Body Fat Data Analysis

jdt

11/10/2020

```
  # clear the environment and set seed
rm(list = ls())
set.seed(123)
```

Read Body Fat Data

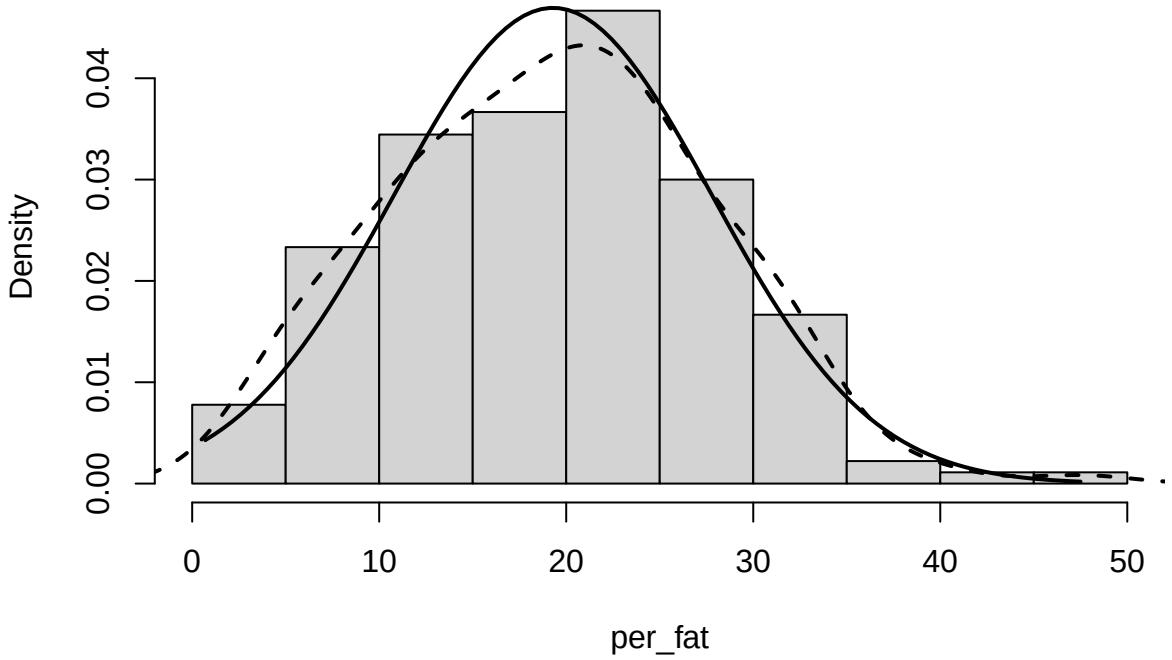
```
library(foreign)
bfat = read.dbf("new_bfat.dbf")
```

Define a few variables

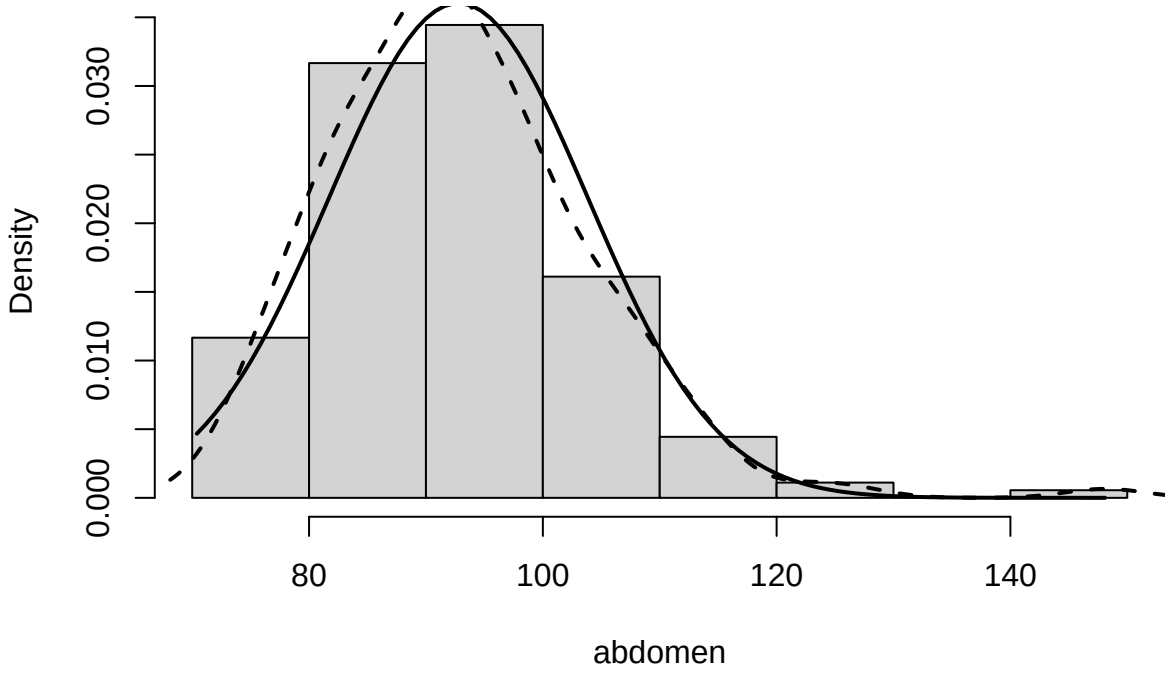
```
abdomen=bfat$abdomen
thigh = bfat$thigh
neck = bfat$neck
per_fat = bfat$per_fat
density = bfat$density
age=bfat$age
wt = bfat$wt
ht = bfat$ht
chest = bfat$chest
hip = bfat$hip
thigh = bfat$thigh
knee = bfat$knee
ankle = bfat$ankle
biceps = bfat$biceps
forearm = bfat$forearm
wrist = bfat$wrist
```

Plots of Percent Fat and Abdomen Circumference

```
with(bfat, hist(per_fat, main="", freq=FALSE))
with(bfat, lines(density(per_fat), main="PERCENT FAT", lty=2, lwd=2))
xvals = with(bfat, seq(from=min(per_fat), to=max(per_fat), length=100))
with(bfat, lines(xvals, dnorm(xvals, mean(per_fat), sd(per_fat)), lwd=2))
```

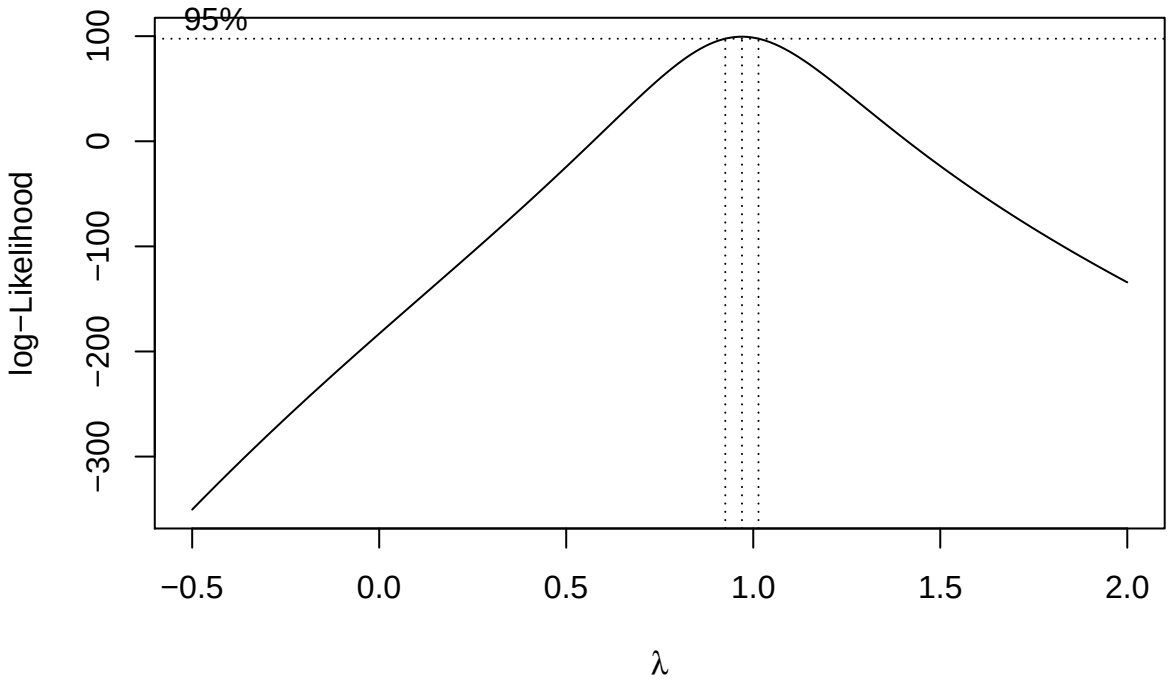



```
with(bfat, hist(abdomen, main="", freq=FALSE))
with(bfat, lines(density(abdomen), main="ABDOMEN", lty=2, lwd=2))
xvals = with(bfat, seq(from=min(abdomen), to=max(abdomen), length=100))
with(bfat, lines(xvals, dnorm(xvals, mean(abdomen), sd(abdomen)), lwd=2))
```



