CS109b Homework 1 Submission

Ihsaan Patel

February 5, 2017

Helper Functions

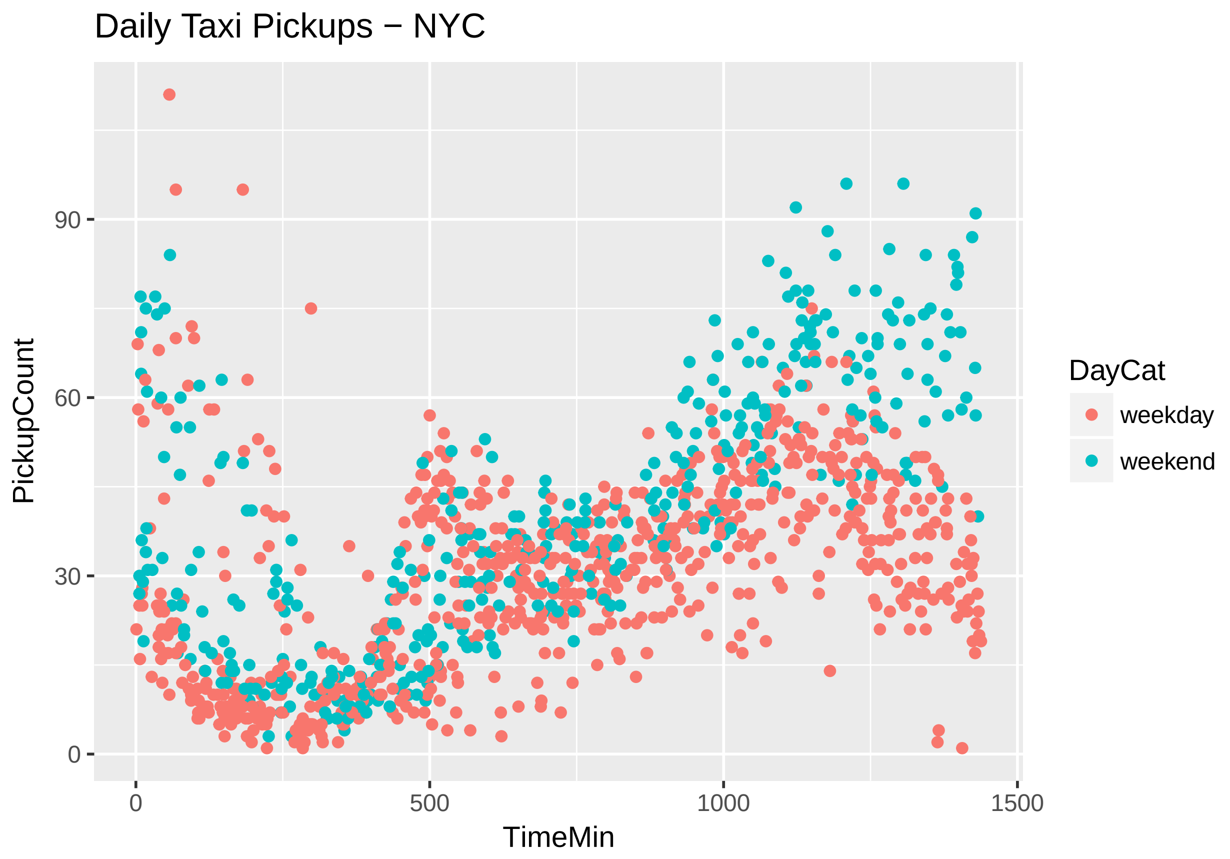
```r
library(ggplot2)
# Function to compute R^2 for observed and predicted responses
rsq = function(y, predict) {
  tss = sum((y - mean(y))^2)
  rss = sum((y-predict)^2)
  r_squared = 1 - rss/tss

  return(r_squared)
}

crossval_ns = 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.ns = lm(PickupCount ~ ns(TimeMin, df = param_val[i]), data = train[folds!=j, ])

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

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)
}

library(gam)
```

```
## Loading required package: splines

## Loading required package: foreach

## Loaded gam 1.14
```

```r
crossval_gams = 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]
      gam_formula <- as.formula(sprintf("ViolentCrimesPerPop ~ s(Population, spar=%1$f) + s(PercentageBl

      model.gams = gam(gam_formula, data = train[folds!=j, ])

      # Make prediction on fold 'j'
      pred = predict(model.gams, train[folds == j, 2:8])

      # 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)
}

crossval_gams_lo1 = function(train, param_val, spar_param, 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]
      gam_formula <- as.formula(sprintf("ViolentCrimesPerPop ~ s(Population, spar=%1$f) + s(PercentageBl

      model.gams = gam(gam_formula, data = train[folds!=j, ])

      # Make prediction on fold 'j'
      pred = predict(model.gams, train[folds == j, 2:8])

      # 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)
}

crossval_gams_lo2 = function(train, param_val, spar_param, 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]
      gam_formula <- as.formula(sprintf("ViolentCrimesPerPop ~ s(Population, spar=%1$f) + s(PercentageBl
      model.gams = gam(gam_formula, data = train[folds!=j, ])

      # Make prediction on fold 'j'
      pred = predict(model.gams, train[folds == j, 2:8])

      # 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)
}

# Function to plot and evaluate model
model_evaluate = function(model_name, model, train_data, test_data) {
  if (class(model) == "smooth.spline") {
    train_predictions <- predict(model, x = train_data$TimeMin)$y
    test_predictions <- predict(model, newdata = test_data$TimeMin)$y
  } else {
    train_predictions <- predict(model, newdata = train_data)
    test_predictions <- predict(model, newdata = test_data)
  }
  train_r2 <- rsq(train_data$PickupCount, train_predictions)
  test_r2 <- rsq(test_data$PickupCount, test_predictions)
  graph_title <- sprintf("%s | Train R^2: %.4f | Test R^2: %.4f", model_name, train_r2, test_r2)
  ggplot(transform(train_data, pred = train_predictions), mapping = aes(x = TimeMin, y=PickupCount)) + g
  }

# Taken from: http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)/
# Multiple plot function
#
# ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objects)
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
  library(grid)

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                    ncol = cols, nrow = ceiling(numPlots/cols))
  }

 if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}
```

Problem 1: Predicting Taxi PickupCount

```
# Load data
train_dataset1 <- read.table("dataset_1_train.txt",header = TRUE, sep = ",")

# Add label for weekday or weekend
train_dataset1$DayCat <- ifelse(train_dataset1$DayOfWeek < 6, c("weekday"), c("weekend"))

# Plot the data
ggplot(data = train_dataset1, mapping = aes(x = TimeMin, y=PickupCount, color = DayCat)) + geom_point()
```



The pattern of taxi pickups makes intuitive sense with relatively low pickups during the early morning for both weekdays and weekends, peak pickups around the start and end of working hours during weekdays, and higher pickups during the night on weekends.

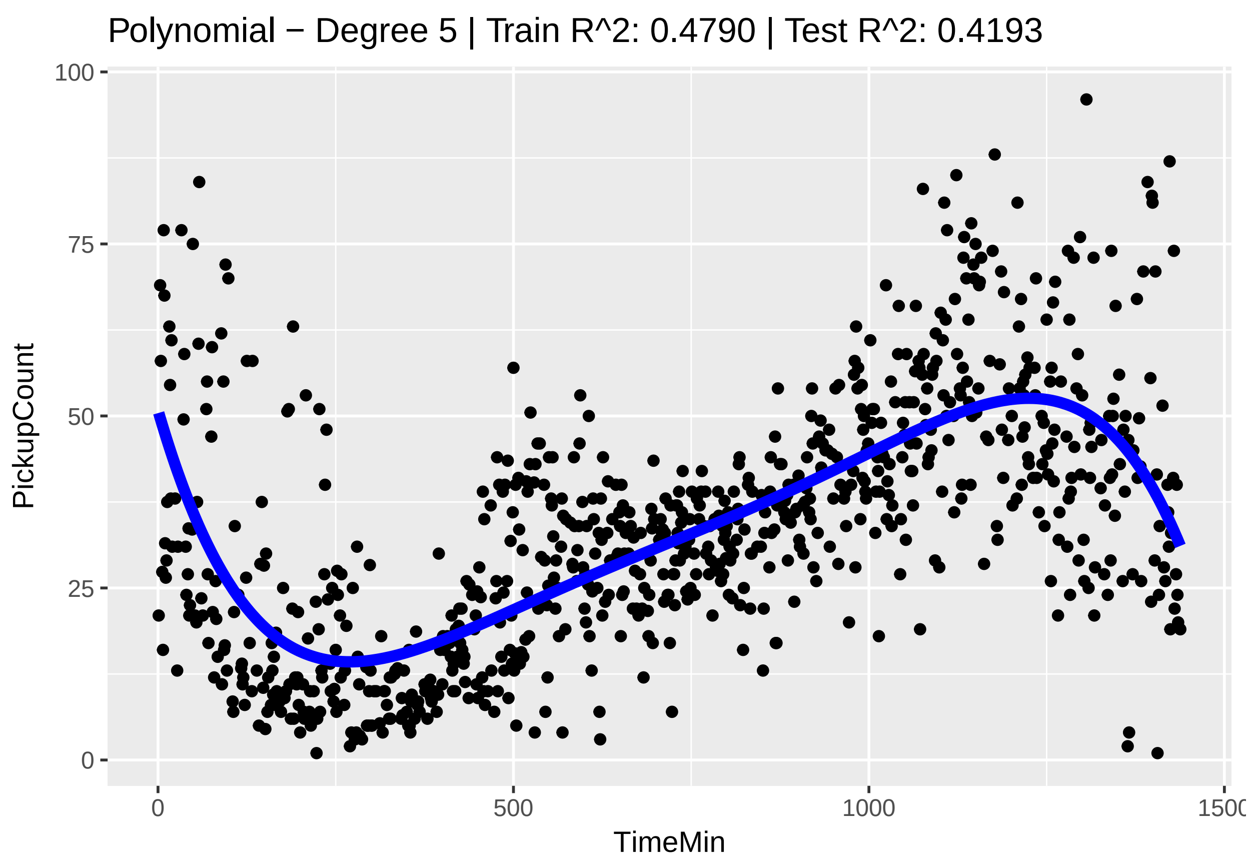
Part 1a: Explore different regression models

```
# Get average taxi PickupCount for training set
train_dataset1_clean <- aggregate(train_dataset1[, 3], list(TimeMin = train_dataset1$TimeMin), mean)
colnames(train_dataset1_clean)[2] <- "PickupCount"

# Load data and get average taxi PickupCount for testing set
test_dataset1 <- read.table("dataset_1_test.txt",header = TRUE, sep = ",")
test_dataset1_clean <- aggregate(test_dataset1[, 3], list(TimeMin = test_dataset1$TimeMin), mean)
colnames(test_dataset1_clean)[2] <- "PickupCount"
```

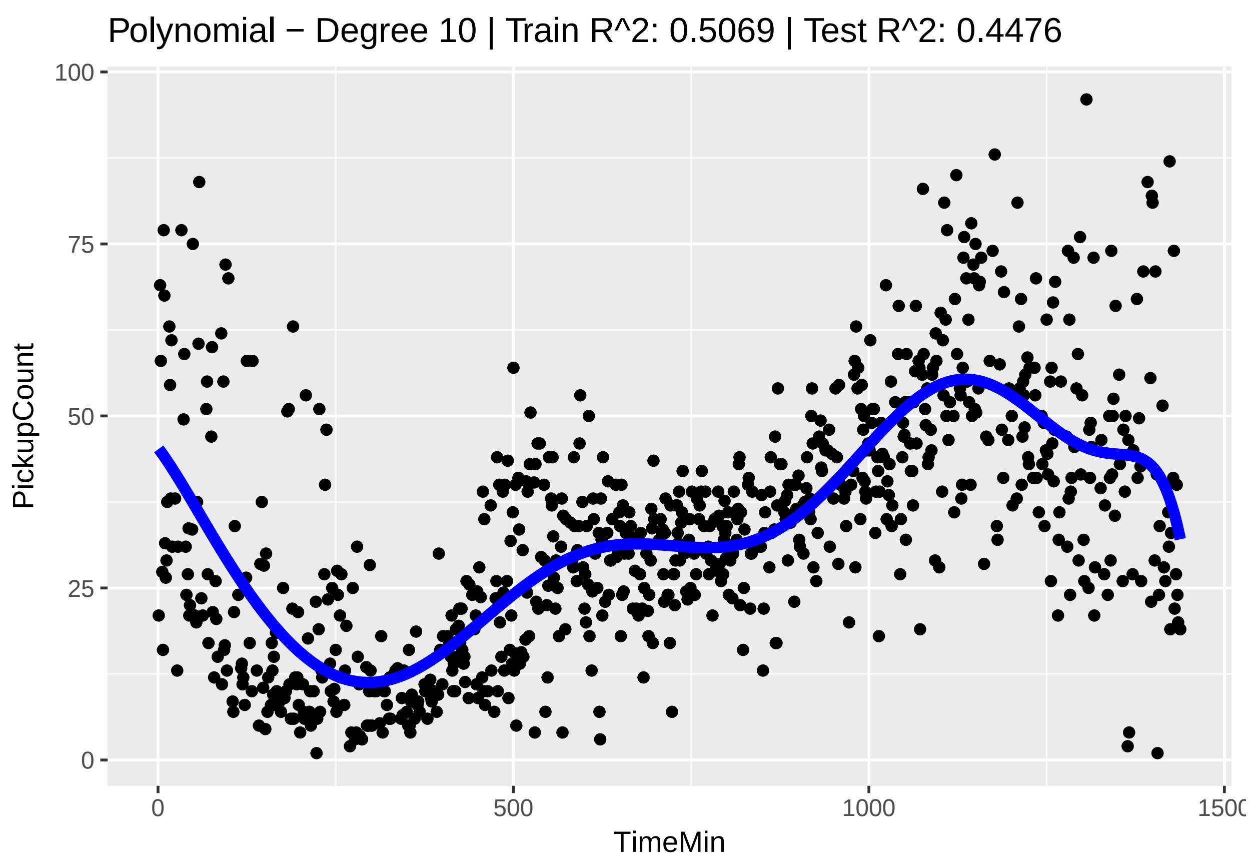
1.Regressions with Different Basis Functions

