Final Exam

Stats 506, Fall 2018

Monday December 17, 1:30-3:30 PM

Instructions

This is the final exam for Stats 506 in Fall 2018. The exam consists of two parts.

Part one has four questions worth 25 points each for a total of 100 points. You should answer all four questions from part one. Answer questions 1-3 from part one directly on the page; answer question four on a separate sheet of paper with your name clearly written at the top.

There are five questions in part two from which you should choose exactly two questions to answer. Each of these is worth 50 points. Answer each question you select from part 2 on a separate piece of paper. Please print your name and unique id on each page submitted. When finished, please staple the pages with your answers to part 2 in order with these exam questions on top.

You have approximately two hours to complete this exam. I will provide notifications when there are 30, 10, and 5 minutes remaining.

You are allowed one, two-sided 8.5 by 11 inch page of notes. Your work space should be clear of all other materials. Please turn off and put away your cell phone, computer, and other devices. You are not permitted to use these devices during the exam.

Good luck!

Scores:

Part one

1.
2.
3.
4.

Part two

5.
6.

7.
8.
9.

Part One

Answer all four questions in part one. Each question is worth 25 points.

1. Regular Expressions

a. [16 points] The table below contains regular expressions as row names and strings to be searched as column headers. Place an “X” in the cells for which a given regular expression matches a given string. Leave cells that are not matches blank.

	string1	**.string_2**	**.3string**	**4^th_string**
ring$			X	X
.+st		X	X	X
^([^.0-9]?s)	X			
^([A-Za-z]|[.][._A-Za-z])[._A-Za-z0-9]*$	X	X		

b. [6 points] Now, consider a similar table that indicates which strings to match using and “X”. Provide a regular expressions in each column that matches the strings indicated but does not match the others.

[your regexp below]	**script.R**	**.RData**	**.my_prefs**	**script.R~**
i.		X	X	
ii.	X			X
iii.				X

Solution: There are many variations that are acceptable. Here are possible answers for each:

```
i. ^[.]

ii. .+\.R  or ^[^.]

iii. ~$
```

c. [3 pts] Xinghui has a number of R scripts in a project folder `some_project` and would like to find all lines where he uses a function from the `stringr` library prefixed with `str_`. He knows that the command line tool `grep` can be used for this and tried: `grep *.R -e'str_'` but that returned desired matches such as `z = str_c(x, y)` but also undesired matches such as `mystr_0 = paste(x, y)` and `my_new_str_1 = paste(x, y)`.

Provide a new regular expression to complete the command below so that it achieves Xinghui’s goal.

```
grep *.R -e '[your regex here]'
```

Solution: The key here is to require the pattern `str_` to be preceded by a symobl not allowed in the name of an R object.

`grep *.R -e '[^\w_.]str_'` or `grep *.R -e '[^._[:alnum:]]str_'`.

2. Split, apply, combine

Consider the two SQL queries below which operate on the table “storms” containing historical information about named hurricanes (category 1-5), tropical storms (category 0), and tropical depressions (category -1) between 1975 and 2015. Each row of “storms” identifies, among other things, the name (“name”) of the storm, and its category (“category”) for a specific 6 hour period during the storms duration. Thus there are multiple and a varied number of rows per storm. The name of a storm does not change, but its category is allowed to vary from row to row.
You should assume that “category” is numeric and that “name” is of type character.

Each of the queries produces results similar to either the dplyr code in part b or the data.table code in part c.

Query 1:

```
SELECT category, N, 100*N/sum(N) as pct
FROM ( SELECT category, COUNT(name) as N
       FROM ( SELECT name, max(category)
              FROM storms
              WHERE category gt 0
              GROUP BY name
            )
       GROUP BY category
     )
```

Query 2:

```
SELECT category, N, 100*N/sum(N) as pct
FROM ( SELECT s.category as category, COUNT(s.name) as N
       FROM storms s
       LEFT JOIN ( SELECT name, max(category) as H
                   FROM storms
                   WHERE category gt 0
                   GROUP BY name
                 ) h
       ON s.name = h.name
       GROUP BY category
       HAVING h.H IS NOT NULL
     )
```

a. [5 points] Briefly describe the tables created by each query, focusing on how their rows are different and what the percentages “pct” represent.

Solution to (a):

Query 1 classifies storms according to the highest category reached, counts the number of such storms by this maximum category and gives what percent each category represents among all *hurricanes*, i.e. storms with highest category 1 or larger. In this table “N” counts storms and there are 5 rows corresponding to categories 1-5.

