ISyE 6404 (EP-2): K-M Estimation, Kernel Regression and Spline

Yuan Gao, Kevin Lee, Akshay Govindaraj
Yijun (Emma) Wan, Peter Williams, Ruixuan Zhang
ygao390, kylee20, ywan40, agovindaraj6, pwilliams60, rzhang438 | @gatech.edu

2018-10-22

Contents

Workload Distribution

Workload and description of tasks and the distribution of work (%) by team member for this project:

Team Member	Task Description
Yuan Gao	TBD
Kevin Lee	TDB
Akshay Govindaraj	TBD
Yijun (Emma) Wan	TBD
Peter Williams	TBD
Ruixuan Zhang	TBD

1. K-M Estimation (25%):

Locate a data set with right-censoring (in Type-I Censoring) in the field of your interest, e.g., eCommerce, medical study, drug development, supply-chain/logistics operations, for applying the K-M Estimator to estimate the survival function with pointwise confidence intervals.

For this exercise, we located a dataset, that consists of measures on 69 different patients who received a heart transplant, taken from the first edition of the text *The Statistical Analysis of Failure Time Data* by Kalbfleisch and Prentice, Appendix I (230-232), published originally in 1980, which can also be found via the following link on the Carnegie Mellon statistics site: http://lib.stat.cmu.edu/datasets/stanford, and has three columns with the following measures:

- Age: Age of patient in years at the time of heart transplant in years
- Status: Survival status (1=dead, 0=alive)
- Days: Survival time in days after transplant (in days)

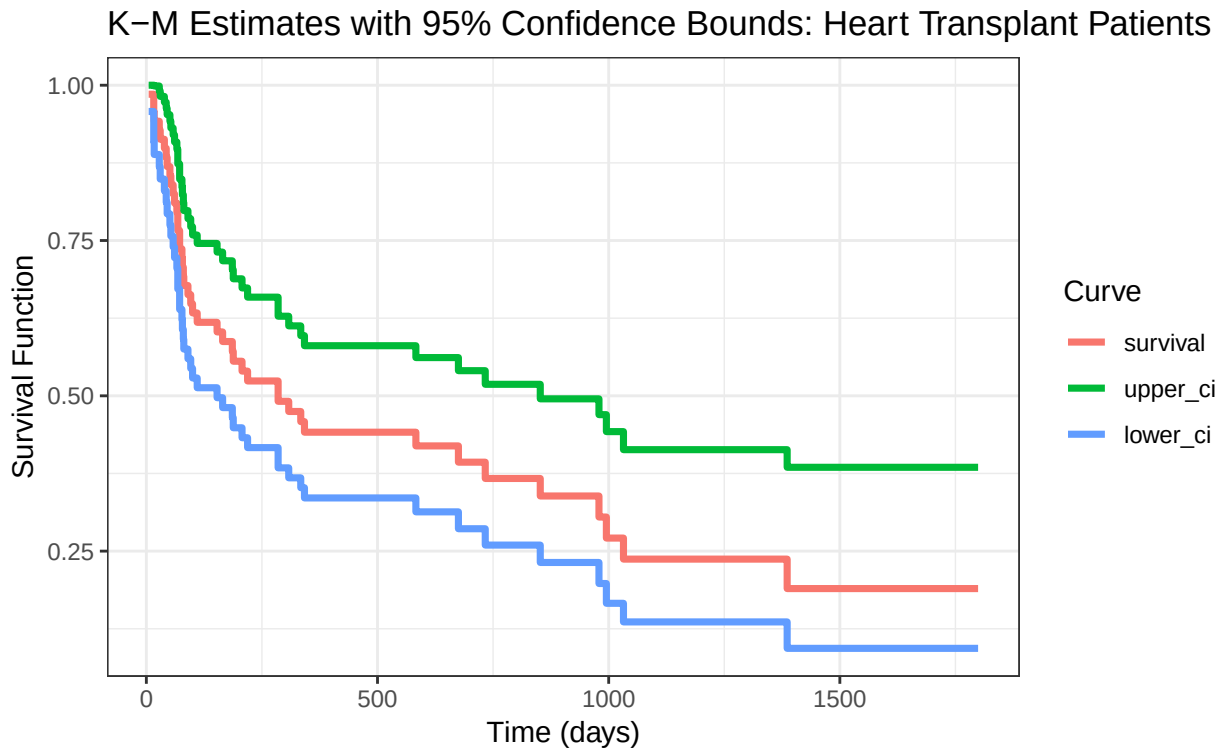
A preview of this dataset is provided here:

Table 1: Preview: Heart Transplant Data

Age	Status	Days
41	1	5
40	1	16
35	0	39
50	1	53
45	1	68
26	0	180

This dataset is right censored, because as shown above, we don't exactly how long patients who currently have a (status=0) will survive, or survived.

