DS 6030 HW02 Resampling

Ben Wilson

'09/14/2022

Contents

Load Packages

```
# Set up R
knitr::opts_chunk$set(echo = TRUE)
data.dir = 'https://mdporter.github.io/DS6030/data/' # data directory
library(R6030)      # functions for DS-6030
library(tidyverse)  # functions for data manipulation
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.8     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.1
## v readr   2.1.2     v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(broom)      # for extraction of model components
library(splines)    # for working with B-splines
library(FNN)
library(dplyr)
```

Problem 1: Bootstrapping

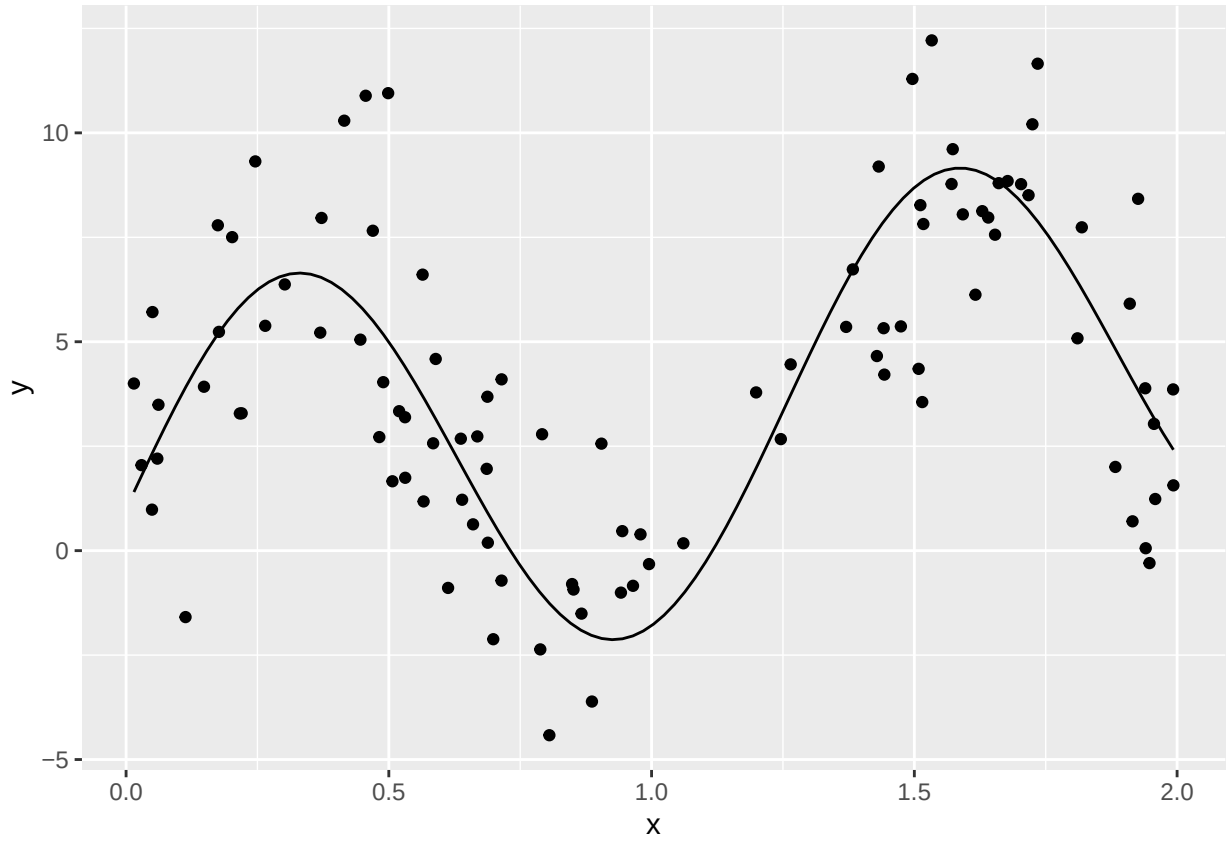
1A. Create a set of functions to generate data from the following distributions:

```
# Generate variables & functions
sim_x <- function(n) runif(n, min = 0, max = 2)
f <- function(x) 1 + 2*x + 5*sin(5*x)
sim_y <- function(x, sd){
  n = length(x)
  f(x) + rnorm(n, sd=sd)
}
```

1B. Simulate n=100 realizations from these distributions. Produce a scatterplot and draw the true regression line. Use set.seed(211) prior to generating the data.

