Homework Week 6

Chris Messer

2022-09-30

Question 9.1

Using the same crime data set uscrime.txt as in Question 8.2, apply Principal Component Analysis and then create a regression model using the first few principal components. Specify your new model in terms of the original variables (not the principal components), and compare its quality to that of your solution to Question 8.2. You can use the R function prcomp for PCA. (Note that to first scale the data, you can include scale. = TRUE to scale as part of the PCA function. Don't forget that, to make a prediction for the new city, you'll need to unscale the coefficients (i.e., do the scaling calculation in reverse)!)

> In the below analysis, I used principal component analysis tor educe the dimensions of the dataset, and then fed them into a linear regressino model. I determined that 7 principal components was the optimal number. The resulting linear model is
>
> y = 65.22*PC1 -70.08*PC2 +25.19*PC3 +69.45*PC4 -229.04*PC5 -60.21*PC6 +117.26*PC7 + 905
>
> We can then transform those coefficients back into their original (non principal) coefficients, and scale them back and that results in the equation:
>
> y = 55.237* M + 139.757* So + -6.804* Ed + 44.586* Po1 + 46.424* Po2 + 673.381* LF + 44.403* M.F + 0.96* Pop + 5.685* NW + -1027.735* U1 + 24.416* U2 + 0.029* Wealth + 12.451* Ineq + -5170.569* Prob + -2.215* Time - 5498.458
>
> The R^2 value of the above approach is .5306. Compare this to my best R^2 value using last weeks pure linear regression model with no PCA, which had an R^2 of .69.
>
> It is reasonable that using PCA to reduce dimensionality did not make a better model than using straight linear regression with only a few predictors. That is because PCA takes into account all of the features, even if only using a handful of principal components in the linear model.
>
> See further analysis below.

Analysis

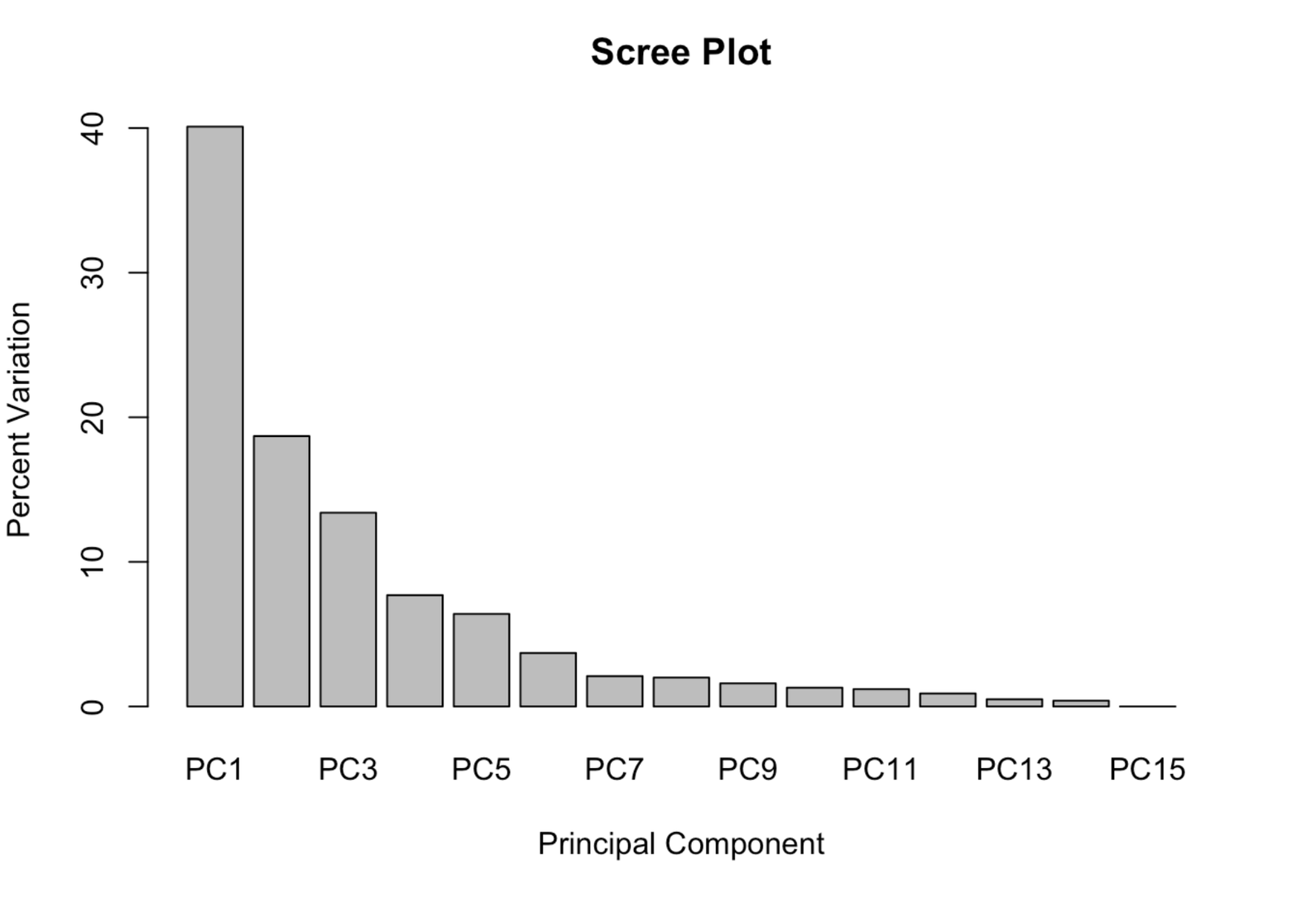
First, let's load in the data.

```
data <- read.csv('uscrime.txt', sep = '\t')
```

Now, let's call prcomp() to perform principal component analysis.

