Electric Motor Rotor Temperature Prediction

with Multiple Linear Regression, Principal Component Regression (PCA Regression)

June 2020, slightly modified

1. Describe the dataset

https://www.kaggle.com/wkirgsn/electric-motor-temperature

The data set comprises several sensor data collected from a permanent magnet synchronous motor (PMSM) deployed on a test bench. The PMSM represents a german OEM's prototype model. Test bench measurements were collected by the LEA department at Paderborn University.

Variables:

- u_q: q component of Voltage measured in Volts
- u_d: d component of Voltage measured in Volts
- i_q: q component of Current measured in Amps
- i_d: d component of Current measured in Amps
- ambient: ambient temperature around the stator in °C (measured by a thermal sensor fixed close to stator)
- coolant: motor coolant (water in this case) temperature of the motor in °C (measured by a fixed thermal sensor at coolant outlet)
- motor speed: ambient temperature around the stator in °C (measured by a fixed thermal sensor)
- stator_tooth: stator tooth temperature in °C
- stator_winding: stator winding temperature in °C
- stator_yoke: stator yoke temperature in °C
- pm: permanent magnet tooth temperature in °C
- profile_id: id of the measurement session

Target:

- The most interesting target features are **rotor temperature ("pm"), stator temperatures ("stator_*") and torque**. Especially rotor temperature and torque are not reliably and economically measurable in a commercial vehicle. Being able to have strong estimators for the rotor temperature helps the automotive industry to manufacture motors with less material and enables control strategies to utilize the motor to its maximum capability.
- Therefore, the target for today is to predict the **rotor temperature**("pm") of a given motor.

2. Load the dataset

In [1]:
```
df<-read.csv("measures_v2.csv",header=T)
head(df,3)
```

A data.frame: 3 × 13

	u_q	coolant	stator_winding	u_d	stator_tooth	motor_speed	i_d	i_q	p
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl
1	-0.4506815	18.80517	19.08667	-0.3500546	18.29322	0.0028655678	0.0044191368	0.0003281022	24.5542

	u_q	coolant	stator_winding	u_d	stator_tooth	motor_speed	i_d	i_q	pr
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl
2	-0.3257370	18.81857	19.09239	-0.3058030	18.29481	0.0002567817	0.0006058724	-0.0007853527	24.5380
3	-0.4408640	18.82877	19.08938	-0.3725026	18.29409	0.0023549714	0.0012895871	0.0003864682	24.5446

In [2]:

```
str(df)
```

```
'data.frame':   1330816 obs. of  13 variables:
 $ u_q           : num  -0.451 -0.326 -0.441 -0.327 -0.471 ...
 $ coolant       : num  18.8 18.8 18.8 18.8 18.9 ...
 $ stator_winding: num  19.1 19.1 19.1 19.1 19.1 ...
 $ u_d           : num  -0.35 -0.306 -0.373 -0.316 -0.332 ...
 $ stator_tooth  : num  18.3 18.3 18.3 18.3 18.3 ...
 $ motor_speed   : num  0.002866 0.000257 0.002355 0.006105 0.003133 ...
 $ i_d           : num  4.42e-03 6.06e-04 1.29e-03 2.56e-05 -6.43e-02 ...
 $ i_q           : num  0.000328 -0.000785 0.000386 0.002046 0.037184 ...
 $ pm            : num  24.6 24.5 24.5 24.6 24.6 ...
 $ stator_yoke   : num  18.3 18.3 18.3 18.3 18.3 ...
 $ ambient       : num  19.9 19.9 19.9 19.9 19.9 ...
 $ torque        : num  0.187 0.245 0.177 0.238 0.208 ...
 $ profile_id    : int  17 17 17 17 17 17 17 17 17 17 ...
```

3. Exploration and initial thoughts

In [3]:

```
#for missing values
sum(is.na(df))
```

0

In [4]:

```
#drop the column profile_id
df <- subset(df, select = -c(profile_id))
head(df,2)
```

A data.frame: 2 × 12

	u_q	coolant	stator_winding	u_d	stator_tooth	motor_speed	i_d	i_q	pr
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl
1	-0.4506815	18.80517	19.08667	-0.3500546	18.29322	0.0028655678	0.0044191368	0.0003281022	24.5542
2	-0.3257370	18.81857	19.09239	-0.3058030	18.29481	0.0002567817	0.0006058724	-0.0007853527	24.5380

In [5]:

```
cor(df)
```

A matrix: 12 × 12 of type dbl

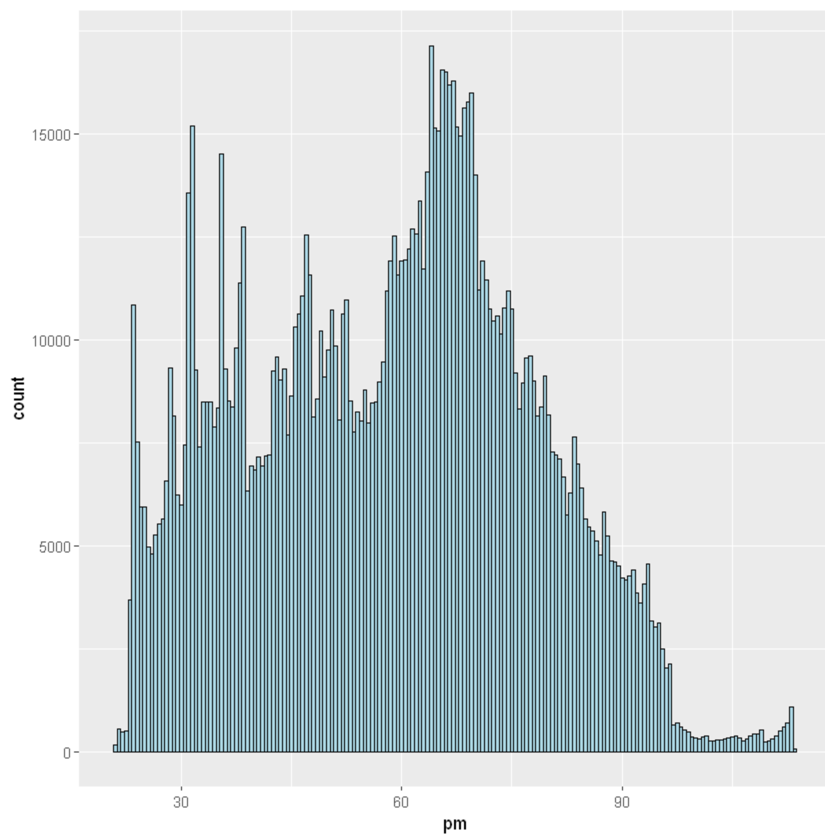
	u_q	coolant	stator_winding	u_d	stator_tooth	motor_speed	i_d	
u_q	1.000000000	0.05172100	0.05060983	0.004701759	0.10437324	0.68355601	-0.10035698	-0.1
coolant	0.051720996	1.00000000	0.50483519	0.195517009	0.67497359	0.01187233	0.07486480	-0.2
stator_winding	0.050609826	0.50483519	1.00000000	-0.234950204	0.97013472	0.43203390	-0.62436963	0.0
u_d	0.004701759	0.19551701	-0.23495020	1.000000000	-0.14274949	-0.28847200	0.44833078	-0.7
stator_tooth	0.104373235	0.67497359	0.97013472	-0.142749488	1.00000000	0.39843075	-0.48706209	-0.0
motor_speed	0.683556010	0.01187233	0.43203390	-0.288471996	0.39843075	1.00000000	-0.70060850	-0.0