Polynomial Functions

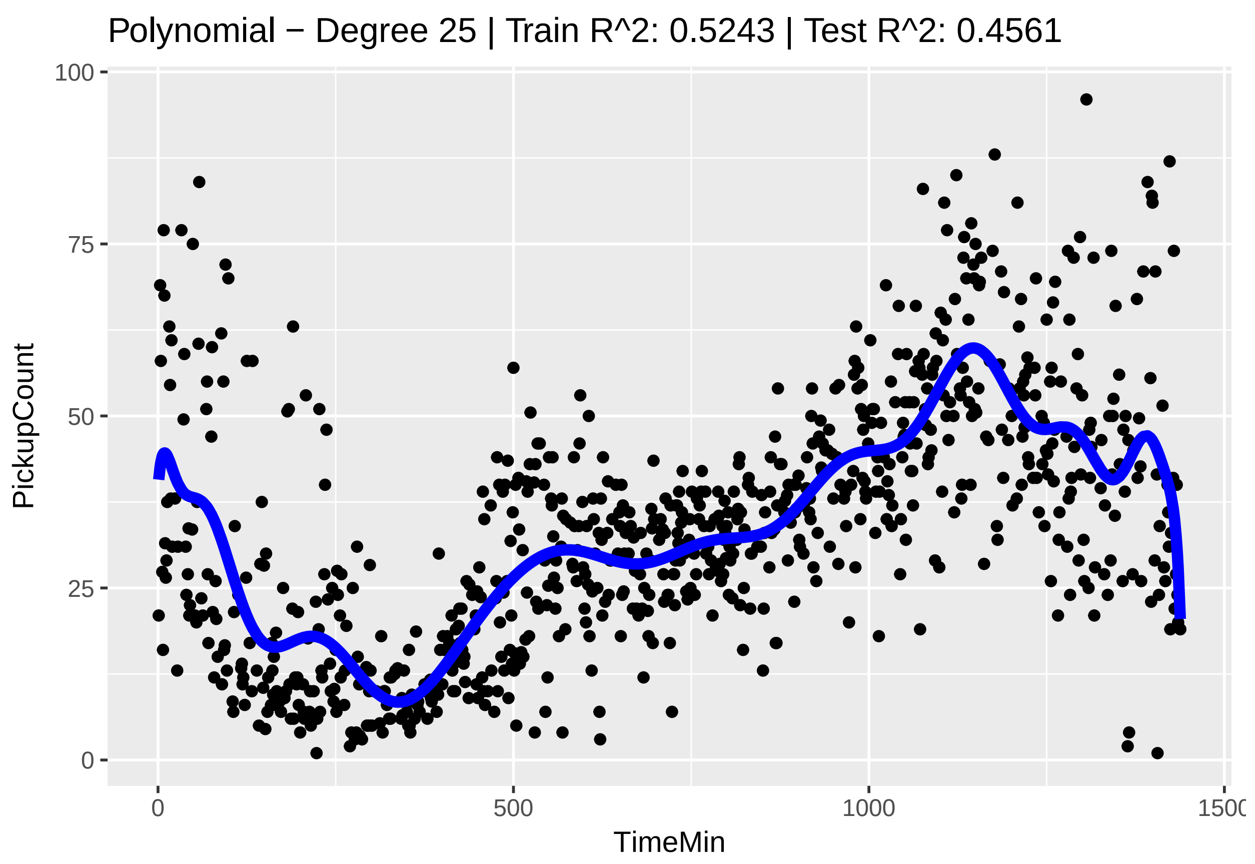
```
# Fit, test, and plot a linear model with polynomial of degree 5
simp_poly_model.deg5 <- lm(PickupCount ~ poly(TimeMin, degree = 5), data = train_dataset1_clean)
model_evaluate("Polynomial - Degree 5", simp_poly_model.deg5, train_dataset1_clean, test_dataset1_clean
```



```
# Fit, test, and plot a linear model with polynomial of degree 10
simp_poly_model.deg10 <- lm(PickupCount ~ poly(TimeMin, degree = 10), data = train_dataset1_clean)
model_evaluate("Polynomial - Degree 10", simp_poly_model.deg10, train_dataset1_clean, test_dataset1_clea
```



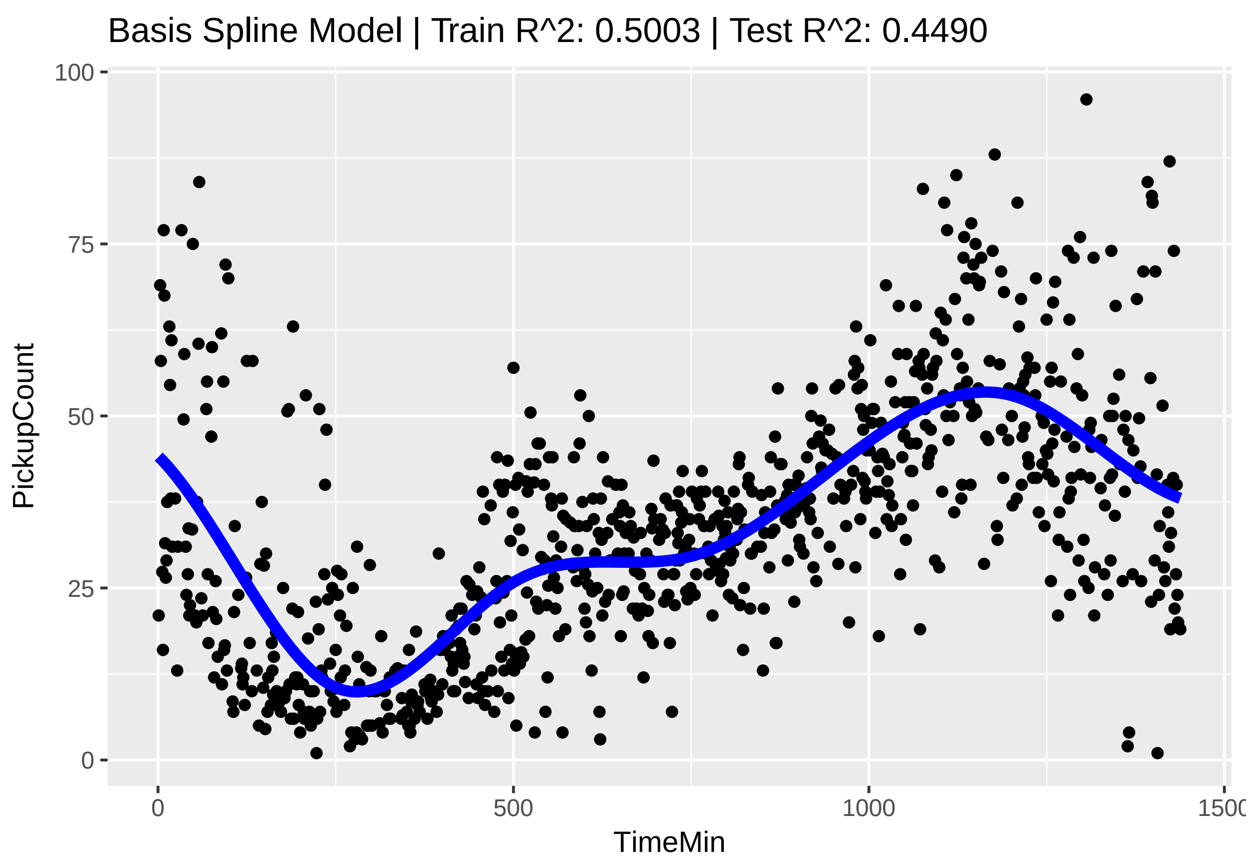
```
# Fit, test, and plot a linear model with polynomial of degree 25
simp_poly_model.deg25 <- lm(PickupCount ~ poly(TimeMin, degree = 25), data = train_dataset1_clean)
model_evaluate("Polynomial - Degree 25", simp_poly_model.deg25, train_dataset1_clean, test_dataset1_clea
```



The polynomial with degree 25 produces the best test R^2, however it looks like the model is starting to over-fit relative to the polynomial with degree 10 as the delta between the train and test R^2 is larger and the visualization shows bends that are not intuitive given the actual data.

Cubic B-Splines

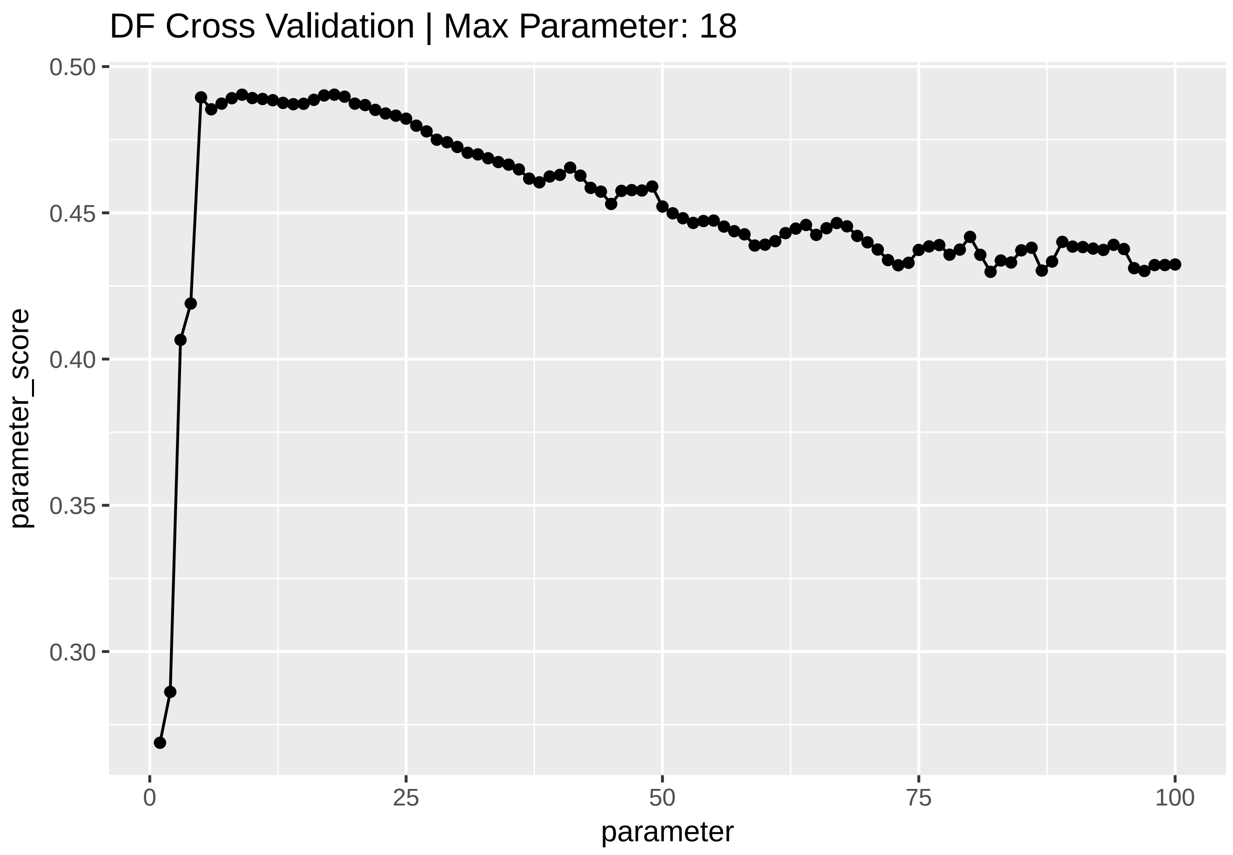
```
# Fit, test, and plot a linear model with cubic basis splines and visually chosen knots
library(splines)
bs_model <- lm(PickupCount ~ bs(TimeMin, knots = c(250, 500, 750, 1200)), data = train_dataset1_clean)
model_evaluate("Basis Spline Model", bs_model, train_dataset1_clean, test_dataset1_clean)
```



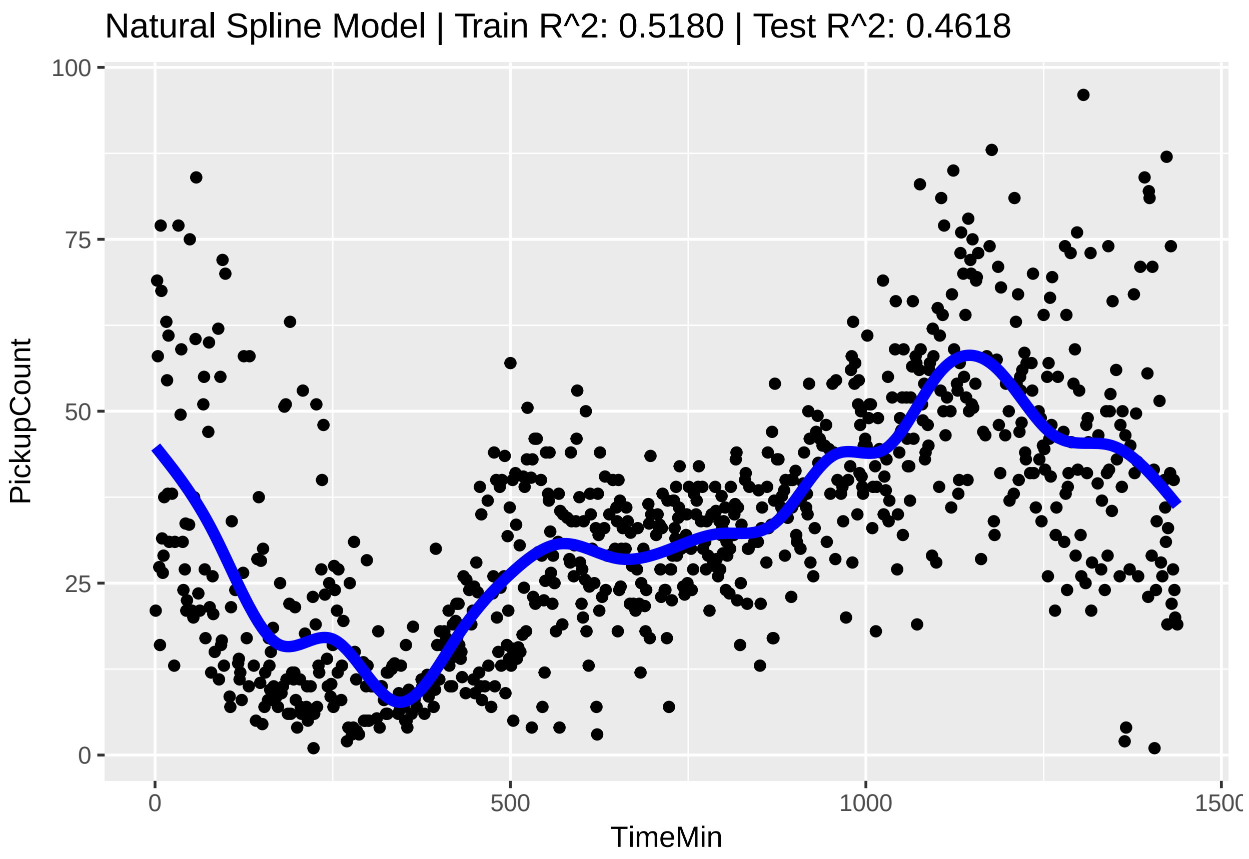
Knots were chosen based on points in the scatter plot where it looks like the slope of the function should change, and the test R^2 shows performance similar to the polynomial with degree 10 but with only 4 knots. It looks like the increase in knots should reduce bias in the model but also increase variance.

