R2 Test.R2
## 1        Poly 5    0.424   0.386
## 2       Poly 10    0.448   0.413
## 3       Poly 25    0.464   0.427
## 4      B-Spline    0.440   0.403
## 5      N-Spline    0.447   0.411
## 6 Smooth.Spline    0.425   0.425
## 7         Loess    0.456   0.427
```

We compare the different models. In general, the test R^2 scores are lower than their corresponding train R^2 score. It appears that the loess model gives the best R^2 result of 0.427. The polynomial with degree 25 model also gives a high R^2 of 0.427 on the test data set. However, it is often preferable to use loess rather than high degree polynomials. Therefore I would choose to use the loess model here.

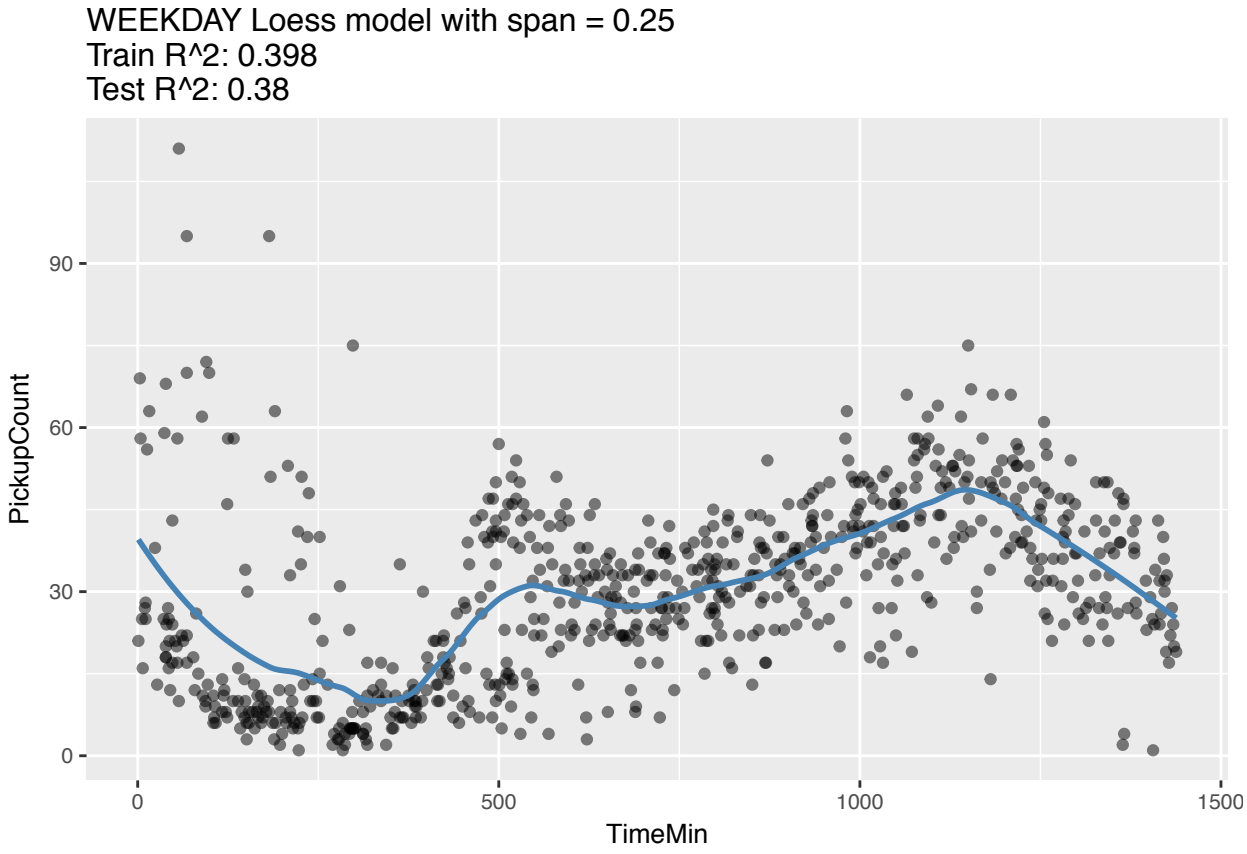
Part 1b Adapting to weekends

Splitting data into weekday and weekends:

```r
# We use the best span from Part 1A: 0.25
span = 0.25

# Divide train and test datasets into train.wkday, train.wkend, test.wkday, test.wkend
train.wkend <- train[train$DayOfWeek == 6 | train$DayOfWeek == 7, ]
train.wkday <- train[train$DayOfWeek == 1 | train$DayOfWeek == 2 |
                       train$DayOfWeek == 3 | train$DayOfWeek == 4 |
                       train$DayOfWeek == 5, ]

test.wkend <- test[test$DayOfWeek == 6 | test$DayOfWeek == 7, ]
test.wkday <- test[train$DayOfWeek == 1 | test$DayOfWeek == 2 |
                     test$DayOfWeek == 3 | test$DayOfWeek == 4 |
                     test$DayOfWeek == 5, ]
```

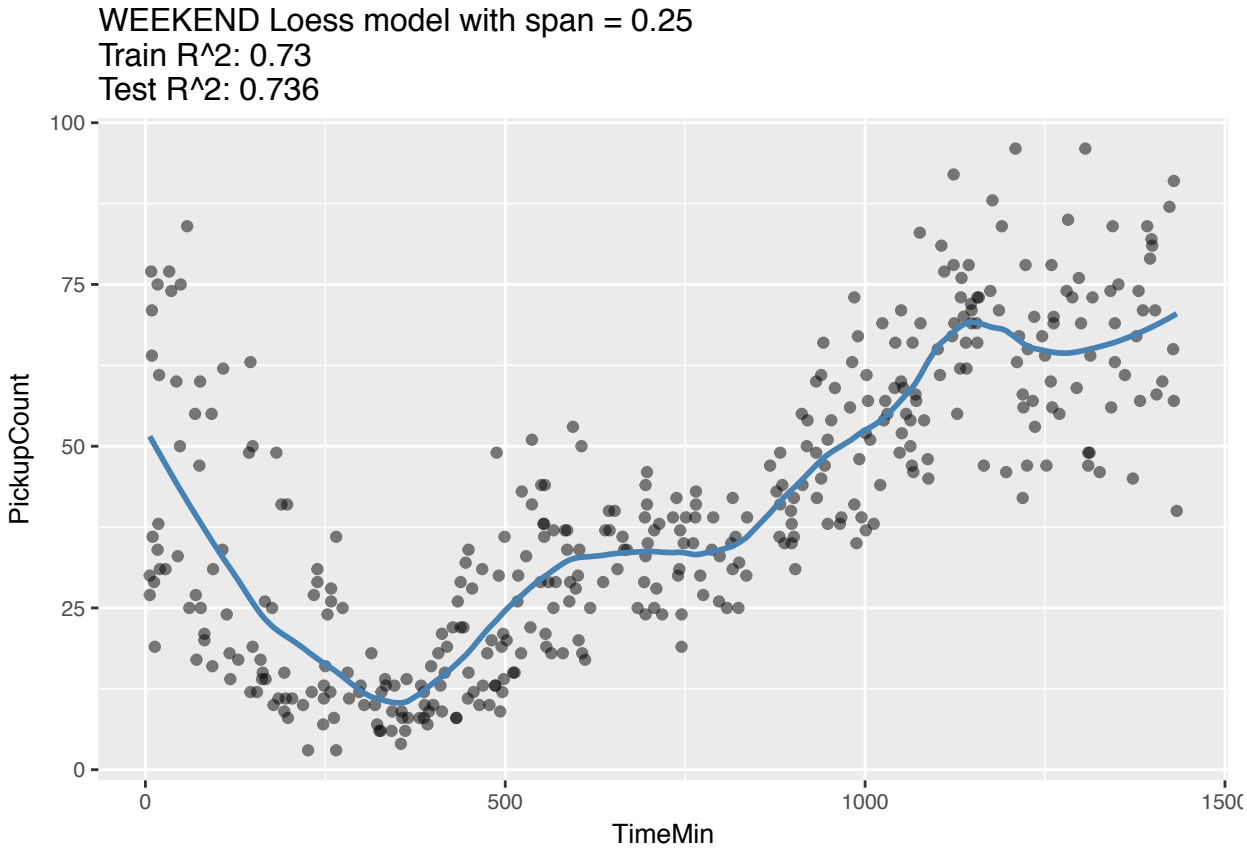
Weekday