Utilizing this dataset, and the *survival* package in R, we generate the Kaplan-Meier estimates, and visualize the survival function, i.e. $S_{KM}(x_{i:n}) = 1 - F_{KM}(x_{i:n})$, with confidence intervals below:



Visual analysis of the plot here, indicates that the probability of survival for 500 days, after a heart transplant is approximately 42%, with a 95% confidence range of approximately 30-55%, among patients in this data, when this data was collected decades ago. We hope that survival rates have increased significantly since this study was conducted.

2. Kernel and Related Regression with One Explanatory Variable (40%):

Locate a data set suitable for nonparametric regression (usually has nonlinear y-x relationship) in the field of your interest, e.g., eCommerce, medical study, drug development, supply-chain/logistics operations. Apply all of the procedures below:

1) *Kernel Regression,*

2) *Local Polynomial Regression,*

3) *LOESS,*

4) *Smoothing Spline, to the y-x data-fit.*
   - Compare fits from the four methods.
     For this exercise, we located another dataset, that consists of Data give the mortality rate and the education level for 58 U.S. cities. It can be found via the following link on the *The Data And Story Library* site:https://dasl.datadescription.com/datafile/education-and-mortality, and has two columns with the following measures:

- Mortality: the mortality rate (deaths per 100,000 people)
- Education: the average number of years in school

1) *Kernel Regression,*

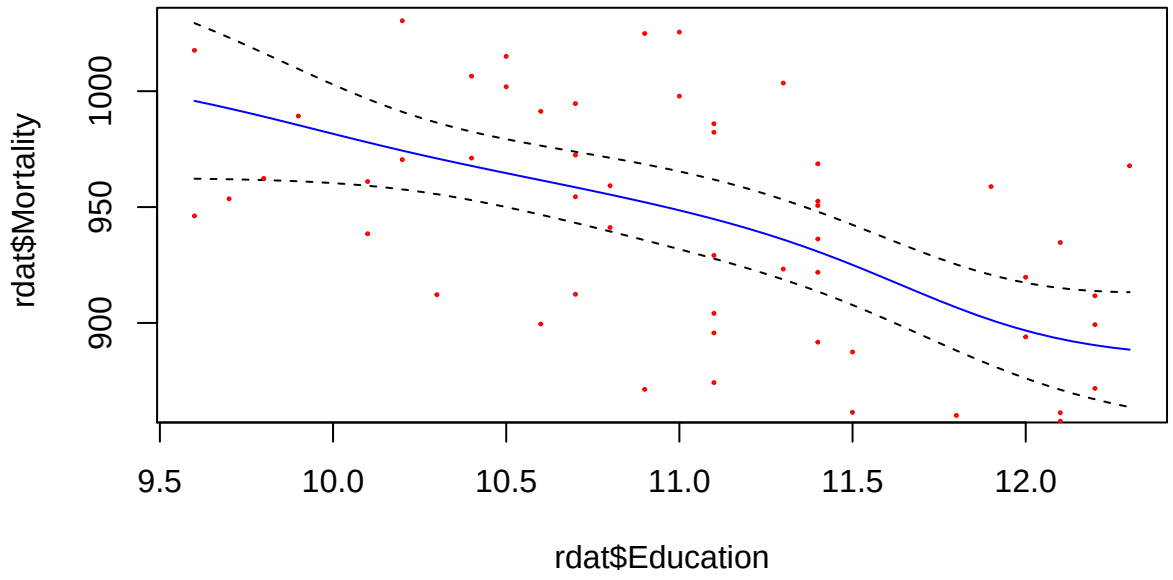
```
rdat <- read.csv('data/education-and-mortality.csv',header=T,stringsAsFactors=F)
bw <- npregbw(formula=rdat$Mortality~rdat$Education, tol=.1, ftol=.1)
```

```
##
Multistart 1 of 1 |
Multistart 1 of 1 |
Multistart 1 of 1 |
Multistart 1 of 1 |
Multistart 1 of 1 |
```

```
# Now take these bandwidths and fit the model and gradients
model <- npreg(bws = bw, gradients = TRUE)
summary(model)
```

```
##
## Regression Data: 58 training points, in 1 variable(s)
##                rdat$Education
## Bandwidth(s):      0.4382052
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 47.22676
## R-squared: 0.4196534
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
```

```
npplot(bws=bw, plot.errors.method="bootstrap",col = "blue")
points(rdat$Education, rdat$Mortality, cex=.2, col="red")
```

