Problem_Set_5

Carlos Sathler

3/27/2019

Contents

Problem 1 - Non-Parametric Regression on Data Set "flights"

(a) EDA and data clean-up

```
# remove rows with na's
sum(is.na(flights$arr_delay))
```

```
## [1] 9430
```

```
sum(is.na(flights$hour))
```

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

```
sum(is.na(flights$minute))
```

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

```
flights.df = flights[-which(is.na(flights$arr_delay)),]
# we're interested in predicting arr_delay from hour and minute only
cols = c('arr_delay', 'hour', 'minute')
flights.df = flights.df[, cols]
# flight delays are heavily skewed, so will log transform
# but need first to shift values, since there are negative and zero delays
arr_delay_shift = abs(min(flights.df$arr_delay)) + 1
flights.df$arr_delay_log = log(flights.df$arr_delay + arr_delay_shift)
summary(flights.df)
```

```
##    arr_delay             hour           minute      arr_delay_log
##  Min.   : -86.000   Min.   : 5.00   Min.   : 0.00   Min.   :0.000
##  1st Qu.: -17.000   1st Qu.: 9.00   1st Qu.: 8.00   1st Qu.:4.248
##  Median :  -5.000   Median :13.00   Median :29.00   Median :4.407
##  Mean   :   6.895   Mean   :13.14   Mean   :26.23   Mean   :4.468
##  3rd Qu.:  14.000   3rd Qu.:17.00   3rd Qu.:44.00   3rd Qu.:4.615
##  Max.   :1272.000   Max.   :23.00   Max.   :59.00   Max.   :7.215
```

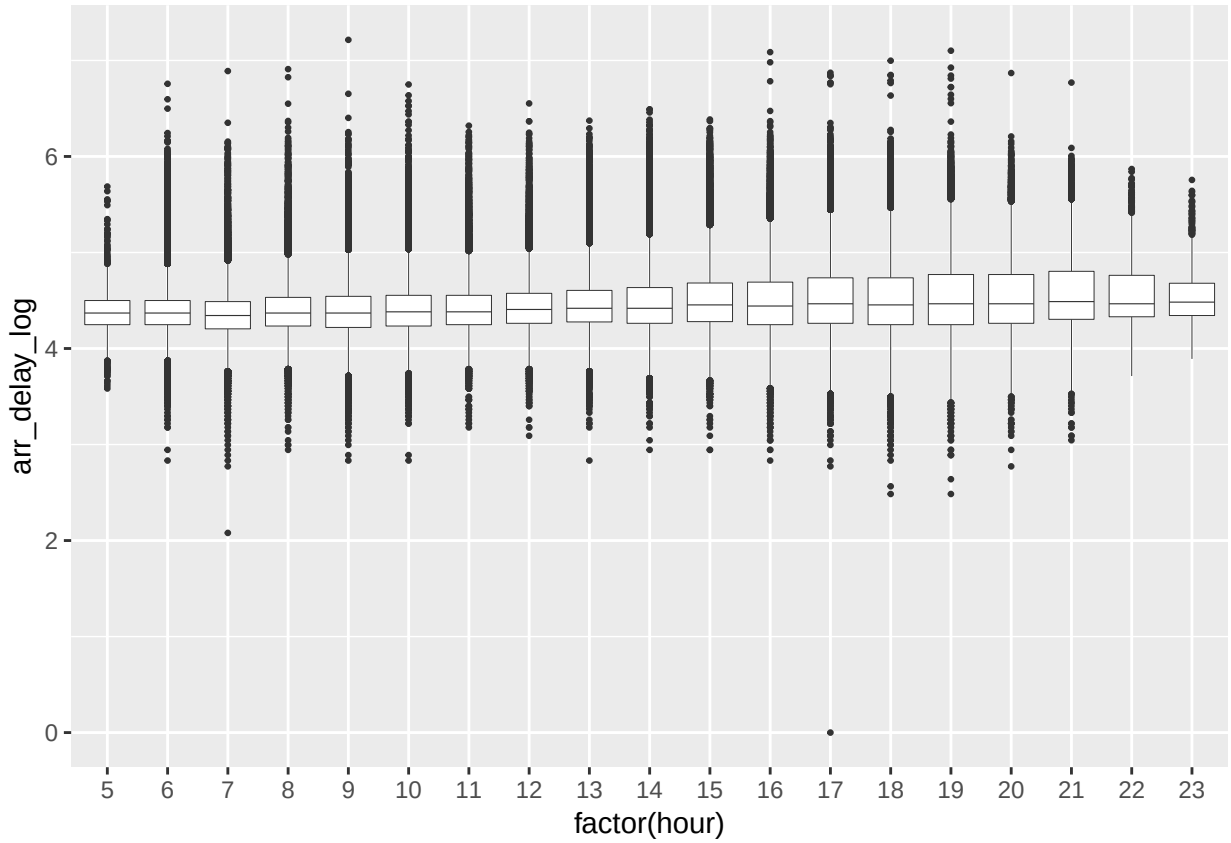
```
str(flights.df)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    327346 obs. of  4 variables:
##  $ arr_delay    : num  11 20 33 -18 -25 12 19 -14 -8 8 ...
##  $ hour         : num  5 5 5 5 6 5 6 6 6 6 ...
##  $ minute       : num  15 29 40 45 0 58 0 0 0 0 ...
##  $ arr_delay_log: num  4.58 4.67 4.79 4.23 4.13 ...
```

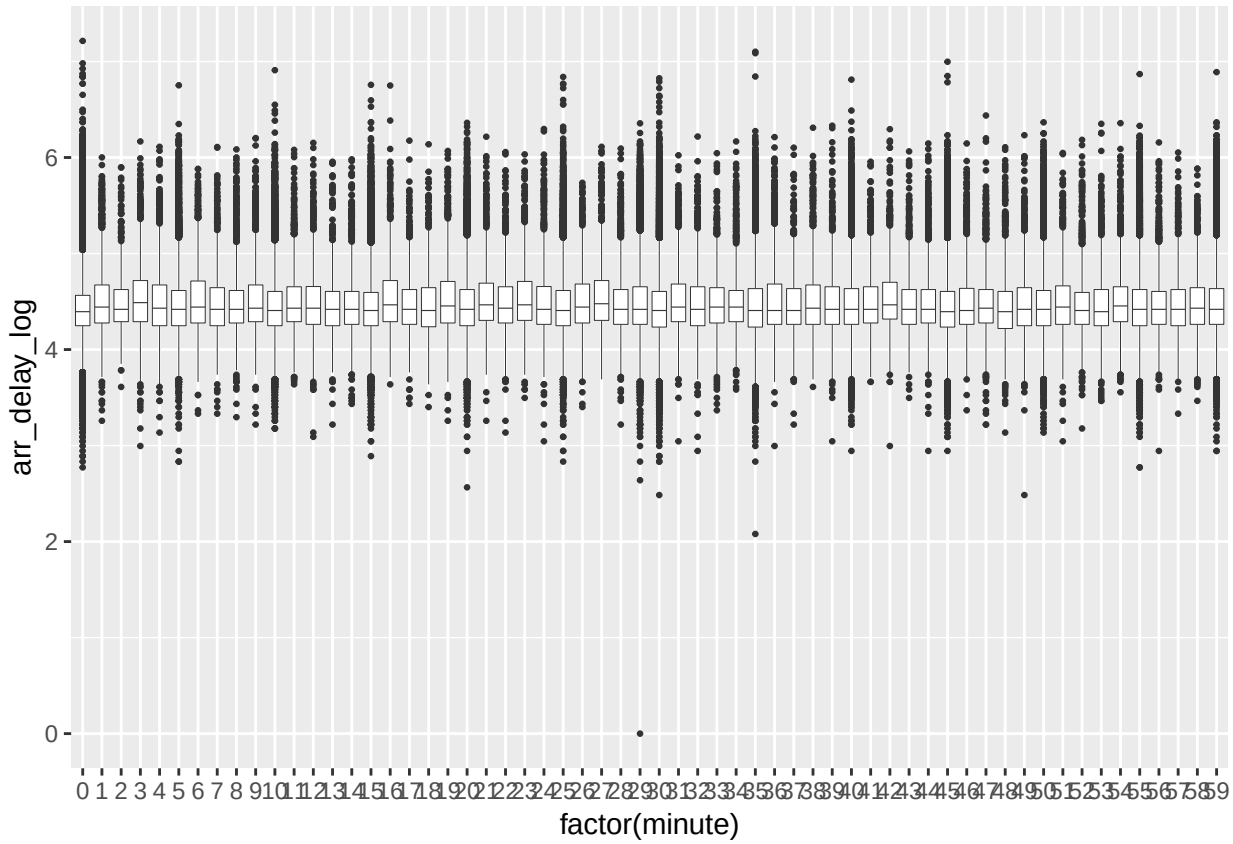
```
cor(flights.df[,c('arr_delay_log', 'hour', 'minute')])
```

```
##               arr_delay_log       hour     minute
## arr_delay_log    1.00000000 0.17464618 0.02274641
## hour             0.17464618 1.00000000 0.04193144
## minute           0.02274641 0.04193144 1.00000000
```

```
ggplot(flights.df, aes(x=factor(hour), y=arr_delay_log)) + geom_boxplot(size=0.1, outlier.size=0.5)
```



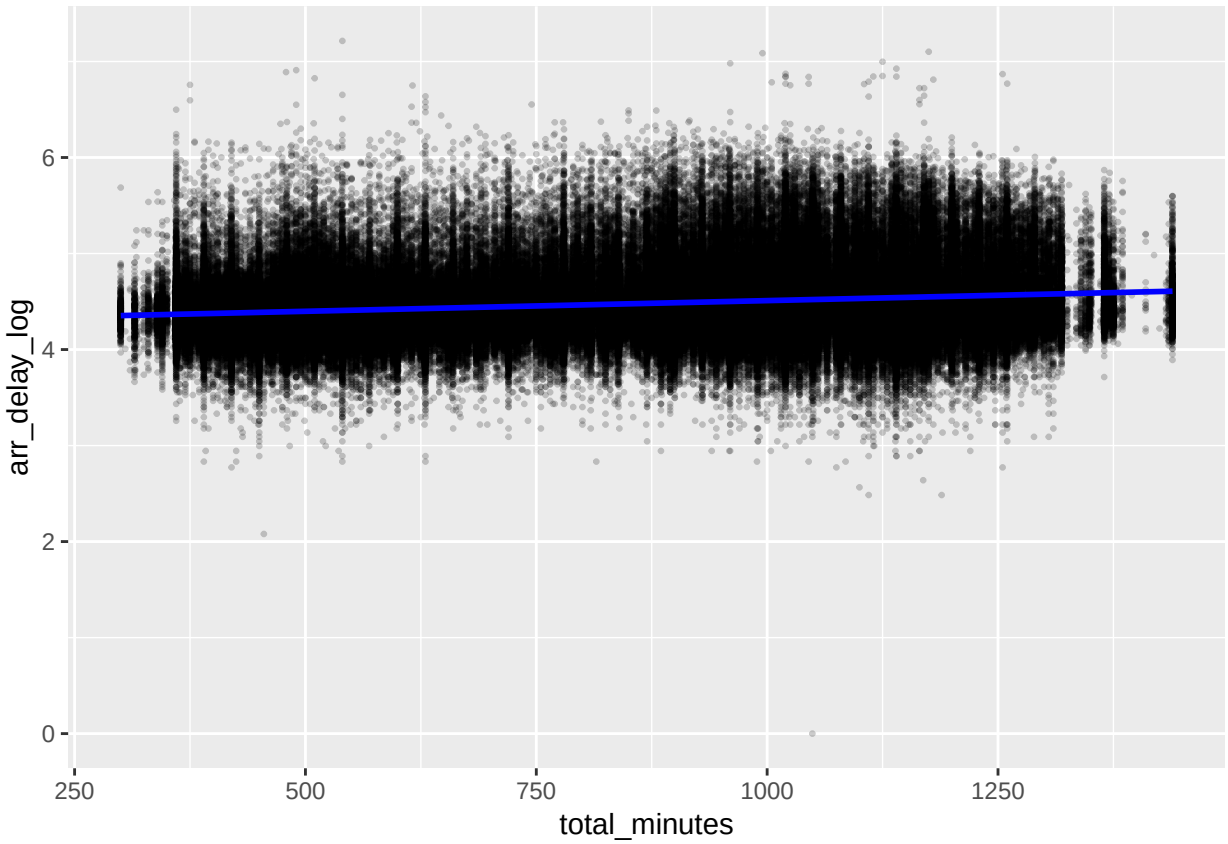

```
ggplot(flights.df, aes(x=factor(minute), y=arr_delay_log)) + geom_boxplot(size=0.1, outlier.size=0.5)
```




```
# will also explore transforming hour and minute into number of minutes after midnight
flights.df$total_minutes = flights.df$hour * 60 + flights.df$minute
cor(flights.df[,c('arr_delay_log', 'total_minutes')])
```

```
##                arr_delay_log total_minutes
## arr_delay_log     1.0000000     0.1752936
## total_minutes     0.1752936     1.0000000
```

```
gg = ggplot(flights.df, aes(x=total_minutes, y=arr_delay_log)) + geom_point(size=0.5, alpha=0.2)
#gg = gg + geom_smooth(method='loess', color='orange', se=F)
gg = gg + geom_smooth(method='lm', color='blue', se=F)
#gg = gg + geom_smooth(method='gam', formula = y ~ s(x), color = 'green', se=F)
gg
```



EDA suggests I should work with log of arrival delay. However, because I found negative and zero "delays", I shifted the data so I could apply log transformation.

Additionally, I combined hours and minutes of departure into a single field "total minutes" which tracks departure time in minutes after midnight. The combined field will be used for local and regularized regression; for the gam model I will explore using hours and minutes separately.

