Stat 121B - Homework 1

Keyan Halperin

February 1, 2017

1.

```r
setwd("C:/Users/Keyan/Google Drive/School/Harvard/Stat 121B")
train1 = read.csv('dataset_1_train.txt')
test1 = read.csv('dataset_1_test.txt')
data = rbind(train1, test1)

str(data)
```

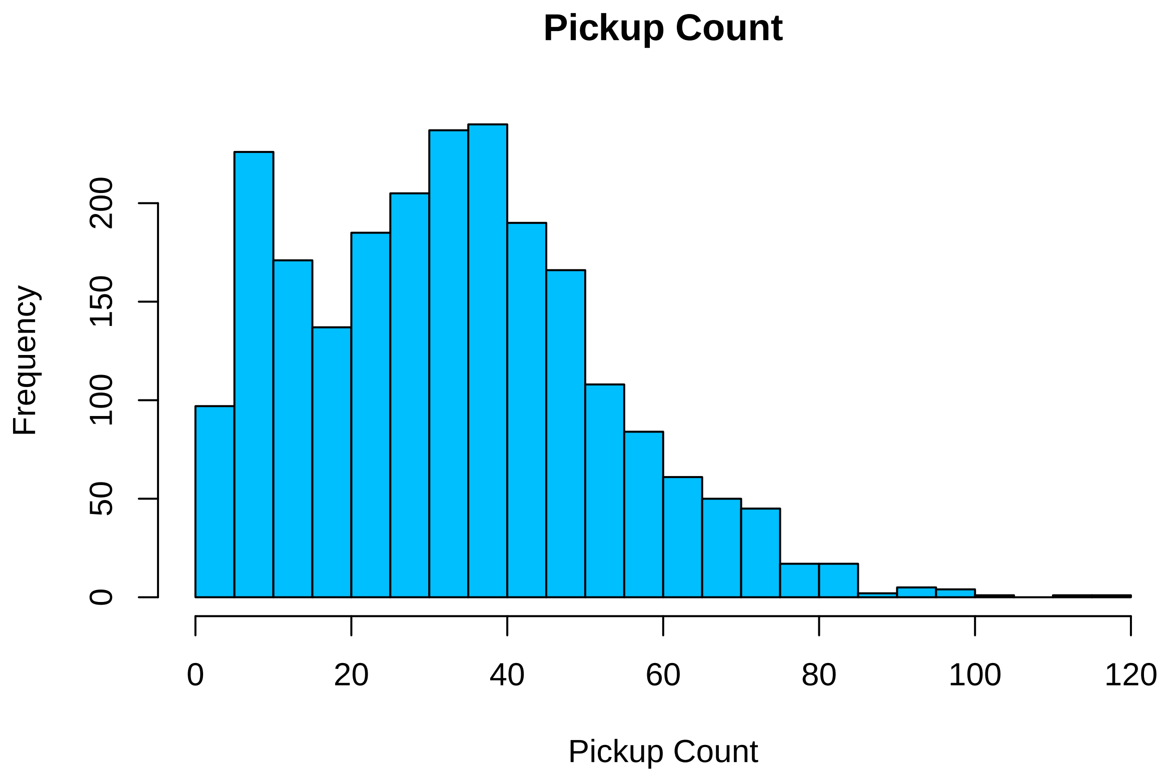
```
## 'data.frame':    2250 obs. of  3 variables:
##  $ TimeMin    : int  57 68 182 298 363 395 483 501 509 514 ...
##  $ DayOfWeek  : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ PickupCount: int  111 95 95 75 35 30 15 13 14 15 ...
```

```r
summary(data)
```

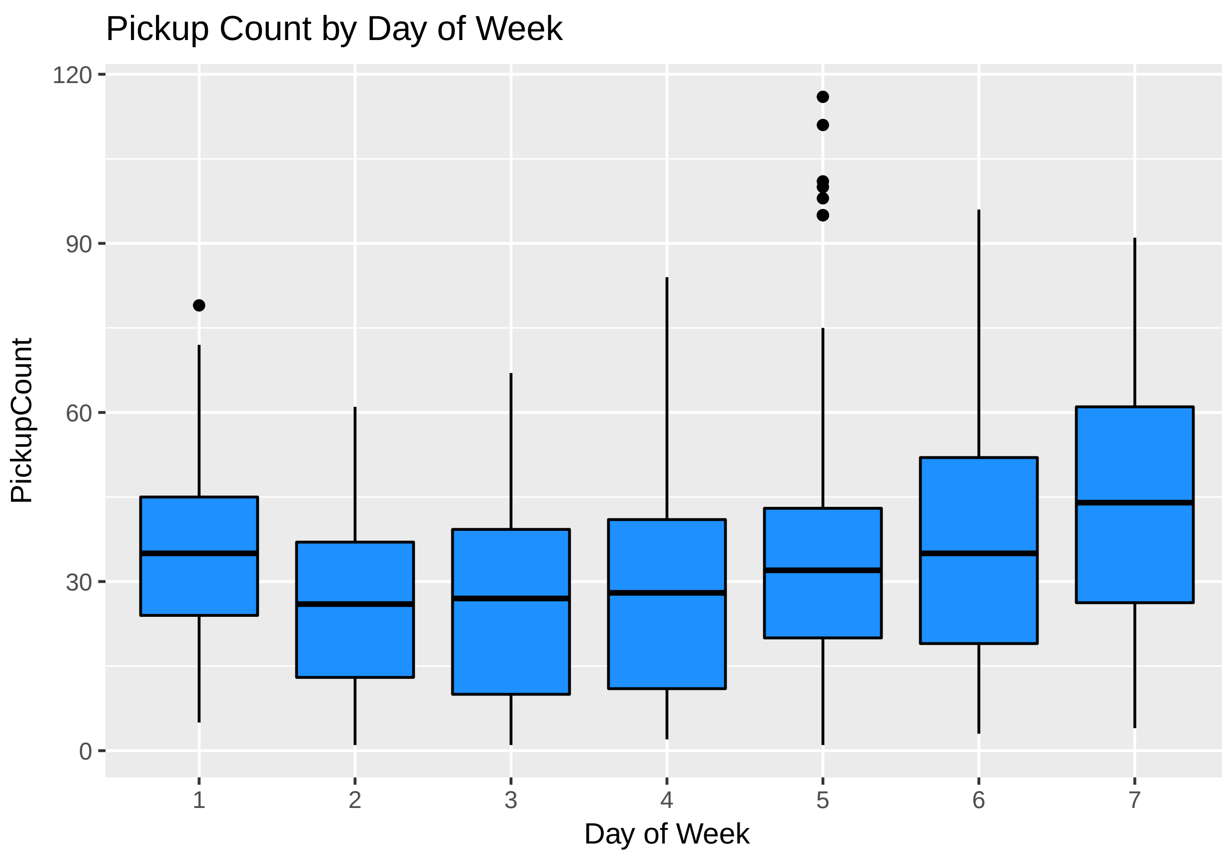
```
##     TimeMin         DayOfWeek     PickupCount
##  Min.   :   0.0   Min.   :1.00   Min.   :  1.00
##  1st Qu.: 366.0   1st Qu.:2.00   1st Qu.: 18.00
##  Median : 696.0   Median :4.00   Median : 33.00
##  Mean   : 706.4   Mean   :4.19   Mean   : 33.39
##  3rd Qu.:1050.0   3rd Qu.:6.00   3rd Qu.: 45.00
##  Max.   :1438.0   Max.   :7.00   Max.   :116.00
```

```r
hist(data$PickupCount, breaks = 20, main = 'Pickup Count', xlab = 'Pickup Count', col = 'deepskyblue')

library(ggplot2)
```

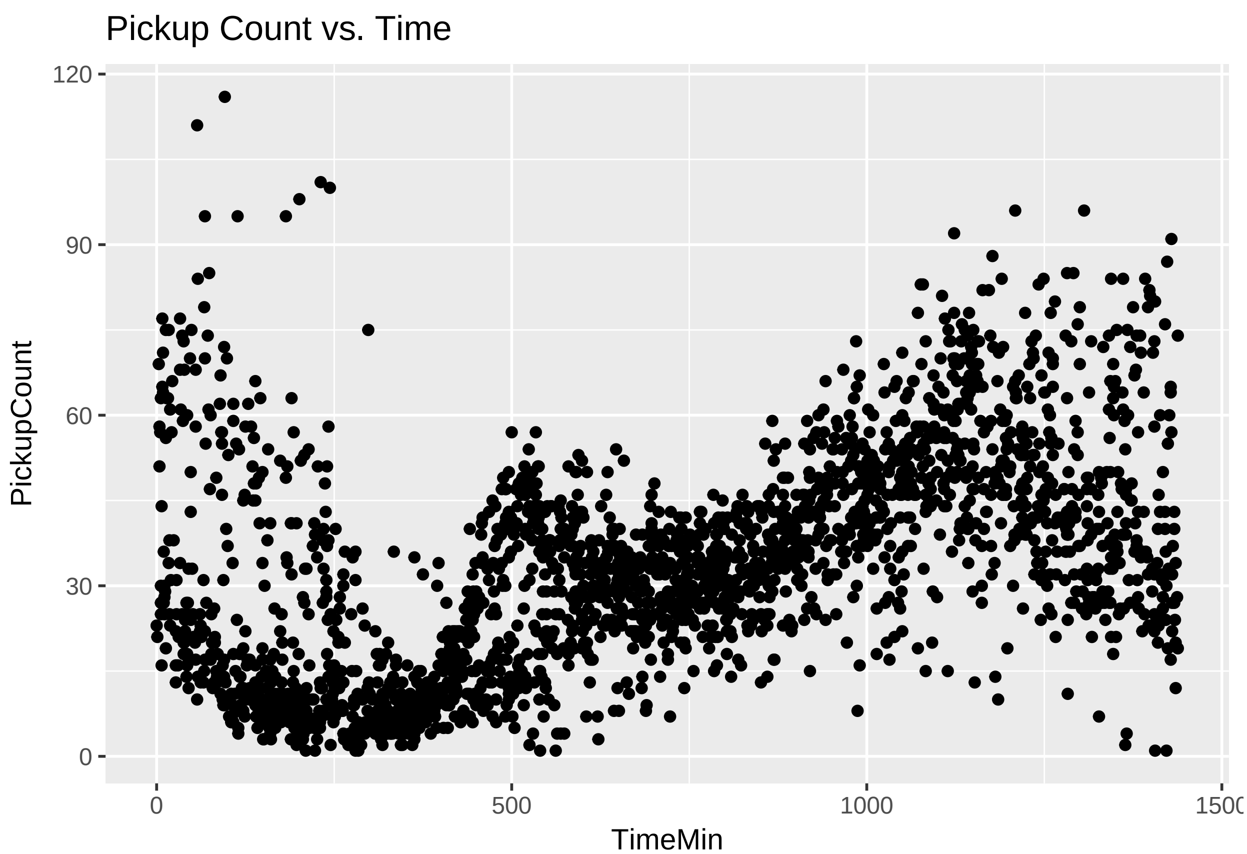
```
## Warning: package 'ggplot2' was built under R version 3.3.2
```




```
ggplot(data, aes(as.factor(DayOfWeek), PickupCount)) + geom_boxplot(fill = 'dodgerblue', color = 'black
                                                       ggtitle('Pickup Count by Day of Week') +
                                                       xlab('Day of Week')
```




```r
ggplot(data, aes(x = TimeMin, y = PickupCount)) + geom_point() + ggtitle('Pickup Count vs. Time')
```



The pattern of taxi cab pickups seems to be reasonable. Pickup count is low in the middle of the night, spikes around rush hour and at night, and is higher on the weekend.

1.a.

```r
# Function to compute R^2 for observed and predicted responses

rsq = function(model, data, y) {
  y = data[[y]]
  predict = predict(model, newdata = data)
  tss = sum((y - mean(y))^2)
  rss = sum((y-predict)^2)
  rsq_ = max(0, 1 - rss/tss)
  return(rsq_)
}
```

```r
#Degree 5 Polynomial
poly5 = lm(PickupCount ~ poly(TimeMin, 5) , data = train1)
summary(poly5)
```

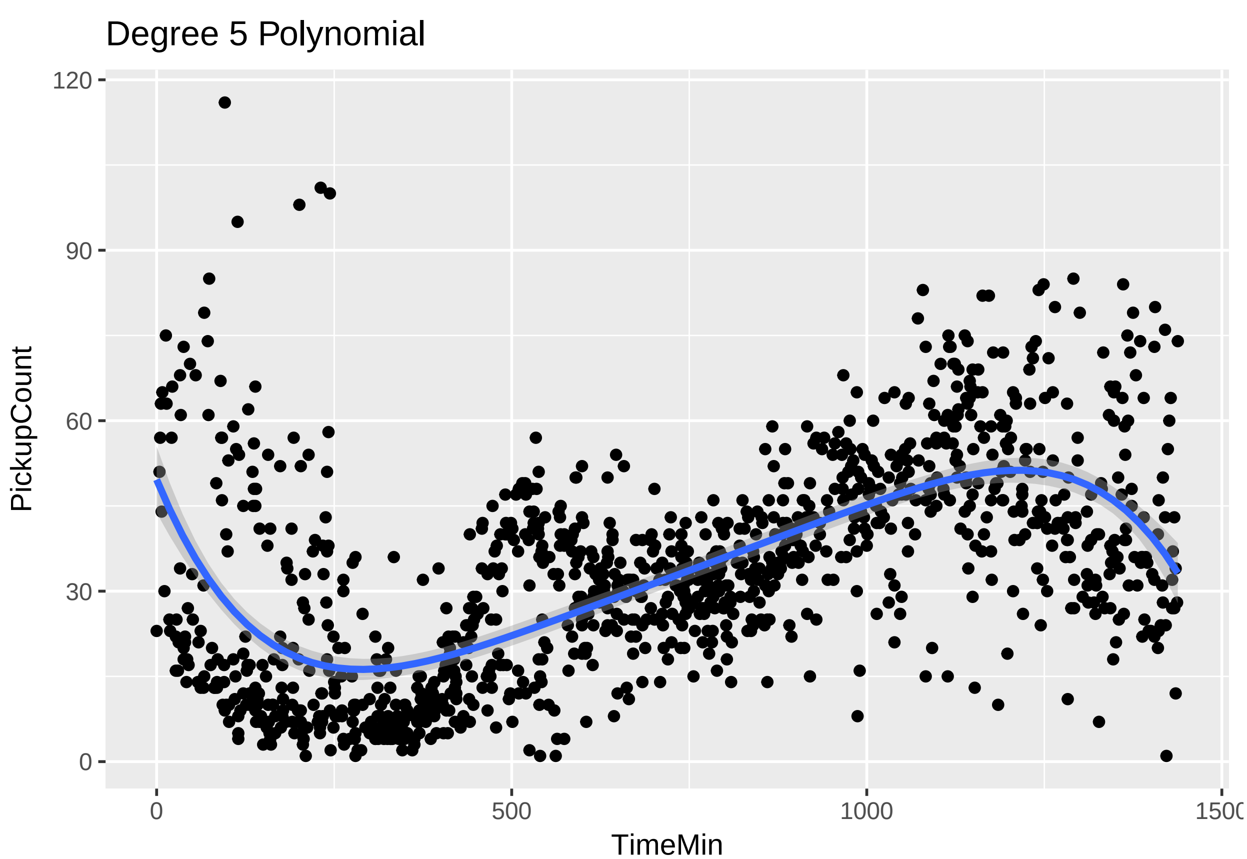
```
##
## Call:
## lm(formula = PickupCount ~ poly(TimeMin, 5), data = train1)
##
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -43.817  -9.380  -3.018   7.022  78.193
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)          33.5591     0.4391  76.428  < 2e-16 ***
## poly(TimeMin, 5)1  331.3054    14.7277  22.495  < 2e-16 ***
## poly(TimeMin, 5)2   90.3621    14.7277   6.136 1.18e-09 ***
## poly(TimeMin, 5)3 -231.0805    14.7277 -15.690  < 2e-16 ***
## poly(TimeMin, 5)4   28.2782    14.7277   1.920   0.0551 .
## poly(TimeMin, 5)5  -80.9429    14.7277  -5.496 4.81e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.73 on 1119 degrees of freedom
## Multiple R-squared:  0.424,  Adjusted R-squared:  0.4214
## F-statistic: 164.8 on 5 and 1119 DF,  p-value: < 2.2e-16
```

```r
#Test R-squared
rsq(poly5, test1, 'PickupCount')
```

```
## [1] 0.3855844
```

```r
p = ggplot(test1, aes(x = TimeMin, y = PickupCount)) + geom_point()
p + ggtitle('Degree 5 Polynomial') + stat_smooth(method = 'lm', formula = y ~ poly(x, 5), size = 1.25)
```




```r
#Degree 10 Polynomial
poly10 = lm(PickupCount ~ poly(TimeMin, 10) , data = train1)
```