Natural Cubic Splines

```
# Cross validate the degrees of freedom for a natural cubic spline model and plot it
df_to_validate <- c(1:100)
ns_cv_scores <- crossval_ns(train_dataset1_clean, df_to_validate, 5)
df_max_parameter <- df_to_validate[which.max(ns_cv_scores)]
ggplot(data = data.frame(parameter = df_to_validate, parameter_score = ns_cv_scores), aes(x = parameter
```



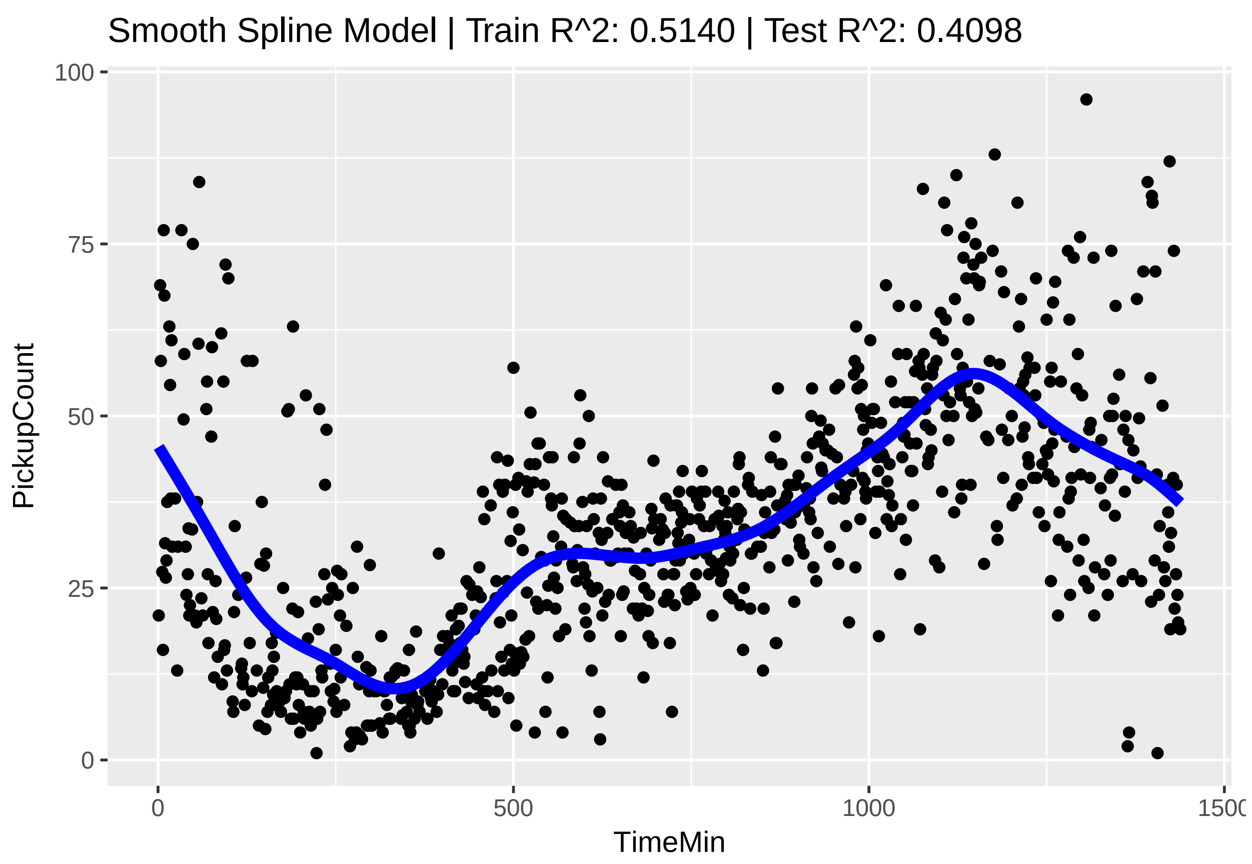
```r
# Fit, test, and plot a linear model with natural cubic splines
ns_model <- lm(PickupCount ~ ns(TimeMin, df = df_max_parameter), data = train_dataset1_clean)
model_evaluate("Natural Spline Model", ns_model, train_dataset1_clean, test_dataset1_clean)
```



The degrees of freedom tuning parameter appears to increase the number of knots in the function, with sharp decreases in bias (and therefore model R^2) up to df = 5. The model appears to start over-fitting due to increased variance after df = 18, which is the chosen parameter for the final model. The model produces a higher test R^2 than the polynomial or B-spline models.

2.Smoothing Spline Model

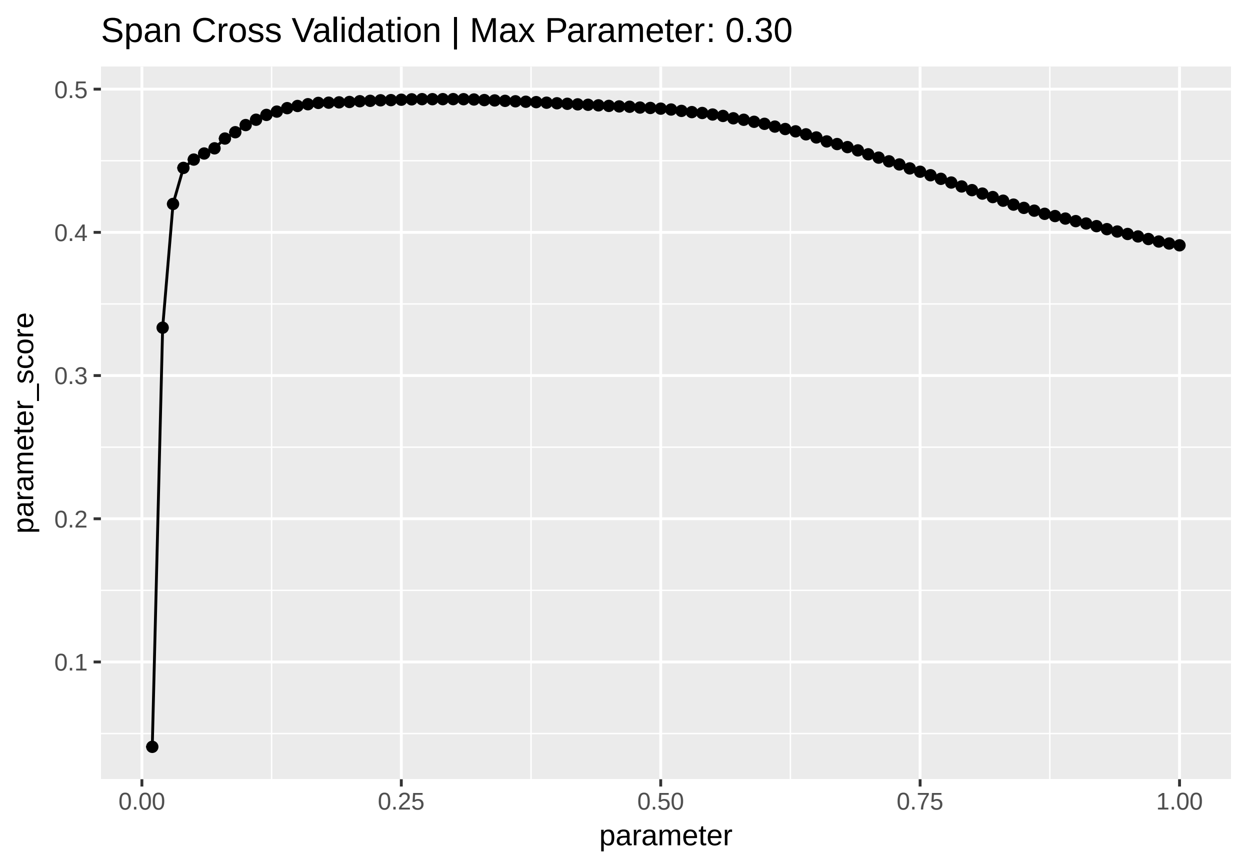
```
# Fit, test, and plot a smoothing spline model
smooth.spline_model <- smooth.spline(train_dataset1_clean$TimeMin, train_dataset1_clean$PickupCount)
model_evaluate("Smooth Spline Model", smooth.spline_model, train_dataset1_clean, test_dataset1_clean)
```



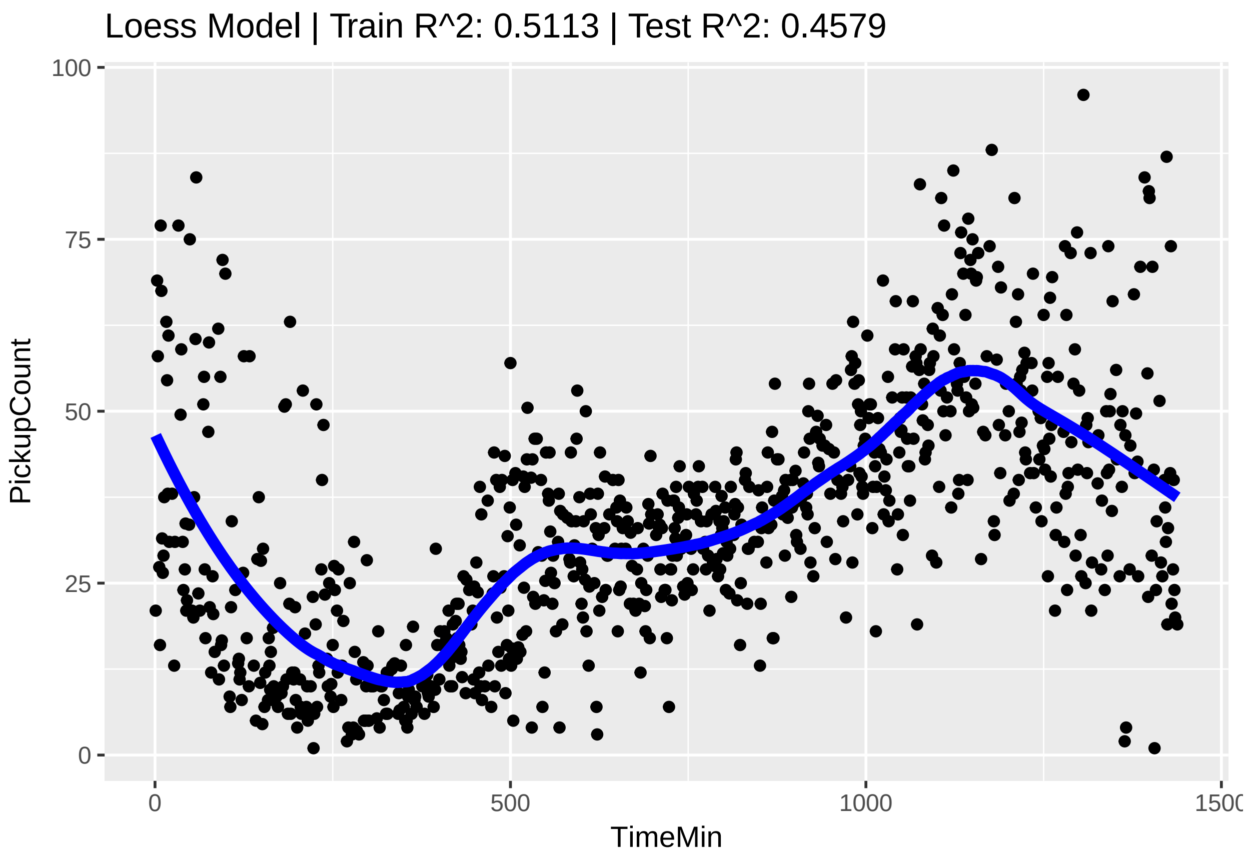
The smoothing spline model with internal cross-validation actually produces a lower test R^2 compared to the previously examined models despite producing a similar train R^2 indicating the model may be over-fitting. This is somewhat surprising given that the model visually appears to fit the data pretty well.

3.Locally-Weighted Regression Model

```
# Cross validate the span parameter for a loess model and plot it
span_to_validate <- c(1.:100.)/100.
loess_cv_scores <- crossval_loess(train_dataset1_clean, span_to_validate, 5)
span_max_parameter <- span_to_validate[which.max(loess_cv_scores)]
ggplot(data = data.frame(parameter = span_to_validate, parameter_score = loess_cv_scores), aes(x = para
```



```
# Fit, test, and plot a loess model
loess_model <- loess(PickupCount ~ TimeMin, span = span_max_parameter, data = train_dataset1_clean, con
model_evaluate("Loess Model", loess_model, train_dataset1_clean, test_dataset1_clean)
```



The best smoothing parameter for the model is relatively low and the model starts to over-fit pretty quickly as it increases. In this case the loess function perform better in terms of test R^2 than all other models besides the natural cubic spline, however it's fit appears much more intuitive visually compared to the natural cubic spline.

Model Choice

In this case the loess model would be preferred because even though it has a slightly lower R^2 than the natural cubic spline, it appears to fit the data much better as the large number of curves in the natural cubic spline indicate potential over-fitting that could cause problems for real-world use.