Query 2 also limits attention to storms classified as hurricanes (category greater than 0) at least once, but then counts the total number of rows corresponding to each category among all such storms. Its rows correspond to *all* categories -1 to 5 and the percentages are for what proportion of time storms that eventually become hurricanes spend in each category.

b. [10 points] The dplyr code below produces the same results as one of the two SQL queries above. First, identify which one and indicate your answer by circling the appropriate choice. Then, write dplyr code to produce the same result as the *other* SQL query.

Here is the dplyr code:

```
storms %>%
  group_by(name) %>%
  summarize( category = max(category) ) %>%
  filter( category > 0) %>%
  group_by(category) %>%
  summarize( N = n() ) %>%
  mutate( pct = 100*N/sum(N) )
```

Which SQL query does this dplyr code correspond to? Circle one:

Solution: Query 1

The dplyr syntax above maches *query 1* counting the number of storms by the highest category reached and giving what percent each category represents among all *hurricanes* (i.e. storms with highest category 1 or larger.)

Write dplyr code for the other query here:

Solution: There ar variations of this which will also work. Note that we avoid the need to remerge by instead mutating by group (“name”).

```
storms %>%
 group_by(name) %>%
 mutate( H = max(category) ) %>%
 ungroup() %>% #Ok to omit
 filter( H > 0) %>%
 group_by(category) %>%
 summarize( N = n() ) %>%
 mutate( pct = 100*N/sum(N) )
```

c. [10 points] The data.table code below produces the same results as one of the two SQL queries above. First, identify which one and indicate your answer by circling the appropriate choice. Then, write data.table code to produce the same result as the *other* SQL query.

Here is the data.table code, using `storms_dt = as.data.table(storms)` .

```
storms_dt[ , h := max(category), name]
storms_dt[h > 0, .N, keyby = category][ , pct := 100*N/sum(N)][]
```

Which SQL query does this data.table code correspond to? Circle one:

Solution: Query 2

The data.table code above matches *query 2* computing the duration (in units of 6 hours) that storms with any rows above category zero spend in each category.

Write data.table code for the other query here:

Solution:

```
storms_dt[category > 0 , .(category = max(category)), .(name)][
  , .N, keyby = .(category)][, pct := 100*N/sum(N)][]
```

Note that in this solution and in the given dplyr code we do not have any joins as in the SQL query. The join in the SQL query is *remerging* group-level summary statistics to the original table. In dplyr and data.table this is unecessary as we can mutate/update by reference by group.

3. Cross validation

The elastic net is a form of regularized regression which interpolates between the the L2 regularization of ridge regression and the L1 regularization of the lasso.

The elastic net optimization problem is:

$$\hat{\beta}(\alpha, \lambda) = \arg\min_{\beta} \frac{1}{2n}||Y - X\beta||_2^2 + \lambda \left(\frac{1-\alpha}{2}||\beta||_2^2 + \alpha||\beta||_1 \right)$$

where both λ and α are tuning parameters controlling the amount and type of regularization.

In this question, you will write code to select α and λ using cross validation.

Assume you have a vector of response variables, `y_train`, and a matrix of explanatory variables `x_train`. The `glmnet()` function in R estimates $\hat{\beta}(\alpha, \cdot)$ for all values of λ given a fixed α. We can use it as follows:

```
fit = glmnet(y, x, alpha = .5)
new_y_hat = predict(fit, new_x, s=fit$lambda)
```

In the example above, `new_y_hat` is a matrix with each column a prediction, i.e. $\hat{y}_{\alpha,\lambda} = x\hat{\beta}(\alpha, \lambda)$ for a sequence of λ's determined by the algorithm. It always has the same number of columns as x.

a. [10 pts] Write a function to compute the mean squared error for each column of `new_y_hat` given a vector of observed values `new_y`.

Solution:

```
mse = function(y_hat, y){
  # y_hat is an n x p matrix
  # y is a length n vector
  # use broadcasting for efficiency
  colMeans( {y_hat - y}^2 )
}
```

b. [10 pts] Use leave-one-out-cross-validation with the training data x_{train} and y_{train} to estimate the MSE for a fixed sequence $\alpha \in (.05, .10, \ldots, .95)$ at all 19 by `ncol(x)` combinations of α and λ. Store the fits for each α in a list.