```r
# Weekday model
train_loess_wkday <- train.wkday
test_loess_wkday <- test.wkday

mod.wkday <- loess(PickupCount ~ TimeMin, span = 0.25, data = train.wkday)

# Train
train_loess_wkday <- transform(train_loess_wkday, pred.wkday = predict(mod.wkday))
train.loess_wkday.r2 <- rsq(train_loess_wkday$PickupCount, train_loess_wkday$pred.wkday)

# Test
test_loess_wkday <- transform(test_loess_wkday,
                              pred.wkday = predict(mod.wkday, newdata = test.wkday))
test.loess_wkday.r2 <- rsq(test_loess_wkday$PickupCount, test_loess_wkday$pred.wkday)

ggplot(train_loess_wkday, aes(x = TimeMin, y = PickupCount))+
  geom_point(alpha = 0.5)+
  geom_line(aes(y = pred.wkday), color = 'steelblue', size = 1)+
  ggtitle(paste('WEEKDAY Loess model with span = 0.25 \nTrain R^2:',
                train.loess_wkday.r2, '\nTest R^2:', test.loess_wkday.r2))+
  theme(text = element_text(size=10))
```



Weekend

```
# Weekday model
train_loess_wkend <- train.wkend
test_loess_wkend <- test.wkend

mod.wkend <- loess(PickupCount ~ TimeMin, span = 0.25, data = train.wkend)

# Train
train_loess_wkend <- transform(train_loess_wkend, pred.wkend = predict(mod.wkend))
train.loess_wkend.r2 <- rsq(train_loess_wkend$PickupCount, train_loess_wkend$pred.wkend)

# Test
test_loess_wkend <- transform(test_loess_wkend,
                              pred.wkend = predict(mod.wkend, newdata = test.wkend))
test.loess_wkend.r2 <- rsq(test_loess_wkend$PickupCount, test_loess_wkend$pred.wkend)

ggplot(train_loess_wkend, aes(x = TimeMin, y = PickupCount))+
  geom_point(alpha = 0.5)+
  geom_line(aes(y = pred.wkend), color = 'steelblue', size = 1)+
  ggtitle(paste('WEEKEND Loess model with span = 0.25 \nTrain R^2:',
                train.loess_wkend.r2, '\nTest R^2:',
                test.loess_wkend.r2))+
  theme(text = element_text(size=10))
```



We also use the loess model trained in Part A `mod.loess` to predict on the WEEKDAY and WEEKEND test datasets:

```
test.wkday.r2 <- rsq_pred(mod.loess, test.wkend, 'PickupCount')
test.wkend.r2 <- rsq_pred(mod.loess, test.wkday, 'PickupCount')

paste('Using the loess model in Part A, the weekday R^2 is',
      test.wkday.r2, 'the weekend R^2 is', test.wkend.r2)
```

```
## [1] "Using the loess model in Part A, the weekday R^2 is 0.538 the weekend R^2 is 0.367"
```

Once we separate our dataset into weekend and weekdays, it looks like the loess model in Part B is able to predict PickupCount on WEEKENDS more accurately with a R^2 of 0.736 compared the loess model in Part A with a R^2 of 0.367. The WEEKDAY prediction using the loess model in Part B is less accurate with a R^2 of 0.380 compared to the loess model in Part A with a R^2 of 0.538. All R^2 values are evaluated on the test data set. Also, by visual inspection of the scatter plot of the data points, it makes sense to differentiate beween weekday and weekends since the data points have different trends.

Problem 2

Data Exploration

```
# Import data
train <- read.csv("./CS109b-hw1-datasets/dataset_2_train.txt", sep = "\t")
test <- read.csv("./CS109b-hw1-datasets/dataset_2_test.txt", sep = "\t")
```

```r
# Check structure of data
str(train)
```

```
## 'data.frame':    498 obs. of  8 variables:
##  $ ViolentCrimesPerPop: num  0.15 0.07 0.12 0.22 0.11 0.63 0.02 0.75 0.28 0.46 ...
##  $ Population         : num  0.03 0.07 0 0 0.01 0.02 0 0.06 0.01 0.16 ...
##  $ PercentageBlack    : num  0.13 0.06 0.05 0.02 0.01 0.2 0.02 0.62 0.72 0.19 ...
##  $ PercentageWhite    : num  0.73 0.88 0.52 0.97 0.99 0.79 0.98 0.43 0.43 0.42 ...
##  $ PercentageAsian    : num  0.61 0.12 0.02 0.05 0.02 0.09 0.01 0.15 0.01 0.96 ...
##  $ PercentageHispanic : num  0.07 0.03 0.03 0 0.01 0.31 0.02 0.46 0 0.44 ...
##  $ PercentageUrban    : num  1 0.98 0 0.73 1 0 1 1 0 1 ...
##  $ MedIncome          : num  0.1 0.43 0.05 0.26 0.53 0.27 0.14 0.22 0.18 0.4 ...
```

```r
str(test)
```

```
## 'data.frame':    1496 obs. of  8 variables:
##  $ ViolentCrimesPerPop: num  0.43 0.12 0.03 0.14 0.03 0.55 0.53 0.15 0.08 0.06 ...
##  $ Population         : num  0 0.04 0.01 0.02 0.01 0.01 0.03 0.01 0.02 0.03 ...
##  $ PercentageBlack    : num  0.49 1 0.02 0.06 0 0.03 0.2 0.06 0.08 0.01 ...
##  $ PercentageWhite    : num  0.56 0.08 0.95 0.54 0.98 0.46 0.84 0.87 0.91 0.96 ...
##  $ PercentageAsian    : num  0.17 0.12 0.09 1 0.06 0.2 0.02 0.3 0.07 0.13 ...
##  $ PercentageHispanic : num  0.04 0.1 0.05 0.25 0.02 1 0 0.03 0.1 0.02 ...
##  $ PercentageUrban    : num  0 1 0.9 1 0.81 0 1 1 1 1 ...
##  $ MedIncome          : num  0.3 0.58 0.5 0.52 0.42 0.16 0.17 0.54 0.72 0.8 ...
```

