Regression

```
## -- Attaching core tidyverse packages ------------------------ tidyverse
2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts()
--
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force
all conflicts to become errors
```

Linear regression

The linear regression model is

$$y_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \epsilon_i$$

where ϵ_i are iid Normal(0,σ^2)

We will use the body fat data.

The data has 86 rows (cases), a response (bfat) and 7 predictor variables.

```
Warning:  package 'seriation' was built under R version 4.2.3
```

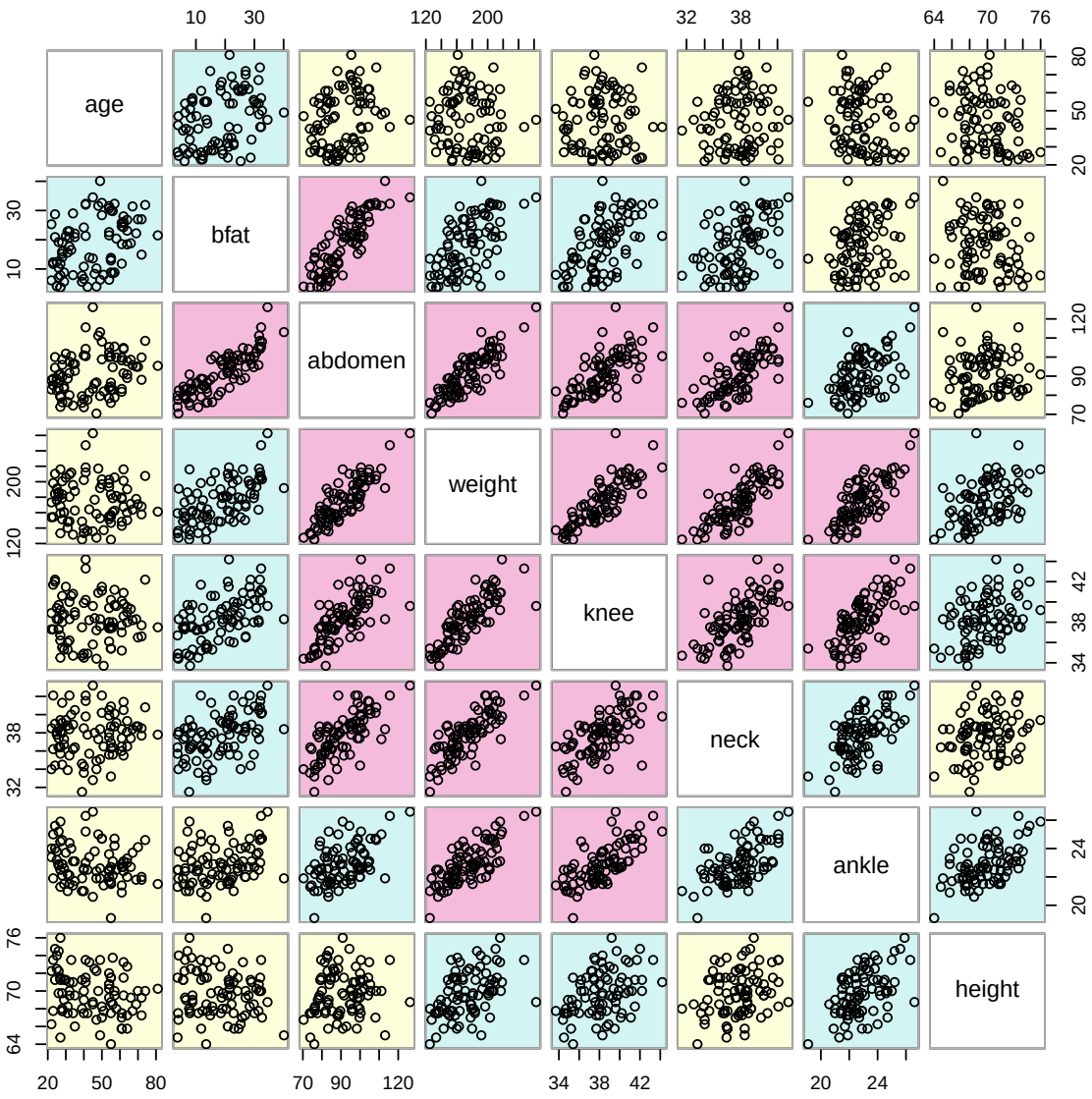
```
bfat <- read.table("https://raw.githubusercontent.com/rafamoral/courses/main/intro_sta
head(bfat)

  bfat age weight height neck abdomen knee ankle
1 12.3  23    154   67.8 36.2    85.2 37.3  21.9
```

```
2  6.1  22    173   72.2 38.5    83.0 37.3  23.4
3 25.3  22    154   66.2 34.0    87.9 38.9  24.0
4 10.4  26    185   72.2 37.4    86.4 37.3  22.8
5 28.7  24    184   71.2 34.4   100.0 42.2  24.0
6 20.9  24    210   74.8 39.0    94.4 42.0  25.6
```