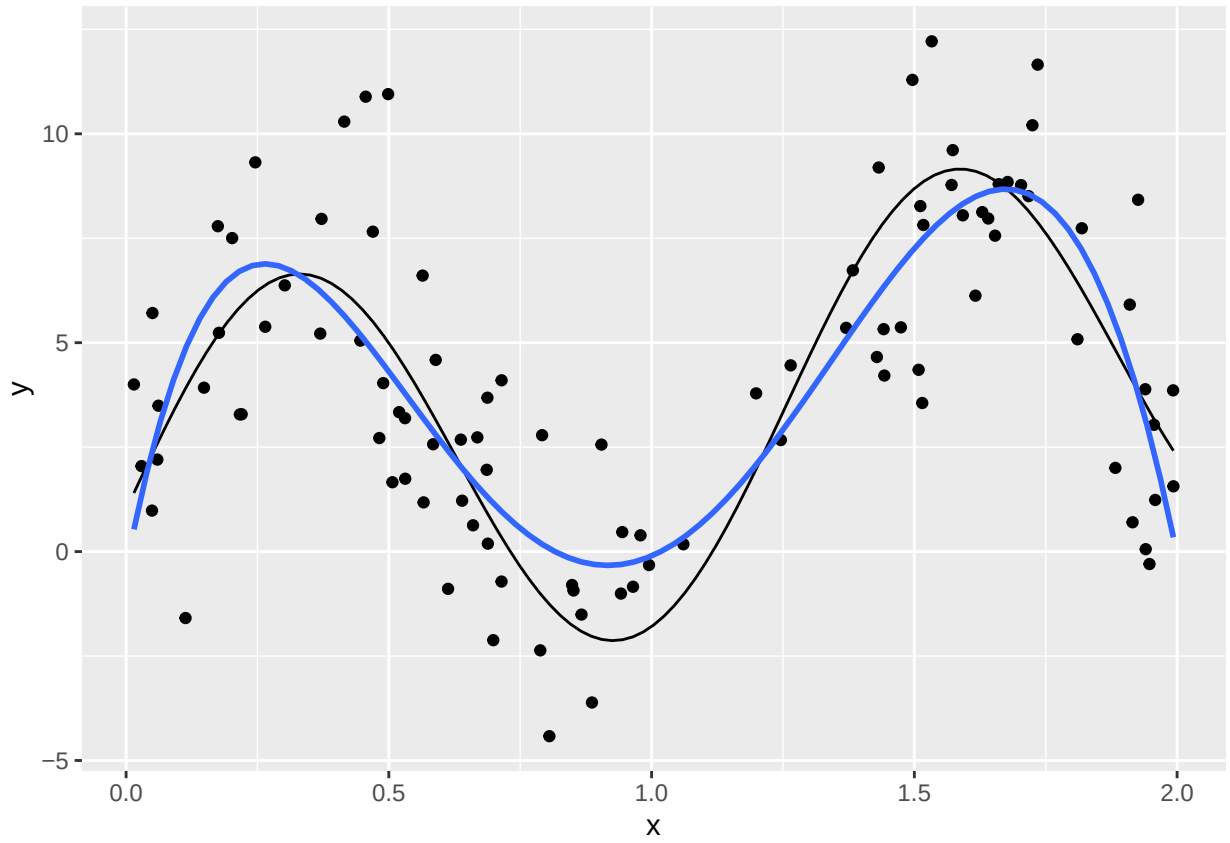
```
set.seed(211)
n = 100
sd = 2.5

# Generate Data
x = sim_x(n)
y = sim_y(x, sd=sd)
df = tibble(x, y)
```

```r
# Generate scatter plot
scatter_df = ggplot(df, aes(x,y)) +
  geom_point()+
  geom_function(fun=f)

# View plot
scatter_df
```



1C. c. Fit a 5th degree polynomial. Produce a scatterplot and draw the estimated regression curve.

```r
# Generate 5th degree poly line
m5 = lm(y~poly(x, degree=5), data=df)

# Generate updated scatter plot
scatter_df +
  geom_smooth(method="lm", formula="y~poly(x,5)", se=FALSE) +
  scale_color_discrete(name="model")
```



1D. Make 200 bootstrap samples. For each bootstrap sample, fit a 5th degree polynomial and make predictions at eval_pts = seq(0, 2, length=100)