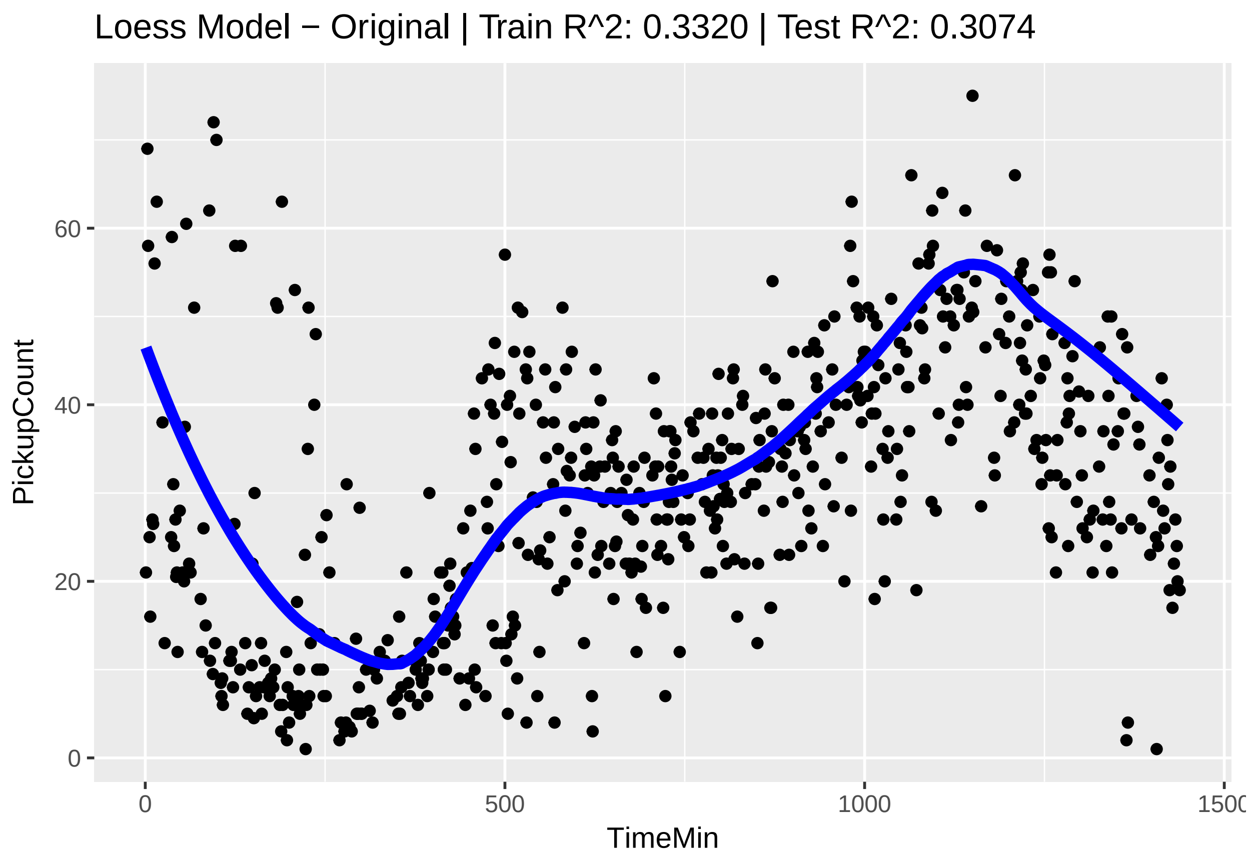
Part 1b: Adapting to weekends

```
# Filter for only weekly data and get average taxi PickupCount for training set
train_dataset1_week <- train_dataset1[train_dataset1$DayOfWeek < 6,]
train_dataset1_week_clean <- aggregate(train_dataset1_week[, 3], list(TimeMin = train_dataset1_week$Tim
colnames(train_dataset1_week_clean)[2] <- "PickupCount"

# Filter for only weekend data and get average taxi PickupCount for training set
train_dataset1_weekend <- train_dataset1[train_dataset1$DayOfWeek > 5,]
train_dataset1_weekend_clean <- aggregate(train_dataset1_weekend[, 3], list(TimeMin = train_dataset1_we
colnames(train_dataset1_weekend_clean)[2] <- "PickupCount"

# Filter for only weekly data and get average taxi PickupCount for testing set
test_dataset1_week <- test_dataset1[test_dataset1$DayOfWeek < 6,]
```

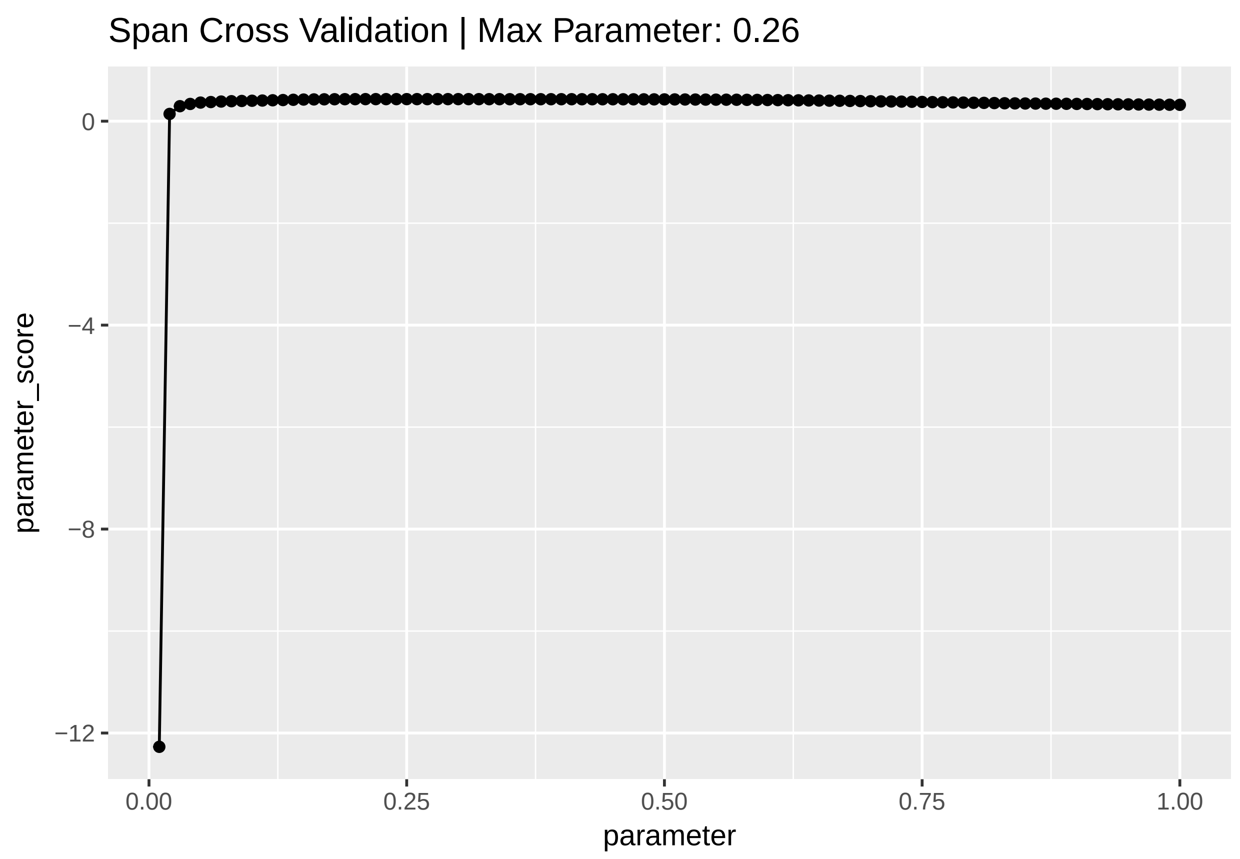
```r
test_dataset1_week_clean <- aggregate(test_dataset1_week[, 3], list(TimeMin = test_dataset1_week$TimeMi
colnames(test_dataset1_week_clean)[2] <- "PickupCount"

# Filter for only weekend data and get average taxi PickupCount for testing set
test_dataset1_weekend <- test_dataset1[test_dataset1$DayOfWeek > 5,]
test_dataset1_weekend_clean <- aggregate(test_dataset1_weekend[, 3], list(TimeMin = test_dataset1_weeke
colnames(test_dataset1_weekend_clean)[2] <- "PickupCount"
```

Predicting Weekday Pickups

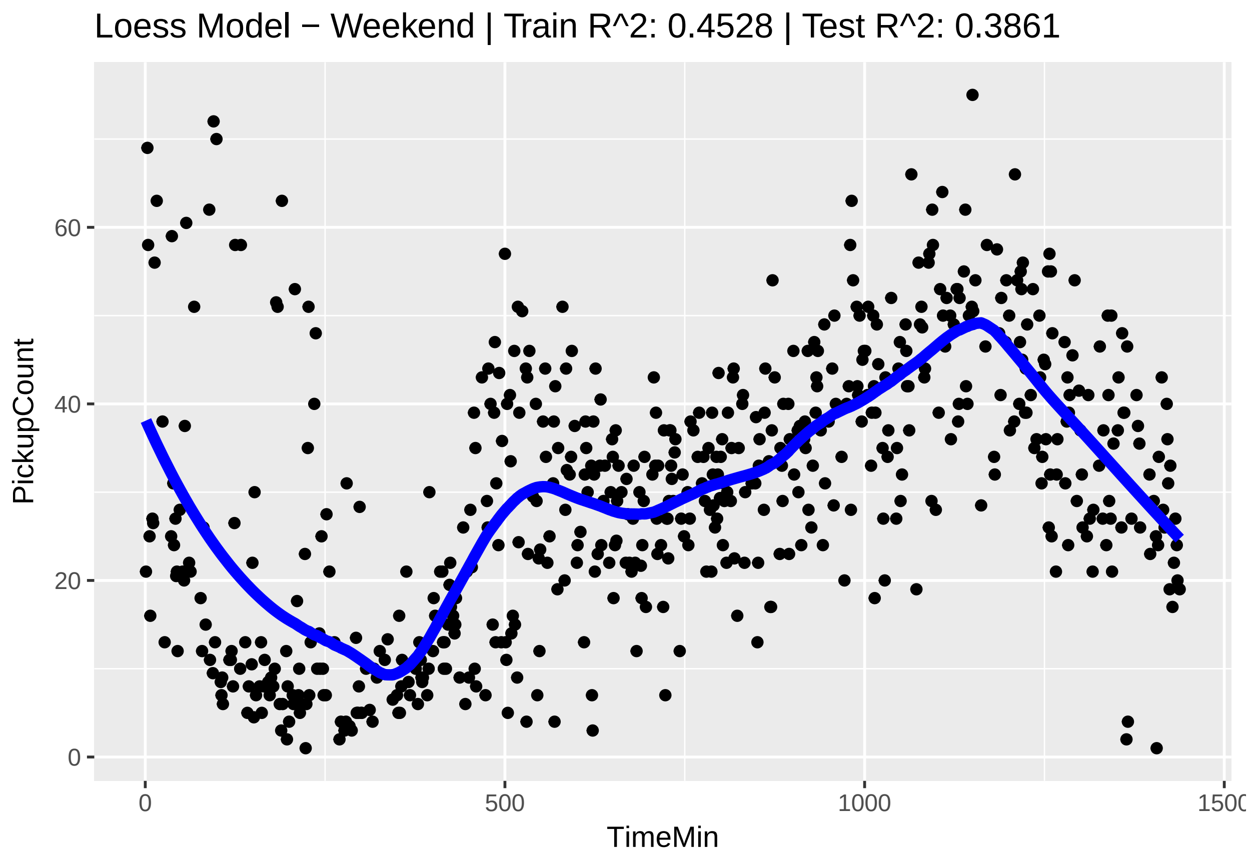
```r
# Test and plot the orignial Loess model on the weekday data
model_evaluate("Loess Model - Original", loess_model, train_dataset1_week_clean, test_dataset1_week_cle
```



```r
# Tune the span parameter through cross validation for the loess model fit only on weekday data
span_to_validate <- c(1.:100.)/100.
loess_cv_scores_week <- crossval_loess(train_dataset1_week_clean, span_to_validate, 5)
span_week_max_parameter <- span_to_validate[which.max(loess_cv_scores_week)]
ggplot(data = data.frame(parameter = span_to_validate, parameter_score = loess_cv_scores_week), aes(x =
```



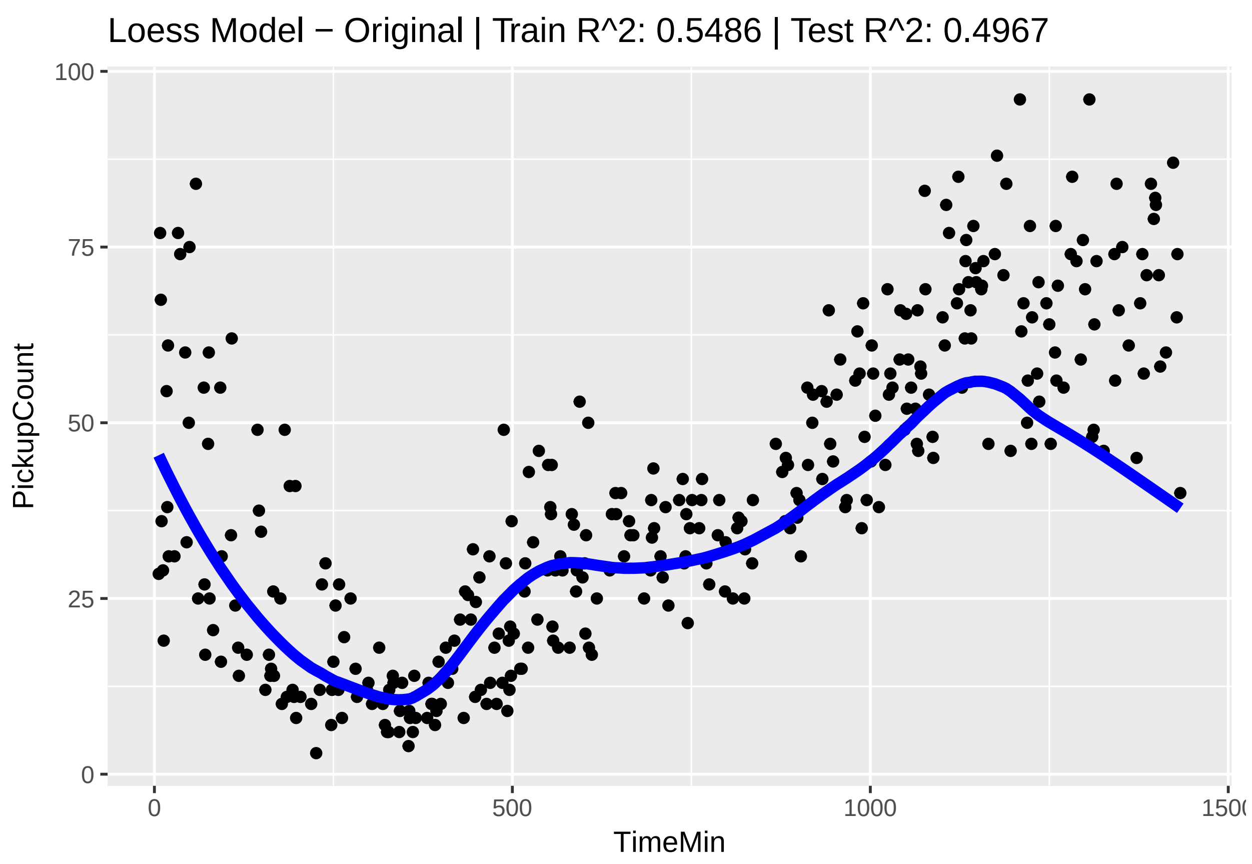
```r
# Fit, test, and plot the Loess model on the weekday data
loess_model_week <- loess(PickupCount ~ TimeMin, span = span_week_max_parameter, data = train_dataset1_w
model_evaluate("Loess Model - Weekend", loess_model_week, train_dataset1_week_clean, test_dataset1_week_
```