(b) Partition dataset

I use the caret package to partition the data. Training partition has 70% of the data, test partition 30%.

```
# simplify dataset
y = flights.df$arr_delay_log
x = flights.df$total_minutes
# for gam model, we will use hour and minutes separately
x1 = flights.df$hour
x2 = flights.df$minute
df = data.frame(y, x, x1, x2)
# i will use caret package, and will train on 70% of the data
set.seed(681)
trainIndex = createDataPartition(df$y, p = .7, list = F, times = 1)
train.df = df[trainIndex,]
test.df = df[-trainIndex,]

if (1 == 2) {
  set.seed(681)
  # will test performance with n records first
```

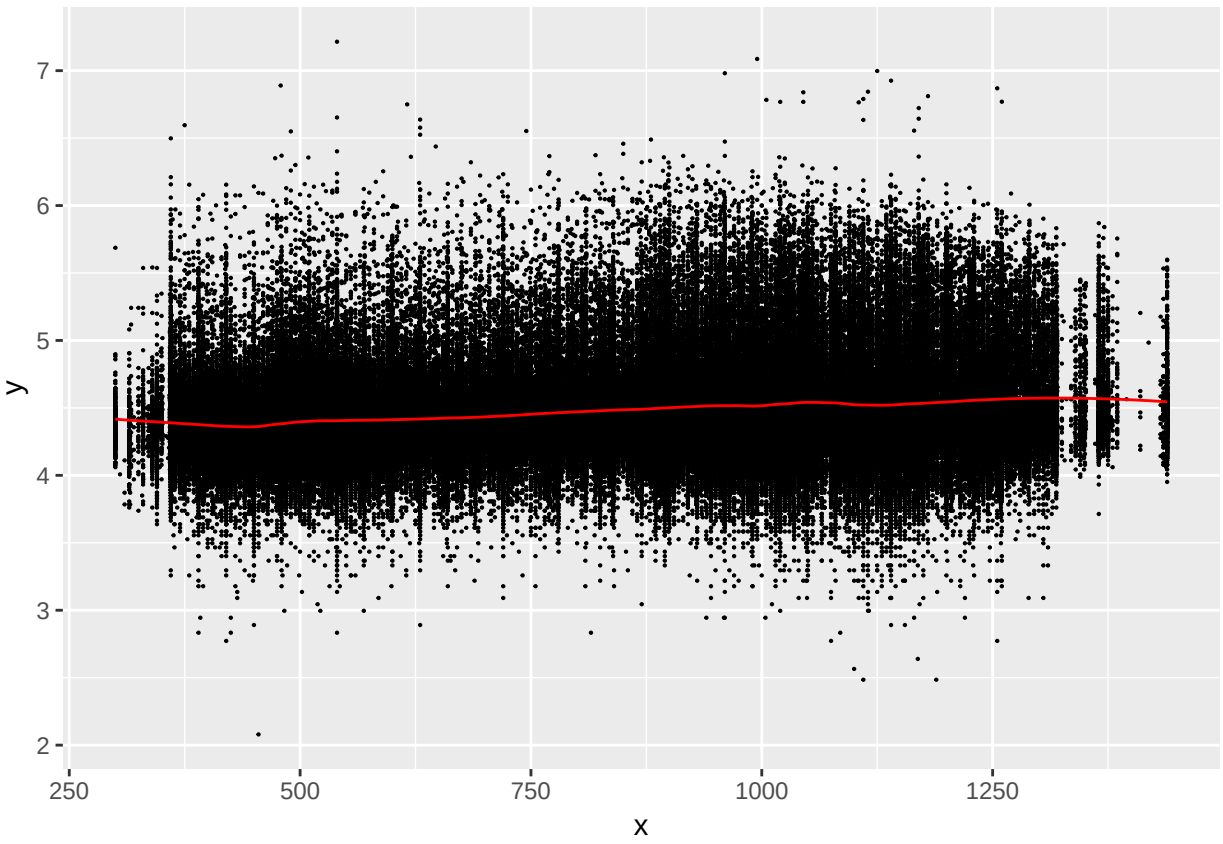
```
  n = 10000
  train.df = sample_n(train.df, size=n, replace=F)
}
# prepare vectors for training
y = train.df$y
x = train.df$x
x1 = train.df$x1
x2 = train.df$x2
```

(c) Model fits

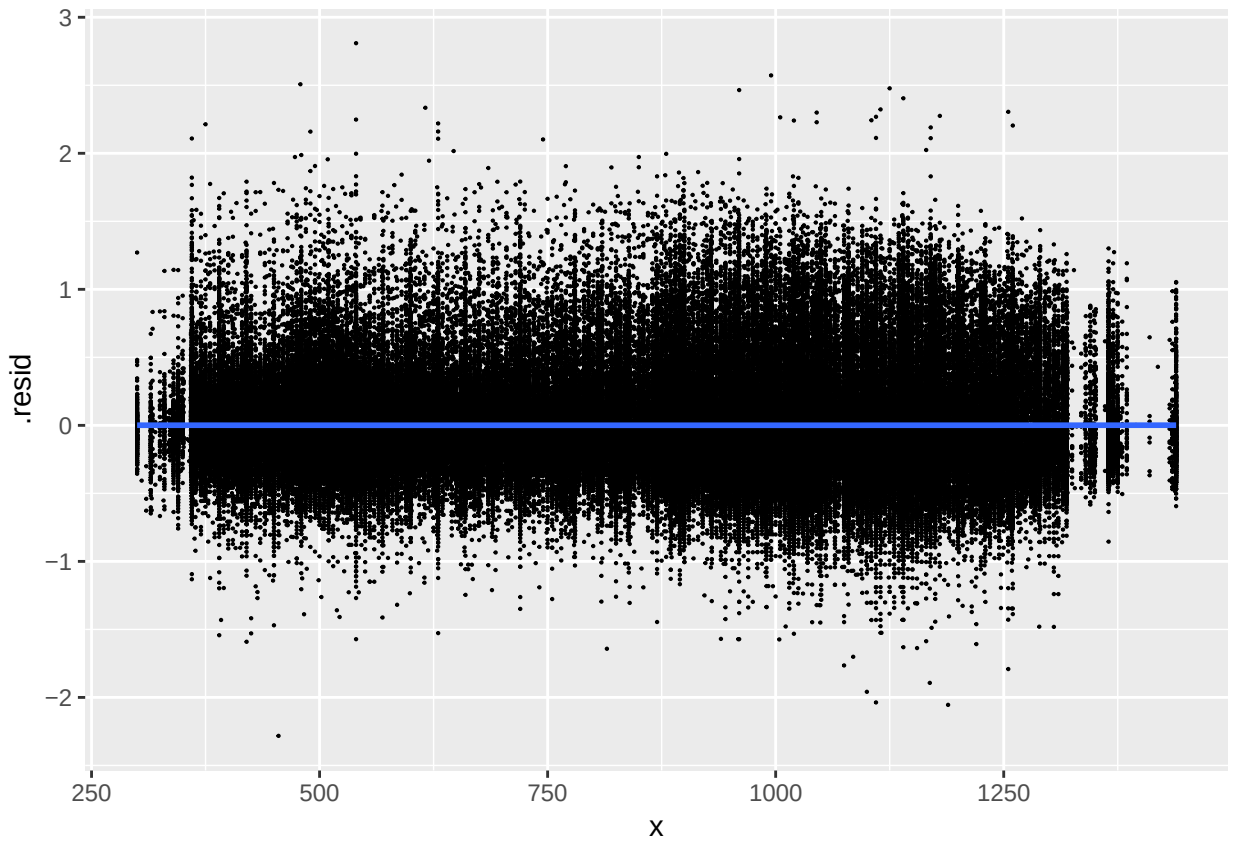
Model A: loess()

I assumed parameter tuning with cross-validation was beyond the scope of problem 1.a. What I did instead was try different parameter values for span and family, inspecting the plot of the fitted values on the training partition, along with plot of residuals, for different parameter values. For loess I tried lower span values in an attempt to reduce smoothing and thus capture subtle changes in the explanatory variable. I settled with 0.25 because the plot of fitted values showed some curviness. I also tried to use family="simmetric" to reduce effect of outliers but the plot of residuals showed positive slope as a result of that change. So I settled with fitting by least-squares. Note: QQ plot of residuals did not change much with different parameter values.

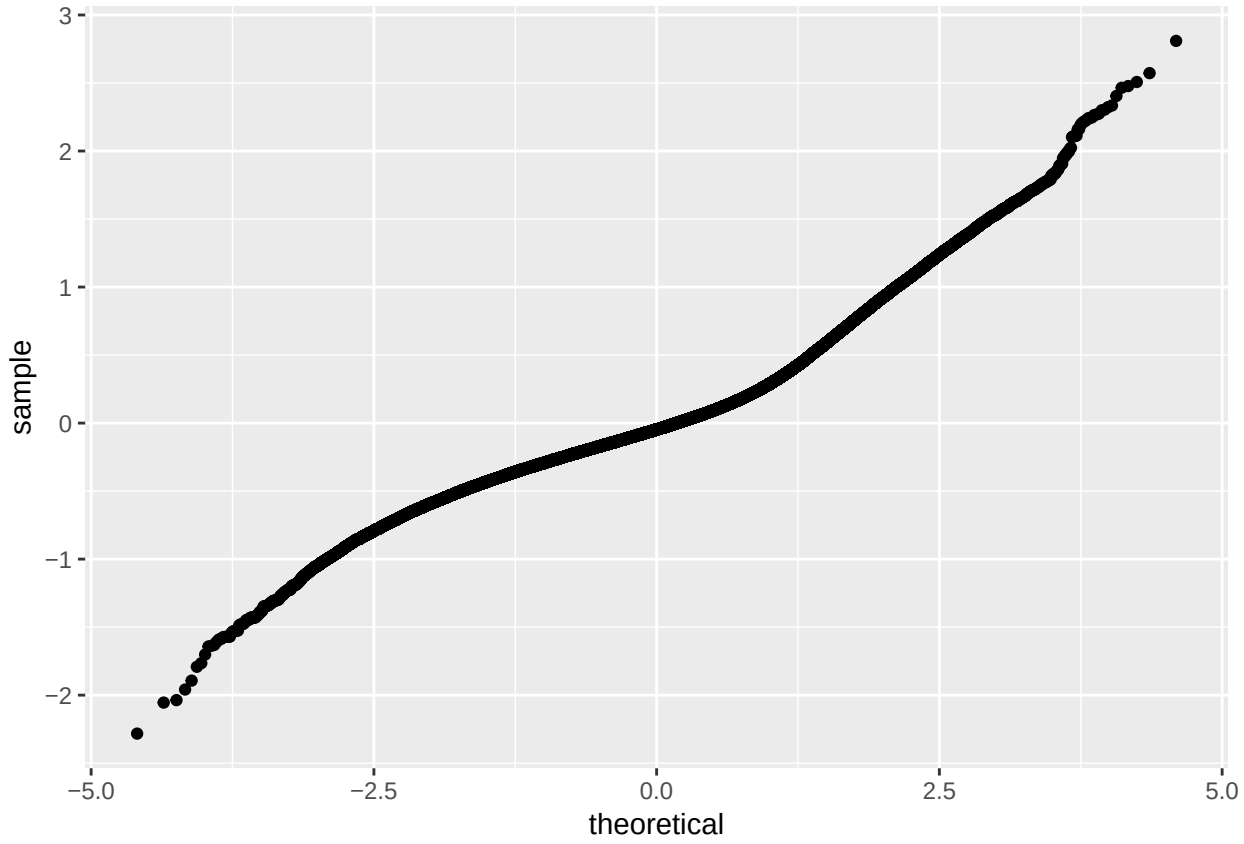
```
time1 = Sys.time()
model.loess = loess(y ~ x, span=0.25)
model.loess.df = augment(model.loess)
ggplot(model.loess.df, aes(x=x, y=y)) + geom_point(size=0.1) + geom_line(aes(x=x, y=.fitted), color='re
```



```
gg = ggplot(model.loess.df, aes(x=x, y=.resid)) + geom_point(size=0.1)
gg = gg + geom_smooth(method = "gam", formula = y ~ s(x))
gg
```




```
ggplot(model.loess.df) + geom_qq(aes(sample=.resid))
```




```
time2 = Sys.time()
difftime(time2, time1)
```

```
## Time difference of 2.519767 mins
```

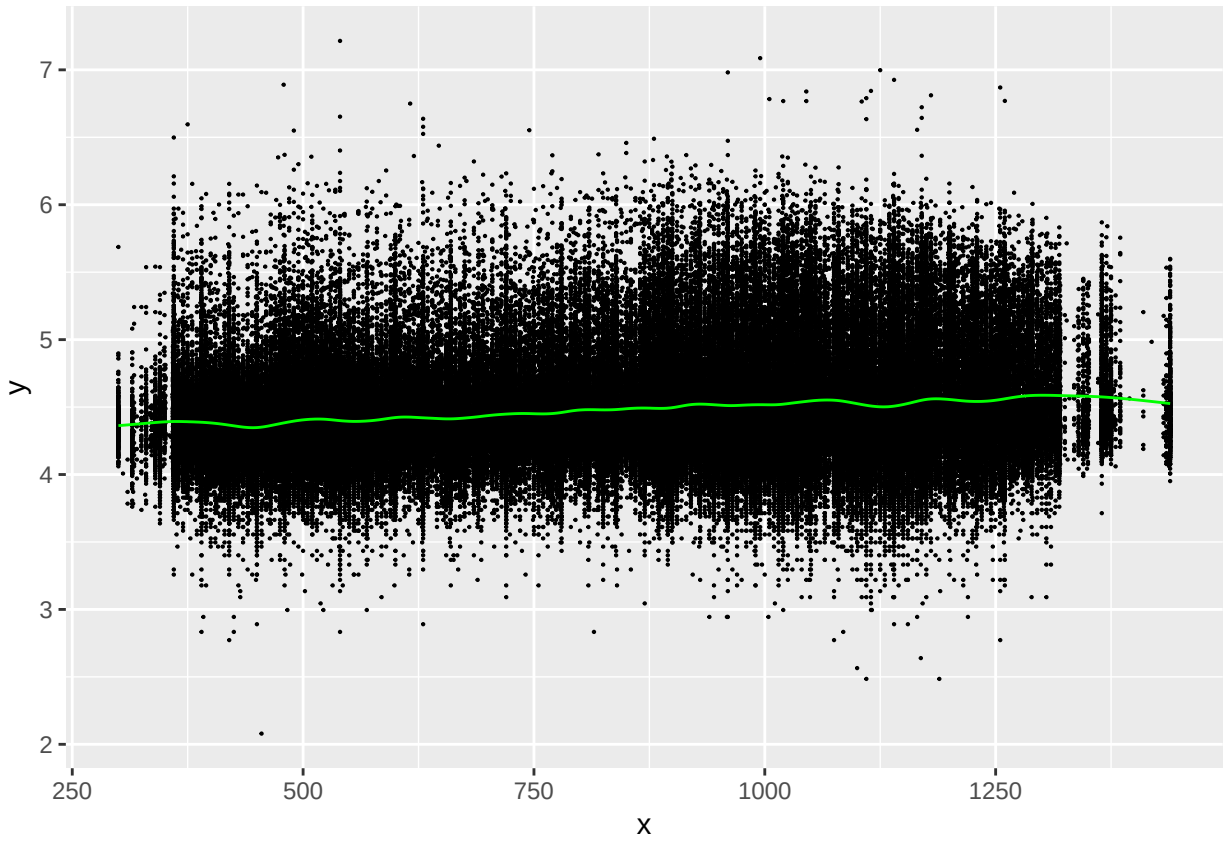
Model B: spline() with smooth.spline()

For regularized regression I chose to use the cv parameter delivered with the smooth.spline function. This choice offers an easy way to use leave-one-out cross validation to tune spar and df parameters. As a results of cross-validation alone, I expect the regularized regression model to outperform loess on the training set later on.

