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

Yuan Gao, Kevin Lee, Akshay Govindaraj,
Yijun (Emma) Wan, Peter Williams, Ruixuan Zhang

Date: 2018-11-19

Contents

Workload Distribution

Workload and description of tasks completed by each team member for this project:

Team Member	Task Description
Yuan Gao	TBD
Kevin Lee	Loess, Smoothing Splines, Part of Regression Comparison
Akshay Govindaraj	CI for KM Estimator, Median estimate Bias and Standard Error
Yijun (Emma) Wan	TBD
Peter Williams	K-M Estimation, Code Compilation, Data Visualization & Latex Formatting
Ruixuan Zhang	Cross-validation Procedure

Problems

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. Is it possible to locate a software program that can provide the confidence band for the K-M estimate?

For this exercise, we located a dataset, that consists of measures on, $n = 69$, 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:

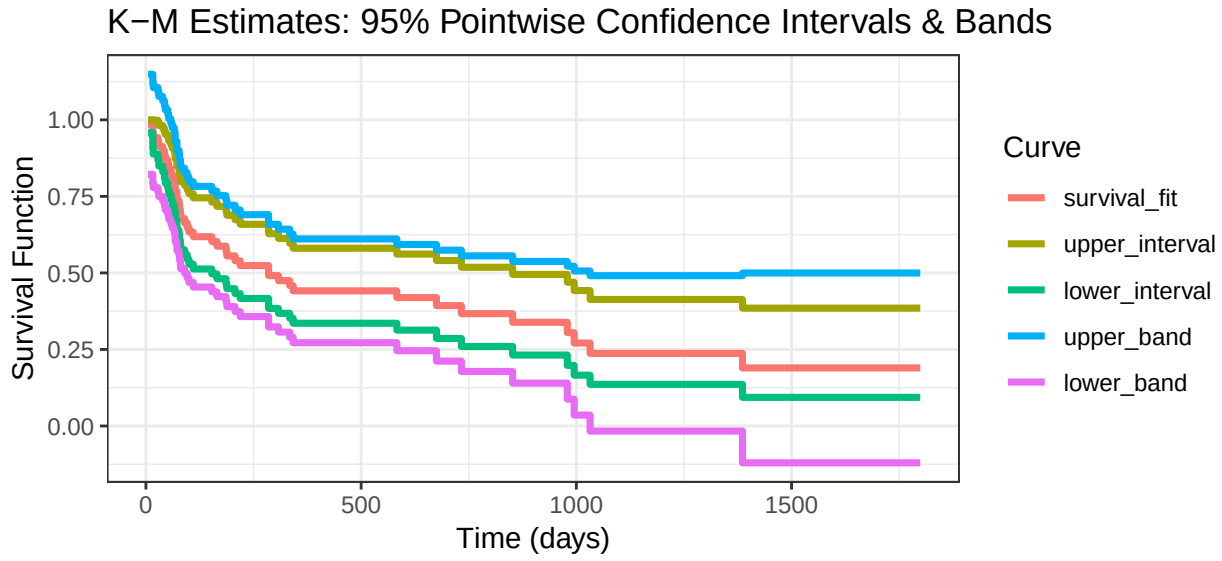
Table 1: Preview: Heart Transplant Data

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, visualized in the first plot below.

In the second plot, relying on the, the *km.ci* *R* package which can be found in the CRAN repository, in R which provides a number of confidence intervals and band for the Kaplan-Meier estimator. To compute confidence bands, the *km.ci* package has an option to utilize the estimation procedure detailed in *Confidence bands for a survival curve from censored data*, authored by W.J. Hall, and Jon A. Wellner, in the journal *Biometrika* in 1980. Code to reproduce these plots is provided in the code appendix:



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 improved since this study was conducted many years ago.

The plot also clearly shows that the *Hall-Wellner* confidence bands are wider than corresponding pointwise confidence intervals across time.

