Part I: Simulation

In this exercise, we will use simulation to illustrate the variability of statistics calculated from random samples. Suppose there is a **normal** population of size **10000**, with mean **100** and standard deviation **15**. Now we draw a sample from the population, of size **100** without replacement, we can calculate sample statistics such as mean and variance. If we further repeat the sampling process many times, say **200**, we will have 200 sets of similar sample statistics. Let's examine these sample statistics.

The necessary parameters are already set up as below.

```
pop.size <- 10000
pop.mean <- 100
pop.sd <- 15

num.of.samples <- 200
sample.size <- 100
```

Questions and Answers

1. Use random seed **1234** to conduct the simulation (i.e., simulate the population as specified, draw 200 samples, and calculate sample mean and variance for each sample, respectively), evaluate the mean and standard deviation of the sample statistics, and compare with their theoretical values. Draw histograms of the sample statistics. (1 Mark)

Answer:

```
set.seed(1234)
# simulate the population
pop <- rnorm(pop.size, pop.mean, pop.sd)

# create vector to store 200 sample means
manymeans <- c(1:200)
# create vector to store 200 sample variances
manyvariances <- c(1:200)

# storing the 200 individual sample means and variances in the vectors earlier created
for (i in 1:num.of.samples){
sample <- sample(pop, sample.size, replace = FALSE)
manymeans[i] <- mean(sample)
manyvariances[i] <- var(sample)
}

mean(manymeans)
```

```
## [1] 100.1435
```

```
sd(manymeans)
```

```
## [1] 1.480945
```

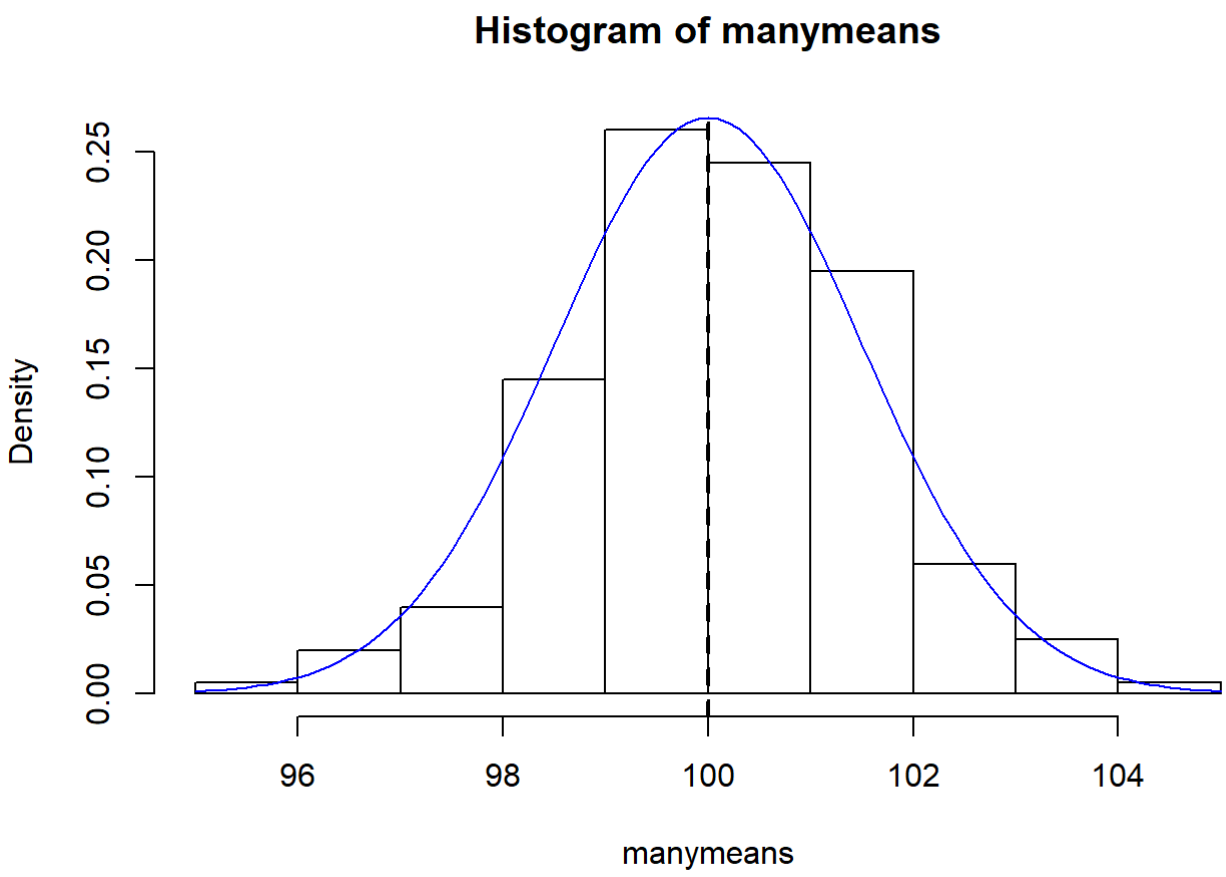
```
mean(manyvariances)
```

```
## [1] 220.4729
```