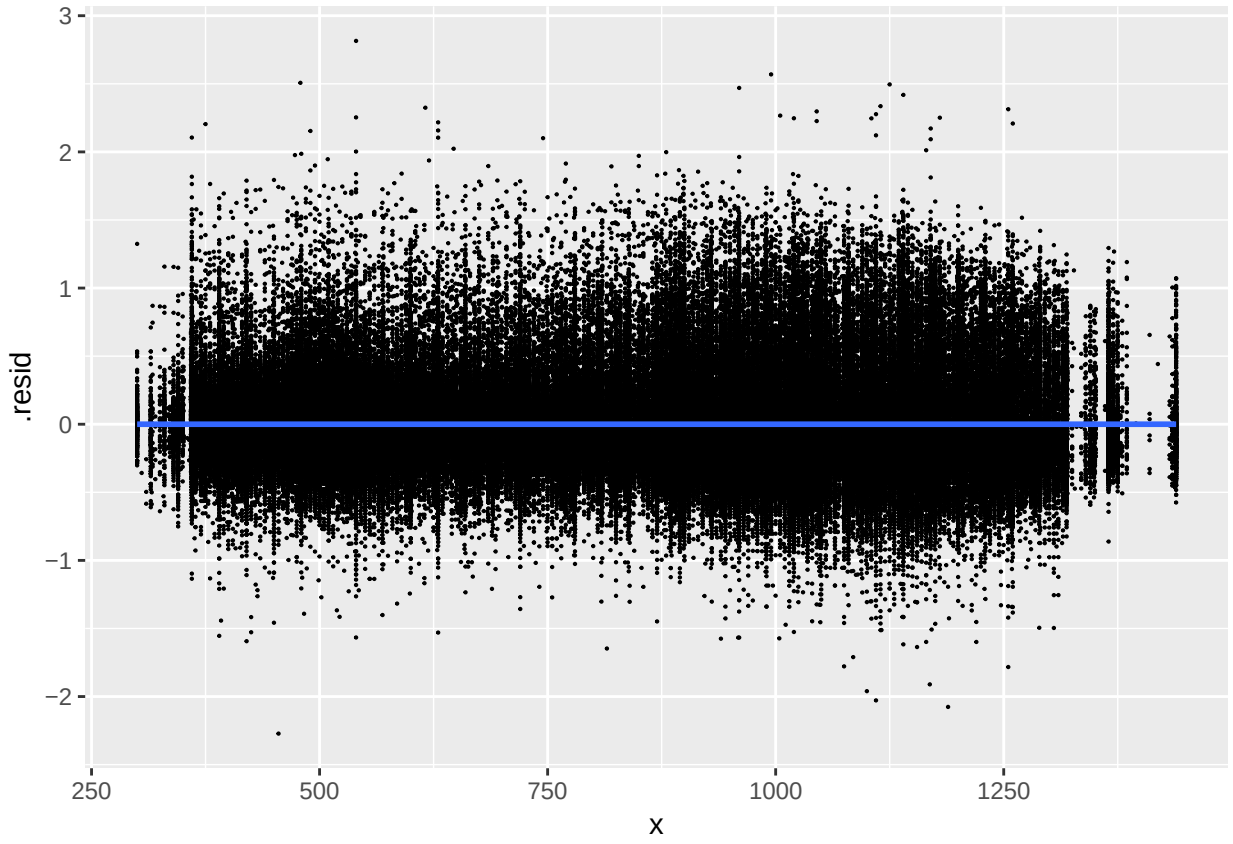
```
time1 = Sys.time()
model.spline = smooth.spline(y ~ x, cv=T)
```

```
## Warning in smooth.spline(y ~ x, cv = T): cross-validation with non-unique
## 'x' values seems doubtful
```

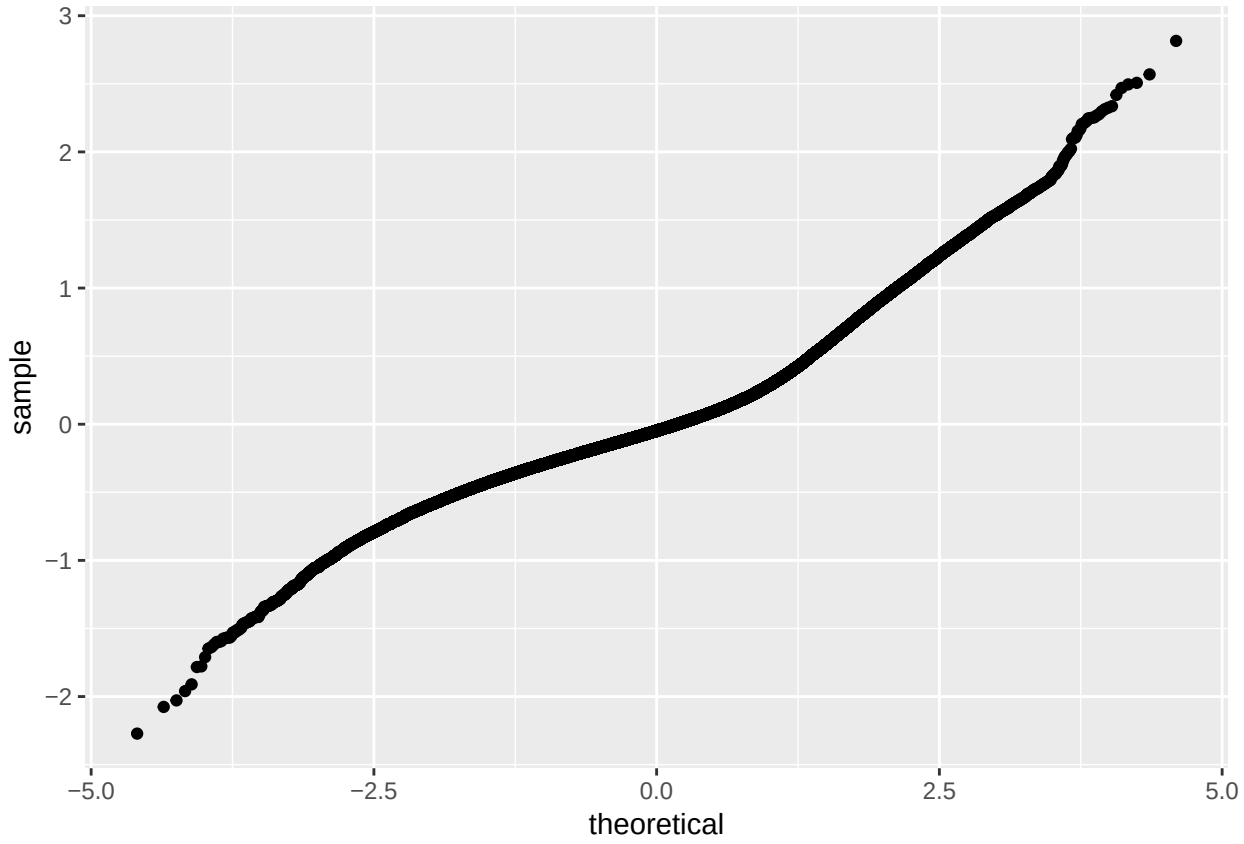
```
model.spline.df = augment(model.spline)
ggplot(model.spline.df, aes(x=x, y=y)) + geom_point(size=0.1) + geom_line(aes(x=x, y=.fitted), color='g
```




```
gg = ggplot(model.spline.df, aes(x=x, y=.resid)) + geom_point(size=0.1)
gg = gg + geom_smooth(method = "gam", formula = y ~ s(x))
gg
```




```
ggplot(model.spline.df) + geom_qq(aes(sample=.resid))
```



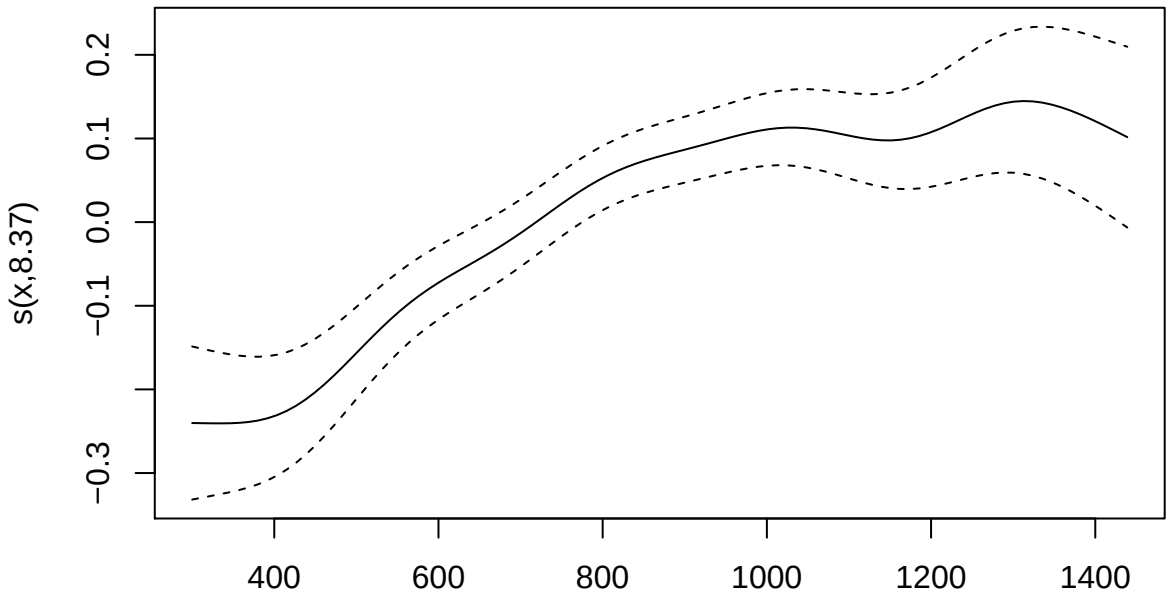

```
time2 = Sys.time()
difftime(time2, time1)
```

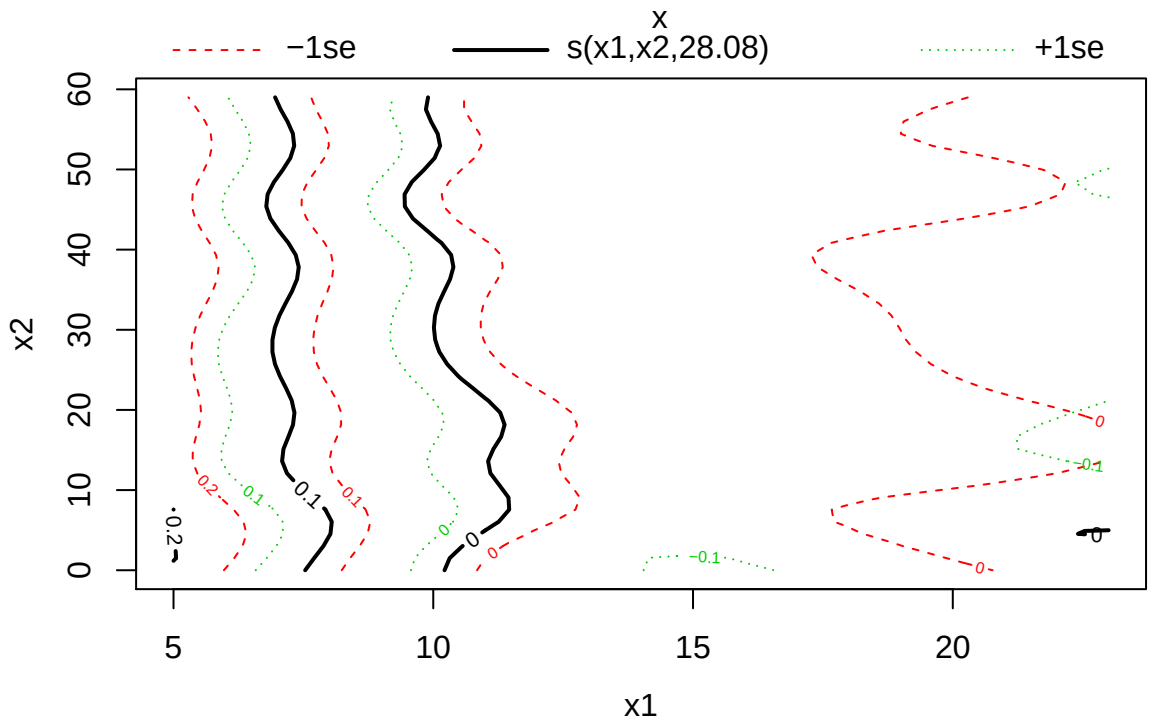
```
## Time difference of 8.416074 secs
```

Model C: gam() with library(mgcv)

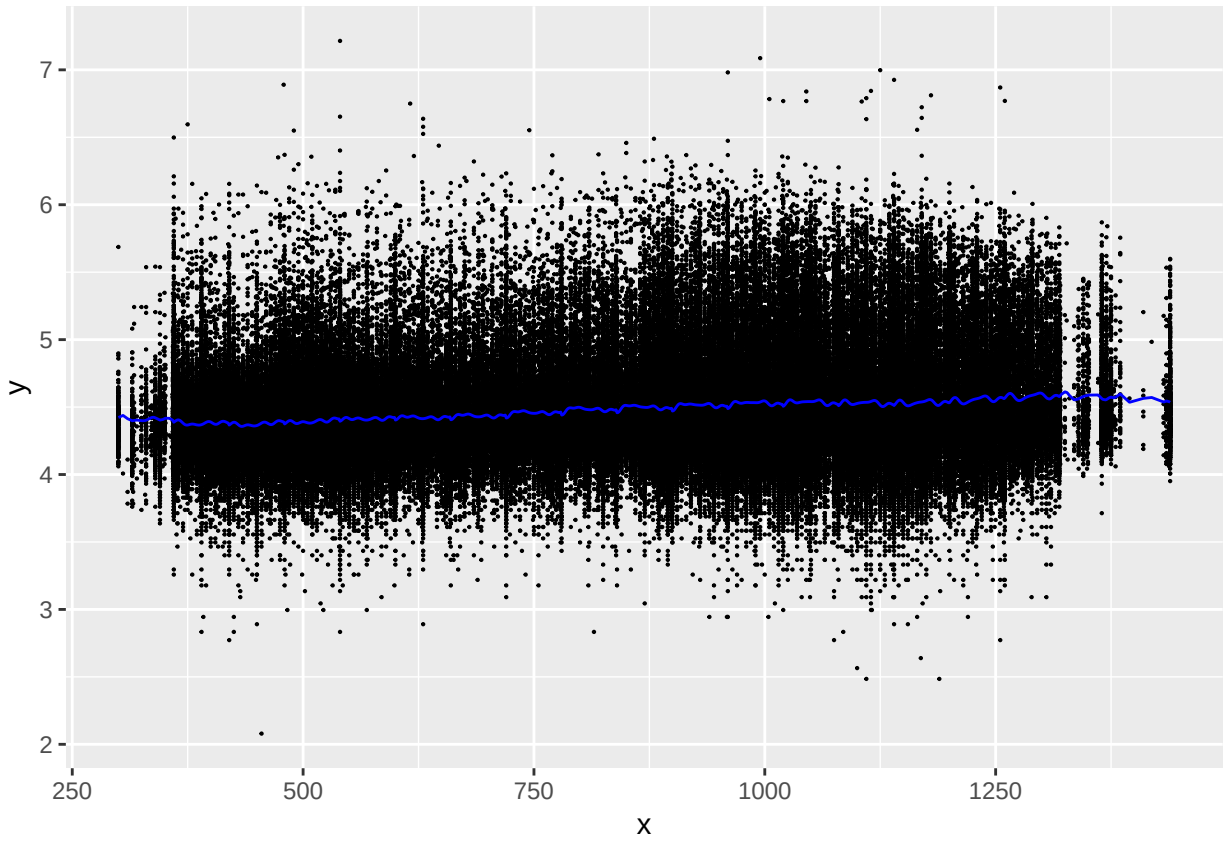
For fit with gam I decided to explore the relationship between hour and minutes of departure. Remember: x = total minutes after midnight; x1 = hours and x2 = minutes. I experimented with fitting s(x), s(x1) + s(x2), s(x) + s(x1, x2), and s(x1, x2). All models showed choppy fits. I assumed beyond the scope of problem 1 to do cross-validation for parameter tunning, so I settled with the model s(x) + s(x1, x2) which, in theory, uses all the information available from explanatory variables, but do not duplicate them. I did that knowing that the model may be overfitting the data more than the previous ones (since plot of fitted values is rather choppy).

```
time1 = Sys.time()
model.gam = gam(y ~ s(x) + s(x1, x2), data=train.df)
plot(model.gam)
```