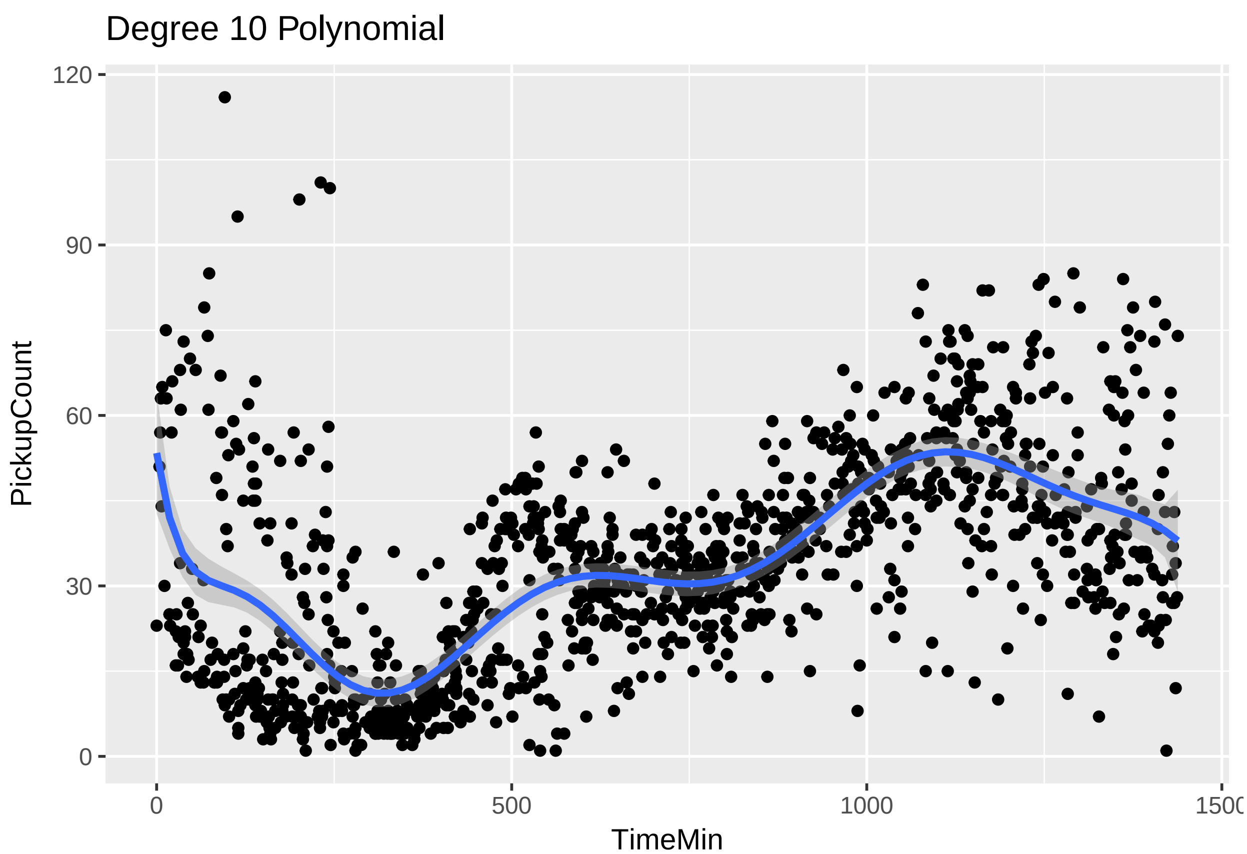
```r
summary(poly10)
```

```
##
## Call:
## lm(formula = PickupCount ~ poly(TimeMin, 10), data = train1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -43.097  -9.162  -2.292   6.894  77.302
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)           33.5591     0.4306  77.927  < 2e-16 ***
## poly(TimeMin, 10)1   331.3054    14.4444  22.937  < 2e-16 ***
## poly(TimeMin, 10)2    90.3621    14.4444   6.256 5.63e-10 ***
## poly(TimeMin, 10)3  -231.0805    14.4444 -15.998  < 2e-16 ***
## poly(TimeMin, 10)4    28.2782    14.4444   1.958 0.050511 .
## poly(TimeMin, 10)5   -80.9429    14.4444  -5.604 2.64e-08 ***
## poly(TimeMin, 10)6    16.4025    14.4444   1.136 0.256385
## poly(TimeMin, 10)7    85.5540    14.4444   5.923 4.21e-09 ***
## poly(TimeMin, 10)8   -15.4178    14.4444  -1.067 0.286027
## poly(TimeMin, 10)9   -48.6058    14.4444  -3.365 0.000791 ***
## poly(TimeMin, 10)10  -10.1605    14.4444  -0.703 0.481939
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.44 on 1114 degrees of freedom
## Multiple R-squared:  0.4484, Adjusted R-squared:  0.4435
## F-statistic: 90.57 on 10 and 1114 DF,  p-value: < 2.2e-16
```

```r
#Test R-squared
rsq(poly10, test1, 'PickupCount')
```

```
## [1] 0.4132089
```

```r
p + ggtitle('Degree 10 Polynomial') + stat_smooth(method = 'lm', formula = y ~ poly(x, 10), size = 1.25
```




```r
#Degree 25 Polynomial
poly25 = lm(PickupCount ~ poly(TimeMin, 25) , data = train1)
summary(poly25)
```

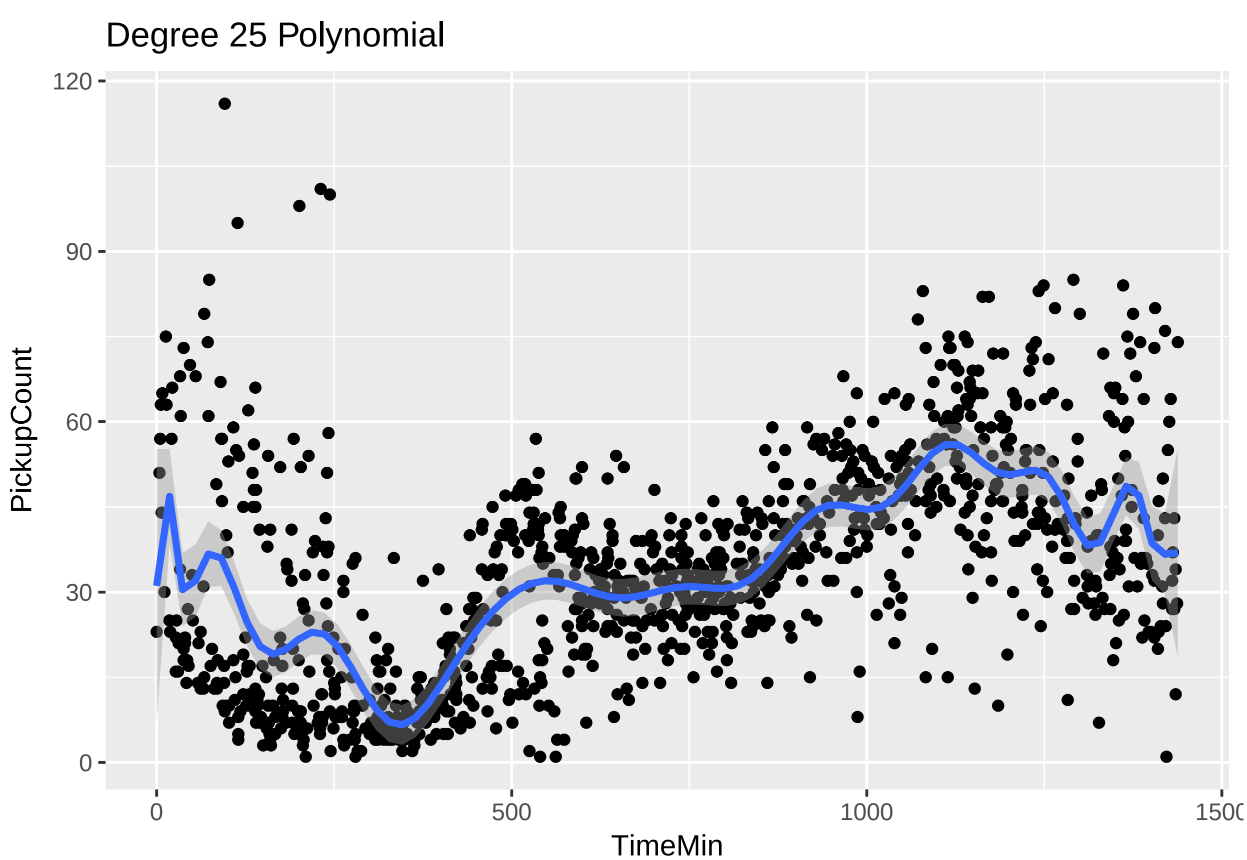
```
##
## Call:
## lm(formula = PickupCount ~ poly(TimeMin, 25), data = train1)
##
## Residuals:
##     Min       1Q   Median       3Q      Max
## -43.389  -9.210  -1.641   7.193  76.823
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)              33.5591     0.4274  78.520  < 2e-16 ***
## poly(TimeMin, 25)1      331.3054    14.3352  23.111  < 2e-16 ***
## poly(TimeMin, 25)2       90.3621    14.3352   6.304 4.21e-10 ***
## poly(TimeMin, 25)3     -231.0805    14.3352 -16.120  < 2e-16 ***
## poly(TimeMin, 25)4       28.2782    14.3352   1.973 0.048787 *
## poly(TimeMin, 25)5      -80.9429    14.3352  -5.646 2.08e-08 ***
## poly(TimeMin, 25)6       16.4025    14.3352   1.144 0.252787
## poly(TimeMin, 25)7       85.5540    14.3352   5.968 3.23e-09 ***
## poly(TimeMin, 25)8      -15.4178    14.3352  -1.076 0.282380
## poly(TimeMin, 25)9      -48.6058    14.3352  -3.391 0.000722 ***
## poly(TimeMin, 25)10     -10.1605    14.3352  -0.709 0.478611
## poly(TimeMin, 25)11      -0.3729    14.3352  -0.026 0.979252
```

```
## poly(TimeMin, 25)12  -55.5530    14.3352  -3.875 0.000113 ***
## poly(TimeMin, 25)13   14.5948    14.3352   1.018 0.308850
## poly(TimeMin, 25)14   21.0185    14.3352   1.466 0.142877
## poly(TimeMin, 25)15  -23.4431    14.3352  -1.635 0.102262
## poly(TimeMin, 25)16    9.1809    14.3352   0.640 0.522017
## poly(TimeMin, 25)17  -13.1077    14.3352  -0.914 0.360725
## poly(TimeMin, 25)18  -20.9970    14.3352  -1.465 0.143285
## poly(TimeMin, 25)19    9.2435    14.3352   0.645 0.519184
## poly(TimeMin, 25)20    9.6623    14.3352   0.674 0.500437
## poly(TimeMin, 25)21   22.6426    14.3352   1.580 0.114508
## poly(TimeMin, 25)22  -15.6077    14.3352  -1.089 0.276496
## poly(TimeMin, 25)23  -17.7619    14.3352  -1.239 0.215597
## poly(TimeMin, 25)24  -18.5191    14.3352  -1.292 0.196676
## poly(TimeMin, 25)25   -0.8832    14.3352  -0.062 0.950884
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.34 on 1099 degrees of freedom
## Multiple R-squared:  0.4641, Adjusted R-squared:  0.4519
## F-statistic: 38.06 on 25 and 1099 DF,  p-value: < 2.2e-16
```

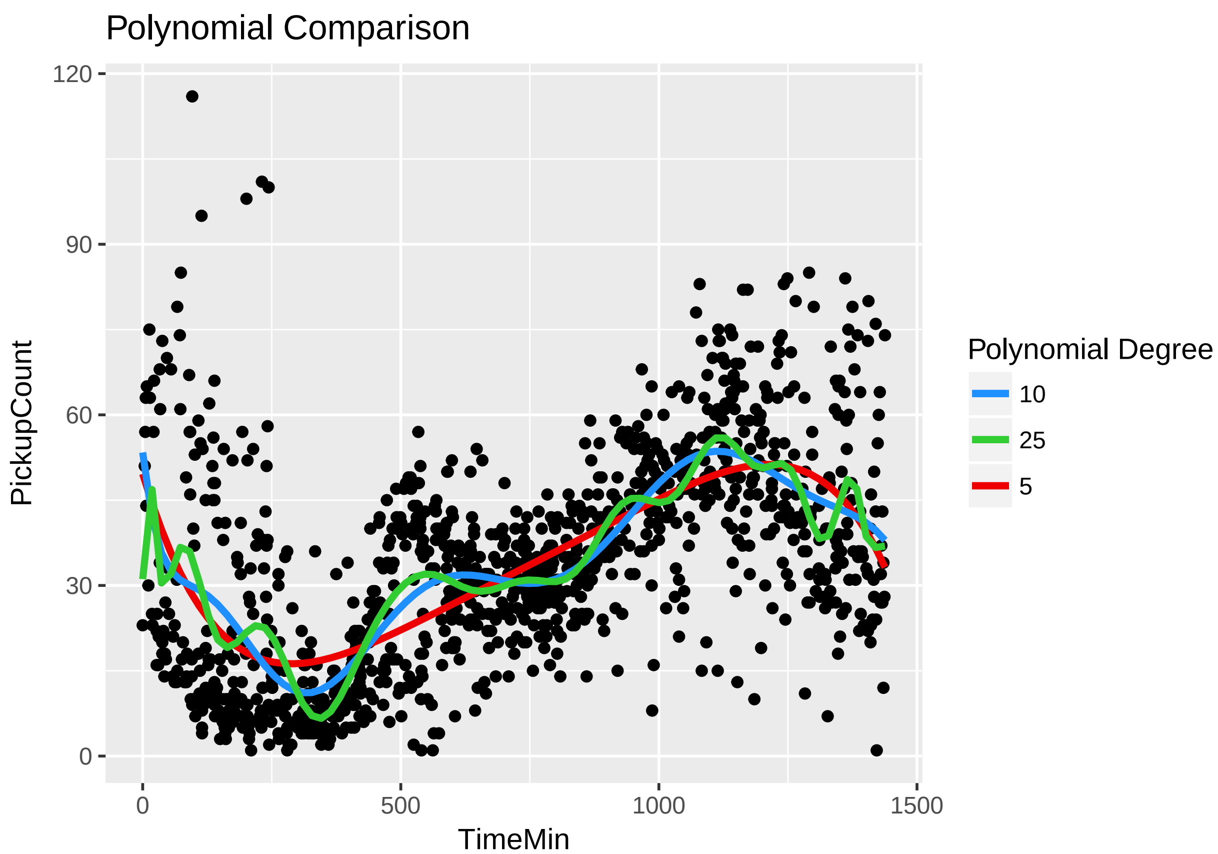
```r
#Test R-squared
rsq(poly25, test1, 'PickupCount')
```

```
## [1] 0.4266038
```

```r
p + ggtitle('Degree 25 Polynomial') + stat_smooth(method = 'lm', formula = y ~ poly(x, 25), size = 1.25)
```



```
#Polynomial Comparison
ggplot(test1, aes(x = TimeMin, y = PickupCount)) + geom_point(size = 1.5) +
          stat_smooth(method = 'lm', formula = y ~ poly(x, 5), se = F, aes(color = '5'), size = 1.25
          stat_smooth(method = 'lm', formula = y ~ poly(x, 10), se = F, aes(color = '10'), size = 1
          stat_smooth(method = 'lm', formula = y ~ poly(x, 25), se = F, aes(color = '25'), size = 1
          scale_colour_manual(name = 'Polynomial Degree', values = c('dodgerblue', 'limegreen', 're
          ggtitle('Polynomial Comparison')
```



After plotting the polynomial fits on the test data, we can see that the degree 5 polynomial does an ok job with $R^2_{Test} = .386$ and a fit that roughly captures the overall trend, but it's not great.

The degree 10 polynomial does a better job with $R^2_{Test} = .413$ and a better fit, but there is definitely still room for improvement.