```r
# Check for any  missing values
sum(is.na(train))
```

```
## [1] 0
```

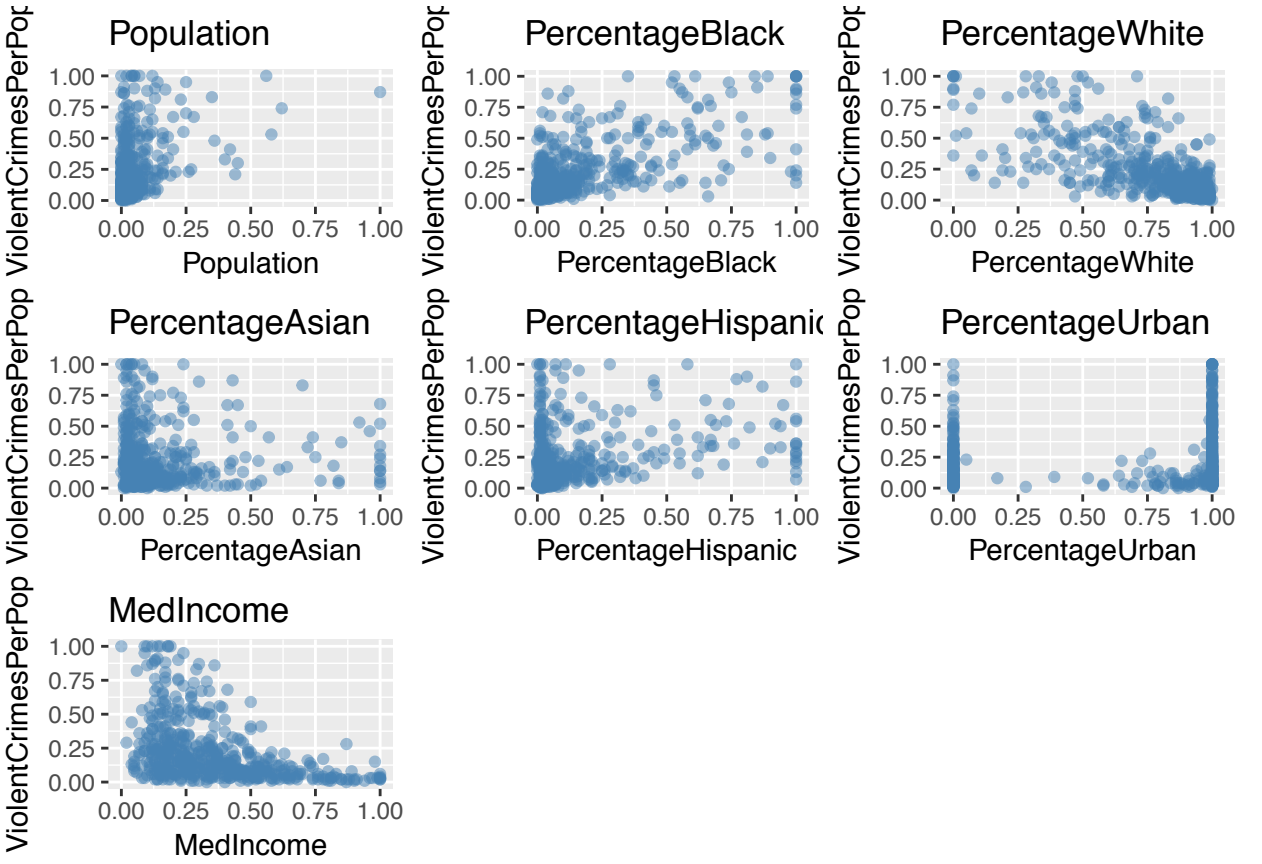
```r
sum(is.na(test))
```

```
## [1] 0
```

There is no missing data.

```r
# Population
p1<- ggplot(train, aes(x = Population, y = ViolentCrimesPerPop))+
  geom_point(alpha = 0.5, color = 'steelblue')+
  ggtitle('Population')+
  xlim(0, 1)

# PercentageBlack
p2 <- ggplot(train, aes(x = PercentageBlack, y = ViolentCrimesPerPop))+
  geom_point(alpha = 0.5, color = 'steelblue')+
  ggtitle('PercentageBlack')+
  xlim(0, 1)

# PercentageWhite
p3 <- ggplot(train, aes(x = PercentageWhite, y = ViolentCrimesPerPop))+
  geom_point(alpha = 0.5, color = 'steelblue')+
  ggtitle('PercentageWhite')+
  xlim(0, 1)

# PercentageAsian
p4 <- ggplot(train, aes(x = PercentageAsian, y = ViolentCrimesPerPop))+
  geom_point(alpha = 0.5, color = 'steelblue')+
```

```
  ggtitle('PercentageAsian')+
  xlim(0, 1)

# PercentageHispanic
p5 <- ggplot(train, aes(x = PercentageHispanic, y = ViolentCrimesPerPop))+
  geom_point(alpha = 0.5, color = 'steelblue')+
  ggtitle('PercentageHispanic')+
  xlim(0, 1)

# PercentageUrban
p6 <- ggplot(train, aes(x = PercentageUrban, y = ViolentCrimesPerPop))+
  geom_point(alpha = 0.5, color = 'steelblue')+
  ggtitle('PercentageUrban')+
  xlim(0, 1)

# MedIncome
p7 <- ggplot(train, aes(x = MedIncome, y = ViolentCrimesPerPop))+
  geom_point(alpha = 0.5, color = 'steelblue')+
  ggtitle('MedIncome')+
  xlim(0, 1)

grid.arrange(p1, p2, p3, p4, p5, p6, p7, ncol = 3)
```



Conclusion: PercentageUrban is not a very important factor. Everything is not linear with the dependent variable.

Part 2A Polynomial Regression

1. Linear Regression

```
# Train model
mod.linear <- lm(ViolentCrimesPerPop ~ Population +
                   PercentageBlack + PercentageWhite +
                   PercentageAsian + PercentageHispanic +
                   PercentageUrban + MedIncome, data = train )

# Calculate R^2 on train and test sets
train.linear.r2 <- rsq_pred(mod.linear, train, 'ViolentCrimesPerPop')
test.linear.r2 <- rsq_pred(mod.linear, test, 'ViolentCrimesPerPop')

print(paste('Train R^2:', train.linear.r2, '\nTest R^2:', test.linear.r2))
```

```
## [1] "Train R^2: 0.618 \nTest R^2: 0.555"
```

In the linear model with `ViolentCrimesPerPop` as the dependent variable and all others as predictors, the R^2 on the training set is 0.618, on the test set is 0.555.

2. Regression with polynomial basis functions of degree 2 (i.e. basis functions x, x2 for each predictor x)

```
# Train model
mod.p2 <- lm(ViolentCrimesPerPop ~ poly(Population, degree = 2) +
               poly(PercentageBlack, degree = 2) +
               poly(PercentageWhite, degree = 2) +
               poly(PercentageAsian, degree = 2) +
               poly(PercentageHispanic, degree = 2) +
               poly(PercentageUrban, degree = 2) +
               poly(MedIncome, degree = 2),
             data = train)

# Calculate R^2 on train and test sets
train.p2.r2 <- rsq_pred(mod.p2, train, 'ViolentCrimesPerPop')
test.p2.r2 <- rsq_pred(mod.p2, test, 'ViolentCrimesPerPop')

print(paste('Train R^2:', train.p2.r2, '\nTest R^2:', test.p2.r2))
```

```
## [1] "Train R^2: 0.633 \nTest R^2: 0.575"
```

3. Regression with polynomial basis functions of degree 3 (i.e. basis functions x, x2, x3 for each predictor x)

```
# Train model
mod.p3 <- lm(ViolentCrimesPerPop ~ poly(Population, degree = 3) +
               poly(PercentageBlack, degree = 3) +
               poly(PercentageWhite, degree = 3) +
               poly(PercentageAsian, degree = 3) +
               poly(PercentageHispanic, degree = 3) +
               poly(PercentageUrban, degree = 3) +
```

```
              poly(MedIncome, degree = 3),
            data = train)

# Calculate R^2 on train and test sets
train.p3.r2 <- rsq_pred(mod.p3, train, 'ViolentCrimesPerPop')
test.p3.r2 <- rsq_pred(mod.p3, test, 'ViolentCrimesPerPop')

print(paste('Train R^2:', train.p3.r2, '\nTest R^2:', test.p3.r2))
```

```
## [1] "Train R^2: 0.644 \nTest R^2: 0.573"
```

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

```
# Train model
bs_degree <- function(d){
  mod.bs <- lm(ViolentCrimesPerPop ~ bs(Population, df = d) +
                 bs(PercentageBlack, df = d) +
                 bs(PercentageWhite, df = d) +
                 bs(PercentageAsian, df = d) +
                 bs(PercentageHispanic, df = d) +
                 bs(PercentageUrban, df = d) +
                 bs(MedIncome, df = d),
               data = train)

  # Test dataset
  train.bs.r2 <- rsq_pred(mod.bs, train, 'ViolentCrimesPerPop')
  test.bs.r2 <- rsq_pred(mod.bs, test, 'ViolentCrimesPerPop')


  return(c(train.bs.r2, test.bs.r2))
}

bs_degree(3)
```

```
## [1] 0.644 0.573
```

```
bs_degree(5)
```

```
## [1] 0.656 0.559
```

```
bs_degree(7)
```

```
## [1] 0.667 0.537
```

The R^2 for the training data set increases as we increase the degree of piecewise polynomial, but the R^2 for the test data set decreases.

Summary of test R^2 scores:

```
# Table of R^2 score on test datasets
data.frame('Method' = c('Linear Model', 'Poly 2', 'Poly 3',
                        'B-Spline Degree 3', 'B-Spline Degree 5', 'B-Spline Degree 7'),
```