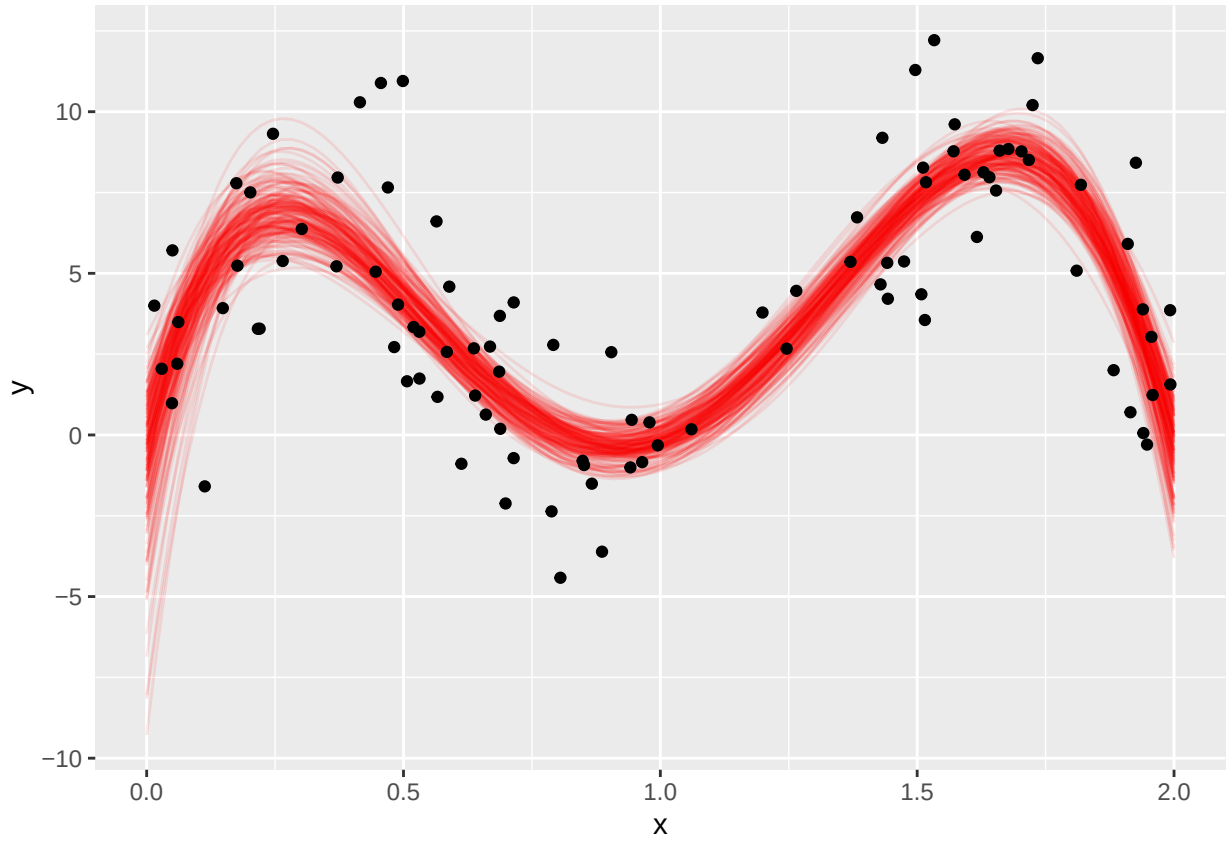
```
M = 200 # number of bootstrap samples
kts.bdry = c(-0.2,2.2)
set.seed(212) # set random seed
eval_pts = tibble(x=seq(0, 2, length=100)) #set evaluation points
yhat = matrix(NA, nrow(eval_pts), M)

# generate bootstrap
for(m in 1:M){

  #sample from empirical dist (sample with replacement)
  ind = sample(n, replace=TRUE)

  #fit bspline model
  m_boot = lm(y~poly(x, df=5),
              data = df[ind,]) #fit bootstrap data

  #predict from bootstrap model
  yhat[,m] = predict(m_boot, eval_pts)
}

#convert to tibble and plot
```

```
data_fit = as_tibble(yhat) %>% #convert matrix to tibble
  bind_cols(eval_pts) %>% #add to eval points
  pivot_longer(-x, names_to = "simulation", values_to = "y") #conver to long form
```

```
## Warning: The ‘x‘ argument of ‘as_tibble.matrix()‘ must have unique column names if ‘.name_repair‘ is
## Using compatibility ‘.name_repair‘.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was generated.
```