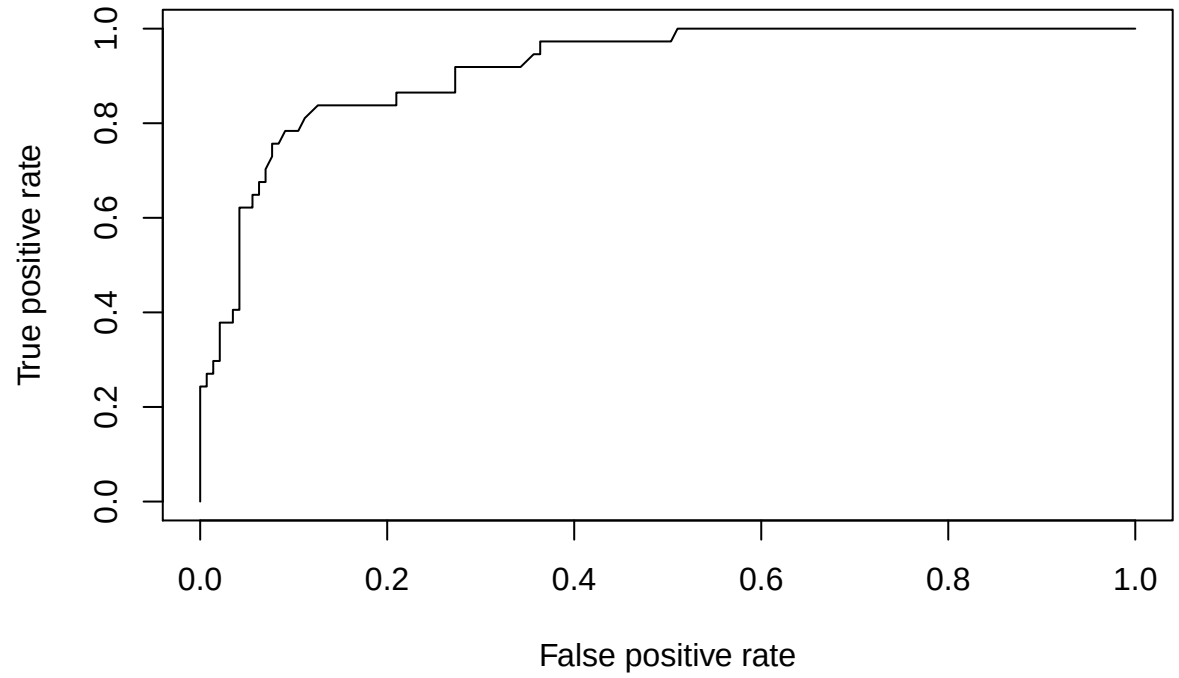
Box Cox transformation for Abdomen

```
library(MASS)
boxcox(per_fat ~. ,data=bfat, lambda=seq(-.5, 2.0, length=200))
```

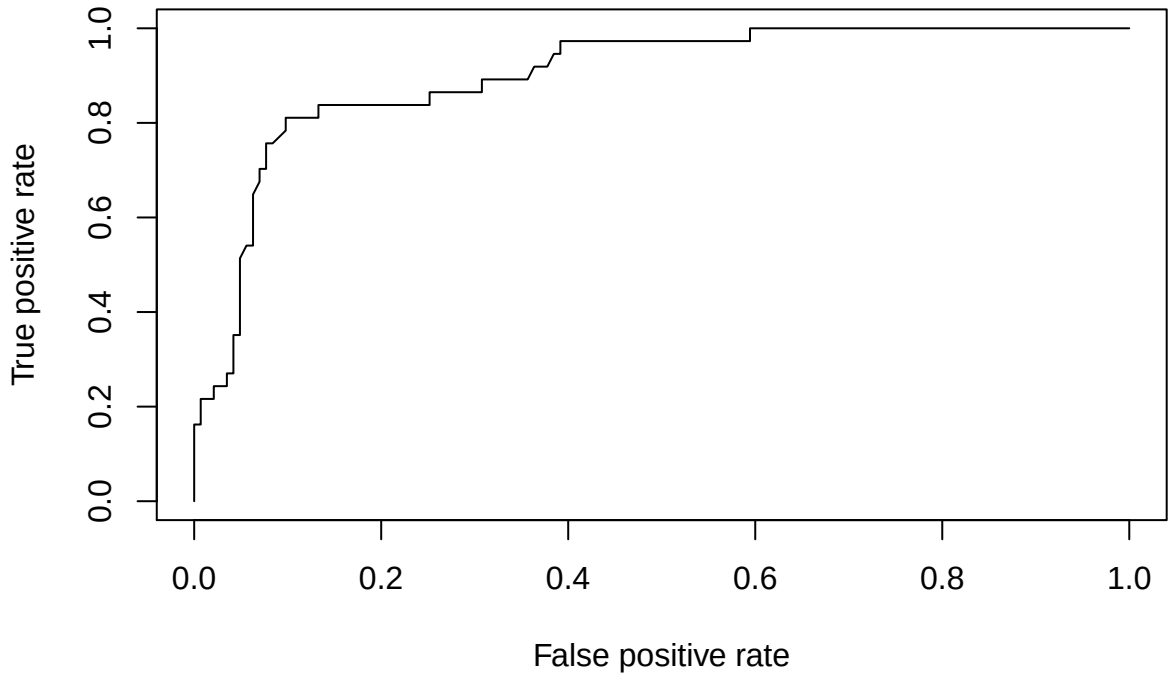


ROC Curves for High Percent Fat

```
library(ROCR)
#The code for the ROC with abdomen
disease=(per_fat > 26)
pred = prediction(abdomen,disease)
perf=performance(pred, "tpr", "fpr")
plot(perf)
```



```r
# ROC for abdomen + thigh
pred = prediction(abdomen + thigh,disease)
perf=performance(pred, "tpr", "fpr")
plot(perf)
```



Linear Regression - model per_fat = abdomen

```r
mod1 = lm(per_fat ~ abdomen, data=bfat)
summary(mod1)
```

```
##
## Call:
## lm(formula = per_fat ~ abdomen, data = bfat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.9840  -3.6341  -0.0102   3.4709  10.2028
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -39.22601    3.06172  -12.81   <2e-16 ***
## abdomen       0.63072    0.03277   19.25   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.854 on 178 degrees of freedom
## Multiple R-squared:  0.6755, Adjusted R-squared:  0.6736
## F-statistic: 370.5 on 1 and 178 DF,  p-value: < 2.2e-16
```

```r
covb = vcov(mod1)
coeff.mod1 = coef(mod1)
```

```
covb = vcov(mod1)
covb
```

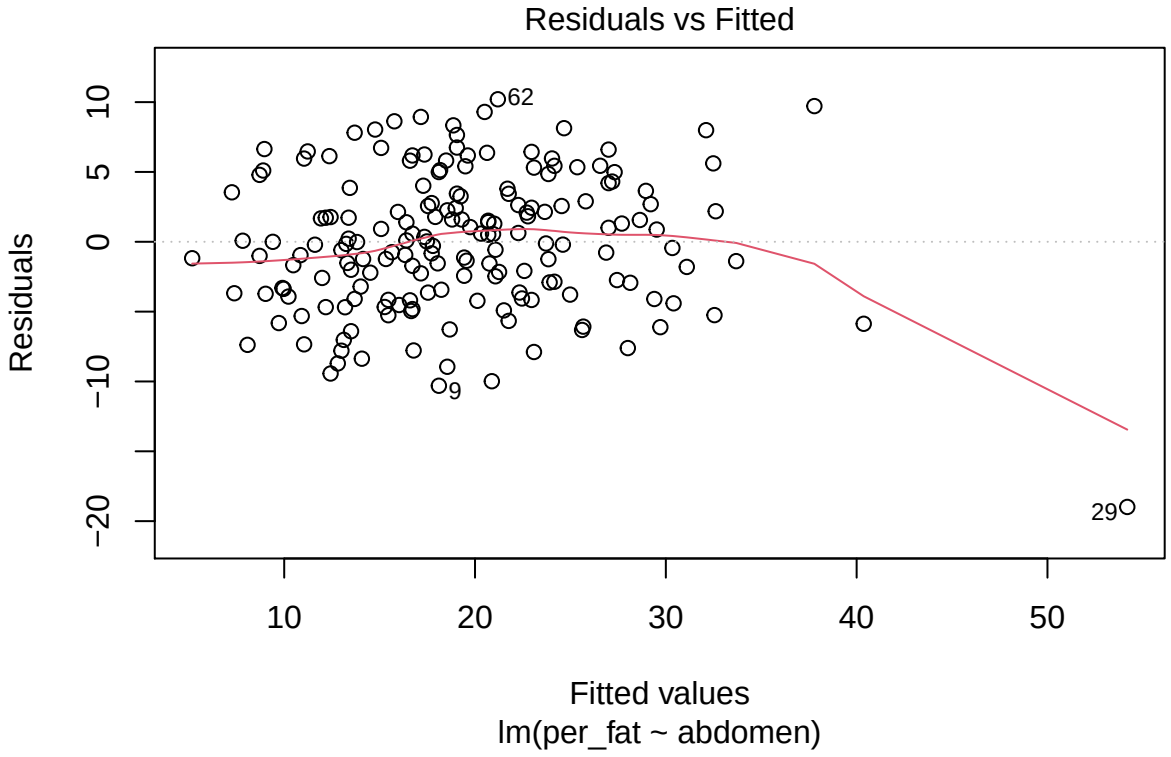
```
##             (Intercept)      abdomen
## (Intercept)  9.37412238 -0.099624319
## abdomen     -0.09962432  0.001073763
```