```
           'Test R2' = c(test.linear.r2, test.p2.r2, test.p3.r2,
                         bs_degree(3)[[2]], bs_degree(5)[[2]], bs_degree(7)[[2]]))
```

```
##              Method Test.R2
## 1      Linear Model   0.555
## 2            Poly 2   0.575
## 3            Poly 3   0.573
## 4 B-Spline Degree 3   0.573
## 5 B-Spline Degree 5   0.559
## 6 B-Spline Degree 7   0.537
```

From the summary of R^2 scores, it looks like the polynomial model with degree 2 performs the best with a R^2 score of 0.575 on the test set.

Part 2B Generalized Additive Model (GAM)

1. Fit a GAM to the training set, and compare the test R2 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.

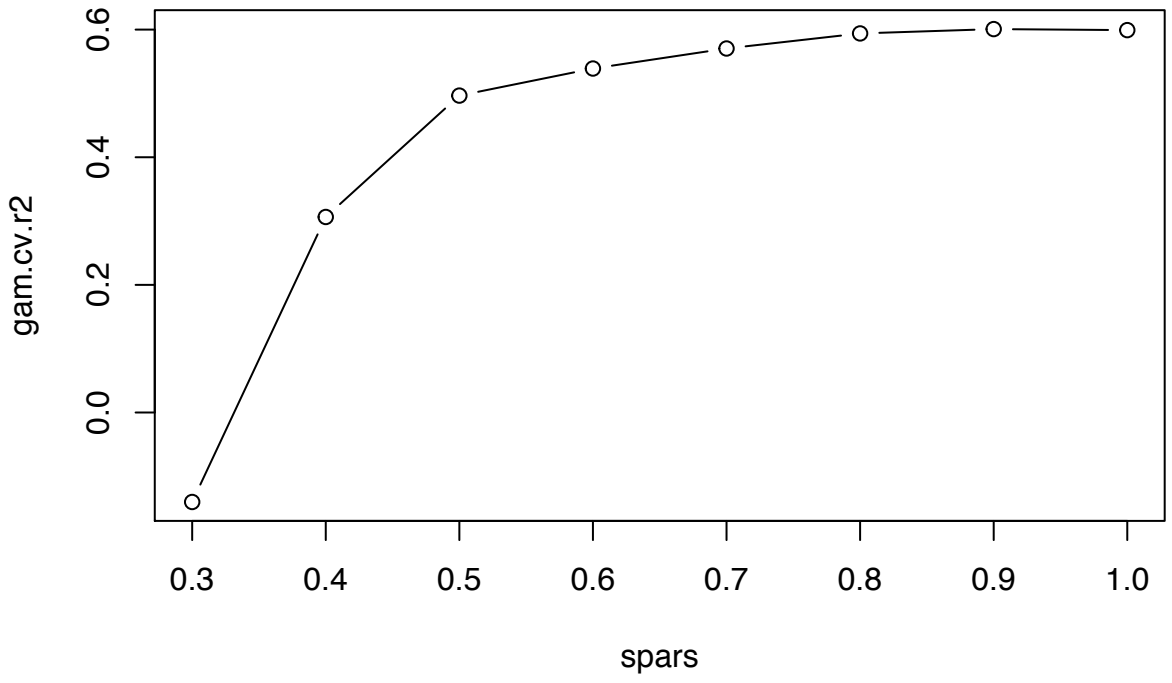
```
# Function for k-fold cross-validation to tune span parameter in gam
library(gam)
```

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

```
## Loaded gam 1.14
```

```
crossval_gam = 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]
      mod_formula = as.formula(paste0('ViolentCrimesPerPop ~
s(Population, spar = ', param_val[i],') +
s(PercentageBlack, spar = ', param_val[i], ') +
                        s(PercentageWhite, spar = ', param_val[i], ')+
                        s(PercentageAsian, spar = ', param_val[i], ') +
```

```
                        s(PercentageHispanic, spar =', param_val[i], ') +
                        s(PercentageUrban, spar =', param_val[i], ') +
                        s(MedIncome, spar =', param_val[i], ')'))

      mod.gam <- gam(mod_formula, data = train[folds!=j, ])

      # Make prediction on fold 'j'
      pred = predict(mod.gam, train[folds == j,]) ###

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


# Use crossval_gam function to determin the best spar value
spars <- seq(0.3, 1, by = 0.1)
gam.cv.r2 <- crossval_gam(train, param_val = spars, k = 5)

plot(spars, gam.cv.r2, type = "b")
```




```
spar.best <- spars[which(gam.cv.r2 == max(gam.cv.r2))]