Solution: For each fit, we will technically get a slightly different value of λ. Here, we aggregrate λ by position.

```
# alphas to search over
alpha = seq(.05, .95, .05)

# Matrix to store MSEs for each alpha / position in lambda
xv_mse = matrix(0, length(alpha), ncol(x_train))

# each row of x_train is its own fold
n = nrow(x_train)

for(i in 1:n){
  # find fits for each alpha
  fits = lapply(alpha,
    function(a){ glmnet(y_train[-i], x_train[-i, ], alpha = a) } )

  # compute predictions for each alpha
  # each prediction is a 1 x p matrix, pred
  preds = sapply(fits,
    function(fit){ predict(fit, x_train[i,], s=fit$lambdas) } )

  # we could apply our function above row by row to t(preds),
  # but here we vectorize instead
  xv_mse = xv_mse + {t(preds) - y_train[i]}^2 / n
}
```

c. [5 pts] Write code to return the α and λ that minimize the cross-validated MSE.

Solution: Beacuse we don't have a fixed lambda, we find the best alpha and the position of the best lambda. We return lambda from the full training data in this position. When grading, it was sufficient to identify the indices of the best α and $\labmda$ and not their actual values.

```
# Get row and column of minimum in xv_mse
index = which.min(xv_mse)
row = {index - 1} %% nrow(xv_mse) + 1
col = {index - 1} %/% nrow(xv_mse) + 1

# Here is a an alternate version
# which( xv_mse == min(xv_mse), arr.ind = TRUE)
# row = index[1]; col = index[2]

# Here is the best alpha
best_alpha = alpha[row]

# Get lambda in appropriate position for fit to full data
fit = glmnet(y_train, x_train, alpha = best_alpha)
best_lambda = fit$lambdas[col]
```

4. SAS

Repeat the analysis from the SAS script below in R. Use either `dplyr` or `data.table` syntax for grouped operations. Write your R script on a separate sheet of paper.

The script reads in data related to hospital stays for pneumonia or other respiratory infections among Medicare patients in 2014 and 2016. Each row represents a hospital ("hosp_id"") and a particular type of hospital stay (" drg"). The script computes z-scores for each hospital comparing the difference in proportions for one type of stay ("drg"=178) to the others between the two years.

Here is the original SAS code:

```
/* Import the data */
proc import datafile='./data2014.csv' out=ip14;

proc import datafile='./data2016.csv' out=ip16;
run;

/* Subset data */
data p14;
 set ip14;
 where drg in (178, 179, 185);
 n14 = total_discharges;
 keep drg hosp_id n14;

data p16;
 set ip16;
 where drg in (178, 179, 185);
 n16 = total_discharges;
 keep drg hosp_id n16;
run;

/* Merge data */
proc sql;

 create table pneum as
 select *
 from p14
 left join p16
 on p14.drg=p16.drg and p14.hosp_id=p16.hosp_id;

quit;

/* Compute totals, remerge, and create pooled estimates */
proc sql;

 /* Compute totals */
 create table totals as
 select hosp_id, sum(n14) as tot14, sum(n16) as tot16
 from pneum
 group by hosp_id;

 /* Remerge and create pooled estimate*/
 create table pneum as
 select *, (n14 + n16) / (tot14 + tot16) as p_pool
 from pneum p
 left join totals t
 on p.hosp_id = t.hosp_id;

quit;

/* Compute z-scores */
data result;
 set pneum;
 where drg = 178 and p_poll lt 1; /* lt = < */
 p14 = n14/tot14;
 p16 = n16/tot16;
 z = (p16 - p14)/(p_pool*(1-p_pool)*(1/n14 + 1/n16));
run;
```

```
/* Sort by z-score */
proc sort data=result;
 by -z;
run;
```

Solution:

Here is a dplyr solution that does not utilize remerging.

```
# Libraries
library(tidyverse)

# Read the data
ip14 = read_delim('./data2014.csv', delim = ',')
ip16 = read_delim('./data2016.csv', delim = ',')

# Subset
p14 = ip14 %>%
 filter(drg %in% c(178, 179, 185)) %>%
 transmute(drg, hosp_id, n14 = total_dicharges)

p16 = ip16 %>%
 filter(drg %in% c(178, 179, 185)) %>%
 transmute(drg, hosp_id, n16 = total_dicharges)

# Join and comptue totals
pneum = left_join(p14, p16, by = c('drg', 'hosp_id')) %>%
 group_by(hosp_id) %>%
 mutate( p14 = n14/sum(n14), p16 = n16/sum(n16),
         p_pool = {n14 + n16} / sum(n14 + n16)
 )

# Compute result, limiting to hospitals that see both types of cases
result = pneum %>%
 filter(drg %in% 178 & p_pool < 1) %>%
 mutate( z = {p16 - p14}/p_pool*{1-p_pool}*{1/n14 + 1/n16}) %>%
 arrange(desc(z))
```