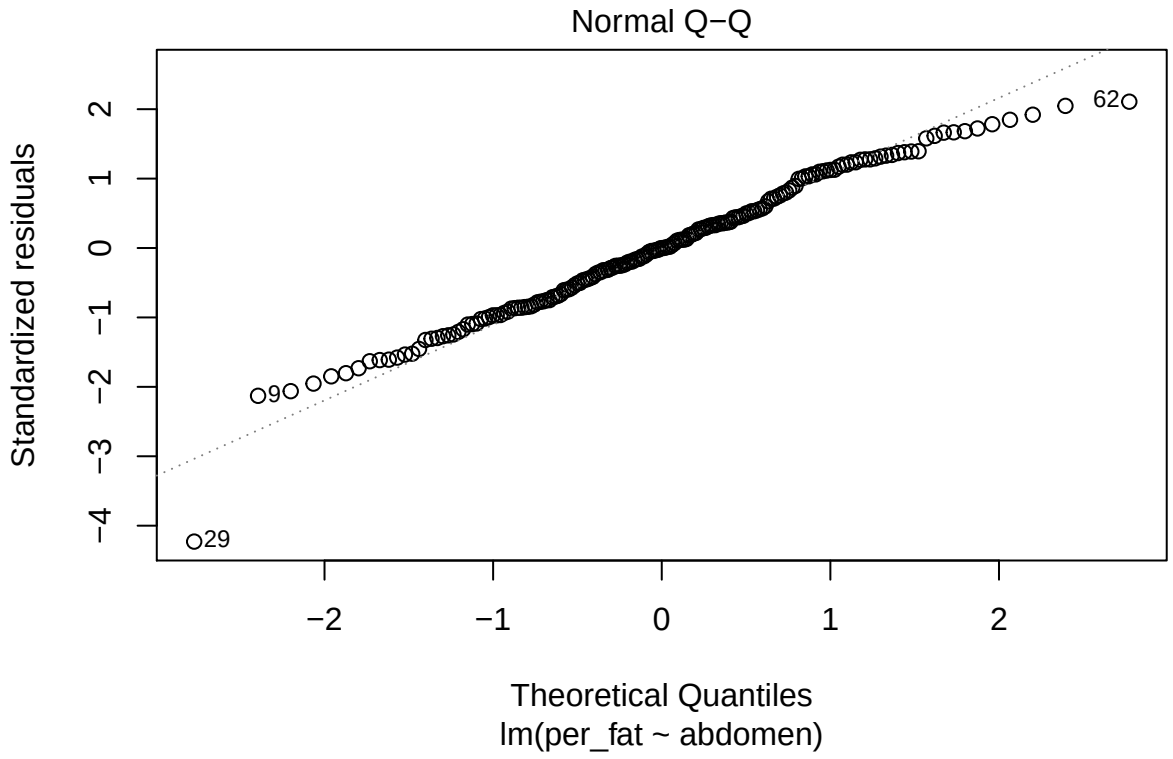
```
pred.per_fat = predict(mod1)
res.per_fat = residuals(mod1)
summary(res.per_fat)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -18.98400  -3.63414  -0.01023   0.00000   3.47087  10.20279
```

Residual Plots

```
#par(mfrow=c(1,1))
plot(mod1, which = c(1, 2))
```





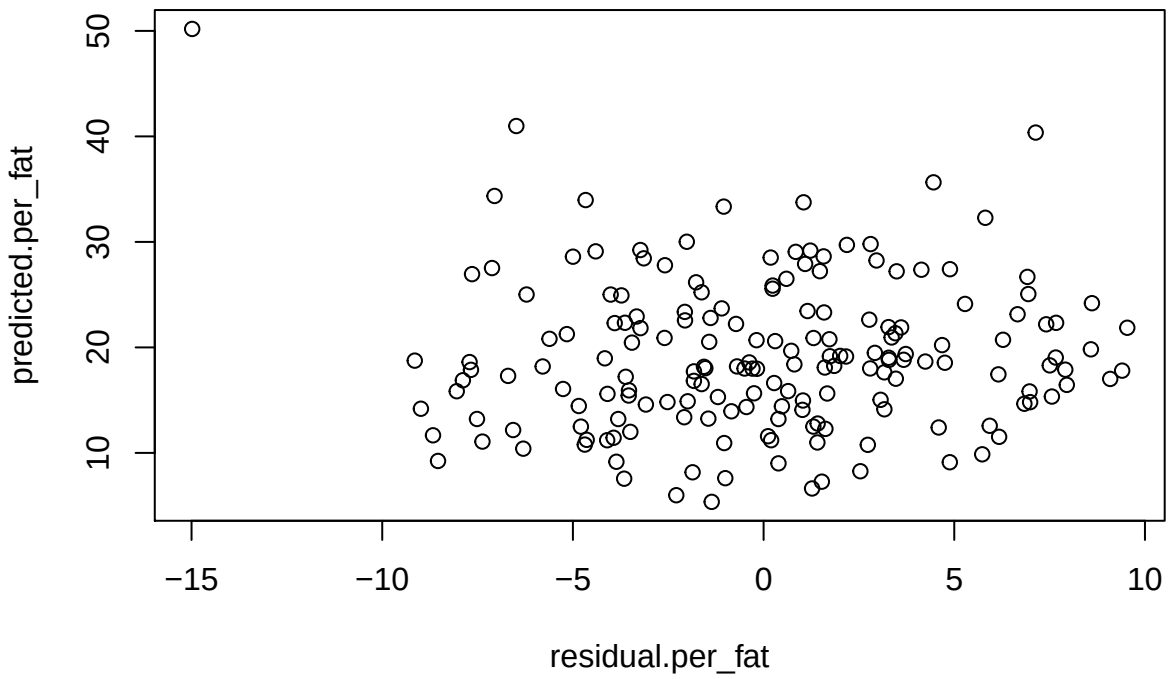
Multiple Regression

```
mod2 = lm(per_fat ~ abdomen + thigh + neck, data=bfat)
summary(mod2)
```

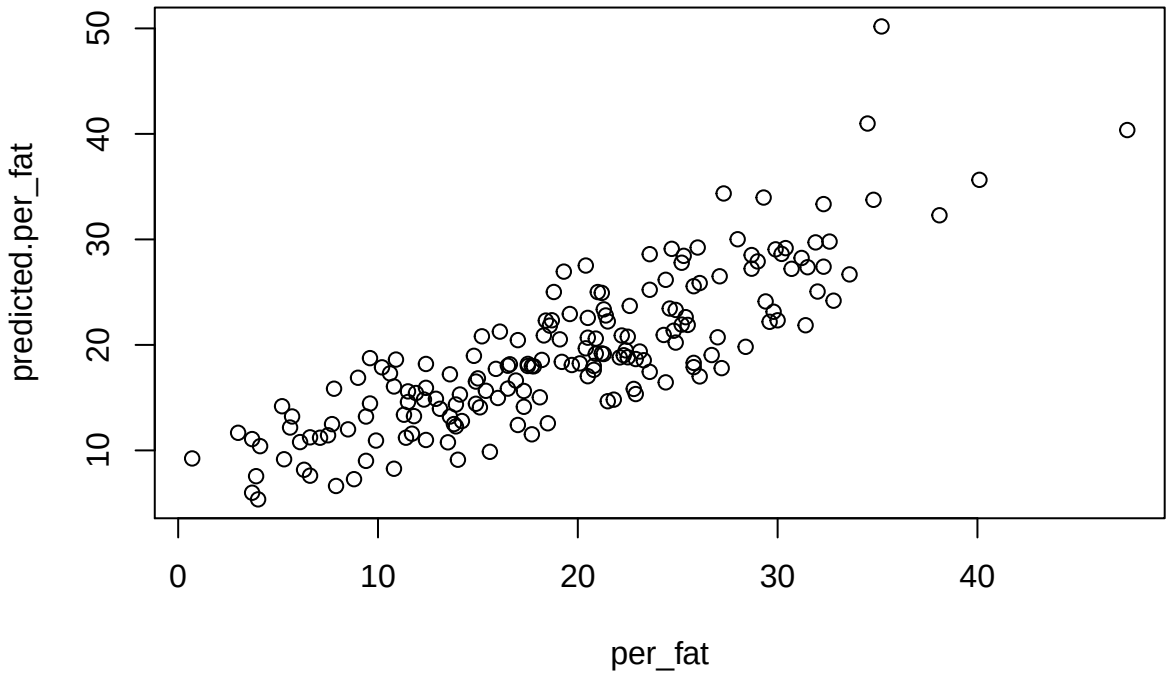
```
##
## Call:
## lm(formula = per_fat ~ abdomen + thigh + neck, data = bfat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.9836  -3.4626   0.2172   3.1647   9.5368
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -15.92951    5.81094  -2.741  0.00675 **
## abdomen       0.83165    0.05705  14.578  < 2e-16 ***
## thigh        -0.08537    0.11205  -0.762  0.44712
## neck         -0.96877    0.23741  -4.081  6.8e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.608 on 176 degrees of freedom
## Multiple R-squared:  0.7109, Adjusted R-squared:  0.7059
## F-statistic: 144.2 on 3 and 176 DF,  p-value: < 2.2e-16
```