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,*

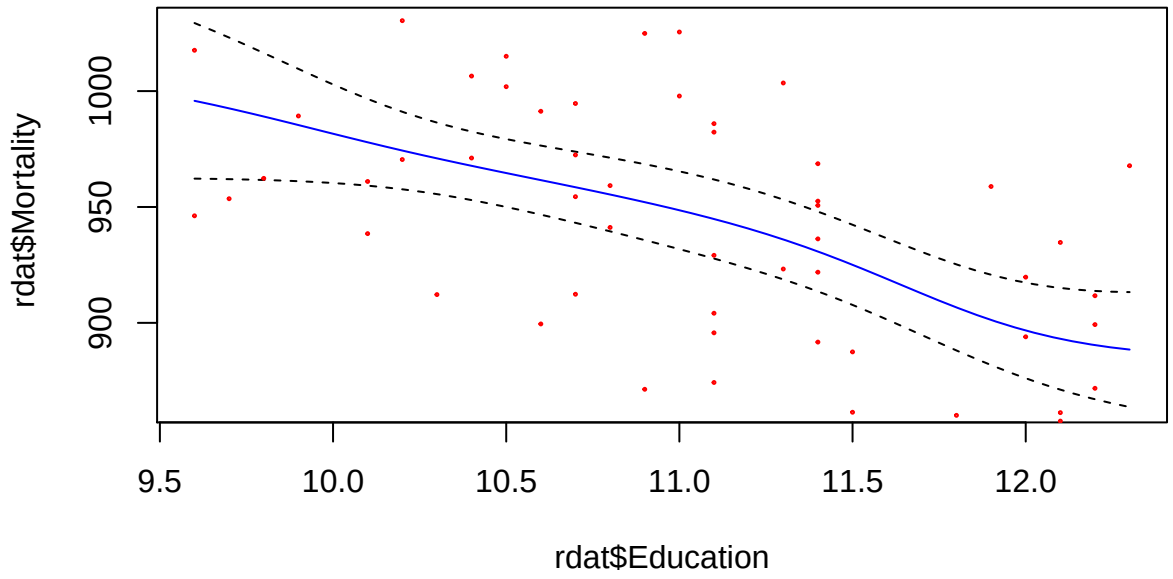
```
rdat <- read.csv('data/education-and-mortality.csv',header=T,stringsAsFactors=F)
bw <- npregbw(formula=rdat$Mortality~rdat$Education, tol=.1, ftol=.1)
```

```
##
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")
```