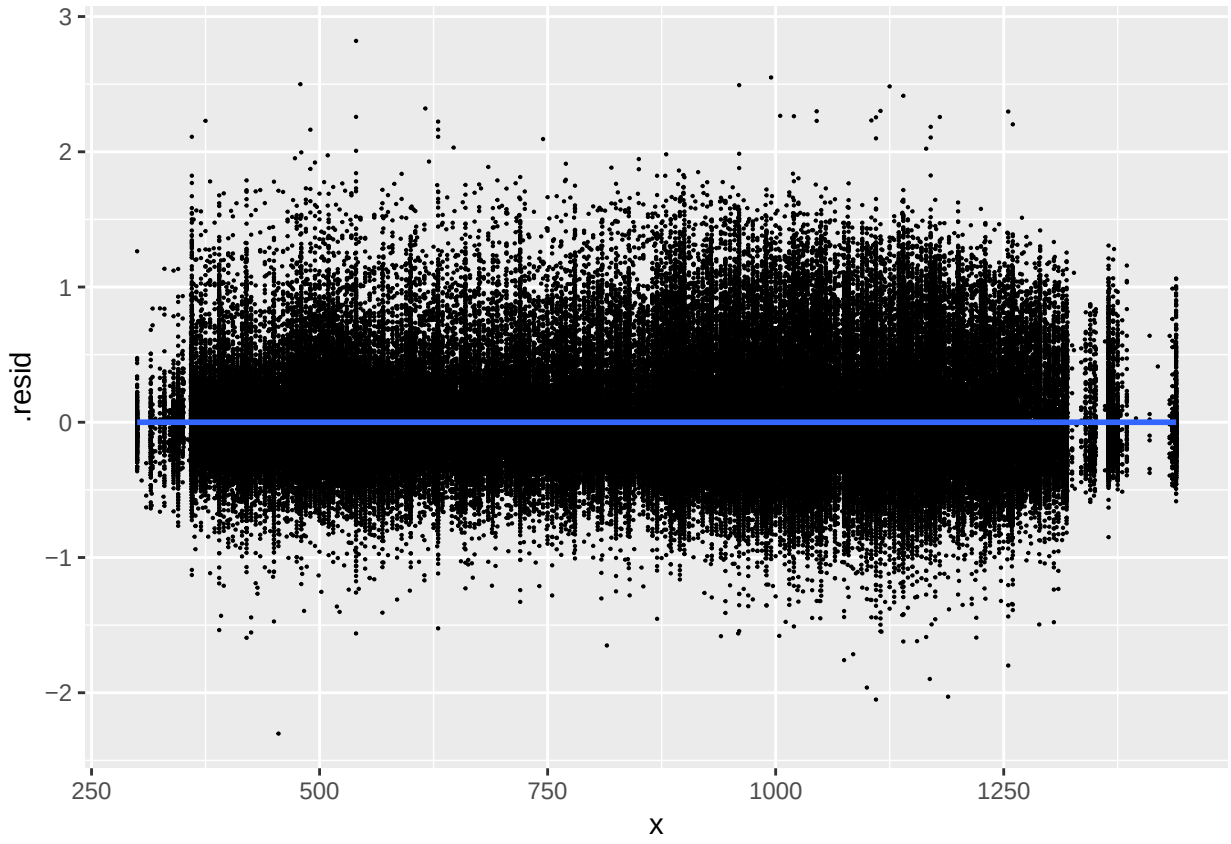




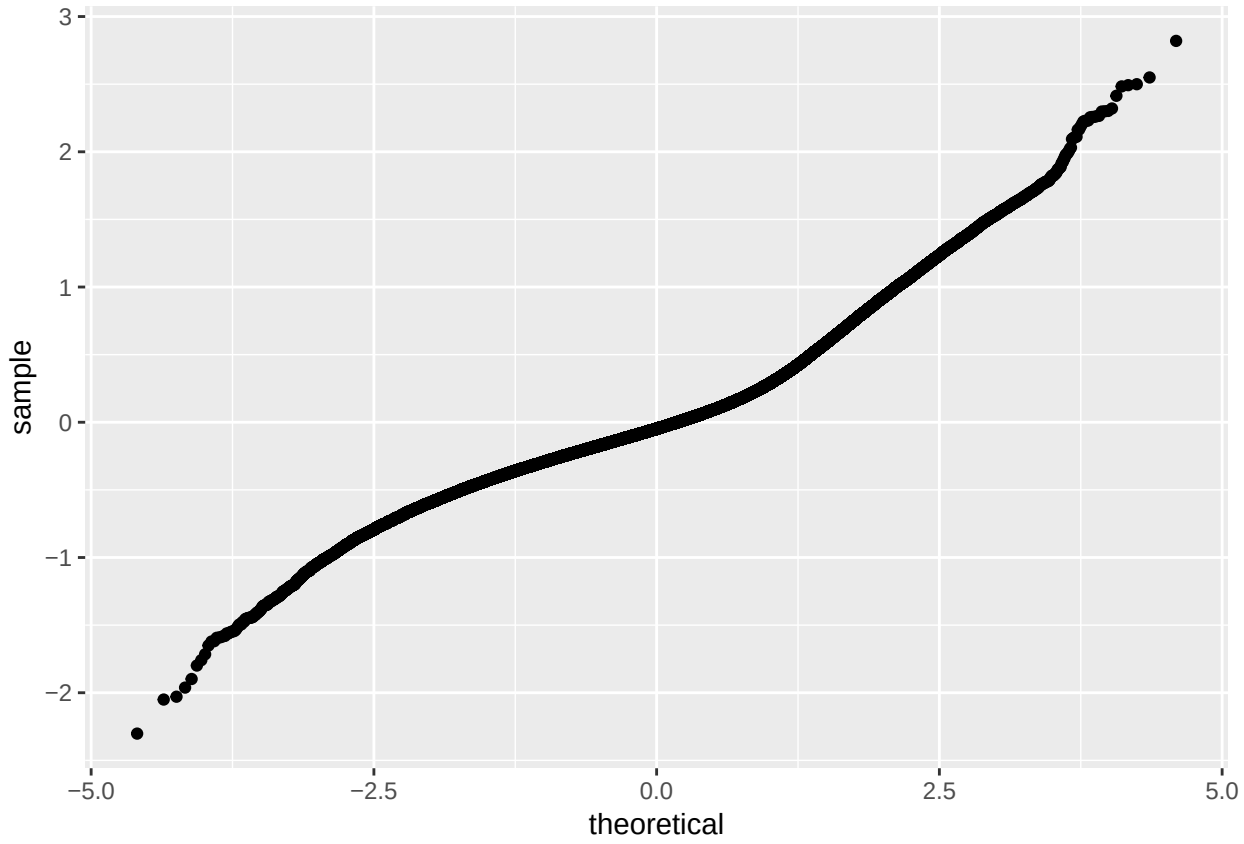

```
model.gam.df = augment(model.gam)
ggplot(model.gam.df, aes(x=x, y=y)) + geom_point(size=0.1) + geom_line(aes(x=x, y=.fitted), color='blue
```




```
gg = ggplot(model.gam.df, aes(x=x, y=.resid)) + geom_point(size=0.1)
gg = gg + geom_smooth(method = "gam", formula = y ~ s(x))
gg
```




```
ggplot(model.gam.df) + geom_qq(aes(sample=.resid))
```

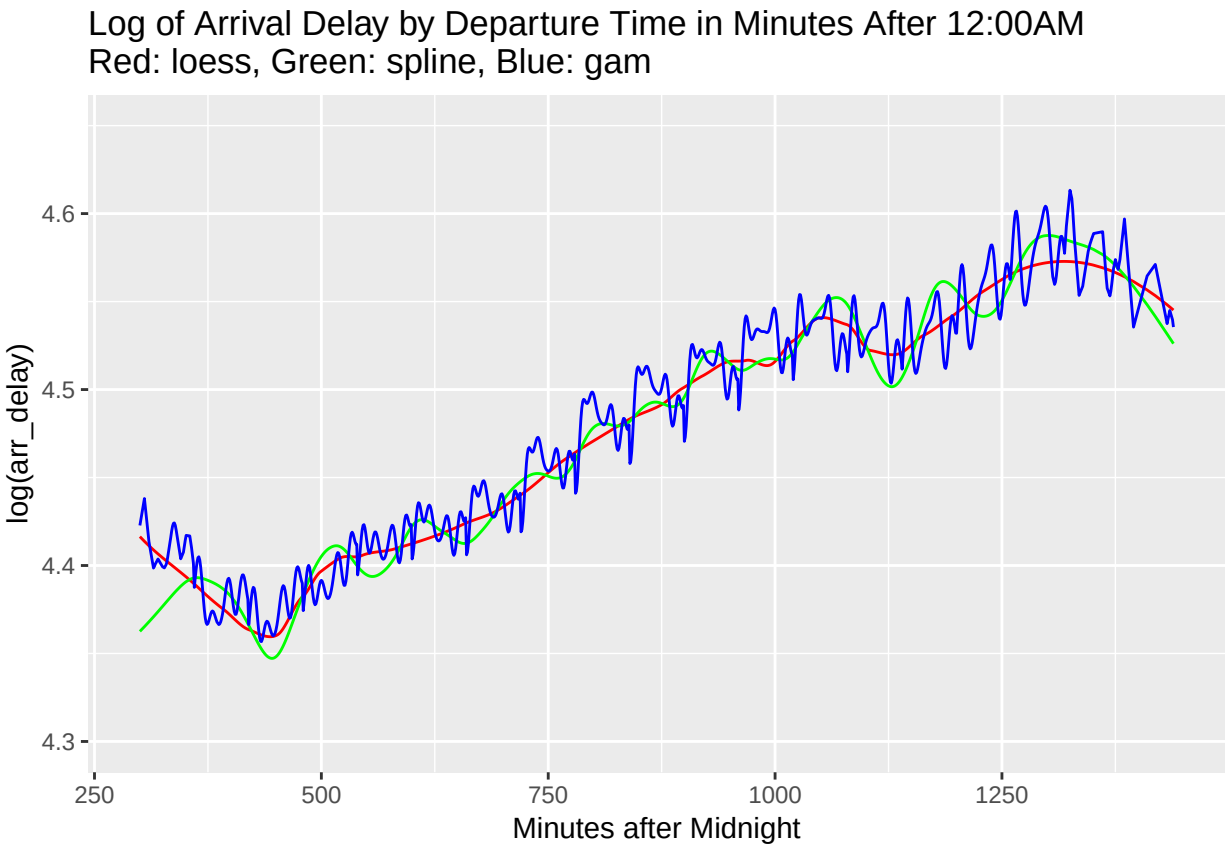



```
time2 = Sys.time()
difftime(time2, time1)
```

```
## Time difference of 26.16414 secs
```

(d) Plots

```
# prepare datasets
gg = ggplot(df, aes(x=x, y=y)) #+ geom_point(size=0.1, alpha=0.1)
gg = gg + geom_line(aes(y=.fitted), color='red', data=model.loess.df)
gg = gg + geom_line(aes(y=.fitted), colour='green', data=model.spline.df)
gg = gg + geom_line(aes(x=x, y=.fitted), colour='blue', data=model.gam.df)
gg = gg + ylab('log(arr_delay)') + xlab('Minutes after Midnight') + ylim(4.3, 4.65)  #ylim(3, 7)
gg = gg + ggtitle("Log of Arrival Delay by Departure Time in Minutes After 12:00AM\nRed: loess, Green: s
gg
```



(e) Model selection

Below we fit our models to the test dataset and select the best model based on the RMSE metric. The models yield surprisingly similar results, but if I had to pick one it would be the spline model, which produces a slighly lower RMSE on the test dataset.

```
get_RMSE = function(y, y_hat) {
  return(sqrt(mean((y - y_hat)^2)))
}
# prepare vectors for predition
x = test.df$x
y = test.df$y
x1 = test.df$x1
x2 = test.df$x2
y_hat.loess = predict(model.loess, data.frame(x))
y_hat.spline = predict(model.spline, x)$y
y_hat.gam = predict(model.gam, data.frame(x1, x2))
```

```
## Warning in predict.gam(model.gam, data.frame(x1, x2)): not all required variables have been supplied
```

```
RMSE.loess = get_RMSE(y, y_hat.loess)
RMSE.spline = get_RMSE(y, y_hat.spline)
RMSE.gam = get_RMSE(y, y_hat.gam)
RMSE.loess
```

```
## [1] 0.3501749
```

```
RMSE.spline
```

```
## [1] 0.3499875
```

```
RMSE.gam
```

```
## [1] 0.3500503
```

```
# RMSE in minutes
exp(RMSE.loess)
```

```
## [1] 1.419316
```

```
exp(RMSE.spline)
```

```
## [1] 1.41905
```

```
exp(RMSE.gam)
```

```
## [1] 1.419139
```

Problem 2 - Predictions on class dataset

EDA

If running in dev mode, read dataset, print basic dataset info and plot.

```
set.seed(681)
# read dataset, but only if running in development
if (Sys.info()[6] == 'carlos2') {
  print('Carlos running, so read training dataset...')
  training = read.table('s681-sp19-training.txt',header=T)
  x.train = training$x
  y.train = training$y
} else {
  print('Skipping reading training dataset...')
}
```

```
## [1] "Carlos running, so read training dataset..."
```

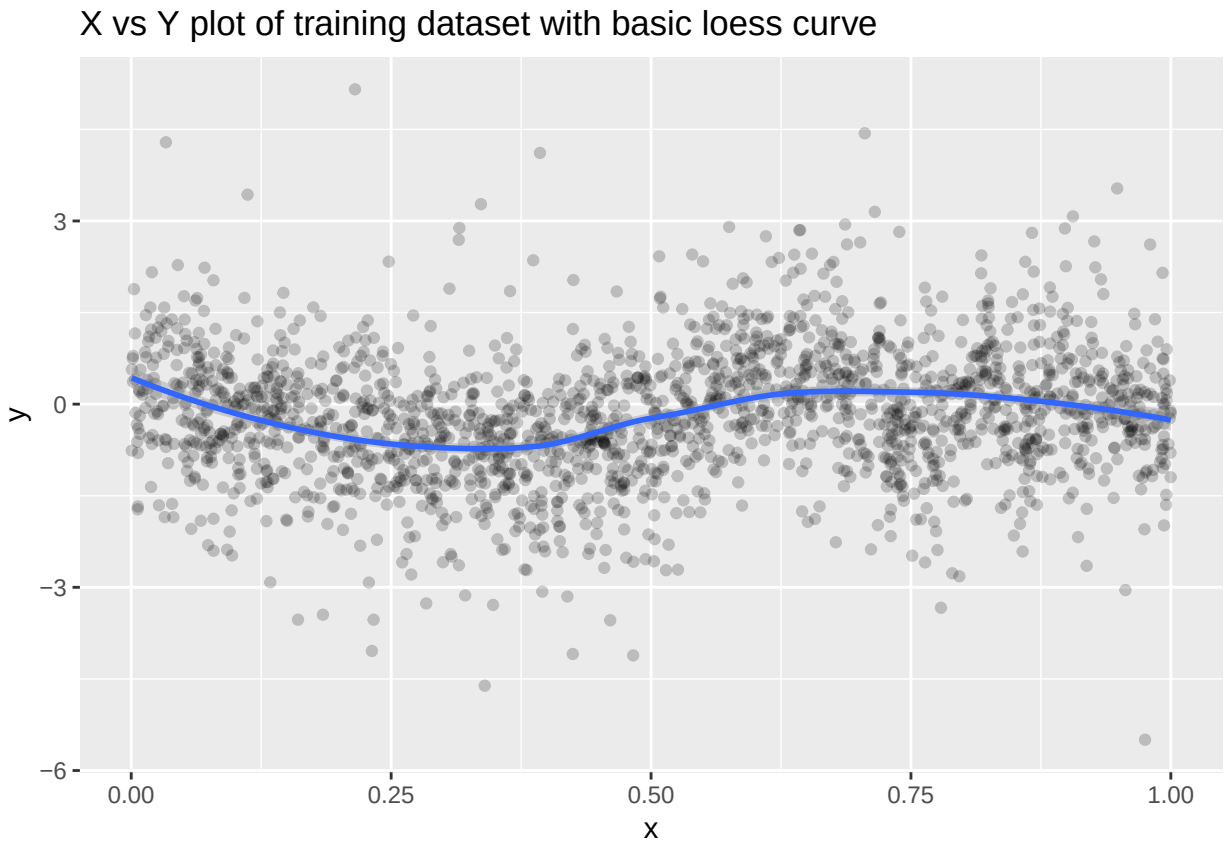
```
summary(training)
```

```
##        x                    y
##  Min.   :0.0005537   Min.   :-5.4955
##  1st Qu.:0.2629169   1st Qu.:-0.7988
##  Median :0.5037265   Median :-0.1651
##  Mean   :0.5052153   Mean   :-0.1788
##  3rd Qu.:0.7491512   3rd Qu.: 0.4353
##  Max.   :0.9999478   Max.   : 5.1557
```

```
str(training)
```