Predicted values and Residuals

```
residual.per_fat=residuals(mod2)
predicted.per_fat=predict(mod2)
plot(residual.per_fat, predicted.per_fat)
```




```
plot(per_fat, predicted.per_fat)
```



Model Selection

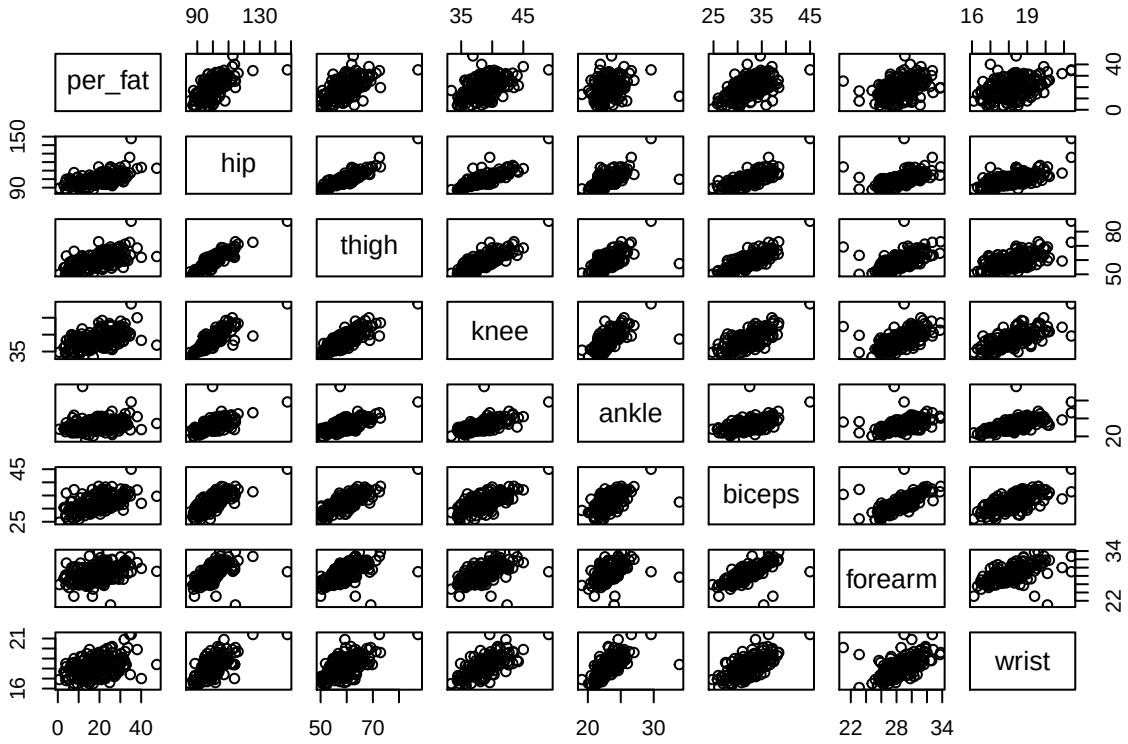
```
#remove missing data
bfat = na.omit(bfat)
colNames <- colnames(bfat[,2:14])
colNames
```

```
##  [1] "per_fat" "age"     "wt"      "ht"      "neck"    "chest"   "abdomen"
##  [8] "hip"     "thigh"   "knee"    "ankle"   "biceps"  "forearm"
```

```
#Define a function to normalize data

normalize <- function(df, cols) {
  result <- df # make a copy of the input data frame
    for (j in cols) { # each specified col
    m <- mean(df[,j]) # column mean
    std <- sd(df[,j]) # column sd

    result[,j] <- sapply(result[,j], function(x) (x - m) / std)
    }
    return(result)
    }

    bfat.norm <- normalize(bfat, colNames)
    bfat.norm = bfat.norm[,2:14]
```

Scatter Matrix for variables

```
pairs(data=bfat, ~per_fat+hip+thigh+knee+ankle+biceps
      +forearm+wrist)
```




```
model.null = lm(per_fat ~ 1, data=bfat.norm)
model.full = lm(per_fat ~., data=bfat.norm)
summary.aov(model.null)
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## Residuals   179    179       1
```

```
summary.aov(model.full)
```

```
##              Df Sum Sq Mean Sq F value   Pr(>F)
## age           1  16.20   16.20  62.544 3.39e-13 ***
## wt            1  70.04   70.04 270.321  < 2e-16 ***
## ht            1  10.16   10.16  39.222 3.09e-09 ***
## neck          1   3.22    3.22  12.432 0.000545 ***
## chest         1   3.35    3.35  12.945 0.000422 ***
## abdomen       1  27.95   27.95 107.888  < 2e-16 ***
## hip           1   0.51    0.51   1.950 0.164490
## thigh         1   2.58    2.58   9.971 0.001888 **
## knee          1   0.17    0.17   0.645 0.422930
## ankle         1   0.18    0.18   0.710 0.400548
## biceps        1   0.97    0.97   3.761 0.054135 .
## forearm       1   0.38    0.38   1.485 0.224677
## Residuals   167  43.27    0.26
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Stepwise Model Selection

```
step(model.null,
     scope = list(upper=model.full),
     direction="both", data=bfat.norm, trace = FALSE)
```

```
##
## Call:
## lm(formula = per_fat ~ abdomen + wt + biceps + neck, data = bfat.norm)
##
## Coefficients:
## (Intercept)      abdomen           wt       biceps         neck
##  -5.242e-16    1.295e+00   -5.233e-01    1.919e-01   -1.841e-01
```

Final Model

```
model.final = lm(formula = per_fat ~ abdomen + hip + wrist
                 + thigh, data = bfat.norm)
summary(model.final)
```

```
##
## Call:
## lm(formula = per_fat ~ abdomen + hip + wrist + thigh, data = bfat.norm)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.42753 -0.34588 -0.05958  0.38643  1.21356
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.10953    1.00400   3.097  0.00228 **
```

```
## abdomen      1.21037    0.08222  14.720  < 2e-16 ***
## hip         -0.47987    0.11928  -4.023 8.54e-05 ***
## wrist       -0.17077    0.05510  -3.100  0.00226 **
## thigh        0.17253    0.09121   1.892  0.06019 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5254 on 175 degrees of freedom
## Multiple R-squared:  0.7301, Adjusted R-squared:  0.724
## F-statistic: 118.4 on 4 and 175 DF,  p-value: < 2.2e-16
```

LASSO, Elastic Net and Ridge Regression

Stepwise regression assumes that the predictor variables are not highly correlated. During each step in stepwise regression, a variable is considered for addition to or subtraction from the set of predictor variables based on some pre-specified criterion (e.g. adjusted R-squared). The two main approaches involve forward selection, starting with no variables in the model, and backwards selection, starting with all candidate predictors.

Lasso (least absolute shrinkage and selection operator) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces.

The elastic net is a regularized regression method that linearly combines the L1 and L2 penalties of the lasso and ridge methods. The elastic net penalty is controlled by alpha, and bridges the gap between lasso (alpha=1) and ridge (alpha=0). Note that the ridge penalty shrinks the coefficients of correlated predictors towards each other while the lasso tends to pick one and discard the others. Ridge is generally good at prediction but tends to be less interpretable.

Data Preparation for Lasso and Ridge Regression

We will use the `glmnet` package in order to perform ridge regression and lasso. The main function in this package is `glmnet()`, which can be used to fit ridge regression models, lasso models, and more. This function has a slightly different syntax from other model-fitting functions that we have encountered thus far in this book. In particular, we must pass in an x matrix as well as a y vector, and we do not use the $y \sim x$ syntax.

```
#Prepare Data for Lasso
#building lasso

XP=data.matrix(bfat.norm[,-1])
summary(XP)
```

```
##       age                wt                 ht                neck
##  Min.   :-1.8521   Min.   :-1.79051   Min.   :-2.39664   Min.   :-2.64721
##  1st Qu.:-0.6386   1st Qu.:-0.64361   1st Qu.:-0.76438   1st Qu.:-0.66736
##  Median :-0.1128   Median :-0.08466   Median :-0.04427   Median :-0.04108
##  Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.00000
##  3rd Qu.: 0.7366   3rd Qu.: 0.58609   3rd Qu.: 0.77185   3rd Qu.: 0.64581
##  Max.   : 2.9208   Max.   : 6.09781   Max.   : 2.21208   Max.   : 5.31260
##      chest            abdomen             hip               thigh
##  Min.   :-2.0426   Min.   :-2.02119   Min.   :-1.98114   Min.   :-1.82517
##  1st Qu.:-0.8146   1st Qu.:-0.75008   1st Qu.:-0.63176   1st Qu.:-0.64667
##  Median :-0.1196   Median :-0.09759   Median :-0.07292   Median :-0.07674
##  Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.00000
##  3rd Qu.: 0.4857   3rd Qu.: 0.54362   3rd Qu.: 0.43480   3rd Qu.: 0.55115
##  Max.   : 4.0740   Max.   : 4.99591   Max.   : 6.52405   Max.   : 5.38107
```

```
##       knee              ankle             biceps            forearm
##  Min.   :-2.19317   Min.   :-2.3968   Min.   :-2.51495   Min.   :-3.85106
##  1st Qu.:-0.71889   1st Qu.:-0.6560   1st Qu.:-0.62450   1st Qu.:-0.66874
##  Median :-0.08705   Median :-0.2358   Median :-0.07347   Median :-0.01207
##  Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000
##  3rd Qu.: 0.58691   3rd Qu.: 0.6046   3rd Qu.: 0.61320   3rd Qu.: 0.65723
##  Max.   : 4.42004   Max.   : 6.4875   Max.   : 4.33475   Max.   : 2.61462
```

```
x=XP
YP=data.matrix(bfat.norm[,1])
summary(YP)
```

```
##        V1
##  Min.   :-2.1880
##  1st Qu.:-0.7346
##  Median : 0.0185
##  Mean   : 0.0000
##  3rd Qu.: 0.6952
##  Max.   : 3.3194
```

```
y=YP
lasso=cv.glmnet(x,
                y, alpha=1,
                nfolds = 5, type.measure="mse",
                family="gaussian")
```

Output the coefficients of the variables selected by lasso.