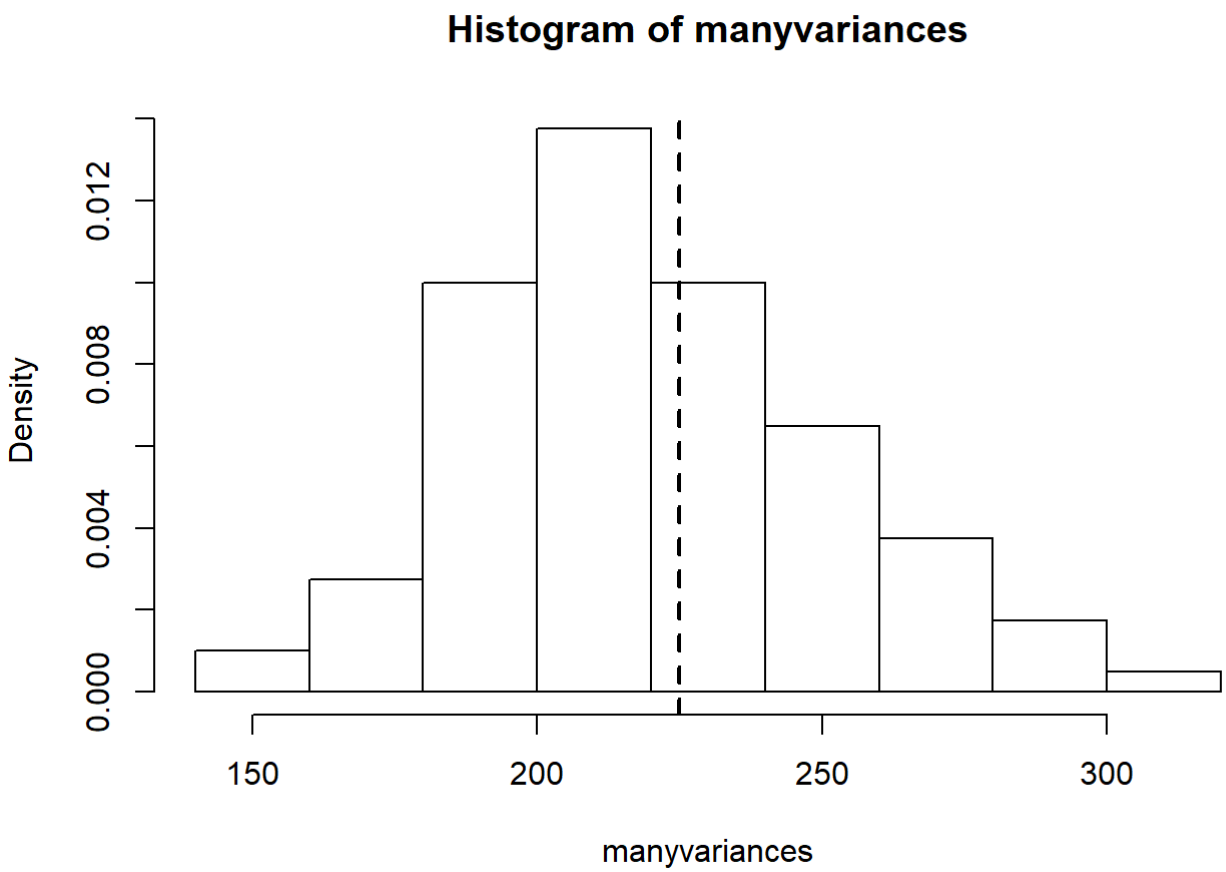
```
sd(manyvariances)
```

```
## [1] 31.31893
```

```
hist(manymeans, probability = TRUE)
abline(v=pop.mean, lty=2, lwd=2)
# overlay theoretical distribution for the sample means of sample size 100, can leave out from
 what to what
curve(dnorm(x, pop.mean, pop.sd/sqrt(sample.size)), from=95, to=105, col="blue",add=TRUE)
```




```
hist(manyvariances, probability = TRUE)
abline(v=pop.sd^2, lty=2, lwd=2)
```




```
# did not overlay theoretical distribution for the sample variances of sample size 100
```

The mean of the 200 sample means is 100.1434803 and the standard deviation of the 200 sample means is 1.4809454. The mean of the 200 sample variances is 220.4729356 and the standard deviation of the 200 sample variances is 31.3189326.