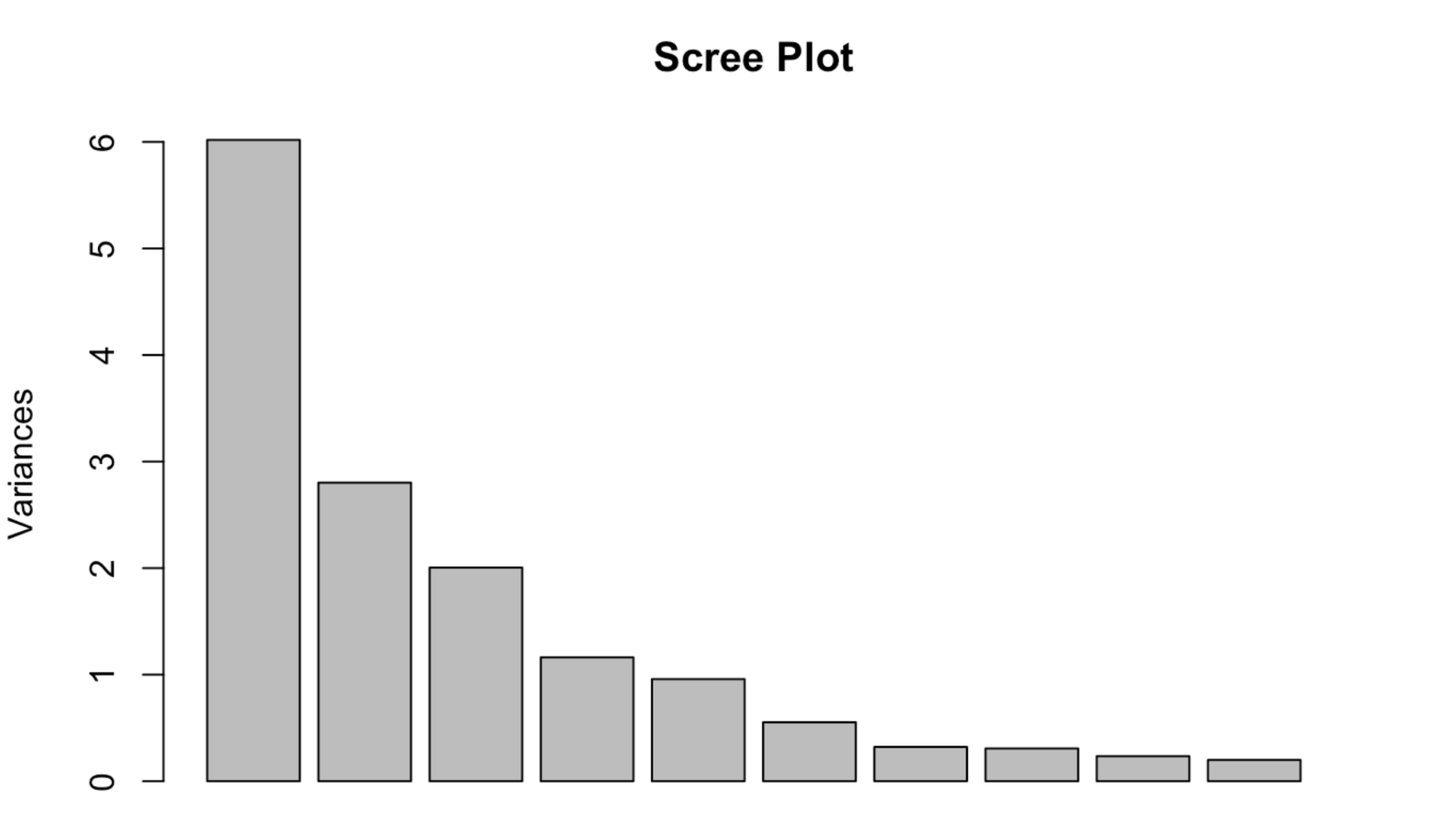
```
pca <- prcomp(~.,data[,1:15], scale = TRUE)
```

```
#These are the eigenvalues, we can manually calculate the percentage variance
pca.var <- pca$sdev^2
pca.var.per <- round(pca.var/sum(pca.var)*100,1)

#or we can get it from prcomp
pca.var.per <- round(summary(pca)$importance[2,]*100,1)

#now plot it out
barplot(pca.var.per,
        main ="Scree Plot",
        xlab= "Principal Component",
        ylab = "Percent Variation")
```