spar.best
```

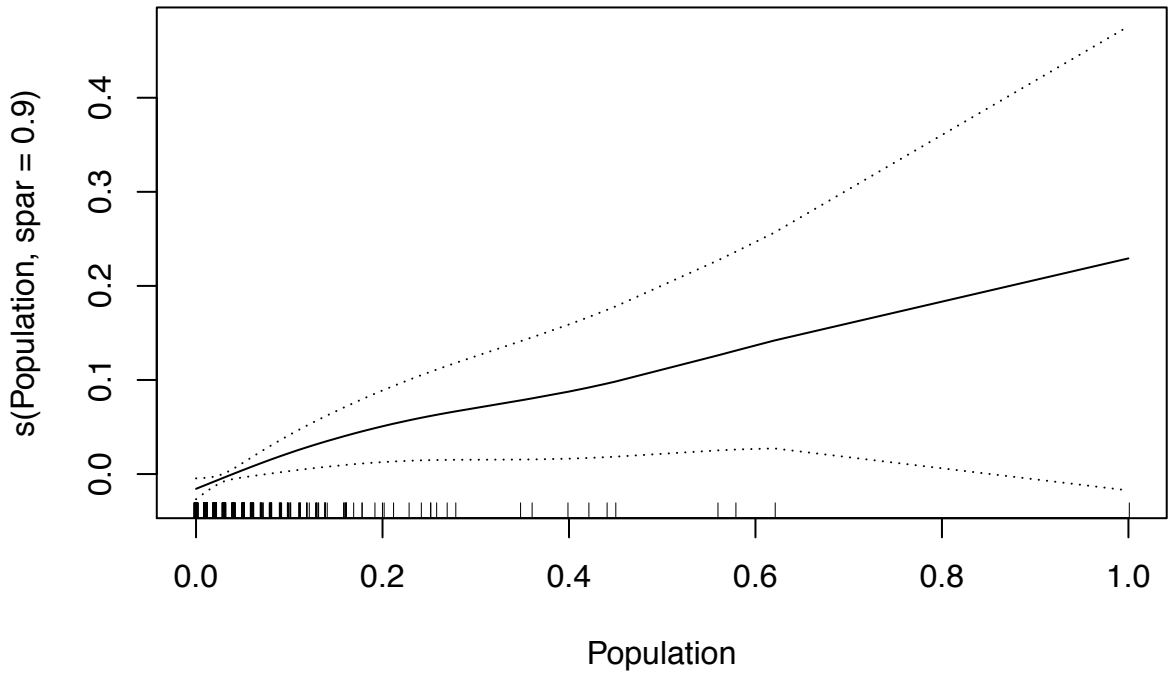
```
## [1] 0.9
```

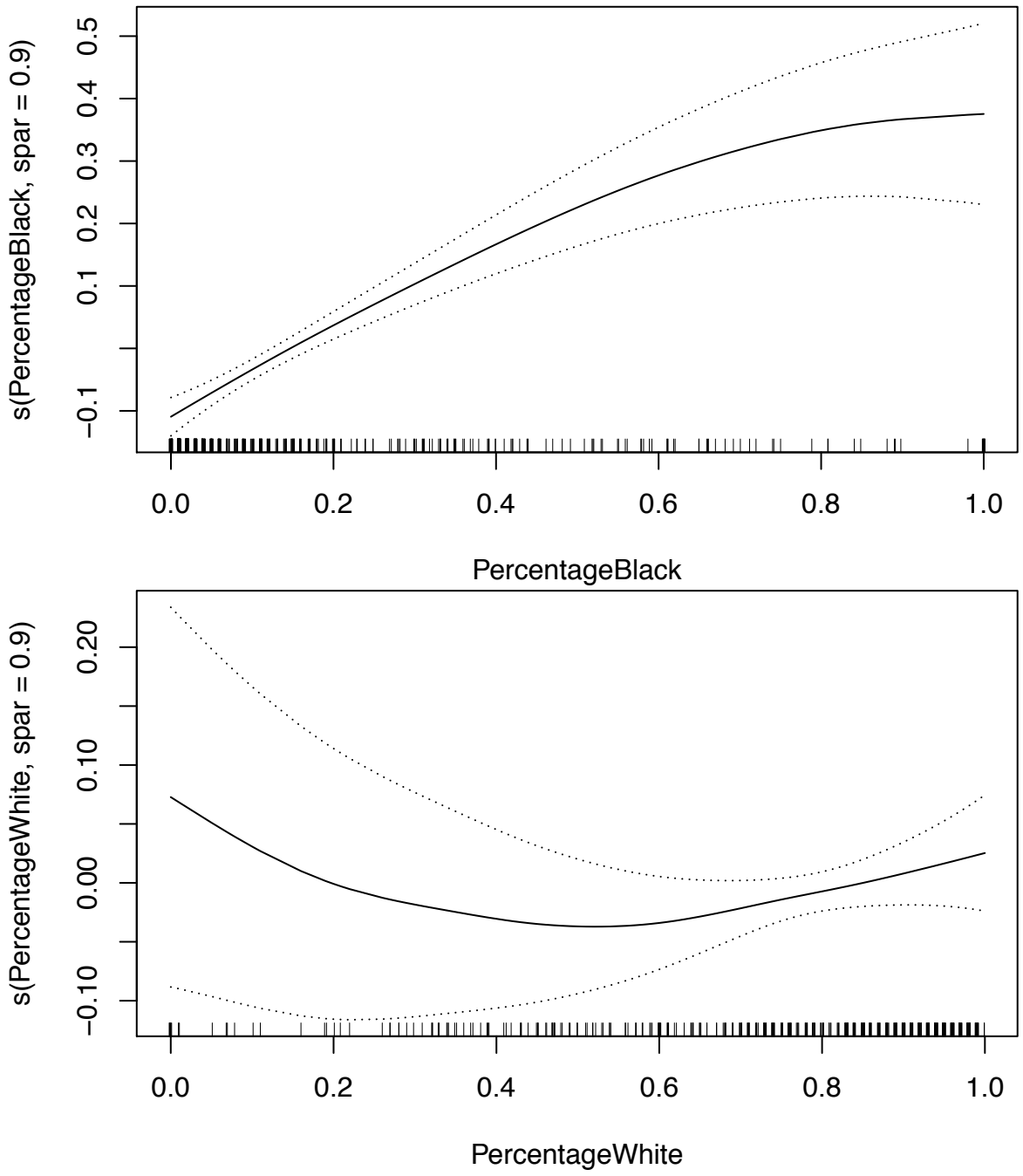
```
gam.cv.r2
```

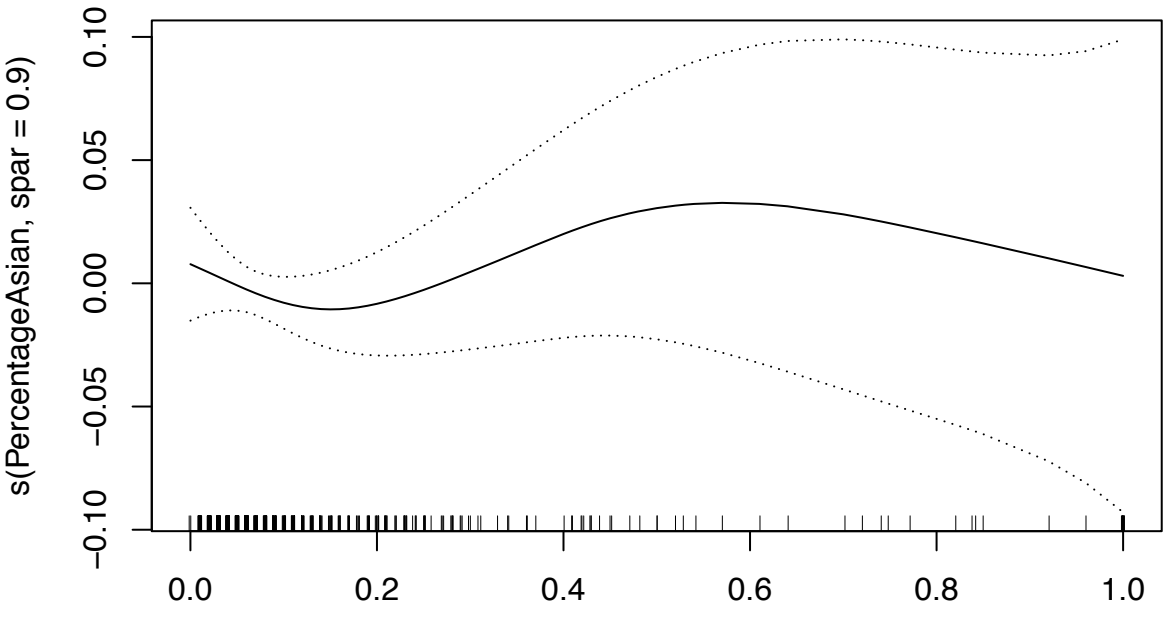
```
## [1] -0.1402  0.3064  0.4966  0.5390  0.5704  0.5940  0.6008  0.5992
```

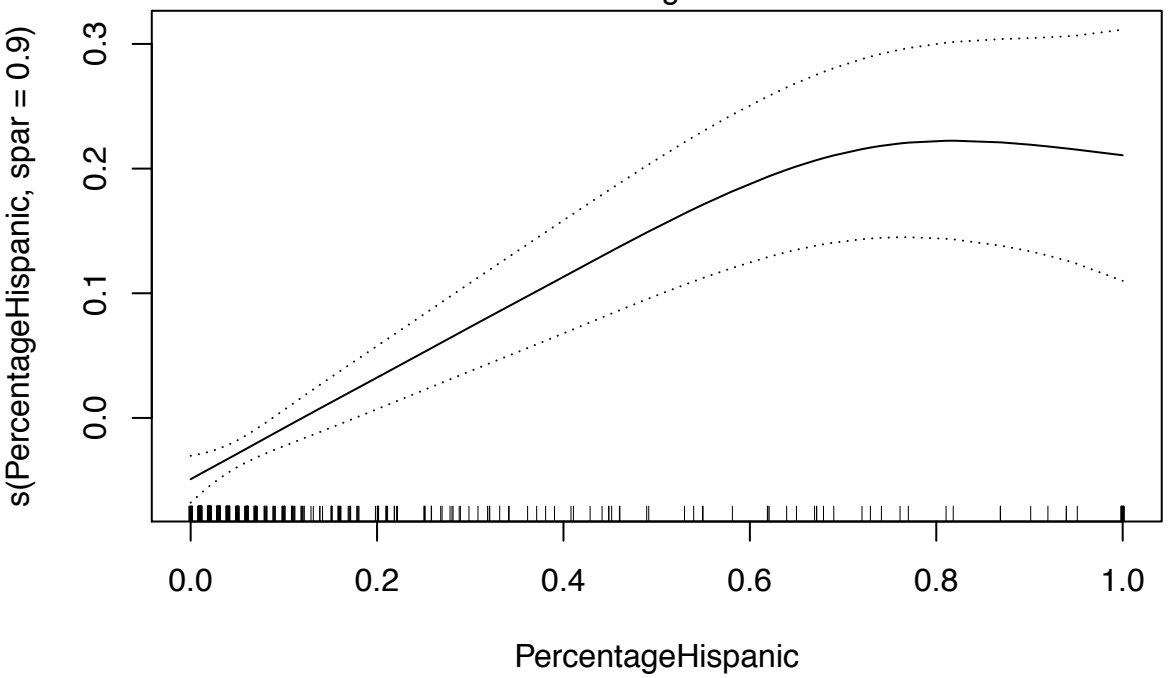
2.Plot and examine the smooth of each predictor for the fitted GAM, along with plots of upper and lower standard errors on the predictions. What are some useful insights conveyed by these plots, and by the coefficients assigned to each local model?

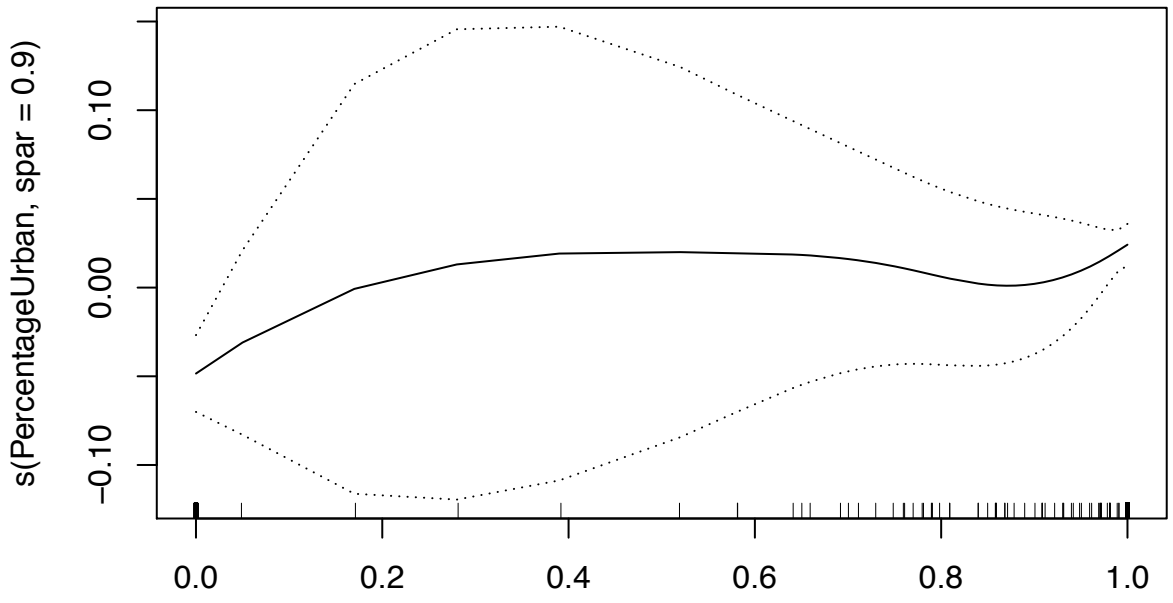
```
# Train model using the best spar value determined by CV