The degree 25 polynomial does the best with $R^2_{Test} = .427$, but it is clear that it is overfitting the data since the fit is capturing some idiosyncrasies in the data that are not representative of the general trend.

```
# Cubic B-splines with knots chosen by inspection
library(splines)
b.spline = lm(PickupCount ~ bs(TimeMin, knots = c(300, 500, 750, 1100)) , data = train1)
summary(b.spline)
```

```
##
## Call:
## lm(formula = PickupCount ~ bs(TimeMin, knots = c(300, 500, 750,
##     1100)), data = train1)
##
## Residuals:
```

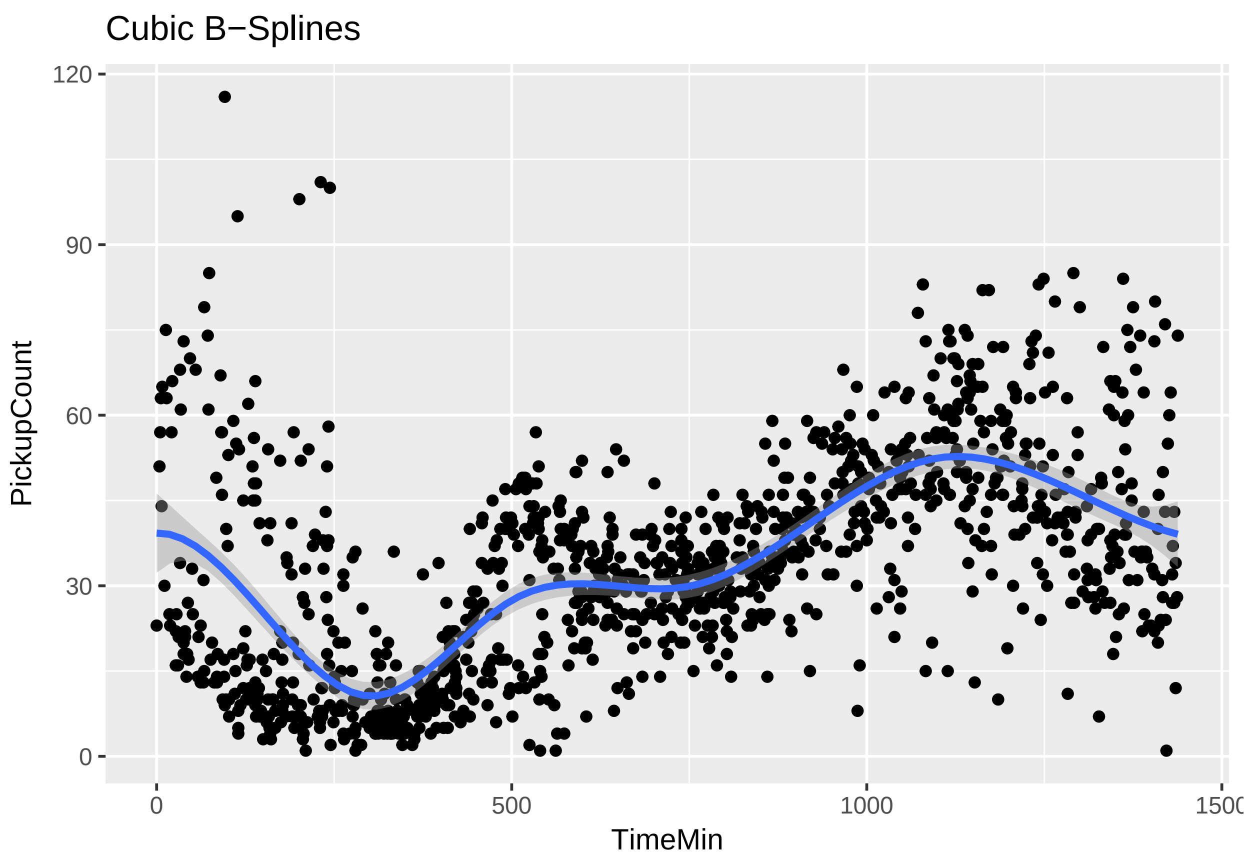
```
##     Min      1Q  Median      3Q     Max
## -42.066  -8.742  -2.143   6.839  77.321
##
## Coefficients:
##                                              Estimate Std. Error t value
## (Intercept)                                    42.426      2.876  14.753
## bs(TimeMin, knots = c(300, 500, 750, 1100))1  -10.269      5.883  -1.745
## bs(TimeMin, knots = c(300, 500, 750, 1100))2  -49.295      3.585 -13.752
## bs(TimeMin, knots = c(300, 500, 750, 1100))3   -5.891      3.963  -1.487
## bs(TimeMin, knots = c(300, 500, 750, 1100))4  -21.830      3.626  -6.020
## bs(TimeMin, knots = c(300, 500, 750, 1100))5   22.123      4.447   4.975
## bs(TimeMin, knots = c(300, 500, 750, 1100))6    4.939      4.212   1.172
## bs(TimeMin, knots = c(300, 500, 750, 1100))7   -4.083      4.148  -0.984
##                                              Pr(>|t|)
## (Intercept)                                   < 2e-16 ***
## bs(TimeMin, knots = c(300, 500, 750, 1100))1   0.0812 .
## bs(TimeMin, knots = c(300, 500, 750, 1100))2  < 2e-16 ***
## bs(TimeMin, knots = c(300, 500, 750, 1100))3   0.1374
## bs(TimeMin, knots = c(300, 500, 750, 1100))4 2.36e-09 ***
## bs(TimeMin, knots = c(300, 500, 750, 1100))5 7.54e-07 ***
## bs(TimeMin, knots = c(300, 500, 750, 1100))6   0.2413
## bs(TimeMin, knots = c(300, 500, 750, 1100))7   0.3251
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.46 on 1117 degrees of freedom
## Multiple R-squared:  0.446,  Adjusted R-squared:  0.4425
## F-statistic: 128.5 on 7 and 1117 DF,  p-value: < 2.2e-16
```

```r
#Test R-squared
rsq(b.spline, test1, 'PickupCount')
```

```
## Warning in bs(TimeMin, degree = 3L, knots = c(300, 500, 750, 1100),
## Boundary.knots = c(1L, : some 'x' values beyond boundary knots may cause
## ill-conditioned bases
```

```
## [1] 0.4143013
```

```r
p + ggtitle('Cubic B-Splines') +
    stat_smooth(method = 'lm', formula = y ~ bs(x, knots = c(300, 500, 750, 1100)), size = 1.25)
```

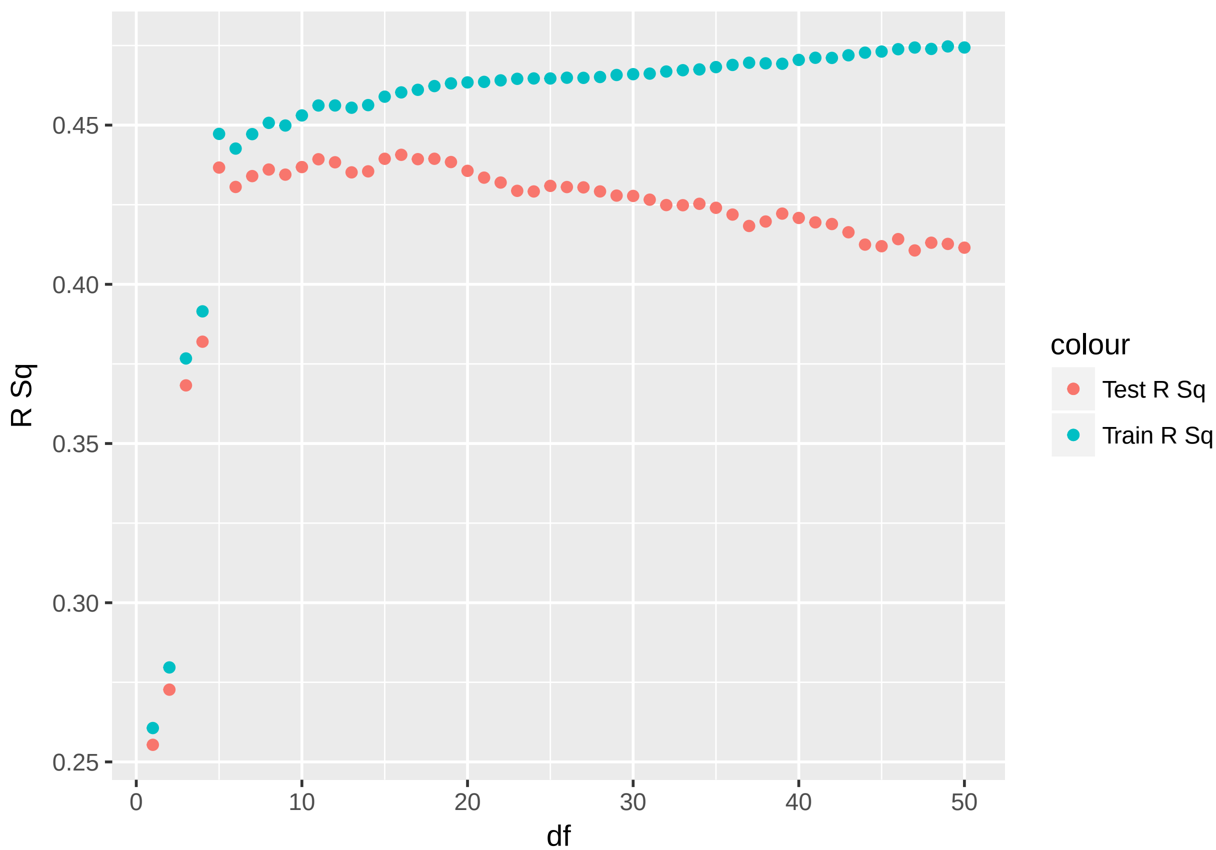


The Cubic B-splines does very similarly as the degree 10 polynomial with $R^2_{Test} = .414$.

We will now try using a natural spline, but we will also use 5-fold cross validation in order to choose the optimal smoothness parameter.

```
#k-fold cross validation adapted from the lab code
#However, there is definitely an easier way to do this as I demonstrate in 2.c.
set.seed(10)
k = 5

train1$partition = cut(sample(1:nrow(train1), nrow(train1)), k)

dfs = 1:50

model.performance = function(df, train, test) {

    mod = lm(PickupCount ~ ns(TimeMin, df = df), data = train)
    c(train.r2 = rsq(mod, train, 'PickupCount'), test.r2 = rsq(mod, test, 'PickupCount'))

}

performance = vector(mode = 'list', length = k)
names(performance) = unique(train1$partition)

for(partition in names(performance)) {

    test = (train1$partition == partition)
    performance[[partition]] = sapply(dfs, model.performance, train = train1[!test, ],
```

```
                                                         test = train1[test, ], simplify = F)
}

performance = sapply(performance, function(x) {
    x = as.data.frame(do.call(rbind, x))
    x$df = dfs; x}, simplify = F)

for(partition in names(performance)) {
    performance[[partition]] = data.frame(performance[[partition]],
                                          partition = partition)
}

performance = do.call(rbind, performance)

test.performance = aggregate(performance$test.r2, by = list(performance$df), mean)
train.performance = aggregate(performance$train.r2, by = list(performance$df), mean)

avg.performance = cbind(test.performance, train.performance[, 2])
names(avg.performance) = c('df', 'Test.Rsq',  'Train.Rsq')

ggplot(avg.performance, aes(x = df)) + geom_point(aes(y = Test.Rsq, color = 'Test R Sq')) +
                                       geom_point(aes(y = Train.Rsq, color = 'Train R Sq')) +
                                       ylab('R Sq')
```



```r
which.max(avg.performance$Test.Rsq)
```

```
## [1] 16
```

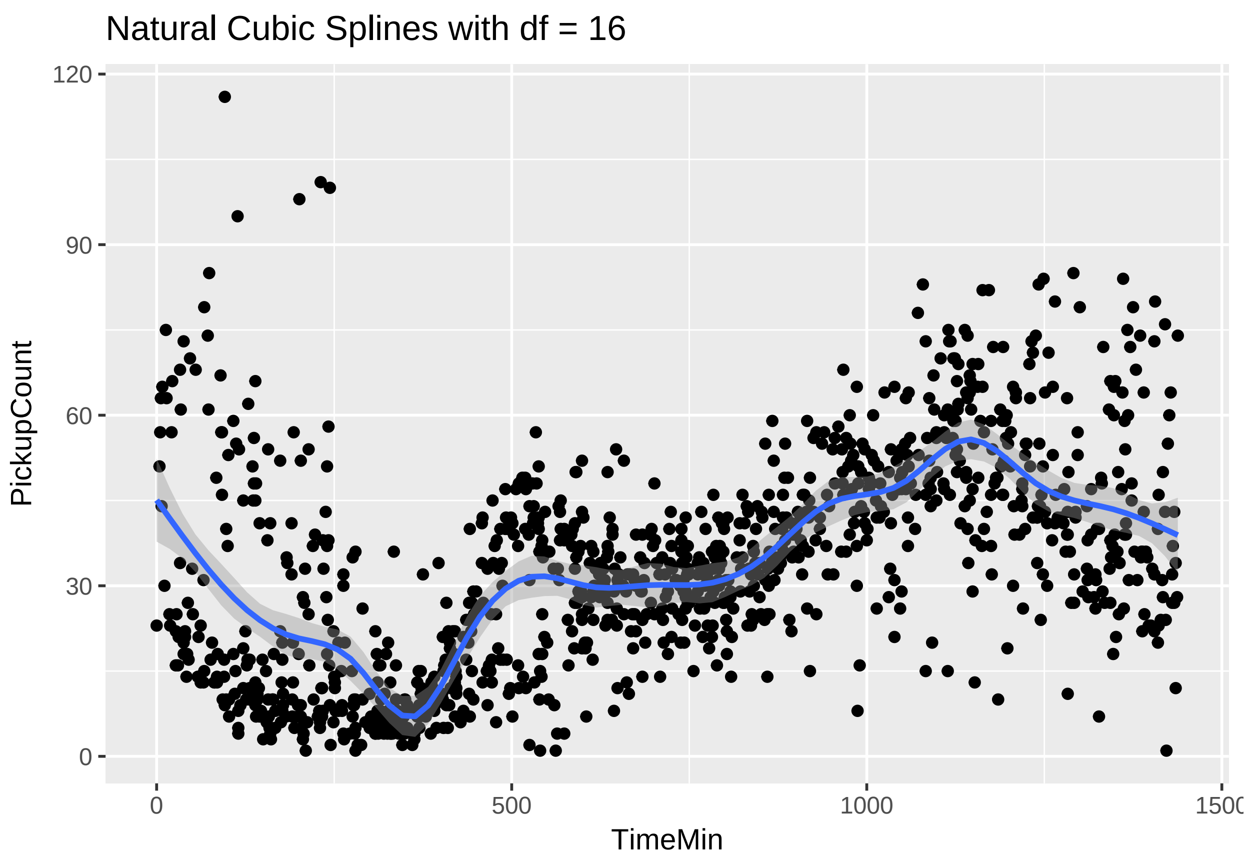
```r
n.spline = lm(PickupCount ~ ns(TimeMin, df = 16), data = train1)
summary(n.spline)
```

```
##
## Call:
## lm(formula = PickupCount ~ ns(TimeMin, df = 16), data = train1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -42.831  -8.997  -1.516   7.329  78.003
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)            44.332726   3.202183  13.845  < 2e-16 ***
## ns(TimeMin, df = 16)1 -30.721730   4.162954  -7.380 3.10e-13 ***
## ns(TimeMin, df = 16)2 -25.672252   5.437155  -4.722 2.64e-06 ***
## ns(TimeMin, df = 16)3 -41.921113   4.985091  -8.409  < 2e-16 ***
## ns(TimeMin, df = 16)4 -21.478699   5.095109  -4.216 2.69e-05 ***
## ns(TimeMin, df = 16)5 -12.600201   4.704577  -2.678 0.007509 **
## ns(TimeMin, df = 16)6 -14.330880   5.009961  -2.860 0.004309 **
## ns(TimeMin, df = 16)7 -16.145697   4.875823  -3.311 0.000958 ***
## ns(TimeMin, df = 16)8 -11.928955   4.957037  -2.406 0.016270 *
## ns(TimeMin, df = 16)9 -11.823912   5.080846  -2.327 0.020137 *
## ns(TimeMin, df = 16)10   4.146611   5.015295   0.827 0.408532
## ns(TimeMin, df = 16)11  -1.690329   4.846602  -0.349 0.727331
## ns(TimeMin, df = 16)12  22.437386   4.995996   4.491 7.83e-06 ***
## ns(TimeMin, df = 16)13  -0.009406   4.864304  -0.002 0.998458
## ns(TimeMin, df = 16)14   7.911671   4.226601   1.872 0.061487 .
## ns(TimeMin, df = 16)15 -14.467900   8.323275  -1.738 0.082445 .
## ns(TimeMin, df = 16)16   1.923007   3.780941   0.509 0.611130
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.35 on 1108 degrees of freedom
## Multiple R-squared:  0.4587, Adjusted R-squared:  0.4508
## F-statistic: 58.67 on 16 and 1108 DF,  p-value: < 2.2e-16
```

```r
#Test R-squared
rsq(n.spline, test1, 'PickupCount')
```

```
## [1] 0.4261959
```

```r
p + ggtitle('Natural Cubic Splines with df = 16') + stat_smooth(method = 'lm', formula = y ~ ns(x, df =
```