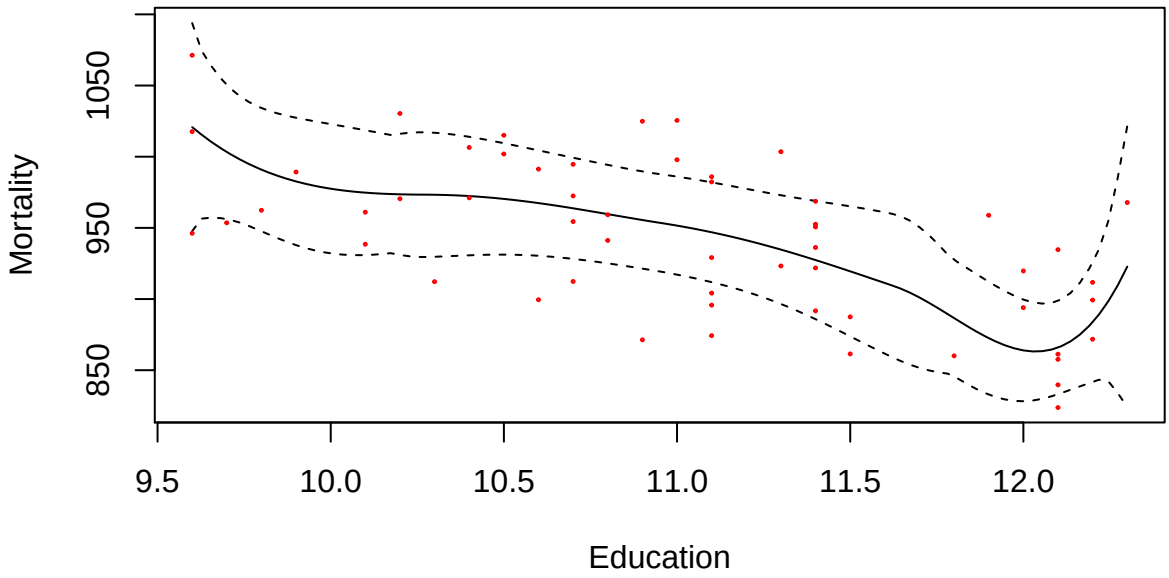


2) *Local Polynomial Regression,*

```
fit <- locfit(Mortality~lp(Education,deg=4),data=rdat)
summary(fit)
```

```
## Estimation type: Local Regression
##
## Call:
## locfit(formula = Mortality ~ lp(Education, deg = 4), data = rdat)
##
## Number of data points:  58
## Independent variables:  Education
## Evaluation structure: Rectangular Tree
## Number of evaluation points:  5
## Degree of fit:  4
## Fitted Degrees of Freedom:  6.982
```

```
crit(fit) <- crit(fit,cov=0.99)
plot(fit,band='local')
points(rdat$Education, rdat$Mortality, cex=.2, col="red")
```




```
res <- residuals(fit)
des <- rdat$Mortality-mean(rdat$Mortality)
r_square = 1- sum((res)^2/sum((des)^2))
```

3) *LOESS,*

```
#Making our variables easy to use in modeling
Mortality=rdat[,1]
Education=rdat[,2]

#packages to help create colors for plotting smoothing lines
library(RColorBrewer)
library(sp)
```