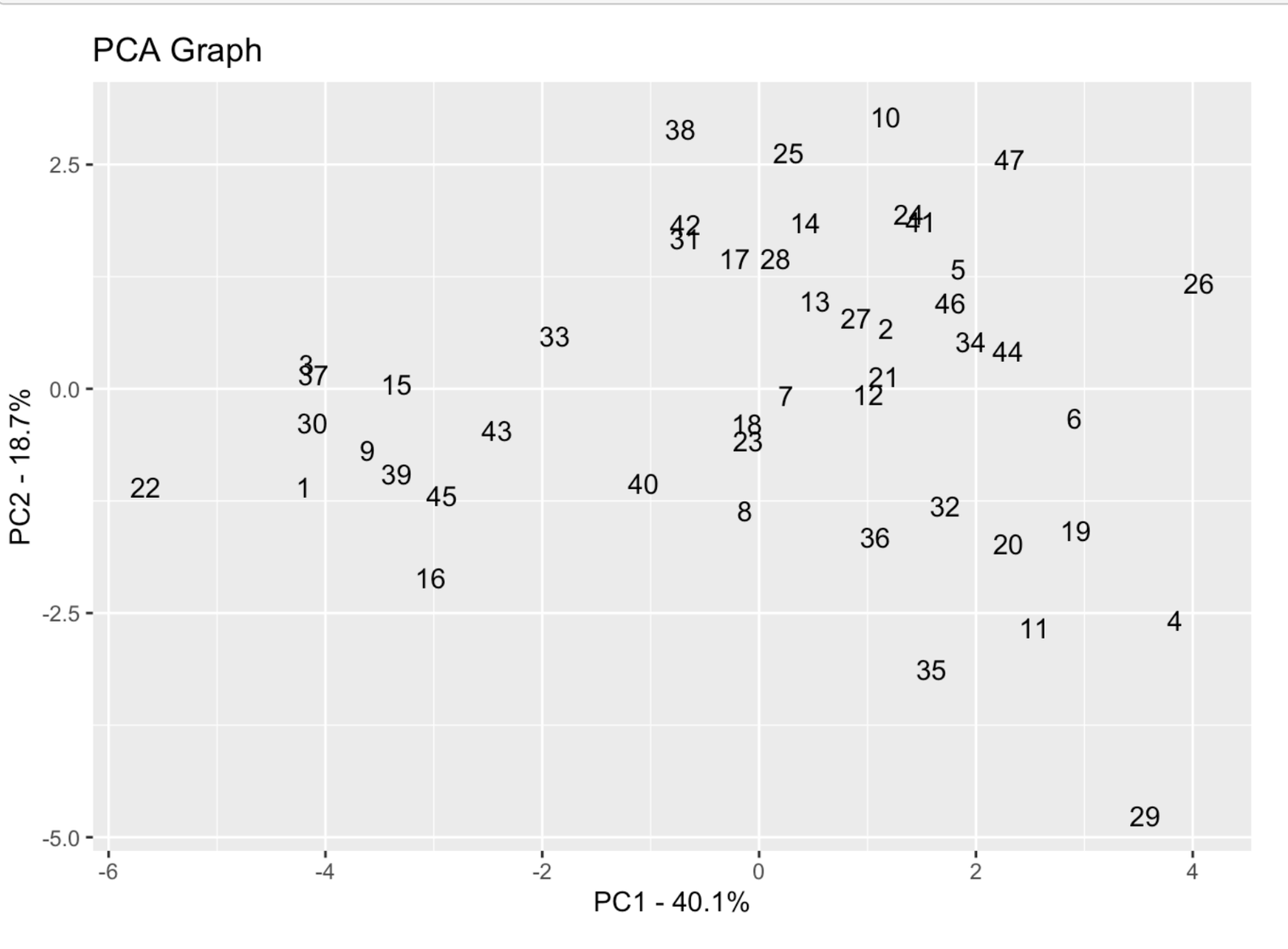
Alternatively, we can just call screeplot on our prcomp() object.

```
screeplot(pca,
          main ="Scree Plot",
          )
```



Now, I'd like to dig in a little further on the first two principal components, as the two of them make up 60% of the variance in the predicted crime rate.

```
library(ggplot2)
pca.data <- data.frame(X=pca$x[,1],
                       Y=pca$x[,2])

ggplot(data = pca.data, aes(x=X, y=Y, label=(1:47)))+
  geom_text() +
  xlab(paste("PC1 - ",pca.var.per[1], "%", sep="")) +
  ylab(paste("PC2 - ",pca.var.per[2], "%", sep="")) +
  ggtitle("PCA Graph")
```