Based on the fit, the cross-validated Natural spline with 16 degrees of freedom appears to do the best job so far. It also has the highest R squared with $R^2_{Test} = .426$.

```
library(gam)
```

```
## Warning: package 'gam' was built under R version 3.3.2
```

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

```
## Loaded gam 1.14
```

```
cv.smooth = smooth.spline(x = train1$TimeMin, y = train1$PickupCount)
cv.smooth
```

```
## Call:
## smooth.spline(x = train1$TimeMin, y = train1$PickupCount)
##
## Smoothing Parameter  spar= 0.7632497  lambda= 0.0005893951 (12 iterations)
## Equivalent Degrees of Freedom (Df): 14.13588
## Penalized Criterion: 157635
## GCV: 208.5989
```

```
s.spline = gam(PickupCount ~ s(TimeMin, df = cv.smooth$df), data = train1)
summary(s.spline)
```

```
##
## Call: gam(formula = PickupCount ~ s(TimeMin, df = cv.smooth$df), data = train1)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -42.055  -9.155  -1.910   7.075  76.999
```

```
##
## (Dispersion Parameter for gaussian family taken to be 205.8145)
##
##     Null Deviance: 421395.3 on 1124 degrees of freedom
## Residual Deviance: 228426 on 1109.864 degrees of freedom
## AIC: 9202.492
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                                   Df Sum Sq Mean Sq F value    Pr(>F)
## s(TimeMin, df = cv.smooth$df)    1.0 109763  109763  533.31 < 2.2e-16 ***
## Residuals                     1109.9 228426     206
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                               Npar Df Npar F     Pr(F)
## (Intercept)
## s(TimeMin, df = cv.smooth$df)    13.1 30.776 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

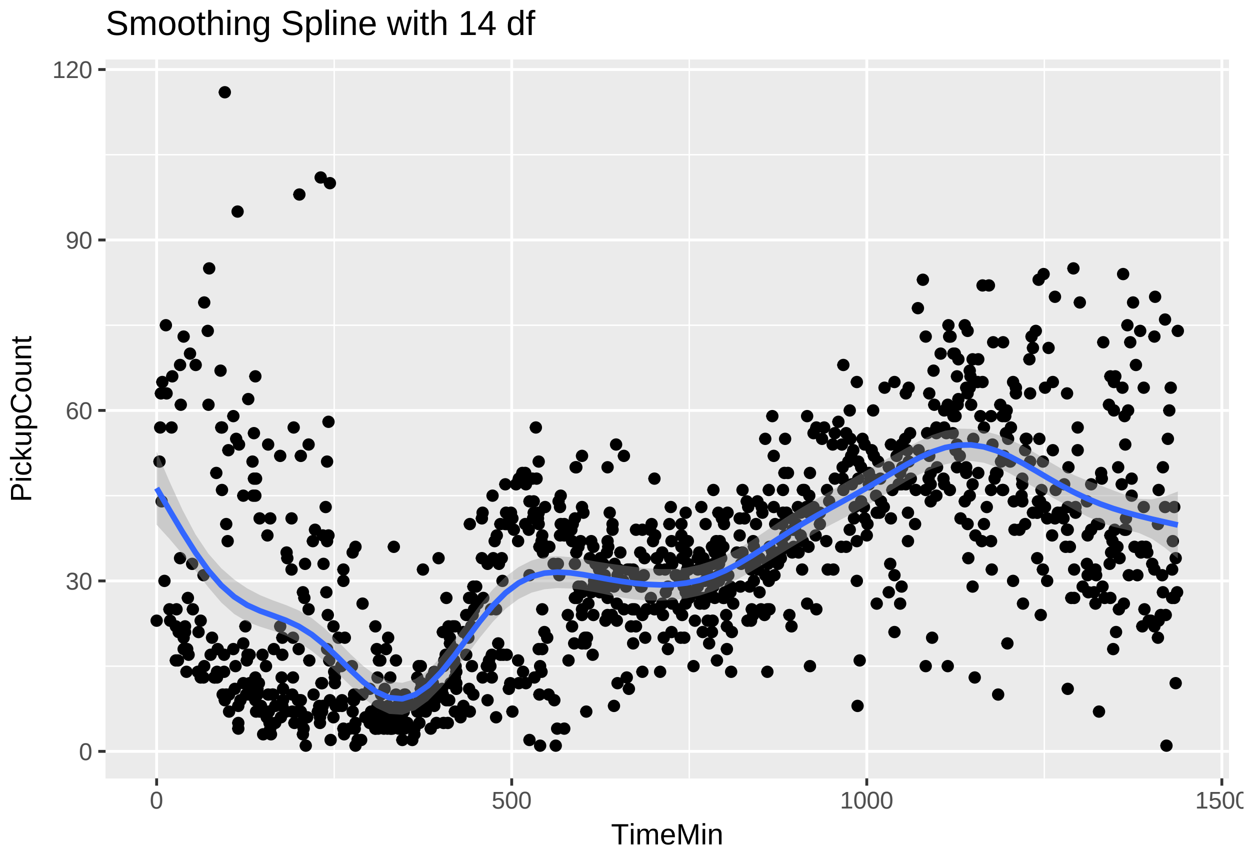
```r
#Test R-squared
rsq(s.spline, test1, 'PickupCount')
```

```
## [1] 0.4263872
```

```r
library(mgcv)
```

```
## Loading required package: nlme

## This is mgcv 1.8-12. For overview type 'help("mgcv-package")'.

##
## Attaching package: 'mgcv'

## The following objects are masked from 'package:gam':
##
##     gam, gam.control, gam.fit, plot.gam, predict.gam, s,
##     summary.gam
```

```r
p + ggtitle('Smoothing Spline with 14 df') + stat_smooth(method = 'gam', formula = y ~ s(x, k = 14))
```



The cross-validated smoothing spline with 14 degrees of freedom performs similarly as the natural spline on the validation set with the same R squared ($R^2_{Test} = .426$). However, the fit does appear to be smoother and thus potentially better.

```
# 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(model.loess, train[folds == j,], 'PickupCount')
    }

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

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

```
set.seed(1)
grid = seq(0.02, 1, by=.02)
cv.loess = crossval_loess(train1, grid, 5)
cv.loess
```

```
##  [1] 0.3399136 0.4088354 0.4203052 0.4234400 0.4271727 0.4297065 0.4330372
##  [8] 0.4363345 0.4386756 0.4398565 0.4406137 0.4408725 0.4408910 0.4410365
## [15] 0.4406677 0.4402733 0.4399648 0.4395259 0.4391186 0.4386192 0.4381921
## [22] 0.4377451 0.4372879 0.4367816 0.4362063 0.4353582 0.4343328 0.4323947
## [29] 0.4304942 0.4279387 0.4243309 0.4208286 0.4175825 0.4131999 0.4083903
## [36] 0.4041544 0.3992913 0.3947106 0.3899588 0.3850693 0.3809989 0.3768416
## [43] 0.3726297 0.3698061 0.3671994 0.3639306 0.3608340 0.3581928 0.3555649
## [50] 0.3533914
```

```
which.max(cv.loess)
```

```
## [1] 14
```

```
s.best = grid[which.max(cv.loess)]
s.best
```

```
## [1] 0.28
```

```
loess.fit = loess(PickupCount ~ TimeMin, span = s.best, data = train1)
loess.fit
```

```
## Call:
## loess(formula = PickupCount ~ TimeMin, data = train1, span = s.best)
##
## Number of Observations: 1125
## Equivalent Number of Parameters: 10.65
## Residual Standard Error: 14.37
```

```
rsq(loess.fit, test1, 'PickupCount')
```

```
## [1] NA
```

```
#For some reason, our R-squared function is returning NA
#Let's investigate
pred.y = predict(loess.fit, newdata = data.frame(TimeMin = test1$TimeMin))
```

```r
any(is.na(pred.y))
```

```
## [1] TRUE
```

```r
pred.y[is.na(pred.y)]
```

```
## 404
##  NA
```

```r
#It appears that our model returns NA for TimeMin = 0
test1$TimeMin[404]
```

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

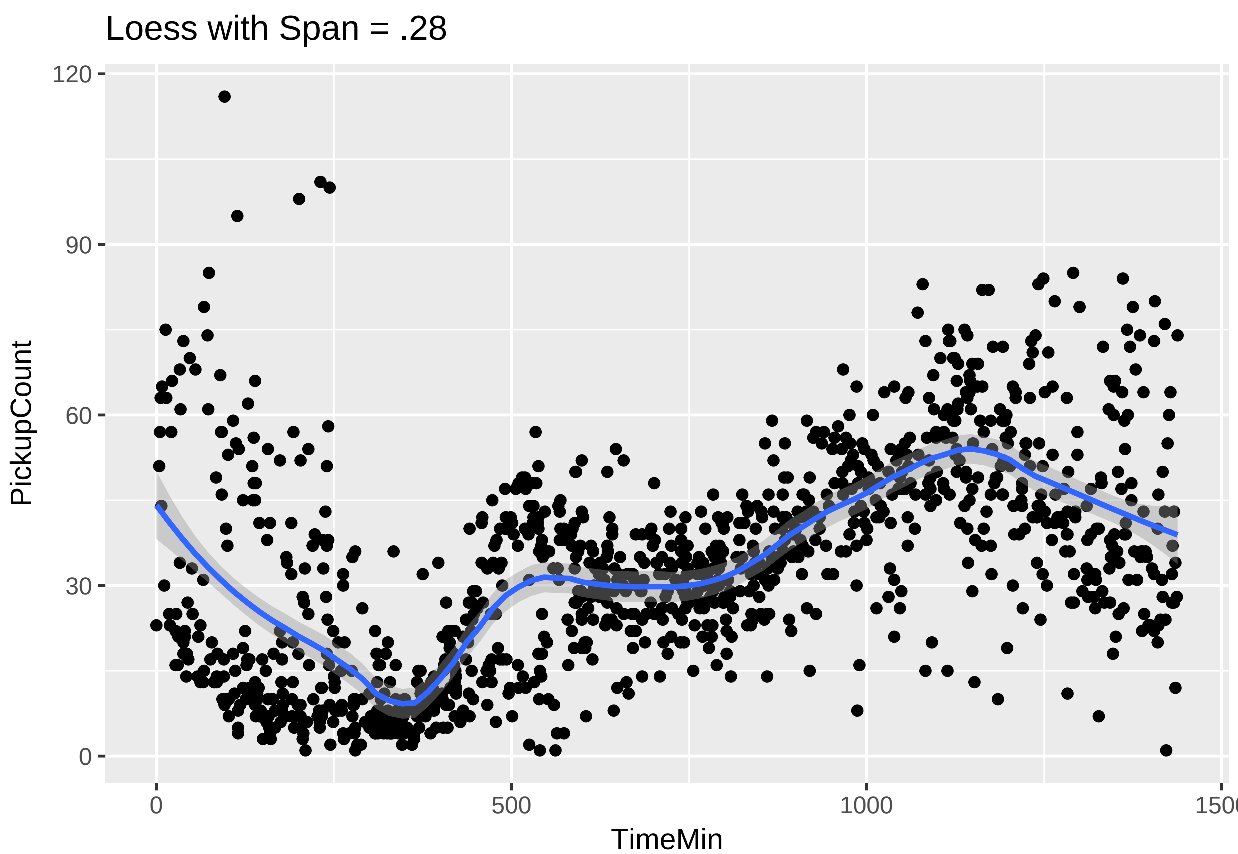
```r
length(test1$TimeMin[test1$TimeMin == 0])
```

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

```r
#Let's redefine our R squared function to ignore NAs
rsq = function(model, data, y) {
  y = data[[y]]
  predict = predict(model, newdata = data)
  tss = sum((y - mean(y))^2, na.rm = T)
  rss = sum((y-predict)^2, na.rm = T)
  rsq_ = max(0, 1 - rss/tss)
  return(rsq_)
}

rsq(loess.fit, test1, 'PickupCount')
```

```
## [1] 0.4255075
```

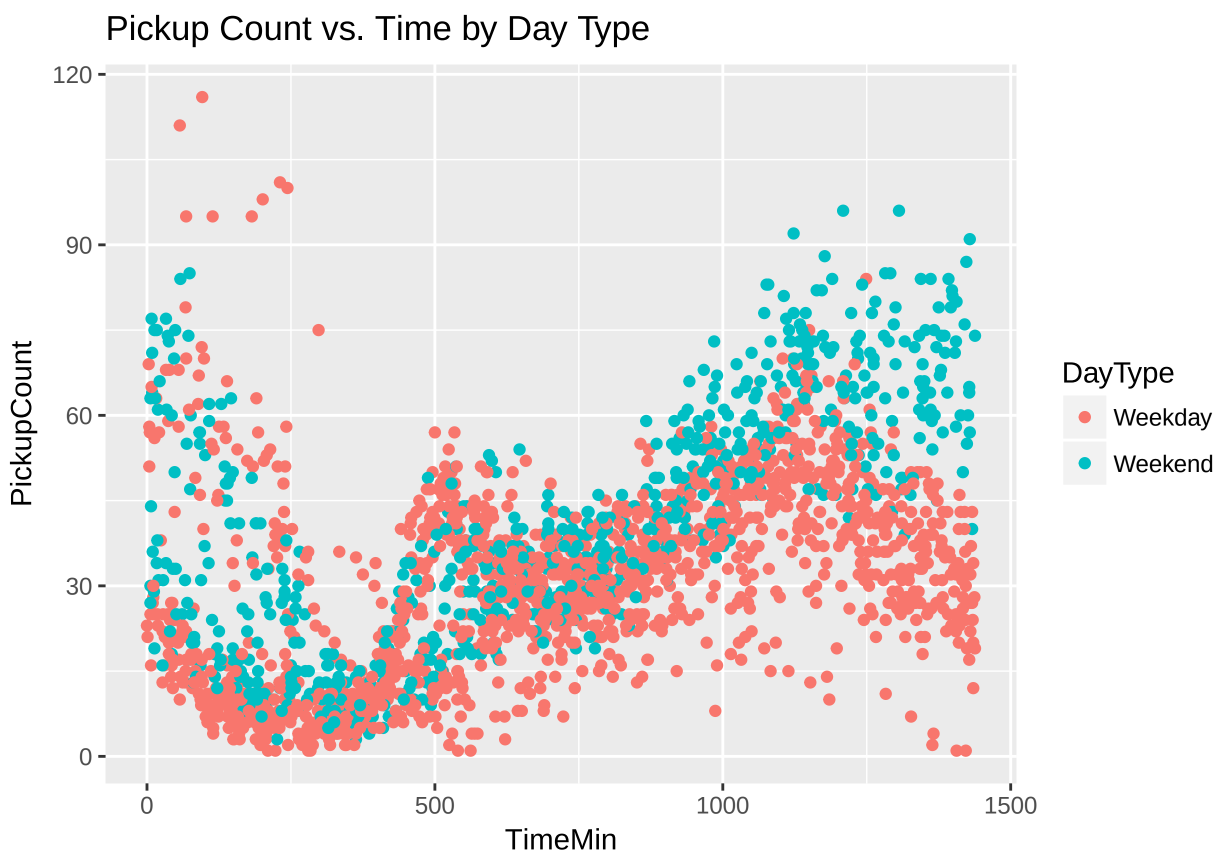
```r
p + ggtitle('Loess with Span = .28') + stat_smooth(method = 'loess', formula = y ~ x, span = s.best)
```



The cross-validated loess performs similarly as the natural spline and the smoothing spline on the validation set with an $R^2_{Test} = .424$. However, this fit does appears to be the smoothest of the three with no apparent overfitting.