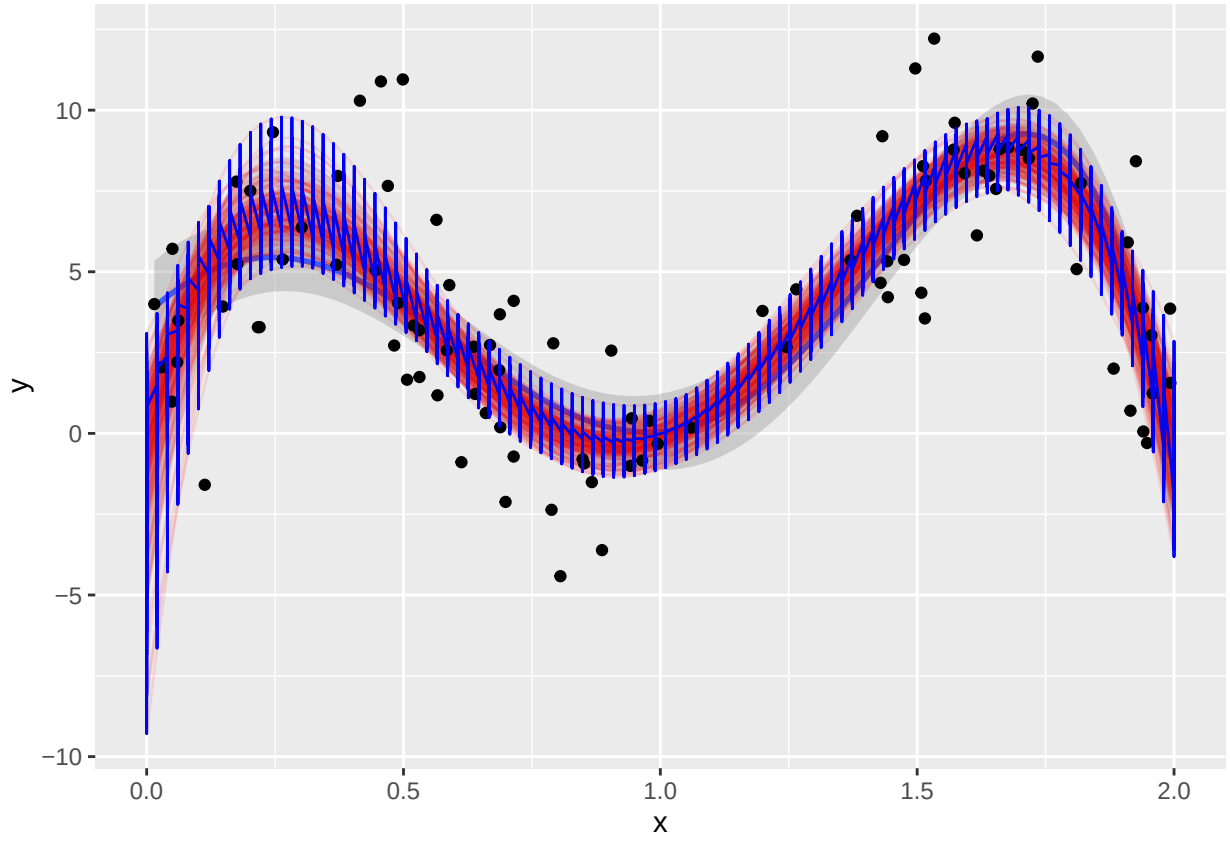
```
#draw ggplot
ggplot(df, aes(x,y)) +
  geom_smooth(method = 'lm',
              formula = 'y~poly(x, degree = 5, Boundary.knots = kts.bdry)-1') +
  geom_line(data = data_fit, color = "red", alpha = .10, aes(group = simulation)) + geom_point()
```

```
## Warning: Computation failed in ‘stat_smooth()‘:
## arguments must have the same length
```



e. Calculate the pointwise 95% confidence intervals from the bootstrap samples. That is, for each x eval_pts, calculate the upper and lower limits such that only 5% of the curves fall outside the interval at x. Remake the plot from part c, but add the upper and lower boundaries from the 95% confidence intervals.

```
conf_int = data_fit %>%
  group_by(x) %>%
  summarize(lower = quantile(y,0.025), upper = quantile(y,0.975))

#draw ggplot
ggplot(df, aes(x,y)) +
  geom_smooth(method = 'lm',
              formula = 'y~bs(x, degree = 5, Boundary.knots = kts.bdry)-1') +
  geom_line(data = data_fit, color = "red", alpha = .10, aes(group = simulation)) + geom_point() +
  geom_line(data = data_fit,
            aes(x = x, y = y), colour = "blue")+
  geom_ribbon(data = conf_int, aes(x = x, ymin = lower, ymax = upper), alpha = 0.1)
```