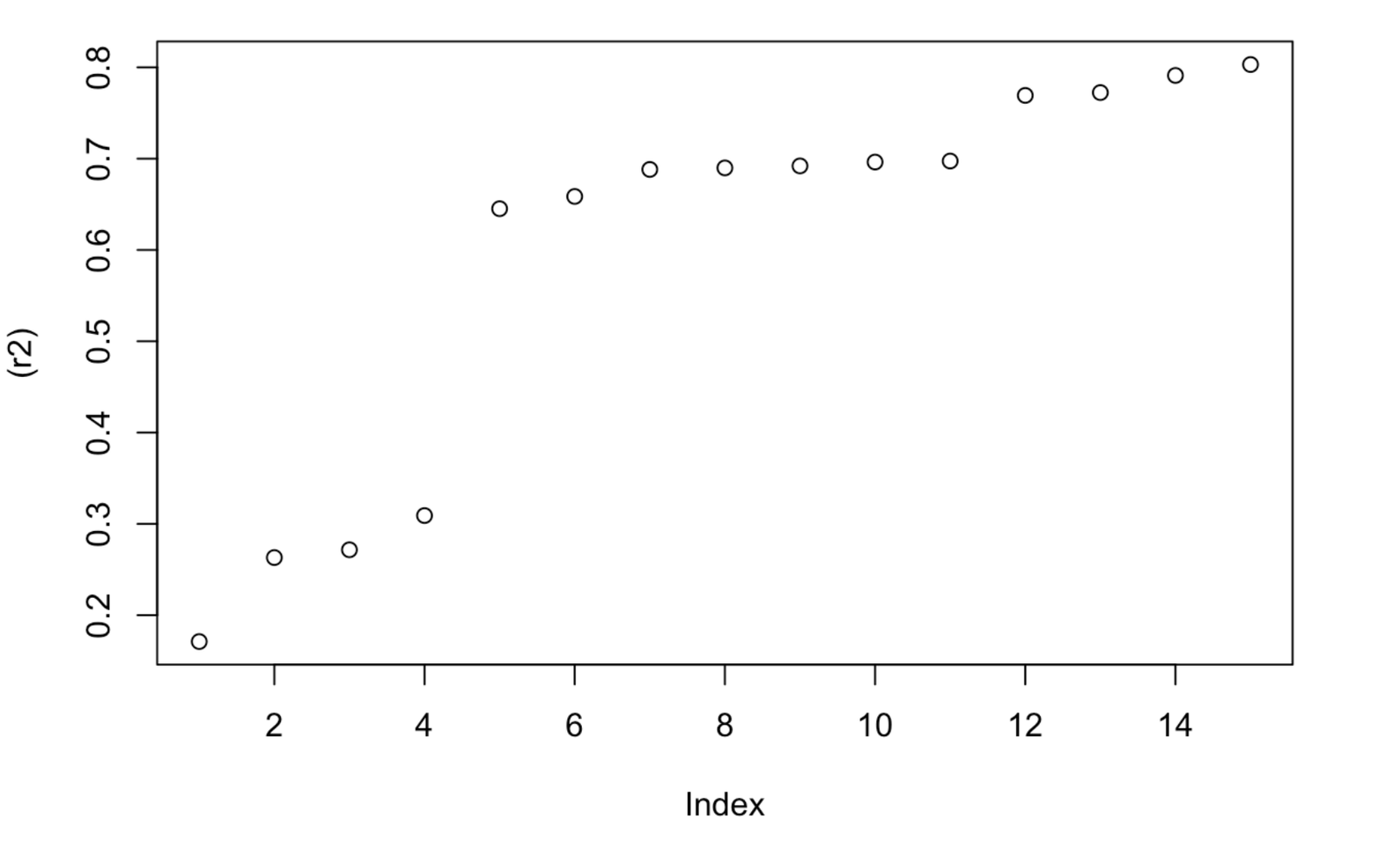
So what is this graph telling us? Honestly, not a lot in this graph! Typically, a graph like this could be potentially helpful in a scenario where we wanted to maybe do some clustering on a high dimensional data set. Unfortunately, even though the first two PC's account for 60% of the variation, no new patterns emerge. The below highlights the different features that are "pushing" the points above to the left or right on the above graph. While not particularly insightful in this analysis, typically, this could give us more information about which features are driving the classification of a data point.

```
loading_scores <- pca$rotation
feature_score <- abs(loading_scores)
feature_score_ranked <- sort(feature_score[,1], decreasing = T)
top <- names(feature_score_ranked)
pca$rotation[top,1]
```

```
##      Wealth        Ineq          Ed          So         Po2         Po1
##  0.37970331 -0.36579778  0.33962148 -0.33088129  0.31099285  0.30863412
##           M          NW        Prob          LF         M.F         Pop
## -0.30371194 -0.29358647 -0.25888661  0.17617757  0.11638221  0.11307836
##          U1        Time          U2
##  0.04050137 -0.02062867  0.01812228
```

Now, let's go ahead and build a linear model, and loop over all of the PC combinations and look at the r^2 for each.

```
r2 <- c()
for (i in 1:15){
  linear_data <- as.data.frame(cbind(pca$x[,1:i],Crime = data[,16]))
  pca.model <- lm(Crime~.,data = linear_data)
  r2[i] <- summary(pca.model)$r.squared
}

plot((r2))
```



Looking at our Scree Plot from above, it looks our optimal number of principal components to look at is probably around 4 or 5. So why does out model just get better and better the more PC's we use in the model? Well, if you remember from last week, there was an issue with over fitting when using all 15 features to predict. And when using all 15 features of PCA in a linear regression model, we are effectively using all 15 features from last week's analysis! In fact, we get the exact same r^2 values.