mod_formula = as.formula(paste0('ViolentCrimesPerPop ~
s(Population, spar = ', spar.best,') +
s(PercentageBlack, spar = ', spar.best, ') +
s(PercentageWhite, spar = ', spar.best, ')+
s(PercentageAsian, spar = ', spar.best, ') +
s(PercentageHispanic, spar =', spar.best, ') +
s(PercentageUrban, spar =', spar.best, ') +
s(MedIncome, spar =', spar.best, ')'))

mod.gam.best <- gam(mod_formula, data = train)

plot(mod.gam.best, se=TRUE)
```

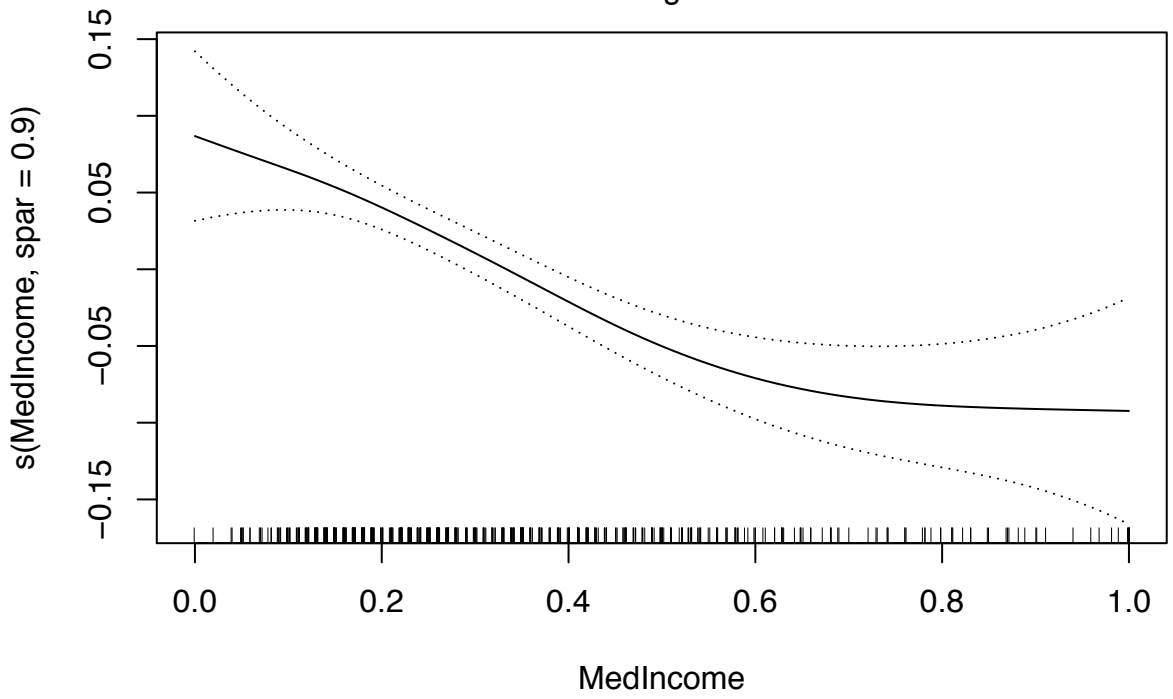



s(PercentageBlack, spar = 0.9)
−0.1
0.1
0.2
0.3
0.4
0.5
0.0
0.2
0.4
0.6
0.8
1.0
PercentageBlack
s(PercentageWhite, spar = 0.9)
−0.10
0.00
0.10
0.20
0.0
0.2
0.4
0.6
0.8
1.0
PercentageWhite


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






```
rsq_pred(mod.gam.best, test, 'ViolentCrimesPerPop')
```

```
## [1] 0.577
```

The R^2 score on the test data set is 0.577, which is higher than the R^2 from all models in Part 2a.

```
mod.gam.best[['coefficients']]
```

```
##                        (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
```

It looks like PercentageAsia and PercentageUrban are not very useful since ViolentCrimesPerPop does not vary much as these two predictors change in value from the graphs, and also the coefficients are very small. `PercentageBlack` and `PercentageHispanic` have larger positive coefficients than the rest of the predictors. We can infer that a larger `PercentageBlack` and `PerentageHispanic` is correlated with a higher `ViolentCrimesPerPop`.

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.

Linear vs. GAM (all predictors)