	u_q	coolant	stator_winding	u_d	stator_tooth	motor_speed	i_d	
i_d	-0.100356980	0.07486480	-0.62436963	0.448330776	-0.48706209	-0.70060850	1.00000000	-0.2
i_q	-0.124589132	-0.25638897	0.06561716	-0.723068930	-0.04229356	-0.06888053	-0.23134438	1.0
pm	0.122364640	0.46711732	0.79589251	-0.172030583	0.83208390	0.45894708	-0.42773622	-0.1
stator_yoke	0.090992024	0.86075028	0.86026836	-0.008097952	0.95311453	0.25578987	-0.27800476	-0.1
ambient	0.150264303	0.52596328	0.33320831	0.203647334	0.44346971	0.11823214	0.01639681	-0.3
torque	-0.136214907	-0.25798240	0.09550988	-0.753779010	-0.01841303	-0.04390256	-0.27409728	0.9

The correlation coefficient matrix shows pairwise correlation among the variables. It can be seen that although the correlation between most of the 12 variables we selected is very weak, the correlation coefficient between i_q and torque has reached 0.996, and the abs of correlation coefficient between i_d and motor speed has reached 0.7. There may exist collinearity between the variables.

In [13]:

```
#distribution for the pm
ggplot(df,aes(x=pm))+geom_histogram(binwidth=0.5,fill="lightblue",colour="black")
```



In [18]:

```
paste(min(df$pm),max(df$pm))
```

'20.8569564819336 113.606628417969'

The density curve of pm is a little right skewed, ranges from 20 to 113.

4. Multiple Linear Regression

Initial attempt

In [20]:

```
model1=lm(pm~.,data=df)
model1
```

```
summary(model1)
```

```
Call:
lm(formula = pm ~ ., data = df)

Coefficients:
   (Intercept)              u_q          coolant  stator_winding              u_d  
    -18.592273        -0.131639        -0.185264       -1.599567        -0.019252  
  stator_tooth      motor_speed              i_d             i_q      stator_yoke  
      3.966738         0.002941         0.042824       -0.010012        -1.571968  
       ambient           torque  
      1.766692         0.005184  
Call:
lm(formula = pm ~ ., data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-45.562  -4.563  -0.453   3.914  40.015 

Coefficients:
                 Estimate Std. Error  t value Pr(>|t|)    
(Intercept)    -1.859e+01  9.405e-02 -197.682  < 2e-16 ***
u_q            -1.316e-01  3.289e-04 -400.283  < 2e-16 ***
coolant        -1.853e-01  1.962e-03  -94.450  < 2e-16 ***
stator_winding -1.600e+00  3.125e-03 -511.813  < 2e-16 ***
u_d            -1.925e-02  2.179e-04  -88.340  < 2e-16 ***
stator_tooth    3.967e+00  7.657e-03  518.050  < 2e-16 ***
motor_speed     2.941e-03  1.316e-05  223.568  < 2e-16 ***
i_d             4.282e-02  3.122e-04  137.179  < 2e-16 ***
i_q            -1.001e-02  1.092e-03   -9.165  < 2e-16 ***
stator_yoke    -1.572e+00  6.824e-03 -230.370  < 2e-16 ***
ambient         1.767e+00  4.156e-03  425.069  < 2e-16 ***
torque          5.184e-03  1.391e-03    3.727 0.000194 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.237 on 1330804 degrees of freedom
Multiple R-squared:  0.8549,	Adjusted R-squared:  0.8549 
F-statistic: 7.131e+05 on 11 and 1330804 DF,  p-value: < 2.2e-16
```

It shows all variables are significant in this model. The p-value for the model F-test is very small, which justifies the use of a linear model here.

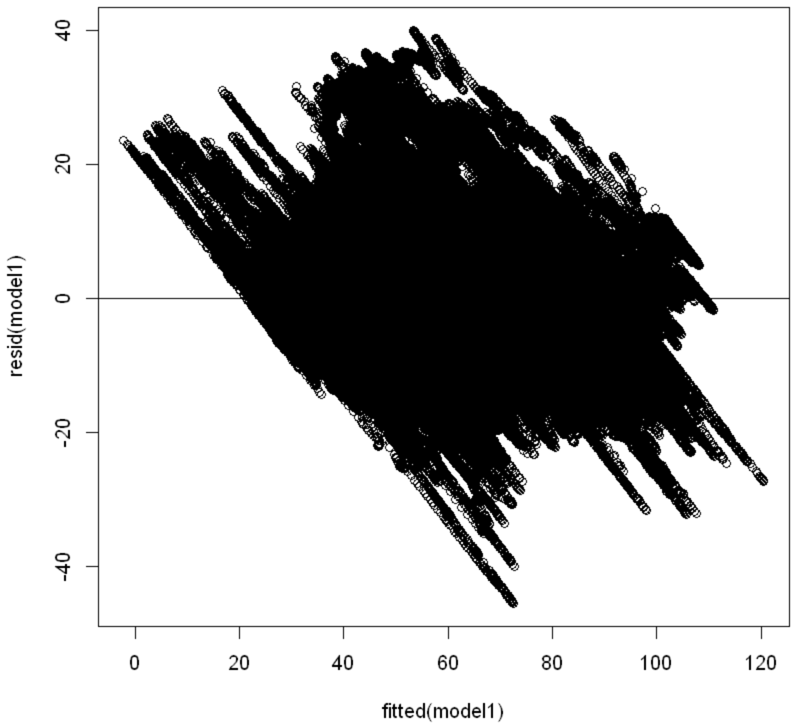
The adjusted R-squared value is 0.8549, which means the predictors explain over 85% of the variation in rotor temperature, which is a good result.