Loess models, with varying spans, with degree=1. When interpreting the summary results for the loess models, we will notice that increasing span will decrease the SSR but the curve of the model will become more jagged, making it so that the curve may not be the best representation. Loess is mostly a visual tool to see curves for non-linear curves through the use of data to form the line rather than a specified mathematical model. Loess is based purely on the span and degree of the polynomial for the potential loess model.

```
loess.model.1=loess(Mortality~Education,span=.25,degree=1)
loess.model.2=loess(Mortality~Education,span=.5,degree=1)
loess.model.3=loess(Mortality~Education,span=.7,degree=1)
loess.model.4=loess(Mortality~Education,span=.9,degree=1)
summary(loess.model.1)
```

```
## Call:
## loess(formula = Mortality ~ Education, span = 0.25, degree = 1)
##
## Number of Observations: 58
## Equivalent Number of Parameters: 7.85
## Residual Standard Error: 49.26
## Trace of smoother matrix: 9.33  (exact)
##
## Control settings:
##   span     :  0.25
##   degree   :  1
```

```
##   family   :  gaussian
##   surface  :  interpolate      cell = 0.2
##   normalize:  TRUE
##  parametric:  FALSE
## drop.square:  FALSE
```

```
summary(loess.model.2)
```

```
## Call:
## loess(formula = Mortality ~ Education, span = 0.5, degree = 1)
##
## Number of Observations: 58
## Equivalent Number of Parameters: 3.85
## Residual Standard Error: 49.02
## Trace of smoother matrix: 4.51  (exact)
##
## Control settings:
##   span     :  0.5
##   degree   :  1
##   family   :  gaussian
##   surface  :  interpolate      cell = 0.2
##   normalize:  TRUE
##  parametric:  FALSE
## drop.square:  FALSE
```

```
summary(loess.model.3)
```

```
## Call:
## loess(formula = Mortality ~ Education, span = 0.7, degree = 1)
##
## Number of Observations: 58
## Equivalent Number of Parameters: 3.12
## Residual Standard Error: 48.58
## Trace of smoother matrix: 3.61  (exact)
##
## Control settings:
##   span     :  0.7
##   degree   :  1
##   family   :  gaussian
##   surface  :  interpolate      cell = 0.2
##   normalize:  TRUE
##  parametric:  FALSE
## drop.square:  FALSE
```

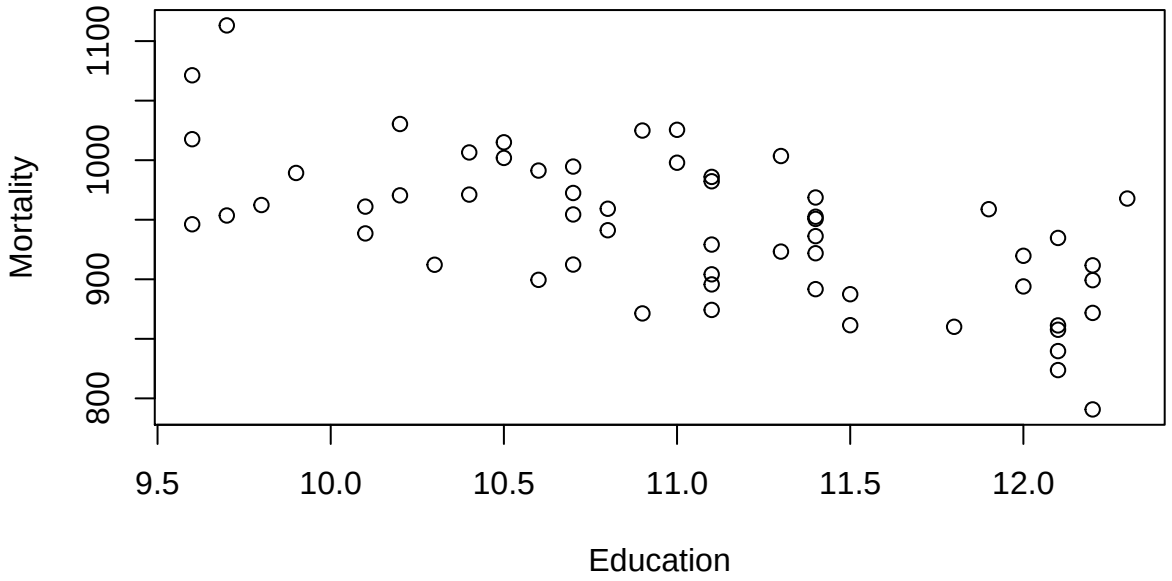
```
summary(loess.model.4)
```

```
## Call:
## loess(formula = Mortality ~ Education, span = 0.9, degree = 1)
##
## Number of Observations: 58
## Equivalent Number of Parameters: 2.49
## Residual Standard Error: 48.21
## Trace of smoother matrix: 2.8  (exact)
##
## Control settings:
##   span     :  0.9
```

```
##   degree   :  1
##   family   :  gaussian
##   surface  :  interpolate      cell = 0.2
##   normalize:  TRUE
##  parametric:  FALSE
## drop.square:  FALSE
```