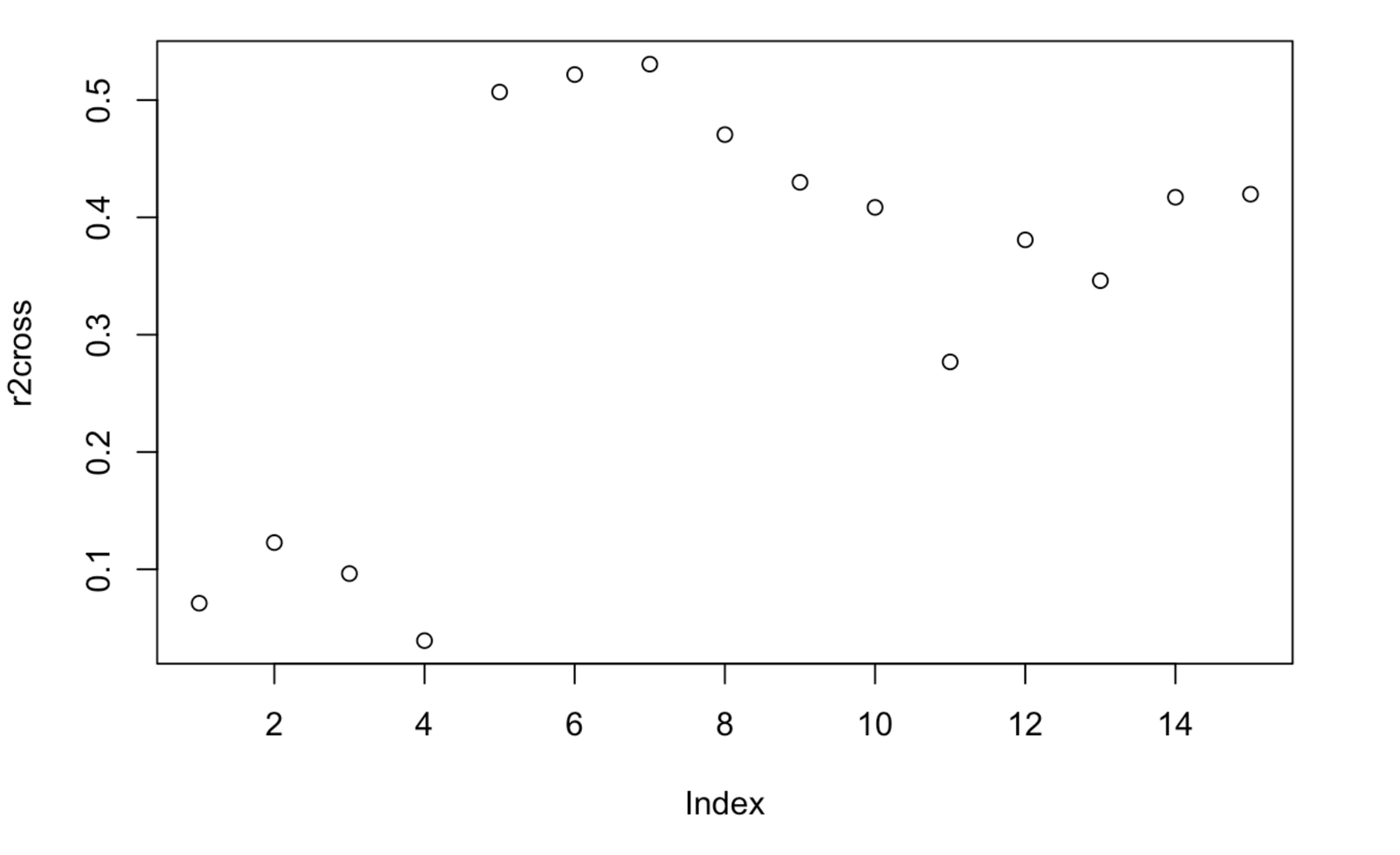
So, a better measure would be to cross validate the linear regression model that uses the principal components.

Run a cross validated linear regression model for all combinations of principal components

```
library("DAAG")

nComp = 15
r2cross <- c()
r2cross2 <- c()
for (Comp in seq(nComp)){

  principal_components <- as.data.frame(cbind(pca$x[,1:Comp],Crime = data[,16]))
  pca.model.lm <- lm(Crime~.,data = principal_components)
  pca.model.cv <- cv.lm(data = principal_components, pca.model.lm, m = 5, printit=F,plotit=F)

  SSres <- attr(pca.model.cv,"ms")*nrow(data)
  SStotal <-  sum((data$Crime - mean(data$Crime))^2)

  r2 <- 1- (SSres/SStotal)
  r2cross[Comp] <- r2

}

r2cross
```

```
##  [1] 0.07108770 0.12277954 0.09633706 0.03917870 0.50684789 0.52176054
##  [7] 0.53062411 0.47056384 0.42989301 0.40853328 0.27681909 0.38084668
## [13] 0.34605680 0.41716859 0.41975895
```