Problem 2: V-Fold cross-validation with k nearest neighbors

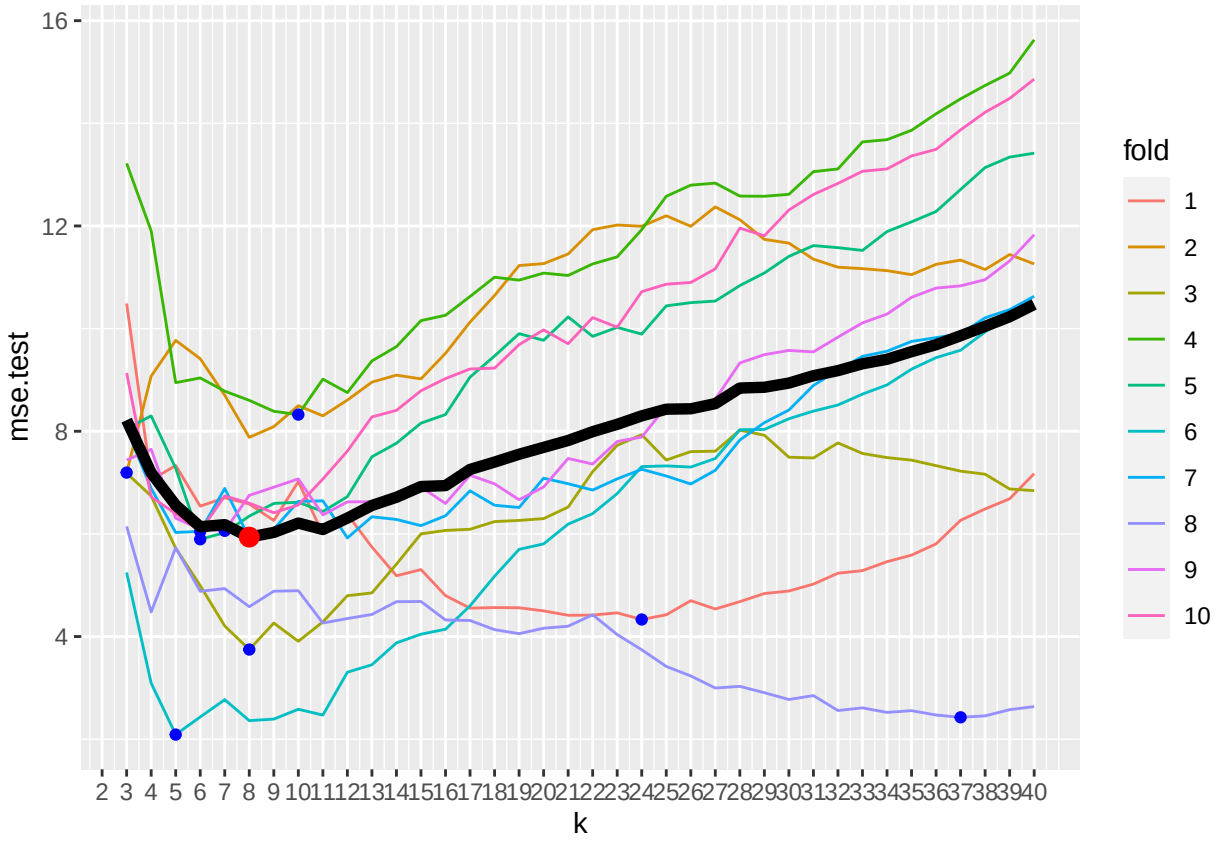
Run 10-fold cross-validation on the data generated in part 1b to select the optimal k in a k-nearest neighbor (kNN) model. Then evaluate how well cross-validation performed by evaluating the performance on a large test set. The steps below will guide you.

2A. Use 10-fold cross-validation to find the value of k (i.e., neighborhood size) that provides the smallest cross-validated MSE using a kNN model.