These values are close to the theoretical values. The theoretical mean of the sample means is 100 and the theoretical standard deviation of the sample means is 1.5. The theoretical mean of the sample variances is 225 and the theoretical standard deviation of the sample variances is 31.98.

Part II: K-Nearest Neighbor Algorithm

Introduction

In this assignment, we are going to experiment the K-Nearest Neighbor (KNN) algorithm on a higher-dimensional dataset and experience the deterioration of prediction performance as the dimensionality grows.

The experiment is built on top of the 3rd-order polynomial model discussed in class (knn_demo.R), i.e.,

$$y = \beta_0 + \beta_1 * x + \beta_2 * x^2 + \beta_3 * x^3 + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$$

y=β0+β1∗x+β2∗x2+β3∗x3+ϵ, ϵ∼N(0,σ2)

and we are going to introduce an extra 20-dimensional predictor zz, which does NOT actually play a role in generating yy. Yet, when in estimation, we do not know the fact and will use both xx and zz as predictors in the KNN algorithm.

Generation of the high-dimensional dataset

We first simulate the 3rd-order polynomial datasets as we did in knn_demo.R.

```
library("ggplot2")

## population parameters
beta0 <- 1
beta1 <- -2
beta2 <- 6
beta3 <- -1
sigma <- 2

set.seed(7890)

## training data
x <- seq(0, 4.95, 0.05)
f_x <- beta0 + beta1 * x + beta2 * x^2 + beta3 * x^3
epsilon <- rnorm(n=100, mean=0, sd=sigma)
y <- f_x + epsilon

## test data
x.test <- seq(0, 5, 0.1)
f_x.test <- beta0 + beta1 * x.test + beta2 * x.test^2 + beta3 * x.test^3
epsilon.test <- rnorm(n=length(x.test), mean=0, sd=sigma)
y.test <- f_x.test + epsilon.test
```

The resulted training and test dataset have 100 and 51 data points, respectively.

Next, we need to generate zz, the 20-dimensional predictors, of the same sizes. Each zz is a 20-dimensional multivariate normal random variable, with mean being $(0, 0, \dots, 0)$ (0,0,…,0) and identity covariance matrix (so that the 20 elements are independent standard normal random variables). The resulted zz is a 100*20 matrix, with each row being a data point with 20 dimensions.

```
library("mvtnorm")  # package for multivariate normal distribution, INSTALL IT BEFORE RUNNING
z <- rmvnorm(n=100, mean=rep(0, 20))  # covariance matrix is identity matrix by default, no nee
d to specify here
z.test <- rmvnorm(n=51, mean=rep(0, 20))
head(z)
```

```
##            [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
## [1,] -1.9361316 -0.34358128 -1.1638774  0.1605231  2.20486714  0.1999970
## [2,]  0.2073874  0.08351621 -0.2201877  0.7665609  0.02909645 -0.7931468
## [3,] -0.6541246  0.01767131 -1.3381875 -1.5689372 -0.34894118  0.4083368
## [4,]  0.5405513 -0.97623013 -0.9655379 -1.2686342 -0.39747994 -1.7813696
## [5,] -0.7049465 -0.86347174 -0.2666028  0.0396169  0.97658103 -0.2921821
## [6,] -0.6152281  0.34972310  0.3442159 -2.4733329  1.60772672  0.9010075
##             [,7]       [,8]       [,9]      [,10]      [,11]      [,12]
## [1,]  0.63703376 -0.3163284 -0.04411668  1.8363856  0.5395264 -1.0504447
## [2,]  0.32640547 -1.2411995 -3.21302032 -1.7544576  1.1230329  0.5253656
## [3,] -1.34247876 -0.9201846  0.41490504 -0.6399194 -1.4481277  0.9424778
## [4,]  1.26541099 -0.9375670  1.19049543  0.2012602  1.7722236  1.3229934
## [5,] -0.03845784 -1.0264802  0.14979145 -0.4021824 -0.9955113 -1.5960321
## [6,] -0.12378738 -0.7137295 -0.74265059  1.0453174 -0.4778248  0.5921299
##            [,13]      [,14]      [,15]       [,16]      [,17]       [,18]
## [1,] -0.09930164  1.7060724  0.2533610 -2.29932737  0.6075511 -0.05006046
## [2,] -0.96761386  0.8290950 -2.3699198 -1.08936417  1.3503342 -0.37279404
## [3,]  1.77713823 -0.8249576 -0.5913526 -0.84274966  0.2850005  0.34198427
## [4,]  0.03379135  0.4903843 -0.2531970 -0.23792899  0.2704636 -0.08672554
## [5,]  0.53734820  0.9362600  0.7000580 -0.05312255  0.8395479  0.65372268
## [6,]  0.95070597 -0.5238348  0.5621953 -0.67832178 -0.5084597 -0.76040599
##           [,19]      [,20]
## [1,] -0.1269237 -0.9816281
## [2,]  0.6210679 -0.5312891
## [3,]  0.4387673  0.3632285
## [4,]  0.7263930 -1.7324591
## [5,] -0.3007821 -0.4409981
## [6,] -0.9644197 -1.7722140
```