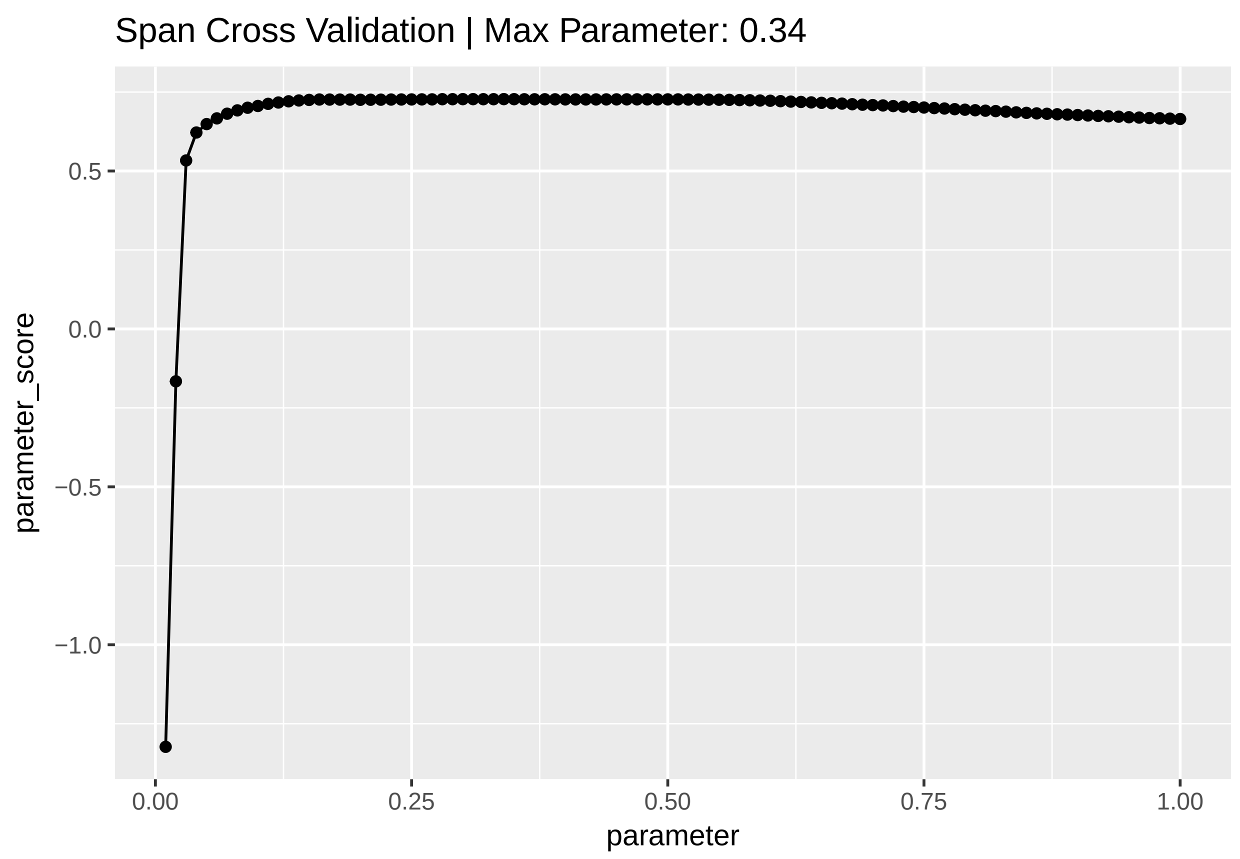
While both test R^2s are relatively low compared to the overall model that doesn't separate the type of day, the loess model fit on the weekday set does perform significantly better than the model fit on the entire dataset.

Predicting Weekend Pickups

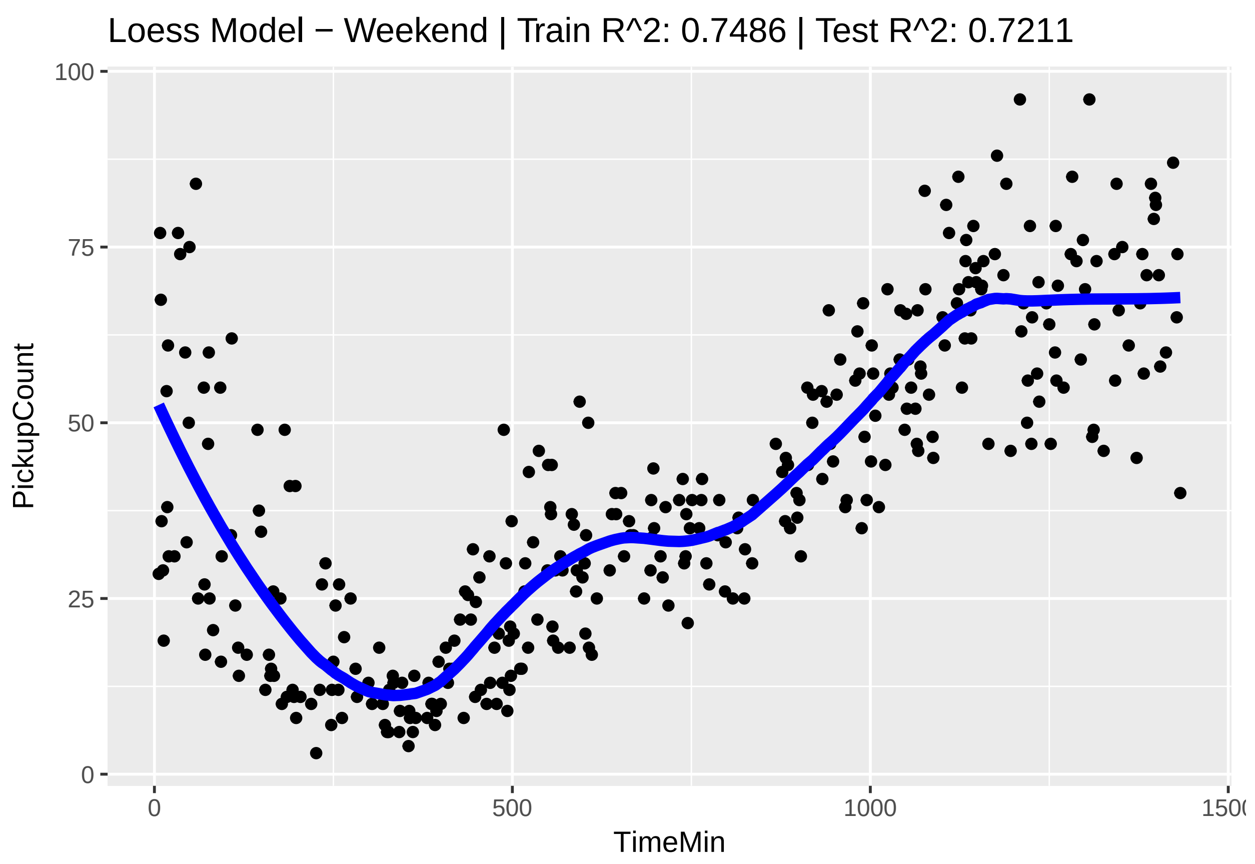
```
# Test and plot the orignial Loess model on the weekend data
model_evaluate("Loess Model - Original", loess_model, train_dataset1_weekend_clean, test_dataset1_weeken
```



```
# Tune the span parameter through cross validation for the loess model fit only on weekdend data
span_to_validate <- c(1.:100.)/100.
loess_cv_scores_weekend <- crossval_loess(train_dataset1_weekend_clean, span_to_validate, 5)
span_weekend_max_parameter <- span_to_validate[which.max(loess_cv_scores_weekend)]
ggplot(data = data.frame(parameter = span_to_validate, parameter_score = loess_cv_scores_weekend), aes(
```



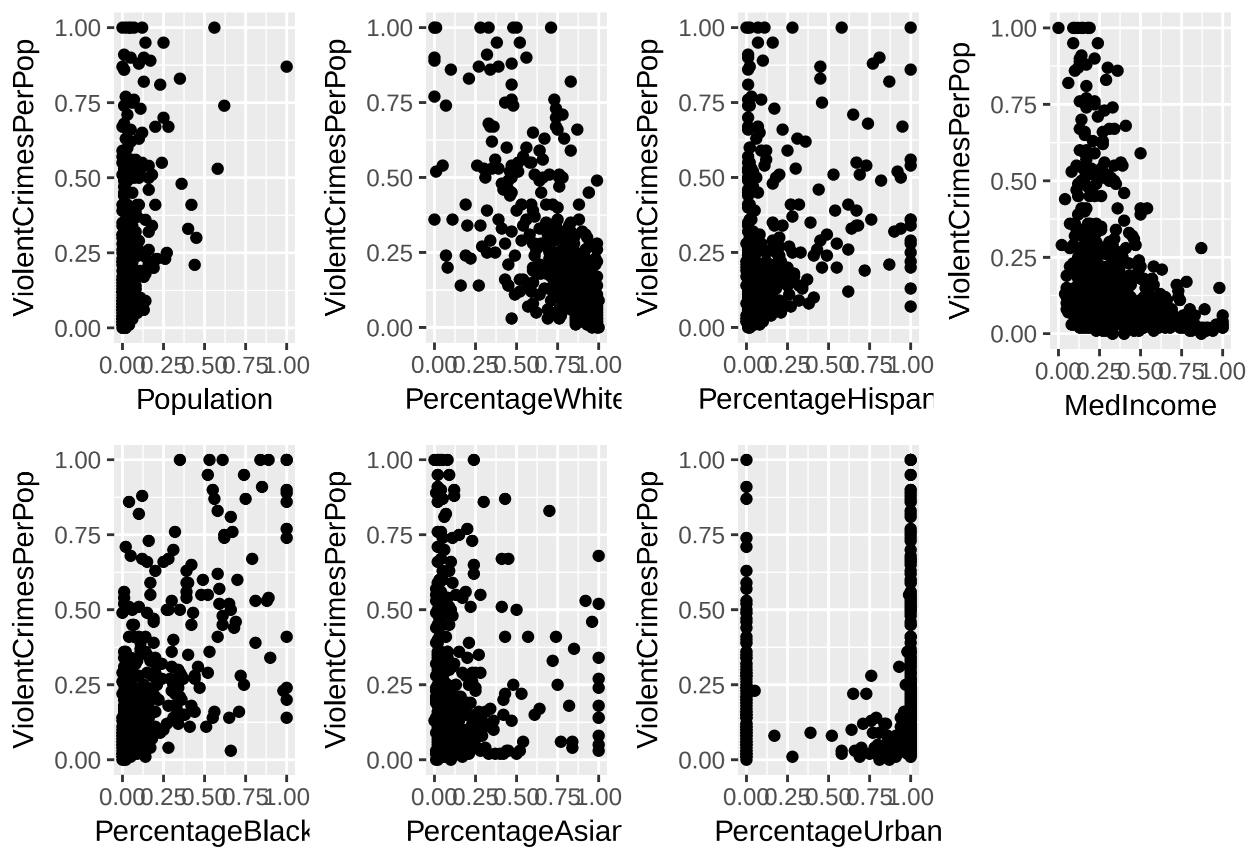
```
# Fit, test, and plot the Loess model on the weekend data
loess_model_weekend <- loess(PickupCount ~ TimeMin, span = span_weekend_max_parameter, data = train_data
model_evaluate("Loess Model - Weekend", loess_model_weekend, train_dataset1_weekend_clean, test_dataset
```



The loess model fit to the weekend data performs significantly better than the model fit to the entire dataset. It's also clear that both models perform better on weekend vs. weekday data, and the combination of the models fit separately to the split dataset perform better than the overall model. Overall, this confirms the visual evidence that taxi pickups do differ over days of the week.