Set evaluation function

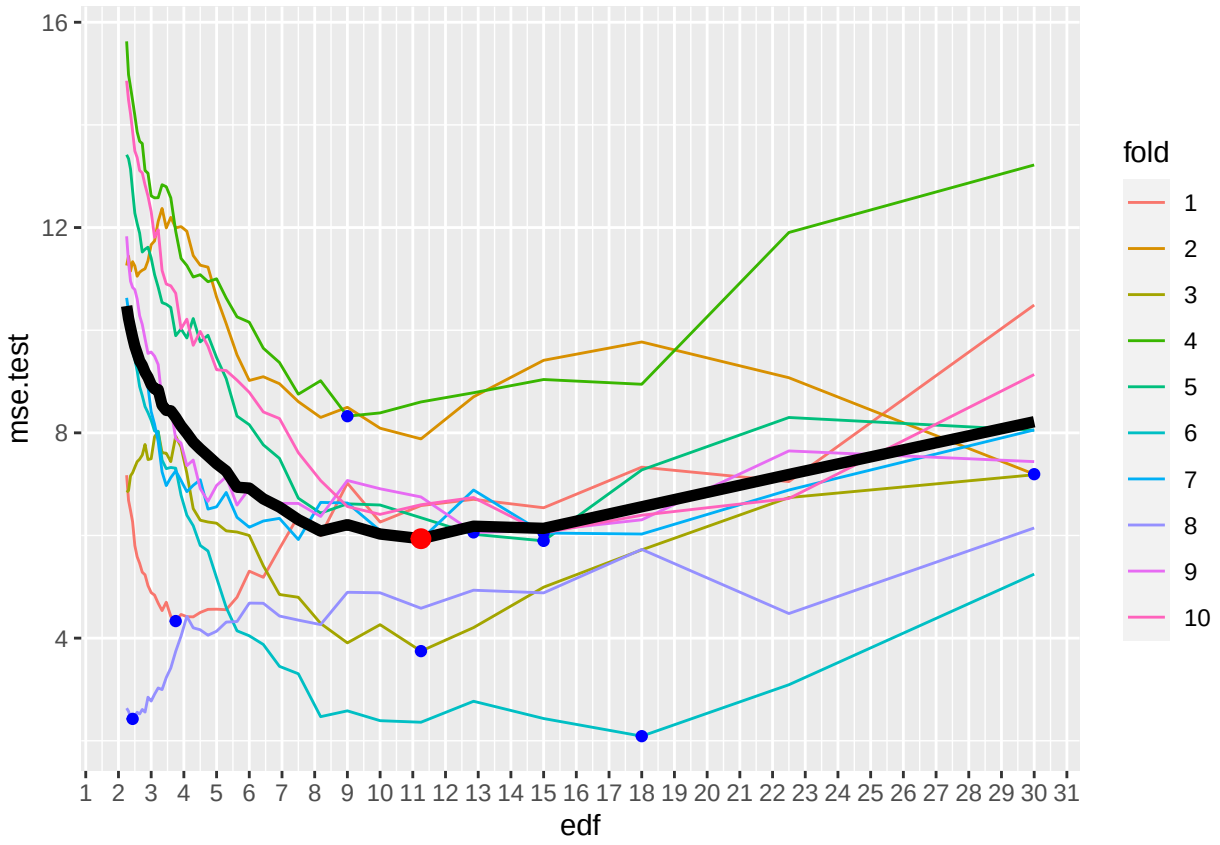
```
knn_eval <- function(k, data_train, data_test){

  # fit model and eval on test data
  knn.test = knn.reg(data_train['x', drop=FALSE],
                     y = data_train$y,
                     test=data_test['x', drop=FALSE],
                     k=k)
  r.test = data_test$y-knn.test$pred # residuals on test data
  mse.test = mean(r.test^2)          # test MSE
  # results
  edf = nrow(data_train)/k         # effective dof (edof)
  tibble(k=k, edf=edf, mse.test)
}

n.folds = 10 # number of folds for cross-validation
set.seed(221) # set seed for reproducibility
fold = sample(rep(1:n.folds, length=n)) # vector of fold labels
K = seq(3,40,by=1)


results = vector("list", n.folds)

#- Iterate over folds
for(j in 1:n.folds){
  #-- Set training/val data
  val = which(fold == j) # indices of holdout/validation data
  train = which(fold != j) # indices of fitting/training data
  n.val = length(val) # number of observations in validation

  #- fit and evaluate models
  results[[j]] = map_df(K, knn_eval,
                        data_train = slice(df, train),
                        data_test = slice(df, val)) %>%
    mutate(fold = j, n.val) # add fold number and number in validation
}

RESULTS = bind_rows(results)

RESULTS %>% mutate(fold = factor(fold)) %>%
  ggplot(aes(k, mse.test)) +
  geom_line(aes(color=fold)) +
  geom_point(data=. %>% group_by(fold) %>% slice_min(mse.test, n=1), color="blue") +
```

```
  geom_line(data = . %>% group_by(k) %>% summarize(mse.test = mean(mse.test)), size=2) +
  geom_point(data = . %>% group_by(k) %>% summarize(mse.test = mean(mse.test)) %>%
               slice_min(mse.test, n=1), size=3, color="red") +
  scale_x_continuous(breaks = seq(0, 40, by=1))
```



2B. The k (number of neighbors) in a kNN model determines the effective degrees of freedom edf. What is the optimal edf? Be sure to use the correct sample size when making this calculation. Produce a plot similar to that from part a, but use edf (effective degrees of freedom) on the x-axis.

```
RESULTS %>% mutate(fold = factor(fold)) %>%
  ggplot(aes(edf, mse.test)) +
  geom_line(aes(color=fold)) +
  geom_point(data=. %>% group_by(fold) %>% slice_min(mse.test, n=1), color="blue") +
  geom_line(data = . %>% group_by(edf) %>% summarize(mse.test = mean(mse.test)), size=2) +
  geom_point(data = . %>% group_by(edf) %>% summarize(mse.test = mean(mse.test)) %>%
               slice_min(mse.test, n=1), size=3, color="red") +
  scale_x_continuous(breaks = seq(0, 40, by=1))
```



