STAT 844 Final Project

UW21011995 Sohoon Youn

1.Data Description

1-1. Variables

This dataset contains 37 features, including `los`, `age`, `gender`, `ethnicity`, `religion`, `insurance` `admission_type`, `admission_location`, `first_hosp_stay`, `diagnosis`, `los_reg_trs`, `aniongap_min`, `aniongap_max`, `bicarbonate_min`, `bicarbonate_max`, `chloride_min`, `chloride_max`, `height`, `weight`, `hematocrit_min`, `hematocrit_max`, `hemoglobin_min`, `hemoglobin_max`, `platelet_min`, `platelet_max`, `bilirubin_max`, `sodium_max`, `potassium_max`, `heartrate_min`, `heartrate_max`, and `heartrate_mean`.

name	p.val
ethnicity(black)	0.0233710
religion	0.0228320
aniongap_min	0.0006960
bicarbonate_min	0.0127080
bicarbonate_max	0.0002090
chloride_min	0.0000000
chloride_max	0.0000001
height	0.0000000
weight	0.0000000
heartrate_mean	0.0000391

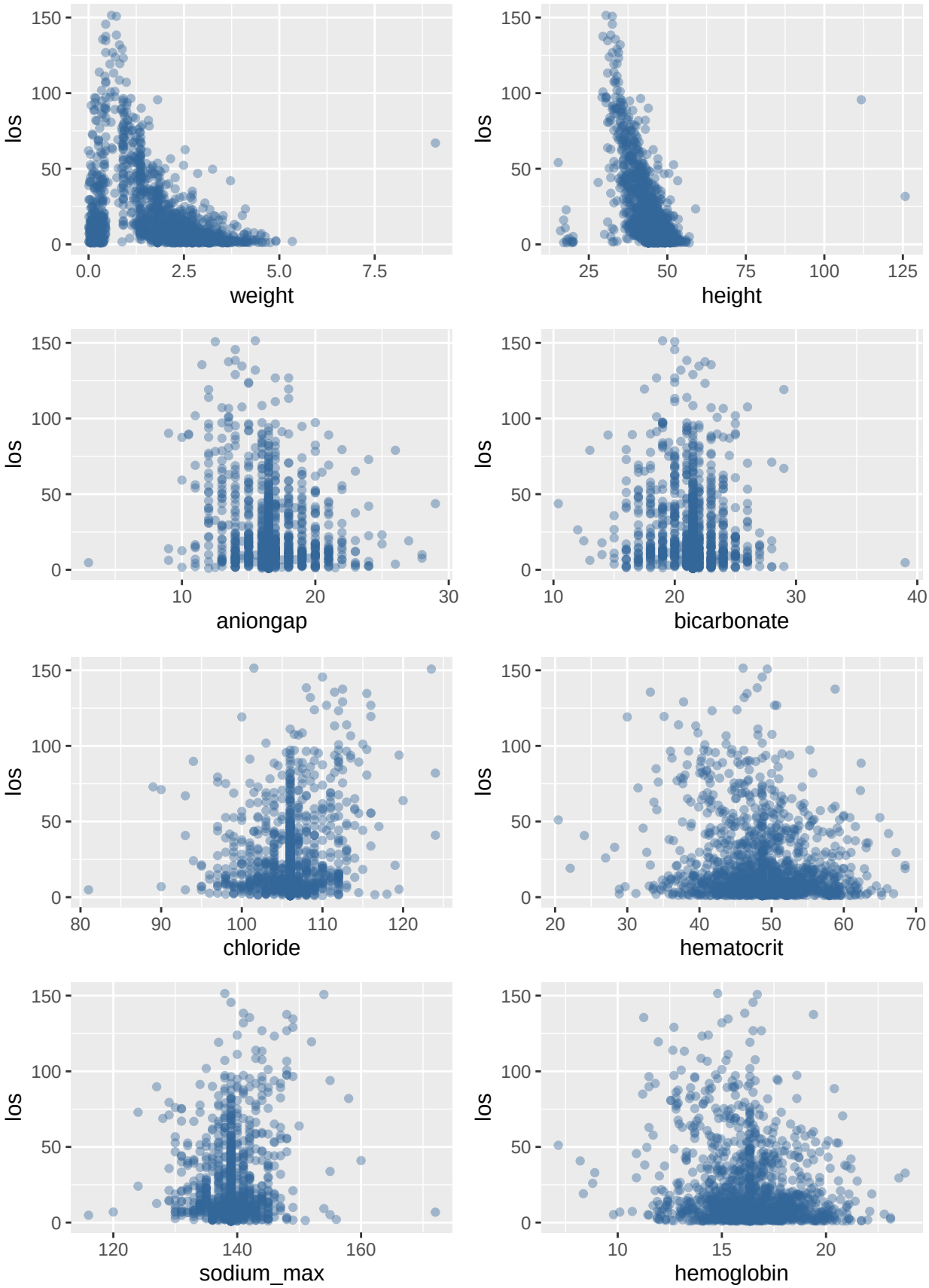
A full model is built in order to identify the significant variables. From the summary of this full model, the variable `ethnicity(black)`, `religion`, `aniongap_min`, `bicarbonate_min`,`bicarbonate_max`, `chloride_min`, `chloride_max`, `height`, `weight`, `heartrate_mean` are significant based on each p-value.

1-2. New variables

The new features `aniongap`, `bicarbonate`, `chloride`, `hematocrit`, `hemoglobin`, and `platelet` are added to dataset since the given data has both max and min of those features.

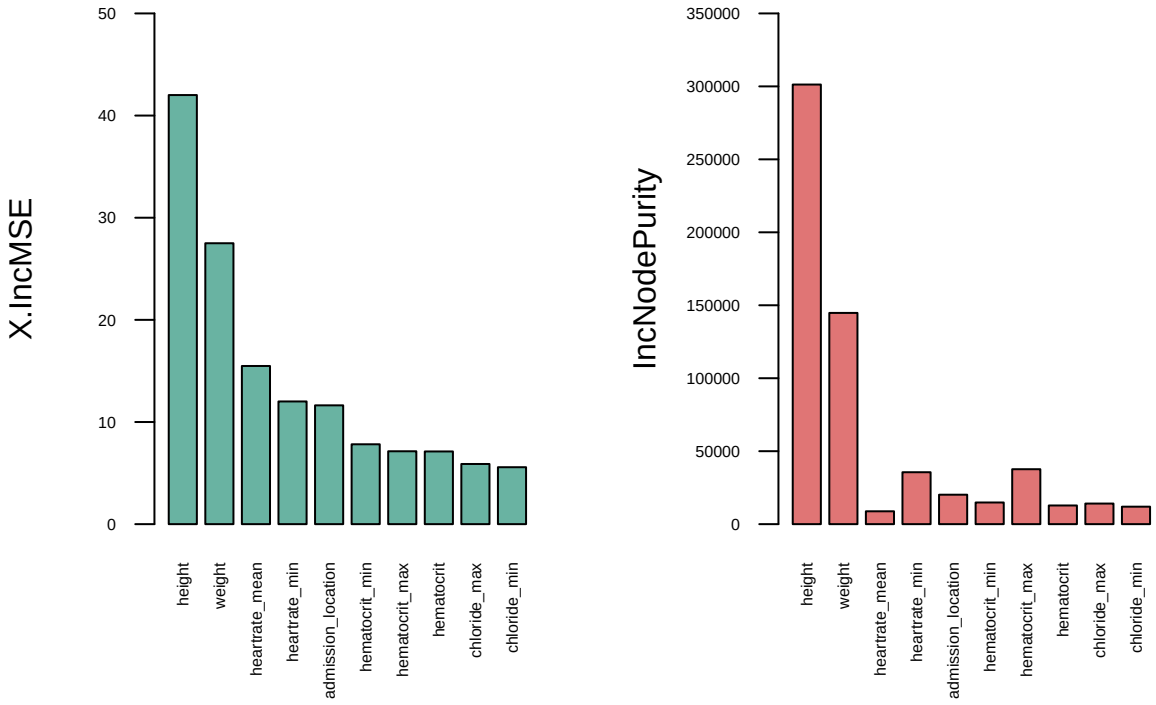
- `aniongap` : An average of aniongap_min and aniongap_max
- `bicarbonate` : An average of bicarbonate_min and bicarbonate_max
- `chloride` : An average of chloride_min and chloride_max
- `hematocrit` : An average of hematocrit_min and hematocrit_max
- `hemoglobin` : An average of hemoglobin_min and hemoglobin_max
- `platelet` : An average of platelet_min and platelet_max

1-3. Outliers


los
weight
height
aniongap
bicarbonate
chloride
hematocrit
sodium_max
hemoglobin

- `height` was found to have a linear trend with `los`.
- `weight` was found to have a linear trend in certain weight divisions.
- `aniongap`, `bicarbonate`, `chloride`, `hematocrit`, `sodium_max`, and `hemoglobin` were hardly found to have a linear trend with `los`.
- Outliers are found from `dat` based on each scatter plot of `weight`, `height`, `aniongap`, `bicarbonate`, `chloride`, `hematocrit`, `sodium_max`, `heartrate_min`.