Here is a data.table solution:

```
# Library
library(data.table)

# Read the data
ip14 = fread('./data2014.csv', delim = ',')
ip16 = fread('./data2016.csv', delim = ',')

# Subset
p14 = ip14[drg %in% c(178, 179, 185), .(drg, hosp_id, n14 = total_dicharges)]
p16 = ip16[drg %in% c(178, 179, 185), .(drg, hosp_id, n16 = total_dicharges)]

# Join and comptue totals
pneum = merge(p14, p16, by = c('drg', 'hosp_id'))

pneum[,`:=`(p14 = n14/sum(n14), p16 = n16/sum(n16),
         p_pool = {n14 + n16} / sum(n14 + n16) ), hosp_id]

# Compute result, limiting to hospitals that see both types of cases
result = pneum[drg %in% 178 & p_pool < 1]
result[ , z := {p16 - p14}/p_pool*{1-p_pool}*{1/n14 + 1/n16} ]
result = result[order(-z)]
```

Part Two

Please choose exactly two questions to answer for this section. Each question is worth 50 points. Answer each of your selected questions on a sheet of white copy paper clearly labeled with your name and the problem selected. Staple these together.

5. Bootstrap resampling and data.table

Background

Contingency tables examining associations between two categorical variables are often analyzed with a chi-squared test of independence. In this test, the null hypothesis of independence is tested using a chi-squared statistic:

$$\chi^2 = \sum_{i,j} (X_{ij} - E_{ij})^2 / E_{ij}, \qquad E_{ij} = X_{i+} X_{+j} / n$$

where $X_{i+} = \sum_j X_{ij}$ and $X_{+j} = \sum_i X_{ij}$. Here X is a contingency table whose rows correspond to the levels of one variable and whose columns correspond to the other.

Pearson residuals $R_{ij} = (O_{ij} - E_{ij}) / \sqrt{E_{ij}}$ can be used to understand which cells differ most from what would be expected under the assumption that two variables are independent.

When the cell counts are sufficiently large, normal approximations allow us to compare the χ^2 test statistic to a chi-squared distribution with degrees of freedom determined by the number of rows and columns in X an to compute a variance for each R_{ij}.

However, when the cell counts are small it is common to compute a p-value using Monte Carlo simulation by redrawing new cell counts under the null distribution. In these settings, if we reject the null we may also wish to use a non-parametric method such as the bootstrap to compute confidence bounds for the Pearson residuals under the alternative.

Instructions

In this problem, you will write data.table code to generate bootstrap confidence bounds for the Pearson residuals. You should use data.table syntax throughout. In your solution, clearly label each section of code with which of the parts below it accomplishes.

Question

Consider the table below (originally from Agresti(2007) p.39, see `help(chisq.test)`:)

```
M <- as.table(rbind(c(762, 327, 468), c(484, 239, 477)))
dimnames(M) <- list(gender = c("F", "M"),
                    party = c("Democrat","Independent", "Republican"))
M
```

```
##       party
## gender Democrat Independent Republican
##      F      762         327        468
##      M      484         239        477
```

Here is the same table in long format as a data.table.

```
library(data.table)
dt = data.table(
  cell_count = as.numeric(M),
  rows = rep(rownames(M), ncol(M)),
  cols = rep(colnames(M), each = nrow(M))
)
dt
```

```
##    cell_count rows        cols
## 1:        762    F    Democrat
## 2:        484    M    Democrat
## 3:        327    F Independent
## 4:        239    M Independent
## 5:        468    F  Republican
## 6:        477    M  Republican
```

a. [5 pts] First, add the following three columns to `dt` *by reference*:

- “cell” - corresponding to the row numbers 1-6,
- “n” - corresponding to the total sample size,
- “p” - the observed probability for each cell.