Model diagnostic

(1) Residual Plot to identify non-linearity/heteroscedasticity.

In [26]:

```
plot(fitted(model1),resid(model1))
abline(h=0)
```



(2) Correlation of error terms: D-W test & paired scatter plot of residual series.

In [29]:

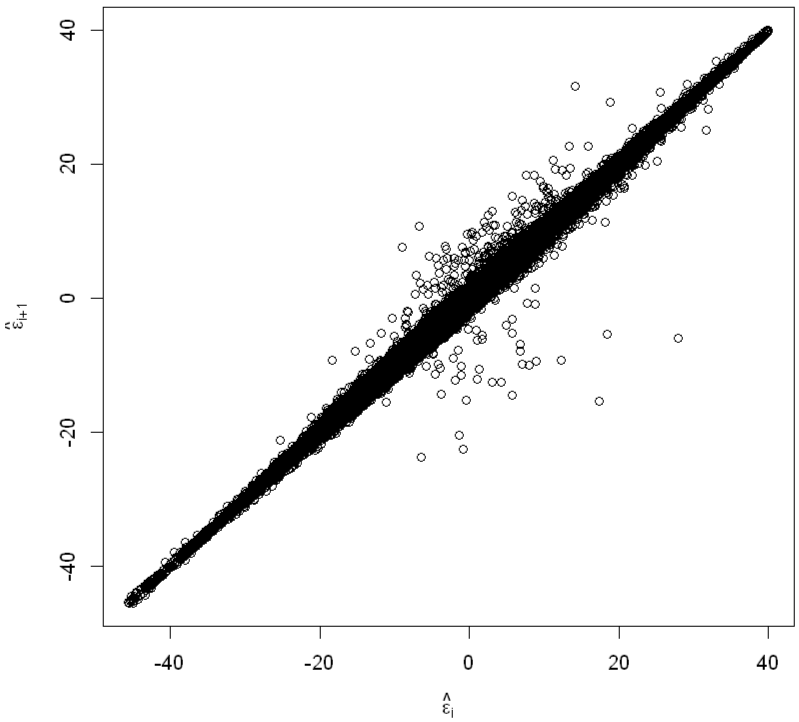
```
#D-W test
require(lmtest)
dwtest(pm~.,data=df)
```

```
        Durbin-Watson test

data:  pm ~ .
DW = 0.0015506, p-value < 2.2e-16
alternative hypothesis: true autocorrelation is greater than 0
```

In [30]:

```
#residual series
n=length(residuals(model1))
plot(tail(residuals(model1),n-1)~head(residuals(model1),n-1),xlab=expression(hat(epsilon)
summary(lm(tail(residuals(model1),n-1)~head(residuals(model1),n-1)))
```

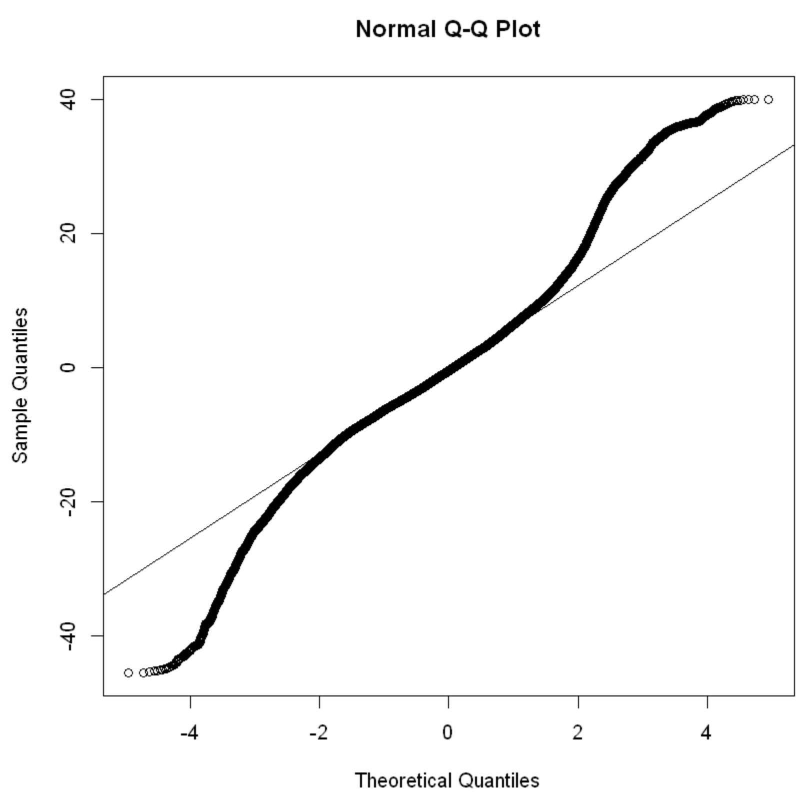
```
Call:
lm(formula = tail(residuals(model1), n - 1) ~ head(residuals(model1), 
    n - 1))

Residuals:
    Min      1Q  Median      3Q     Max 
-33.847  -0.102  -0.002   0.100  17.504 

Coefficients:
                                Estimate Std. Error   t value Pr(>|t|)    
(Intercept)                    9.735e-06  2.470e-04     0.039    0.969    
head(residuals(model1), n - 1) 9.992e-01  3.413e-05 29279.078   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2849 on 1330813 degrees of freedom
Multiple R-squared:  0.9985,    Adjusted R-squared:  0.9985 
F-statistic: 8.573e+08 on 1 and 1330813 DF,  p-value: < 2.2e-16
```



The residuals are highly correlated... It may because the measures are recorded in a time sequence.

(3) Normality of residuals

In [31]:
```
qqnorm(resid(model1))
qqline(resid(model1))
```



The residuals may not satisfy the normality assumption...