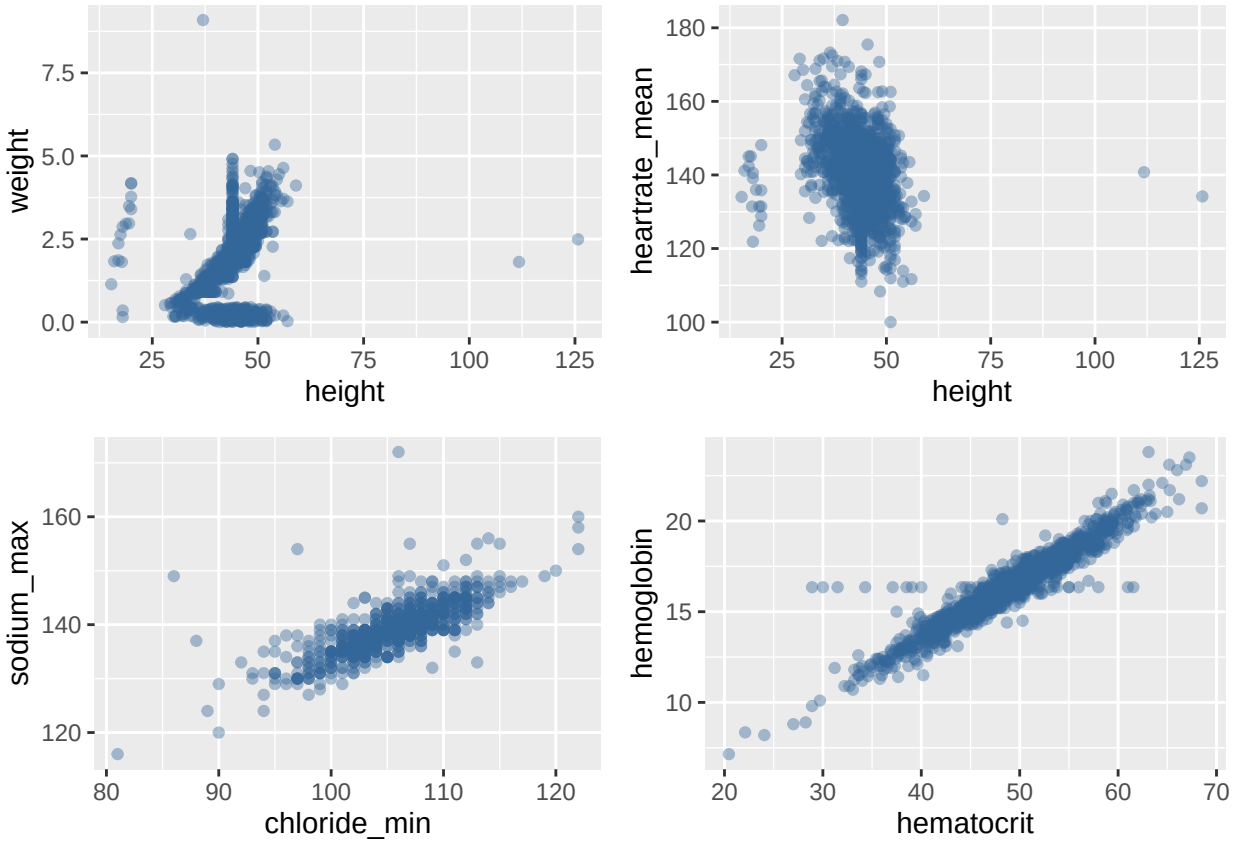
1-4. Variable Importance - Random Forest



The importance of each variable is measured by the percentage increase in the mean squared error (MSE) when that variable is removed from the model. The higher the percentage increase in MSE, the more important the variable is to the model.

The two plots represent the IncreasedMSE and IncreasedNodepurity of each variable. According to the two plots, `height`, `weight`, `heartrate_mean`, are highly significant, while `admission_location`, `hematocrit`, `chloride_max`, `hematocrit_min`, `aniongap_min`, `hemoglobin_max` are weakly significant.

1-5. Interaction terms

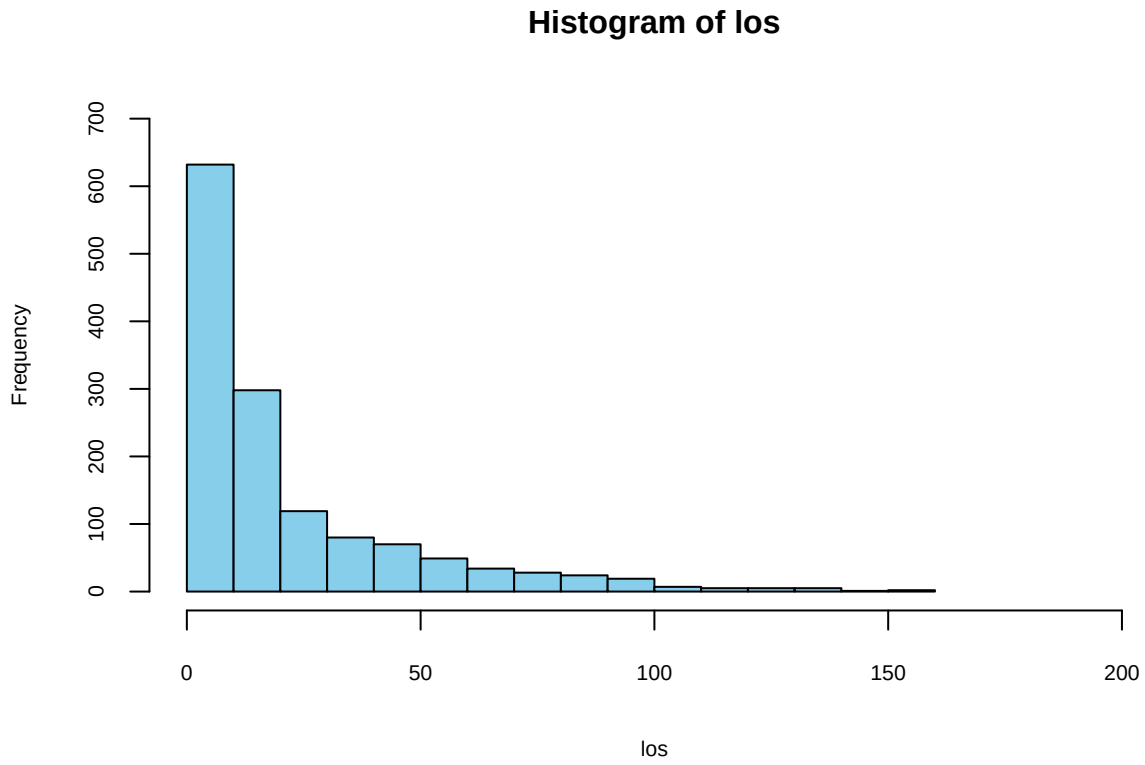


Interactions	Correlation
weight * height	0.3589791
height * heartrate_mean	-0.2749231
chloride_min * sodium_max	0.7473136
hematocrit * hemoglobin	0.9470136

Correlations between variables are calculated, and scatter plots are drawn. According to the table, correlations between `weight&height`, `height&heartrate_mean`, `sodium_max&chloride_min`, and `hematocrit&hemoglobin` seem to be significant as the absolute values of correlations are significantly large. Based on scatter plots, `sodium_max&chloride_min` and `hematocrit&hemoglobin` were found to have a strong linear trend, while `height&heartrate_mean` has a relatively weak trend.

Therefore, they are considered to be included in the following models.

1-6. Histogram of los



The feature `los` will be predicted by the following two models. As this project aims to predict `los` with high accuracy, it is crucial to examine the frequency of this feature. Therefore, the frequency of `los` is plotted, and it is found to be highly concentrated when `los` is less than 30.

2.Statistical Analysis

The given data is fitted with three different models, including smoothing splines, random forests and lasso regression. For each model, the aim was to find the best model that minimize the 10-fold cross validation error.

2-1. Smoothing spline

The smoothing spline model is a statistical method used to fit a smooth curve to a given dataset. The algorithm works by minimizing the sum of squared residuals subject to a constraint on the roughness of the curve. The roughness penalty is controlled by a smoothing parameter that determines the trade-off between fitting the data and having a smooth curve. The final output of the smoothing spline model is a smooth curve that passes through the data points and has minimal roughness.

This is the smoothing spline model for the given dataset.

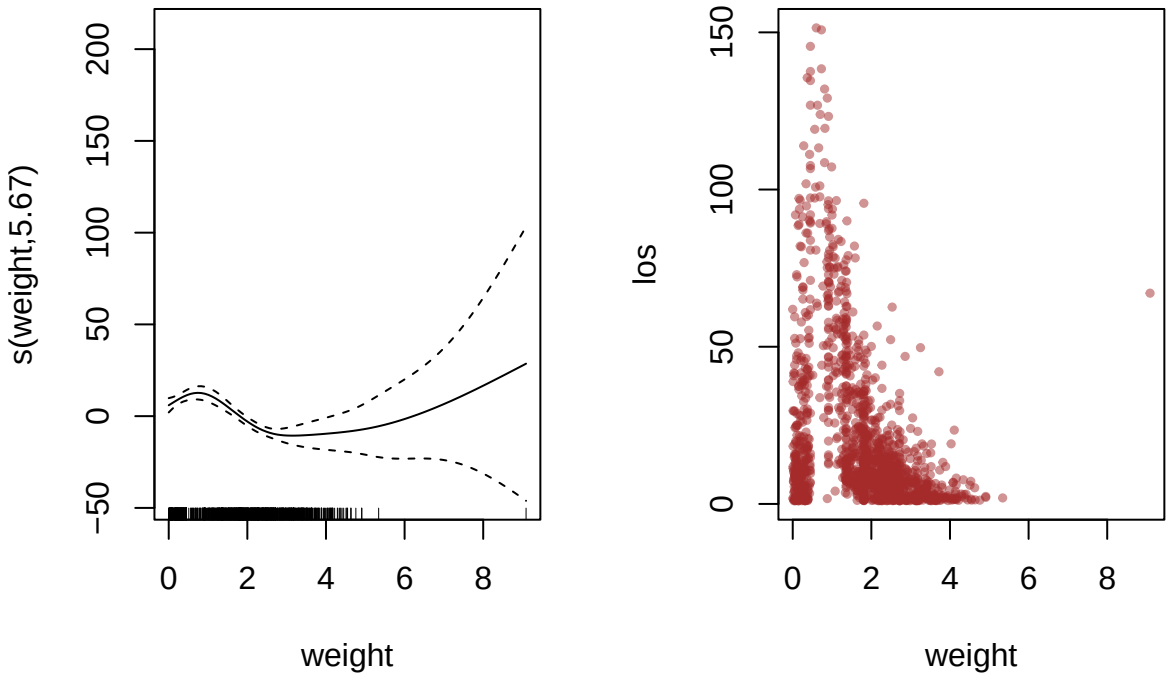
```
ss = gam(los ~ s(weight) + s(height) + s(heartrate_mean) + ti(weight, height), data = dat)
```

`s()` is used to specify that we want to fit a smoothing spline to each of the three variables. The `ti()` function is used to specify that we want to include an interaction between `weight` and `height`.

name	p.val
s(weight)	0.00e+00
s(height)	1.70e-06
s(heartrate_mean)	2.27e-04
ti(weight,height)	0.00e+00

According to the summary, all the smooth terms in this model are significant as each p-value is less than 0.05. 62.7% of the variance in the response variable is explained by the model, and GCV is 258.72.