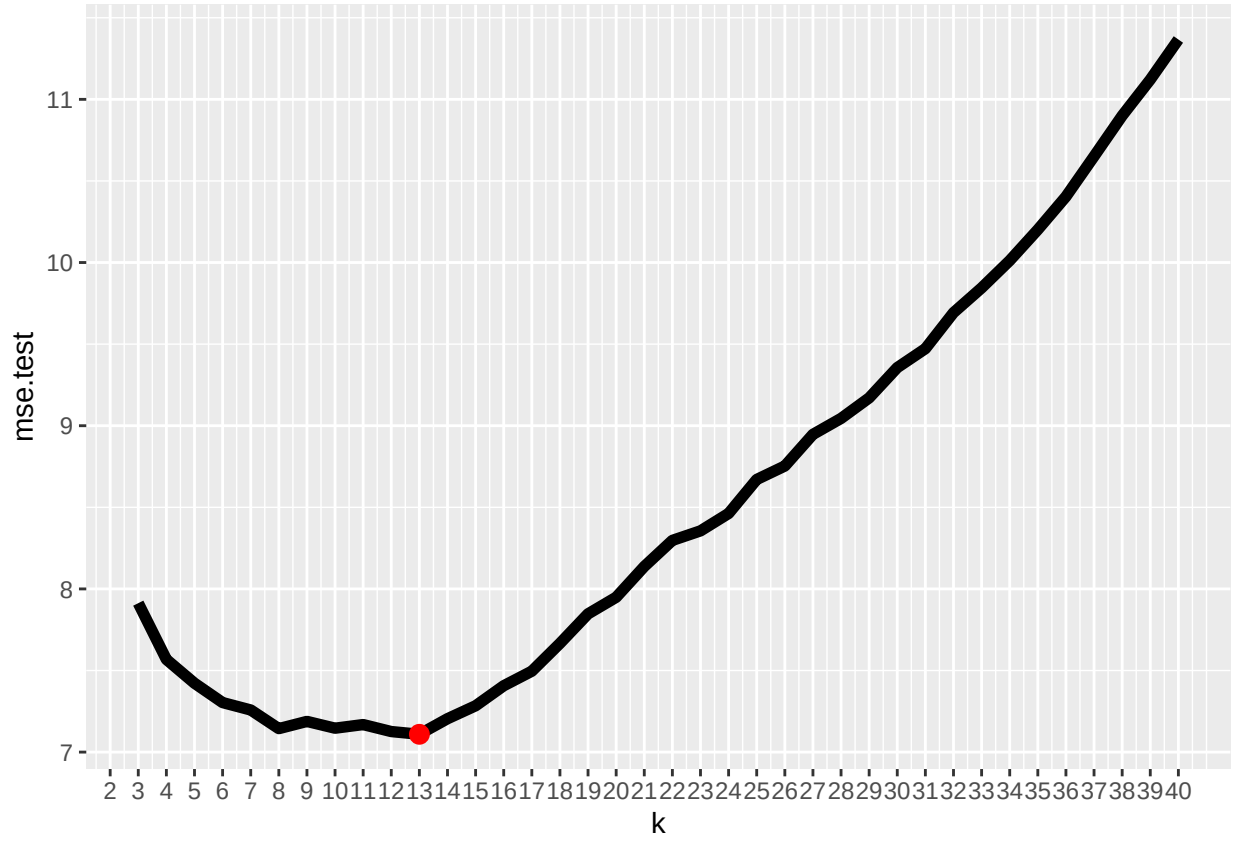
2C. After running cross-validation, a final model fit from all of the training data needs to be produced to make predictions. What value of k would you choose? Why?

K = 3 is the choice as the lowest MSE is 8 and when you choose the lowest df using the 1 sided deviation rule you can reduce by 1 standard error without losing prediction power (materially) yet simplify the model.

2D. Now we will see how well cross-validation performed. Simulate a test data set of 50000 observations from the same distributions. Use set.seed(223) prior to generating the test data.

```
set.seed(223)
K = seq(3,40,by=1)

n2=50000
sd=2.5

# Generate Data
x_test = sim_x(n2)
y_test = sim_y(x_test, sd=sd)
df_test = tibble(x = x_test,y = y_test)
```

```
#- fit and evaluate models
results2 = map_df(K, knn_eval,df, df_test)


results2 %>%
  ggplot(aes(k, mse.test)) +
  geom_line(data = . %>% group_by(k) %>% summarize(mse.test = mean(mse.test)), size=2) +
  geom_point(data = . %>% group_by(k) %>% summarize(mse.test = mean(mse.test)) %>%
               slice_min(mse.test, n=1), size=3, color="red") +
  scale_x_continuous(breaks = seq(0, 40, by=1))
```