Later, we will use (x, z) (x,z) to predict yy. Let's first combine xx and zz into matrices, as required by function knn.reg().

```
train.x <- cbind(x, z)
test.x <- cbind(x.test, z.test)
head(train.x)
```

```
##         x
## [1,] 0.00 -1.9361316 -0.34358128 -1.1638774  0.1605231  2.20486714
## [2,] 0.05  0.2073874  0.08351621 -0.2201877  0.7665609  0.02909645
## [3,] 0.10 -0.6541246  0.01767131 -1.3381875 -1.5689372 -0.34894118
## [4,] 0.15  0.5405513 -0.97623013 -0.9655379 -1.2686342 -0.39747994
## [5,] 0.20 -0.7049465 -0.86347174 -0.2666028  0.0396169  0.97658103
## [6,] 0.25 -0.6152281  0.34972310  0.3442159 -2.4733329  1.60772672
##
## [1,]  0.1999970  0.63703376 -0.3163284 -0.04411668  1.8363856  0.5395264
## [2,] -0.7931468  0.32640547 -1.2411995 -3.21302032 -1.7544576  1.1230329
## [3,]  0.4083368 -1.34247876 -0.9201846  0.41490504 -0.6399194 -1.4481277
## [4,] -1.7813696  1.26541099 -0.9375670  1.19049543  0.2012602  1.7722236
## [5,] -0.2921821 -0.03845784 -1.0264802  0.14979145 -0.4021824 -0.9955113
## [6,]  0.9010075 -0.12378738 -0.7137295 -0.74265059  1.0453174 -0.4778248
##
## [1,] -1.0504447 -0.09930164  1.7060724  0.2533610 -2.29932737  0.6075511
## [2,]  0.5253656 -0.96761386  0.8290950 -2.3699198 -1.08936417  1.3503342
## [3,]  0.9424778  1.77713823 -0.8249576 -0.5913526 -0.84274966  0.2850005
## [4,]  1.3229934  0.03379135  0.4903843 -0.2531970 -0.23792899  0.2704636
## [5,] -1.5960321  0.53734820  0.9362600  0.7000580 -0.05312255  0.8395479
## [6,]  0.5921299  0.95070597 -0.5238348  0.5621953 -0.67832178 -0.5084597
##
## [1,] -0.05006046 -0.1269237 -0.9816281
## [2,] -0.37279404  0.6210679 -0.5312891
## [3,]  0.34198427  0.4387673  0.3632285
## [4,] -0.08672554  0.7263930 -1.7324591
## [5,]  0.65372268 -0.3007821 -0.4409981
## [6,] -0.76040599 -0.9644197 -1.7722140
```

Questions

1. For a fixed $k = 15$ k=15, fit a KNN model to predict yy with (x, z) (x,z), and measure the training and test MSE. (1 Mark)

Answer:

```
library("FNN")
model15.train <- knn.reg(train=train.x, test=train.x, y=y, k=15)
model15.test <- knn.reg(train=train.x, test=test.x, y=y, k=15)
# Training MSE
mean((y - model15.train$pred)^2)
```

```
## [1] 25.27996
```

```
# Test MSE
mean((y.test - model15.test$pred)^2)
```

```
## [1] 30.90956
```

The training MSE is 25.2799557 and the test MSE is 30.9095621.