2-1-1. Plot model fit



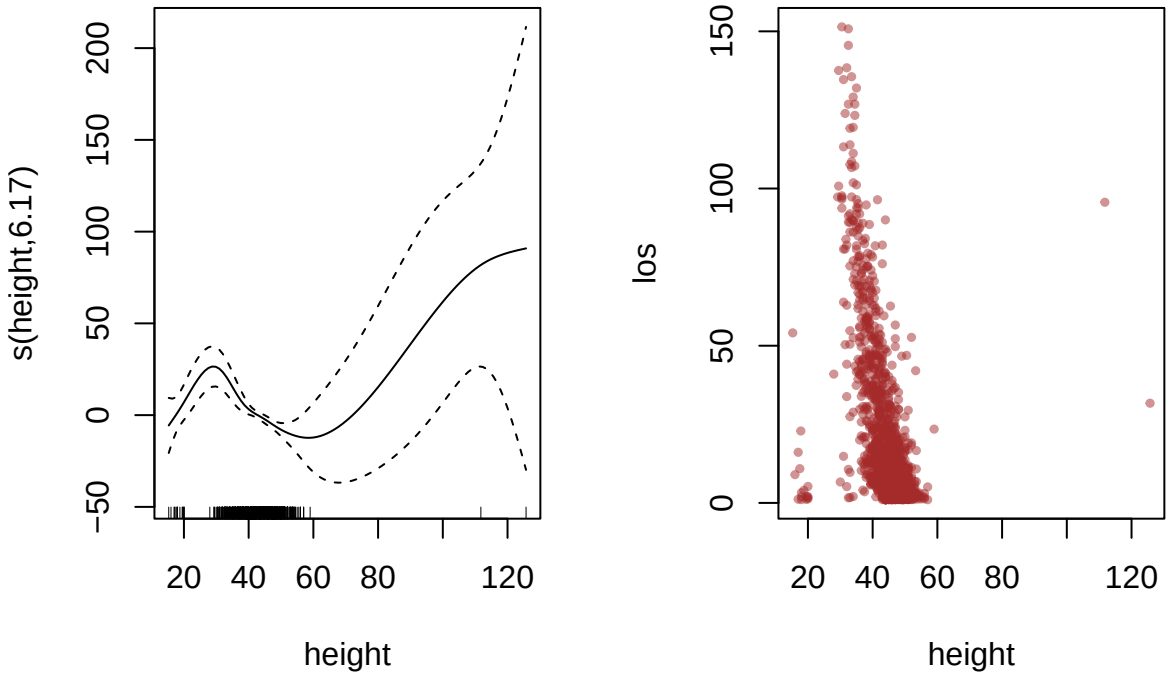
The smoothing spline in `GAM()` is a flexible non-linear function that can capture complex relationships between variables. The partial plot of a smoothing spline shows the relationship between one predictor variable and the response variable while holding all other predictor variables constant.

(a) ‘weight‘ vs ‘s(weight)‘

The resulting plot shows the relationship between `weight` and `los` while holding `height` and `heartrate_mean` constant. The x-axis shows `weight`, the y-axis shows `los`, and the dotted line shows each lower bound and upper bound of 95% confidence interval for the estimated relationship.

The feature `los` increases as `weight` increases when `weight` is between 0 and 1 while holding `height` and `heartrate_mean` constant. Then it has a decreasing tendency until `weight<3`.

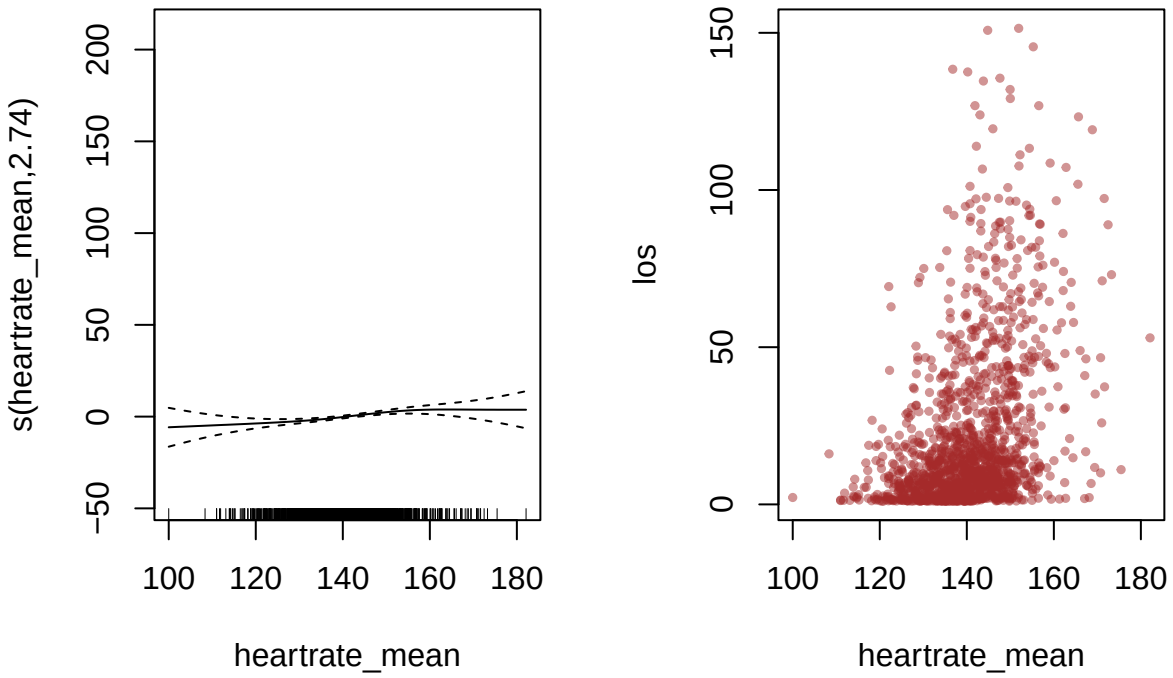
When `weight < 4`, `weight` is highly concentrated with a certain pattern and the 95% confidence interval for the estimated relationship is relatively narrow. It means there is a high degree of certainty that the true population parameter lies within a small range of values. However, when `weight > 8`, the 95% confidence interval for the estimated relationship is relatively wider due to less data and one suspected outlier. For higher prediction accuracy, it is recommended to remove the outlier.



(b) ‘height‘ vs ‘s(height)‘

The feature `los` increases as `height` increases when `height` is between 20 and 35 when holding `weight` and `heartrate_mean` constant. Then it has a decreasing tendency when `height` is between 35 and 60.

When `30 < height < 50`, the 95% confidence interval for the estimated relationship is significantly narrow, height data is highly concentrated with a certain pattern. It means that there is a high degree of certainty that the true population parameter lies within a small range of values. However, when `height > 60`, the 95% confidence interval for the estimated relationship is wider due to less data and two suspected outlier. For higher prediction accuracy, it is recommended to remove those two outliers.

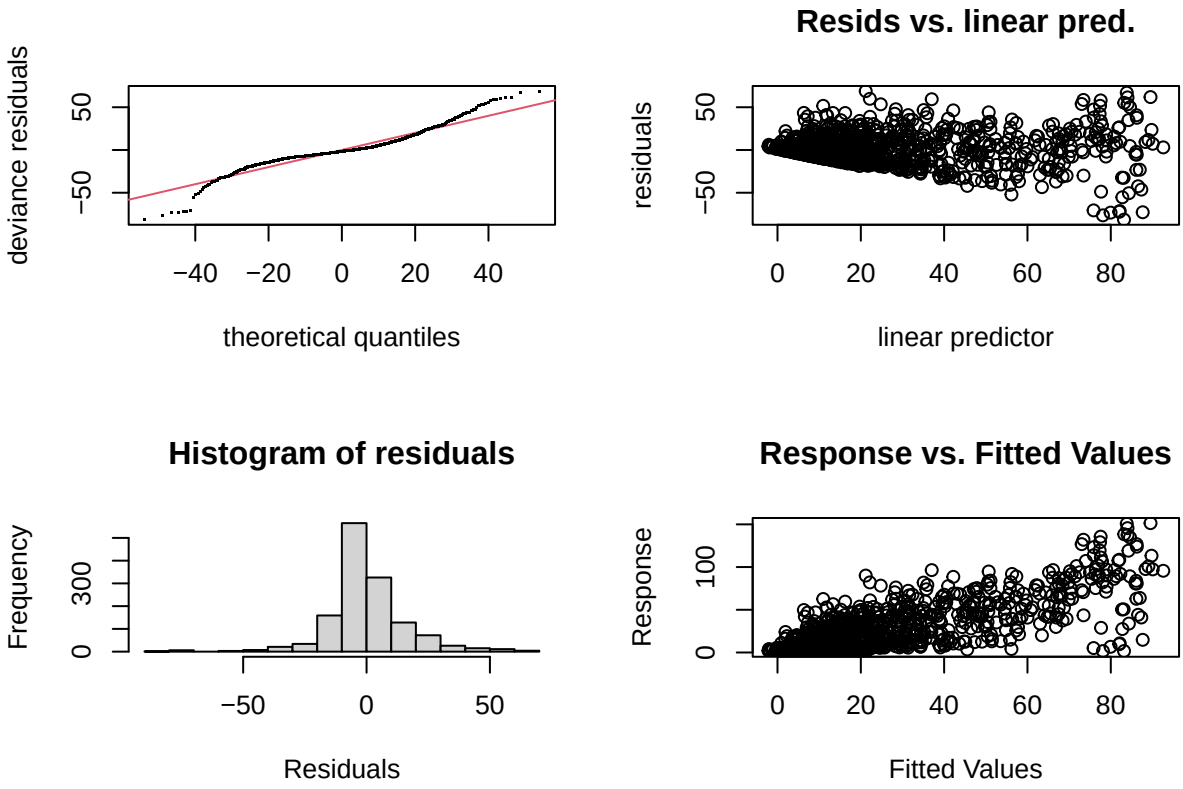


(c) ‘heartrate_mean‘ vs ‘s(heartrate_mean)‘

The feature `los` has a uniform distribution regardless of the value of `heartrate_mean`, unlike `weight` and `height`. It indicates that there is no significant difference in `los` with respect to changes in `heartrate_mean`.

When `120 < heartrate_mean < 160`, `height` is highly concentrated and the 95% confidence interval for the estimated relationship is significantly narrow. On the other hand, when `heartrate_mean < 120` or `160 < heartrate_mean`, the 95% confidence interval for the estimated relationship is wider due to less data.