(4) Outliers(yi is far from the predicted value)

```
In [34]: stud=rstudent(model1)#studendized residual
         stud[which.max(abs(stud))]#the max value
         stud[abs(stud)>abs(qt(0.05/(1330816*2),1330804))]#outliers
         #abs(qt(0.05/(1330816*2),1330804): the critical value of Bonferroni test: 1330816: sample
```

606281: -6.29588472013882

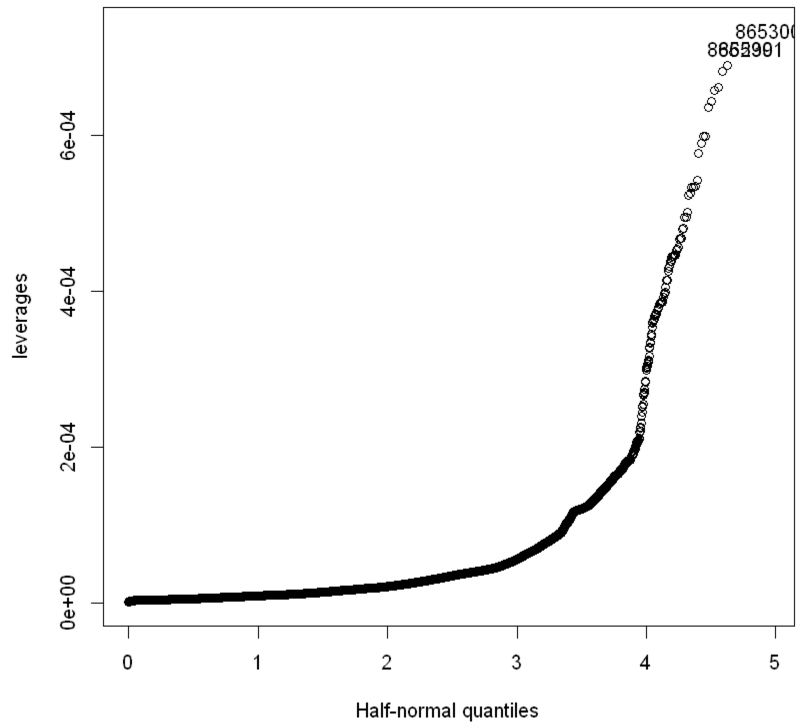
252299: 5.50887373968047 **252300:** 5.52377180153 **252301:** 5.52595141168213 **252302:** 5.52938004732797 **252303:** 5.52339071369702 **252304:** 5.51460370193626 **252305:** 5.50585368558079 **606245:** -5.52335959273835 **606246:** -5.53743265860978 **606247:** -5.55638436970531 **606248:** -5.58054276653191 **606249:** -5.57071234400933 **606250:** -5.55570709256389 **606251:** -5.55241806286432 **606252:** -5.5801386410793 **606253:** -5.60498614389451 **606254:** -5.62248922420427 **606255:** -5.6597686690229 **606256:** -5.70985256915616 **606257:** -5.75720404230742 **606258:** -5.74826530274597 **606259:** -5.74343533498571 **606260:** -5.75289453633535 **606261:** -5.73385598734988 **606262:** -5.71755023569418 **606263:** -5.72851724262032 **606264:** -5.72824845842823 **606265:** -5.73394590643329 **606266:** -5.747460132154 **606267:** -5.76213272661388 **606268:** -5.79103289844231 **606269:** -5.81352293111764 **606270:** -5.83038701974874 **606271:** -5.86464708397679 **606272:** -5.91023840626215 **606273:** -5.91468334708973 **606274:** -5.87791731277027 **606275:** -5.85937911482168 **606276:** -5.87494339399717 **606277:** -5.98871609180371 **606278:** -6.11734057881365 **606279:** -6.2178807983757 **606280:** -6.2929595292655 **606281:** -6.29588472013882 **606282:** -6.27346852783353 **606283:** -6.25279343919459 **606284:** -6.23652234881465 **606285:** -6.20033851565941 **606286:** -6.15041693346629 **606287:** -6.08592266758685 **606288:** -6.01333960005301 **606289:** -5.94594166430456 **606290:** -5.88190188723579 **606291:** -5.82261478846628 **606292:** -5.77718631315851 **606293:** -5.7454469849415 **606294:** -5.73825178702402 **606295:** -5.73505983649651 **606296:** -5.74517082054795 **606297:** -5.77349537014241 **606298:** -5.80347822665353 **606299:** -5.83736268660171 **606300:** -5.86416673898432 **606301:** -5.91598482847699 **606302:** -5.97489677842312 **606303:** -6.03325148582463 **606304:** -6.0843376273247 **606305:** -6.13342898667953 **606306:** -6.16739673845106 **606307:** -6.1980353505169 **606308:** -6.22050393637589 **606309:** -6.24151108157401 **606310:** -6.15022439774967 **606311:** -6.06311453433666 **606312:** -5.9999645275572 **606313:** -5.96540345170887 **606314:** -5.96998618850871 **606315:** -5.97632268635861 **606316:** -5.96330728985304 **606317:** -5.9486224915011 **606318:** -5.90073498941939 **606319:** -5.84404860759224 **606320:** -5.80106236062509 **606321:** -5.76866387070777 **606322:** -5.72373745461788 **606323:** -5.69059595179595 **606324:** -5.67994716268437 **606325:** -5.6820113180848 **606326:** -5.67382883091595 **606327:** -5.67865610364727 **606328:** -5.68029818428397 **606329:** -5.6930068983953 **606330:** -5.7122792250661 **606331:** -5.72493636581381 **606332:** -5.71441603439353 **606333:** -5.69338524290191 **606334:** -5.66682024192866 **606335:** -5.61385214953006 **609327:** -5.51940959143588 **609328:** -5.53681120758763 **609329:** -5.52656587842656

The outliers are continuously: 252299-252305, 606245-606335, 609327-609329. I will recommend check if there's some measure error or something else with those samples. With no additional information provided, I will just keep those outliers.