Solution:

```
dt[, n := sum(cell_count)]
dt[, `:=`(cell = 1:.N,  p =  cell_count / n) ]
```

b. [5 pts] Next, compute sums for each level of `rows` and each level of `cols`, use these to compute the expected cell counts `E` and finally the Pearson residuals `R`. Add each of these four columns to `dt` *by reference*.

Solution:

```
dt[, row_sum := sum(cell_count), .(rows)]
dt[, col_sum := sum(cell_count), .(cols)]
dt[, E := row_sum*col_sum/n]
dt[, R := {cell_count - E}/sqrt(E)]
```

c. [10 pts] Now, set the number of bootstrap samples `B` to `1e4` and then create a new data.table `dt_boot` containing `B*n` samples of the cell labels using the cell probabilities `p`. Also create a variable `boot` labeling each row as belonging to one of `B` bootstrap samples.

Solution:

```
B = 1e4
dt_boot =
 dt[ , .(cell = sample(cell, B*n, replace = TRUE, prob = p),
         boot = rep(1:B, each = n[1])
        ) ]
```

d. [5 pts] Use grouped aggregation to create a new data.table `dt_cells` containing the cell counts in each of the `B` bootstrap samples.

```
dt_cell = dt_boot[ , .(n_boot = .N), keyby = .(boot, cell)]
```

e. [5 pts] Add the `rows` and `cols` labels from `dt` to `dt_cell` along with the original sample size `n`. Hint: use `merge()`.

```
dt_cell = merge( dt_cell, dt[ , .(cell, rows, cols, n)], by = 'cell' )
```

f. [10] Add columns `row_sum` and `col_sum` by reference giving the row and column totals for each bootstrap sample. Then, again by reference, compute the expected cell counts and Pearson residuals for each bootstrap sample.

```
dt_cell[, row_sum := sum(n_boot), .(boot, rows)]
dt_cell[, col_sum := sum(n_boot), .(boot, cols)]
dt_cell[, E := row_sum*col_sum/n]
dt_cell[, R := {n_boot - E} / sqrt(E)]
```

g. [5 pts] Assuming a 95% confidence level, use the percentile method to compute lower and upper confidence bounds for the Pearson residuals of the 6 cells.

```
r_boot = dt_cell[ , .(lwr = quantile(R, .025), upr = quantile(R, .975)),
            by = .(rows, cols)]
```

h. [3 pts] Merge with the observed residuals and create a nicely formatted string “R (L, U)”.

```
tab = merge(r_boot[ , .(rows, cols, lwr, upr)], dt[, .(rows, cols, R)], by = c('rows', 'cols')
 )[ , est := sprintf('%4.1f (%4.1f %4.1f)', R, lwr, upr) ][]
```

i. [2 pts] Reshape the resulting table to wide format so that it resembles the original 2 x 3 contingency table.

```
dcast(tab, rows ~ cols, value.var = 'est')
```

6. Principle Components Regression

Principal Component regression (PCR) is a form of regularized linear regression in which, the dependent variable Y is regressed on the principal components of the explanatory variables X rather than on X directly. In PCR we typically use only a subset of these principle components leading resulting in regularization.

Here is the procedure for PCR using the first k principal components:

1. Compute the mean of Y $\bar{y}$ and then center Y.
2. Compute the column means of X, $\bar{x}$ and then center each column of X.
3. Compute the singular value decomposition (SVD) of this centered X, $(X - \bar{x}) = UDV'$.
4. Compute the first k principal components of X, $W_k = XV_{1:k}$ using the first k columns of V. W_k can be thought of as a new set of "derived" covariates.
5. Regress Y on W_k, i.e. find $\hat{\gamma}_k$ to minimize $||Y - W_k\gamma||^2$ over γ. Note that $\hat{\gamma}_k = (W_k'W_k)^{-1}W_k'Y$ but that we should compute the estimate as in standard linear regression.
6. Compute regression coefficients $\hat{\beta}_k = V_k\gamma_k$ on the original scale (of $(X - \bar{x})$).

Given new values of X we can then estimate the mean response as $\hat{Y}_k = \bar{y} + (X - \bar{x})\beta_k$.

Instructions

Write the core components of an R script implementing principle components regression and then show how to use cross-validation to select k. Evaluate the final model using a test-train split as described below. In your solution, clearly label each section of code with which of the parts below it accomplishes.