2-1-2. Check model fit




```
##
## Method: GCV   Optimizer: magic
## Smoothing parameter selection converged after 11 iterations.
## The RMS GCV score gradient at convergence was 0.0005128099 .
## The Hessian was positive definite.
## Model rank =  44 / 44
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                       k'   edf k-index p-value
## s(weight)           9.00  5.67    1.01    0.60
## s(height)           9.00  6.17    0.97    0.12
## s(heartrate_mean)   9.00  2.74    1.01    0.61
## ti(weight,height)  16.00  9.61    1.01    0.63
```

```
(a) Quantile-quantile plot (QQ plot)
```

The QQ plot does not lie on a straight line well since the data includes outliers. QQ plot is used to compare two probability distributions. It is used to check if two sets of data come from the same distribution or to compare the distribution of a sample with a theoretical distribution such as the normal distribution.

(b) Residuals VS Linear predictor

The scatter plot of residuals vs linear predictor shows that there is no data when linear predictor < 40 and Residuals < 0. This indicates that there is a pattern in the residuals that is not explained by the linear predictor. This could mean that the model is not appropriate for the data or that there is a problem with the data itself. Since this data includes outliers, it is believed to have this pattern.

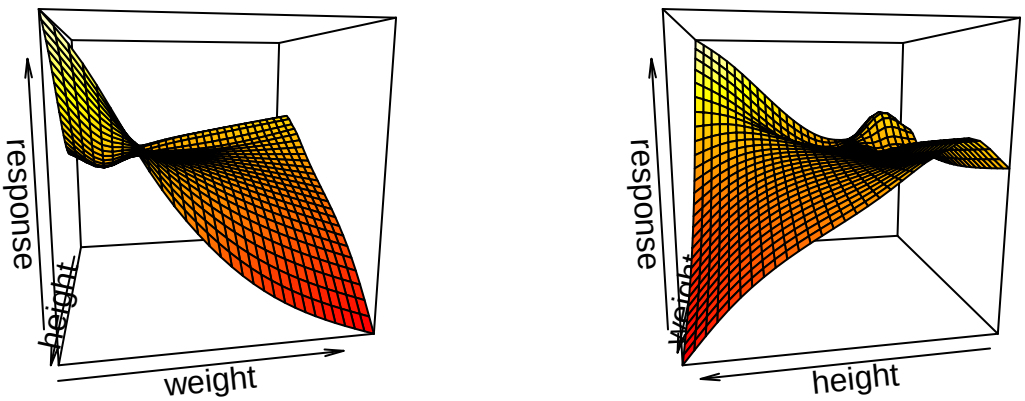
(c) Histogram of Residuals

The histogram of residuals includes outliers, which means that the model is not appropriate for the data or that there is a problem with the data itself. The histogram of residuals shows the distribution of the residuals for all observations. Since this data includes outliers, it is believed to have outliers with respect to residuals.

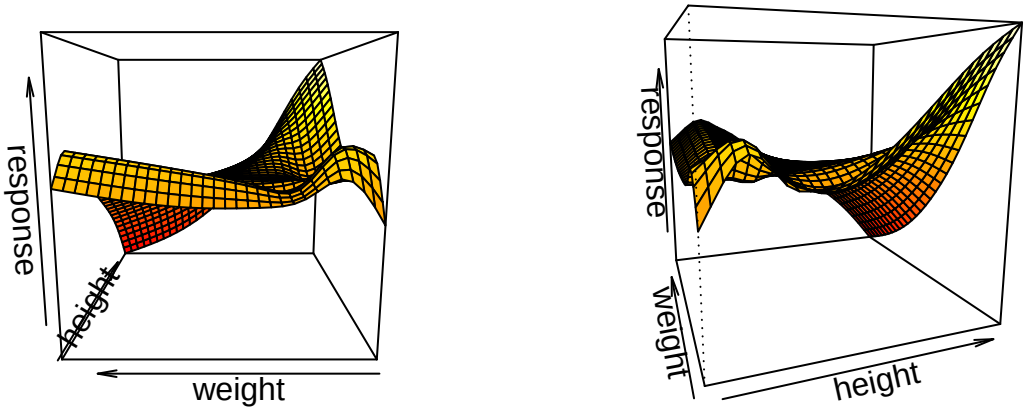
(d) Response VS Fitted values

The Residuals vs Fitted values plot is used to check how well the smoothing spline fits the data. The plot shows the relationship between the observed response values and the fitted values from the smoothing spline. Since the points in the plot should be close to a diagonal line, the smoothing spline fits the data well.

2-1-3. 3D plot



3D plots are drawn with respect to `height` and `weight`. `los` decreases with respect to `weight` when `height` is significantly large. However, `los` increases with respect to `height` when `weight` is significantly large. Hence, `height` and `weight` are strongly correlated.



When `height` is low, `los` is relatively constant with regardless of `weight`. When `weight` is low, `los` is very changeable with respect to `height`. To be specific, `los` significantly increases when holding `weight` in a low value.

2-2. Random Forest

The random forest is a machine learning algorithm that uses an ensemble of decision trees to make predictions on a given dataset. The algorithm works by constructing multiple decision trees and combining their predictions to produce a final output. Each decision tree is constructed using a random subset of the features in the dataset, which helps to reduce overfitting and improve the accuracy of the model. The final output of the random forest is determined by averaging the predictions of all the decision trees in the ensemble.

This is an optimized random forest for the given dataset.

```
rf = randomForest(los ~ weight + height + heartrate_mean + aniongap + admission_location,
                  mtry = 3, ntree=1000, maxnodes= 300,
                  nodesize = 20, sampsize = 350,
                  data = dat, importance=TRUE)
```

2-2-1. tuned hyperparameters

- mtry = 3
- ntree = 1000
- maxnodes = 300
- nodesize = 20

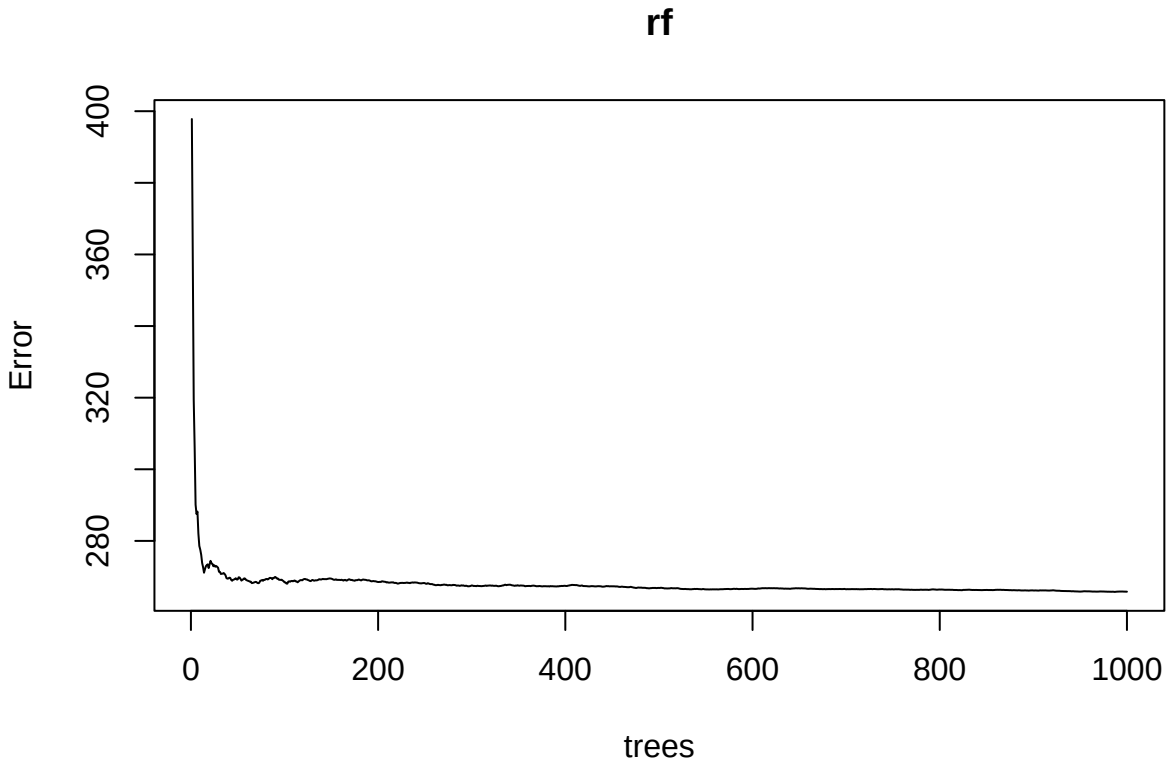
- sampsize = 350

Hyperparameters are tuned by 10-fold cross validation. 3 variables will be randomly sampled as candidates at each split, 1000 trees will be grown, each tree in the forest will have at most 300 terminal nodes, a terminal node will have at least 20 observations, and 350 sample will be drawn for each tree.

(a) mtry

There are 5 features in the model, but mtry is set 3. It means that at each split in each tree, three variables will be selected randomly from the five variables. This is used in random forests to reduce the correlation between trees and improve the performance of the forest. A common rule of thumb is to set mtry=sqrt(p), where p is the number of variables in the data set.

(b) ntree



The error rate decreases as the number of trees increases for regression random forests because when there is a robust number of decision trees in random forests, the classifier will not overfit the model since the averaging of uncorrelated trees lowers the overall variance and prediction error. A larger value of ntree will generally improve the performance of the forest, but it will also increase the computational time and memory requirements.