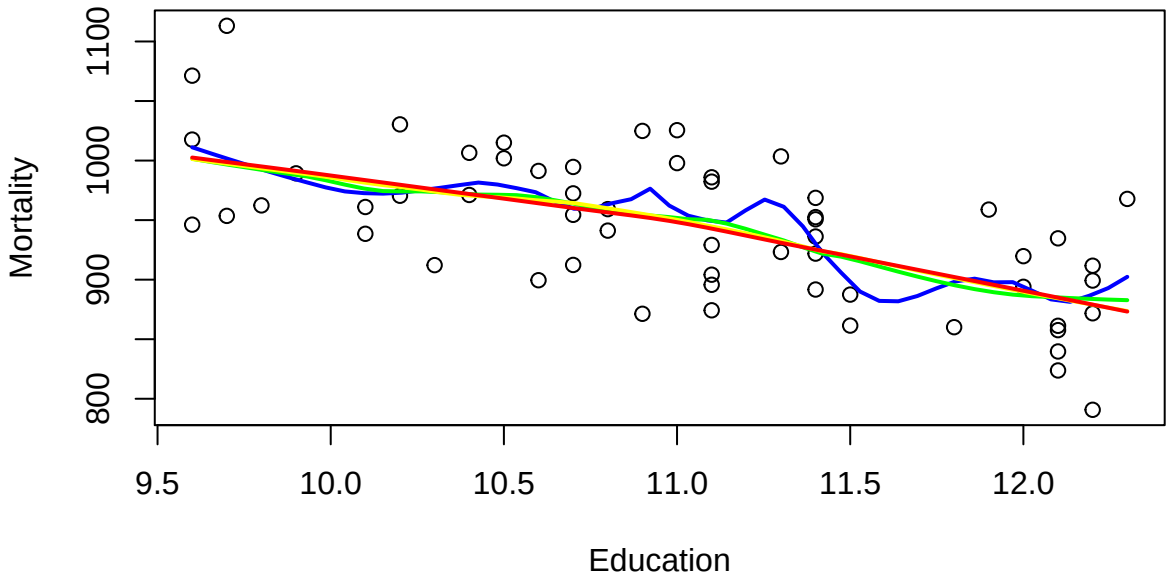
We will now plot the data points from our our mortality data,

```
plot(Mortality~Education)
```