First we make a pairs plot of the variables

opairs is a fancy version of pairs that shows which variables have high correlation.

```
source("https://raw.githubusercontent.com/rafamoral/courses/main/intro_stats_ml/script
opairs(bfat)
```



Next fit a linear regression model:

```
f <- lm(bfat ~ ., data = bfat)
summary(f)


Call:
lm(formula = bfat ~ ., data = bfat)

Residuals:
   Min     1Q Median     3Q    Max
-8.133 -2.926 -0.775  3.351  8.478

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -13.8993    26.1396   -0.53    0.596
age           0.0351     0.0392    0.89    0.374
weight       -0.0943     0.0745   -1.27    0.209
height       -0.6232     0.2693   -2.31    0.023 *
neck         -0.5094     0.3359   -1.52    0.133
abdomen       0.9006     0.1370    6.57    5e-09 ***
knee          0.4512     0.3769    1.20    0.235
ankle         0.4560     0.5905    0.77    0.442
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.21 on 78 degrees of freedom
Multiple R-squared:  0.81,Adjusted R-squared:  0.793
F-statistic: 47.5 on 7 and 78 DF,  p-value: <2e-16
```

- If we fit a regression model with too few predictors the model will systematically underestimate some responses and overestimate others. Such a model produces **biased** predictions.
- If we fit a regression model with good predictors but some extra uneeded predictors, the fit could have low bias but high variability, ie it would change a lot if we had a different set of data.

If there is bias, it is often evident in residual plots versus fitted values and predictors, where it shows up as curvature.
If there is high variability due to the inclusion of extra predictors in the model, this can be suggested by

1. Strong correlation among the predictors. This is called collinearity. We see plenty in the pairs plot.

2. In the regression summary, predictors are unexpectedly non-significant and may have an unexpected sign.
   In the output, only two predictors are significant. Neck negatively impacts on bodyfat, which is unexpected.

3. You can also check variance inflation factors.

All possible regressions

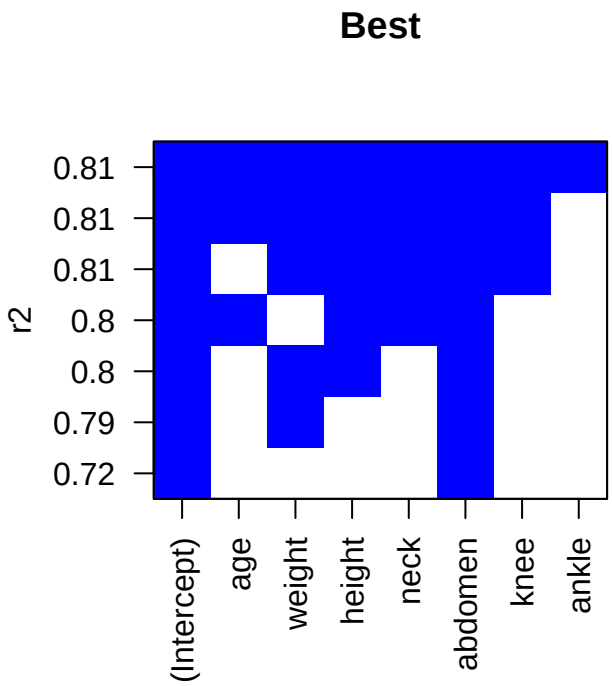
Setup: Suppose there are k available predictors. Then there are 2^k possible model choices, discounting transformations, interactions etc.