2. With the same data, plot the training and test MSE of the KNN model against kk, and find the optimal kk and the corresponding test MSE. (1 Mark)

Answer:

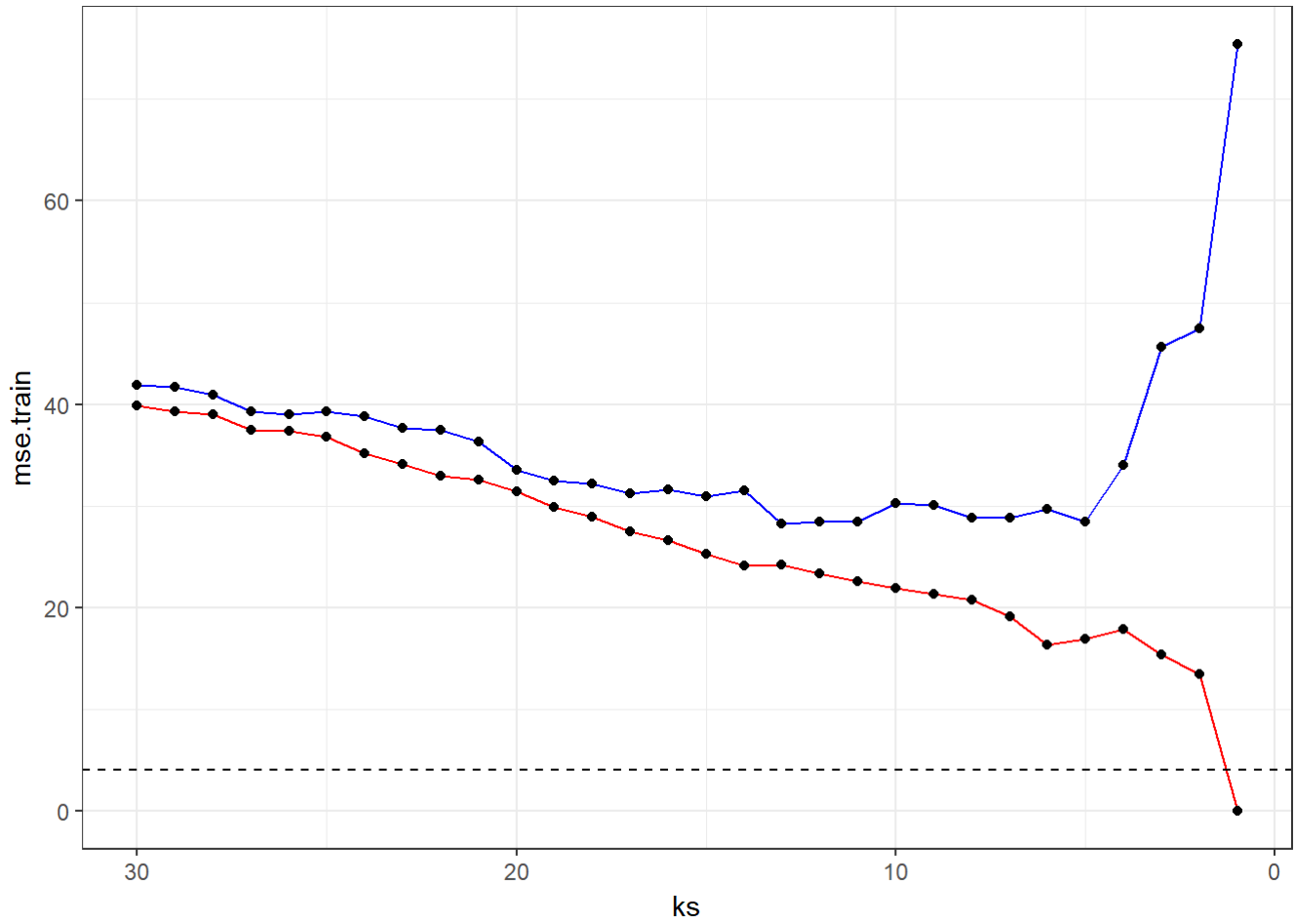
```
# k's that will be evaluated
ks <- 1:30
# construct empty vectors for keeping the MSE for each k
mse.train <- numeric(length=length(ks))
mse.test  <- numeric(length=length(ks))

# loop over all the k and evaluate MSE in each of them
for (i in seq(along=ks)) {
    model.train <- knn.reg(train.x, train.x, y, k=ks[i])
    model.test  <- knn.reg(train.x, test.x, y, k=ks[i])
    mse.train[i] <- mean((y - model.train$pred)^2)
    mse.test[i] <- mean((y.test - model.test$pred)^2)
}
mse.train
```

```
##  [1]  0.00000 13.44851 15.40687 17.87187 16.85736 16.35781 19.15542
##  [8] 20.73851 21.32666 21.94045 22.59267 23.30333 24.24809 24.07687
## [15] 25.27996 26.65289 27.45065 28.95038 29.84570 31.44479 32.56622
## [22] 32.97976 34.12066 35.18099 36.75183 37.39806 37.41895 39.04934
## [29] 39.29471 39.84186
```

```
mse.test
```

```
##  [1] 75.38950 47.41891 45.61282 33.96436 28.45206 29.69826 28.85134
##  [8] 28.82283 30.05319 30.22976 28.41302 28.47921 28.26073 31.52024
## [15] 30.90956 31.58330 31.19329 32.19299 32.51377 33.56734 36.35270
## [22] 37.45248 37.70311 38.82965 39.24606 38.95774 39.29131 40.89234
## [29] 41.69166 41.84667
```

```
# plot MSE on Training and Test
ggplot() + geom_line(aes(x=ks, y=mse.train), color="red") + geom_point(aes(x=ks, y=mse.train))
+ geom_line(aes(x=ks, y=mse.test), color="blue") + geom_point(aes(x=ks, y=mse.test)) + scale_x_
reverse(lim=c(30, 1)) + geom_hline(yintercept=sigma^2, linetype=2) + theme_bw()
```




```
# optimal k
k.opt <- which.min(mse.test)
k.opt
```