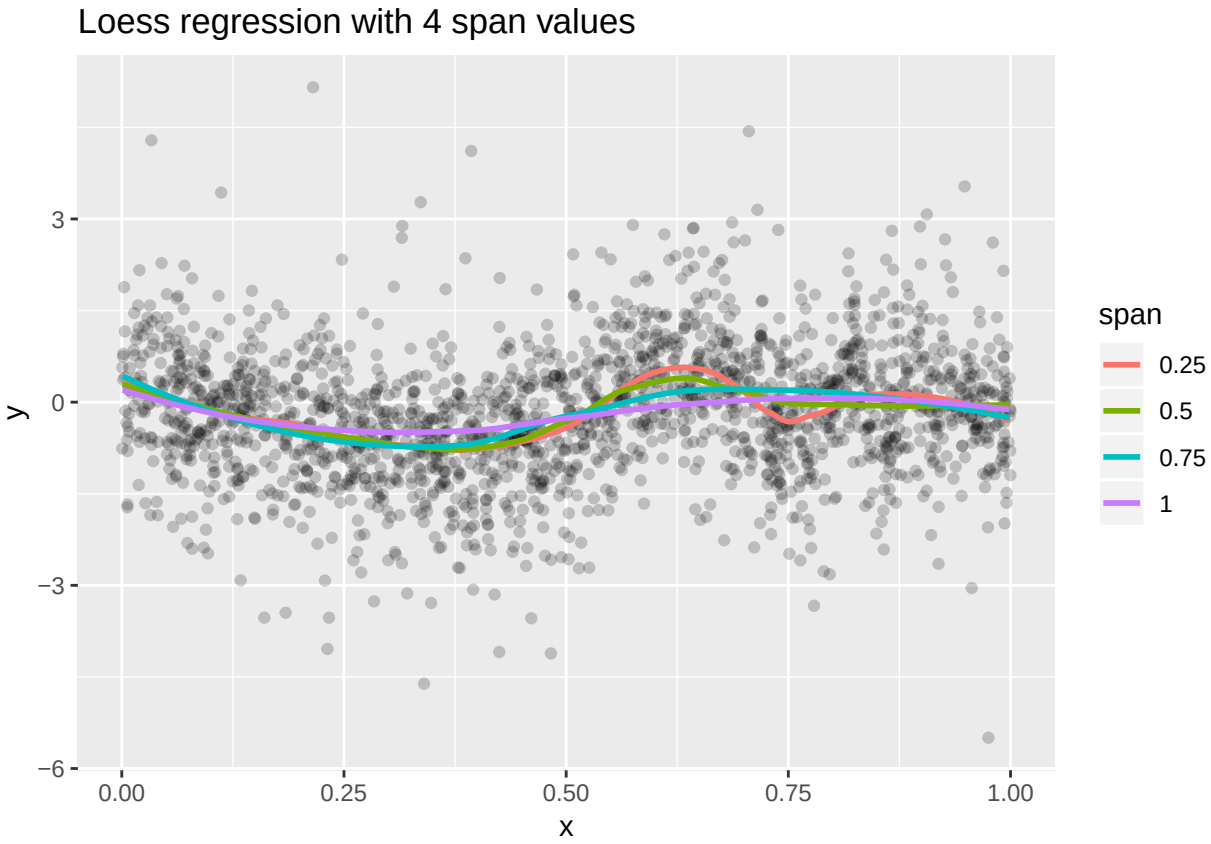
```
## 'data.frame':    2000 obs. of  2 variables:
##  $ x: num  0.2883 0.5211 0.4906 0.0442 0.2369 ...
##  $ y: num  -1.719 0.991 -0.938 1.243 -0.889 ...
```

```
gg = ggplot(training, aes(x=x, y=y)) + geom_point(alpha=0.2) + geom_smooth(method='loess')
gg + ggtitle("X vs Y plot of training dataset with basic loess curve")
```



Of the methods we learned in class, tuning loess span seems like a reasonable next step to better fit the sinuidal pattern oberved with the ggplot loess smoother. We check for 4 span values.

```
set.seed(681)
x.test = seq(0.001, 0.999, 0.001)
gg = ggplot(training, aes(x=x, y=y)) + geom_point(alpha=0.2, show.legend = FALSE)
model4 = loess(y.train ~ x.train, span=0.25)
model5 = loess(y.train ~ x.train, span=0.5)
model6 = loess(y.train ~ x.train, span=0.75)
model7 = loess(y.train ~ x.train, span=1)
fit4 = predict(model4, x.test)
fit5 = predict(model5, x.test)
fit6 = predict(model6, x.test)
fit7 = predict(model7, x.test)
plot.df = data.frame(x=rep(x.test,4), y=c(fit4, fit5, fit6, fit7),
                     span=as.character(rep(c(0.25, 0.5, 0.75, 1), each = length(x.test))))
gg = gg + geom_line(data=plot.df, aes(x=x, y=y, group=span, color=span), size=1)
gg + ggtitle("Loess regression with 4 span values")
```

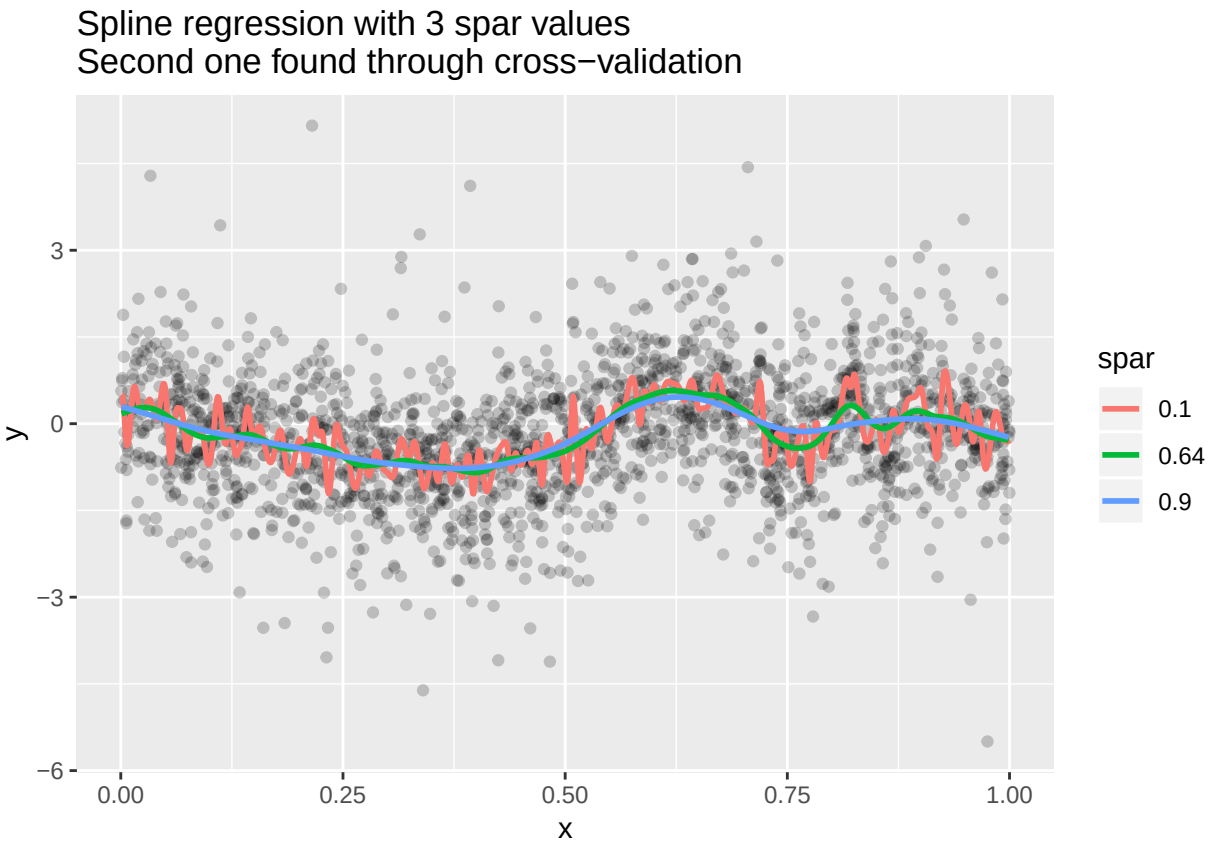


Low span loess seem to suggest r(x) could be a damped sine wave (https://en.wikipedia.org/wiki/Damped_sine_wave). Can we get similar results with penalized regression, specifically spline?

```
set.seed(681)
x.test = seq(0.001, 0.999, 0.001)
gg = ggplot(training, aes(x=x, y=y)) + geom_point(alpha=0.2, show.legend = FALSE)
model1 = smooth.spline(x.train, y.train, spar=0.1)
model2 = smooth.spline(x.train, y.train, cv=TRUE)
```

```
## Warning in smooth.spline(x.train, y.train, cv = TRUE): cross-validation
## with non-unique 'x' values seems doubtful
```

```
spar2 = round(model2$spar,2)
model3 = smooth.spline(x.train, y.train, spar=0.9)
fit1 = predict(model1, x.test)$y
fit2 = predict(model2, x.test)$y
fit3 = predict(model3, x.test)$y
plot.df = data.frame(x=rep(x.test,3), y=c(fit1, fit2, fit3),
                     spar=as.character(rep(c(0.1, spar2, 0.9), each = length(x.test))))
gg = gg + geom_line(data=plot.df, aes(x=x, y=y, group=spar, color=spar), size=1)
gg + ggtitle("Spline regression with 3 spar values\nSecond one found through cross-validation")
```



Low spar will clearly lead to overfitting, but spar tuned with cross validation confirms r(x) could well be a damped sinuidal.

Note that, of the methods we learned in class, we won't use gam because we're estimating a single function.

Model selection with cross-validation

I will train loess regression using cross validation to find best span parameter. I will train for each parameter value 10 times on 9 different fold sets, leaving one fold out for validation each time, and will store the mean MSE from the 10 validation runs for comparison later on.

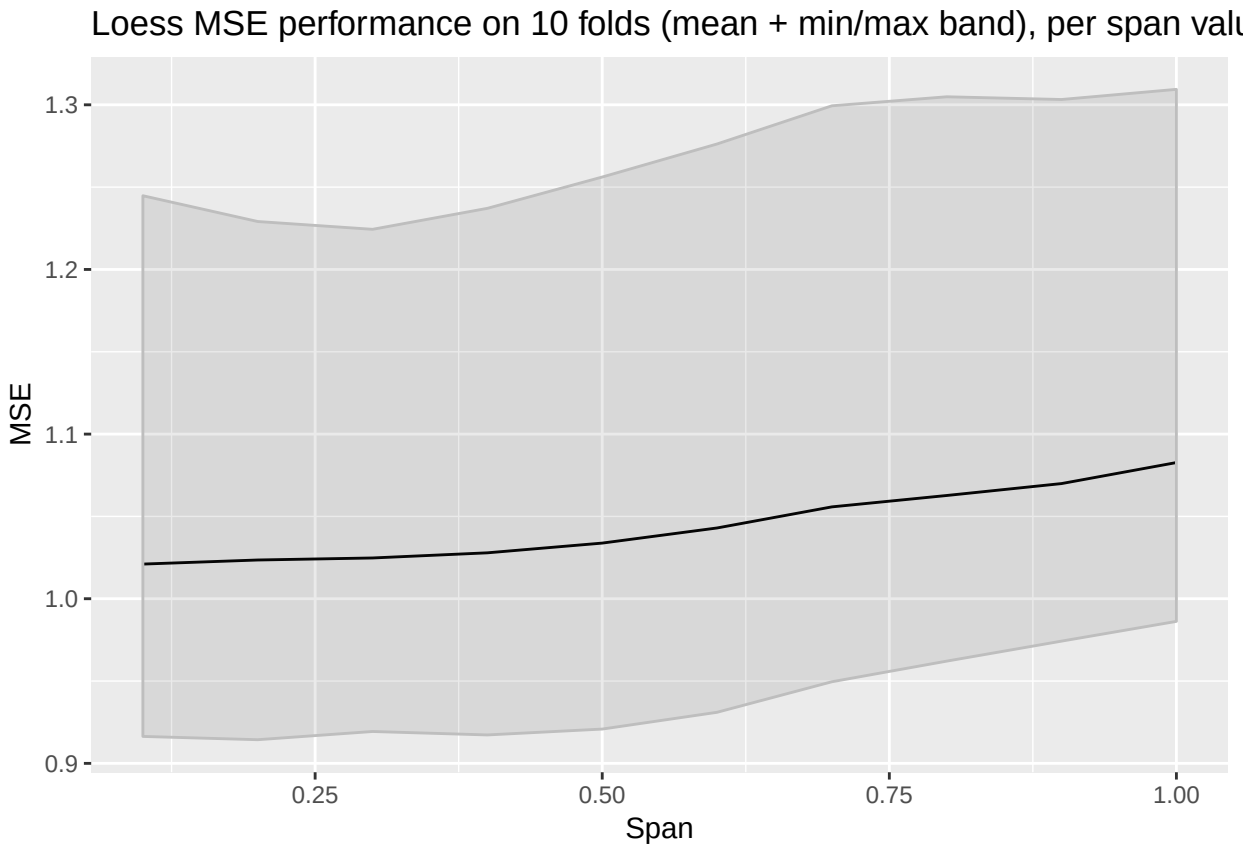
After I find the best loess tuning, I will compare performance against spline with spar = 0.64, found through cross-validation available with smooth.spline method.

```
set.seed(681)
# create tunning parms
spans = seq(0.1, 1, 0.1)
# create dataframe to track loess results per tunning parm
folds = 10
foldMSEs.loess = data.frame(matrix(NA, folds, length(spans)))
colnames(foldMSEs.loess) = spans
# create vector to track spline results (spar = 0.64) for each fold
foldMSEs.spline = c(rep(NA,10))
# randomly create indexes to map n/folds rows of data to n folders
case.folds = sample(rep(1:folds, nrow(training)/folds))
# this should output 2000/10=200 records
sum(case.folds==1)
```

```
## [1] 200
```