Problem 2: Predicting Crime in the City

```r
# Load the training and testing dataset
train_dataset2 <- read.table("dataset_2_train.txt",header = TRUE)
test_dataset2 <- read.table("dataset_2_test.txt",header = TRUE)

# Generate and populate graphs of each predictor against the outcome
graph.pop <- ggplot(data = train_dataset2, mapping = aes(x = Population, y=ViolentCrimesPerPop)) + geom_
graph.pctblack <- ggplot(data = train_dataset2, mapping = aes(x = PercentageBlack, y=ViolentCrimesPerPop
graph.pctwhite <- ggplot(data = train_dataset2, mapping = aes(x = PercentageWhite, y=ViolentCrimesPerPop
graph.pctasian <- ggplot(data = train_dataset2, mapping = aes(x = PercentageAsian, y=ViolentCrimesPerPop
graph.pcthispanic <- ggplot(data = train_dataset2, mapping = aes(x = PercentageHispanic, y=ViolentCrime
graph.pcturban <- ggplot(data = train_dataset2, mapping = aes(x = PercentageUrban, y=ViolentCrimesPerPop
graph.medincome <- ggplot(data = train_dataset2, mapping = aes(x = MedIncome, y=ViolentCrimesPerPop)) +

multiplot(graph.pop, graph.pctblack, graph.pctwhite, graph.pctasian, graph.pcthispanic, graph.pcturban,
```



There looks to be a log-type relationship between Median Income and Violent Crimes, perhaps a negative non-linear relationship between Percentage White and Violent Crimes, and some non-linear relationship between Percentage Urban and Violent Crimes, though generally the distribution of Percentage Urban is bi-modal.

Part 2a: Polynomial regression

```
# Fit and Score Linear Model
model.linear <- lm(ViolentCrimesPerPop ~ Population + PercentageBlack + PercentageWhite + PercentageAsia
model.linear.test_predictions <- predict(model.linear, newdata = test_dataset2)
model.linear.test_r2 <- rsq(test_dataset2$ViolentCrimesPerPop, model.linear.test_predictions)

# Fit and Score Polynomial Degree - 2
model.poly2 <- lm(ViolentCrimesPerPop ~ poly(Population + PercentageBlack + PercentageWhite + Percentag
model.poly2.test_predictions <- predict(model.poly2, newdata = test_dataset2)
model.poly2.test_r2 <- rsq(test_dataset2$ViolentCrimesPerPop, model.poly2.test_predictions)

# Fit and Score Polynomial Degree - 3
model.poly3 <- lm(ViolentCrimesPerPop ~ poly(Population + PercentageBlack + PercentageWhite + Percentag
model.poly3.test_predictions <- predict(model.poly3, newdata = test_dataset2)
model.poly3.test_r2 <- rsq(test_dataset2$ViolentCrimesPerPop, model.poly3.test_predictions)

# Fit and Score BSplines
model.bs <- lm(ViolentCrimesPerPop ~ bs(Population + PercentageBlack + PercentageWhite + PercentageAsian
model.bs.test_predictions <- predict(model.bs, newdata = test_dataset2)
model.bs.test_r2 <- rsq(test_dataset2$ViolentCrimesPerPop, model.bs.test_predictions)

# Print Output
print(sprintf("Linear Model R^2: %.4f", model.linear.test_r2))
```

```
## [1] "Linear Model R^2: 0.5554"
```

```
print(sprintf("Polynomial Degree 2 R^2: %.4f", model.poly2.test_r2))
```

```
## [1] "Polynomial Degree 2 R^2: 0.0149"
```

```
print(sprintf("Polynomial Degree 3 R^2: %.4f", model.poly3.test_r2))
```

```
## [1] "Polynomial Degree 3 R^2: 0.0089"
```

```
print(sprintf("Basis Splines R^2: %.4f", model.bs.test_r2))
```

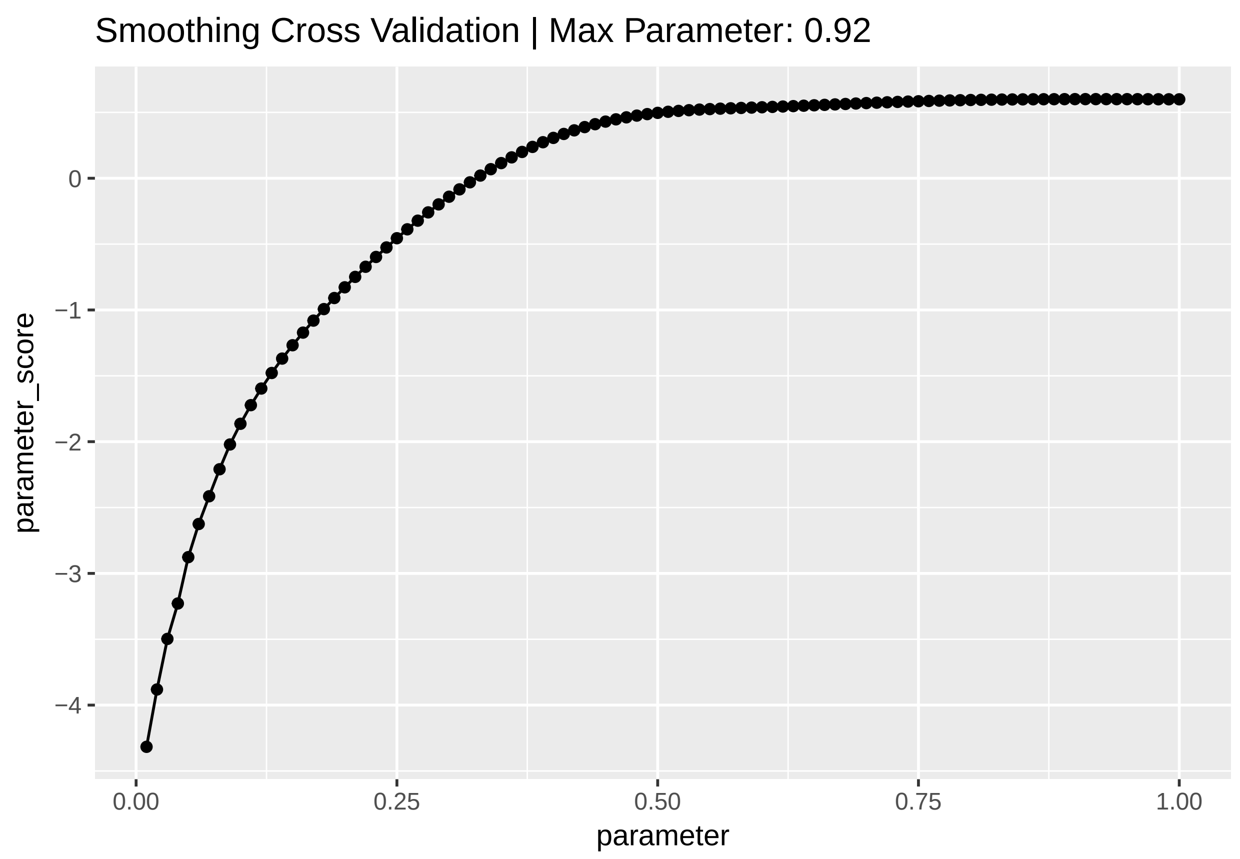
```
## [1] "Basis Splines R^2: 0.0089"
```

The model with linear predictors performs significantly better than the models with non-linear predictors.

Part 2b: Generalized Additive Model (GAM)

1.Fit GAM Model

```
# Tune the span parameter through cross validation for all predictors in the GAM model
spar_to_validate <- c(1.:100.)/100.
gam_cv_scores <- crossval_gams(train_dataset2, spar_to_validate, 5)
spar_max_parameter <- spar_to_validate[which.max(gam_cv_scores)]
ggplot(data = data.frame(parameter = spar_to_validate, parameter_score = gam_cv_scores), aes(x = parame
```



```
# Fit and test the GAM model
model.gam_formula <- as.formula(sprintf("ViolentCrimesPerPop ~ s(Population, spar = %1$f) + s(Percentag
model.gam <- gam(model.gam_formula, data = train_dataset2)
gam_test_predictions <- predict(model.gam, newdata = test_dataset2)
gam_test_r2 <- rsq(test_dataset2$ViolentCrimesPerPop, gam_test_predictions)
print(sprintf("GAM Test R^2: %.4f", gam_test_r2))
```

```
## [1] "GAM Test R^2: 0.5762"
```

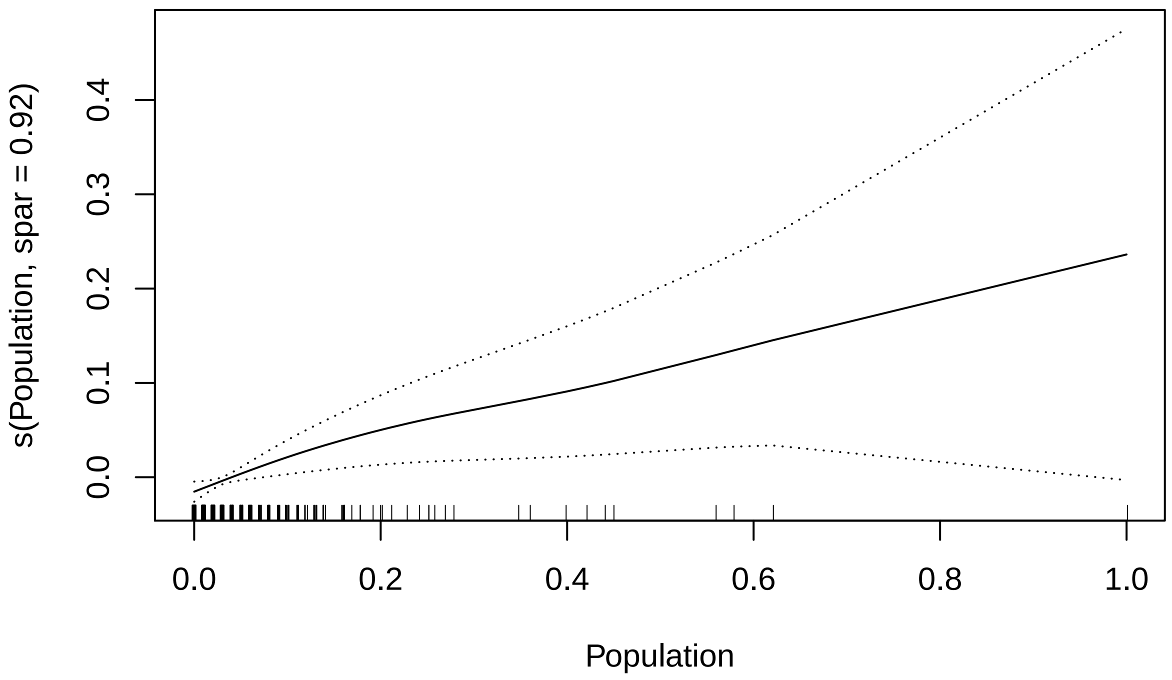
The smoothing parameter chosen through cross-validation is relatively high and the GAM model with smoothed predictors performs slightly better than the model with linear predictors.

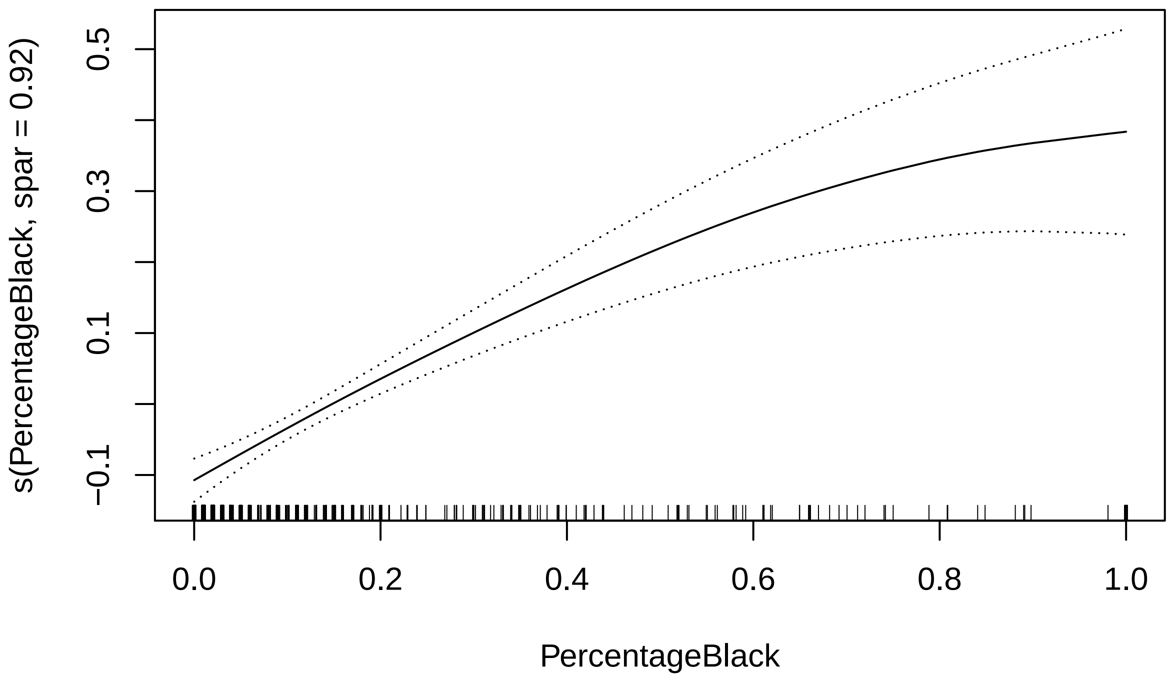
2.Plot GAM Model

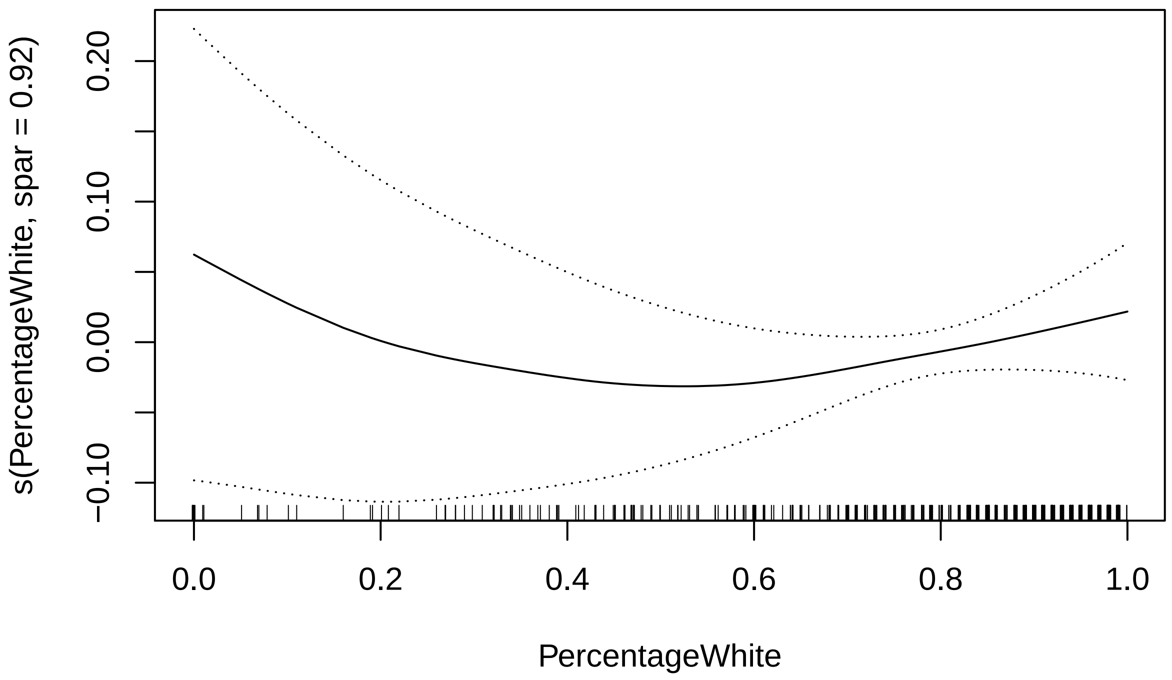
```
# Print GAM coefficients and plot the smoothed predictors
model.gam$coefficients
```