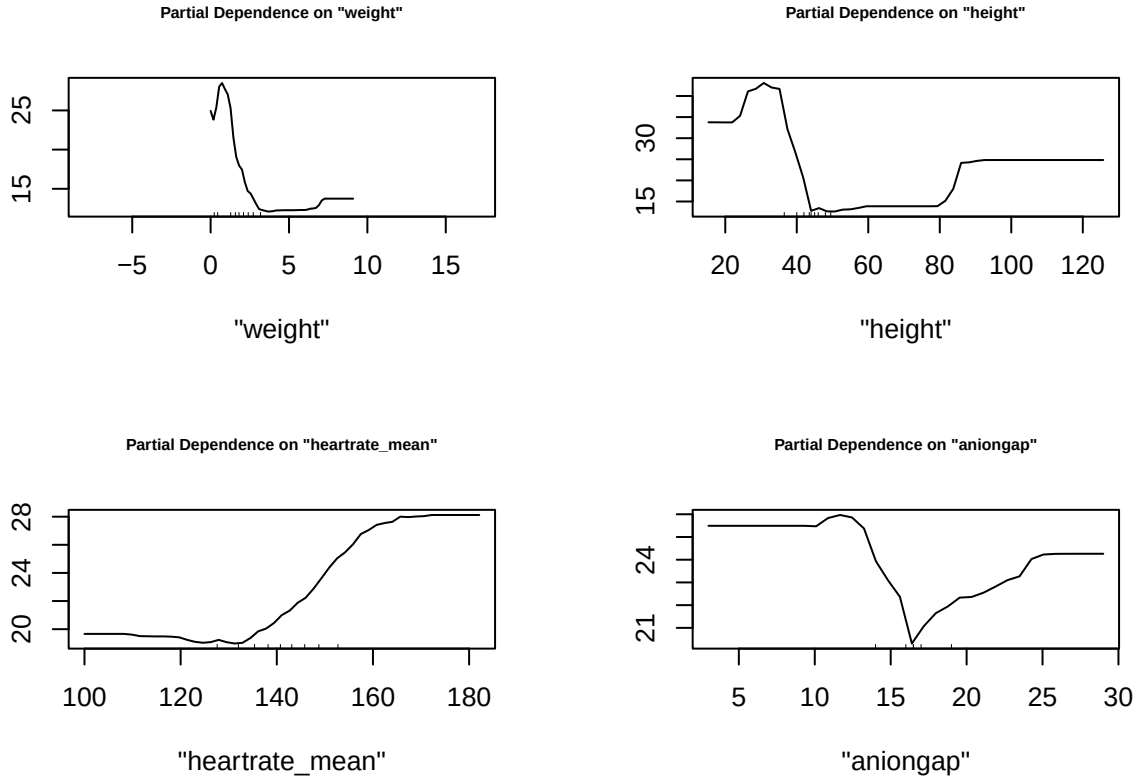
(c) maxnodes and nodesize

Both maxnode and node size are used to control the size of the trees in the forest. A larger value of maxnodes or a smaller value of nodesize will result in larger trees with more splits and more complex decision boundaries. However, the values of maxnodes (300) and nodesize (20) are not too different. It means that the trees in the forest will have a similar size and complexity. A smaller value of maxnodes or a larger value of nodesize will result in smaller trees with fewer splits and simpler decision boundaries.

(d) sampsize

sampsize is a parameter that controls the size of the sample to draw. With sampsize 350, it means that the random forest algorithm will randomly select 350 rows from the dataset to build each decision tree in the forest. This is done to reduce overfitting and improve the accuracy of the model.

2-2-2. Partial Dependence Plots (PDP)



Four partial dependence plots (PDP) are drawn. They give a graphical depiction of the marginal effect of a variable on the response. The effect of a variable is measured in change in the mean response. PDP assumes independence between the feature for which is the PDP computed and the rest.

The y-axis of a partial dependence plot shows the average predicted response for each value of the feature after adjusting for other features in the model. If partial dependence is high, it means that there is a strong relationship between that feature and the response.

Four significant variables including, `weight`, `height`, `heartrate_mean`, `aniongap`, are with high partial dependence at certain value. Particularly, `weight` < 3, `height` < 40, `heartrate_mean` > 140, and `aniongap` < 15 or `aniongap` > 25 have the strong relationship with response.

2-2-3. First tree of Random forest

left_daughter	right_daughter	split_var	split_point	prediction
2	3	weight	1.3450	19.352832
4	5	height	41.2500	38.710157
6	7	weight	2.1250	10.362610
8	9	height	35.7500	54.322813
10	11	weight	0.3400	10.997692
12	13	height	39.1850	16.906382
14	15	weight	3.0550	6.549021
16	17	weight	0.4000	74.363800
18	19	heartrate_mean	136.6867	45.324818
20	21	height	45.5000	9.148309
0	0	NA	0.0000	23.943380
0	0	NA	0.0000	31.594430
22	23	heartrate_mean	149.3892	15.023299
24	25	height	88.8650	8.045167
26	27	aniongap	16.7500	3.615400
0	0	NA	0.0000	4.596567
0	0	NA	0.0000	85.379679
0	0	NA	0.0000	27.461288
28	29	weight	0.1550	48.810385
0	0	NA	0.0000	16.106140

* `‘left daughter‘: The index of the left daughter node`

* `‘right daughter‘: The index of the right daughter node`

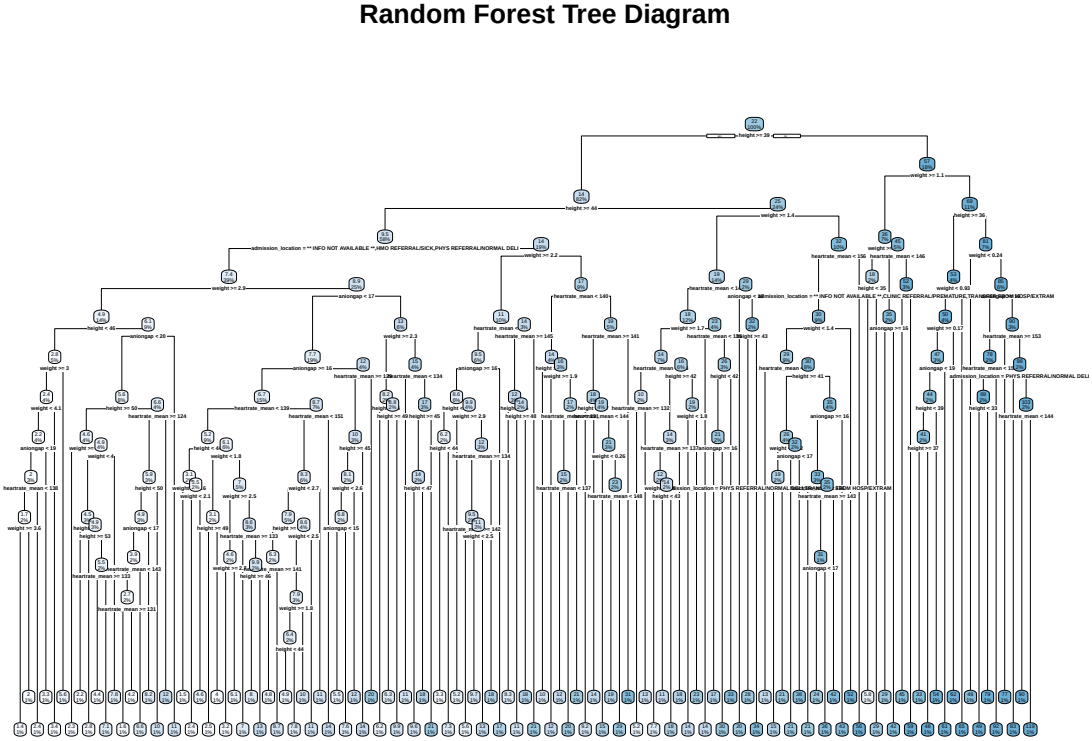
* `‘split var‘: The index of the variable used to split the node`

* `‘split point‘: The value of the split variable`

* `‘prediction‘: The predicted value for terminal nodes`

The first tree is extracted from a random forest. The rows explain nodes from top to bottom of a single tree. According to the first row of the table, the first node has a left daughter with index 2 and a right daughter with index 3. This node is split by the variable `weight` with the value 1.34. If `weight` is greater than 1.34, then the data passes down to left node; otherwise, it passes down to the right node. Lastly, the response variable is predicted based on this node, which is `prediction`.

2-2-4. Visualization



A single tree from the random forest is created. Based on the top node of the tree diagram, the data passes down to the left node if `height` is greater than or equal to 39. Then, the data is split by `weight`, `height`, `aniongap`, `admission_location` based on the certain values of each variable.

This information is not directly related to our model interpretation. This is the only a single tree out of 1000 trees in a random forest. While each decision tree in random forests is easy to interpret, the ensemble of trees is more of a black box. This is because the output of a random forest is an average of the outputs of all the individual trees, which can make it difficult to understand how each variable contributes to the final prediction. Therefore, it can be difficult to interpret the model.

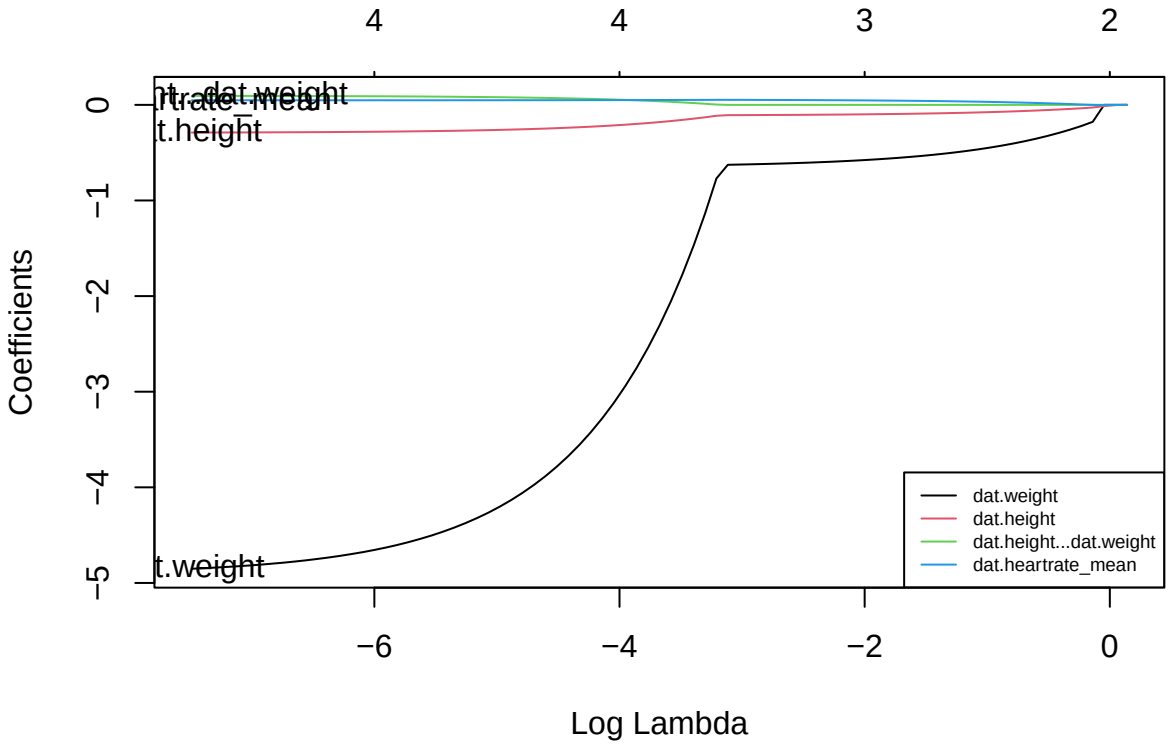
2-3. Lasso Regression

This data is also fitted with lasso regression. Lasso regression is a type of linear regression that uses L1 regularization to shrink the coefficients of the regression variables towards zero. This helps to prevent overfitting and can lead to more interpretable models. The L1 regularization term in the objective function of Lasso regression is the sum of the absolute values of the coefficients multiplied by a tuning parameter lambda. The value of lambda determines how much regularization is applied. However, since lasso regression is a modified linear regression and linear regression is a subset of smoothing splines, the same predictors with the smoothing spline are fitted with this model for comparison to the smoothing spline.