In R the regsubsets function in package leaps will calculate the regressions. By default it reports only the best regression for each number of predictors.

```
library(leaps)
allfits <- regsubsets(bfat ~ ., data = bfat)
summary(allfits)$which
```

```
  (Intercept)   age weight height  neck abdomen  knee ankle
1        TRUE FALSE  FALSE  FALSE FALSE    TRUE FALSE FALSE
2        TRUE FALSE   TRUE  FALSE FALSE    TRUE FALSE FALSE
3        TRUE FALSE   TRUE   TRUE FALSE    TRUE FALSE FALSE
4        TRUE  TRUE  FALSE   TRUE  TRUE    TRUE FALSE FALSE
5        TRUE FALSE   TRUE   TRUE  TRUE    TRUE  TRUE FALSE
6        TRUE  TRUE   TRUE   TRUE  TRUE    TRUE  TRUE FALSE
7        TRUE  TRUE   TRUE   TRUE  TRUE    TRUE  TRUE  TRUE
```

```
par(mar=c(3,3,0,0))
plot(allfits, scale = "r2", col = "blue", main = "Best")
```

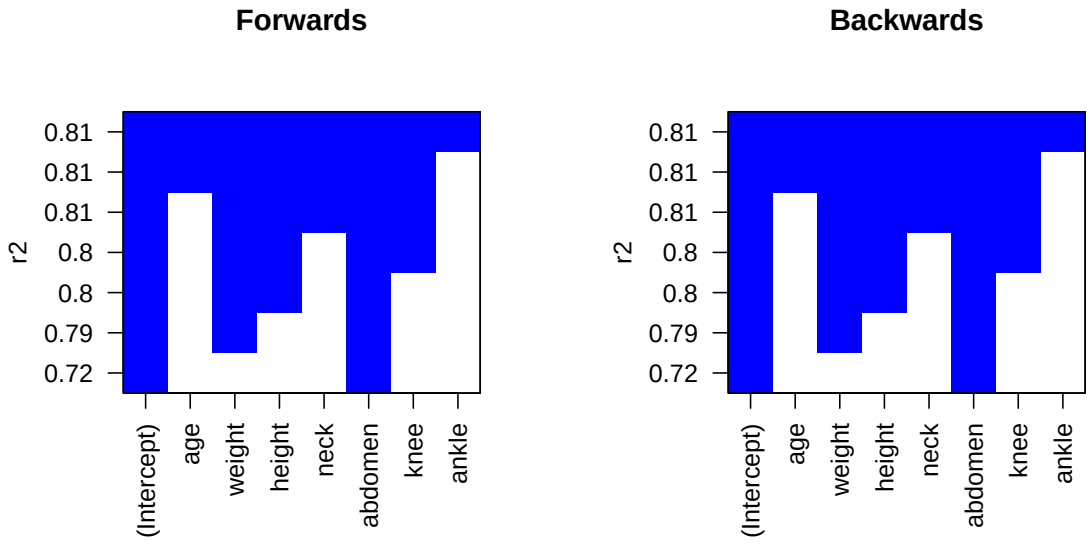


When there are a very large number of predictors, looking at all possible models might not be feasible.

Then using sequential methods is an alternative: supply `method= "backward"` or `method = "forward"` as an input to regsubsets.

In the wide data setting where the number of predictors is more than the number of cases, only `method = "forward"` is an option.

```
fwfits <- regsubsets(bfat ~ ., data = bfat, method = "forward")
bwfits <- regsubsets(bfat ~ ., data = bfat, method = "backward")
par(mfrow=c(1,2))
par(mar=c(3,3,0,0))
plot(fwfits, scale = "r2", col = "blue", main = "Forwards")
plot(bwfits, scale = "r2", col = "blue", main = "Backwards")
```



The forwards algorithm finds the best single predictor- here abdomen, then finds the next best predictor to include along with abdomen, here weight. After that height, knee, neck, age and ankle are added in that order.

The backwards algorithm first fits all predictors, and then drops the worst one, here ankle, then drops the next worst age, and continues to drop variables neck, knee height and weight in that order.
In this example, forwards and backwards visit the same sequence of models.

Both visit the overall best 3 variable model, weight, height and abdomen.

Neither visits the best 4 variable model, age, height, neck and abdomen.

Measures for comparing model fits

1. R^2 = SSR/SST = 1 - SSE/SST
2. $\hat{\sigma}^2$
3. $C_p = \text{SSE}/\hat{\sigma}_F^2 + 2p - n$

   Here $\hat{\sigma}_F^2$ denotes the best available estimate of σ^2, usually from the full model, i.e. with all predictors included.

   p is the number of model parameters, usually the number of predictors + 1 for the

intercept.