```
coef(lasso, s=lasso$lambda.min)
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) -6.404803e-16
## age          6.885520e-02
## wt          -2.714618e-01
## ht           7.913540e-03
## neck        -2.146274e-01
## chest       -3.668275e-02
## abdomen      1.231445e+00
## hip         -2.277887e-01
## thigh        2.159657e-01
## knee        -3.105270e-02
## ankle       -5.329363e-02
## biceps       8.570149e-02
## forearm      6.167666e-02
```

Ridge Regression

The `glmnet()` function has an alpha argument that determines what type of model is fit. If `alpha = 0` then a ridge regression model is fit, and if `alpha = 1` then a lasso model is fit. We first fit a ridge regression model:

```
grid = 10^seq(10, -2, length = 100)
ridge_mod = glmnet(x, y, alpha = 0, lambda = grid)
```

By default the `glmnet()` function performs ridge regression for an automatically selected range of λ values.

However, here we have chosen to implement the function over a grid of values ranging from $\lambda = 10^{10}$ to $\lambda = 10^{-2}$, essentially covering the full range of scenarios from the null model containing only the intercept, to the least squares fit.

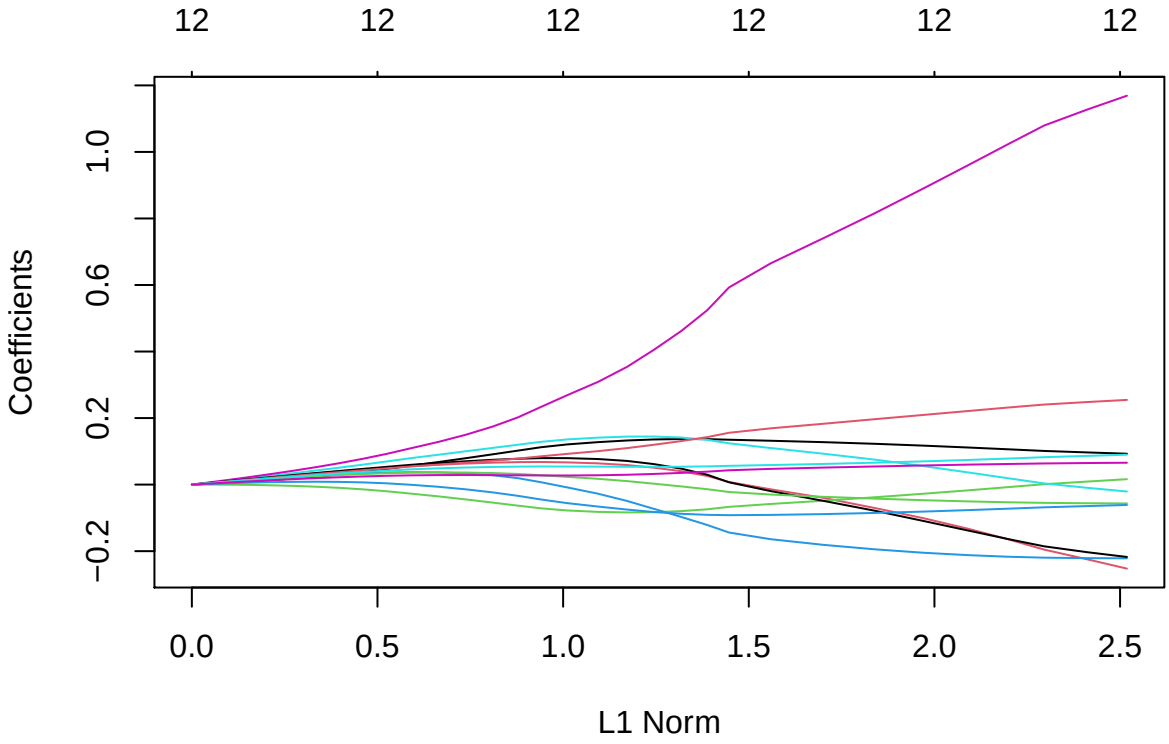
As we will see, we can also compute model fits for a particular value of λ that is not one of the original grid values. Note that by default, the `glmnet()` function standardizes the variables so that they are on the same scale. To turn off this default setting use the argument `standardize = FALSE`.

Associated with each value of λ is a vector of ridge regression coefficients, stored in a matrix that can be accessed by `coef()`. In this case, it is a 15×100 matrix, with 15 rows (one for each predictor, plus an intercept) and 100 columns (one for each value of λ).

```
dim(coef(ridge_mod))
```

```
## [1]  13 100
```

```
plot(ridge_mod)     # Draw plot of coefficients
```



We expect the coefficient estimates to be much smaller, in terms of l_2 norm when a large value of λ is used, as compared to when a small value of λ is used. These are the coefficients when $\lambda = 11498$, along with their l_2 norm:

```
ridge_mod$lambda[50] # Display 50th lambda value"
```

```
## [1] 11497.57
```

```
coef(ridge_mod)[,50] # Display coefficients associated with 50th lambda value"
```

```
##   (Intercept)           age            wt            ht          neck
## -1.526432e-16  2.609254e-05  5.445809e-05  1.388594e-06  4.591450e-05
##         chest       abdomen           hip         thigh          knee
##  6.161509e-05  7.125235e-05  5.440005e-05  5.053996e-05  4.333254e-05
##         ankle        biceps       forearm
##  2.307746e-05  4.685840e-05  3.369981e-05
```

```
sqrt(sum(coef(ridge_mod)[-1,50]^2)) # Calculate l2 norm"
```

```
## [1] 0.0001608888
```

In contrast, here are the coefficients when $\lambda = 705$, along with their l_2 norm. Note the much larger l_2 norm of the coefficients associated with this smaller value of λ."

```
ridge_mod$lambda[60] #Display 60th lambda value\n",
```

```
## [1] 705.4802
```

```
coef(ridge_mod)[,60] # Display coefficients associated with 60th lambda value\n",
```

```
##   (Intercept)           age            wt            ht          neck
## -1.524550e-16  4.243217e-04  8.804158e-04  2.003903e-05  7.415898e-04
##         chest       abdomen           hip         thigh          knee
##  9.969227e-04  1.153624e-03  8.787311e-04  8.159913e-04  6.985972e-04
##         ankle        biceps       forearm
##  3.701699e-04  7.558705e-04  5.429845e-04
```

```
sqrt(sum(coef(ridge_mod)[-1,60]^2)) # Calculate l2 norm"
```

```
## [1] 0.002599901
```

We can use the `predict()` function for a number of purposes. For instance we can obtain the ridge regression coefficients for a new value of λ, say 50:

```
predict(ridge_mod, s = 50, type =  "coefficients")
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) -1.503188e-16
## age          5.895880e-03
## wt           1.101510e-02
## ht          -2.948972e-04
## neck         9.170962e-03
## chest        1.278125e-02
## abdomen      1.503427e-02
## hip          1.107665e-02
## thigh        1.025128e-02
## knee         8.615043e-03
## ankle        4.224826e-03
## biceps       9.446253e-03
## forearm      6.691531e-03
```

We now split the samples into a training set and a test set in order to estimate the test error of ridge regression and the lasso.

```
set.seed(1)

train = bfat.norm %>%
  sample_frac(0.5)

test = bfat.norm %>%
  setdiff(train)

d_train = train[,-1]
d_test = test[,-1]
```

```
x_train = model.matrix(train$per_fat~., data=d_train)
x_test = model.matrix(test$per_fat~., data=d_test)

y_train = train %>%
  select(per_fat) %>%
  unlist() %>%
  as.numeric()

y_test = test %>%
  select(per_fat) %>%
  unlist() %>%
  as.numeric()
```

Next we fit a ridge regression model on the training set, and evaluate its MSE on the test set, using $\lambda = 4$. Note the use of the `predict()` function again: this time we get predictions for a test set, by replacing `type=\"coefficients\"` with the `newx` argument.