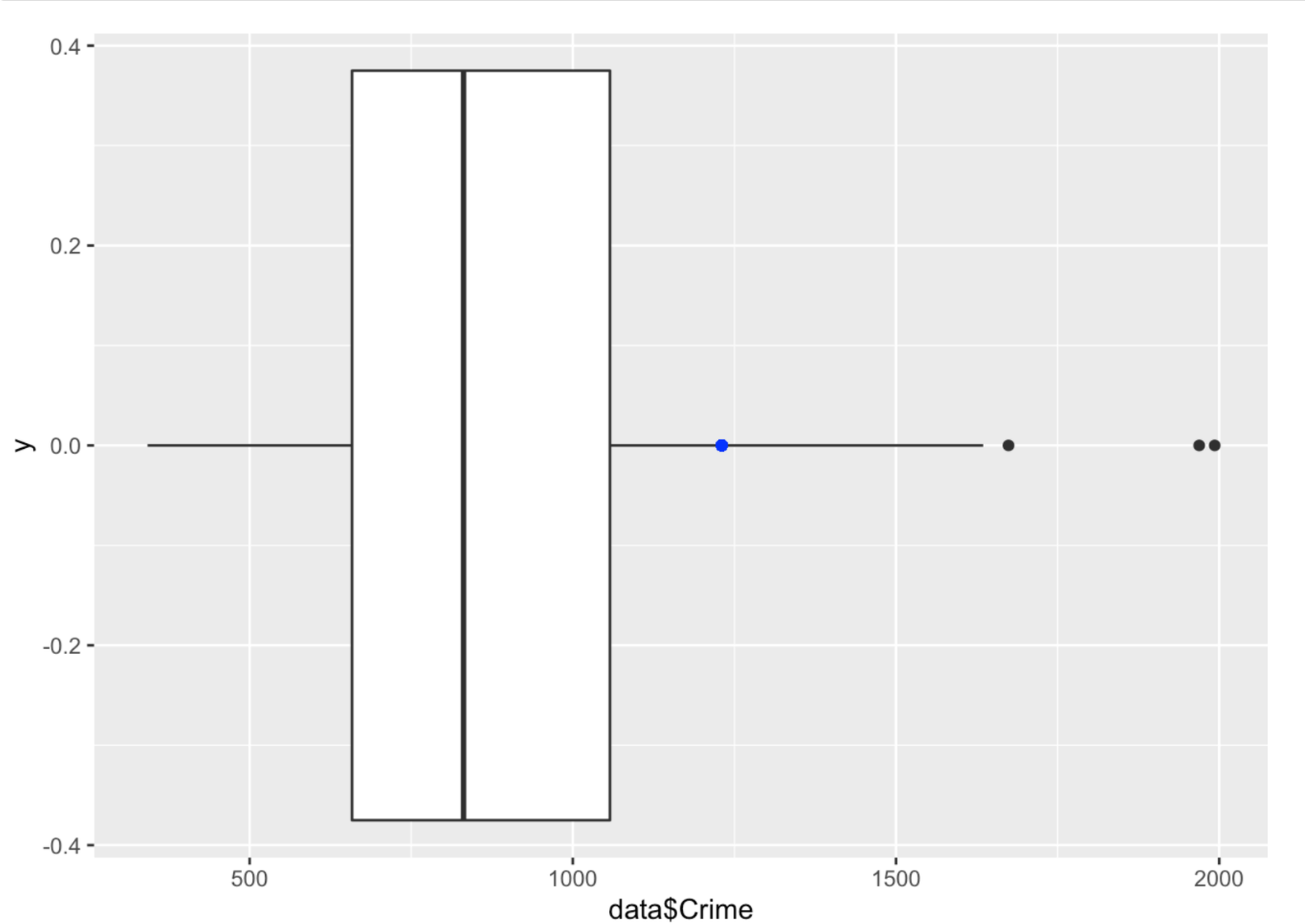
```
plot(r2cross)
```



As shown above, it is clear now that using all 15 PC's does not give us the best model. The r^2 value actually went from .5 to around .4, and our highest r^2 is when we used 7 principal components.

Now that we have our best model that uses PCA, we will use make a prediction using our model.

```
#Store the point to predict into a dataframe
new_data <- data.frame(M = 14.0,So = 0, Ed = 10.0, Po1 = 12.0, Po2 = 15.5,LF = 0.640, M.F = 94.0, Pop = 150, NW =
1.1, U1 = 0.120, U2 = 3.6, Wealth = 3200, Ineq = 20.1, Prob = 0.040,Time = 39.0)

#transform our datapoint into it's principal components, so that we can multiply it by the coefficients from our
linear model. Also note, we DO NOT need to scale our data, because when we call predict in the below line of code
and pass in our pca object and the data point to convert to principal components, it scales and centers the data
for us.
new_data.pc <- as.data.frame(predict(pca, new_data))

#create a new model using the best cross validated model from above

linear_data.best <- as.data.frame(cbind(pca$x[,1:7],Crime = data[,16]))
pca.model.winner <- lm(Crime~.,data = linear_data.best)



#predict our crime rate using our principal components. We now pass our new_data object (which is our original da
ta point to predict) that has been scaled and centered, into a predict() call.
new_data.pred <-predict(pca.model.winner, new_data.pc[,1:7])

p <- ggplot(as.data.frame(data$Crime), aes(x=data$Crime)) +
  geom_boxplot()
p + geom_point(aes(x=new_data.pred[[1]], y = 0), colour="blue")
```




```
new_data.pred
```

```
##        1
## 1230.418
```

```
pca.model.winner
```

```
##
## Call:
## lm(formula = Crime ~ ., data = linear_data.best)
##
## Coefficients:
## (Intercept)          PC1          PC2          PC3          PC4          PC5
##      905.09        65.22       -70.08        25.19        69.45      -229.04
##         PC6          PC7
##      -60.21       117.26
```