(5) High-leverage points(unusual xi values, tend to have a higher impact on estimated regression line, not necessarily an influential point.

```
In [ ]: library(faraway)
        hatv=hatvalues(model1)
```

```
In [40]: halfnorm(hatv,3,ylab="leverages")
```

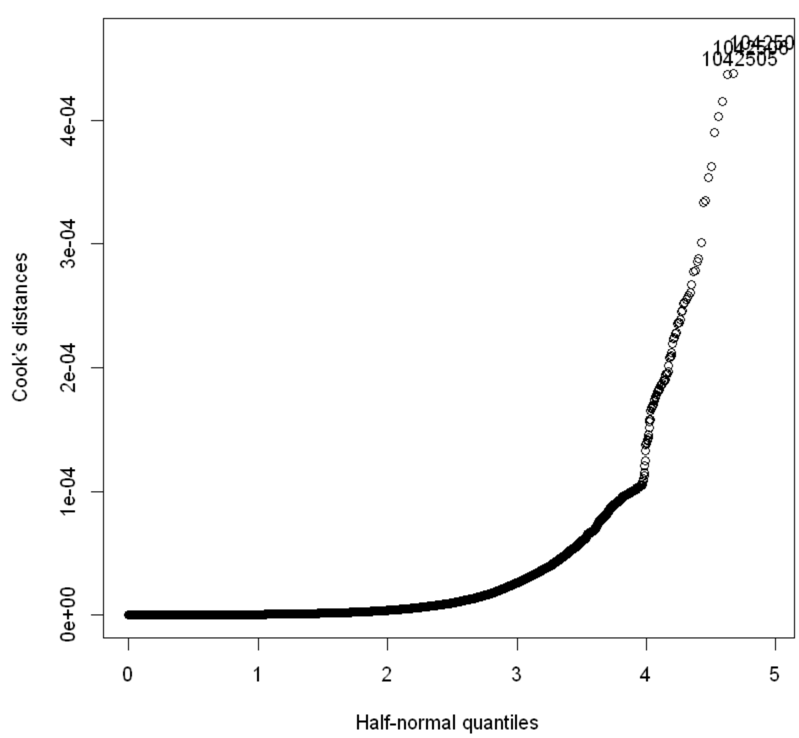


(6) Influential points: observations having a relatively large effect on the regression model's predictions

In [41]:

```
cook=cooks.distance(model1)
head(sort(cook))
halfnorm(cook,3,ylab="Cook's distances")
```

283720: 1.31736754180472e-20 **1048974:** 8.08961295260762e-20 **157119:** 2.00301841732884e-19 **927761:** 3.72650111828008e-19 **881375:** 2.2769042897166e-18 **1136521:** 3.32186025895295e-18



(7) Collinearity: two or more predictor variables are closely related to one another.

In [42]:

```
library(faraway)
x=model.matrix(model1)[,-1]
e=eigen(t(x)%*%x)
e$val
sqrt(e$val[1]/e$val)
require(faraway)
vif(x)
```

11083808494529.2 · 25125620140.2722 · 6835331966.32689 · 3411148252.04839 · 1613920530.16857 · 376335260.029233 · 186457890.748006 · 75531388.5265116 · 16893229.5093236 · 4880515.12865432 · 492590.361433562

1 · 21.0032311454082 · 40.2684303234793 · 57.0025401787884 · 82.8711629908281 · 171.615713954457 · 243.811464520264 · 383.072320458402 · 810.00586884545 · 1506.99452693885 · 4743.52891102688

u_q: 5.36233256846282 **coolant:** 46.4025855659739 **stator_winding:** 204.034827432902 **u_d:** 4.8039400273843 **stator_tooth:** 784.8343685044 **motor_speed:** 15.2077050806087 **i_d:** 10.4409308001865 **i_q:** 257.681067035605 **stator_yoke:** 472.833428258411 **ambient:** 1.6342034805265 **torque:** 292.460138747691

5 variables have VIF>100, which shows that there exists collinearity problem in the model. In the face of collinearity, we may decide to eliminate some variables in the model.

Variable Selection

Some measures for the "best model":

In [45]:

```
#subset of predictors by # of predictors that produce **min RSS**
require(leaps)
b=regsubsets(pm~.,data=df)
rs=summary(b)
print(rs$which)
```

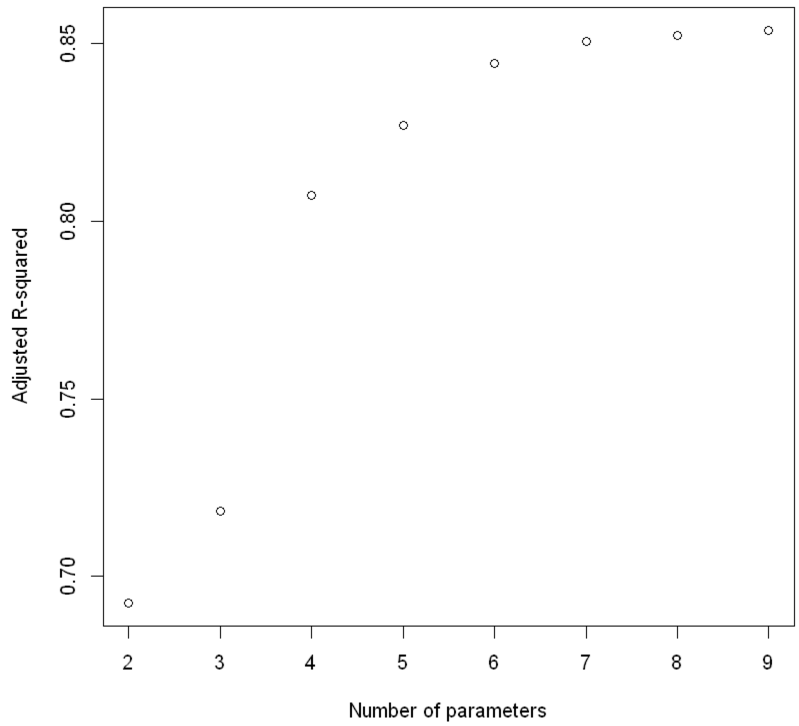
```
  (Intercept)   u_q coolant stator_winding   u_d stator_tooth motor_speed   i_d
1        TRUE FALSE   FALSE          FALSE FALSE         TRUE       FALSE FALSE
2        TRUE FALSE   FALSE          FALSE FALSE         TRUE       FALSE FALSE
3        TRUE FALSE   FALSE           TRUE FALSE         TRUE       FALSE FALSE
4        TRUE  TRUE   FALSE           TRUE FALSE         TRUE       FALSE FALSE
5        TRUE  TRUE   FALSE           TRUE FALSE         TRUE       FALSE FALSE
6        TRUE  TRUE   FALSE           TRUE FALSE         TRUE        TRUE FALSE
7        TRUE  TRUE   FALSE           TRUE FALSE         TRUE        TRUE  TRUE
8        TRUE  TRUE   FALSE           TRUE  TRUE         TRUE        TRUE  TRUE
    i_q stator_yoke ambient torque
1 FALSE       FALSE   FALSE  FALSE
2 FALSE       FALSE    TRUE  FALSE
3 FALSE        TRUE   FALSE  FALSE
4 FALSE        TRUE   FALSE  FALSE
5 FALSE        TRUE    TRUE  FALSE
6 FALSE        TRUE    TRUE  FALSE
7 FALSE        TRUE    TRUE  FALSE
8 FALSE        TRUE    TRUE  FALSE
```