```
ridge_mod = glmnet(x_train, y_train, alpha=0, lambda = grid, thresh = 1e-12)
ridge_pred = predict(ridge_mod, s = 4, newx = x_test)
mean((ridge_pred - y_test)^2)
```

```
## [1] 0.6560573
```

The test MSE is 101242.7. Note that if we had instead simply fit a model with just an intercept, we would have predicted each test observation using the mean of the training observations. In that case, we could compute the test set MSE like this:

```
mean((mean(y_train) - y_test)^2)
```

```
## [1] 1.071273
```

We could also get the same result by fitting a ridge regression model with a very large value of λ. Note that `1e10` means 10^{10}.

```
ridge_pred = predict(ridge_mod, s = 1e10, newx = x_test)
mean((ridge_pred - y_test)^2)
```

```
## [1] 1.071273
```

So fitting a ridge regression model with $\lambda = 4$ leads to a much lower test MSE than fitting a model with just an intercept. We now check whether there is any benefit to performing ridge regression with $\lambda = 4$ instead of just performing least squares regression. Recall that least squares is simply ridge regression with $\lambda = 0$.

* Note: In order for `glmnet()` to yield the **exact** least squares coefficients when $\lambda = 0$ we use the argument `exact=T` when calling the `predict()` function. Otherwise, the `predict()` function will interpolate over the grid of λ values used in fitting the `glmnet()` model, yielding approximate results. Even when we use `exact = TRUE`, there remains a slight discrepancy in the third decimal place between the output of `glmnet()` when $\lambda = 0$ and the output of `lm()`; this is due to numerical approximation on the part of `glmnet()`.

```
ridge_pred = predict(ridge_mod, s = 0, newx = x_test, exact = FALSE)
mean((ridge_pred - y_test)^2)
```

```
## [1] 0.2939128
```

```
lm(per_fat~., data = train)
```

```
##
## Call:
## lm(formula = per_fat ~ ., data = train)
```

```
##
## Coefficients:
## (Intercept)          age           wt           ht         neck        chest
##    -0.04207      0.06705     -0.10845      0.06784     -0.11087     -0.11178
##     abdomen          hip        thigh         knee        ankle       biceps
##     1.20913     -0.36838      0.30119     -0.17599     -0.06556      0.01185
##     forearm
##     0.11493
```

```
predict(ridge_mod, s = 0, exact = FALSE, type="coefficients")
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) -0.03975030
## (Intercept)  .
## age          0.08942505
## wt          -0.09216846
## ht           0.06019398
## neck        -0.11133160
## chest       -0.03786918
## abdomen      1.04018522
## hip         -0.26435594
## thigh        0.27130829
## knee        -0.17971659
## ankle       -0.07036256
## biceps       0.02012120
## forearm      0.10807213
```

It looks like we are indeed improving over regular least-squares! Side note: in general, if we want to fit a (unpenalized) least squares model, then we should use the `lm()` function, since that function provides more useful outputs, such as standard errors and p-values for the coefficients.

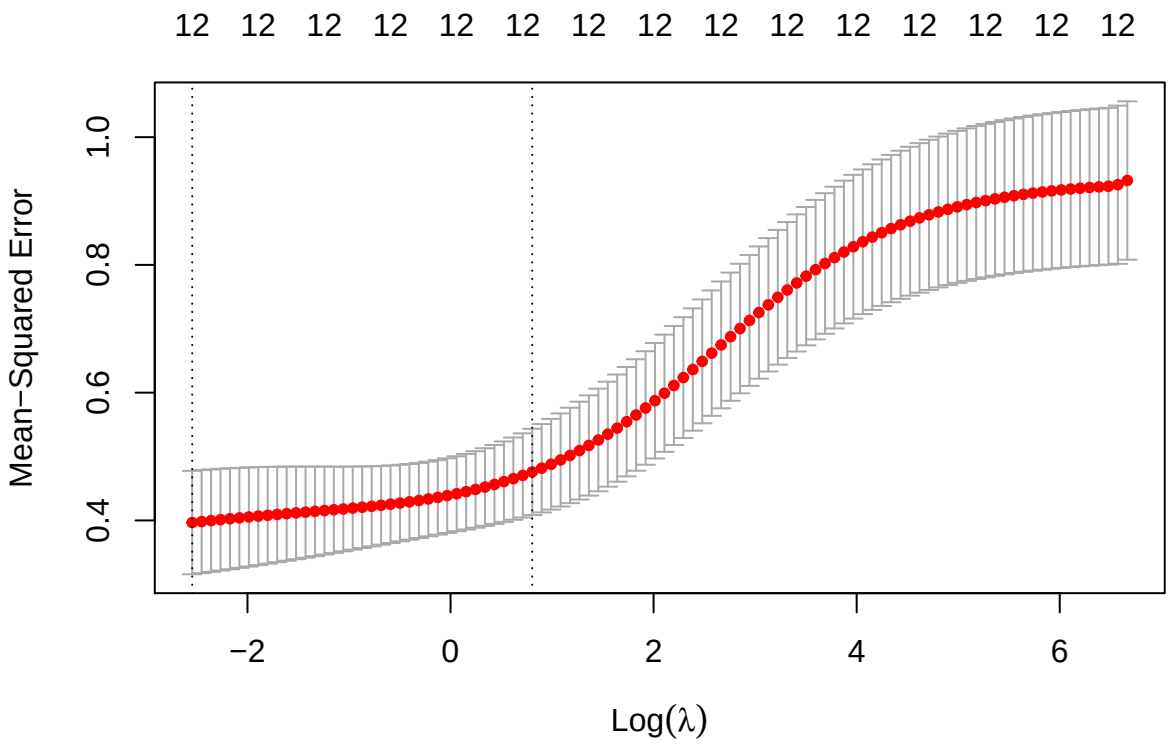
Instead of arbitrarily choosing $\lambda = 4$, it would be better to use cross-validation to choose the tuning parameter λ. We can do this using the built-in cross-validation function, `cv.glmnet()`. By default, the function performs 10-fold cross-validation, though this can be changed using the argument `folds`. Note that we set a random seed first so our results will be reproducible, since the choice of the cross-validation folds is random.

```
set.seed(1)

cv.out = cv.glmnet(x_train, y_train, alpha = 0) # Fit ridge regression model on training data
bestlam = cv.out$lambda.min  # Select lamda that minimizes training MSE
bestlam
```

```
## [1] 0.07850514
```

Therefore, we see that the value of λ that results in the smallest cross-validation error is 339.1845 We can also plot the MSE as a function of λ:

```
plot(cv.out) # Draw plot of training MSE as a function of lambda
```



What is the test MSE associated with this value of λ?

```
ridge_pred = predict(ridge_mod, s = bestlam, newx = x_test) # Use best lambda to predict test data
mean((ridge_pred - y_test)^2) # Calculate test MSE
```

```
## [1] 0.3473058
```

This represents a further improvement over the test MSE that we got using $\lambda = 4$. Finally, we refit our ridge regression model on the full data set using the value of λ chosen by cross-validation, and examine the coefficient estimates.