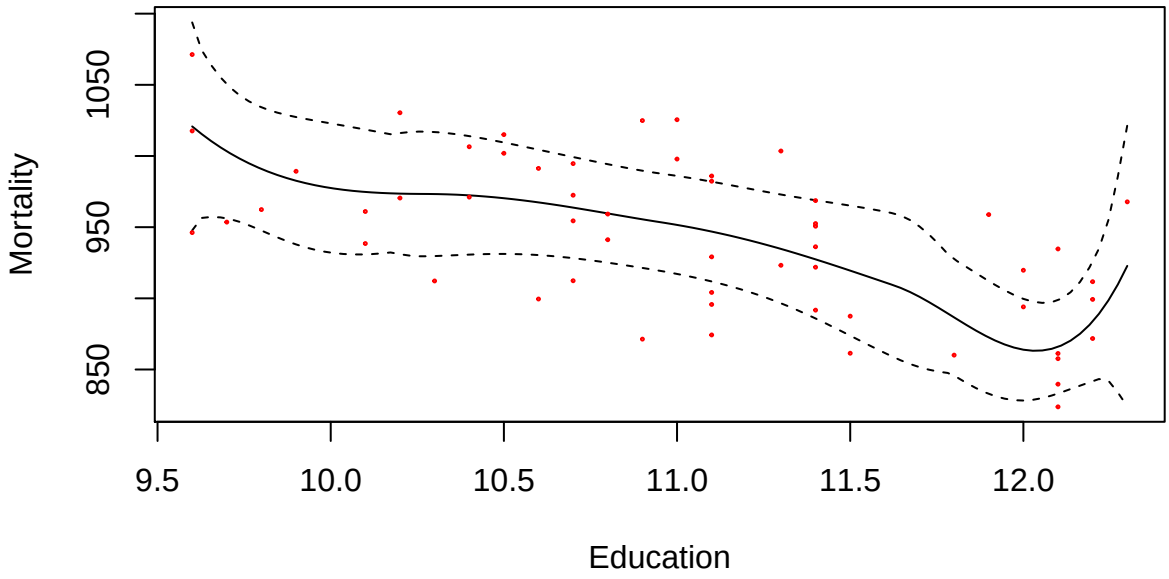


2) *Local Polynomial Regression,*

```
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,*

```
#named vectors for some reason
Mortality=rdat$Mortality
Education=rdat$Education
```

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 (quadratic or linear) for the potential loess model. For the purposes of demonstrating this technique on our data, we will use loess models with a degree of "1" which indicates a linear smoothing function compared to a quadratic form. Below, different spans are utilized to show the effect the span value has on the appearance of the curve.

```
loess.model.1=loess(Mortality~Education,span=.25,degree=2)
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : pseudoinverse used at 11.3
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : neighborhood radius 0.2
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : reciprocal condition number 2.6121e-21
```

```
loess.model.2=loess(Mortality~Education,span=.5,degree=2)
loess.model.3=loess(Mortality~Education,span=.7,degree=2)
loess.model.4=loess(Mortality~Education,span=.9,degree=2)
##Below is a summary of the loess model with span of .25 (as suggested by textbook)
summary(loess.model.1)
```

```
## Call:
## loess(formula = Mortality ~ Education, span = 0.25, degree = 2)
##
## Number of Observations: 58
## Equivalent Number of Parameters: 12.83
## Residual Standard Error: 47.61
## Trace of smoother matrix: 14.19  (exact)
##
## Control settings:
##   span     :  0.25
##   degree   :  2
##   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 as well as the loess smoothing fits.

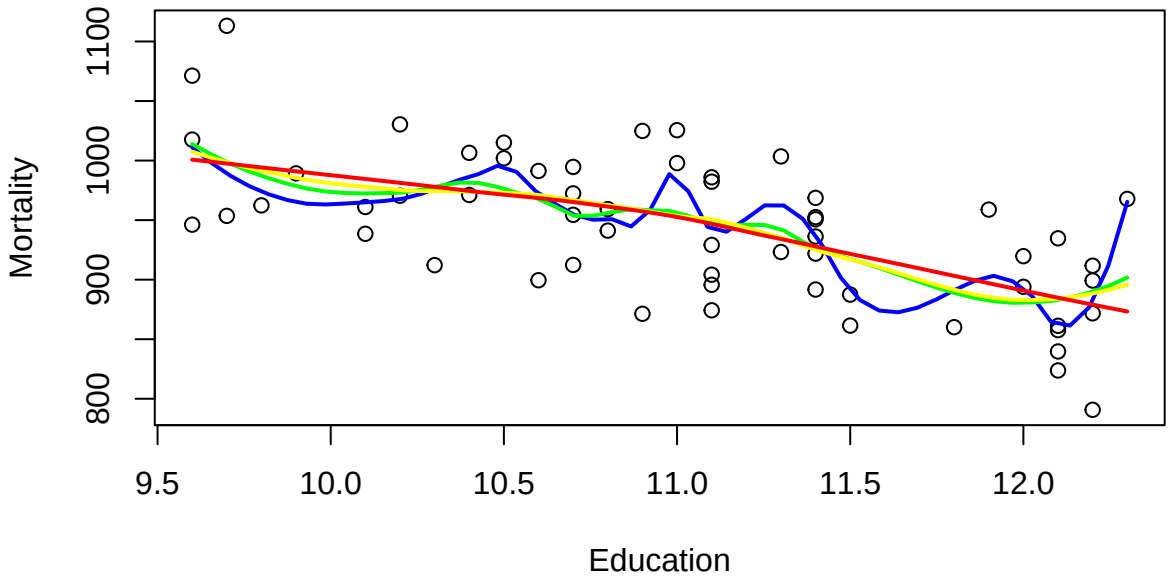
```
#The loess.smooth() function allows us to make fitted lines for the smoothed lines with varying
#We will plot various functions to be able to see the effect of span on the smoothing line
loessfit1=loess.smooth(x=Education,y=Mortality,span=.25,degree=2)
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 11.3
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 0.2
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 2.6121e-21

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 11.3

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 0.2

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 6.51e-17

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 11.3

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 0.2

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 9.7874e-17

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 11.3

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 0.2

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 5.4959e-17

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 11.3

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 0.2

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 2.6121e-21
```

```r
loessfit2=loess.smooth(x=Education,y=Mortality,span=.4,degree=2)
loessfit3=loess.smooth(x=Education,y=Mortality,span=.6,degree=2)
loessfit4=loess.smooth(x=Education,y=Mortality,span=.9,degree=2)
#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
#Spans of .25, .4, .6, and .9 are applied for the loess model fits below, with respective colors
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)
```