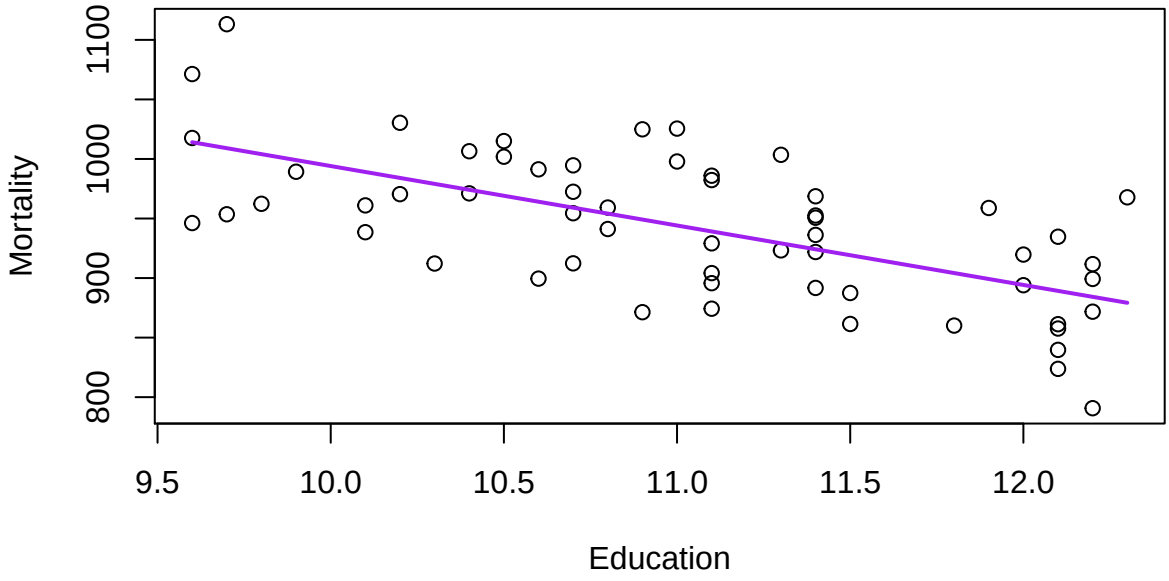

```
#The loess.smooth() function allows us to make fitted lines for the smoothed lines with varying spans
#We will plot various functions to be able to see the effect of span on the smoothing line
loessfit1=loess.smooth(x=Education,y=Mortality,span=.25,degree=1)
loessfit2=loess.smooth(x=Education,y=Mortality,span=.4,degree=1)
loessfit3=loess.smooth(x=Education,y=Mortality,span=.6,degree=1)
loessfit4=loess.smooth(x=Education,y=Mortality,span=.9,degree=1)

#Recall our plot with the mortality and education data. *****Needs legend*****
#We will now display the loess fits of varying span onto the plot to see the possible smoothing lines w
plot(Mortality~Education)
lines(loessfit1$x,loessfit1$y,col="blue", lwd=2)
lines(loessfit2$x,loessfit2$y,col="green", lwd=2)
lines(loessfit3$x,loessfit3$y,col="yellow", lwd=2)
lines(loessfit4$x,loessfit4$y,col="red", lwd=2)
```



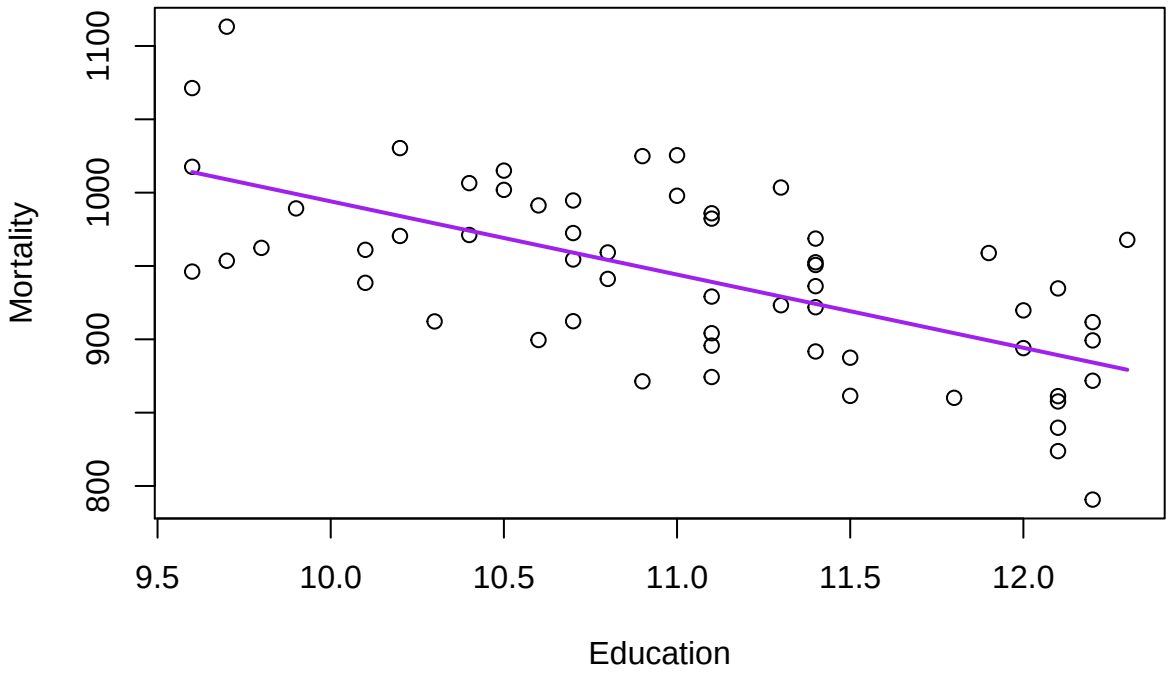
If we use 0.25 span for the case that it is the least jagged curve fit, while still holding some obvious grouping of values visually. One thing to keep note of is that the loess fit will be useful in visually displaying the relationship between the two variables mortality and education, but since our sample size is not particularly large, as loess fitting requires quite a large sample size to utilize effectively.

```
#Smoothing Spline Plot
plot(Mortality~Education)
spline.mortality=smooth.spline(Education,Mortality)
lines(spline.mortality, col="purple", lwd=2)
```