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

```
# optimal MSE
mse.opt <- min(mse.test)
mse.opt
```

```
## [1] 28.26073
```

The optimal k is 13 and the corresponding test MSE is 28.2607279.

3. Based on the analysis above, compare the above model with (x, z) (x,z) being the predictors and the previous model with x x only (as in knn_demo.R). Briefly explain why. (1 Mark)

Answer: The test MSE is 28.2607279, which is significantly higher than in the previous model without zz, where the test MSE was around 44. This is purely driven by the inclusion of predictor zz. The reason is two-fold: (1) zz is actually not affecting yy, so is irrelevant in prediction. KNN does not have a variable selection process built in and cannot ignore those. (2) Inclusion of irrelevant variables increases the dimensionality. KNN performance deteriorates quickly when the dimensionality grows and the data points become sparse and far away from each other.

4. We have seen that the test MSE is significantly worse than what we had without using predictor zz (in knn_demo.R). To better understand the impact of including irrelevant predictors in the KNN algorithm, let's try to include the 20 dimensions of

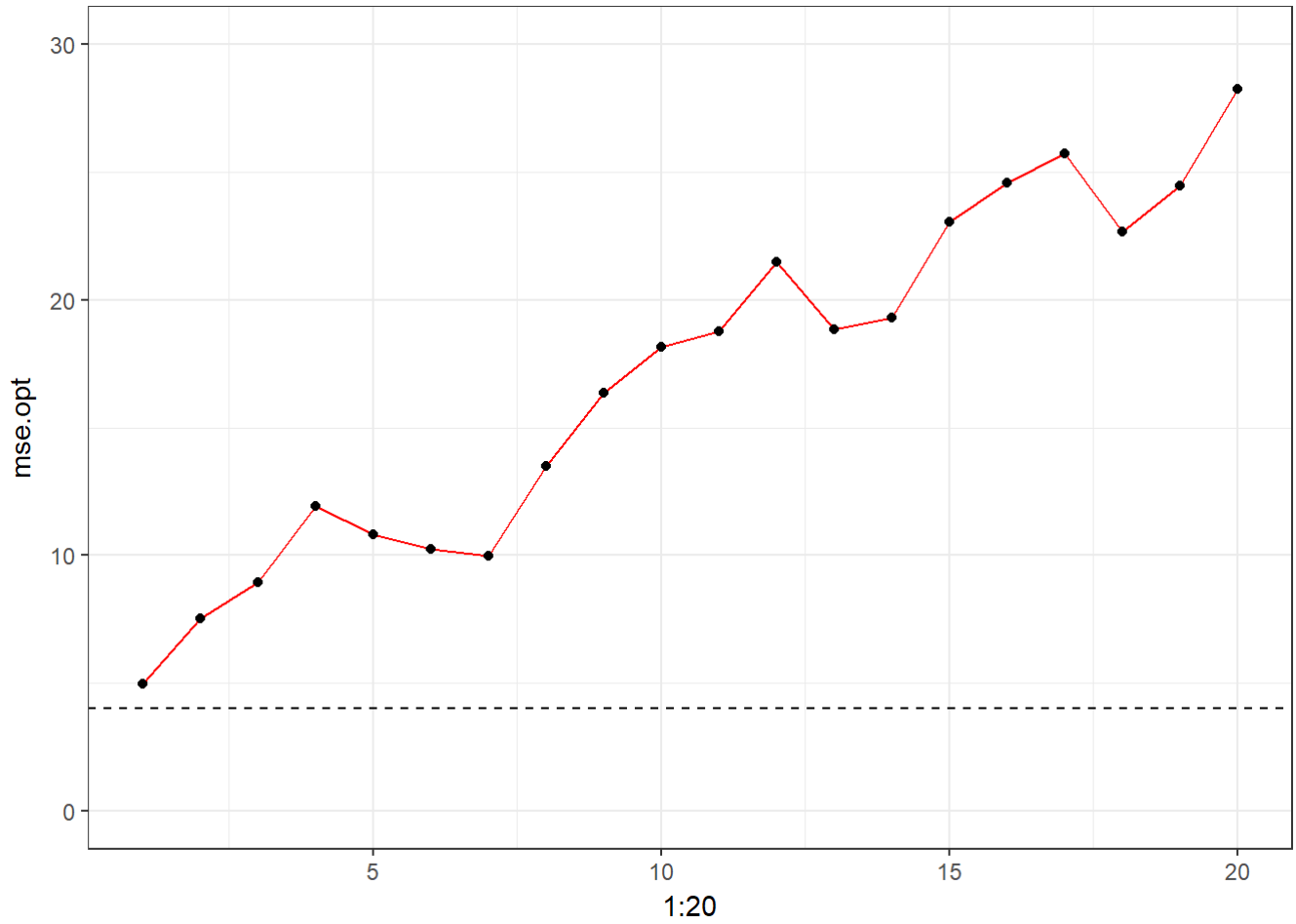