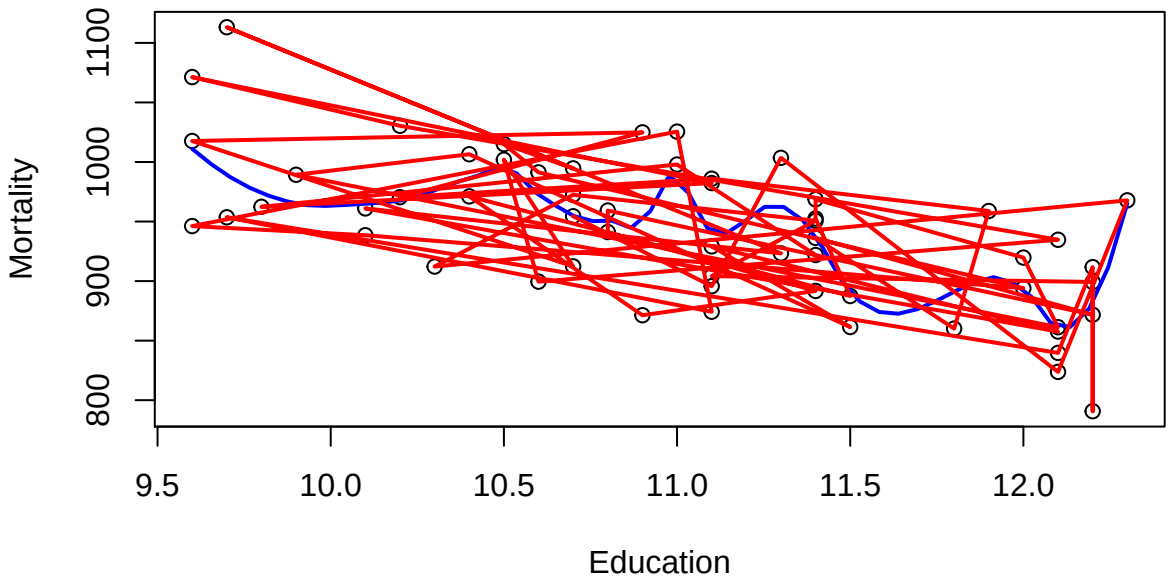
There are methods of optimizing the span to reduce sum of squared errors to have the best-fitting loess model. We use 0.25 span for the case that it is the least jagged curve fit, while still holding some obvious grouping of values visually. One thing to keep note of is that the loess fit will be useful in visually displaying the relationship between the two variables mortality and education, but our sample size is not particularly large, and oftentimes loess fitting requires quite a large sample size to utilize effectively. In minimizing sum of squared errors for our loess model, we would normally utilize cross-validation methods to determine an appropriate span, but there are also 'R' functions that automatically perform generalized cross-validation for choosing span.

```
#The loess.smooth() function allows us to make fitted lines for the smoothed lines with varying
#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=2)
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 11.3

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 0.2

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 2.6121e-21

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 11.3

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 0.2

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 6.51e-17

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 11.3

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
```

