Cross-validation

```
n <- 60
p <- 100

ind <- sample(1:20, 6)

X <- matrix(rnorm(n*p), n, p)

beta0 <- rep(0, p)
beta0[ind] <- rnorm(length(ind), 5, 2)

y <- rnorm(n, X %*% beta0)
```

Experiment 1 We consider two model alternatives, M_1 : Fit the model with all the 100 predictors, and M_2 Fit the model with first 30 predictors.

Use `lm` to fit these models.

First fit the two models separately and compute the error sum of squares i.e. `sum(fit$residual**2)`. What do you see?

```
X30 <- X[, 1:30]

fit1 <- lm(y~X-1)
fit2 <- lm(y~X30-1)

mean(fit1$residuals**2)
mean(fit2$residuals**2)
```

Error sum of squares for the first model 0. Thus M_1 seems to be a better model than M_2. This does not seem right, as M_1 has more unnecessary variables than M_2. In general, we want a statistical model with fewer independent variables as possible. For example, let 'A' decides to have lunch depending on time of the day and whether feeling hungry or not. But 'B' incorporates so many other variables like what 'B' had in the morning, at what time, what time had dinner last night, what time had yesterday's lunch, so on.... Now you want to be like 'A', less complicated. Hence, based on our simulation setting, M_2 contains all the relevant predictors (all the relevant predictors are in 1:20) and also much smaller in dimension. Thus, it should be a better model than M_1.

Now we use hold out cross-validation to compare M_1 and M_2. Specifically, take the first 45 observations in the train set and the rest 15 in the test set.

```
train <- 1:45
test  <- 46:60

fit1 <- lm(y[train]~X[train, ]-1)
fit2 <- lm(y[train]~X30[train, ]-1)

coef <- fit1$coefficients
coef[which(is.na(coef))] <- 0

yhat1 <- X[test, ] %*% coef
yhat2 <- X30[test, ] %*% fit2$coefficients

mean((y[test] - yhat1)^2, na.rm=T)
mean((y[test] - yhat2)^2, na.rm=T)
```

Now prediction error for model 1 is much greater than model 2. Thus, based on hold-out CV, model 2 is better. This makes sense.

Experiment 2 We run K-fold cross-validation for LASSO problem (specifically 10-fold cross-validation). Like in the SAFE-rule, we need to find λ_∞. The definition of λ_∞ is that for this value of λ, the LASSO solution will have approximately all zeroes. Actually, for any large value of λ, the LASSO solution will be all zeroes. We need to find the smallest value of λ for which this will happen and set that to λ_∞ or `lambdamax`.

```
m <- t(X)%*%y
lambdamax <- max(abs(m))/length(y)
lambdamin <- 0.001*lambdamax #glmnet recommendation
lambdagrid <- seq(lambdamin, lambdamax, length.out = 100)

K = 10
ind <- matrix(0, K, n/K) #building the index matrix

for(k in 1:K){
  ind[k, ] <- 1:(n/K) + (n/K)*(k-1)
}


error <- matrix(0, K, 100)

for(i in 1:K){
  for(j in 1:100){
    indSel <- ind[i, ]
    Xtrain <- X[-indSel, ]
    Ytrain <- y[-indSel]

    Xtest <- X[indSel, ]
    Ytest <- y[indSel]

    lambda <- lambdagrid[j]
    fit <- glmnet::glmnet(Xtrain, Ytrain, alpha = 1, intercept = F, lambda = lambda)

    yhat <- predict(fit, Xtest)

    error[i, j] <- sum((Ytest-yhat)^2)
  }
}

lambdagrid[which.min(colSums(error))]

fit <- glmnet::cv.glmnet(X, y, alpha = 1, intercept = F)
fit$lambda.min
```

They would be close, but not same as the `cv.glment` builds those fold by assigning n/K datapoints randomly. But this randomization step will happen only once at the beginning of the algorithm.