1.b.

```
#Create day type indicator variable
data$DayType = ifelse(data$DayOfWeek >= 6, 'Weekend', 'Weekday')
data$DayType = as.factor(data$DayType)

ggplot(data, aes(x = TimeMin, y = PickupCount, color = DayType)) + geom_point() +
             ggtitle('Pickup Count vs. Time by Day Type')
```



It is clear that pickup behavior on weekdays is significantly different than pickup behavior on weekends. Consequently, we should build a seperate model for each day type.

```
train.weekday = train1[train1$DayOfWeek <= 5,]
train.weekend = train1[train1$DayOfWeek >= 6,]

test.weekday = test1[test1$DayOfWeek <= 5,]
test.weekend = test1[test1$DayOfWeek >= 6,]


#Weekday Model
set.seed(1)
cv.loess.weekday = crossval_loess(train.weekday, grid, 5)
cv.loess.weekday
```

```
##  [1] 0.1440499 0.2856528 0.3190133 0.3377309 0.3480177 0.3568786 0.3658076
##  [8] 0.3702141 0.3722902 0.3724946 0.3717452 0.3715243 0.3718855 0.3720634
## [15] 0.3718412 0.3713670 0.3711443 0.3711518 0.3706568 0.3698070 0.3688355
## [22] 0.3678488 0.3668643 0.3655380 0.3643841 0.3621737 0.3601304 0.3587497
## [29] 0.3562313 0.3527024 0.3492655 0.3453142 0.3410300 0.3362798 0.3315770
## [36] 0.3256184 0.3196642 0.3133472 0.3071695 0.3015242 0.2959568 0.2906996
## [43] 0.2869180 0.2834338 0.2804890 0.2770571 0.2736195 0.2707303 0.2679315
## [50] 0.2651003
```

```
which.max(cv.loess.weekday)
```

```
## [1] 10
```

```r
s.best.weekday = grid[which.max(cv.loess.weekday)]
s.best.weekday
```

```
## [1] 0.2
```

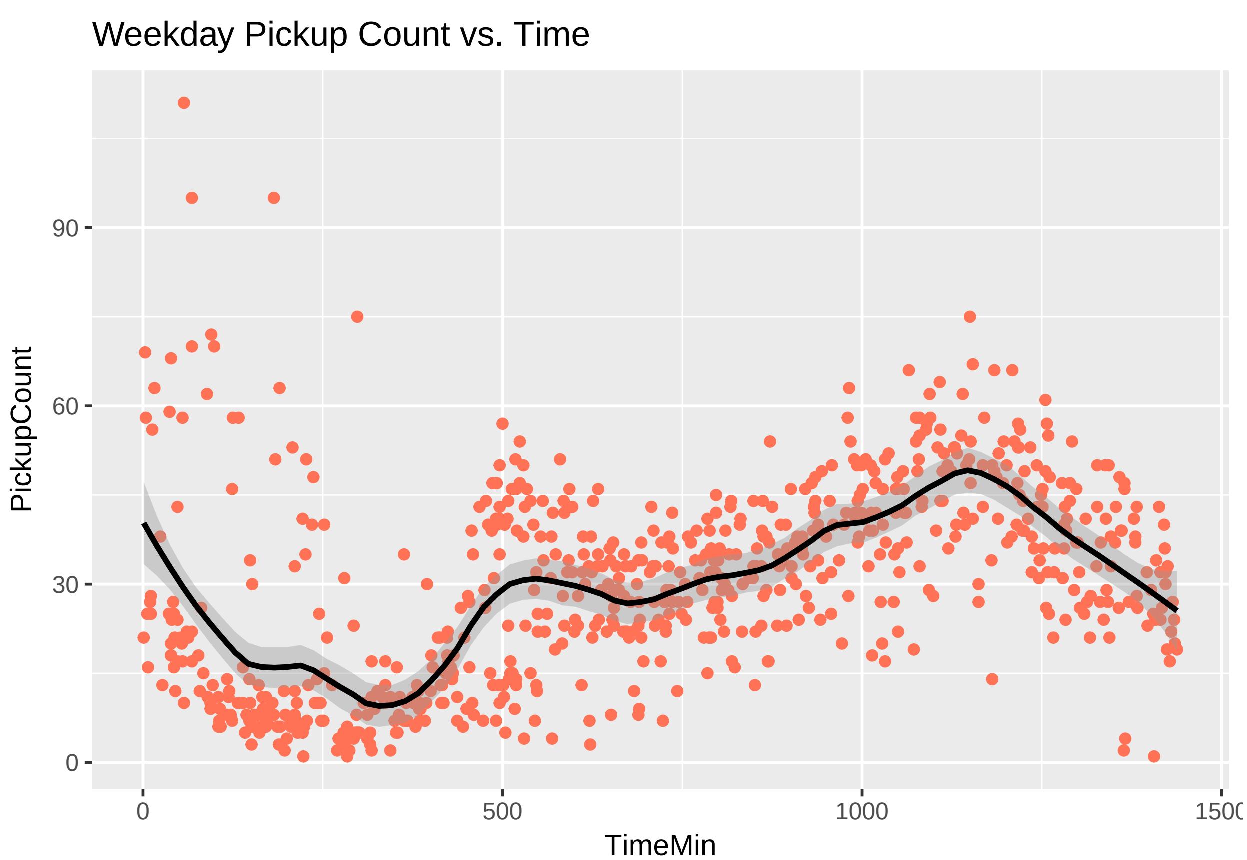
```r
loess.fit.weekday = loess(PickupCount ~ TimeMin, span = s.best, data = train.weekday)
loess.fit.weekday
```

```
## Call:
## loess(formula = PickupCount ~ TimeMin, data = train.weekday,
##     span = s.best)
##
## Number of Observations: 734
## Equivalent Number of Parameters: 10.66
## Residual Standard Error: 12.97
```

```r
rsq(loess.fit.weekday, test.weekday, 'PickupCount')
```

```
## [1] 0.3552042
```

```r
ggplot(train.weekday, aes(x = TimeMin, y = PickupCount)) + geom_point(color = 'coral1') +
       ggtitle('Weekday Pickup Count vs. Time') +
       stat_smooth(method = 'loess', formula = y ~ x, span = s.best.weekday, color = 'black')
```




```r
#Weekend Model
set.seed(1)
cv.loess.weekend = crossval_loess(train.weekend, grid, 5)
cv.loess.weekend
```

```
##  [1] 0.2933608 0.6151385 0.6613795 0.6827544 0.6957140 0.6970738 0.6996755
##  [8] 0.7032240 0.7050188 0.7062529 0.7079984 0.7083878 0.7082749 0.7085801
## [15] 0.7083362 0.7077127 0.7067926 0.7061173 0.7050708 0.7044304 0.7041239
## [22] 0.7039232 0.7034544 0.7032649 0.7032231 0.7029096 0.7025030 0.7021605
## [29] 0.7012298 0.6993859 0.6975165 0.6956908 0.6935743 0.6906366 0.6880606
## [36] 0.6853141 0.6819153 0.6794078 0.6764441 0.6722727 0.6701673 0.6677869
## [43] 0.6634477 0.6611507 0.6587239 0.6559366 0.6534746 0.6510605 0.6489640
## [50] 0.6468383
```

```r
which.max(cv.loess.weekend)
```

```
## [1] 14
```

```r
s.best.weekend = grid[which.max(cv.loess.weekend)]
s.best.weekend
```

```
## [1] 0.28
```

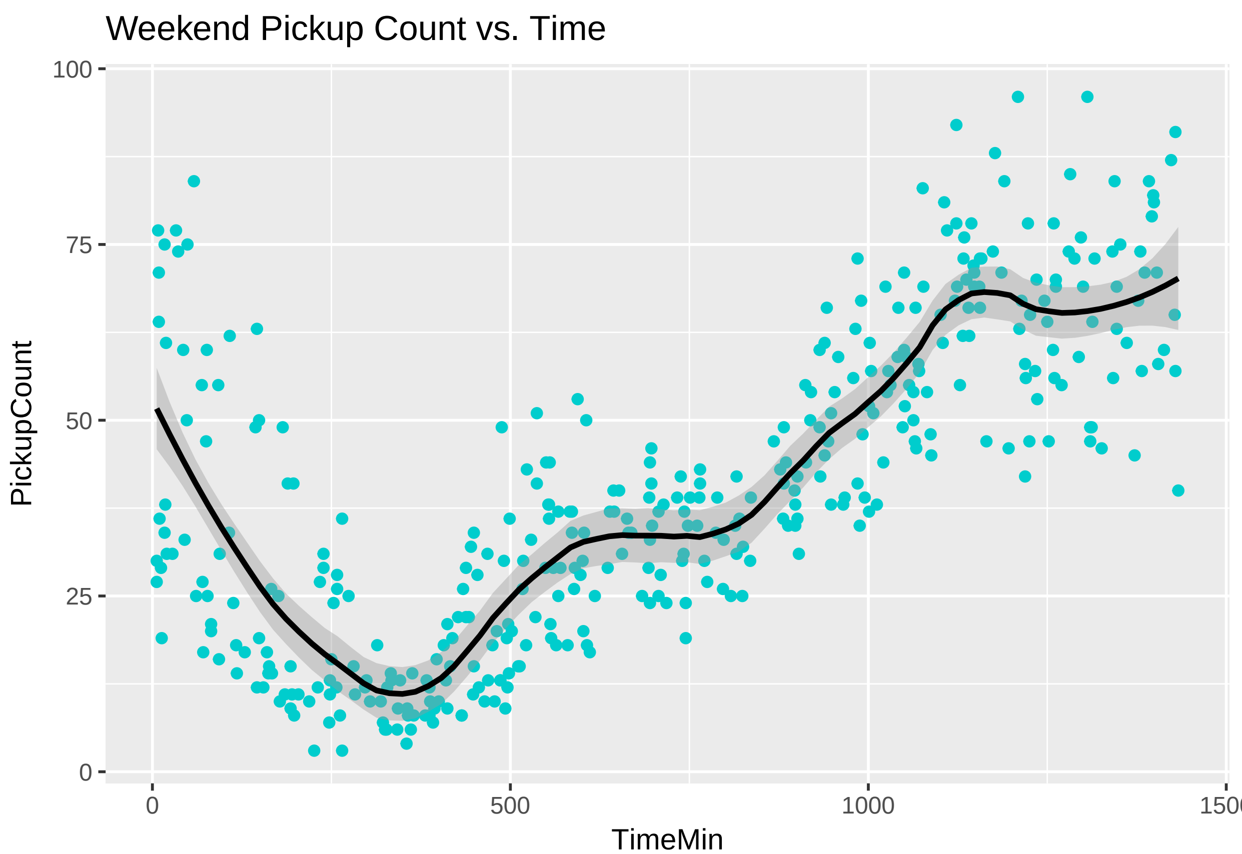
```r
loess.fit.weekend = loess(PickupCount ~ TimeMin, span = s.best.weekend, data = train.weekend)
loess.fit.weekend
```

```
## Call:
## loess(formula = PickupCount ~ TimeMin, data = train.weekend,
##     span = s.best.weekend)
##
## Number of Observations: 391
## Equivalent Number of Parameters: 10.8
## Residual Standard Error: 11.78
```

```r
rsq(loess.fit.weekend, test.weekend, 'PickupCount')
```

```
## [1] 0.7356815
```

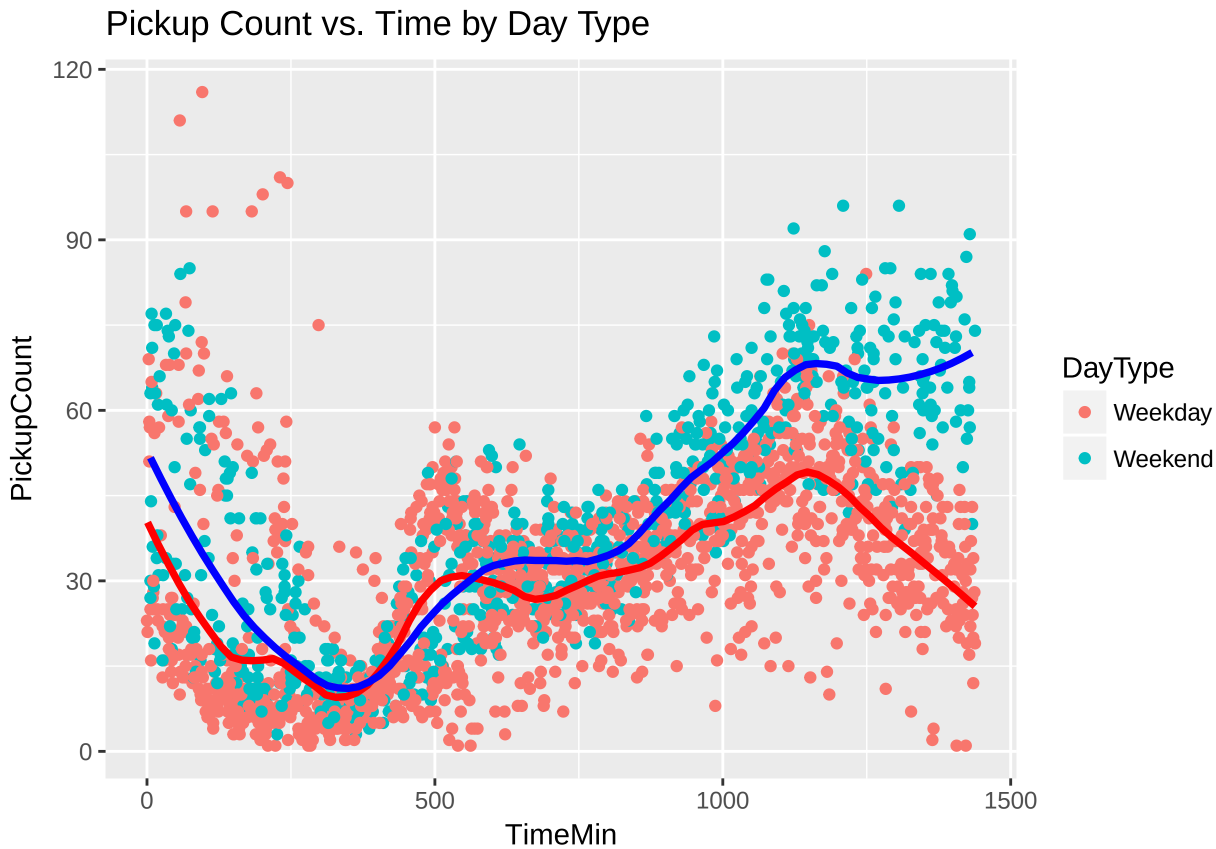
```r
ggplot(train.weekend, aes(x = TimeMin, y = PickupCount)) + geom_point(color = 'cyan3') +
       ggtitle('Weekend Pickup Count vs. Time') +
       stat_smooth(method = 'loess', formula = y ~ x, span = s.best.weekend, color = 'black')
```




```r
#Redefine R squared function so we can calculate combined R squared of both models
rsq2 = function(y, predict) {
tss = sum((y - mean(y))^2, na.rm = T)
rss = sum((y-predict)^2, na.rm = T)
r_squared = 1 - rss/tss
return(r_squared)
}

new1 = data.frame(TimeMin = test.weekday$TimeMin)
pred.y1 = predict(loess.fit.weekday, new1)

new2 = data.frame(TimeMin = test.weekend$TimeMin)
pred.y2 = predict(loess.fit.weekend, new2)

#Combined test R squared
rsq2( c(test.weekday$PickupCount, test.weekend$PickupCount), c(pred.y1, pred.y2) )
```

```
## [1] 0.5375969
```

```r
ggplot(data, aes(x = TimeMin, y = PickupCount, color = DayType)) + geom_point() +
           ggtitle('Pickup Count vs. Time by Day Type') +
           stat_smooth(data = train.weekday, method = 'loess', formula = y ~ x, span = s.best.weekday
                      color = 'red', size = 1.3, se = F) +
           stat_smooth(data = train.weekend, method = 'loess', formula = y ~ x, span = s.best.weekend
                      color = 'blue', size = 1.3, se = F)
```



After seperating weekday from weekend data and creating a seperate loess model for each, we obtained a combined $R^2_{Test} = .537$, which is significantly higher than our previous loess model ($R^2_{Test} = .424$). We can also see the improvement by simply looking at the scatterplot by day type, and noticing the considerable difference between both the points and the fits of weekday vs. weekend.

Stat 121B - Homework 1

Keyan Halperin

February 1, 2017

2.