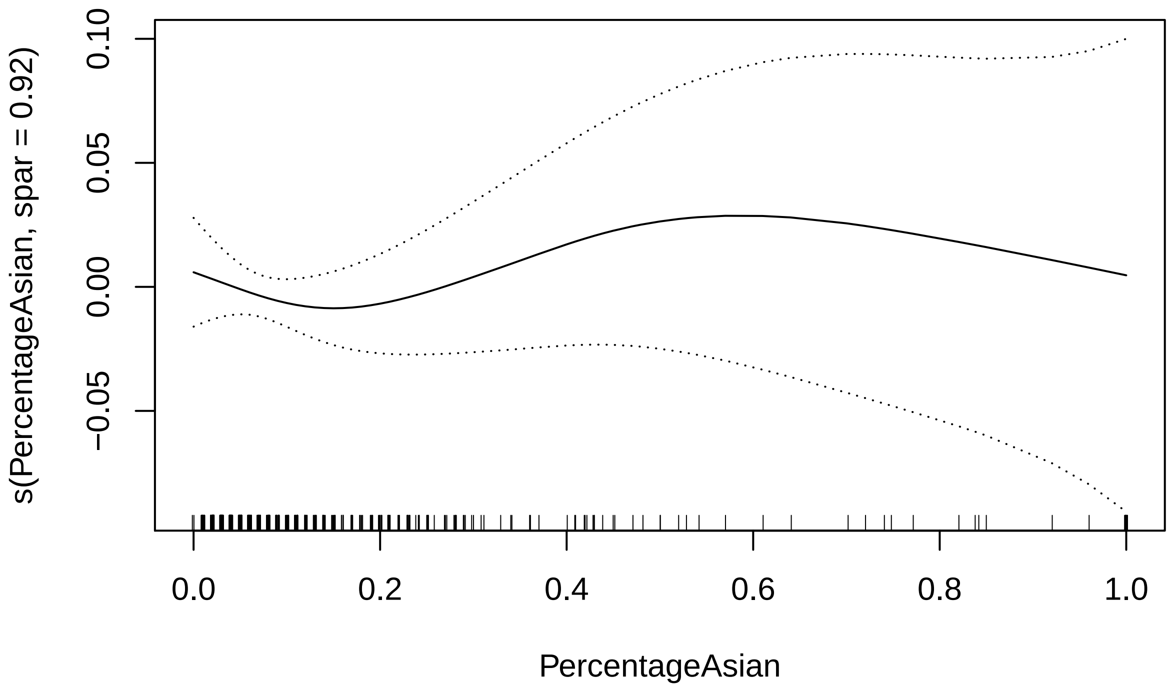
```
##                          (Intercept)         s(Population, spar = 0.92)
##                           0.08723918                         0.27592968
##     s(PercentageBlack, spar = 0.92)    s(PercentageWhite, spar = 0.92)
##                           0.55674798                         0.02740143
##     s(PercentageAsian, spar = 0.92) s(PercentageHispanic, spar = 0.92)
##                           0.01748142                         0.30583728
##     s(PercentageUrban, spar = 0.92)          s(MedIncome, spar = 0.92)
##                           0.07113980                        -0.22706765
```

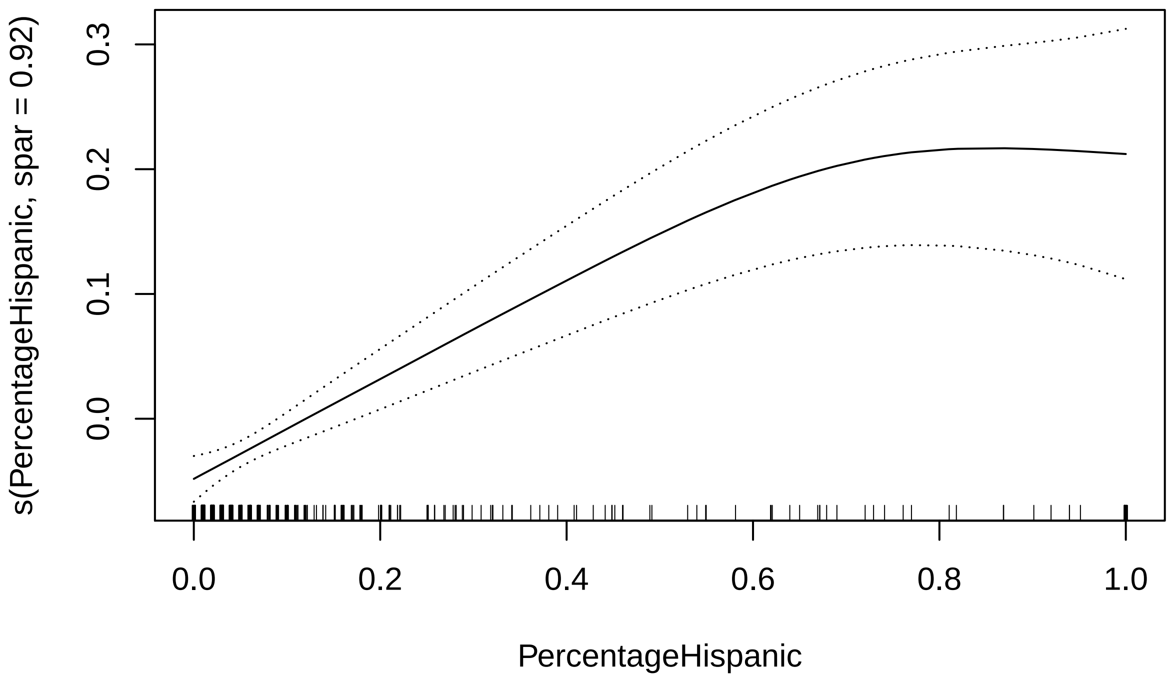
```
plot(model.gam, se=TRUE)
```

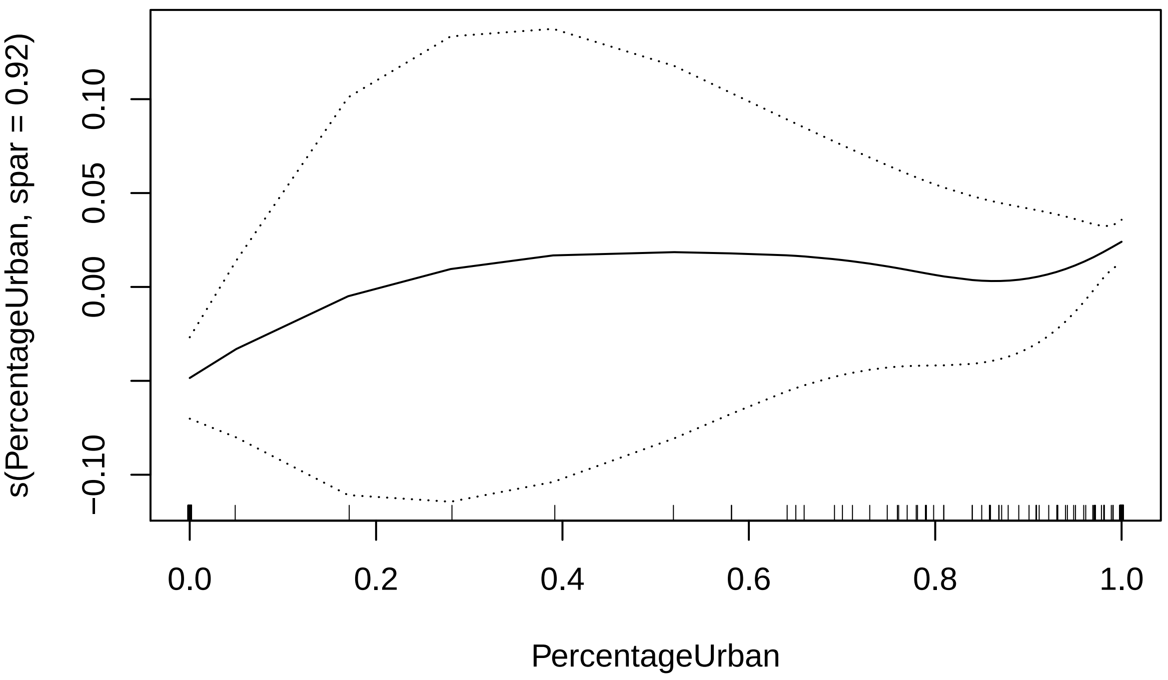

s(Population, spar = 0.92)
0.0
0.1
0.2
0.3
0.4
0.0
0.2
0.4
0.6
0.8
1.0
Population

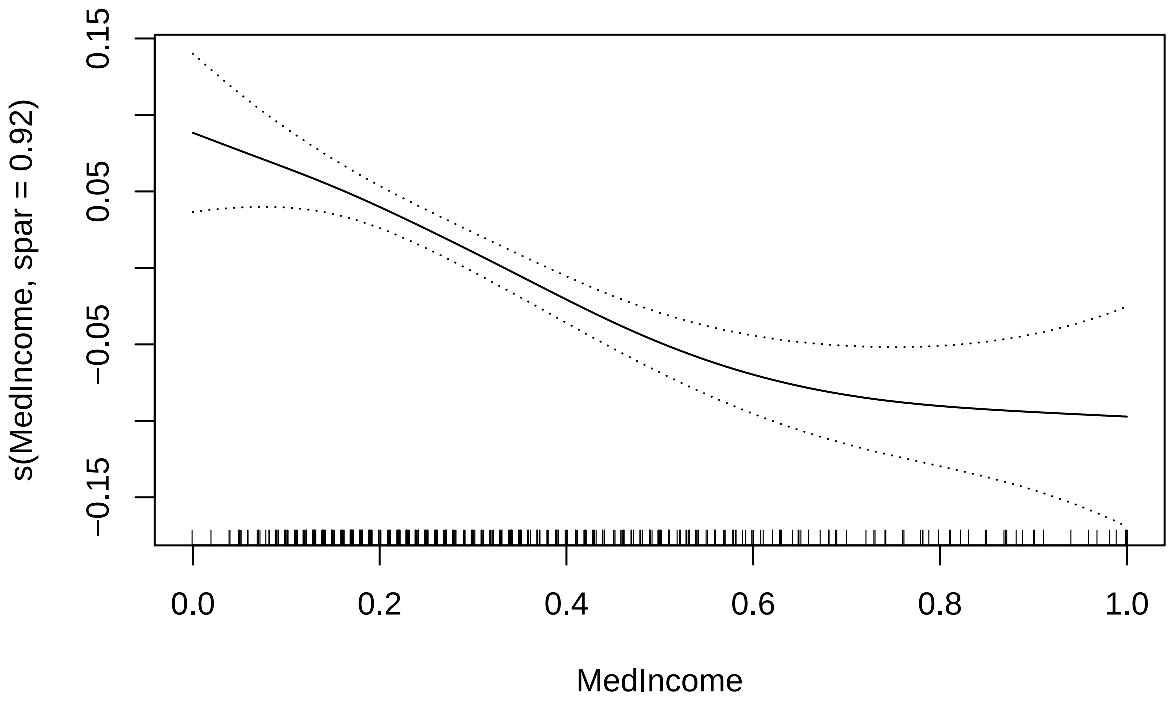

s(PercentageBlack, spar = 0.92)
−0.1
0.1
0.3
0.5
0.0
0.2
0.4
0.6
0.8
1.0
PercentageBlack


s(PercentageWhite, spar = 0.92)
−0.10
0.00
0.10
0.20
0.0
0.2
0.4
0.6
0.8
1.0
PercentageWhite


s(PercentageAsian, spar = 0.92)
0.10
0.05
0.00
−0.05
0.0
0.2
0.4
0.6
0.8
1.0
PercentageAsian


s(PercentageHispanic, spar = 0.92)
0.0
0.1
0.2
0.3
0.0
0.2
0.4
0.6
0.8
1.0
PercentageHispanic


s(PercentageUrban, spar = 0.92)
0.10
0.05
0.00
−0.10
0.0
0.2
0.4
0.6
0.8
1.0
PercentageUrban



All predictors aside from median income are positively associated with the Violent Crime rate, with Population, PercentageBlack, and PercentageHispanic having the largest coefficients. From the visual inspection of the dataset I assumed PercentageWhite would have a negative coefficient, but the smoothing plot shows that while the relationship is initially negative, it turns positive at higher percentages with much lower standard errors, which is actually the reverse of the other race-based metrics. It's important to note that these racially-based predictors are almost certainly correlated with unobservable metrics and bear further investigation or the model will be racially-biased. Median Income's coefficient and plot align with intuition from the initial visualization.