The resulting formula is

y = 65.22*PC1 -70.08*PC2 +25.19*PC3 +69.45*PC4 -229.04*PC5 -60.21*PC6 +117.26*PC7 + 905

Now, if we'd like, we can convert the principal component coefficients back into the feature coefficients by matrix multiplying them by their rotation values, and then "unscale" them, so that they could be applied to unscaled features later.

```
beta0 <- pca.model.winner$coefficients[1]
betas <- pca.model.winner$coefficients[2:8]
betas
```

```
##       PC1       PC2       PC3       PC4       PC5       PC6       PC7
##  65.21593 -70.08312  25.19408  69.44603 -229.04282 -60.21329 117.25590
```

```
alphas <- pca$rotation[,1:7] %*% betas
t(alphas)
```

```
##             M       So        Ed      Po1      Po2       LF      M.F      Pop
## [1,] 69.42028 66.94019 -7.611451 132.5061 129.8085 27.21254 130.8437 36.54482
##            NW        U1       U2   Wealth     Ineq      Prob      Time
## [1,] 58.45756 -18.52881 20.62032 27.82379 49.67512 -117.5631 -15.69815
```

y = 69.42028*M + 66.94019*So -7.611451*Ed + 132.5061 Po1 + 129.8085Po2 + 27.21254*LF + 130.8437*MF + 36.54482 * Pop + 58.45756*NW -18.52881*U1 + 20.62032*U2 + 27.82379*Wealth + 49.67512*Ineq -117.5631*Prob -15.69815*Time + 905

Now that the values have been transformed out of their principal component coefficients and back into their original coefficients, we need to "unscale" them, since they were calculated using scaled inputs.

Algebraically, to scale a point, you need to subtract the mean of the population and divide by the standard deviation of the population. So to unscale it, we need to do the opposite. Another way to look at it is if

y = b0 + sum($s_i * a_i$)

and s_i is our scaled input point, and a_i is our scaled alpha coefficient, we can solve for our unscaled alpha of u_i using an equality.

Looking at just the coefficient part of the above formula, $a_i * s_i$, lets replace s_i with (x_i - mean)/sd. And as discussed above, the scaled alpha * the scaled datapoint should be equal to **unscaled alpha * unscaled data point.** So we will put the bolded part on the right side of the equation.

So now we have

a_i * (x_i - mean)/sd = $x_i * u_i$

So now solve for u_i and you get

u_i = a_i / sd

and the beta coefficient =

b = (a_i * mean)/ sd

Now lets write that out in r

```
#features of the data (excludes the crime predicition)
feats <- data[,1:15]

unscaled_alphas <- alphas/sapply(data[,1:15],sd)
unscaled_beta0 <- beta0 - sum(alphas * sapply(feats, mean) / sapply(feats, sd))

sum_product_of_coefficients <- sum(as.data.frame(unscaled_alphas[,1] * as.data.frame(new_data)))

manually_calculated_pred <- sum_product_of_coefficients + unscaled_beta0
print(c(manually_calculated_pred, new_data.pred))
```

```
## (Intercept)           1
##    1230.418    1230.418
```

And just like that, we have proven that you can calculate your prediction by either:

a. Using PCA, running a linear regression model, and then to make a prediction just transform the datapoint by scaling it and converting it by the same PCA factors like shown in the first part of this analysis

or

b. Using PCA, running a linear regression model, and then to make a prediction, convert the resulting PCA coefficients back to the original coefficients with matrix multiplication, and then unscale them.

Either approach gives the same answer! However, it is generally advisable to go with the second option. Converting the coefficients back to their original values rather than PC's helps reduce the black box impact of explaining your algorithm. Below is the resulting formula:

```
rbind(round(unscaled_alphas,3), unscaled_beta0)
```

```
##                (Intercept)
## M                   55.237
## So                 139.757
## Ed                  -6.804
## Po1                 44.586
## Po2                 46.424
## LF                 673.381
## M.F                 44.403
## Pop                  0.960
## NW                   5.685
## U1               -1027.735
## U2                  24.416
## Wealth               0.029
## Ineq                12.451
## Prob             -5170.569
## Time                -2.215
## unscaled_beta0   -5498.458
```

y = 55.237* M + 139.757* So + -6.804* Ed + 44.586* Po1 + 46.424* Po2 + 673.381* LF + 44.403* M.F + 0.96* Pop + 5.685* NW + -1027.735* U1 + 24.416* U2 + 0.029* Wealth + 12.451* Ineq + -5170.569* Prob + -2.215* Time - 5498.458