Models with small values of Cp are preferred.

4. AIC is based on the likelihood. Small values are preferred. It is equivalent to Cp for linear regression models.

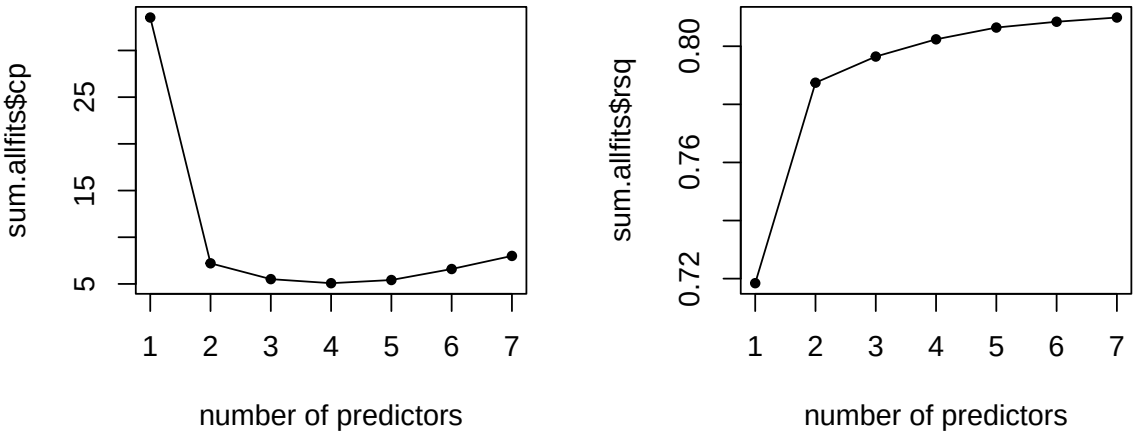
5. BIC is based on the likelihood. Small values are preferred.

All of these measures can be extracted from regsubsets.

```
sum.allfits <- summary(allfits)
names(sum.allfits)

[1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

We can see how C_p and R^2 behave as the number of predictors increase.

```
par(mfrow = c(1,2))
npred <- 1:7
plot(npred, sum.allfits$cp, pch = 20, xlab = "number of predictors")
lines(npred, sum.allfits$cp)
plot(npred, sum.allfits$rsq, pch = 20, xlab = "number of predictors")
lines(npred, sum.allfits$rsq)
```



You can see that R^2 increases as the number of predictors increases.

The model with smallest C_p is the 4 predictor model. The predictors are age, height, neck abdomen.

The 3 predictor model (weight, height, abdomen) also has a low Cp

```
f <- lm(bfat ~ age + height + neck + abdomen, data = bfat)
summary(f)


Call:
lm(formula = bfat ~ age + height + neck + abdomen, data = bfat)

Residuals:
   Min     1Q Median     3Q    Max
-9.405 -3.056 -0.481  3.423  8.622

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  16.9559    13.7507    1.23  0.22111
age           0.0504     0.0320    1.57  0.11966
height       -0.7230     0.1948   -3.71  0.00038 ***
neck         -0.6406     0.2814   -2.28  0.02542 *
abdomen       0.8087     0.0612   13.21  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.21 on 81 degrees of freedom
Multiple R-squared:  0.802,Adjusted R-squared:  0.793
F-statistic: 82.2 on 4 and 81 DF,  p-value: <2e-16
```

Notice this fit has predictors that are not significant. Requiring all p values to be below .05 is too strict in this context.

- In the above example, we used the full dataset to select a good model or models.
- We have seen previously that training error SSE/n is usually lower than test error. There is a danger of overfitting to the training set, ie choosing a model that is too flexible.
- Comparing models via C_p guards against this overfitting, becuase C_p includes a penalty for the number of model parameters.

Validation via a test set

In the above example, instead of using C_p to compare model fits, we could have used the training and test set approach.