```
out = glmnet(x, y, alpha = 0) # Fit ridge regression model on full dataset
predict(out, type = "coefficients", s = bestlam) # Display coefficients using lambda chosen by CV
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                              1
## (Intercept) -5.369552e-16
## age          1.298104e-01
## wt          -2.686305e-02
## ht          -5.341804e-02
## neck        -1.722698e-01
## chest        1.033859e-01
## abdomen      7.002786e-01
## hip         -3.260382e-02
## thigh        1.759446e-01
## knee        -3.318926e-02
## ankle       -9.015065e-02
## biceps       5.988103e-02
## forearm      4.888949e-02
```

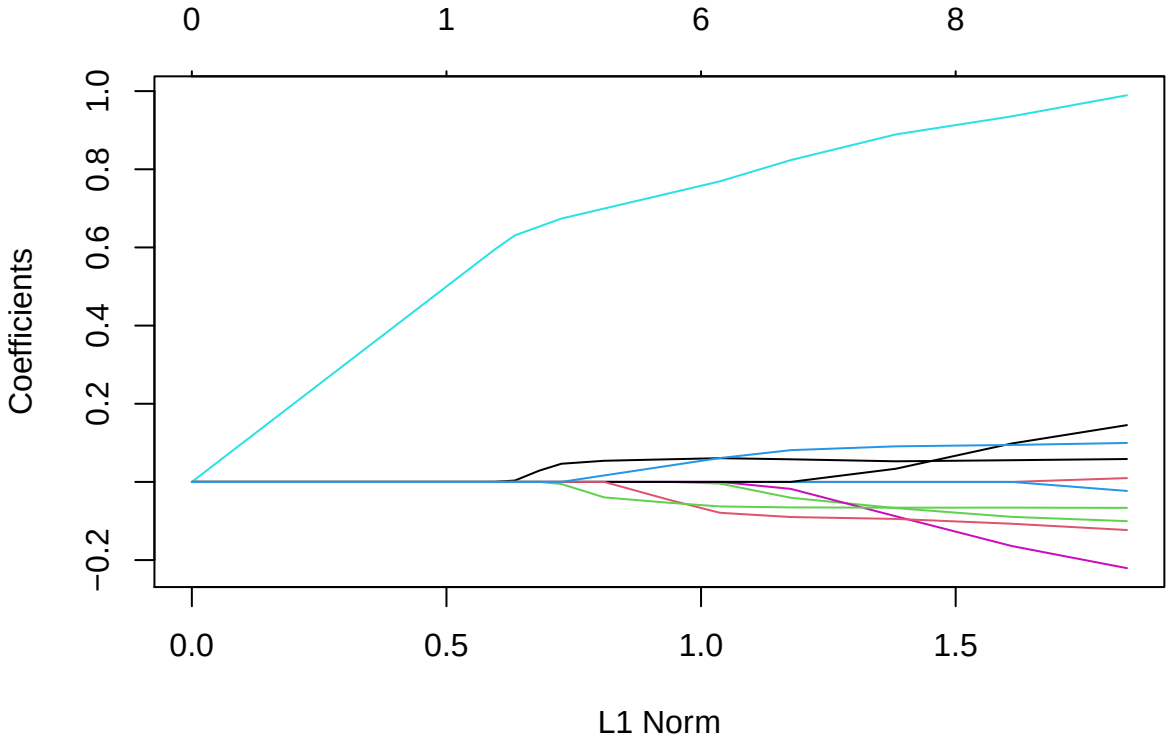
As expected, none of the coefficients are exactly zero - ridge regression does not perform variable selection!

The Lasso

We saw that ridge regression with a wise choice of λ can outperform least squares as well as the null model on the Hitters data set. We now ask whether the lasso can yield either a more accurate or a more interpretable model than ridge regression. In order to fit a lasso model, we once again use the `glmnet()` function; however, this time we use the argument `alpha=1`. Other than that change, we proceed just as we did in fitting a ridge model:

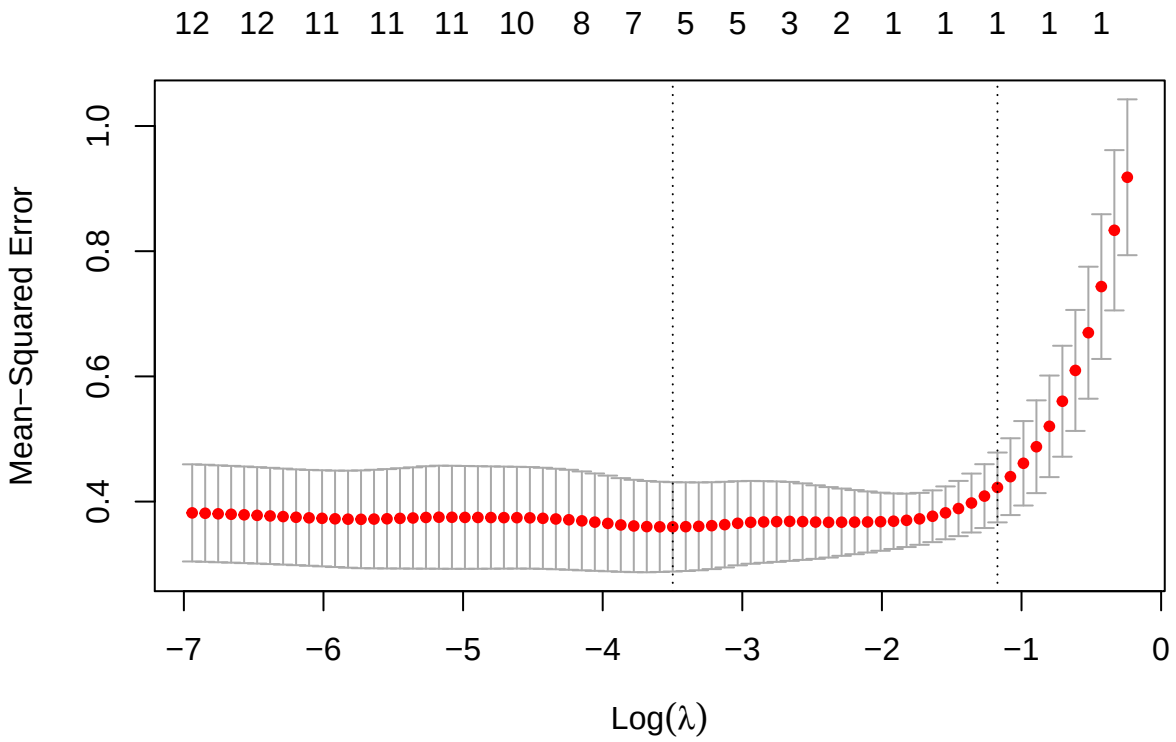
```
lasso_mod = glmnet(x_train,
                   y_train,
                   alpha = 1,
                   lambda = grid) # Fit lasso model on training data

plot(lasso_mod)     # Draw plot of coefficients
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```



Notice that in the coefficient plot that depending on the choice of tuning parameter, some of the coefficients are exactly equal to zero. We now perform cross-validation and compute the associated test error:

```
set.seed(1)
cv.out = cv.glmnet(x_train, y_train, alpha = 1) # Fit lasso model on training data
plot(cv.out) # Draw plot of training MSE as a function of lambda
```




```
bestlam = cv.out$lambda.min # Select lamda that minimizes training MSE
lasso_pred = predict(lasso_mod, s = bestlam, newx = x_test) # Use best lambda to predict test data
mean((lasso_pred - y_test)^2) # Calculate test MSE
```

```
## [1] 0.3392375
```

This is substantially lower than the test set MSE of the null model and of least squares, and very similar to the test MSE of ridge regression with λ chosen by cross-validation.

However, the lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse. Here we see that 12 of the 14 coefficient estimates are exactly zero:

```
out = glmnet(x, y, alpha = 1, lambda = grid) # Fit lasso model on full dataset
lasso_coef = predict(out, type = "coefficients",
s = bestlam) # Display coefficients using lambda chosen by CV
lasso_coef
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) -5.488650e-16
## age          5.148006e-02
## wt           .
## ht          -4.491412e-02
## neck        -1.127499e-01
## chest        .
## abdomen      8.975255e-01
## hip         -6.285664e-04
## thigh        .
## knee         .
## ankle       -6.341860e-02
## biceps       3.817580e-04
## forearm      1.963505e-02
```

Selecting only the predictors with non-zero (greater than 1e-02) coefficients, we see that the lasso model with λ chosen by cross-validation contains only seven variables:

```
lasso_coef[lasso_coef > 1e-02] # Display only non-zero coefficients
```

```
## <sparse>[ <logic> ] : .M.sub.i.logical() maybe inefficient
```

```
## [1] 0.05148006 0.89752545 0.01963505
```

Check for Multcollinearity

This file produces the resulte for VIF using R

Method 1

```
##    Variables       VIF
## 1    per_fat  4.136974
## 2        age  1.877750
## 3         wt 47.879102
## 4         ht  3.017498
## 5       neck  4.871078
## 6      chest 11.262578
## 7    abdomen 19.457248
## 8        hip 15.475037
## 9      thigh  9.126259
## 10      knee  5.010651
## 11     ankle  2.207485
## 12    biceps  4.343836
## 13   forearm  2.170823
```

Method 2

```
# Source of function is
# http://highstat.com/Books/BGS/GAMM/RCodeP2/HighstatLibV6.R

#To use:  corvif(YourDataFile)
corvif <- function(dataz) {
  dataz <- as.data.frame(dataz)
  #correlation part
  #cat("Correlations of the variables\n\n")
  #tmp_cor <- cor(dataz,use="complete.obs")
  #print(tmp_cor)

  #vif part
  form    <- formula(paste("fooy ~ ",paste(strsplit(names(dataz)," "),
                           collapse=" + ")))
  dataz   <- data.frame(fooy=1,dataz)
  lm_mod  <- lm(form,dataz)

  cat("\n\nVariance inflation factors\n\n")
#  print(myvif(lm_mod))
}
```

The results for the Body Fat data are

```
myvif = corvif(bfat.norm)
```

```
##
##
## Variance inflation factors
```

```
myvif
```

```
## NULL
```

Method 3

```
## Install mcvis package

library(mcvis)
####################################
#devtools::install_github("kevinwang09/mcvis")
####################################
```

```
X = cbind(age,wt,ht,neck,chest,abdomen,hip,thigh,knee,
          ankle,biceps,forearm)
mcvis_result = mcvis(X = X)
mcvis_result
```

```
##       age   wt ht neck chest abdomen  hip thigh knee ankle biceps forearm
## tau12   0 0.93  0    0  0.01       0 0.04     0    0     0      0       0
```