```
anova(mod.linear, mod.gam.best, 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
```

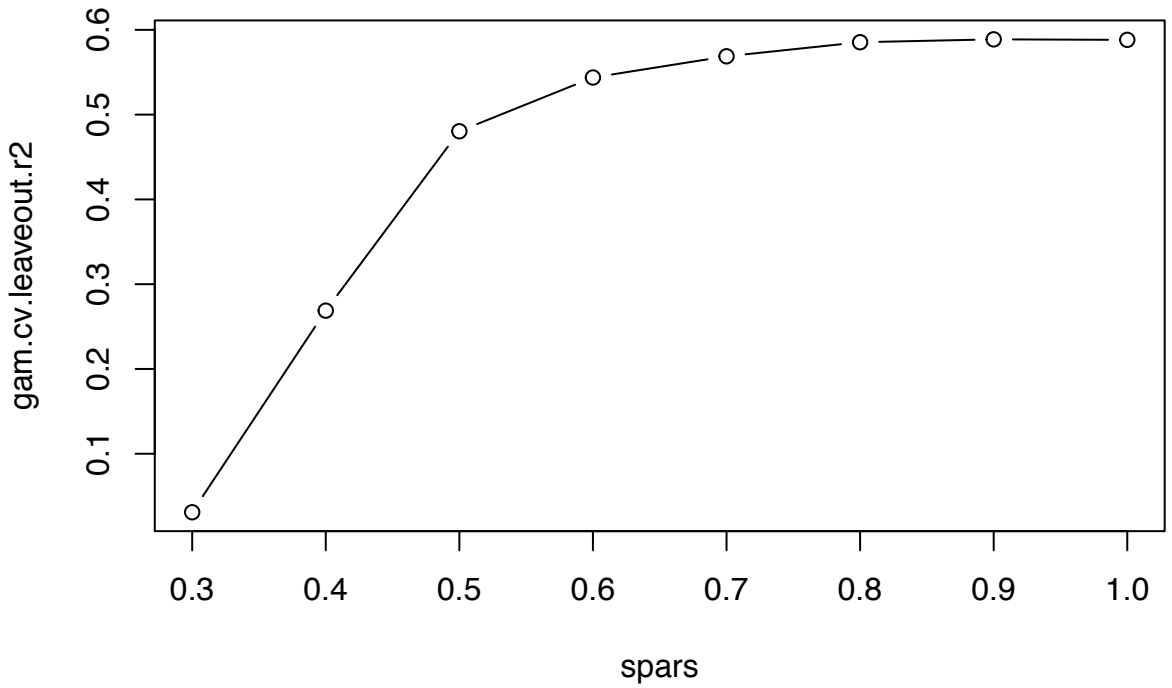
Since the p-value is $0.003 < 0.05$, we reject the null hypothesis (linear model) and concludes that the GAM model performs better. This is consistent with the R^2 score on the test set, with the linear model giving a score of 0.555, and the GAM model giving a score of 0.577.

Compare GAM (all predictors) model and GAM without PercentageAsia and PercentageUrban

First, we use cross-validation to determine the best spar value to use in this model:

```
# Function for k-fold cross-validation to tune span parameter in gam
crossval_gam_leaveout = 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
```

```r
  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]

      mod_formula = as.formula(paste0('ViolentCrimesPerPop ~
s(Population, spar = ', param_val[i],') +
                        s(PercentageBlack, spar =', param_val[i],') +
                        s(PercentageWhite, spar =', param_val[i],')+
                        s(PercentageHispanic, spar =', param_val[i],') +
                        s(MedIncome, spar =', param_val[i],')'))
      mod.gam <- gam(mod_formula, data = train[folds!=j, ])

      # Make prediction on fold 'j'
      pred = predict(mod.gam, train[folds == j,]) ###

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


# Use crossval_gam function to determin the best spar value
spars <- seq(0.3, 1, by = 0.1)
gam.cv.leaveout.r2 <- crossval_gam_leaveout(train, param_val = spars, k = 5)

plot(spars, gam.cv.leaveout.r2 , type = "b")
```




```
spar.leaveout.best <- spars[which(gam.cv.leaveout.r2 == max(gam.cv.leaveout.r2))]

spar.leaveout.best
```

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

We then build the model with the best spar value:

```
# Built model with best spar

mod_formula <- as.formula(paste0(
  'ViolentCrimesPerPop ~ s(Population, spar =', spar.leaveout.best,')+
s(PercentageBlack, spar =', spar.leaveout.best,') +
                        s(PercentageWhite, spar =', spar.leaveout.best,')+
                        s(PercentageHispanic, spar =', spar.leaveout.best,') +
                        s(MedIncome, spar =', spar.leaveout.best,')'))

mod.gam.leaveout <- gam(mod_formula, data = train)
```

GAM (all predictors) vs.GAM without PercentageAsia and PercentageUrban

```
anova(mod.gam.best, mod.gam.leaveout, 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(PercentageHispanic,
##     spar = 0.9) + s(MedIncome, spar = 0.9)
```

```
##   Resid. Df Resid. Dev      Df Deviance  Pr(>Chi)
## 1    476.59     9.5165
## 2    482.85     9.9815 -6.2595 -0.46496 0.0008693 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The p-value of 0.00087 < 0.05 indicates that we can reject the null hypothesis and conclude that the GAM model without PercentageAsia and PercentageUrban performs better than the GAM model with all predictors.

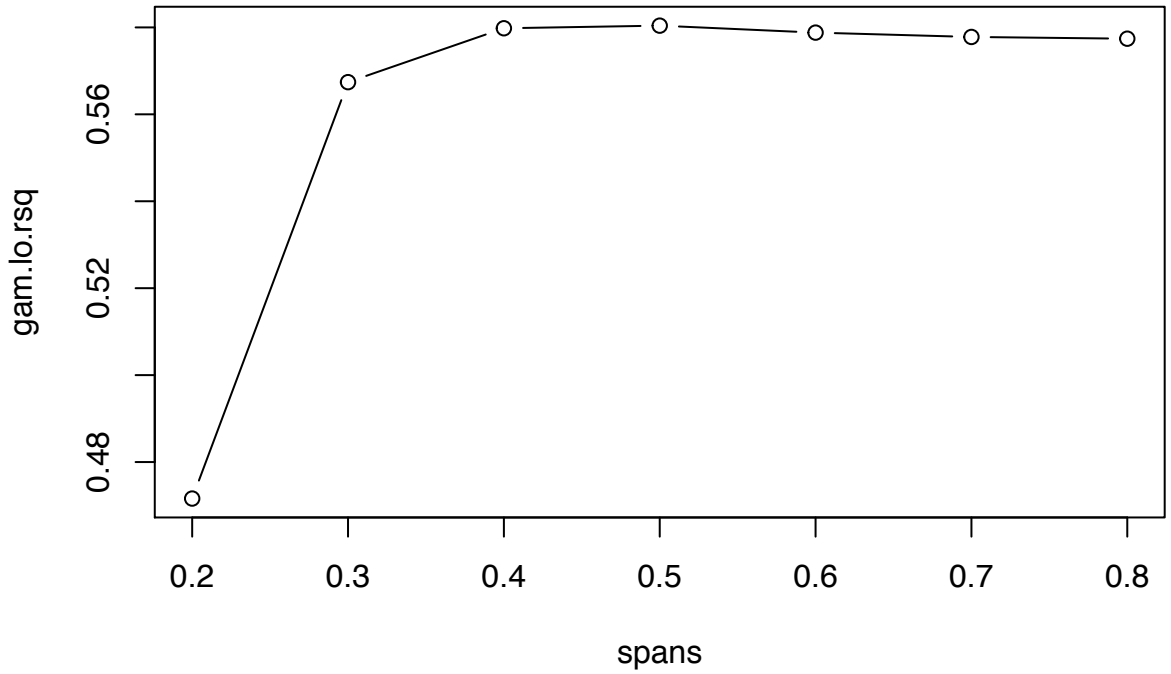
2c: Including interaction terms

Re-fit the GAM with the following interaction terms included: ### A local regression basis function involving attributes 'Population', 'PercentageUrban' and 'MedIncome'

```
# Function for k-fold cross-validation to tune span parameter in gam
crossval_lo = 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(103) # 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]

      mod_formula <- as.formula(paste0(
'ViolentCrimesPerPop ~ s(Population, spar =', spar.best,')+
                       s(PercentageBlack, spar =', spar.best,') +
                       s(PercentageWhite, spar =', spar.best,')+
                       s(PercentageAsian, spar =', spar.best,')+
                       s(PercentageHispanic, spar =', spar.best,') +
                       s(PercentageUrban, spar =', spar.best,')+
                       s(MedIncome, spar =', spar.best,')+
                       lo(Population, PercentageUrban, MedIncome, span =', param_val[i],')'))

      mod.gam <- gam(mod_formula,data = train[folds!=j, ])

      # Make prediction on fold 'j'
      pred = predict(mod.gam, train[folds == j,]) ###

      # Compute R^2 for predicted values
```

```
      cv_rsq[i] = cv_rsq[i] + rsq(train$ViolentCrimesPerPop[folds == j], pred)
    }

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

  return(cv_rsq)
}
```

```
# Perform 5-fold CV
spans <- seq(0.2, 0.8, by = 0.1)
gam.lo.rsq <- crossval_lo(train, param_val = spans, k = 5)

# Identify the best span
plot(spans, gam.lo.rsq, type = 'b')
```




```
span.best <- spans[which(gam.lo.rsq == max(gam.lo.rsq))]
print(paste('List of R^2 values:'))
```

```
## [1] "List of R^2 values:"
```

```
gam.lo.rsq
```

```
## [1] 0.4716 0.5674 0.5798 0.5804 0.5788 0.5778 0.5774
```

```
print(paste('Best span =', span.best))
```

```
## [1] "Best span = 0.5"
```