Adjusted R2:

In [55]:

```
##plot Adjusted R-squared
plot(2:9,rs$adjr2,xlab="Number of parameters",ylab="Adjusted R-squared")
##find the Number of parameters that has max R-squared
print(which.max(rs$adjr2))
```

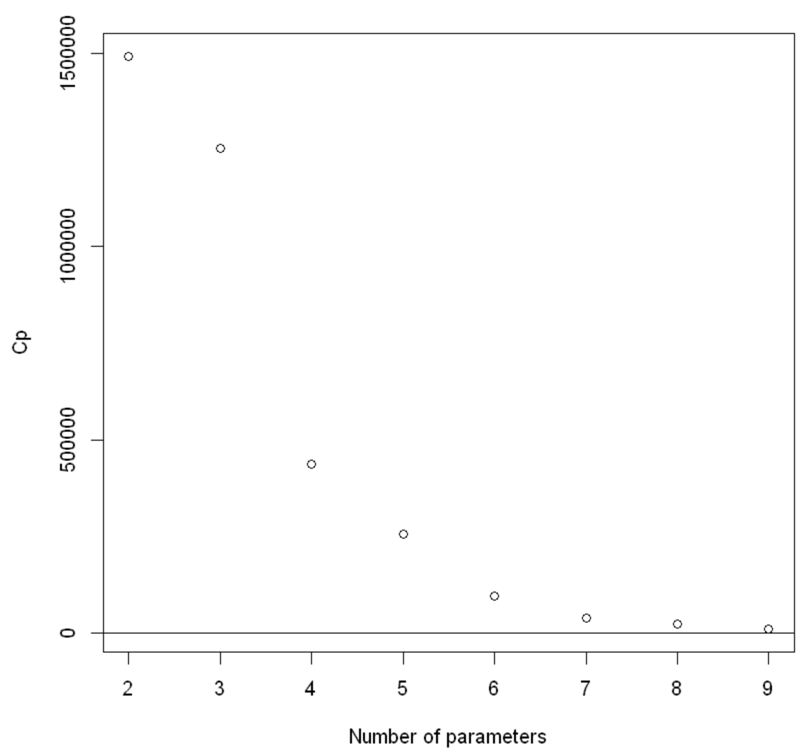
```
[1] 8
```



8/9 parameters according to Ra^2.

Mallow's CP:

In [57]:
```
require(leaps)
b=regsubsets(pm~.,data=df)
rs=summary(b)
plot(2:9,rs$cp,xlab="Number of parameters",ylab="Cp")
abline(0,1)
```

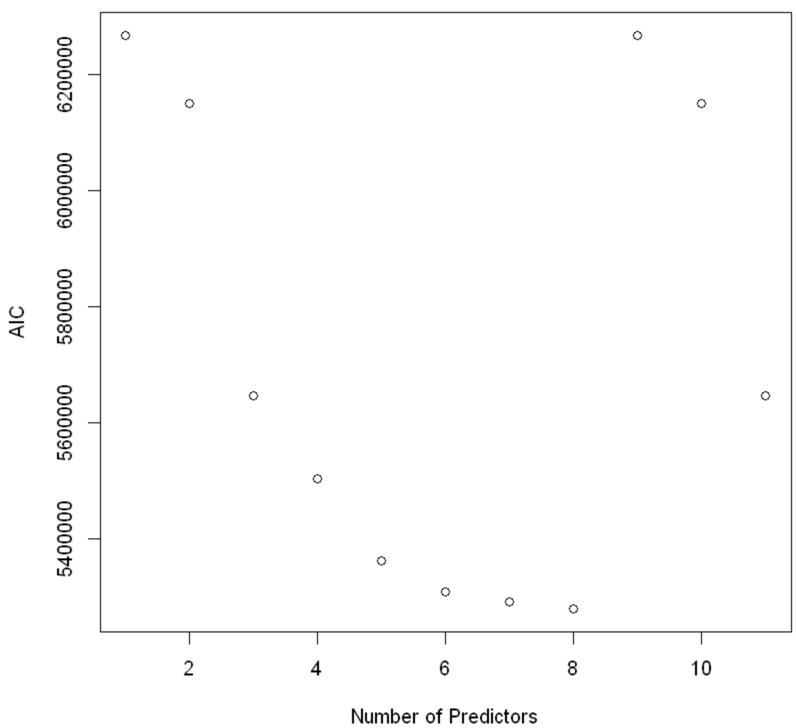


9 parameters according to the Cp.

AIC:

In [67]:
```
require(leaps)
AIC=1330816*log(rs$rss/1330816)+(2:12)*2
par(mfrow=c(1,1))
plot(AIC~I(1:11),ylab="AIC",xlab="Number of Predictors")
```

```
Warning message in 1330816 * log(rs$rss/1330816) + (2:12) * 2:
"longer object length is not a multiple of shorter object length"
```



8 predictors according to the AIC.

Therefore, based on these measures, I will choose to keep 8 predictors in the model. And based on the provided model selection matrix which provides min RSS, the predictors would be: u_q, stator_winding, u_d, stator_tooth, motor_speed, i_d, stator_yoke, ambient.

Model 2 (after variable selection)

In [69]:
```
model2=lm(pm~u_q+stator_winding+u_d+stator_tooth+motor_speed+i_d+stator_yoke+ambient,data=
summary(model2)
```

```
Call:
lm(formula = pm ~ u_q + stator_winding + u_d + stator_tooth + 
    motor_speed + i_d + stator_yoke + ambient, data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-45.317  -4.606  -0.474   3.960  41.687 

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)    
(Intercept)    -1.859e+01  9.366e-02  -198.4   <2e-16 ***
u_q            -1.308e-01  2.921e-04  -447.8   <2e-16 ***
stator_winding -1.695e+00  2.926e-03  -579.4   <2e-16 ***
u_d            -1.299e-02  1.161e-04  -111.9   <2e-16 ***
```

```
stator_tooth     4.472e+00  5.721e-03   781.6   <2e-16 ***
motor_speed      2.816e-03  1.156e-05   243.6   <2e-16 ***
i_d              4.371e-02  3.000e-04   145.7   <2e-16 ***
stator_yoke     -2.173e+00  2.866e-03  -758.2   <2e-16 ***
ambient          1.771e+00  4.155e-03   426.2   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.264 on 1330807 degrees of freedom
Multiple R-squared:  0.8538,	Adjusted R-squared:  0.8538 
F-statistic: 9.718e+05 on 8 and 1330807 DF,  p-value: < 2.2e-16
```