This is an optimized lasso regression for the given dataset.

```
X = data.frame(dat$weight, dat$height, dat$height*dat2$weight, dat$heartrate_mean)
Y = (dat$los)^(1/2)
final_lasso = glmnet(X, Y, alpha = 1, lambda = 0.0005592371)
```

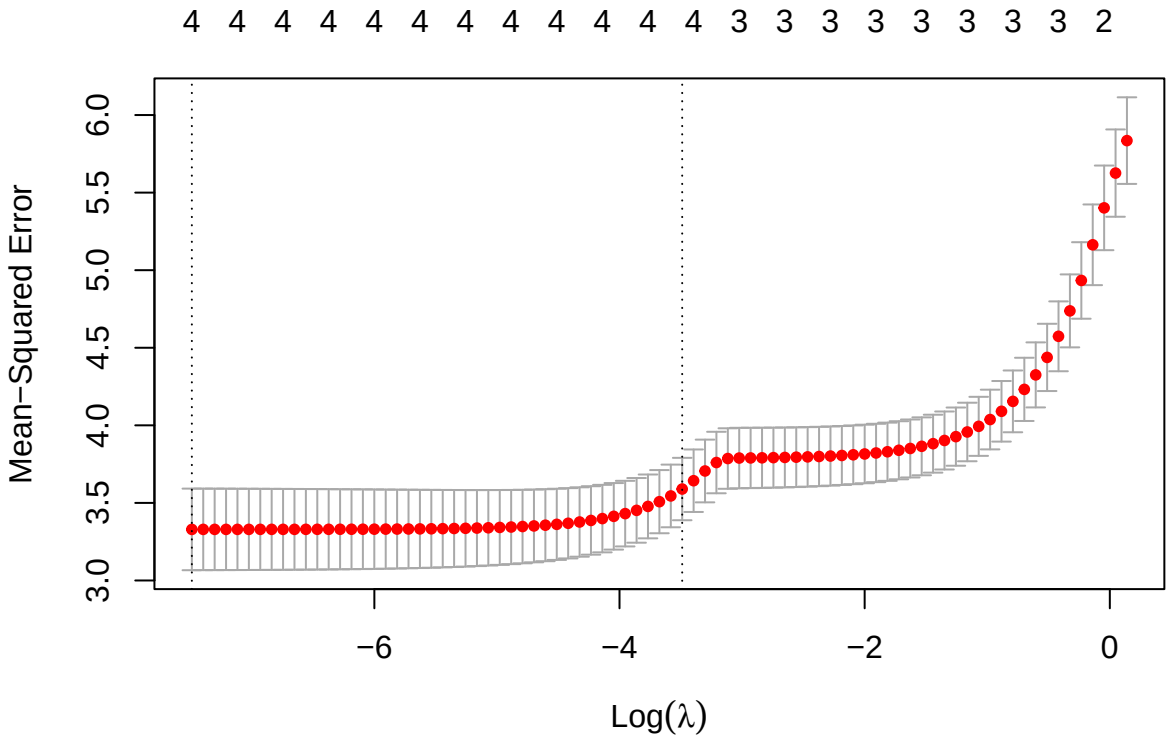
2-3-1. Initial Lasso regression



The initial Lasso regression is fitted with X and Y, without defining any lambda. Each colored line represents the value taken by a different coefficient in this model. λ is the weight given to the regularization term (the L1 norm), so as lambda approaches zero, the loss function of this model approaches the Ordinary least squares regression (OLS) loss function.

The above plot shows how the coefficients change as λ changes. The coefficients of `heartrate_mean` and `height*weight` have a constant coefficient regardless of the value of λ, which means that the coefficient is not affected by changes in λ. It implies that those terms might be highly correlated with other predictors. On the other hand, `height` increased rapidly as λ increases, it means that this coefficient is important for predicting the outcome variable.

2-3-2. log(λ) VS Mean Squared Error



The above plot is used to determine the optimal value of λ that minimizes the mean squared error (MSE) of the model. The plot shows how MSE changes as lambda changes. The optimal value of λ is chosen based on the minimum MSE value. If lambda is too small, then the model may be overfit and may not generalize well to new data. If lambda is too large, then the model may underfit and may not capture important predictors. Therefore, 10-fold cross validation is utilized for the optimal value of λ.

According to the plot, the MSE increases as log(λ) increases since as λ increases, the model complexity decreases. This means that fewer predictors are included in the model. If λ is too large, then the model may underfit and may not capture important predictors. This can lead to an increase in MSE.

2-3-3. Lasso Regression and Interpretation

Df	X.Dev	Lambda
4	44.69	0.0005592

According to the table, degree of freedom, the deviation, λ are equal to 4, 44.69 , and 0.0005592, respectively. Since λ is too small, it means that it has very little shrinkage and those estimates are expected to approximately equal to the one found with linear regression. Therefore, this data is fitted with linear model for comparison.

Here is the linear model for this data.

```
m1 = lm((los)^(1/2)~weight+height+weight*height+heartrate_mean,data=dat)
```

Table 1: OLS Coefficients

	ols_coefficients
(Intercept)	11.1586021
weight	-4.9426185
height	-0.2934311
heartrate_mean	0.0476804
weight:height	0.0964251

Table 2: Lasso Coefficients

	lasso_coefficients
dat.weight	-4.8516722
dat.height	-0.2895344
dat.height...dat.weight	0.0944018
dat.heartrate_mean	0.0478307

Those two tables show the coefficients of each variable. Since λ approximates to 0, this lasso regression approaches the Ordinary least squares regression (OLS) loss function. Those estimates are approximately equal to the one found with linear regression.

As lasso regression is a modification of linear regression, the coefficients in lasso regression can be interpreted similarly to those in linear regression. The coefficients represent the change in the response variable for a one-unit change in the predictor variable while holding all other predictors constant. Therefore, the coefficient of `weight` is equal to -4.8516, which means that the expected `los` decreases by –4.8516 when there is one-unit increase in `weight` while holding other covariates constant. The other variables also can be interpreted in the same way.

3.Comparison between Three Models

Algorithm	Random Forest	Smoothing Spline	Lasso Regression
Type	Ensemble learning method that uses decision trees for prediction purposes	Non-parametric regression method that fits a smooth curve to data points	Linear regression method that uses regularization to prevent overfitting
Overfitting risk	Low risk of overfitting because it uses multiple decision trees	Low risk of overfitting because it fits a smooth curve to data points	Low risk of overfitting because it uses regularization
General Running time	Can be slow for large datasets because it creates multiple decision trees	Can be slow for large datasets because it fits a smooth curve to data points	Fast because it uses linear regression

Algorithm	Random Forest	Smoothing Spline	Lasso Regression
Interpretability	Not very interpretable because it creates multiple decision trees	More interpretable than Random Forest because it fits a smooth curve to data points	Very interpretable because it uses linear regression
Ease of use	Easy but can be difficult to use because it requires tuning of hyperparameters	Hard because it requires choosing the degree of smoothing	Easy to use because it uses linear regression
Ease of model building	Easy but can be difficult to build models with because it requires tuning of hyperparameters	Difficult to build models with because it requires choosing the degree of smoothing	Easy to build models with because it uses linear regression
Sensitivity to outliers	Not very sensitive to outliers because it creates multiple decision trees	Sensitive to outliers because it fits a smooth curve to data points	Sensitive to outliers

3-1. Time complexity

Algorithm	Random Forest	Smoothing Spline	Lasso regression
Running time	2~3 seconds	1~2 seconds	1 second

`(a) Random Forest`

Tuning hyperparameters is a computationally expensive process that aims to find the best combination of hyperparameters with the least mean squared error (MSE). The more hyperparameters there are, the longer it takes to find the best combination. Grid search is a common method used to find the best combination of hyperparameters. It involves searching over a range of hyperparameter values to find the optimal combination. This process can take hours or even longer depending on the number of hyperparameters and their ranges. However, it takes 2~3 seconds to build an optimized random forest once the hyperparameters are set. With more trees, it takes longer time to build a model.

For building a random forest corresponding to the given data, each range of hyperparameters is set as below.

```
- mtry = c(15,5,3,1)

- ntree=c(1000)

- maxnodes=c(10,100,300)

- nodesize=c(1,20,50)

- sampsize=c(250,300,350))
```

There are 108 combinations of hyperparameters, the best hyperparameters which has the least mean squared error is found.

`(b) Smoothing Spline`

The time complexity of smoothing spline model built using the `GAM()` function varies depending on hyperparameters, the number of interaction terms and the number of variables. A large value of `k`, a large number of variables, or a large number of interaction terms causes high time complexity.

For this model, it takes around 1~2 seconds to build a model as k is set to 10 and only three variable with one interaction term are included.

`(c) Lasso regression`

Lasso regression is a modification of linear regression. It is relatively fast because it uses linear regression.

3-2. Ease of use

`(a) Random Forest`

Random forest is a type of ensemble learning method that combines multiple decision trees to produce better predictive performance than a single decision tree. They are relatively easier to build than smoothing splines and do not require interaction terms because they consider variables sequentially, which makes them handy for considering interactions without specifying them. Interactions that are useful for prediction will be easily picked up with a large enough forest, so there is no real need to include an explicit interaction term. As long as tuned hyperparameters and significant variables are found, the random forest is ready to be built.

`(b) Smoothing Spline`

It requires more steps to build an optimized smoothing spline. Since smoothing spline can be fitted with smooth terms including `s()`, `ti()`, `te()`, `t2()`, the predictors need to be specified in the model depending on smooth functions of predictors. For this smoothing spline model, `s()` is used to specify each of the three variables and `ti()` is used to specify the interaction between `weight` and `height`.

Furthermore, there is the other parameter to be tuned. `k` is the number of basis functions used for smooth terms that needs to be checked to ensure that they are not so small that they force oversmoothing. If effective degree of freedom (edf) is significantly less than `k`, `k` needs to decrease. For those four models, each edf are signifinatly less than each `k`, any `k` is not tuned.