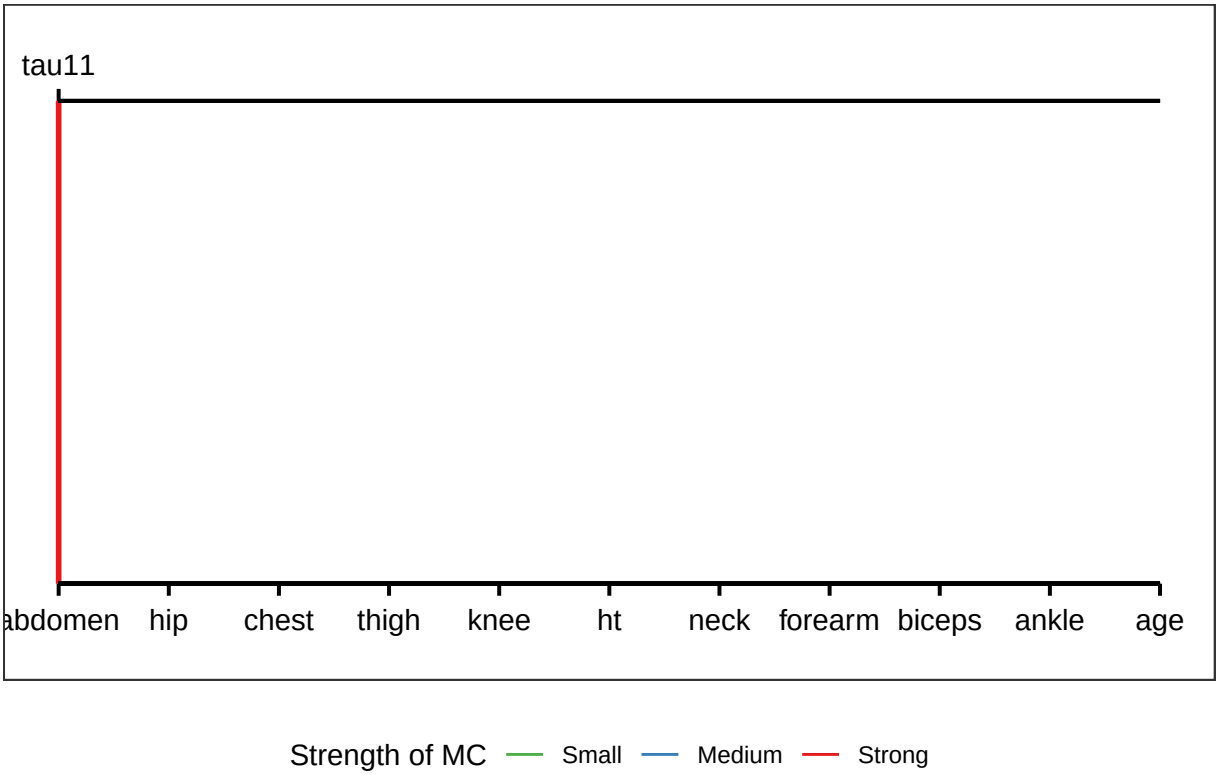
```
plot(mcvis_result)
```

Remove wt

```
X = cbind(age,ht,neck,chest,abdomen,hip,thigh,knee,
          ankle,biceps,forearm)
mcvis_result = mcvis(X = X)
mcvis_result
```

```
##       age   ht neck chest abdomen hip thigh knee ankle biceps forearm
## tau11   0 0.01 0.01  0.08    0.75 0.1  0.02 0.02  0.01   0.01    0.01
```

```
plot(mcvis_result)
```

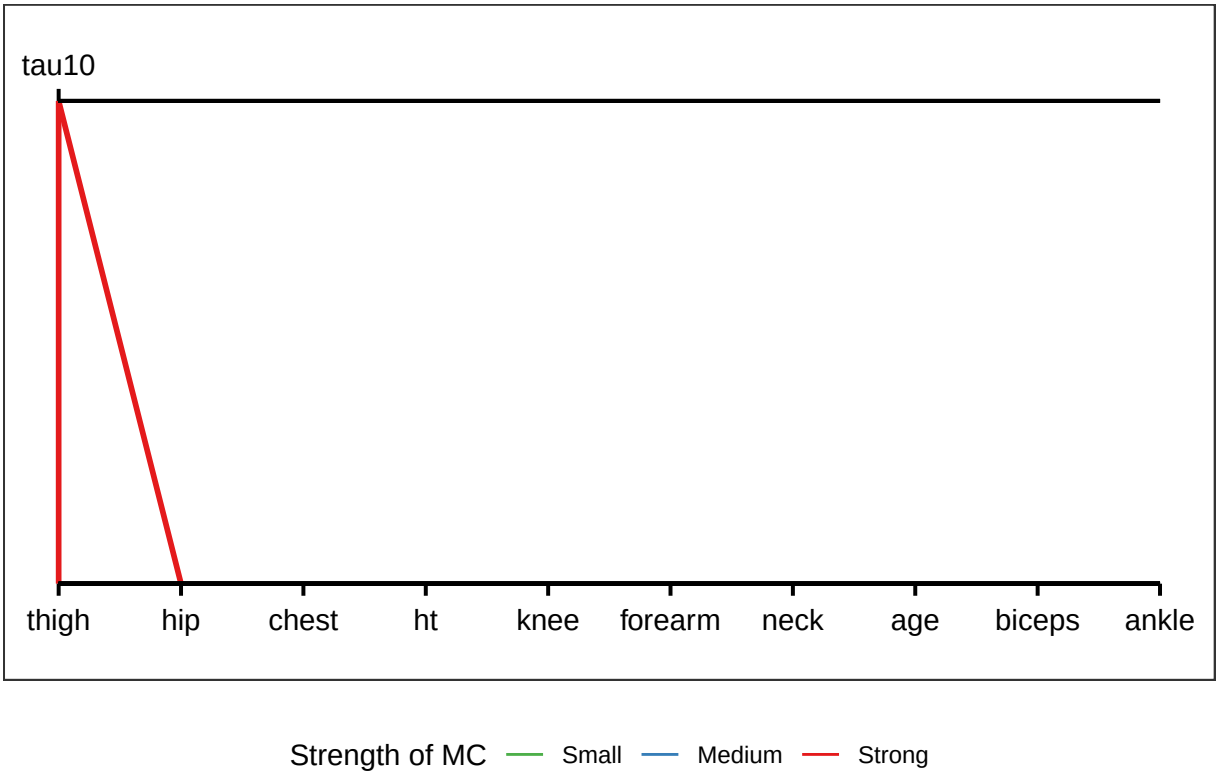


Remove Abdomen

```
X = cbind(age,ht,neck,chest,hip,thigh,knee,
          ankle,biceps,forearm)
mcvis_result = mcvis(X = X)
mcvis_result
```

```
##         age   ht neck chest hip thigh knee ankle biceps forearm
## tau10 0.01 0.02 0.01  0.05 0.4  0.49 0.01     0      0    0.01
```

```
plot(mcvis_result)
```



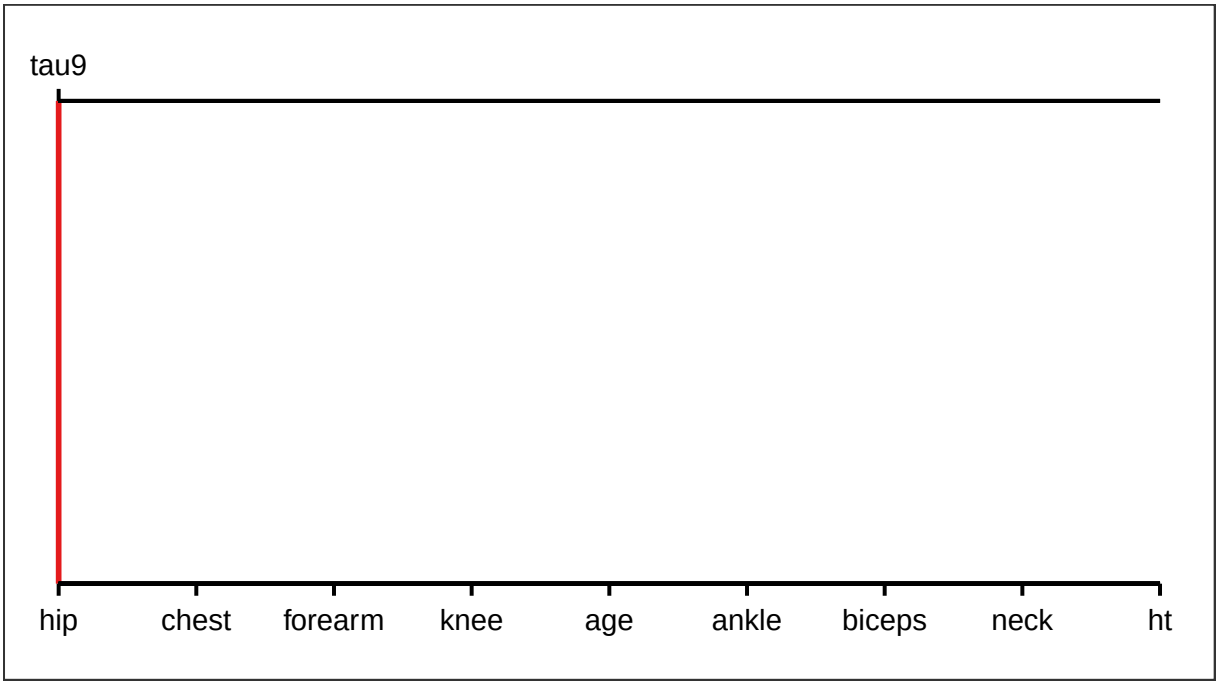
Remove thigh

```
X = cbind(age,ht,neck,chest,hip,knee,
          ankle,biceps,forearm)
mcvis_result = mcvis(X = X)
mcvis_result
```

```
##       age ht neck chest hip knee ankle biceps forearm
## tau9 0.01  0 0.01  0.12 0.7 0.06  0.01   0.01    0.09
```

```
plot(mcvis_result)
```

Multi−collinearity plot



Strength of MC — Small — Medium — Strong