zzone by one. So in each round jj, we construct the predictors by combining xx and the first jj columns of zz, then repeat the analysis in Question 2 and find the optimal kk and test MSE. At the end, plot the optimal MSE agaist jj, and interpret the result. (1 Mark)

Answer:

```
# construct empty vectors for keeping the optimal k and corresponding Test MSE for each round o
f j, there are 20 rounds
k.opt <- numeric(length=20)
mse.opt <- numeric(length=20)

for (j in 1:20){
  train.x <- cbind(x, z[,1:j])
  test.x <- cbind(x.test, z.test[,1:j])
  # k's that will be evaluated
  ks <- 1:30
  # construct empty vectors for keeping the MSE for each k
  mse.train <- numeric(length=length(ks))
  mse.test  <- numeric(length=length(ks))

  # loop over all the k and evaluate MSE in each of them
  for (i in 1:30) {
    model.train <- knn.reg(train.x, train.x, y, k=ks[i])
    model.test  <- knn.reg(train.x, test.x, y, k=ks[i])
    mse.train[i] <- mean((y-model.train$pred)^2)
    mse.test[i] <- mean((y.test - model.test$pred)^2)
  }
# for each round of 20 combinations, find the best k out of the 30 k values
# optimal k for each round of j
k.opt[j] <- which.min(mse.test)
# optimal MSE Train for each round of j
mse.opt[j] <- min(mse.test)
}
```