3.GAM Model Comparison

```
# Compare linear and GAM model with an ANOVA test
print(anova(model.linear, model.gam, test = "Chi"))
```

```
## Analysis of Variance Table
##
## Model 1: ViolentCrimesPerPop ~ Population + PercentageBlack + PercentageWhite +
##     PercentageAsian + PercentageHispanic + PercentageUrban +
##     MedIncome
## Model 2: ViolentCrimesPerPop ~ s(Population, spar = 0.92) + s(PercentageBlack,
##     spar = 0.92) + s(PercentageWhite, spar = 0.92) + s(PercentageAsian,
##     spar = 0.92) + s(PercentageHispanic, spar = 0.92) + s(PercentageUrban,
##     spar = 0.92) + s(MedIncome, spar = 0.92)
##   Res.Df     RSS     Df Sum of Sq Pr(>Chi)
## 1 490.00 10.1458
## 2 478.15  9.5852 11.855   0.56064 0.005189 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The difference between the GAM and linear model is small but statistically significant at the 99% level, with the GAM model performing slightly better.

```
# Fit a GAM model excluding PercentageAsian and PercentageUrban
model.gam_adj_formula <- as.formula(sprintf("ViolentCrimesPerPop ~ s(Population, spar = %1$f) + s(Percen
model.gam_adj <- gam(model.gam_adj_formula, data = train_dataset2)

# Compare the original GAM model with the GAM ex PctAsian and PctUrban
print(anova(model.gam, model.gam_adj, test = "Chi"))
```

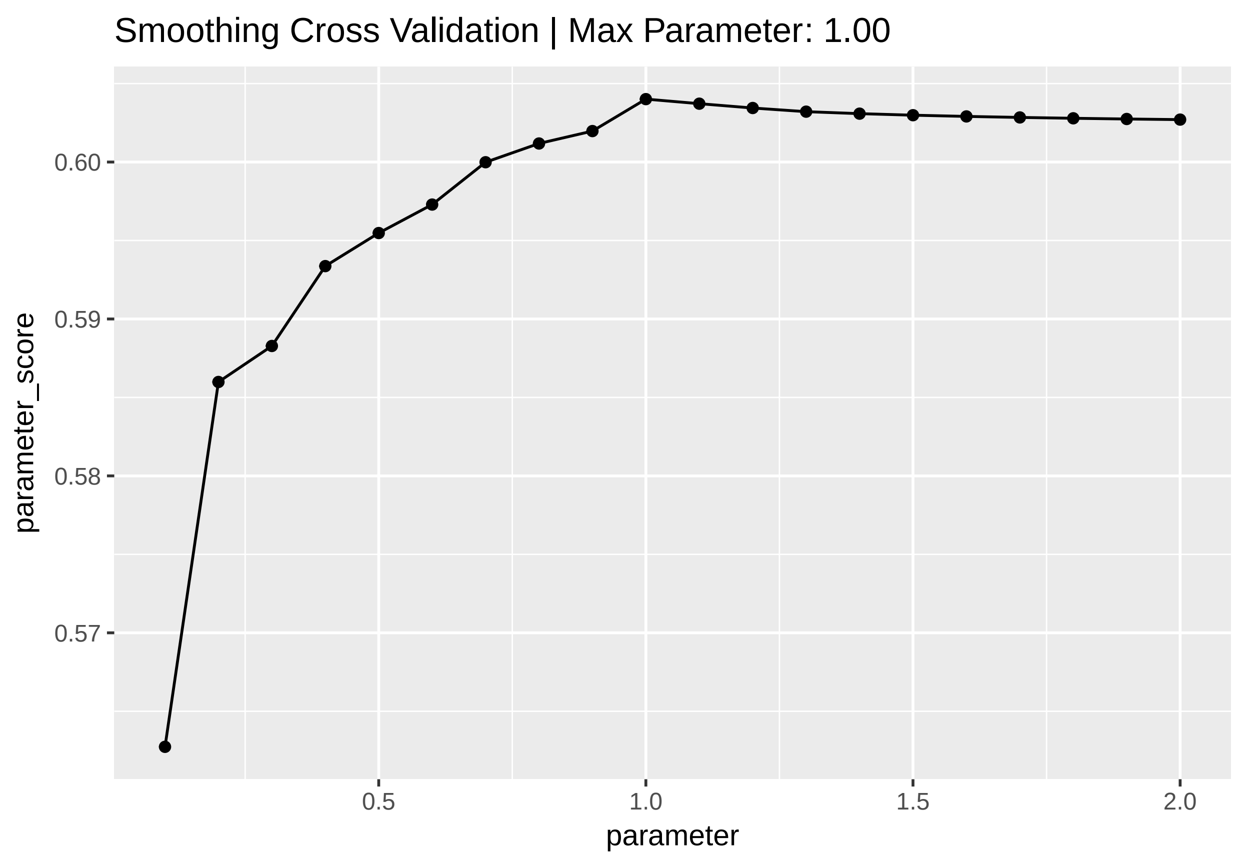
```
## Analysis of Deviance Table
##
## Model 1: ViolentCrimesPerPop ~ s(Population, spar = 0.92) + s(PercentageBlack,
##     spar = 0.92) + s(PercentageWhite, spar = 0.92) + s(PercentageAsian,
##     spar = 0.92) + s(PercentageHispanic, spar = 0.92) + s(PercentageUrban,
##     spar = 0.92) + s(MedIncome, spar = 0.92)
## Model 2: ViolentCrimesPerPop ~ s(Population, spar = 0.92) + s(PercentageBlack,
##     spar = 0.92) + s(PercentageWhite, spar = 0.92) + s(PercentageHispanic,
##     spar = 0.92) + s(MedIncome, spar = 0.92)
##   Resid. Df Resid. Dev     Df Deviance  Pr(>Chi)
## 1    478.15     9.5852
## 2    483.95    10.0394 -5.805  -0.4542 0.0007854 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The GAM model without the excluded terms performs better and the difference is statistically significant at the 99.9% level, indicating there is some predictive value to the PercentageAsian and PercentageUrban predictors.

Part 2c: GAM with Interaction Terms

GAM with Population, PercentageUrban, and Median Income Interaction Term

```
# Tune the span parameter through cross validation for the interaction predictor in the GAM model
span_to_validate_gam_lo1 <- c(1.:20.)/10.
gam_lo1_cv_scores <- crossval_gams_lo1(train_dataset2, span_to_validate_gam_lo1, spar_max_parameter, 5)
span_gam_lo1_max_parameter <- span_to_validate_gam_lo1[which.max(gam_lo1_cv_scores)]
ggplot(data = data.frame(parameter = span_to_validate_gam_lo1, parameter_score = gam_lo1_cv_scores), aes
```



```r
# Fit and test gam model with Population, PercentageUrban, and MedIncome interaction term
model.gam_lo1_formula <- as.formula(sprintf("ViolentCrimesPerPop ~ s(Population, spar = %1$f) + s(Percen
model.gam_lo1 <- gam(model.gam_lo1_formula, data = train_dataset2)
gam_lo1_test_predictions <- predict(model.gam_lo1, newdata = test_dataset2)
gam_lo1_test_r2 <- rsq(test_dataset2$ViolentCrimesPerPop, gam_lo1_test_predictions)
print(sprintf("GAM Test R^2: %.4f", gam_lo1_test_r2))
```

```
## [1] "GAM Test R^2: 0.5704"
```

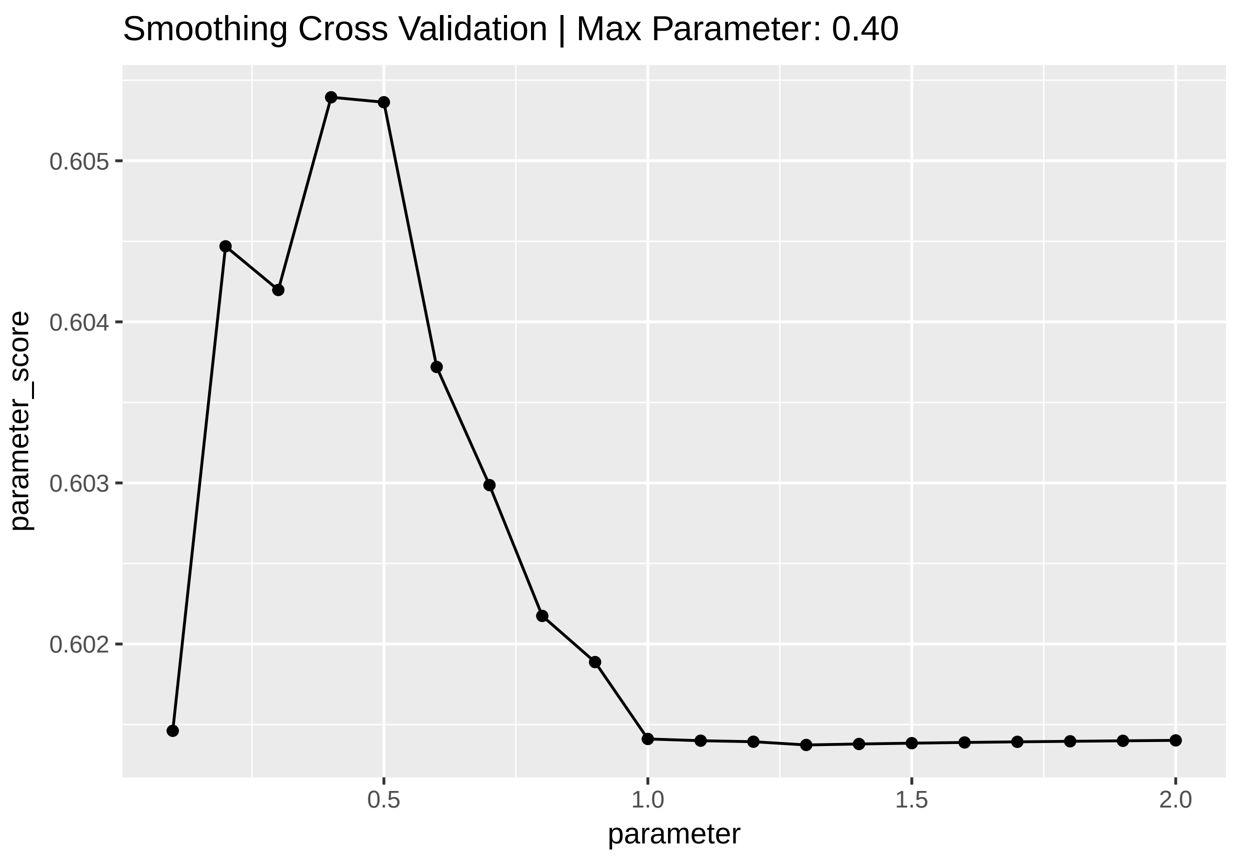
```r
print(anova(model.gam, model.gam_lo1, test = "Chi"))
```

```
## Analysis of Deviance Table
##
## Model 1: ViolentCrimesPerPop ~ s(Population, spar = 0.92) + s(PercentageBlack,
##     spar = 0.92) + s(PercentageWhite, spar = 0.92) + s(PercentageAsian,
##     spar = 0.92) + s(PercentageHispanic, spar = 0.92) + s(PercentageUrban,
##     spar = 0.92) + s(MedIncome, spar = 0.92)
## Model 2: ViolentCrimesPerPop ~ s(Population, spar = 0.92) + s(PercentageBlack,
##     spar = 0.92) + s(PercentageWhite, spar = 0.92) + s(PercentageAsian,
##     spar = 0.92) + s(PercentageHispanic, spar = 0.92) + s(PercentageUrban,
##     spar = 0.92) + s(MedIncome, spar = 0.92) + lo(Population *
##     PercentageUrban * MedIncome, span = 1)
##   Resid. Df Resid. Dev    Df Deviance Pr(>Chi)
## 1    478.15     9.5852
## 2    476.43     9.4574 1.719  0.12784  0.02968 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The difference in performance between the original GAM model and the GAM model with the locally-weighted regression term is very small and only statistically significant at the 95% level, so without some domain expertise to justify the inclusion of the interaction term I would be hesitant to use it.

GAM with PercentageWhite and Median Income Interaction Term

```
# Tune the span parameter through cross validation for the interaction predictor in the GAM model
span_to_validate_gam_lo2 <- c(1.:20.)/10
gam_lo2_cv_scores <- crossval_gams_lo2(train_dataset2, span_to_validate_gam_lo2, spar_max_parameter, 5)
span_gam_lo2_max_parameter <- span_to_validate_gam_lo2[which.max(gam_lo2_cv_scores)]
ggplot(data = data.frame(parameter = span_to_validate_gam_lo2, parameter_score = gam_lo2_cv_scores), ae
```



```
# Fit and test gam model with PercentageWhite and MedIncome interaction term
model.gam_lo2_formula <- as.formula(sprintf("ViolentCrimesPerPop ~ s(Population, spar = %1$f) + s(Percen
model.gam_lo2 <- gam(model.gam_lo2_formula, data = train_dataset2)
gam_lo2_test_predictions <- predict(model.gam_lo2, newdata = test_dataset2)
gam_lo2_test_r2 <- rsq(test_dataset2$ViolentCrimesPerPop, gam_lo2_test_predictions)
print(sprintf("GAM Test R^2: %.4f", gam_lo2_test_r2))
```

```
## [1] "GAM Test R^2: 0.5825"
```

```
print(anova(model.gam, model.gam_lo2, test = "Chi"))
```

```
## Analysis of Deviance Table
##
## Model 1: ViolentCrimesPerPop ~ s(Population, spar = 0.92) + s(PercentageBlack,
##     spar = 0.92) + s(PercentageWhite, spar = 0.92) + s(PercentageAsian,
##     spar = 0.92) + s(PercentageHispanic, spar = 0.92) + s(PercentageUrban,
##     spar = 0.92) + s(MedIncome, spar = 0.92)
## Model 2: ViolentCrimesPerPop ~ s(Population, spar = 0.92) + s(PercentageBlack,
##     spar = 0.92) + s(PercentageWhite, spar = 0.92) + s(PercentageAsian,
##     spar = 0.92) + s(PercentageHispanic, spar = 0.92) + s(PercentageUrban,
##     spar = 0.92) + s(MedIncome, spar = 0.92) + lo(PercentageWhite *
##     MedIncome, span = 0.4)
##   Resid. Df Resid. Dev     Df Deviance Pr(>Chi)
## 1    478.15     9.5852
## 2    473.25     9.3768 4.8912  0.20838  0.05803 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

While the test R^2 of the model with the interaction term is higher than any of the other models tested, it's performance is not statistically significant when compared to the original GAM model, and so without domain expertise justifying the inclusion of the interaction term I would still use the original model.