```
set.seed(123)
s <- sample(nrow(bfat), 21)
bfatTrain <- bfat[-s,]
bfatTest <- bfat[s,]
f1 <- lm(bfat ~ age + height + neck + abdomen, data = bfatTrain)
mean(residuals(f1)^2) # train

[1] 16.9

pred1 <- predict(f1, bfatTest)
mean((pred1 - bfatTest$bfat)^2) # test

[1] 16.8

f2 <- lm(bfat ~ weight + height + abdomen, data = bfatTrain)
mean(residuals(f2)^2) # train

[1] 17.2

pred2 <- predict(f2, bfatTest)
mean((pred2 - bfatTest$bfat)^2) # test

[1] 17.9
```

In this calculation the 4 predictor fit had a better test performance.
But, this might change if we ran the calculation again, ie with a different seed. (More likely to happen with small data sets)

Recap on training, test data

- A training set of data is used to fit a model.
- Test data is a designated set of data used to evaluate a model fit in the form of the test error rate.

- As the model is fit to the training set, the training error rate is usually smaller than the test error rate, especially when the model is overfit.

In most examples, such a test data set is not available so we have randomly split the available data.

The pretend test data is more accurately termed a **validation set**.
Some drawbacks with this approach

1. The estimate of test error will depend on the randomly selected validation set, especially for smaller n.

2. If we also use the validation set to compare model fits and **select a final model** this will produce a too small test error estimate (ie biased downwards)

Solutions to this issue are

- give up on the validation set and use Cp, AIC, BIC on the full available data to select a model.

- A more general alternative is cross-validation. This is good for both model selection and test error estimation.

Validation and cross validation

Cross validation is a generalisation of the test and training set procedure.
It is a way of estimating test error rate that is more accurate because it is less dependent on the random split into test and training sets.
The k-fold cross-validation procedure is

1. Split the n observations into k groups of roughly equal size. Call them Fold_1, Fold_2.... Fold_k.

2. Apply a model fitting procedure to all data except Fold_1. Calculate the test MSE on Fold_1. Call this MSE_1.

3. Repeat step 2 for each of Fold_2 to Fold_k

4. Calculate the average MSE on the hold out sets:

$$\text{CV} = \frac{1}{k}\sum_{1}^{k}\text{MSE}_i$$

The average MSE is called the k-fold cross-validation estimate for test MSE.

The advantages and disadvantages of this approach are

1. It uses all the data to estimate the test MSE. and so produces a more reliable estimate of test MSE than that based on a single test sample.
2. If the k-fold CV estimate is used for model selection, the selected model will be better than that from a single test set.
3. It works for any model fitting procedure, not just regression.
4. The choice of k is usually between 5 and 10 which have been shown to give accurate error estimates.
5. If $k = n$, the procedure is called *leave-one-out cross validation* (LOOCV).
6. If the model fitting procedure is time consuming, this process will be time consuming.

Note: with linear least squares regression the leave-one-out cross validation error rate is known as the PRESS statistic and is very efficiently calculated.

Model choice via cross validation

Cross validation may be used to select a model fit

Calculate CV for the competing model fits.

The winning procedure is that with the lowest CV.

Then refit the winning model using the complete training data.

CV example, bodyfat

We will implement this for the body fat data, using $k = 5$.

First construct a vector fold. We will use this to assign each observation to one of 5 folds.

```
set.seed(123)
k <- 5
fold <- sample(k, nrow(bfat), replace=T)
fold

 [1] 3 3 2 2 3 5 4 1 2 3 5 3 3 1 4 1 1 5 3 2 2 1 3 4 1 3 5 4 2 5 1 1 2 3 4 5 5 3
[39] 1 2 5 5 4 5 2 1 1 3 1 5 1 2 4 4 3 1 2 1 2 4 5 5 3 1 4 1 1 3 4 1 3 5 3 2 5 5
[77] 3 2 2 2 4 2 2 4 4 1

fsize <- table(fold)
```

Calculate the CV error rate for the age, height, neck, abdomen model fit.

```
mse <- vector(length = k)
for(i in 1:k) {
  foldi <- bfat[fold == i,]
  foldOther <- bfat[fold != i,]
  f <- lm(bfat ~ age + height + neck + abdomen, foldOther)
  pred <- predict(f, foldi)
  mse[i] <- mean((pred - foldi$bfat)^2) # MSEi
}
mse

[1] 23.3 20.4 14.1 17.4 20.3

cv <- weighted.mean(mse, fsize)
cv

[1] 19.3
```

Calculate the CV error rate for the weight, height, abdomen model fit.