`(c) Lasso Regression`

There is only one tunning parameter for this model. Since the optimized lambda can be found the function `cv.glmnet()`, this model is relatively easier to be used.

3-3. Interpretation

Each model is interpreted in section 2.

`(a) Random Forest`

Random forest is considered a black box model because it is difficult to interpret how each feature contributes to the final prediction.

`(b) Smoothing Spline`

Smoothing spline is more interpretable because they provide a smooth curve that can be easily visualized and understood.

`(c) Lasso Regression`

Lasso regression is very interpretable because it uses linear regression. As the interpretation of a linear regression model depends on the coefficients of the model, it can be easily interpreted with coefficients.

3-4. Sensitivity to outliers

First of all, random forests, smoothing splines, lasso regression, and linear model are built with the dataset including outliers. Since estimates for lasso regression are approximately equal to the one found with linear model, linear model is used for model evaluation. (section 2-3-3)

For comparing sensitivity to outliers, dataset `dat` is duplicated and outliers are removed. Each dataset (with outliers, and without outliers) is fitted with random forests, smoothing splines, and linear model, respectively. Then, each model is tested with the data that includes outliers.

CV	Random Forest	Smoothing Spline	Linear Regression
With outliers	16.11	16.44	18.92399
Without outliers	15.4598	20.86	18.88445

According to the table of cross-validation score, which is calculated based on mean squared errors, random forests and linear model have a similar cross validation score regardless of outliers, while smoothing spline has a significantly better cross validation score when outliers are included.

The result suggests that the outliers may not be important for the model's performance in random forests and linear regression and it implies that they have a low overfitting risk. On the other hand, the smoothing spline model without outliers has a much higher score than the smoothing spline model with outliers. This means that the outliers have a significant impact on the model's performance and a smoothing spline is more likely to overfit the data than random forests due to smoothing parameters.

However, linear regression got a similar cross validation score regardless of outliers. It implies that linear regression is less sensitive to outliers than smoothing splines because smoothing splines use a smoothing parameter that controls the amount of smoothing applied to the data. Hence, smoothing splines are more sensitive to outliers than linear regression.

4. Conclusion

Random forests, smoothing splines, and lasso regression are three different models used for regression analysis. Linear regression is also fitted for analysis since the penalty parameter λ is approximately equal to 0, which means that estimates from lasso regression are expected to equal to the one found with linear regression.

Random forests are considered a black box model because it is difficult to interpret how each feature contributes to the final prediction. Also, this information is not directly related to our model interpretation as an output of a random forest is an average of the outputs of all the individual trees, which can make it difficult to understand how each variable contributes to the final prediction. Therefore, it can be difficult to interpret the model. On the other hand, the smoothing spline and the lasso regression are more interpretable because that can be easily visualized by plots and understood by coefficients.

However, random forests show better cross validation score than the other two models, regardless of outliers. Random forests show slightly better performance when the dataset excludes outliers while the smoothing

spline shows a significantly better performance when the dataset includes outliers. It implies that random forests are less sensitive to outliers and less likely to overfit the data.

The linear regression can be regarded as smoothing splines without any smoothing parameters. However, linear regression gets a similar cross validation score regardless of outliers, unlike smoothing splines. It implies that linear regression is less sensitive to outliers than smoothing splines because smoothing splines use a smoothing parameter that controls the amount of smoothing applied to the data. Therefore, smoothing splines are more sensitive to outliers than linear regression.

In summary, random forests are a powerful method for prediction but lacks interpretability while the other two models are more interpretable but may not be as powerful as random forests due to a higher risk of overfitting and prediction accuracy.

Appendix - Model Building Process

Step 1. Add the new variables

Step 2. Model 1 - Smoothing Spline

2-a. Identify the significant variables

2-b. Build smoothing spline models

2-c. Model evaluation using K-fold Cross-Validation

Step 3. Model 2 - Random Forest

3-a. Identify the importance variables

3-b. Model matrix for hyperparameters

3-c. Tuned hyperparameters by 10-fold Cross-Validation

3-d. Optimized Random Forest

3-e. Model evaluation using K-fold cross-validation

Step 4. Model 3 - Lasso

4-a. Tuned parameter by k-fold Cross-Validation

4-b. Build a Lasso regression

4-c. Build a Linear regression

4-d. Model evaluation using K-fold Cross-Validation

Step 1. Add the new variables

- `aniongap` : An average of aniongap_min and aniongap_max
- `bicarbonate` : An average of bicarbonate_min and bicarbonate_max
- `chloride` : An average of chloride_min and chloride_max
- `hematocrit` : An average of hematocrit_min and hematocrit_max
- `hemoglobin` : An average of hemoglobin_min and hemoglobin_max
- `platelet` : An average of platelet_min and platelet_max

Above six new variables are added to the dataset.

Step 2. Model 1 :: Smoothing Spline

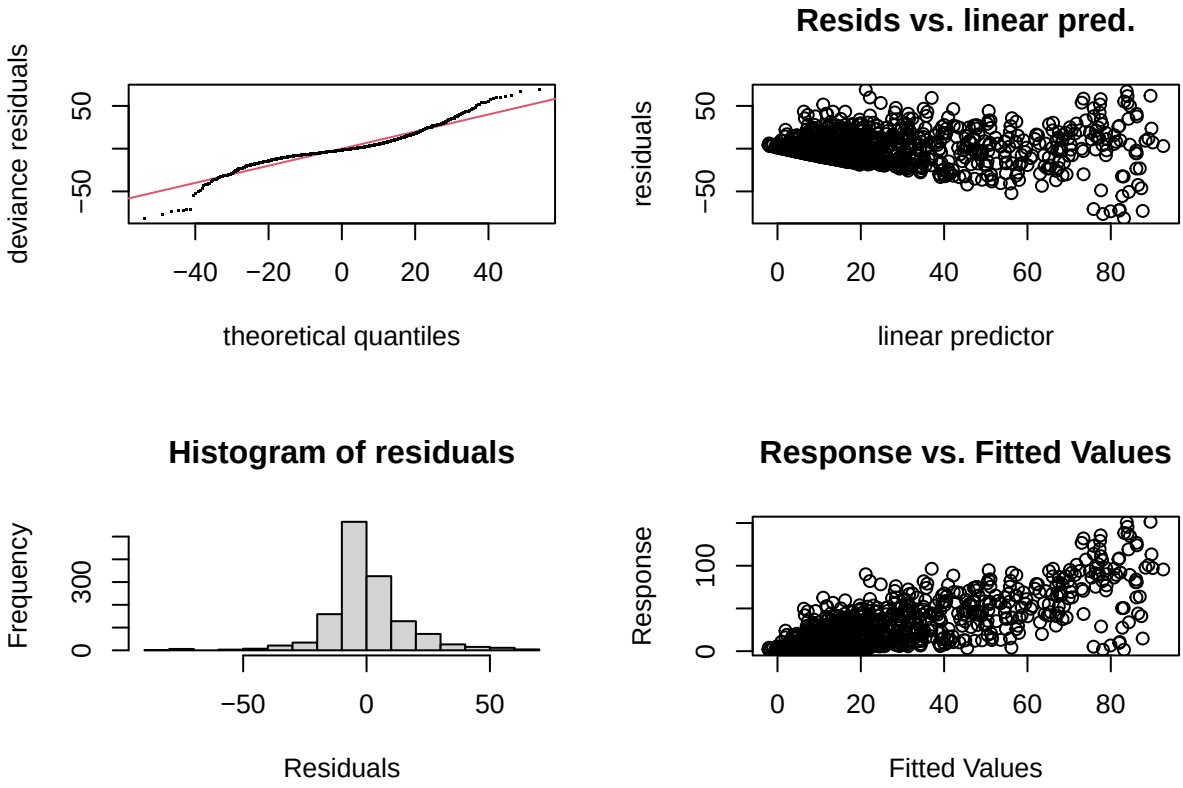
2-a. Identify the significant variables

Significant_terms	Correlation
weight * los	-0.4247845
height * los	-0.4716727
weight * height	0.3589791
heartrate_mean * los	0.3715606

Significant variables are identified by correlation and scatter plots (scatter plots are drawn on 1-5). Since each correlation coefficient of `weight * los`,`height * los`, `weight * height`, `heartrate_mean * los` are greater than 0.35, they are significantly correlated. Hence, they are included in the following smoothing spline models.

2-b. Build smoothing spline models

```
ss1 = gam(los ~ s(weight) + s(height) + s(heartrate_mean), data = dat) #16.60
ss2 = gam(los ~ s(weight) + s(height) + s(heartrate_mean) + ti(weight, height), data = dat) # 16.44
ss3 = gam(los ~ s(weight) + s(height) + ti(weight, height), data = dat) # 16.5027
ss4 = gam(los ~ s(weight) + s(height), data = dat) #16.769
gam.check(ss2)
```




```
##
## Method: GCV   Optimizer: magic
## Smoothing parameter selection converged after 11 iterations.
## The RMS GCV score gradient at convergence was 0.0005128099 .
## The Hessian was positive definite.
## Model rank =  44 / 44
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                       k'   edf k-index p-value
## s(weight)           9.00  5.67    1.01   0.580
## s(height)           9.00  6.17    0.97   0.085 .
## s(heartrate_mean)   9.00  2.74    1.01   0.580
## ti(weight,height)  16.00  9.61    1.01   0.635
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

For the basis function, cubic spline `s()` is utilized to specify the smoothness of the spline. The larger the value of s(), the smoother the curve will be. Also, `ti()` is used over `te()`, for interaction terms, since `ti()` specifies a single knot location for each term while `te()` specifies a separate knot location for each observation. This makes `ti()` more efficient than `te()`.

If effective degree of freedom (edf) is significantly less than k, k needs to decrease. For those four models, each edf are signifinatly less than each k, any k is not tuned.

```
kfold = function(n){
  k = 10
  data = dat
  predictors = names(data)[2:ncol(data)]
  response = names(data)[1]

  set.seed(777)
  folds = createFolds(data$los, k)
  dat = data[-folds[[n]], ]
  dtest = data[folds[[n]], ]

  set.seed(777)
  #fit = gam(los ~ s(weight) + s(height) + s(heartrate_mean), data = dat) #16.60 ss1
  fit = gam(los ~ s(weight) + s(height) + s(heartrate_mean) + ti(weight, height),
            data = dat) # 16.44 ss2
  #fit = gam(los ~ s(weight) + s(height) + ti(weight, height), data = dat) # 16.5027 ss3
  # fit = gam(los ~ s(weight) + s(height), data = dat) #16.769 ss4
  pred = predict(fit, newdata=dtest)
  res = data.frame(Id=folds[[n]], los=pred)
  return(sqrt(mean((res$los - dtest$los)^2)))
}


cv_score = 0
for (i in 1:10){
   cv_score = kfold(i) + cv_score
 }
cv_score/10
```

```
## [1] 16.44116
```

Mean squared errors (MSE) by 10-fold crossvalidation is calculated as above code. The 10-fold cross-validation score is a measure of how well a model generalizes to new data. A lower score indicates better performance.