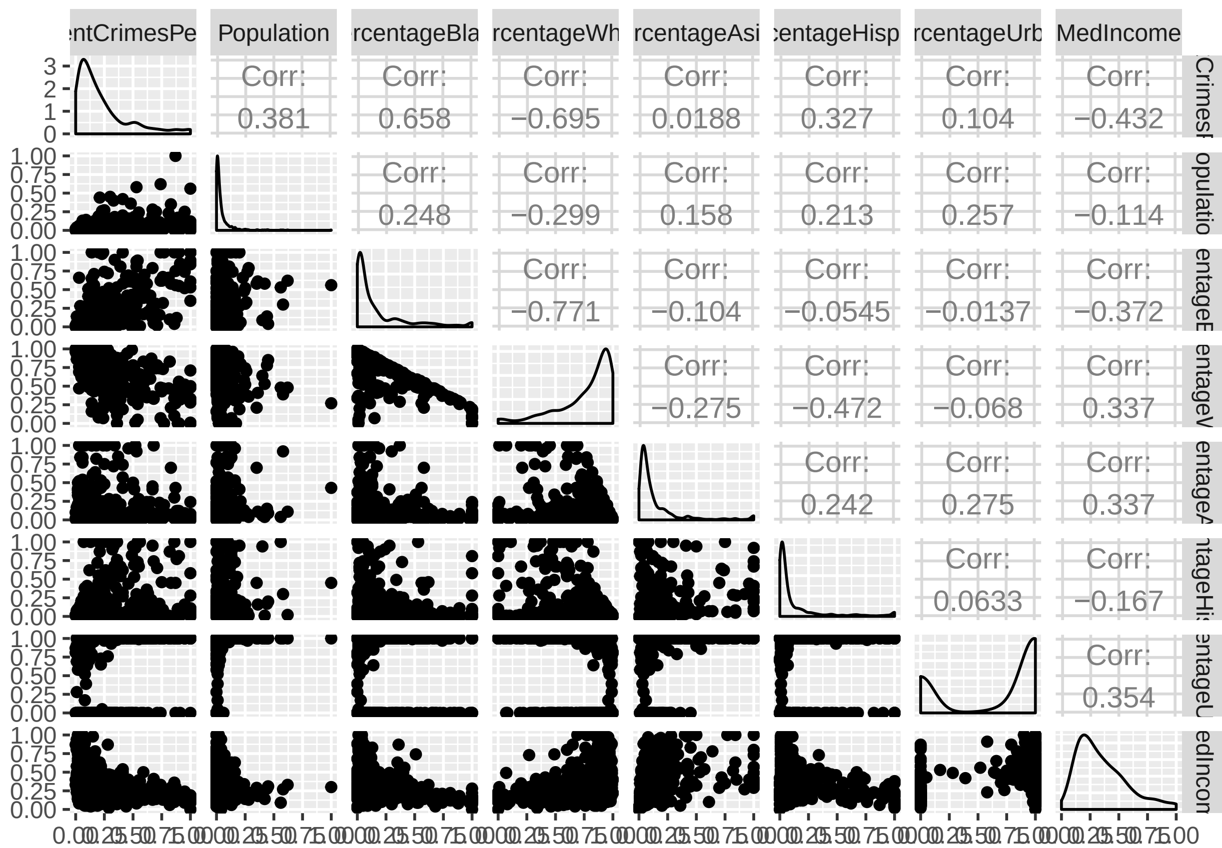
```
setwd("C:/Users/Keyan/Google Drive/School/Harvard/Stat 121B")
train2 = read.table('dataset_2_train.txt', header = T)
test2 = read.table('dataset_2_test.txt', header = T)
names(train2)
```

```
## [1] "ViolentCrimesPerPop" "Population"          "PercentageBlack"
## [4] "PercentageWhite"     "PercentageAsian"     "PercentageHispanic"
## [7] "PercentageUrban"     "MedIncome"
```

```
library(GGally)
```

```
## Warning: package 'GGally' was built under R version 3.3.2
```

```
ggpairs(train2)
```



By looking at the pairwise scatterplot, it is clear that there are several predictors (e.g. Median Income) that have a non-linear relationship with the response variable. Consequently, we should either perform some

transformations, or use a non-linear regression model.
#2.a.

```r
#Linear model on all predictors
linear.fit = lm(ViolentCrimesPerPop ~ ., data = train2)
summary(linear.fit)
```

```
##
## Call:
## lm(formula = ViolentCrimesPerPop ~ ., data = train2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.52696 -0.07106 -0.01740  0.05287  0.64075
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)          0.10125    0.09922   1.020 0.308015
## Population           0.30944    0.08413   3.678 0.000261 ***
## PercentageBlack      0.55365    0.08381   6.606 1.03e-10 ***
## PercentageWhite      0.01297    0.09927   0.131 0.896095
## PercentageAsian      0.03698    0.04922   0.751 0.452846
## PercentageHispanic   0.28411    0.05404   5.258 2.18e-07 ***
## PercentageUrban      0.06779    0.01624   4.174 3.54e-05 ***
## MedIncome           -0.23109    0.03829  -6.035 3.14e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1439 on 490 degrees of freedom
## Multiple R-squared:  0.6185, Adjusted R-squared:  0.613
## F-statistic: 113.5 on 7 and 490 DF,  p-value: < 2.2e-16
```

```r
rsq = function(model, data, y) {
  y = data[[y]]
  predict = predict(model, newdata = data)
  tss = sum((y - mean(y))^2)
  rss = sum((y-predict)^2)
  rsq_ = max(0, 1 - rss/tss)
  return(rsq_)
}

#Test R-squared
rsq(linear.fit, test2, 'ViolentCrimesPerPop')
```

```
## [1] 0.5553841
```

```r
polyfit2 = lm(ViolentCrimesPerPop ~ poly(Population, 2) + poly(PercentageBlack, 2) + poly(PercentageWhi
                                    + poly(PercentageAsian, 2) + poly(PercentageHispanic, 2) +
                                      poly(PercentageUrban, 2) + poly(MedIncome, 2), data = train2)
summary(polyfit2)
```

```
##
## Call:
## lm(formula = ViolentCrimesPerPop ~ poly(Population, 2) + poly(PercentageBlack,
##     2) + poly(PercentageWhite, 2) + poly(PercentageAsian, 2) +
##     poly(PercentageHispanic, 2) + poly(PercentageUrban, 2) +
```

```
##     poly(MedIncome, 2), data = train2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.42630 -0.07023 -0.01336  0.05123  0.62143
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  0.226747   0.006372  35.587  < 2e-16 ***
## poly(Population, 2)1         0.494333   0.165666   2.984 0.002990 **
## poly(Population, 2)2        -0.094483   0.149825  -0.631 0.528584
## poly(PercentageBlack, 2)1    3.234671   0.492531   6.567 1.33e-10 ***
## poly(PercentageBlack, 2)2   -0.822564   0.234082  -3.514 0.000483 ***
## poly(PercentageWhite, 2)1    0.462658   0.572827   0.808 0.419676
## poly(PercentageWhite, 2)2    0.822857   0.258722   3.180 0.001565 **
## poly(PercentageAsian, 2)1    0.179585   0.243992   0.736 0.462073
## poly(PercentageAsian, 2)2   -0.044009   0.157178  -0.280 0.779601
## poly(PercentageHispanic, 2)1  1.776410   0.309737   5.735 1.72e-08 ***
## poly(PercentageHispanic, 2)2 -0.440379   0.164701  -2.674 0.007754 **
## poly(PercentageUrban, 2)1    0.721804   0.174145   4.145 4.02e-05 ***
## poly(PercentageUrban, 2)2   -0.051331   0.155159  -0.331 0.740919
## poly(MedIncome, 2)1         -1.200304   0.201365  -5.961 4.85e-09 ***
## poly(MedIncome, 2)2          0.399239   0.163523   2.441 0.014985 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1422 on 483 degrees of freedom
## Multiple R-squared:  0.6328, Adjusted R-squared:  0.6221
## F-statistic: 59.45 on 14 and 483 DF,  p-value: < 2.2e-16
```

```r
#Test R-squared
rsq(polyfit2, test2, 'ViolentCrimesPerPop')
```

```
## [1] 0.5753024
```

```r
polyfit3 = lm(ViolentCrimesPerPop ~ poly(Population, 3) + poly(PercentageBlack, 3) + poly(PercentageWhi
                                    + poly(PercentageAsian, 3) + poly(PercentageHispanic, 3) +
                                      poly(PercentageUrban, 3) + poly(MedIncome, 3), data = train2)

#Test R-squared
rsq(polyfit3, test2, 'ViolentCrimesPerPop')
```

```
## [1] 0.5734467
```

```r
library(splines)
b.spline = lm(ViolentCrimesPerPop ~ bs(Population, df = 3) + bs(PercentageBlack, df = 3) +
                                    bs(PercentageWhite, df = 3) + bs(PercentageAsian, df = 3) +
                                    bs(MedIncome, df = 3) , data = train2)

#Test R-squared
rsq(b.spline, test2, 'ViolentCrimesPerPop')
```

```
## [1] 0.5734467
```

Both polynomial fits and the B-spline fit perform about the same on the test data with an $R^2_{Test} = .57$, but they do only slightly better than the linear model which has an $R^2_{Test} = .56$.

2.b.

```r
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

```r
library(gam)
```

```
## Warning: package 'gam' was built under R version 3.3.2
```

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

```
## Loaded gam 1.14
```

```r
control = trainControl(method = "cv", number = 5)
grid = expand.grid(df = 1:25)

fit = train(ViolentCrimesPerPop ~ ., data = train2, method = "gamSpline",
                                     tuneGrid = grid, trControl = control, metric = "Rsquared")
fit
```

```
## Generalized Additive Model using Splines
##
## 498 samples
##   7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 399, 398, 397, 400, 398
## Resampling results across tuning parameters:
##
##   df  RMSE       Rsquared
##    1  0.1446990  0.6140879
##    2  0.1444811  0.6148378
##    3  0.1449544  0.6134721
##    4  0.1466213  0.6075290
##    5  0.1479422  0.6029873
##    6  0.1482600  0.6013419
##    7  0.1481195  0.6006272
##    8  0.1482267  0.5987618
##    9  0.1488474  0.5950440
##   10  0.1499509  0.5896527
##   11  0.1514387  0.5829598
##   12  0.1532509  0.5752061
##   13  0.1553576  0.5665602
##   14  0.1577965  0.5570071
##   15  0.1606714  0.5463703
##   16  0.1642208  0.5342014
##   17  0.1687891  0.5200098
##   18  0.1747866  0.5035845
##   19  0.1825815  0.4853027
##   20  0.1923765  0.4661741
##   21  0.2042505  0.4472395
```

```
##   22  0.2181118  0.4294178
##   23  0.2338087  0.4131571
##   24  0.2512206  0.3984862
##   25  0.2699889  0.3853630
##
## Rsquared was used to select the optimal model using  the largest value.
## The final value used for the model was df = 2.
```

```
gam.fit = gam(ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack, df = 3) + s(PercentageWhi
                                    s(PercentageAsian, df = 3) + s(PercentageHispanic, df = 3) +
                                    s(PercentageUrban, df = 3) + s(MedIncome, df = 3) , data = train2)

summary(gam.fit)
```

```
##
## Call: gam(formula = ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack,
##     df = 3) + s(PercentageWhite, df = 3) + s(PercentageAsian,
##     df = 3) + s(PercentageHispanic, df = 3) + s(PercentageUrban,
##     df = 3) + s(MedIncome, df = 3), data = train2)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -0.44046 -0.06599 -0.01526  0.05165  0.61386
##
## (Dispersion Parameter for gaussian family taken to be 0.0199)
##
##     Null Deviance: 26.5927 on 497 degrees of freedom
## Residual Deviance: 9.4958 on 476 degrees of freedom
## AIC: -512.6927
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                                 Df Sum Sq Mean Sq  F value    Pr(>F)
## s(Population, df = 3)            1 3.5912  3.5912 180.0159 < 2.2e-16 ***
## s(PercentageBlack, df = 3)       1 8.6604  8.6604 434.1243 < 2.2e-16 ***
## s(PercentageWhite, df = 3)       1 1.7614  1.7614  88.2927 < 2.2e-16 ***
## s(PercentageAsian, df = 3)       1 0.4701  0.4701  23.5645 1.641e-06 ***
## s(PercentageHispanic, df = 3)    1 0.7069  0.7069  35.4339 5.125e-09 ***
## s(PercentageUrban, df = 3)       1 0.1213  0.1213   6.0795   0.01403 *
## s(MedIncome, df = 3)             1 0.7132  0.7132  35.7519 4.403e-09 ***
## Residuals                      476 9.4958  0.0199
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                               Npar Df Npar F     Pr(F)
## (Intercept)
## s(Population, df = 3)               2 1.2279 0.2938185
## s(PercentageBlack, df = 3)          2 7.6588 0.0005326 ***
## s(PercentageWhite, df = 3)          2 6.2782 0.0020365 **
## s(PercentageAsian, df = 3)          2 2.5273 0.0809416 .
## s(PercentageHispanic, df = 3)       2 4.3385 0.0135737 *
## s(PercentageUrban, df = 3)          2 0.7415 0.4769328
## s(MedIncome, df = 3)                2 2.9539 0.0530894 .
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

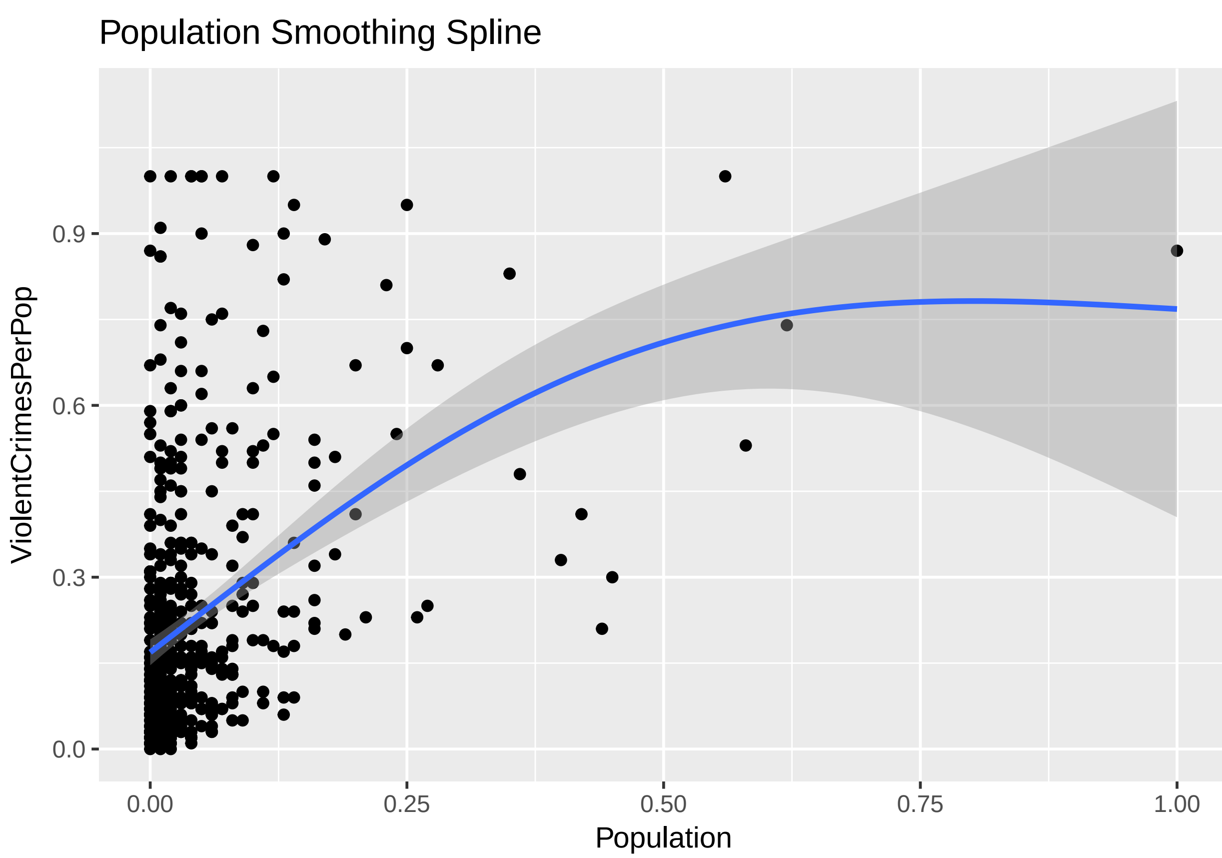
```r
#Test R-squared
rsq(gam.fit, test2, 'ViolentCrimesPerPop')
```