Check the collinearity that lies in the model:

In [70]:

```
library(faraway)
x=model.matrix(model2)[,-1]
e=eigen(t(x)%*%x)
sqrt(e$val[1]/e$val)
require(faraway)
vif(x)
```

1 · 40.0611644115625 · 46.0980172643339 · 65.6877776128029 · 142.672953046082 · 381.353130674305 · 515.029339547659 · 3132.47395772863

u_q: 4.19886109145618 **stator_winding:** 177.449369925511 **u_d:** 1.35314235384114 **stator_tooth:** 434.815412716237 **motor_speed:** 11.6542325182297 **i_d:** 9.57014892192291 **stator_yoke:** 82.7864124405666 **ambient:** 1.62118527081255

There's still collinearity that exists in the model, especially for **stator_tooth**. I would remove it and check again.

Model 3

In [71]:

```
model3=lm(pm~u_q+stator_winding+u_d+motor_speed+i_d+stator_yoke+ambient,data=df)
summary(model3)
```

```
Call:
lm(formula = pm ~ u_q + stator_winding + u_d + motor_speed + 
    i_d + stator_yoke + ambient, data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-56.534  -5.826  -0.429   5.344  41.423 

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)    
(Intercept)    -3.222e+01  1.112e-01 -289.86   <2e-16 ***
u_q            -1.816e-01  3.440e-04 -527.87   <2e-16 ***
stator_winding  5.192e-01  8.820e-04  588.66   <2e-16 ***
u_d            -2.216e-02  1.395e-04 -158.86   <2e-16 ***
motor_speed     7.909e-03  1.153e-05  685.71   <2e-16 ***
i_d             1.686e-01  3.068e-04  549.48   <2e-16 ***
stator_yoke    -4.837e-02  1.097e-03  -44.08   <2e-16 ***
ambient         2.527e+00  4.881e-03  517.68   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.775 on 1330808 degrees of freedom
Multiple R-squared:  0.7867,	Adjusted R-squared:  0.7867 
F-statistic: 7.014e+05 on 7 and 1330808 DF,  p-value: < 2.2e-16
```

In [72]:

```
library(faraway)
x=model.matrix(model3)[,-1]
```

```
e=eigen(t(x)%*%x)
sqrt(e$val[1]/e$val)
require(faraway)
vif(x)
```

1 · 42.0379964785745 · 51.5672273295648 · 66.3755534379605 · 145.437848864981 · 382.657048026741 · 515.729394898524

u_q: 3.99121952220891 **stator_winding:** 11.0542784645857 **u_d:** 1.33932149262132 **motor_speed:** 7.95145231069437 **i_d:** 6.85752973127513 **stator_yoke:** 8.31733705408081 **ambient:** 1.53334557264949

There's no collinearity concern now, but the R-squared value has dropped a lot(below 80%)! emmm. It seems that the stator_tooth plays a rather important role with the response variable here...Okay I will put it back and remove the variable with the second biggest VIF, which is stator_winding.

Model 4 - the final model for the MLR

In [73]:

```
model4=lm(pm~u_q+u_d+motor_speed+i_d+stator_yoke+ambient+stator_tooth,data=df)
summary(model4)
```

```
Call:
lm(formula = pm ~ u_q + u_d + motor_speed + i_d + stator_yoke +
    ambient + stator_tooth, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-54.433  -5.249  -0.531   4.833  40.175

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)  -3.029e+01  1.023e-01  -295.9   <2e-16 ***
u_q          -1.681e-01  3.188e-04  -527.4   <2e-16 ***
u_d          -1.748e-02  1.296e-04  -134.8   <2e-16 ***
motor_speed   6.644e-03  1.062e-05   625.9   <2e-16 ***
i_d           1.547e-01  2.583e-04   599.0   <2e-16 ***
stator_yoke  -7.664e-01  1.705e-03  -449.6   <2e-16 ***
ambient       2.388e+00  4.495e-03   531.2   <2e-16 ***
stator_tooth  1.262e+00  1.598e-03   789.6   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.129 on 1330808 degrees of freedom
Multiple R-squared:  0.817,	Adjusted R-squared:  0.817
F-statistic: 8.486e+05 on 7 and 1330808 DF,  p-value: < 2.2e-16
```

In [74]:

```
library(faraway)
x=model.matrix(model4)[,-1]
e=eigen(t(x)%*%x)
sqrt(e$val[1]/e$val)
require(faraway)
vif(x)
```

1 · 42.8732981028739 · 54.0745341912743 · 65.8718088849116 · 143.750299868054 · 393.659918033344 · 939.857173868918

u_q: 3.99466387090316 **u_d:** 1.34712531969276 **motor_speed:** 7.84773308845241 **i_d:** 5.66594010045207 **stator_yoke:** 23.3885815885312 **ambient:** 1.51488263054324 **stator_tooth:** 27.086997575009

Now we got a model with lower R-squared value, but it is acceptable since the collinearity has been eliminated here.

Cross-validation to check the model's performance in prediction

https://www.rdocumentation.org/packages/lmvar/versions/1.5.2/topics/cv.lm

In [90]:

```
library(DAAG)
```

In [97]:

```
MLR<-lm(pm~u_q+u_d+motor_speed+i_d+stator_yoke+ambient+stator_tooth,data=df, x = TRUE, y =
```

In [98]:

```
cv.lm(df=df,MLR)
```

```
Mean absolute error         :  6.202322
Sample standard deviation   :  0.02414958

Mean squared error          :  66.08449
Sample standard deviation   :  0.5892648

Root mean squared error     :  8.129164
Sample standard deviation   :  0.03626784
```

5. Principal Component Regression

Since there are collinearity exists in the predictors, I will also try a principal component regression.