We can see that using a smoothing spline gives us a relatively linear line with small bumps along it.

4) *Smoothing Spline, to the y-x data-fit.*



We can see that using a smoothing spline gives us a relatively linear line with small bumps along it.

Comparison for above Kernel Estimators: 1) the Nadaraya-Watson kernel estimator suffers from bias, both at the boundaries due to the one-sided neighborhoods and in the interior when the xi's are not uniformly distributed. The reason is that in local constant modeling more or less the same points are used to estimate the curve near the boundary. 2) Local polynomial regression overcomes the above problem by fitting a higher degree polynomial here. What's more, Local polynomial regression also has other advantage such as,independence of the regression design (fixed or random, highly clustered or nearly uniform), and efficiency in an asymptotic minimax sense.

Comparison for Kernel Estimators and Nearest neighbor methods: 1) In general, nearest neighbor and kernel estimators produce similar results. For example: The loess approach is similar to the Nadaraya-Watson approach in that both are taking linear combinations of the responses y_i

2) In terms of bias and variance, the nearest neighbor estimator performs well if the variance decreases more than the squared bias increases. For example: Suppose that f is continuously differentiable up to second order and that $K(x, x_0) = 0$ if $|x - x_0| > h$ Then Loess: $Biasf(x_0) = O(h^2)$
Nadaraya-Watson: $Biasf(x_0) = f_0(x_0) \sum_i wi(x_i - x_0) + O(h^2)$,
where $w_i = K_h(x_i, x_0) / \sum_j K_h(x_j, x_0)$

The leading term for the bias of the Nadaraya-Watson estimator is referred to as design bias which is not present for loess estimators. In other words, loess performs naturally eliminates design bias, and the resulting estimator is free of bias up to second order.

3) Nearest neighbor methods is better if we have spaces with clustered design points followed by intervals with sparse design points.

Comparison for Kernel Estimators, Nearest neighbor methods and spline regressions:

1) In general, for one input variable and one output variable, smoothing splines can basically do everything which kernel regression can do.

2)While a higher order polynomial will provide a closer fit at any particular point, the loss of parsimony is not the only potential problem of over fitting, unwanted oscillations can appear between data points.Spline functions avoid this pitfall.

3) Kernels however are easier to program, analyze mathematically,extend more straightforwardly to

multiple variables, and to combinations of discrete and continuous variables.

4)Splines has faster computational speed and simplicity,as well as the clarity of controlling curvature directly.

3. Cross-Validation With the "Leave-One-Out" Procedure (10%):

Compare the above four methods with a leave-one-out cross-validation procedure.

4. Resampling Procedures: Bootstrap and Jackknife (25%):

1) *Select an input x_0 in the [min(x-data), max(x-data)].*

2) *Use all four regression models built in Task #2 to make point-predictions of Y at x_0.*

3) *Use both bootstrap ($B = 1000$) and jackknife resampling procedures to find a 90% pointwise confidence interval (CI) for the point-prediction. If the resampled distribution of the point-prediction is symmetric, use 5% in each tail to find the CI-bounds. If the distribution is not symmetric, use the HPD-interval idea to find the CI-bound. Compare the results from four regression methods.*