```
mse <- vector(length = k)
for(i in 1:k) {
  foldi <- bfat[fold == i,]
  foldOther <- bfat[fold != i,]
  f <- lm(bfat ~ weight + height + abdomen, foldOther)
  pred <- predict(f, foldi)
```

```
  mse[i] <-mean((pred - foldi$bfat)^2) # MSEi
}
mse

[1] 24.0 21.3 15.5 16.1 18.4

cv<- weighted.mean(mse, fsize)
cv

[1] 19.3
```

The cross validation method prefers the three variable model.

One should then refit this model on the entire data set.
Any model fitting procedure could be used here in place of lm. For example, using regsubsets with all predictors to select the lowest Cp model on that data subset.

Recap: the bias-variance trade-off

It can be shown that for a given value of x_0, the expected test MSE can be decomposed into the sum of three quantities: the variance of $\hat{f}(x_0)$, the squared bias of $\hat{f}(x_0)$ and the variance of the error terms ϵ. The expected test MSE: the average test MSE that we would get if we repeatedly estimated f using a large number of training sets and tested each one at x_0.
Variance of $\hat{f}(x_0)$: the amount by which $\hat{f}$ would change if estimated by a different training set. More flexible methods for estimating $\hat{f}$ will have higher variance.
Bias of $\hat{f}(x_0)$: error that is introduced by approximating a real life problem, which may be extremly complicated by a simple model. Generally, more flexible methods result in less bias.

High dimensional data

Traditional statistical techniques for regression and classification are intended for low dimensional setting $n >> p$. These approaches are often not appropriate for high dimensional settings (i.e. $n \leq p$).

For example, let $p = 1$ and $n = 2$. The least squares fit will go through both points and the residuals will be equal to 0. This 'perfect' fit overfits the data and will generalise badly. In

general, if $n \approx p$, the OLS is too flexible and overfits the data.

Model performance should always be evaluated on an independent test set.

The approaches we have seen for reducing the number of variables C_p, AIC, BIC are also not appropriate for high dimensional data settings as estimating $\hat{\sigma}^2$ is problematic (in the example above, $\hat{\sigma}^2 = 0$).

Curse of dimensionality: when dimensionality increases, the volume of the space increases so fast that the available data become sparse.

Adding features that are truly associated with the response will in general improve the fit (i.e. reduce test error), but adding spurious features will lead to a deterioration of the fitted model. Even for the relevant features, their inclusion sometimes incurs the increase in variance which is not offset by the reduction in bias.

In high dimensions ($n << p$), multicollinearity is extreme: any variable can be written as a linear combination of all the other variables in the model. So we can never really know which variables are truly predictive and what is the best subset of them. A different training set is likely to identify a different subset of features as useful for prediction.

Shrinkage methods

Recall that the idea behind dropping out predictors from a regression model is that too many predictors can result in an overly flexible model where predictions and some coefficients have large variance.
As an alternative to omitting predictors, shrinkage methods downweight the contribution of predictors by shrinking coefficient estimates towards zero.
We will look at two algorithms that do this, ridge regression and Lasso.
These methods are particularly useful for regression with wide data.

Ridge regression

Recall that least squares regression finds β_0, β_1, β_2,...,β_k to minimise

$$\text{SSE} = \sum_1^n (y_i - \beta_0 - \beta_1 x_{i1} - .... - \beta_k x_{ik})^2$$

Ridge regression seeks β values to minimise

$$\text{SSE} + \lambda \sum_1^k \beta_j^2$$

- Here λ is a parameter that must be selected. $\lambda = 0$ is just least squares.
- The quantity $\lambda \sum_1^k \beta_j^2$ penalises large β values.
- The larger the λ the more the β estimates are shrunk.
- Predictors are standardised to standard deviation 1 prior to calculating this criterion.
- Cross validation is used to select λ.
- Coefficient standard errors and prediction SE's are available.
- $||\beta||_2 = \sqrt{\sum_1^k \beta_j^2}$, the l_2 norm. $||\hat{\beta}||_2$ is a measure of distance of $\hat{\beta}$ to 0 and it decreases as λ increases.
- Whereas the OLS solution is

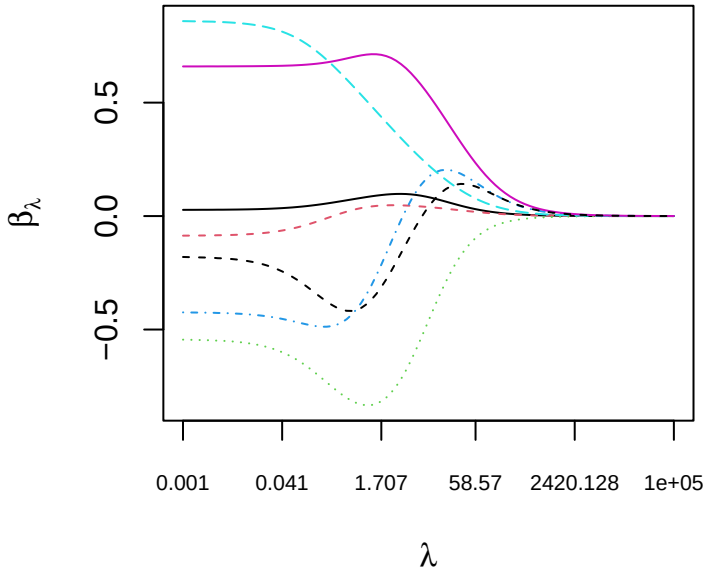
  $$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y},$$

  the ridge regression solution can be shown to be:

  $$\hat{\beta} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}.$$

Lasso and ridge regression can be calculated with the package glmnet.