You may consider writing parallel code for parts (a) or (b). If you do, you may choose any of the paradigms for parallel or asynchronous computing we learned in the course. If you write parallel code, you can earn 5 additional points in each section though you can not earn more than 50 points total for this question.

a. [20 points] Write a function `pcreg` that accept a response vector $Y \in \mathbb{R}^n$, a data matrix $X \in \mathbb{R}^{n \times p}$ and a sequence of k's defaulting to $1 \ldots p$, and then computes the following quantities and stores them in a list: (1) the estimated mean $\bar{y}$ of y, (2) the column means $\bar{x}$ used to center each column of X, (3) a matrix of regression coefficients for each value of k. Change the class attribute of this list to "pcreg" before returning it.

Solution:

```
library(future)
pc_reg = function(Y, X, k = 1:p){

 # Center X and Y
 ybar = mean(Y)
 xbar = colMeans(X)
 X = X - rep(xbar, each = nrow(X))
 Y = Y - ybar

 # Compute SVD
 X_svd = svd(X)

 # Compute new variables, Wk = X %*% V[,1:k]
 # then solve normal equations Wk'Wk %*% g = Wk'Y

 # Note that Wk'Y = V[,1:k]'X'Y which is
 # just the first k rows of  V'X'Y.
 # We don't need to compute this repeatedly in
 # the loop below.

 VtXtY = crossprod(X_svd$v, crossprod(X, Y))

 # Since, Wk'Wk = V'[1:k, ]X'XV[,1:k] we can also
 # compute X'X just once
 XtX = crossprod(X, X)
 v = X_svd$v

 # Finally, note that because V is orthogonal, V'[1:k,](XtX)V[,1:k]
 # is diagonal.

 future_beta = list()
 for(i in k){
   future_beta[[i]] = future(
   {
     gamma = solve( crossprod(v[,1:i], XtX) %*% v[,1:i],  VtXtY[1:i,])
     v[,1:i] %*% matrix(gamma, ncol=1)
   } )
 }

 betas = lapply(future_beta, value)
 betas = do.call('cbind', betas)

 # List to return
 out = list(ybar, xbar, betas)

 # Modify class and return
 class(out) = 'pcreg'
 out
}
```

Note, it was not required that your solution incorporate all of these efficiencies. However, you did lose 1 point if using `lm` and not attempting to use the normal equations and a second point if, in doing so, you add an intercept to X, e.g. `coef(lm(Y~W))` and not `coef(lm(Y~0+W))`. The intercept is redundant as Y has already been centered.

b. [10 points] Write an S3 *predict* method for an object of class “pcreg” as returned by your function of the same name above that takes three arguments: (1) the “pcreg” object, (2) values of `X` on which to

predict, (3) an integer `k` giving the number of principal components to use when forming the predictions. Note that the prediction should be $\hat{Y}_k = \bar{y} + (X - \bar{x})\beta_k$.

Solution:

```
predict.pcreg(pcreg, X, k = ncol(X) ){
  with(pcreg,
   ybar + {X - rep(xbar, each = nrow(X))} %*% betas[,k]
  )
}
```

c. [5 points] Consider a data matrix 'X' of independent variables and a vector 'Y' of outcomes each with $n = 1e3$ rows. Using an 80-20 split, divide this data into testing and training sets.

Solution:

```
train = sample(nrow(X), ceiling(.8*nrow(X)), replace = FALSE)
Xtrain = X[train,]
Ytrain = Y[train,]
Xtest = X[-train,]
Ytest = Y[-train,]
```

d. [10 points] Write serial or parallel code to select a `k` that minimizes the mean-squared error (MSE) using ten-fold cross-validation.

Solution:

```
folds = 1 + {nrow(Xtrain) - 1} %% 10
p = ncol(Xtrain)
mse_future = vector(length = 10, mode = 'list')

# We call predict with all k's at once
for(fold in 1:10){
  mse_future[[fold]] = future({
     in_rows = folds == fold

     # Fit
     fit = pcreg(Ytrain[in_rows,], Xtrain[in_rows,], k=1:p)

     # Predict
     yhat = predict(fit, Xtrain[!in_rows,], k=1:p)

     # MSE for each K
     colMeans({Ytrain[!in_rows] - yhat}^2)
  })
}

# Get values for all futures
mse = lapply(mse_future, value)
mse = colMeans( do.call('rbind', mse) )

best_k = which.min(mse)
```

e. [5 points] Compute the MSE on the test data using the selected `k`.

```
# refit to all training data
fit = pcreg(Ytrain, Xtrain)

# predict using selected k
test_mse =  mean({Ytest - predict(fit, Xtest, k=best_k)}^2)
```

7. Permutation testing

Suppose you work for a data science team in a mid-size company and have developed a kernel regression model to predict the salaries of non-hourly employees in your company. Your model will be used to help target planned salary increases toward those who, according to the model, are paid significantly less than predicted i.e. have large negative residuals.