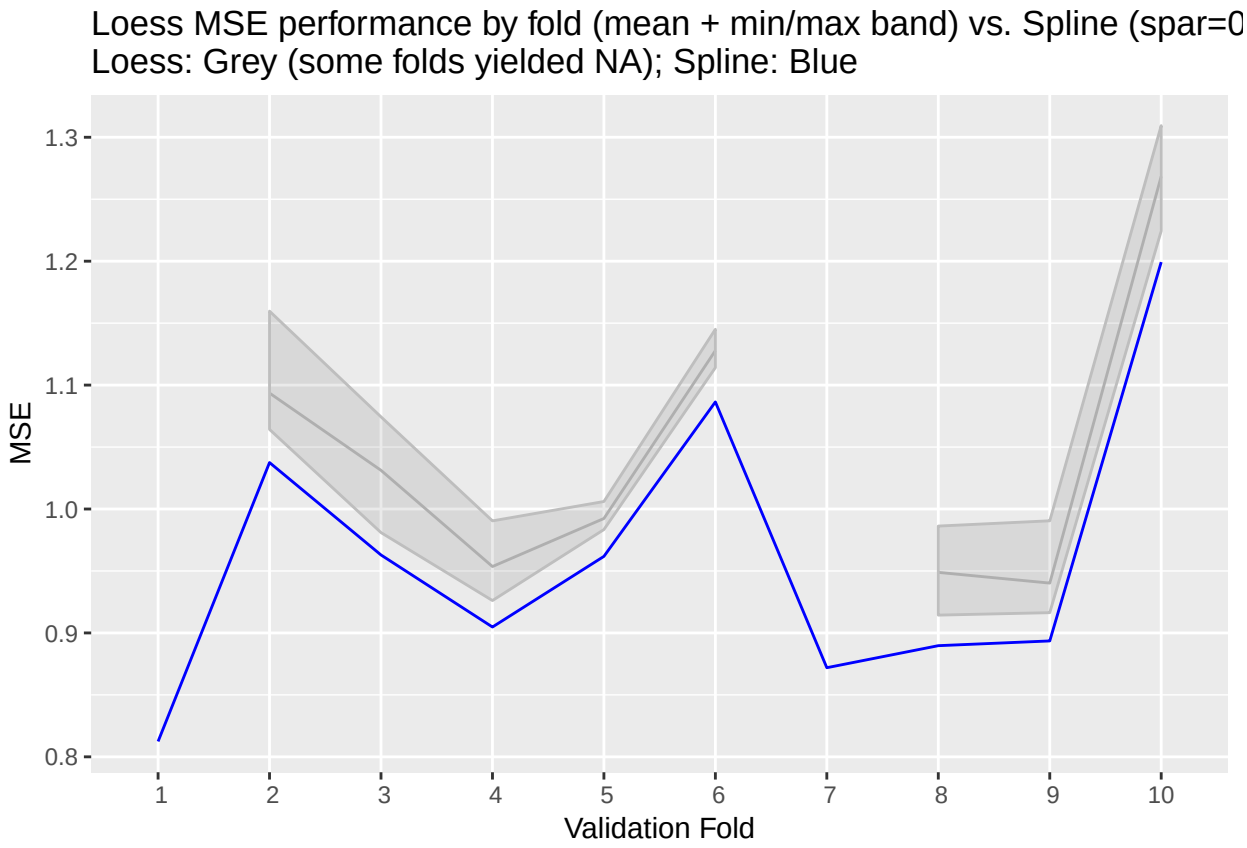
```
# traverse folds
for(fold in 1:folds){
  # validation partition is defined by fold number
  valid.rows = which(case.folds == fold)
  valid.data = training[valid.rows, ]
  # the other nine folds will be used for training
  train.data = training[-valid.rows, ]
  for(span in spans) {

    model.loess = loess(train.data$y ~ train.data$x, span=span)
    preds.loess = predict(model.loess, valid.data$x)
    foldMSEs.loess[fold, paste(span)] = mean((valid.data$y - preds.loess)^2)

    model.spline = smooth.spline(x.train, y.train, spar=0.64)
    preds.spline = predict(model.spline, valid.data$x)$y
    foldMSEs.spline[fold] = mean((valid.data$y - preds.spline)^2)

  }
}
```

```
set.seed(681)
# get loess mse for each span parameter (mean across folds) along with bands
cv.mse.mean = colMeans(na.omit(foldMSEs.loess))
cv.mse.min = apply(na.omit(foldMSEs.loess), 2, min)
cv.mse.max = apply(na.omit(foldMSEs.loess), 2, max)
MSE.df = data.frame(spans, cv.mse.min, cv.mse.mean, cv.mse.max)
gg = ggplot(MSE.df, aes(x=spans, y=cv.mse.mean)) + geom_line() + ylab('MSE') + xlab('Span')
gg = gg + geom_ribbon(aes(ymin=cv.mse.min, ymax=cv.mse.max), alpha=0.1, color='grey')
gg + ggtitle("Loess MSE performance on 10 folds (mean + min/max band), per span value")
```



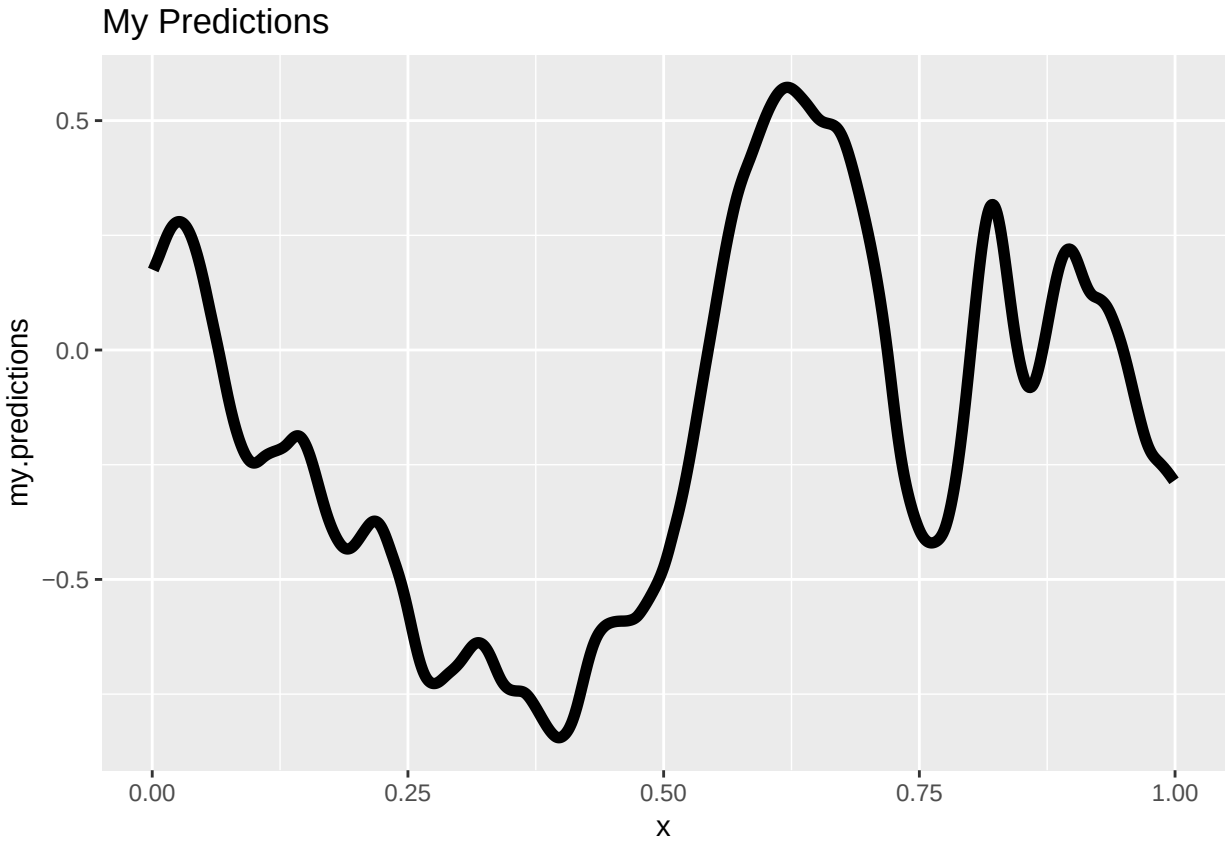

```
# get loess mse for each folder (mean across across spans) along with bands
# and compare with spline performance
cv.mse.mean = rowMeans(foldMSEs.loess)
cv.mse.min = apply(foldMSEs.loess, 1, min)
cv.mse.max = apply(foldMSEs.loess, 1, max)
MSE.df = data.frame(fold=1:10, cv.mse.min, cv.mse.mean, cv.mse.max, spline=foldMSEs.spline)
gg = ggplot(MSE.df, aes(fold, spline)) + geom_line(color='blue') + ylab("MSE") + xlab('Validation Fold')
gg = gg + geom_line(aes(y=cv.mse.mean), color='grey')
gg = gg + geom_ribbon(aes(ymin=cv.mse.min, ymax=cv.mse.max), alpha=0.1, color='grey')
gg = gg + scale_x_discrete(limits=c(1:10))
gg + ggtitle("Loess MSE performance by fold (mean + min/max band) vs. Spline (spar=0.64)\nLoess: Grey (s
```

```
## Warning: Removed 1 rows containing missing values (geom_path).
```



Clearly regulirized regression (spline with spar=0.64) outperforms loess. We will generate model in .R file using spline with spar=0.64 for instructor to run against test dataset. Final model representing r(x) estimate is included below, along with plot of predictions for specified test set.

```
set.seed(681)
x.test = seq(0.001, 0.999, 0.001)
r_x_estimate = smooth.spline(x.train, y.train, spar=0.64)
my.predictions = predict(r_x_estimate, x.test)$y
df = data.frame(x=x.test, my.predictions)
ggplot(df, aes(x=x, y=my.predictions)) + geom_line(size=2) + ggtitle("My Predictions")
```



In conclusion, I started with EDA of the training dataset. The plot with basic loess smoother suggested a sinuidal pattern to the data. Of the methods taught in class kernel regression was discarded because both loess and spline are improvements to kernel regression; gam was discarded because the problem states y is obtained from a single function of x: r(x). Therefore, I chose local regression (loess) and regularized regression (spline) to model r(x). I compared performance of loess and spline on the training dataset. I used 10 fold cross validation to tune loess span, and I used the cv parameter in smooth.spline to find the optimum spar value for the spline regression. I then compared results and spline clearly outperformed loess on the training dataset, so I used it for my final model to generate my.predictions.

Problem 3 - Prediction on fossils dataset

EDA

For EDA we include variable ln_mass_change = ln_mass - ln_old_mass which will be used in part b and c of Problem 3.

```
training = read.table('fossils.txt', header = T)
# add column for change in ln_mass for question 3.b
training$ln_mass_change = training$ln_mass - training$ln_old_mass
summary(training)
```

```
##                   species        ln_mass        ln_old_mass
##  Aaptoryctes_ivyi      :   1   Min.   : 0.000   Min.   : 0.000
##  Abelmoschomys_simpsoni:   1   1st Qu.: 4.170   1st Qu.: 4.041
##  Absarokius_abbotti    :   1   Median : 6.454   Median : 6.227
##  Absarokius_australis  :   1   Mean   : 6.953   Mean   : 6.779
```

```
##  Absarokius_gazini     :   1   3rd Qu.: 9.671   3rd Qu.: 9.417
##  Absarokius_metoecus   :   1   Max.   :15.983   Max.   :15.311
##  (Other)               :2028                    NA's   :925
##  first_appear_Mya last_appear_Mya ln_mass_change
##  Min.   : 0.00    Min.   : 0.00   Min.   :-2.8350
##  1st Qu.:14.30    1st Qu.:13.90   1st Qu.:-0.3430
##  Median :34.55    Median :33.70   Median : 0.1020
##  Mean   :33.77    Mean   :32.89   Mean   : 0.1087
##  3rd Qu.:53.80    3rd Qu.:52.70   3rd Qu.: 0.5500
##  Max.   :99.40    Max.   :98.10   Max.   : 2.9420
##                   NA's   :70      NA's   :925
```

```
str(training)
```

```
## 'data.frame':    2034 obs. of  6 variables:
##  $ species         : Factor w/ 2034 levels "Aaptoryctes_ivyi",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ ln_mass         : num  4.83 2.77 6.02 6.1 5.74 ...
##  $ ln_old_mass     : num  NA NA 5.96 5.96 5.96 ...
##  $ first_appear_Mya: num  57.2 9.9 53.7 49.9 53.5 53.7 50 55.4 45.6 14.5 ...
##  $ last_appear_Mya : num  56.8 9.9 50.7 49.8 53.4 49.8 50 54.8 44.7 14.3 ...
##  $ ln_mass_change  : num  NA NA 0.066 0.147 -0.222 ...
```

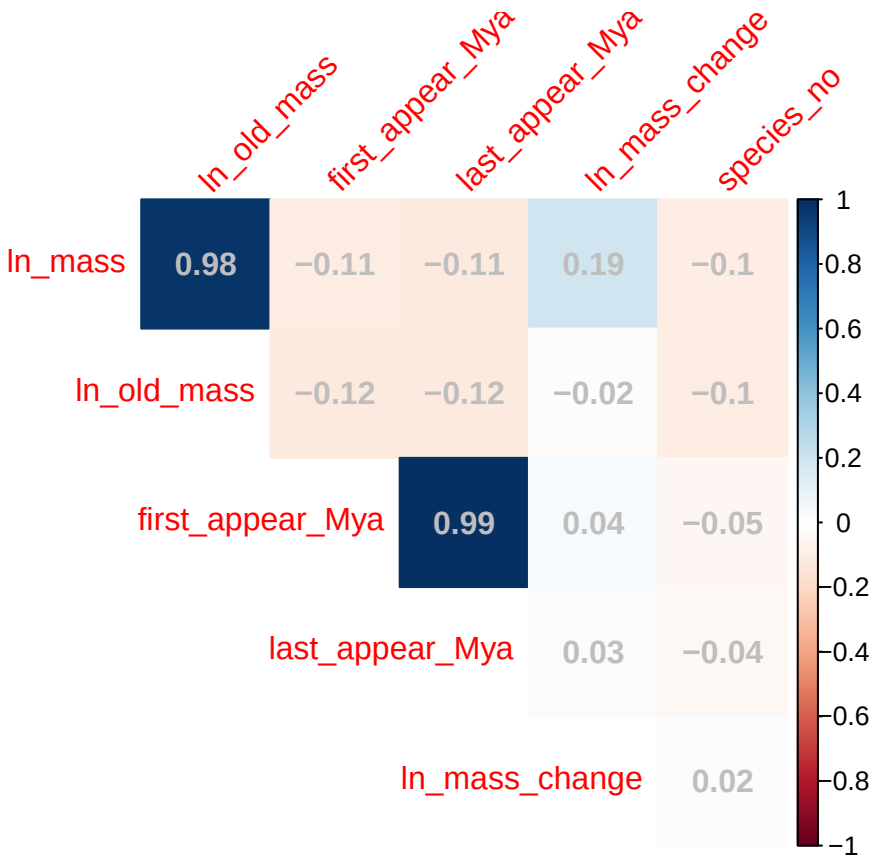
```
# find na's
apply(is.na(training), 2, sum)
```

```
##          species          ln_mass      ln_old_mass first_appear_Mya
##                0                0              925                0
##  last_appear_Mya   ln_mass_change
##               70              925
```

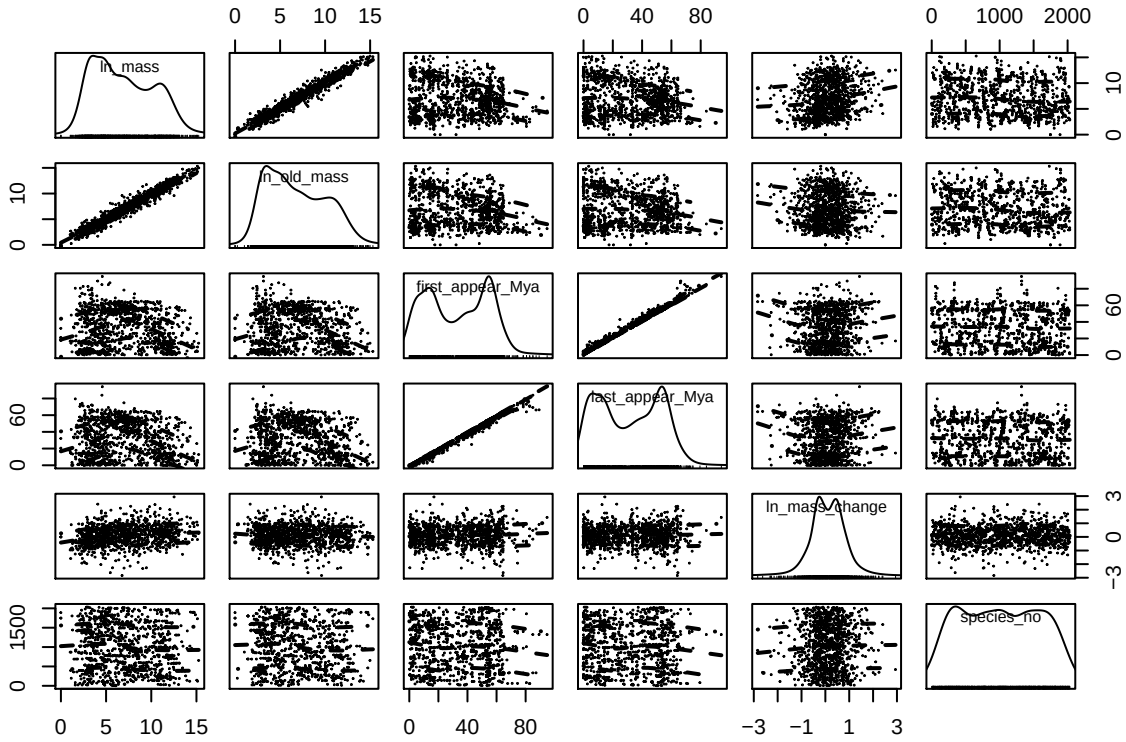
```
# find correlations
training$species_no = as.numeric(training$species)
cols = names(training)
cor = cor(na.omit(training[,cols[!cols=='species']]))
cor
```

```
##                     ln_mass ln_old_mass first_appear_Mya last_appear_Mya
## ln_mass           1.00000000  0.97704707       -0.1066489     -0.11161358
## ln_old_mass       0.97704707  1.00000000       -0.1163804     -0.11978660
## first_appear_Mya -0.10664889 -0.11638042        1.0000000      0.99411218
## last_appear_Mya  -0.11161358 -0.11978660        0.9941122      1.00000000
## ln_mass_change    0.19381718 -0.01961561        0.0354196      0.02780470
## species_no       -0.09553227 -0.10277548       -0.0468454     -0.03836252
##                  ln_mass_change  species_no
## ln_mass              0.19381718 -0.09553227
## ln_old_mass         -0.01961561 -0.10277548
## first_appear_Mya     0.03541960 -0.04684540
## last_appear_Mya      0.02780470 -0.03836252
## ln_mass_change       1.00000000  0.02493965
## species_no           0.02493965  1.00000000
```

```
# print pair plots
corrplot(cor, method='color', type='upper', addCoef.col = "grey", diag=F, tl.srt=45)
```




```
scatterplotMatrix(na.omit(training[,cols[!cols=='species']]), cex=0.1, regLine=F, col='black')
```