```
df_s1 = c('s(weight) + s(height) + s(heartrate_mean)','s(weight) + s(height) + s(heartrate_mean) + ti(w
df_s2 = c(16.60413,16.44116,16.5027,16.769)
df_ss = data.frame(df_s1,df_s2)
colnames(df_ss) = c('Model','MSE')

kable(df_ss) %>%
  kable_styling(position = "center", latex_options = "HOLD_position")
```

Model	MSE
s(weight) + s(height) + s(heartrate_mean)	16.60413
s(weight) + s(height) + s(heartrate_mean) + ti(weight, height)	16.44116
s(weight) + s(height) + ti(weight, height)	16.50270
s(weight) + s(height)	16.76900

According to the 10-fold crossvalidation, `los ~ gam(s(weight) + s(height) + s(heartrate_mean) + ti(weight, height)` has the least mean squared errors (MSE), which means it predicts the data better than other three.

Step 3. Model 2 :: Random Forest

3-a. Identify the importance variables

```
set.seed(777)
fit = randomForest(los ~ ., data = dat, importance=TRUE)

imp=data.frame(importance(fit))
ord <- order(imp$X.IncMSE, decreasing=TRUE)
imp=imp[ord, ]
imp2 = imp[1:10,]
imp2
```

```
##                     X.IncMSE IncNodePurity
## height             40.096886    299128.158
## weight             29.710649    153065.254
## admission_location 15.363313      9158.993
## heartrate_min      12.021320     21869.836
## heartrate_mean     10.728145     35093.284
## hematocrit_min      8.759363     14666.307
## hematocrit          8.113970     12202.574
## hemoglobin_min      5.531266     11777.434
## hemoglobin          5.432374     10063.135
## platelet_max        5.098622     12754.859
```

The importance of each variable is measured by the percentage increase in the mean squared error (MSE) when that variable is removed from the model. According to the above table, `weight`, `height`, `admission_location`, and `heartrate_mean` show the significantly greater values in X.IncMSE. However, `hematocrit_min`, `hemoglobin_min`, `hematocrit_max`, `aniongap` have similar values in X.IncMSE but `aniongap` has a significant value in IncNodePurity.

Therefore, `weight`, `height`, `heartrate_mean`, `aniongap`, `admission_location` are added to the random forests.

3-b. Model matrix for hyperparameters

```
X = model.matrix(~ weight + height + heartrate_mean + aniongap + admission_location, data=dat)
Y = dat$los
```

The model matrix is built to run rfcv(), which is a cross validation function for random forests.

```
hyperparams = expand.grid(mtry = c(15,5,3,1), ntree=c(1000),
                          maxnodes=c(10,100,300), nodesize=c(1,20,50), sampsize=c(250,300,350))
hyperparams[1:10,]
```

```
##    mtry ntree maxnodes nodesize sampsize
## 1    15  1000       10        1      250
## 2     5  1000       10        1      250
## 3     3  1000       10        1      250
```

```
## 4     1  1000       10        1      250
## 5    15  1000      100        1      250
## 6     5  1000      100        1      250
## 7     3  1000      100        1      250
## 8     1  1000      100        1      250
## 9    15  1000      300        1      250
## 10    5  1000      300        1      250
```

The ranges for each hyperparameter are randomly chosen. Those hyperparameters will be tuned by cross-validation based on hyperparagrams. A random forest is fitted with all the combinations of hyperparameters according to the hyperparams, then the hyperparameters with the least mean squared error will be fitted to the final model.

3-c. Tuned hyperparameters by 10-fold Cross-Validation

```
result = data.frame()

for (i in 1:nrow(hyperparams)) {
  set.seed(777)
  model = rfcv(X, Y, cv.fold = 10, mtry = function(x){hyperparams$mtry[i]},
               ntree=hyperparams$ntree[i], maxnodes=hyperparams$maxnodes[i],
               nodesize=hyperparams$nodesize[i], sampsize=hyperparams$sampsize[i])
  tem = list(n.var = model$n.var, error.cv = model$error.cv)
  tem2 = c(mse = min(tem$error.cv), mtry = hyperparams$mtry[i],
           ntree=hyperparams$ntree[i],maxnodes=hyperparams$maxnodes[i],
           nodesize=hyperparams$nodesize[i], sampsize=hyperparams$sampsize[i])
  print(tem2)
  result = rbind(result, tem2)
}
colnames(result) = c('mse','mtry','ntree','maxnodes','nodesize','sampsize')
head(result)
optimalFeatures = c(result$mtry[which.min(result$mse)],result$ntree[which.min(result$mse)],
                    result$maxnodes[which.min(result$mse)],
                    result$nodesize[which.min(result$mse)],
                    result$sampsize[which.min(result$mse)])
optimalFeatures
```

The list of hyperparameters to be tuned :

- mtry = Number of variables randomly sampled as candidates at each split
- ntree = Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times.
- maxnodes = Maximum number of terminal nodes trees in the forest can have.
- nodesize = Minimum size of terminal nodes.
- sampsize = Size(s) of sample to draw.

The ranges for each hyperparameter are randomly chosen, and those hyperparameters are tuned by 10-fold cross-validation. A random forest is fitted with all the combinations of hyperparameters according to the hyperparams, then the hyperparameters with the least mean squared error are chosen to the the final model.

3-d. Optimized random forest

```
rf = randomForest(los ~ weight + height + heartrate_mean + aniongap + admission_location,
                  mtry = 3, ntree=1000, maxnodes= 300,
                  nodesize = 20, sampsize = 350,
                  data = dat, importance=TRUE)
```

tuned hyperparameters :

- mtry: 3
- ntree: 1000
- maxnodes = 300
- nodesize = 20
- sampsize = 350

3-e. Model evaluation using K-fold cross-validation

```
kfold = function(n){
  data = dat
  k = 10
  predictors = names(data)[2:ncol(data)]
  response = names(data)[1]

  folds = createFolds(data$los, k)
  dat = data[-folds[[n]], ]
  dtest = data[folds[[n]], ]

  set.seed(777)
  fit = randomForest(los ~ weight + height + heartrate_mean + aniongap + admission_location,
                  mtry = 3, ntree=1000, maxnodes= 300,
                  nodesize = 20, sampsize = 350,
                  data = dat) # rf
  pred = predict(fit, newdata=dtest)
  res = data.frame(Id=folds[[n]], los=pred)
  return(sqrt(mean((res$los - dtest$los)^2)))
}

cv_score = 0
for (i in 1:10){
   cv_score = kfold(i) + cv_score
 }
cat("According to the 10-fold cross-validation,
    the MSE of an optimized randomForest is approximately ", cv_score/10)
```

```
## According to the 10-fold cross-validation,
##     the MSE of an optimized randomForest is approximately  14.42342
```

Step 4. Model 3 :: Lasso Regression

4-a. Tuned parameter by k-fold Cross-Validation

```
X = data.frame(dat$weight, dat$height, dat$height*dat$weight, dat$heartrate_mean)
X = as.matrix(X)
Y = (dat$los)^(1/2)
```

X is a matrix of predictor variables and Y is a vector of response variable values. They are defined to build a lasso regression. Since lasso regression is a modified linear regression and linear regression is a subset of smoothing splines, the same predictors with the smoothing spline are fitted with this model for comparison to the smoothing spline.

```
set.seed(777)
lasso_mod = cv.glmnet(X, Y, alpha = 1, nfolds = 10)
```

A lasso regression is fitted with X and Y. `alpha` is a value between 0 and 1 that determines the type of regularization, and `nfolds` is the number of folds to use in cross-validation. When alpha = 1, Lasso regression is performed while Ridge regression is performed when alpha = 0. And the lasso regression is trained using `cv.glmnet()` function.

4-b. Build a Lasso regression

```
set.seed(777)
bestlam = lasso_mod$lambda.min
# Fit the final Lasso model using the best lambda value
final_lasso = glmnet(X, Y, alpha = 1, lambda = bestlam)
bestlam
```

```
## [1] 0.0005592371
```

The value of `bestlam` is 0.0005592371. This value is obtained by performing cross-validation on the Lasso regression model using the training data. The value of `bestlam` is chosen such that it minimizes the mean squared error (MSE) of the model on the training data. Once `bestlam` is obtained, a new Lasso regression model is trained using all of the training data and this value of `bestlam`. The resulting model is then used to make predictions on the test data.

Since λ,`bestlam`, is too small, it means that it has very little shrinkage and those estimates are expected to approximately equal to the one found with linear regression. Therefore, this data is fitted with linear regression for comparison.

Here is the linear model for this data.

```
m1 = lm((los)^(1/2)~weight+height+weight*height+heartrate_mean,data=dat)
```

4-c. Build a linear regression

The linear model is built. It is fitted with the same features for comparison.

	ols_coefficients
(Intercept)	11.1586021
weight	-4.9426185
height	-0.2934311
heartrate_mean	0.0476804
weight:height	0.0964251

	lasso_coefficients
dat.weight	-4.8516722
dat.height	-0.2895344
dat.height...dat.weight	0.0944018
dat.heartrate_mean	0.0478307

According to those tables, two models approximately got the same coefficients. Therefore, they are expected to have the similar cross validation score.

4-d. Model evaluation using K-fold cross-validation

```
kfold = function(n){
  k = 10
  data = dat0

  set.seed(777)
  folds = createFolds(data$los, k)
  dat = data[-folds[[n]], ]
  dtest = data[folds[[n]], ]

  set.seed(777)
  m1 = lm((los)^(1/2)~weight+height+weight*height+heartrate_mean,data=dat)
  pred = predict(m1, newdata=dtest)
  res = data.frame(Id=folds[[n]], los=pred)
  return(sqrt(mean((res$los^2 - dtest$los)^2)))
}

cv_score = 0
for (i in 1:10){
  cv_score = kfold(i) + cv_score
}
cv_score/10
```

```
## [1] 18.92399
```

The resulting cross-validation MSE score is around 18.9.