Compare to last week

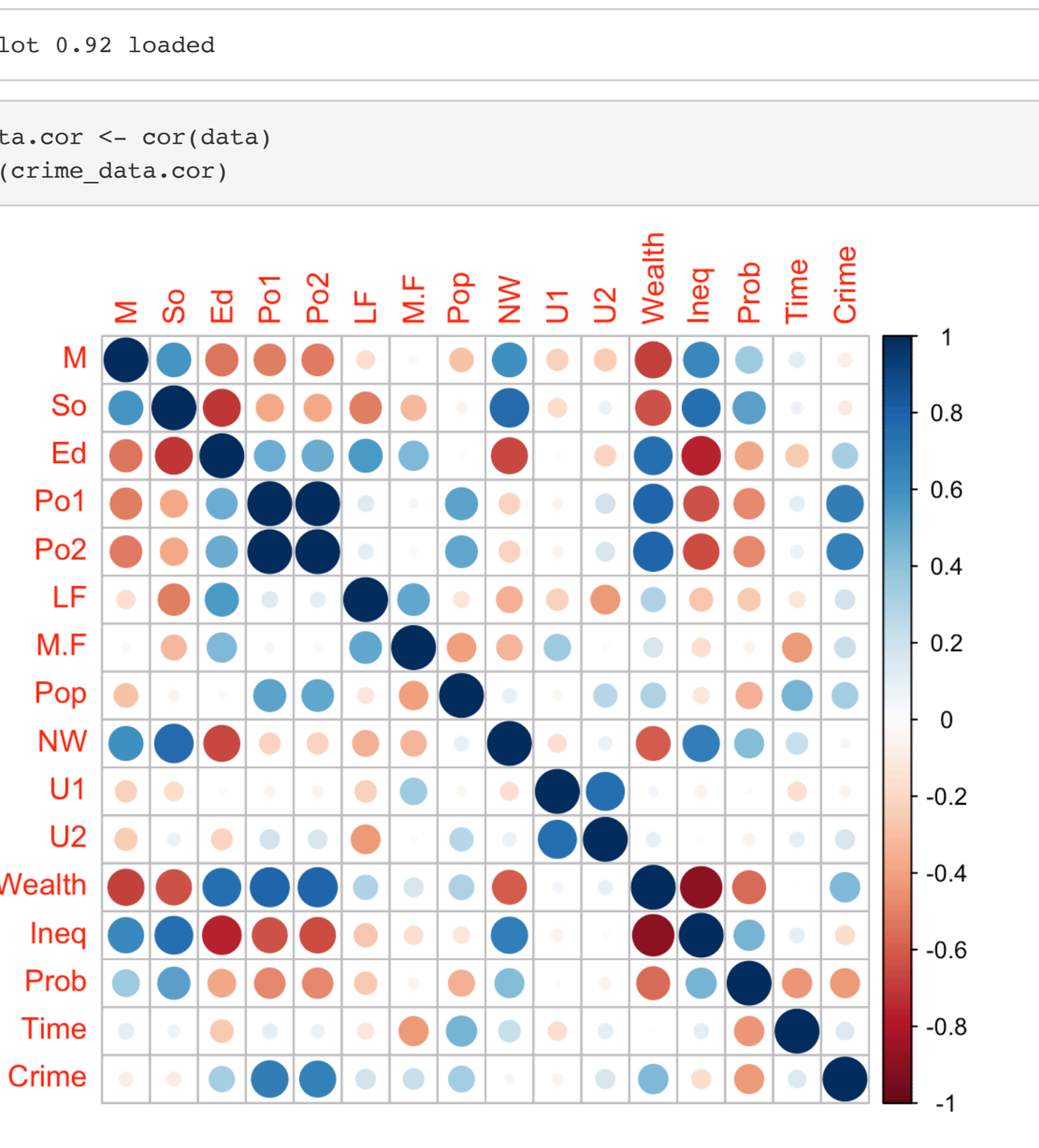
So our R^2 of the best performing model using the optimal number of principal components was .53, but why is that lower than last weeks model using just regular linear regression with only 5 variables?

Well, lets take a look at the correlation plot of our original variables.

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
crime_data.cor <- cor(data)
corrplot(crime_data.cor)
```



We can see that there are a lot of combinations of predictors that do not have much of a correlation with crime. PCA ultimately takes into account all dimensions we include in it, evenif we don't use all of the principal components in the linear model. Each PC is a combination of all of the predictors! So including ALL of the features in the PCA ultimately is not always the best decision. Doing so introduced a lot of "fuzz" in our model, because the features we were using to predict crime did not have any correlation to crime.

As such, it would be best to compare our principal component linear model to a linear model using all predictors.

```
lm_model <- lm(Crime~., data)
model <- cv.lm(data, lm_model, m=5, seed=10, printit=F, plotit =F)

#the model does not outpuut r^2, so we will have to calculate ourselves. The formula is 1-(Sum of Squared Errors/
sumofsquared differences)
SSres <- attr(model,"ms")*nrow(data)
SStotal <- sum((data$Crime - mean(data$Crime))^2)

rs <- 1- SSres/SStotal

rs
```

```
## [1] 0.39845
```

Now we can see that using all predictors in a linear model results in an r^2 (.398) that is lower than a model using PCA and 7 principal components (.53).