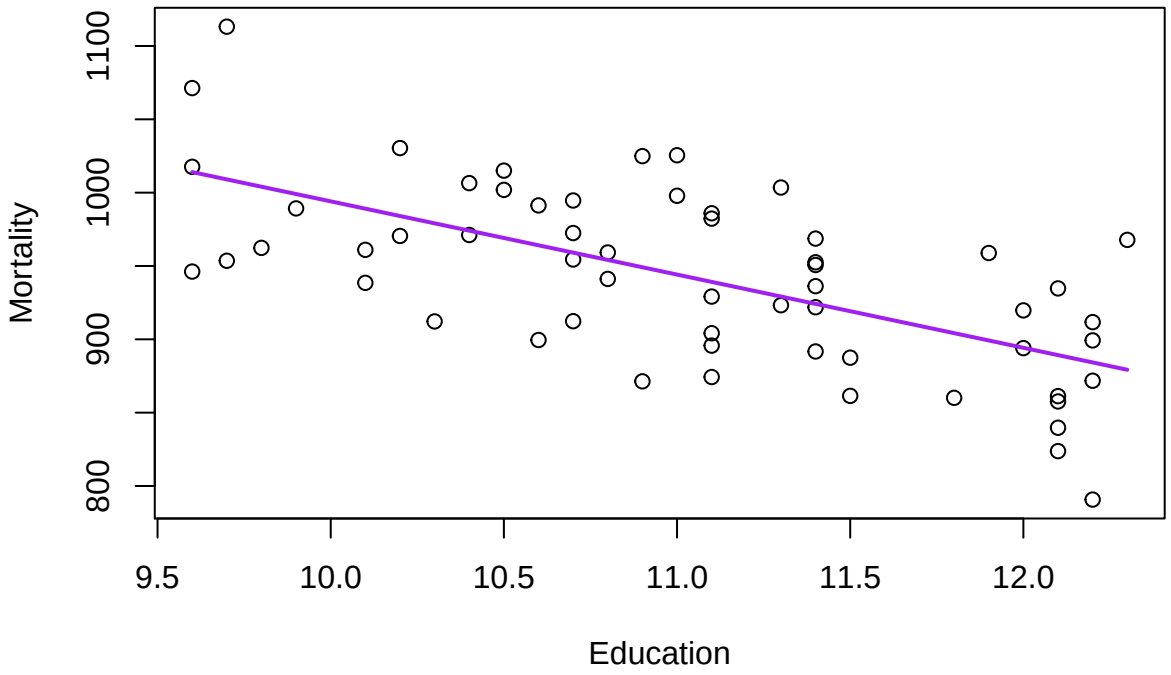
```
## FALSE, : neighborhood radius 0.2

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 9.7874e-17

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 11.3

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 0.2

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 5.4959e-17

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 11.3

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 0.2

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 2.6121e-21
```

```
loessfit2=loess.as(x=Education,y=Mortality,cv="gcv")
plot(Mortality~Education)
lines(loessfit1$x,loessfit1$y,col="blue", lwd=2)
lines(loessfit2$x,loessfit2$y,col="red",lwd=2)
```



Comparisons of these two models will be done with the upcoming cross-validation calculations.