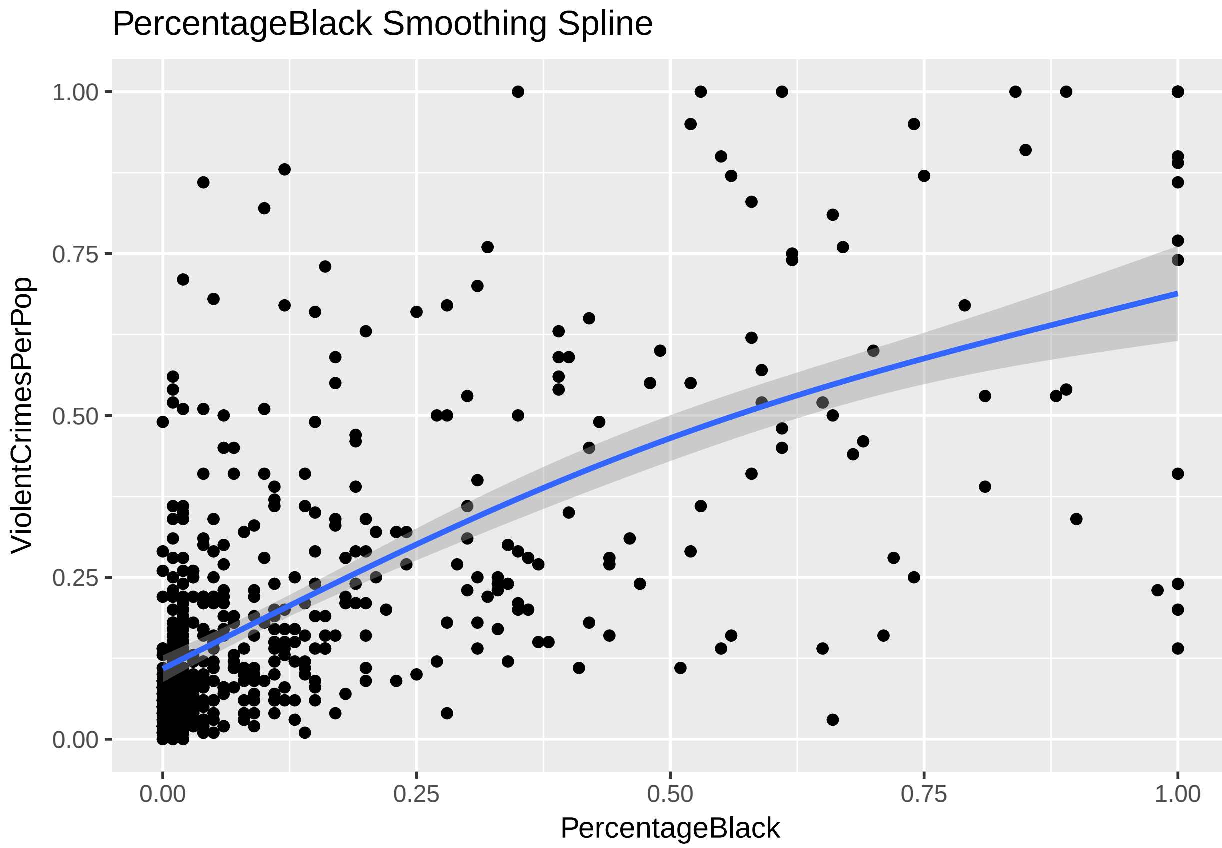
```
## [1] 0.5771822
```

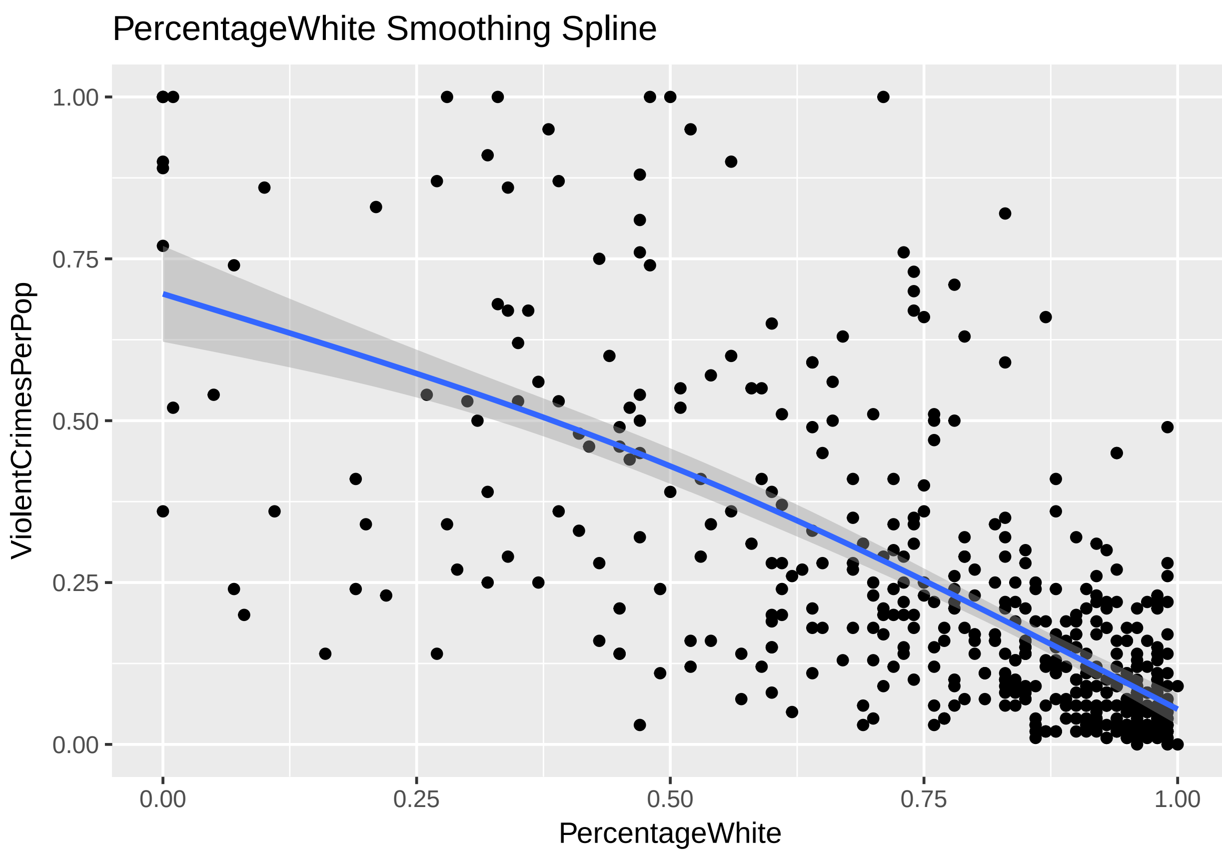
```r
library(mgcv)
```

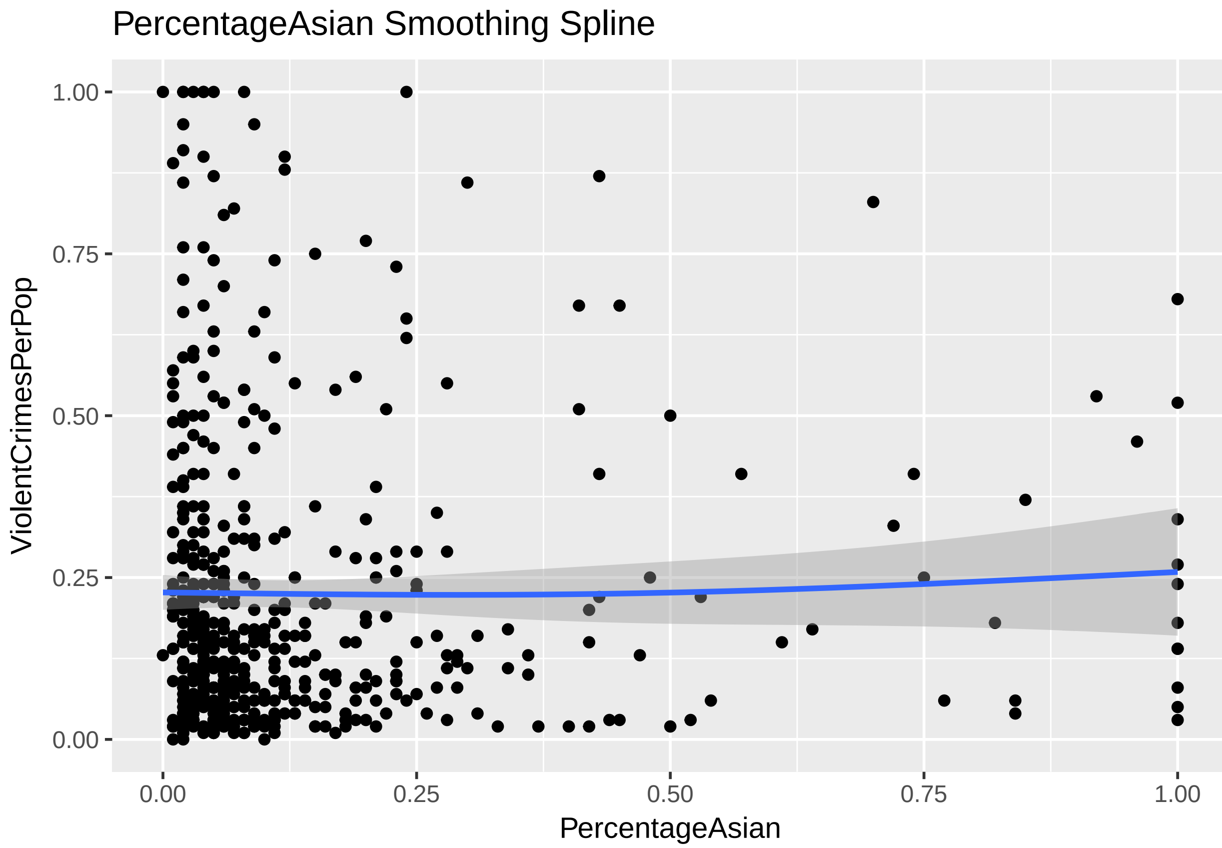
```
## Loading required package: nlme
## This is mgcv 1.8-12. For overview type 'help("mgcv-package")'.
##
## Attaching package: 'mgcv'
## The following objects are masked from 'package:gam':
##
##     gam, gam.control, gam.fit, plot.gam, predict.gam, s,
##     summary.gam
```

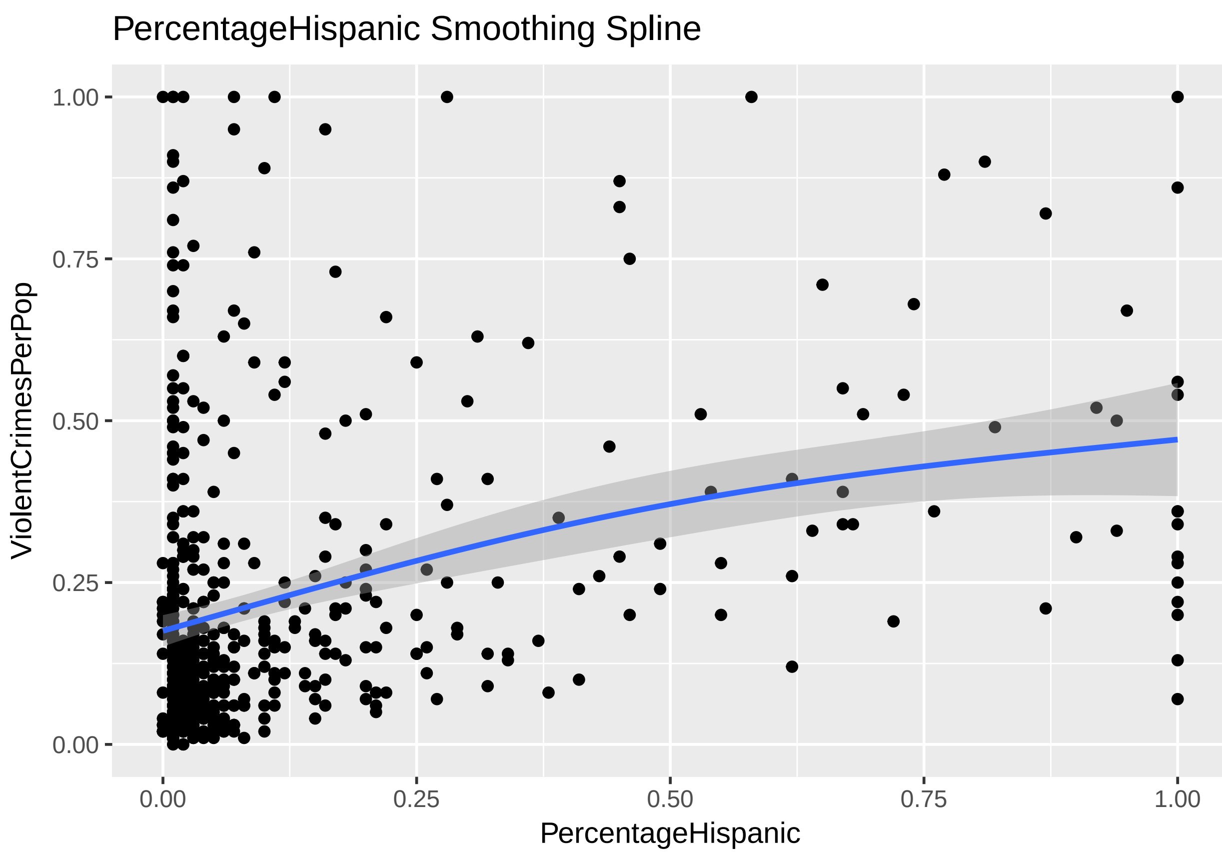
```r
for (name in names(train2)[-1]){
  p = ggplot(train2, aes_string(x = name, y = 'ViolentCrimesPerPop')) + geom_point()
  print(p + ggtitle(paste(name, 'Smoothing Spline')) + stat_smooth(method = 'gam', formula = y ~ s(x, k
}
```