(a) Fitting gam model with ln_mass as response variable

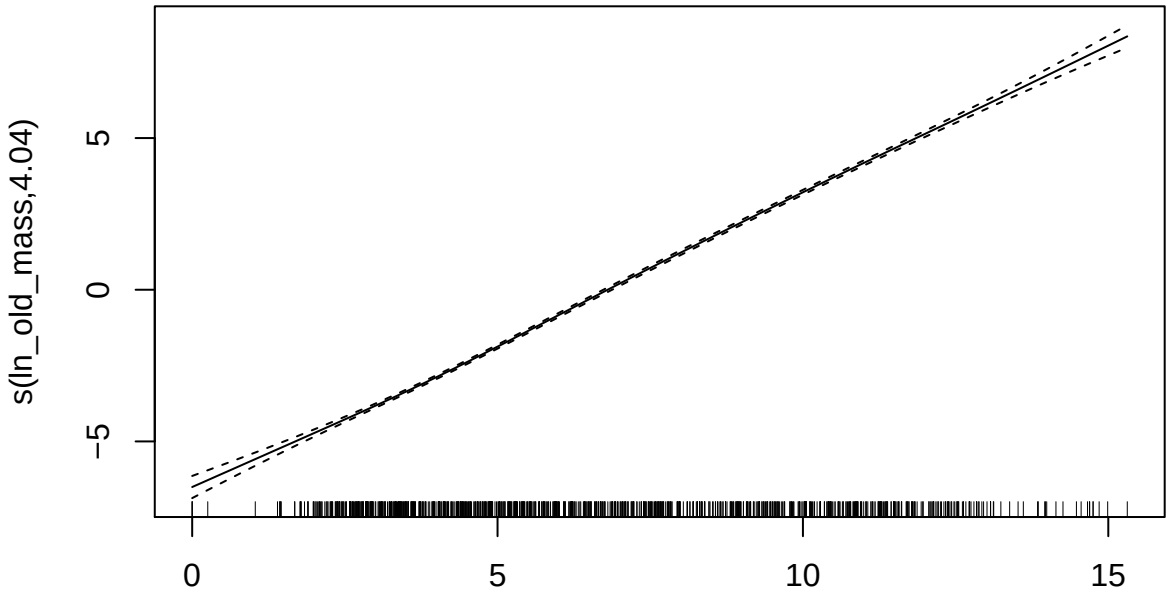
Below I fit a number of models using different sets of explanatory variables and compare the result of each model using adjusted r-squared. The model with maximum adjusted r-squared is selected. I do not perform residual analysis since it doesn't appear to be a requirement of the question. R-squared was chosen as a metric for model selection for convenience reasons only.

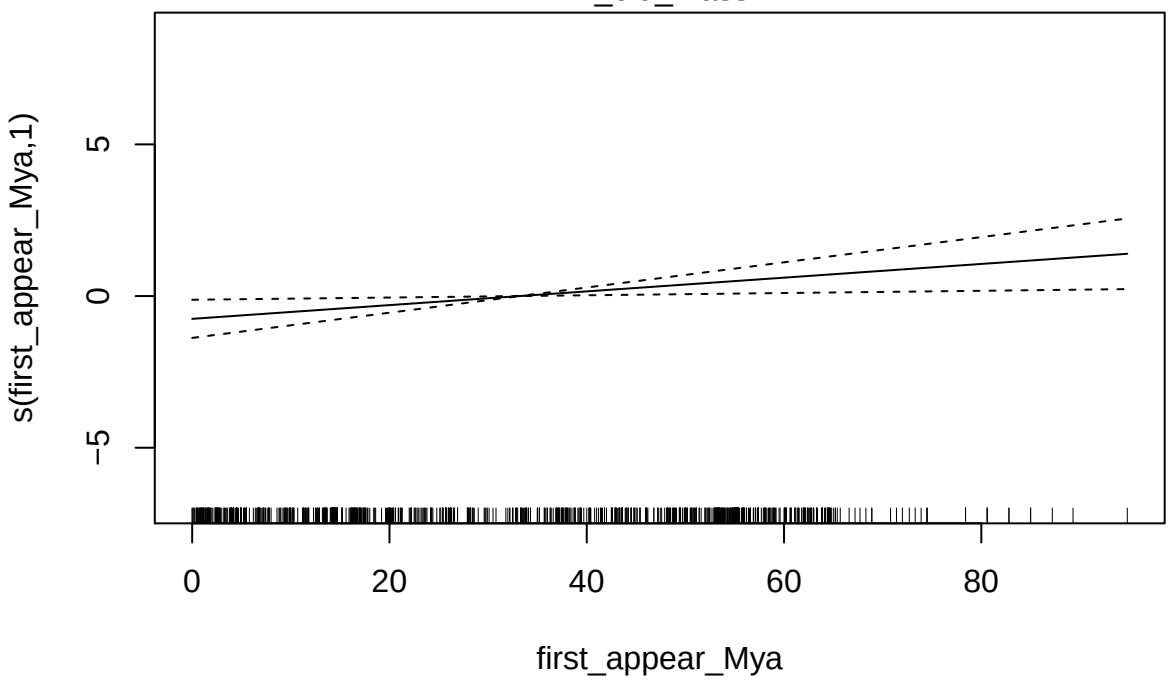
```
model = list()
# variations on all vars, ln_old_mass is always included by itself
model[[1]] = gam(ln_mass ~ s(ln_old_mass) + s(first_appear_Mya) + s(last_appear_Mya) + s(species_no), da
model[[2]] = gam(ln_mass ~ s(ln_old_mass) + s(first_appear_Mya, last_appear_Mya, species_no), data=trai
model[[3]] = gam(ln_mass ~ s(ln_old_mass) + s(first_appear_Mya, last_appear_Mya) + s(species_no), data=
model[[4]] = gam(ln_mass ~ s(ln_old_mass) + s(first_appear_Mya) + s(last_appear_Mya, species_no), data=
model[[5]] = gam(ln_mass ~ s(ln_old_mass) + s(last_appear_Mya) + s(first_appear_Mya, species_no), data=
# variations on 3 vars, ln_old_mass is always included by itself
model[[6]] = gam(ln_mass ~ s(ln_old_mass) + s(first_appear_Mya) + s(last_appear_Mya), data=training)
model[[7]] = gam(ln_mass ~ s(ln_old_mass) + s(first_appear_Mya, last_appear_Mya), data=training)
model[[8]] = gam(ln_mass ~ s(ln_old_mass) + s(first_appear_Mya) + s(species_no), data=training)
model[[9]] = gam(ln_mass ~ s(ln_old_mass) + s(first_appear_Mya, species_no), data=training)
model[[10]] = gam(ln_mass ~ s(ln_old_mass) + s(species_no) + s(last_appear_Mya), data=training)
model[[11]] = gam(ln_mass ~ s(ln_old_mass) + s(species_no, last_appear_Mya), data=training)
# variations on 2 vars, ln_old_mass is always included by itself
model[[12]] = gam(ln_mass ~ s(ln_old_mass) + s(first_appear_Mya), data=training)
model[[13]] = gam(ln_mass ~ s(ln_old_mass) + s(last_appear_Mya), data=training)
model[[14]] = gam(ln_mass ~ s(ln_old_mass) + s(species_no), data=training)
# ln_old_mass by itself
model[[15]] = gam(ln_mass ~ s(ln_old_mass), data=training)
r.squared = c(rep(NA, 15))
for (i in 1:length(model)) {
  smry = summary(model[[i]])
  r.squared[i] = smry$r.sq
}
# get idx for best model
best.model.a.idx = which.max(r.squared)
best.model.a = model[[best.model.a.idx]]
summary(best.model.a)
```

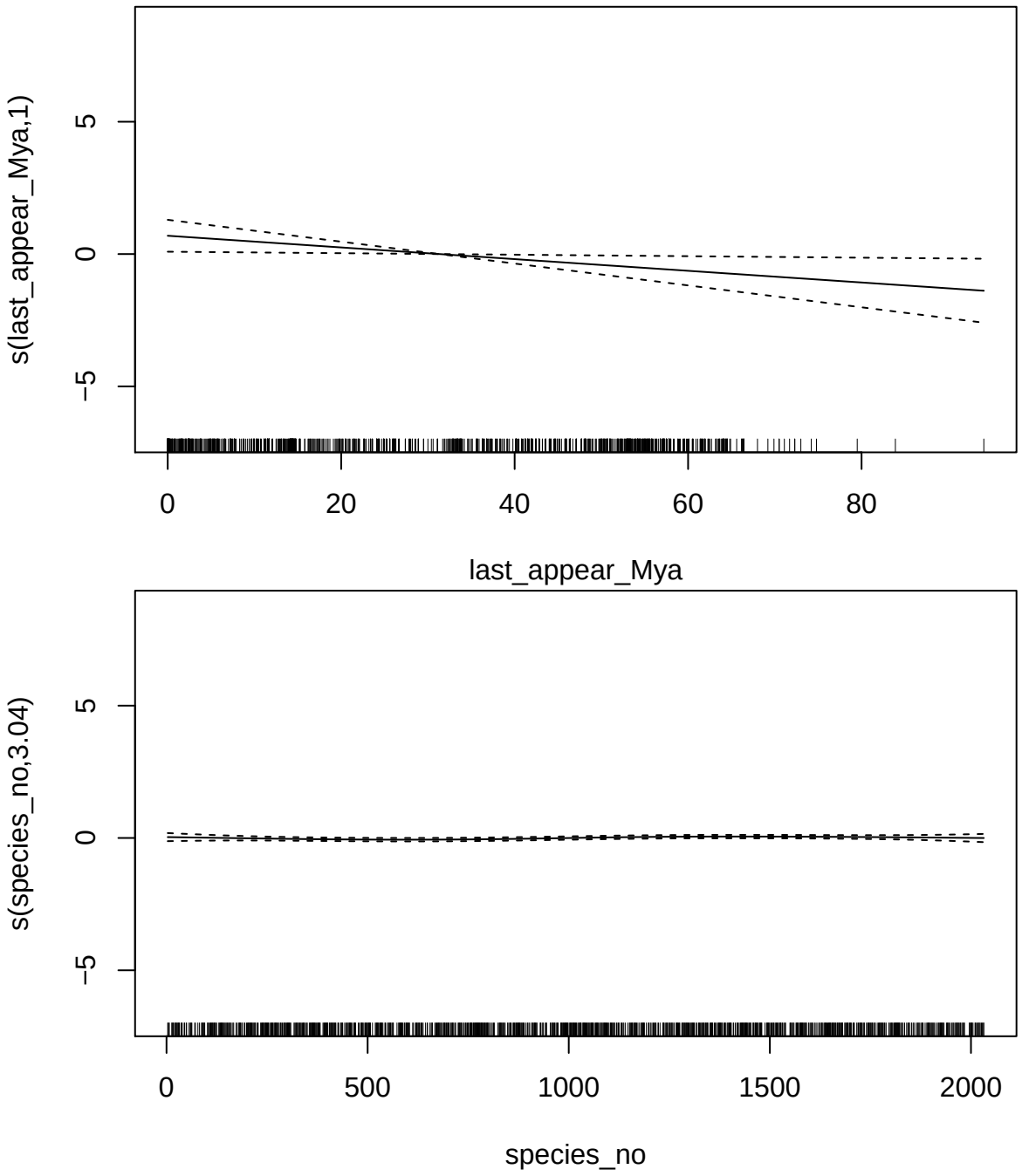
```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## ln_mass ~ s(ln_old_mass) + s(first_appear_Mya) + s(last_appear_Mya) +
##     s(species_no)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.93426    0.02189   316.7   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                        edf Ref.df        F p-value
## s(ln_old_mass)       4.042  5.068 4329.134  <2e-16 ***
## s(first_appear_Mya) 1.000  1.000    5.720  0.0169 *
```

```
## s(last_appear_Mya)  1.000  1.000    5.262  0.0220 *
## s(species_no)       3.039  3.783    1.357  0.2857
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.955   Deviance explained = 95.6%
## GCV = 0.51442  Scale est. = 0.50955   n = 1063
```

```
plot.gam(best.model.a)
```