Using the best span of 0.5, we fit the model mod.gam.lo and calculate the test R^2:

```
mod_formula <- as.formula(paste0(
'ViolentCrimesPerPop ~ s(Population, spar = ',spar.best,')+
                       s(PercentageBlack, spar = ', spar.best,') +
```

```
                        s(PercentageWhite, spar = ', spar.best,')+
                        s(PercentageAsian, spar = ', spar.best,')+
                        s(PercentageHispanic, spar = ', spar.best,') +
                        s(PercentageUrban, spar = ', spar.best,')+
                        s(MedIncome, spar = ', spar.best,')+
                        lo(Population, PercentageUrban, MedIncome, span = ', span.best,')'))
mod.gam.lo <- gam(mod_formula,data = train)

mod.gam.lo.r2 <- rsq(test$ViolentCrimesPerPop, predict(mod.gam.lo, newdata = test))
print(paste('R2 value of mod.gam.lo:', mod.gam.lo.r2))
```

```
## [1] "R2 value of mod.gam.lo: 0.581"
```

Anova test of GAM model vs GAM model with lo function of Population, PercentageUrban and MedIncome

```
anova(mod.gam.best, mod.gam.lo)
```

```
## 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
```

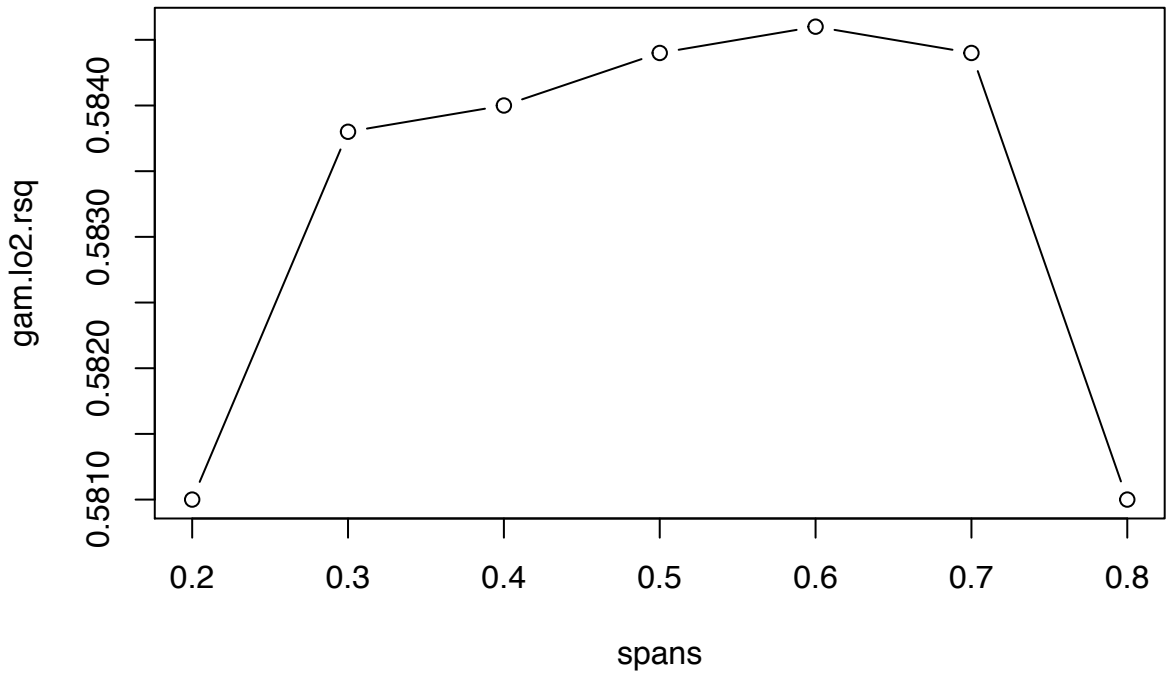
The p-value of $0.0096 < 0.05$ indicates that we can reject the null hypothesis and conclude that the GAM model with interaction term performs better than the GAM model without interaction terms. The R^2 score of 0.581 for GAM model with interaction terms (`Population`, `PercentageUrban`, `MedIncome`) is also higher than the previous 0.555 for GAM model without interaction terms.

A local regression basis function involving a race-related attribute and 'MedIncome'

```
# Function for k-fold cross-validation to tune span parameter in gam
crossval_lo2 = 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(103) # 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]
      mod_formula <- as.formula(paste0(
'ViolentCrimesPerPop ~ s(Population, spar =', spar.best,')+
                       s(PercentageBlack, spar =', spar.best,') +
                       s(PercentageWhite, spar =', spar.best,')+
                       s(PercentageAsian, spar =', spar.best,')+
                       s(PercentageHispanic, spar =', spar.best,') +
                       s(PercentageUrban, spar =', spar.best,')+
                       s(MedIncome, spar =', spar.best,')+
                       lo(PercentageBlack, MedIncome, span =', param_val[i],')'))

      mod.gam <- gam(mod_formula,data = train[folds!=j, ])

      # Make prediction on fold 'j'
      pred = predict(mod.gam, train[folds == j,]) ###

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

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

  return(cv_rsq)
}
```

```
# Perform 5-fold CV
spans <- seq(0.2, 0.8, by = 0.1)
gam.lo2.rsq <- crossval_lo2(train, param_val = spans, k = 5)

# Identify the best span
plot(spans, gam.lo2.rsq, type = 'b')
```




```
span.best2 <- spans[which(gam.lo2.rsq == max(gam.lo2.rsq))]
print(paste('List of R^2 values:'))
```

```
## [1] "List of R^2 values:"
```

```
gam.lo2.rsq
```

```
## [1] 0.5810 0.5838 0.5840 0.5844 0.5846 0.5844 0.5810
```

```
print(paste('Best span =', span.best2))
```

```
## [1] "Best span = 0.6"
```

We use the span value of 0.6 which gave us the highest R^2 value:

```
mod_formula <- as.formula(paste0(
'ViolentCrimesPerPop ~ s(Population, spar =', spar.best,')+
                       s(PercentageBlack, spar =', spar.best,') +
                       s(PercentageWhite, spar =', spar.best,')+
                       s(PercentageAsian, spar =', spar.best,')+
                       s(PercentageHispanic, spar =', spar.best,') +
                       s(PercentageUrban, spar =', spar.best,')+
                       s(MedIncome, spar =', spar.best,')+
                       lo(PercentageBlack, MedIncome, span =', span.best2,')'))

mod.gam.lo2 <- gam(mod_formula,data = train)

mod.gam.lo2.r2 <- rsq(test$ViolentCrimesPerPop, predict(mod.gam.lo2, newdata = test))
print(paste('R2 value of mod.gam.lo:', mod.gam.lo2.r2))
```

```
## [1] "R2 value of mod.gam.lo: 0.592"
```

Anova test of GAM model vs GAM model with lo function of PercentageBlack and MedIncome

```
anova(mod.gam.best, mod.gam.lo2)
```

```
## 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 = 0.6)
##   Resid. Df Resid. Dev     Df Deviance Pr(>Chi)
## 1    476.59     9.5165
## 2    471.43     9.1464 5.1649  0.37016 0.002127 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The p-value of $0.0021 < 0.05$ indicates that we can reject the null hypothesis and conclude that the GAM model with interaction term performs better than the GAM model without interaction terms. The R^2 score of 0.592 for GAM model with interaction terms (`PercentageBlack`, `MedIncome`) is also higher than the previous 0.555 for GAM without interaction terms.