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. One thing to notice with the usage of the spline function in R is the ease with which one can apply the method through programs (e.g. R). With the spline function, we can see a fairly clear decreasing linear relationship between the two variables. This makes sense considering the spread of the original data on a scatterplot, but this also falls in line with the loess smoothing used prior. They both show a generally decreasing relationship between the response and predictor variables, but the varying spans in the loess function allow one to control the minute details within the data's distribution one might want to account for. Also, the smooth.spline() function in 'R' is, similarly to usage in interpolating splines, defined through the use of a cubic spline in representing the fit of the model on the data. One significant tool in the smooth.spline function is the option to calculate two different cross-validation critical values (which is minimized to find an effective model of reducing error). If 'cv=TRUE' then the 'R' program will calculate the leave-one-out-cross-validation criterion, while setting it to false will provide a generalized cross-validation criterion, which can be used to help find the best smoother.

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. Loess smoothing curves do not provide a particular mathematical formula to be applied for prediction, but rather are used for visual analysis of the data and its relationships.

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.

The cross-validation score CV(h) is a criteria serving as an substitution of MISE, and usually used to determine the best bandwidth for the regression model for achieving the smallest variance of error. Below are the calculations of the cross-validation criteria for the Kernal Regression and Local polynomial Regression.

As we can see in the result of computing, the CV scores are rather high in Kernal Regression and Local Polynomial Regression. The scores of these two models are really close to each other, while the CV of Local Polynomial Regression seems slightly lower than that of Kernal Regression (with automated determined bandwidth), which can be due to the stochastic error.

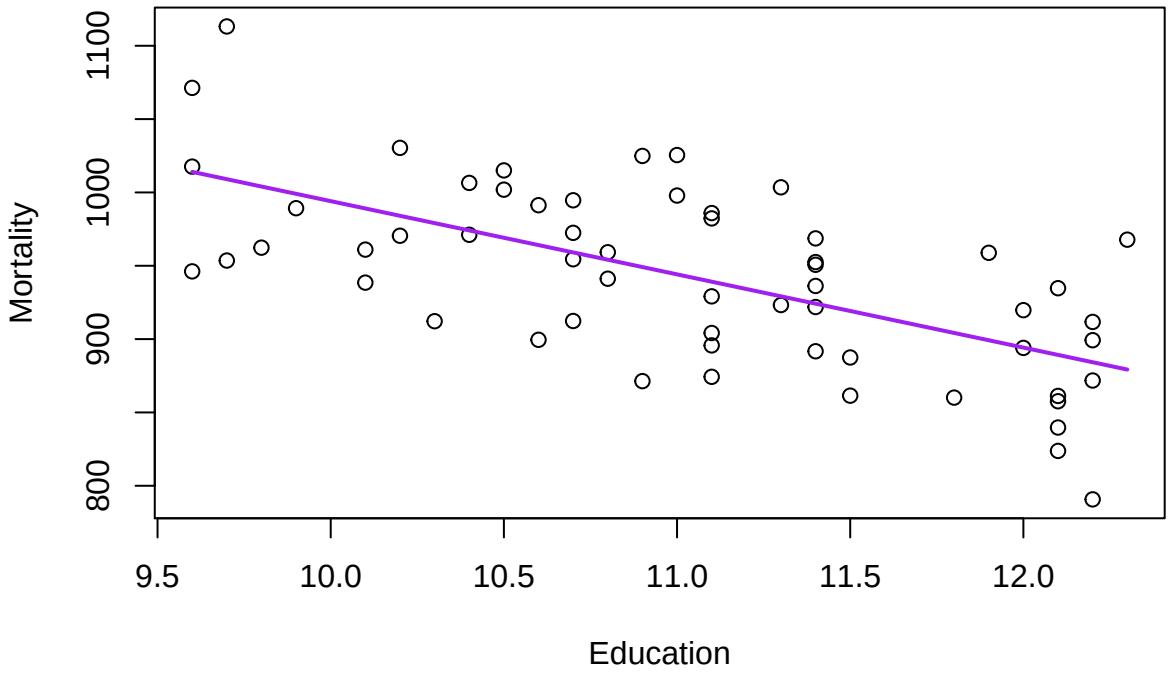
For LOESS models, cross-validation is a procedure commonly used to determine an optimally performing bandwidth with which our sum of squared errors would be reduced while retaining a smooth fit along the data (preferable to jagged fits). Below are the methods for computing the cross validation criteria for the LOESS models with different spans (0.25 and the automated span from (3.4):