```
library(glmnet)
x <- model.matrix(bfat ~ . - 1, data = bfatTrain)
y <- bfatTrain$bfat
grid <- 10^seq(-3, 5, length = 100)
ridge_fit <- glmnet(x, y, alpha = 0, lambda = grid) # alpha = 0 for ridge
```




```
set.seed(2024)
cv_out <- cv.glmnet(x, y, alpha = 0)
cv_out$lambda.min

[1] 0.784

ridge_fit2 <- glmnet(x, y, alpha = 0,
                     lambda = cv_out$lambda.min)
coef(ridge_fit2)

8 x 1 sparse Matrix of class "dgCMatrix"
                   s0
(Intercept) 14.8432
age          0.0798
weight       0.0348
height      -0.8300
neck        -0.3862
abdomen      0.5369
knee         0.7084
ankle       -0.4058

coef(lm(bfat ~ ., data = bfatTrain))
```

```
(Intercept)         age      weight      height        neck     abdomen
   -13.5964      0.0269     -0.0886     -0.5396     -0.4211      0.8632
       knee       ankle
     0.6607     -0.1719
```

Ridge regression advantage over least squares is related to the bias variance trade-off. As λ increases, the flexibility of the regression fit decreases, leading to decreased variance but increased bias (since the regression coefficients are underestimated).

For wide data (p is large relative to n), the least squares estimates will have a high variance and at $p > n$ there isn't a unique solution, whereas ridge regression can still perform well.

Compared to best subsets selection, which requires searching through 2^p models, there are significant computational advantages to ridge regression.

Lasso

Lasso is similar to ridge. It seeks β values to minimise

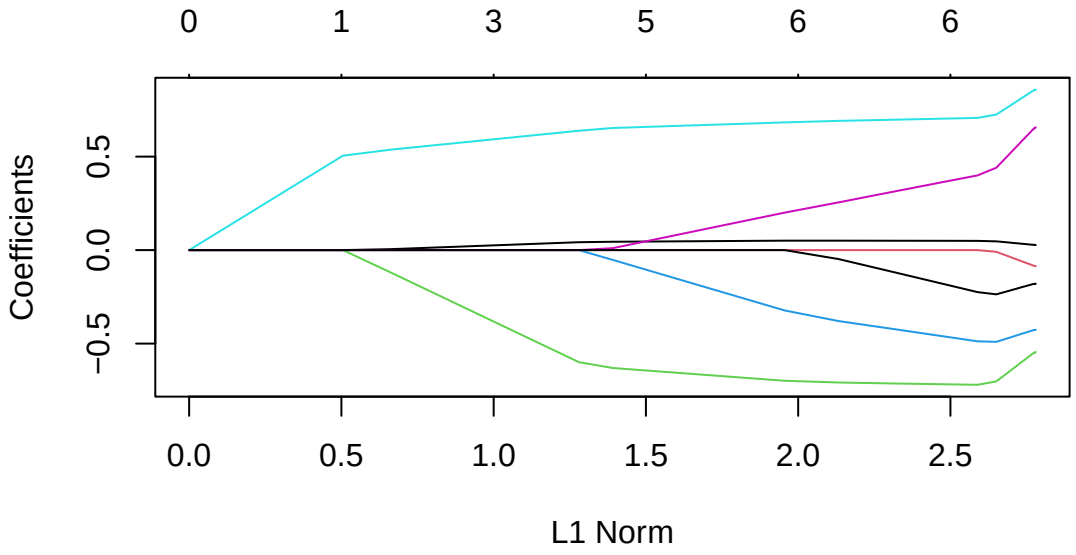
$$\text{SSE} + \lambda \sum_{1}^{k} |\beta_j|$$

Thus, it has the l_1 penalty ($||\beta||_1 = \sum_1^k |\beta_j|$). As with ridge regression, increasing λ shrinks the coefficinets to 0, however, the l_1 penalty has the effect of forcing some β_j to zero. In this way, it does automatic variable selection.