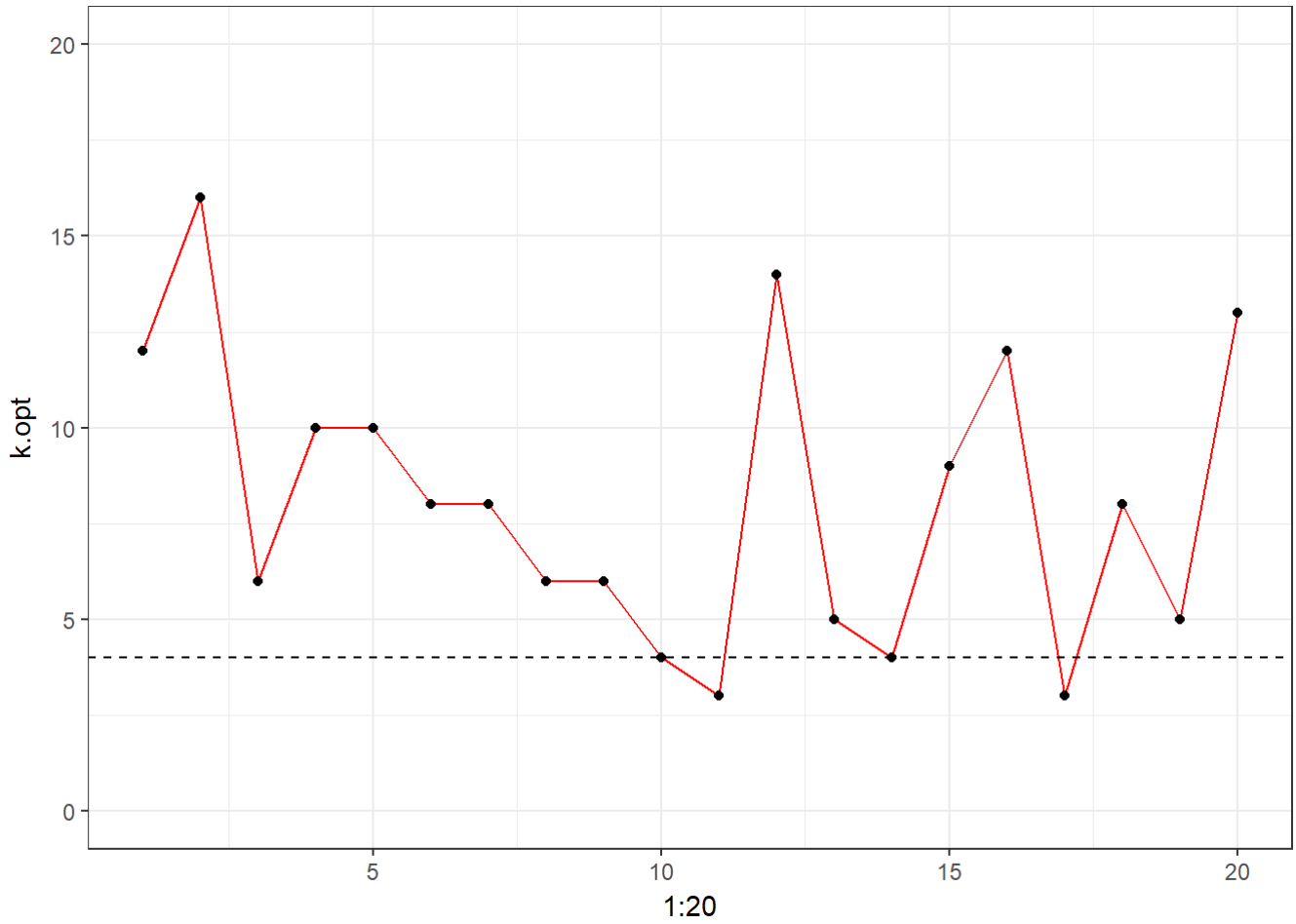
The optimal k is 13 and the corresponding test MSE is 28.2607279.

```
# plot optimal MSE for each of the 20 rounds against j
ggplot() + geom_line(aes(x=1:20, y=mse.opt), color="red") + geom_point(aes(x=1:20, y=mse.opt))
+ scale_y_continuous(lim=c(0, 30)) + geom_hline(yintercept=sigma^2, linetype=2) + theme_bw()
```



Including irrelevant predictors will not improve the predictive power of the model. Including irrelevant predictors increases the dimensionality and undermines the performance of the model. The addition of a higher dimensional predictor to x results in the points becoming further from one another, which increases the distance of the k-nearest neighbours to the original x point.

```
# ADDITIONAL CODES
# plot optimal k for each of the 20 rounds against j
ggplot() + geom_line(aes(x=1:20, y=k.opt), color="red") + geom_point(aes(x=1:20, y=k.opt)) + sc
ale_y_continuous(lim=c(0, 20)) + geom_hline(yintercept=sigma^2, linetype=2) + theme_bw()
```



Session Info

```
print(sessionInfo(), locale=FALSE)
```

```
## R version 3.3.3 (2017-03-06)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 18362)
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] FNN_1.1       mvtnorm_1.0-5 ggplot2_2.2.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.6      digest_0.6.9     rprojroot_1.2    plyr_1.8.4
##  [5] grid_3.3.3       gtable_0.2.0     backports_1.0.5  formatR_1.4
##  [9] magrittr_1.5     evaluate_0.9     scales_0.4.1     stringi_1.1.1
## [13] lazyeval_0.2.0   rmarkdown_1.3    labeling_0.3     tools_3.3.3
## [17] stringr_1.0.0    munsell_0.4.3    colorspace_1.2-6 htmltools_0.3.5
## [21] knitr_1.14       tibble_1.3.0
```