Table 2: Leave Out One Cross Validation Results

Model Fit Type	CV Squared Error
Kernel Regression	2541
Local Polynomial	2688
Loess Span = (.25), Degree = 2	3361
Loess Auto Span, Degree = 2	2440

Through the leave-one-out procedure, we are able to see that the model with the smoother, automated span (which was optimized through another generalized cross validation procedure) has a lower criterion value, which is proportional to a lower mean squared error compared to the LOESS model with span=0.25.

For smoothing splines, as stated in the previous question, there is an automated option in the smooth.spline() 'R' function that allows one to perform leave-one-out cross-validation on a model with a spline fitted upon the data. This provides us with a CV criterion value of 2401.714 based on the smoothing functions chosen by R to fit the data.




```
## [1] 2378.615
```

The CV criterion calculated through the leave-one-out procedure (note that this would provide a different value compared to GCV procedure) seems lower compared to those fitted by either of the other three models above, showing the capability of splines in creating smoothing lines along the

data. Compared to smoothing spline, Other three models have higher variance and SSE and their fittings are not as good as smoothing spline which has the lowest CV.

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

i) Exercise the FIVE versions of CIs stated in CP-1 for the survival function S(x) give in 1) above. Compare these CIs. Moreover, focus on the MEDIAN (50th percentile), and evaluate the histogram of the bootstrap-estimates, bootstrap-bias and -standard-error. Comment about the computational results.

(TO ADD: Histogram and CI Table with different methods)

The normal approximation gives a much wider estimate for the Confidence Interval. The Bias corrected bootstrap estimate has a higher estimate for lower Confidence Interval which is in agreement with the qq-plot.

As seen on the qq-plot, the bootstrap estimates are at the normal approximation line for the upper CI estimate but for lower CI estimate the bootstrap estimates are different from the normal approximation. (Look at t values for standard quantile = +2 and -2)

Bias

The bias is calculated by taking the difference of the mean of bootstrap estimates and the original estimate from the Kaplan Meier estimator for median survival time. It turns out that the calculated value of bias is -4.32 Days.

This implies that the Kaplan Meier estimator underestimates the median survival time. Similarly, for different quantile estimates (median is 50%) the estimator will underestimate survival time.

In general, the samples of survival times are highly skewed. The bias for mean estimates for our sample turns out to be -9.45 days. Therefore, median is a much better measure of the central location than the mean.

Standard Error

The Standard error from the bootstrap estimates is 55.13 days. This is in agreement with our normal Confidence Estimates.

ii) Describe how to apply the bootstrap procedures to kernel regression for constructing pointwise CIs for the kernel regression function. You do not need to compute.

Following steps for finding Pointwise Confidence Interval of Kernel Regression

• Fix Bandwidth and Amplitude There will be a different Confidence Interval Estimate for different values of bandwidth and amplitude. The choice of kernel (eg. Epanechnickov, Normal) is not important. Ideally, we would expect different answers for pointwise Confidence Interval for different choice of kernel, but the difference in the results will not be significant.

• Find Bootstrap Samples For the given dataset find 10000 bootstrap samples and calculate the probability distribution using Kernel Density estimator. At the end of this we will have 10000 possible probability distribution functions with the same domain.