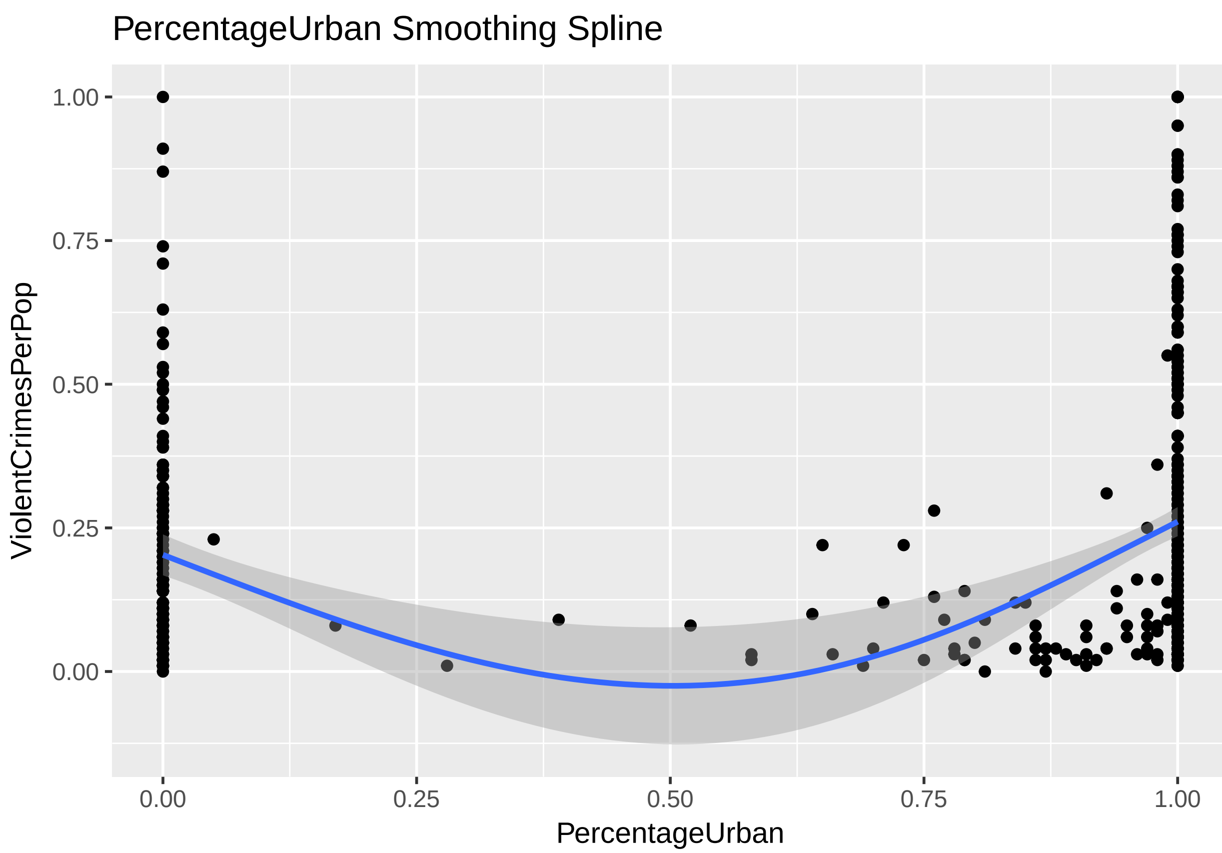


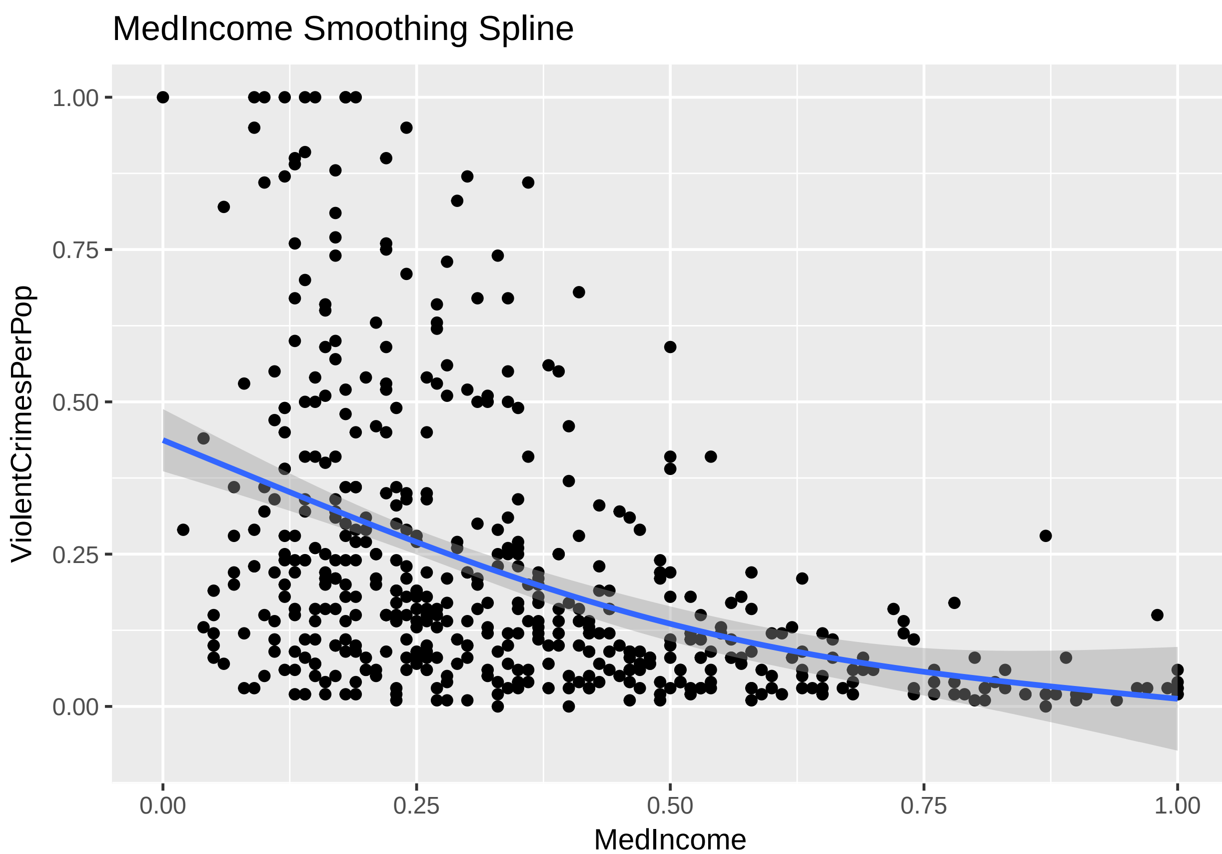

PercentageBlack Smoothing Spline
ViolentCrimesPerPop
PercentageBlack
0.00
0.25
0.50
0.75
1.00
0.00
0.25
0.50
0.75
1.00


PercentageWhite Smoothing Spline
ViolentCrimesPerPop
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
PercentageWhite


PercentageAsian Smoothing Spline
1.00
0.75
0.50
0.25
0.00
ViolentCrimesPerPop
0.00
0.25
0.50
0.75
1.00
PercentageAsian


PercentageHispanic Smoothing Spline
ViolentCrimesPerPop
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
PercentageHispanic


PercentageUrban Smoothing Spline
ViolentCrimesPerPop
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
PercentageUrban



These plots reveal our suspicions that some of the predictors have a non-linear relationship with ViolentCrimesPerPop. However, in general, these smooths do not do a very good job of capturing these trends. This is due to both heteroscedasticity and irreducible error. The heteroscedasticity could potentially be fixed with transformations.

```
detach('package:mgcv', unload = T, force = T)
library(gam)

anova(linear.fit, gam.fit, test = 'Chi')
```

```
## Analysis of Variance Table
##
## Model 1: ViolentCrimesPerPop ~ Population + PercentageBlack + PercentageWhite +
##     PercentageAsian + PercentageHispanic + PercentageUrban +
##     MedIncome
## Model 2: ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack,
##     df = 3) + s(PercentageWhite, df = 3) + s(PercentageAsian,
##     df = 3) + s(PercentageHispanic, df = 3) + s(PercentageUrban,
##     df = 3) + s(MedIncome, df = 3)
##   Res.Df     RSS Df Sum of Sq Pr(>Chi)
## 1    490 10.1458
## 2    476  9.4958 14   0.65004 0.003306 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
gam.fit2 = gam(ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack, df = 3) +
                                     s(PercentageWhite, df = 3) + s(PercentageHispanic, df = 3) +
                                     s(MedIncome, df = 3) , data = train2)
```

```r
summary(gam.fit2)
```

```
##
## Call: gam(formula = ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack,
##     df = 3) + s(PercentageWhite, df = 3) + s(PercentageHispanic,
##     df = 3) + s(MedIncome, df = 3), data = train2)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -0.43959 -0.06970 -0.01944  0.04648  0.64055
##
## (Dispersion Parameter for gaussian family taken to be 0.0206)
##
##     Null Deviance: 26.5927 on 497 degrees of freedom
## Residual Deviance: 9.9459 on 482.0001 degrees of freedom
## AIC: -501.6304
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                                  Df Sum Sq Mean Sq F value    Pr(>F)
## s(Population, df = 3)             1 3.5838  3.5838 173.677 < 2.2e-16 ***
## s(PercentageBlack, df = 3)        1 8.8361  8.8361 428.216 < 2.2e-16 ***
## s(PercentageWhite, df = 3)        1 1.7113  1.7113  82.935 < 2.2e-16 ***
## s(PercentageHispanic, df = 3)     1 1.1134  1.1134  53.956 8.815e-13 ***
## s(MedIncome, df = 3)              1 0.5008  0.5008  24.269 1.153e-06 ***
## Residuals                       482 9.9459  0.0206
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                                 Npar Df Npar F     Pr(F)
## (Intercept)
## s(Population, df = 3)                 2 3.0774 0.0469872 *
## s(PercentageBlack, df = 3)            2 7.1344 0.0008844 ***
## s(PercentageWhite, df = 3)            2 6.5742 0.0015249 **
## s(PercentageHispanic, df = 3)         2 3.4709 0.0318654 *
## s(MedIncome, df = 3)                  2 1.8098 0.1647904
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
anova(gam.fit2, gam.fit, test = 'Chi')
```

```
## Analysis of Deviance Table
##
## Model 1: ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack,
##     df = 3) + s(PercentageWhite, df = 3) + s(PercentageHispanic,
##     df = 3) + s(MedIncome, df = 3)
## Model 2: ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack,
##     df = 3) + s(PercentageWhite, df = 3) + s(PercentageAsian,
##     df = 3) + s(PercentageHispanic, df = 3) + s(PercentageUrban,
##     df = 3) + s(MedIncome, df = 3)
##   Resid. Df Resid. Dev     Df Deviance  Pr(>Chi)
## 1       482     9.9459
## 2       476     9.4958 6.0001   0.4501 0.0009573 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Based on the p-values from the likelihood ratio tests, we have enough to suggest that the GAM model is significantly better than the linear model at the .001 significance level, and that the full GAM model (including PercentageAsian and PercentageUrban) is better than the reduced GAM model.

2.c.

```r
library(caret)
y = 1:nrow(train2)
folds = createFolds(y, k = 5, list = TRUE, returnTrain = FALSE)
folds
```

```
## $Fold1
##   [1]   1   5  25  28  30  46  58  59  62  70  71  73  77  78  81  83  84
##  [18]  89  92  94 111 113 114 115 122 129 131 133 138 139 141 142 144 145
##  [35] 151 152 164 168 174 181 186 192 195 196 200 211 228 237 243 246 254
##  [52] 262 275 291 297 301 304 307 309 312 313 319 325 326 327 330 331 343
##  [69] 345 351 352 360 361 364 372 385 397 402 409 413 414 416 420 441 454
##  [86] 455 463 469 471 473 474 477 482 484 485 486 488 490 492 497
##
## $Fold2
##   [1]  12  16  18  23  31  33  35  39  40  41  43  45  47  56  72  74  82
##  [18]  88  97  99 100 104 116 119 125 126 127 135 136 140 143 146 148 153
##  [35] 155 170 179 189 190 209 213 218 226 227 233 234 235 241 242 244 251
##  [52] 255 261 276 278 281 283 287 292 299 303 308 310 320 321 323 324 333
##  [69] 340 348 349 356 357 367 368 374 375 377 386 387 388 389 401 405 411
##  [86] 412 421 423 424 428 434 440 442 448 453 457 459 467 475 483
##
## $Fold3
##   [1]   4   7   8  11  20  26  29  36  37  42  51  55  57  60  69  80  93
##  [18]  95 101 102 103 105 106 107 112 128 147 158 159 162 163 165 167 172
##  [35] 175 177 178 180 182 183 185 193 198 199 201 204 205 214 220 225 250
##  [52] 252 258 264 265 267 268 269 282 284 286 289 306 316 317 318 334 339
##  [69] 350 353 359 366 369 371 373 379 380 384 391 394 404 406 410 417 419
##  [86] 427 431 439 447 449 451 458 460 461 464 466 476 478 489 491
##
## $Fold4
##  [1]   2   3   6   9  14  15  17  19  21  24  32  44  49  50  54  65  66
## [18]  67  68  75  85  86  91 108 120 132 137 149 150 154 160 161 166 169
## [35] 171 173 191 194 197 207 208 210 219 222 223 229 231 238 245 248 253
## [52] 257 260 263 266 270 273 277 290 295 296 302 311 315 329 335 336 337
## [69] 338 344 347 355 358 365 376 378 381 382 400 403 407 408 425 426 433
## [86] 437 438 446 450 456 465 468 470 479 487 493 494 495 496
##
## $Fold5
##  [1]  10  13  22  27  34  38  48  52  53  61  63  64  76  79  87  90  96
## [18]  98 109 110 117 118 121 123 124 130 134 156 157 176 184 187 188 202
## [35] 203 206 212 215 216 217 221 224 230 232 236 239 240 247 249 256 259
## [52] 271 272 274 279 280 285 288 293 294 298 300 305 314 322 328 332 341
## [69] 342 346 354 362 363 370 383 390 392 393 395 396 398 399 415 418 422
## [86] 429 430 432 435 436 443 444 445 452 462 472 480 481 498
```

```r
param.grid = seq(.1, 2, by = .1)
avg.rsq1 = c()
avg.rsq2 = c()
for (s in param.grid){

  test.rsq1 = c()
  test.rsq2 = c()
  for (i in 1:5){

    test.index = folds[[i]]
    temp.test = train2[test.index, ]
    temp.train = train2[-test.index, ]

    gam.fit1 = gam(ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack, df = 3) +
                                         s(PercentageWhite, df = 3) + s(PercentageAsian, df = 3) +
                                         s(MedIncome, df = 3) + lo(Population*PercentageUrban, span = s)
                                         lo(Population*MedIncome, span = s) + lo(MedIncome*PercentageUrl
                                                                                 span = s), data = temp

    gam.fit2 = gam(ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack, df = 3) +
                                         s(PercentageWhite, df = 3) + s(PercentageAsian, df = 3) +
                                                                          data = temp.train)

    test.rsq1 = c(test.rsq1, rsq(gam.fit1, temp.test, 'ViolentCrimesPerPop'))
    test.rsq2 = c(test.rsq2, rsq(gam.fit2, temp.test, 'ViolentCrimesPerPop'))


    if (i == 5){avg.rsq1 = c(avg.rsq1, mean(test.rsq1))
                avg.rsq2 = c(avg.rsq2, mean(test.rsq2))
                }

  }
}

# gam.fit1 results
cbind(span = param.grid, avg.rsq1)
```

```
##       span  avg.rsq1
##  [1,]  0.1 0.4449836
##  [2,]  0.2 0.5443038
##  [3,]  0.3 0.5784091
##  [4,]  0.4 0.5888384
##  [5,]  0.5 0.5964961
##  [6,]  0.6 0.6011031
##  [7,]  0.7 0.6041028
##  [8,]  0.8 0.6036658
##  [9,]  0.9 0.6027810
## [10,]  1.0 0.6036008
## [11,]  1.1 0.6034233
## [12,]  1.2 0.6031582
## [13,]  1.3 0.6016863
## [14,]  1.4 0.6014846
## [15,]  1.5 0.6012941
## [16,]  1.6 0.6011208
```

```
## [17,]  1.7 0.6009654
## [18,]  1.8 0.6008265
## [19,]  1.9 0.6007025
## [20,]  2.0 0.6005918
```

```
# gam.fit1 best span
bs1 = param.grid[which.max(avg.rsq1)]

# gam.fit2 results
cbind(span = param.grid, avg.rsq2)
```

```
##       span  avg.rsq2
##  [1,]  0.1 0.5720792
##  [2,]  0.2 0.5891954
##  [3,]  0.3 0.5959968
##  [4,]  0.4 0.5996095
##  [5,]  0.5 0.6011539
##  [6,]  0.6 0.6018136
##  [7,]  0.7 0.6016750
##  [8,]  0.8 0.6019169
##  [9,]  0.9 0.6026947
## [10,]  1.0 0.6024256
## [11,]  1.1 0.6026446
## [12,]  1.2 0.6027849
## [13,]  1.3 0.6029537
## [14,]  1.4 0.6029863
## [15,]  1.5 0.6030058
## [16,]  1.6 0.6030184
## [17,]  1.7 0.6030268
## [18,]  1.8 0.6030328
## [19,]  1.9 0.6030370
## [20,]  2.0 0.6030402
```

```
# gam.fit2 best span
bs2 = param.grid[which.max(avg.rsq2)]
```

```
gam.fit1 = gam(ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack, df = 3) +
                                     s(PercentageWhite, df = 3) + s(PercentageAsian, df = 3) +
                                     s(MedIncome, df = 3) + lo(Population*PercentageUrban, span = bs1) +
                                     lo(MedIncome*PercentageUrban, span = bs1), data = train2)

gam.fit2 = gam(ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack, df = 3) +
                                     s(MedIncome, df = 3) + lo(MedIncome*PercentageBlack, span = bs2),
                                                          data = train2)

anova(gam.fit1, gam.fit, test = 'Chi')
```

```
## Analysis of Deviance Table
##
## Model 1: ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack,
##     df = 3) + s(PercentageWhite, df = 3) + s(PercentageAsian,
##     df = 3) + s(PercentageHispanic, df = 3) + s(PercentageUrban,
##     df = 3) + s(MedIncome, df = 3) + lo(Population * PercentageUrban,
##     span = bs1) + lo(Population * MedIncome, span = bs1) + lo(MedIncome *
##     PercentageUrban, span = bs1)
## Model 2: ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack,
```

```
##     df = 3) + s(PercentageWhite, df = 3) + s(PercentageAsian,
##     df = 3) + s(PercentageHispanic, df = 3) + s(PercentageUrban,
##     df = 3) + s(MedIncome, df = 3)
##   Resid. Df Resid. Dev      Df Deviance Pr(>Chi)
## 1    465.95     9.1436
## 2    476.00     9.4958 -10.053 -0.35221  0.05708 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
anova(gam.fit2, gam.fit, test = 'Chi')
```

```
## Analysis of Deviance Table
##
## Model 1: ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack,
##     df = 3) + s(PercentageWhite, df = 3) + s(PercentageAsian,
##     df = 3) + s(PercentageHispanic, df = 3) + s(PercentageUrban,
##     df = 3) + s(MedIncome, df = 3) + lo(MedIncome * PercentageBlack,
##     span = bs2)
## Model 2: ViolentCrimesPerPop ~ s(Population, df = 3) + s(PercentageBlack,
##     df = 3) + s(PercentageWhite, df = 3) + s(PercentageAsian,
##     df = 3) + s(PercentageHispanic, df = 3) + s(PercentageUrban,
##     df = 3) + s(MedIncome, df = 3)
##   Resid. Df Resid. Dev      Df Deviance  Pr(>Chi)
## 1    474.82     9.2003
## 2    476.00     9.4958 -1.1783 -0.29554 0.0001332 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Based on these tests, we have enough evidence to suggest that including the interactions between 'Population', 'PercentageUrban', and 'MedIncome' significantly improves our model, and we have evidence that the interaction between 'PercentageBlack' and 'MedIncome' improves our model.