Coefficient standard errors and prediction SE's are not available for lasso regression.

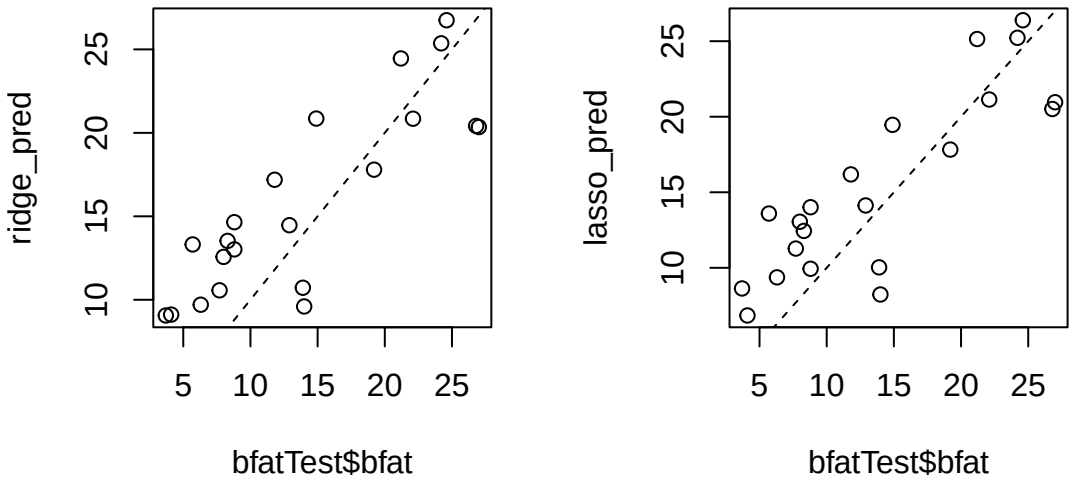
```
lasso_fit <- glmnet(x, y, alpha = 1, lambda = grid) # alpha = 1 for lasso
plot(lasso_fit)
```

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




```
cv_out <- cv.glmnet(x, y, alpha = 1)
cv_out$lambda.min

[1] 0.0169

lasso_fit2 <- glmnet(x, y, alpha = 1,
                     lambda = cv_out$lambda.min)
coef(lasso_fit2)

8 x 1 sparse Matrix of class "dgCMatrix"
                  s0
(Intercept) -9.7671
age          0.0304
weight      -0.0747
height      -0.5682
neck        -0.4339
abdomen      0.8390
knee         0.6246
ankle       -0.1871

ridge_pred <- predict(ridge_fit2,
                      newx = model.matrix(bfat ~ . - 1, data = bfatTest))
lasso_pred <- predict(lasso_fit2,
```

```
                        newx = model.matrix(bfat ~ . - 1, data = bfatTest))

calcMSE <- function(y, yhat) mean((yhat - y)^2)

calcMSE(bfatTest$bfat, ridge_pred)

[1] 20.6

calcMSE(bfatTest$bfat, lasso_pred)

[1] 17.9

par(mfrow = c(1,2))
plot(bfatTest$bfat, ridge_pred); abline(0, 1, lty = 2)
plot(bfatTest$bfat, lasso_pred); abline(0, 1, lty = 2)
```



Lasso outperforms ridge regression in settings where relatively few predictors have substantial coefficients and the the remaining predictors have coefficients that are small or close to zero. Ridge regression performs better where the response is a function of many predictors with coefficients of roughly the same size.

There are very efficient algorithms for fitting both ridge regression and lasso models, where the entire coefficient paths can be computed with the same amount of work as least squares regression.