• Pointwise Confidence Intervals Using the bootstrap estimates of probability density at each point calculate the probability density of the 5%ile and 95%ile estimate from the bootstrap sample distributions and mark them as the lower and upper Confidence Interval estimates respectively.

```
hdat <- read.csv('data/heart_transplant.csv',header=T,stringsAsFactors=F)

km_fit <- survfit(Surv(time = Days, event = Status) ~ 1, data = hdat,
                  type = 'kaplan-meier') #only one group of patients
km_dat <- data.frame(time=km_fit$time,
                     survival=km_fit$surv,
                     upper_ci = km_fit$upper,
                     lower_ci = km_fit$lower) #model results
melted_km_dat <- melt(km_dat, id.vars = c('time')) #transform for viz

#Finding the Standard Error and survival
s_error = summary(km_fit)$std.err

#Calculate pointwise CI
CI_lower = km_fit$surv - 1.645*km_fit$std.err
CI_upper = km_fit$surv + 1.645*km_fit$std.err

km_dat1 <- data.frame(time=km_fit$time,
                      survival=km_fit$surv,
                      upper_ci = CI_upper,
                      lower_ci = CI_lower)

melted_km_dat1 <- melt(km_dat1, id.vars = c('time'))

# Table for 1st version of CI


#ggplot(aes(x=time,y=value, color = variable ,group = variable), data=melted_km_dat1) +
#  geom_step(size = 1.25) + theme_bw() + xlab('Time (days)') +
#  ylab('Survival Function') +
#  ggtitle('90% CI using the standard error') +
#  labs(color='Curve')

#############Function for bootstrapping Estimates for the Median
km_est_boot <- function(data, indices){
  bs_dat <- data[indices,]
  km_fitbs <- survfit(Surv(time = Days, event = Status) ~ 1, data = bs_dat,
                      type = 'kaplan-meier') #only one group of patients
  km_datbs <- data.frame(time=km_fitbs$time,
                         survival=km_fitbs$surv) #model results
  return(median(km_datbs$time))
  ### INCOMPLETE
}
```

```r
B <- 1e04
bs_CI <- boot(hdat, km_est_boot, R=B,parallel = 'multicore',
              ncpus = 10)
#plot(bs_CI)

boot.ci(boot.out = bs_CI,type = 'bca')
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bs_CI, type = "bca")
##
## Intervals :
## Level       BCa
## 95%   (165, 342 )
## Calculations and Intervals on Original Scale
```

```r
# Standard error of Bootstrap samples of Median
se <- sd(bs_CI$t)

#Bias calculation (Mean of bootstrap estimates - Median of KM estimator)
bias <- mean(bs_CI$t) - median(km_dat$time)
```

Code Appendix

Problem 1

Exemplary code to produce pointwise confidence intervals, and confidence bands, based on the 'Hall-Wellner' methodology and implemented in the *km.ci* package in *R*:

```r
#pointwise intervals
km_fit <- survfit(Surv(time = Days, event = Status) ~ 1, data = hdat,
                  type = 'kaplan-meier') #only one group of patients
#confidence band
conf_band <- km.ci(km_fit, conf.level=0.95, tl=min(hdat$Days), tu=max(hdat$Days), method="hall-w
km_dat <- data.frame(time=km_fit$time,
                     survival=km_fit$surv,
                     upper_interval = km_fit$upper,
                     lower_interval = km_fit$lower,
                     upper_band = conf_band$upper,
                     lower_band = conf_band$lower
                     ) #model results
melted_km_dat <- melt(km_dat, id.vars = c('time'))
#data vizualization
# ggplot(aes(x=time,y=value, color = variable ,group = variable), data=melted_km_dat) +
#   geom_step(size = 1.25) + theme_bw() + xlab('Time (days)') +
#   ylab('Survival Function') +
```

```
#   ggtitle('K-M Estimates with 95% Pointwise Confidence Intervals: Heart Transplant Patients')
#   labs(color='Curve') + xlim(0,1800)
```