In [128…

```
library(caret)
```

In [129…

```
#first we need to split the dataset into training and testing sets, because we don't have
set.seed(101)
sample <- sample.int(n = nrow(df), size = floor(.75*nrow(df)), replace = F)
train <- df[sample, ]
test  <- df[-sample, ]
#scale the data respectively(standardized: mean=0, var=1)
normParam <- preProcess(train)
train <- predict(normParam, train)
test <- predict(normParam, test)
```

In [131…

```
X_train <- subset(train, select = -c(pm))
X_test <- subset(test, select = -c(pm))
y_train <- subset(train, select = c(pm))
y_test <- subset(test, select = c(pm))
```

In [136…

```
pca <- prcomp(X_train)
```

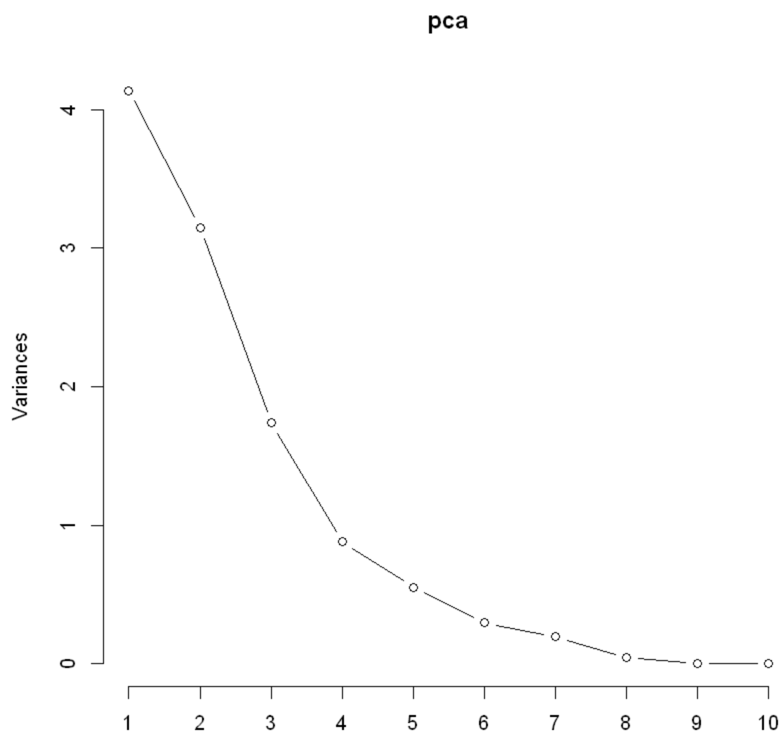
In [137…

```
summary(pca)
```

```
Importance of components:
                         PC1   PC2    PC3     PC4     PC5     PC6     PC7
Standard deviation     2.034 1.774 1.3181 0.93851 0.74418 0.54419 0.44388
Proportion of Variance 0.376 0.286 0.1579 0.08007 0.05035 0.02692 0.01791
Cumulative Proportion  0.376 0.662 0.8199 0.90000 0.95035 0.97727 0.99518
                           PC8     PC9    PC10    PC11
Standard deviation     0.20956 0.08070 0.04277 0.02713
Proportion of Variance 0.00399 0.00059 0.00017 0.00007
Cumulative Proportion  0.99917 0.99977 0.99993 1.00000
```

In [138…

```
plot(pca, type="lines")
```



According to the scree plot, we need maybe 5 (or maybe 8?) variables. Using the Kaiser criterion we will retain 3 components, which can retain approximately 82% of the variability in the data. I'll retain the first 4 components to keep cumulative proportion >90%.

the scores of the original observations on those 4 components:

In [142…

```
scr <- pca$x[,1:4]
```

run PCReg for pm as a response...

In [145…

```
pcregmod <- lm(pm ~ scr, data=train)
summary(pcregmod)
```

```
Call:
lm(formula = pm ~ scr, data = train)

Residuals:
     Min       1Q   Median       3Q      Max 
-2.45900 -0.35104 -0.04349  0.31483  2.57187 

Coefficients:
              Estimate Std. Error  t value Pr(>|t|)    
(Intercept)  9.120e-17  5.584e-04    0.000        1    
scrPC1       4.055e-01  2.746e-04 1476.739   <2e-16 ***
scrPC2      -2.128e-02  3.148e-04  -67.574   <2e-16 ***
scrPC3      -3.811e-03  4.237e-04   -8.995   <2e-16 ***
scrPC4       9.027e-02  5.950e-04  151.714   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5579 on 998107 degrees of freedom
Multiple R-squared:  0.6887,	Adjusted R-squared:  0.6887 
F-statistic: 5.521e+05 on 4 and 998107 DF,  p-value: < 2.2e-16
```

Only 68% of the variation in y is explained. So I will use more PCs. (8 pcs)

In [146...

```
scr2 <- pca$x[,1:8]
pcregmod <- lm(pm ~ scr2, data=train)
summary(pcregmod)
```

```
Call:
lm(formula = pm ~ scr2, data = train)

Residuals:
    Min      1Q  Median      3Q     Max 
-2.9899 -0.2621 -0.0260  0.2475  2.1204 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  7.635e-17  4.299e-04    0.00        1    
scr2PC1      4.055e-01  2.114e-04 1918.46   <2e-16 ***
scr2PC2     -2.128e-02  2.423e-04  -87.79   <2e-16 ***
scr2PC3     -3.811e-03  3.261e-04  -11.69   <2e-16 ***
scr2PC4      9.027e-02  4.580e-04  197.09   <2e-16 ***
scr2PC5     -2.053e-01  5.776e-04 -355.36   <2e-16 ***
scr2PC6     -1.773e-01  7.899e-04 -224.50   <2e-16 ***
scr2PC7      5.734e-01  9.684e-04  592.13   <2e-16 ***
scr2PC8      8.182e-01  2.051e-03  398.89   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4295 on 998103 degrees of freedom
Multiple R-squared:  0.8156,	Adjusted R-squared:  0.8156 
F-statistic: 5.517e+05 on 8 and 998103 DF,  p-value: < 2.2e-16
```

Now 80% of the variation in y is explained(maybe I'll try PLS later). I will see it's performance on the test set.

In [149...

```
#pca$rotation
#Rotate standardized test data to the same space as train data
test_scores = predict(pca, test) #test = test %*% pr$rotation
```

In [156...

```
model_test<-lm(test$pm ~ test_scores)
```

In [159...

```
#calculate MSE
mean(model_test$residuals^2) #mean((predict(lm(test$pm ~ test_scores))-test$pm)^2)
```

0.145357131893531

Maybe the MLR also needs to perform on scaled data(not essential for the model, but just for compare the MSE here). :)