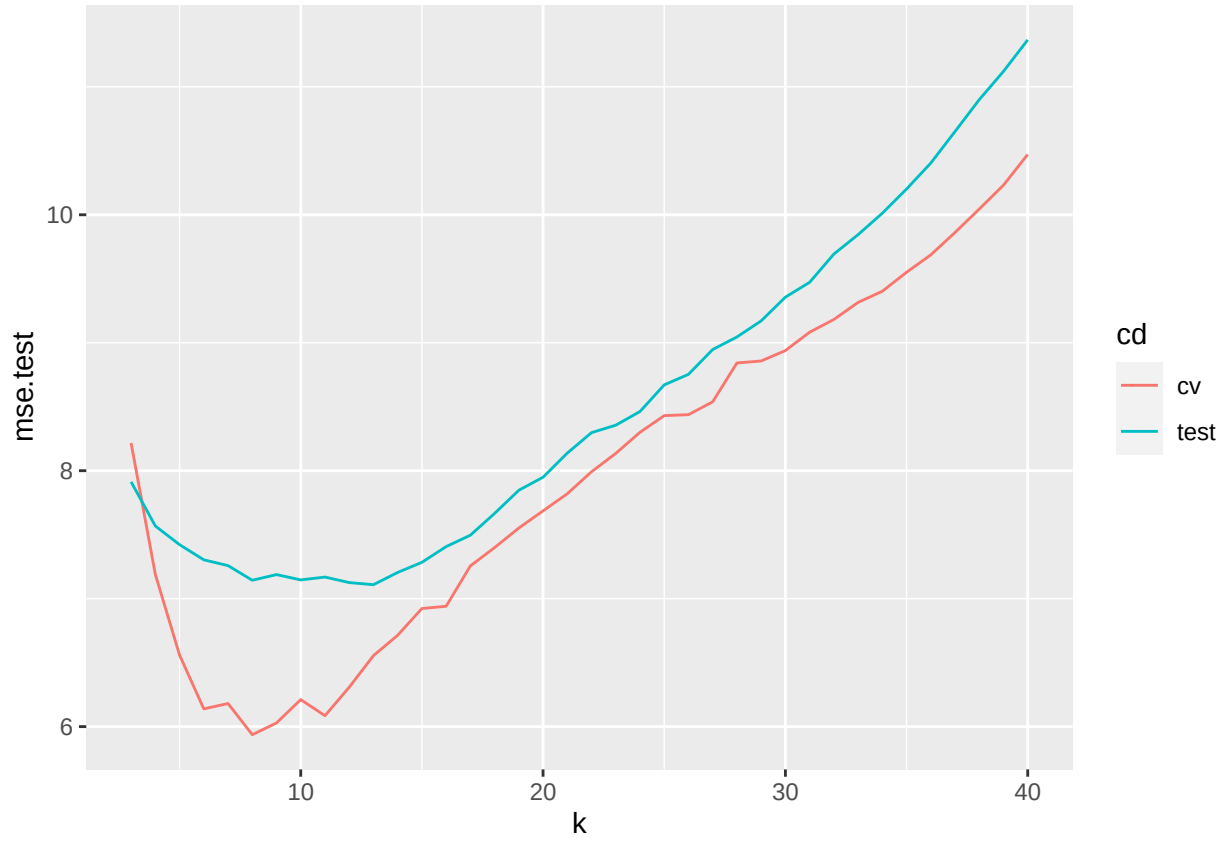



```
results2 %>% filter(k==13)
```

```
## # A tibble: 1 x 3
##       k   edf mse.test
##   <dbl> <dbl>    <dbl>
## 1    13  7.69     7.11
```

2E. Plot both the cross-validation estimated and (true) error calculated from the test data on the same plot. See Figure 5.6 in ISL (pg 182) as a guide.

```
results_final = RESULTS %>%
    select(-'fold',-'n.val') %>%
  group_by(k,edf) %>%
  summarise_at(vars(mse.test), list(mse.test = mean))

results_final$cd <- c('cv')

cross_val <- rbind(results_final,results2 %>% mutate(cd = 'test'))

ggplot(data=cross_val, aes(x = k, y=, mse.test, color = cd))+
  geom_line()
```



2F. Based on the plots from e, does it appear that cross-validation worked as intended? How sensitive is the choice of k on the resulting test MSE?

The cross-validation model appears to have greater variances in swings of the charts in comparison to the test data set making it likely more sensitive. Cross-validation did successfully minimize MSE in the test data with a k=8 value.