(b) Change in log mass, ln_mass - ln_old_mass, is response

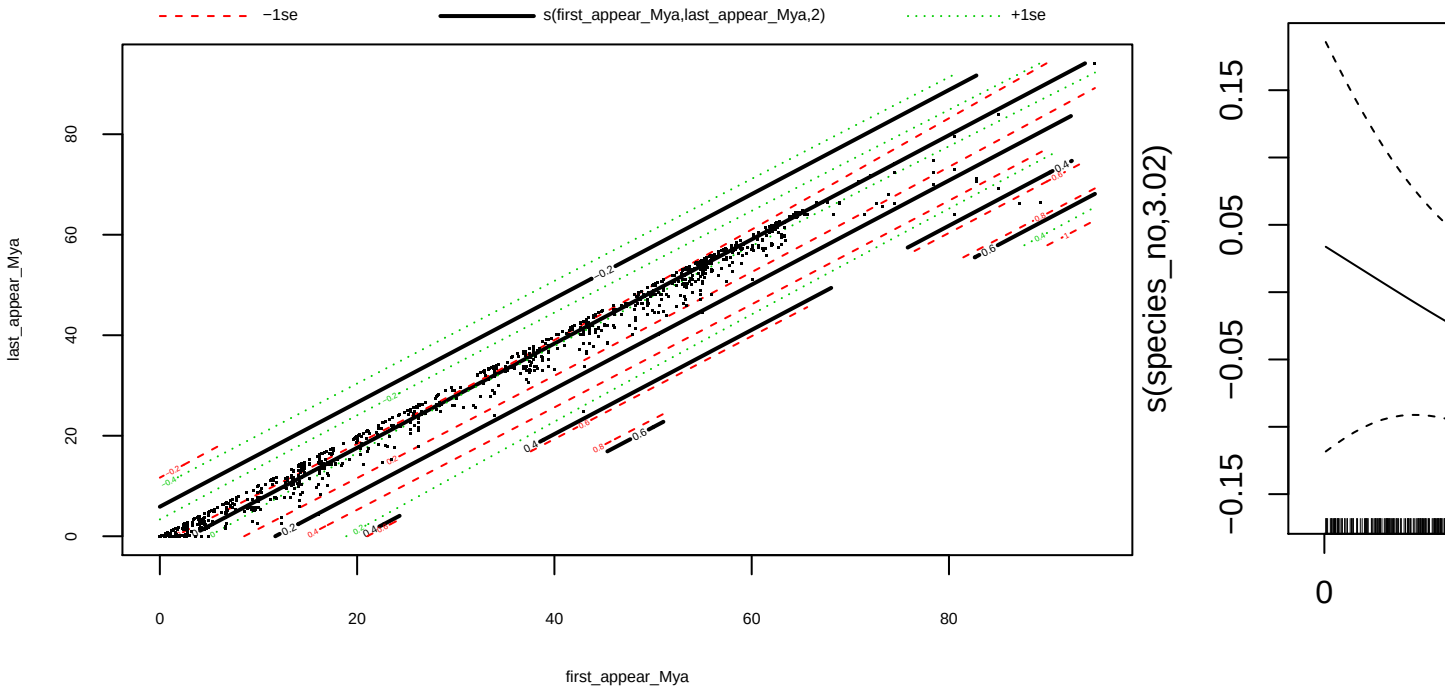
Below I repeat the procedure used in part (a), i.e., I compare a number of gam models using different sets of explanatory variables and use r-squared to select the "best" model.

```
model = list()
# variations on 3 vars
model[[1]] = gam(ln_mass_change ~ s(first_appear_Mya) + s(last_appear_Mya) + s(species_no), data=traini
model[[2]] = gam(ln_mass_change ~ s(first_appear_Mya, last_appear_Mya, species_no), data=training)
model[[3]] = gam(ln_mass_change ~ s(first_appear_Mya, last_appear_Mya) + s(species_no), data=training)
```

```
model[[4]] = gam(ln_mass_change ~ s(first_appear_Mya) + s(last_appear_Mya, species_no), data=training)
model[[5]] = gam(ln_mass_change ~ s(last_appear_Mya) + s(first_appear_Mya, species_no), data=training)
# variations on 2 vars
model[[5]] = gam(ln_mass_change ~ s(first_appear_Mya) + s(last_appear_Mya), data=training)
model[[6]] = gam(ln_mass_change ~ s(first_appear_Mya, last_appear_Mya), data=training)
model[[7]] = gam(ln_mass_change ~ s(first_appear_Mya) + s(species_no), data=training)
model[[8]] = gam(ln_mass_change ~ s(first_appear_Mya, species_no), data=training)
model[[9]] = gam(ln_mass_change ~ s(species_no) + s(last_appear_Mya), data=training)
model[[10]] = gam(ln_mass_change ~ s(species_no, last_appear_Mya), data=training)
# variations on 1 var
model[[11]] = gam(ln_mass_change ~ s(first_appear_Mya), data=training)
model[[12]] = gam(ln_mass_change ~ s(last_appear_Mya), data=training)
model[[13]] = gam(ln_mass_change ~ s(species_no), data=training)
#
r.squared = c(rep(NA, 13))
for (i in 1:length(model)) {
  smry = summary(model[[i]])
  r.squared[i] = smry$r.sq
}
# get idx for best model
best.model.b.idx = which.max(r.squared)
best.model.b = model[[best.model.b.idx]]
summary(best.model.b)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## ln_mass_change ~ s(first_appear_Mya, last_appear_Mya) + s(species_no)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.10999    0.02196   5.008 6.43e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                       edf Ref.df     F p-value
## s(first_appear_Mya,last_appear_Mya) 2.000  2.000 3.422   0.033 *
## s(species_no)                       3.017  3.759 1.169   0.291
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.00724   Deviance explained = 1.19%
## GCV = 0.51556  Scale est. = 0.51264   n = 1063
```

```
plot.gam(best.model.b)
```

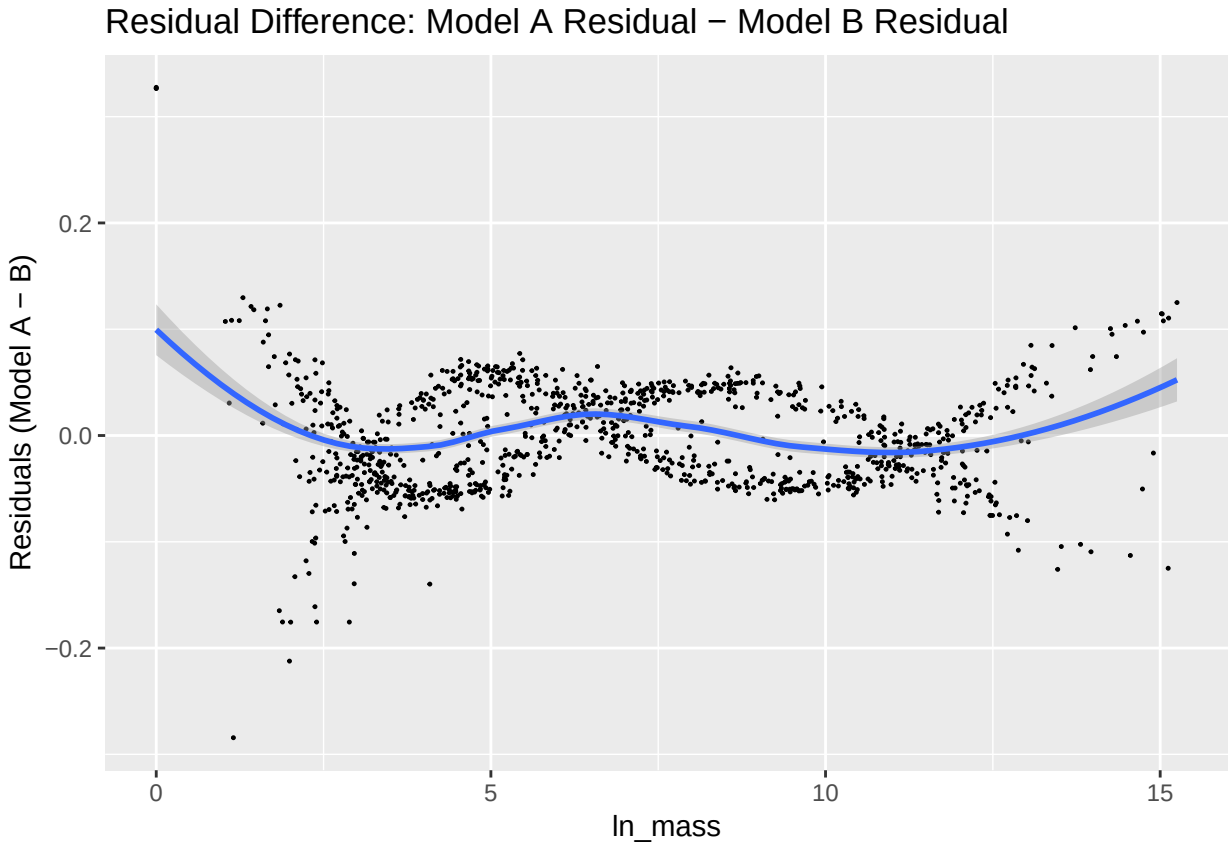


(c) Interpreting and comparing the models

The first model directly predicts the response variable, while the second one does that indirectly. Because the correlation between ln_mass and ln_old_mass is so high, I thought using the difference of log mass as the response variable might improve the relevance of other explanatory variables in the model. That did not happen, since both models selected (A and B) had the same variables included. The only difference is that model B used the interaction of first and last appearance variables. My results suggest the models are equivalent, on average.

Below I compare the predictions from the 2 best models selected in parts (a) and (b), Model A and Model B, respectively, by analysing the plot of residuals obtained from their predictions.

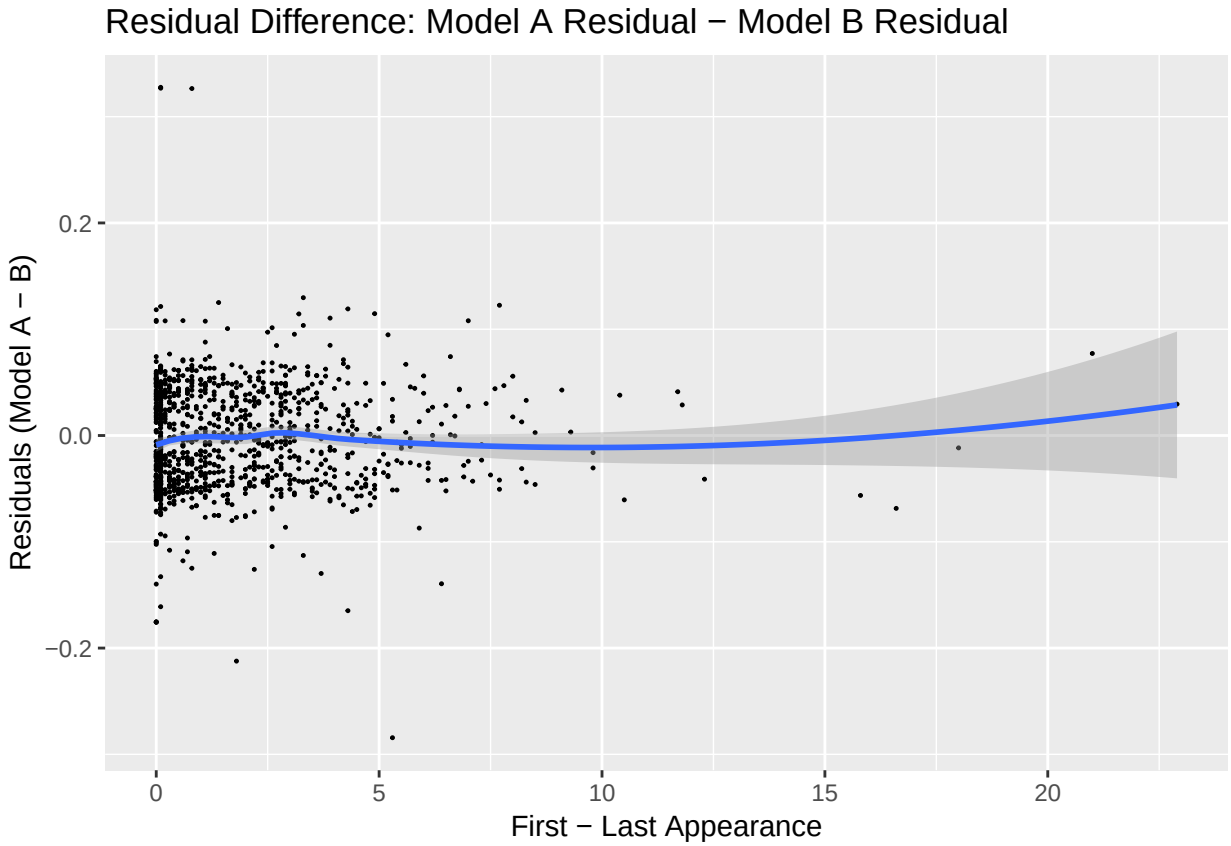
```
# augment best model for 3.a and 3.b, rename columns, and combine in single data frame
model.a.df = augment(best.model.a)
model.b.df = augment(best.model.b)
df = data.frame(x = model.a.df$ln_mass,
                diff_appear_Mya = model.a.df$first_appear_Mya - model.a.df$last_appear_Mya,
                species_no = model.b.df$species_no,
                a.resid = abs(model.a.df$.resid),
                b.resid = abs(model.a.df$ln_mass - (model.b.df$.fitted + model.a.df$ln_old_mass)))
df$abs.resid.diff = df$a.resid - df$b.resid
# print difference only
gg = ggplot(df, aes(x=x, y=abs.resid.diff)) + geom_point(size=0.2) + geom_smooth(method='loess')
gg = gg + xlab('ln_mass') + ylab('Residuals (Model A - B)')
gg = gg + ggtitle('Residual Difference: Model A Residual - Model B Residual')
gg
```



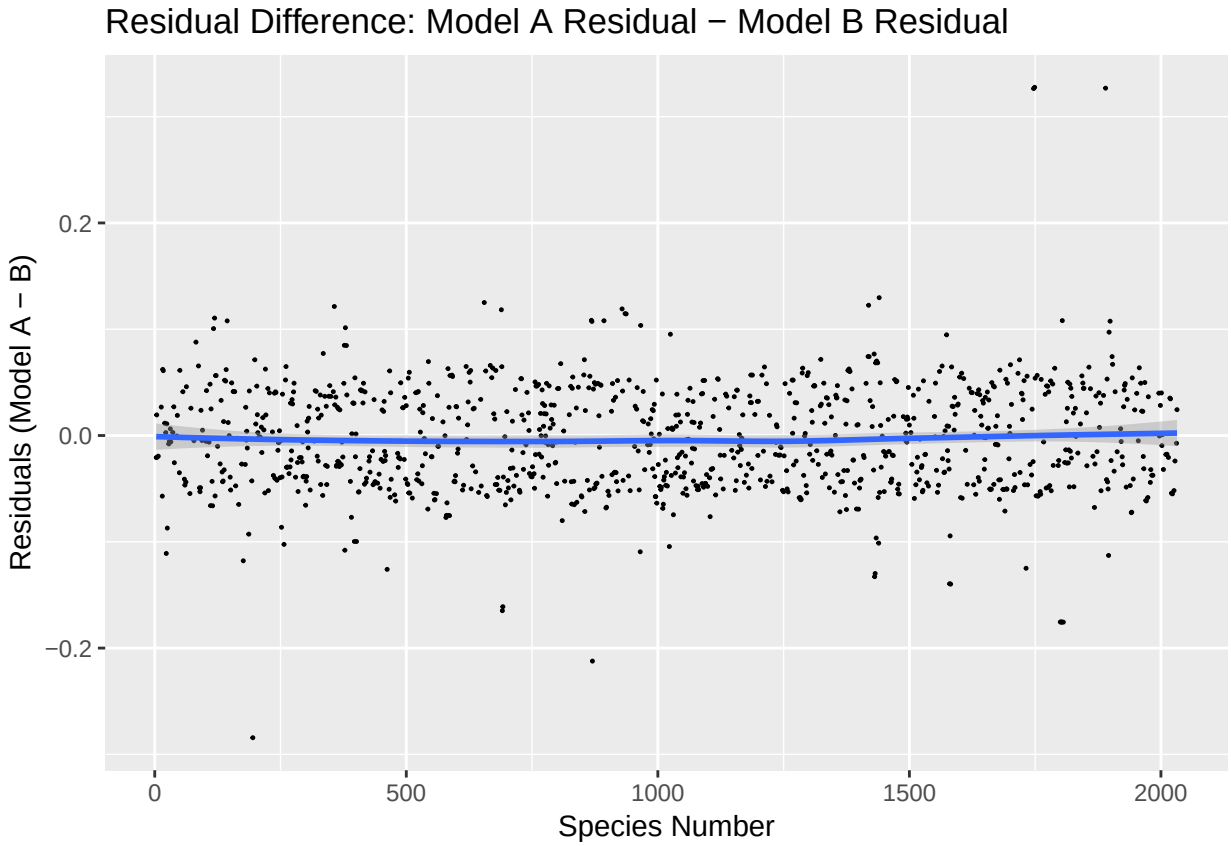
As stated before, the plot suggests the models are mostly equivalent for ln_mass between 2.5 and 12.5. Model (b) outperforms model (a), on avarage, for predictions of extreme values of ln_mass, both high and low.

However, in order to predict values of ln_mass on a test dataset, we need to avaluate the models with respect to how they perform relative to values of explanatory variables. A contour plot showing residuals as a function of first and last appearance did not provide much insight on what model would perform best based on appearance variables. The two plots below show the models produce comparable results regardless of values of explanatory variables.

```
gg = ggplot(df, aes(x=diff_appear_Mya, y=abs.resid.diff))
gg = gg + geom_point(size=0.2) + geom_smooth(method='loess')
gg = gg + xlab('First - Last Appearance') + ylab('Residuals (Model A - B)')
gg = gg + ggtitle('Residual Difference: Model A Residual - Model B Residual')
gg
```




```
gg = ggplot(df, aes(x=species_no, y=abs.resid.diff))
gg = gg + geom_point(size=0.2) + geom_smooth(method='loess')
gg = gg + xlab('Species Number') + ylab('Residuals (Model A - B)')
gg = gg + ggtitle('Residual Difference: Model A Residual - Model B Residual')
gg
```


Residual Difference: Model A Residual − Model B Residual
Residuals (Model A − B)
0.2
0.0
−0.2
0
500
1000
1500
2000
Species Number