You have already selected features and tuning parameters using cross-validation and evaluated your model on hold out data. The project leader is happy with the model performance, but worried that your model may reinforce disparities in pay based on gender. Although your model does not explicitly include gender as a feature, you have been asked to *determine whether model residuals (actual less predicted salary) are significantly associated with gender*.

a. [35 pts] Write custom R code to implement a permutation test to assess whether the median residuals significantly differ by gender. You should assume your data and model predictions are in a data.frame `pay_data` with columns: `salary` (actual salary), `pred_salary` (predicted salary), `gender`, along with other features included in the model. Write serial code for this part, but structure it in a way that will be easy to make parallel in part b.

Solution:

```
library(data.table)
# Create a new data frame with only the residuals and gender
# to avoid reordering irrelevant variables.

resid_gender = with( pay_data, data.table( r = salary - pred_salary,
                                           gender = gender) )

# function to compute the differences in medians for the residuals
# note the use of keyby to get a consistent order
diff_med = function(df, what = 'r', by = 'gender'){
  diff( df[ , .(mr = median(.SD[[what]])), keyby = by]$mr )
}

# Observed difference
obs_diff = diff_med(resid_gender, 'r', 'gender')

# function for permuting gender label in a new data.frame

permute_gender = function(df){
    df[ , .(r, gender = sample(gender, .N, replace = TRUE) )]
}

# put it all together
diff_perm = function(df, ...){
  diff_med( permute_gender(df), ... )
}

# do the permutations
nperm = 1e4
perm_diffs = sapply(1:nperm, diff_perm,
                    df = resid_gender, what = 'r', by = 'gender')

# compute the two-side permutation p-value
# also acceptable without the + 1
p = { sum( abs(perm_diffs) >= abs(obs_diff) )  + 1 } / {nperm + 1}
```

b. [15 pts] Write a parallel version of your code above. You may choose any of the paradigms for parallel or asynchronous computing we learned in the course.

Solution: We did most of the work in part a, here we can just replace `lapply` with parallel computations.

```
perm_diffs = mclapply(1:nperm, diff_perm,
                    df = resid_gender, what = 'r', by = 'gender')

# compute the two-side permutation p-value
p = { sum( abs( unlist(perm_diffs) ) >= abs(obs_diff) )  + 1 } / {nperm + 1}
```

8. data.table and SAS

Below is a portion of an R script from problem set 3, question 1 in which we use data.table and the RECS data set to estimate the proportion of US homes in each census division with stucco construction and compute confidence bounds using the replicate weights method.

Repeat this analysis in SAS assuming the RECS data is available as `recs.sas7bdat` and the `weights_long` table is also available as `weights_long.sas7bdat` both in the current directory.

```
# libraries: ------------------------------------------------------------
library(tidyverse); library(data.table)

# Multiplier for confidence level: --------------------------------------
m = qnorm(.975)

# data: -----------------------------------------------------------------
file = './recs2015_public_v3.csv'
recs = fread(file)

# weights: --------------------------------------------------------------
brrwt_cols =  paste0('BRRWT',1:96)
weights =  recs[, c('DOEID', brrwt_cols), with = FALSE]
weights_long = melt(weights, id.vars = 'DOEID', measure.vars = brrwt_cols,
                    variable.name = 'repl', value.name = 'w')

# Point estimate: -------------------------------------------------------
p_stucco = recs[ , .(p_stucco = sum( NWEIGHT*{WALLTYPE == 4} ) / sum(NWEIGHT) ),
                 division]

# Estimates from replicate weights: -------------------------------------
p_stucco_r =
  weights_long[recs, .(w, repl, WALLTYPE, division), on = 'DOEID'] %>%
  .[ , .( r_stucco =  sum( w*{WALLTYPE == 4} ) / sum(w) ), .(division, repl)]

# Compute standard errors: ----------------------------------------------
p_stucco_r = merge(p_stucco_r, p_stucco, by = 'division')

p_stucco = p_stucco_r[ , .(p_stucco = p_stucco[1],
                           se_stucco = 2*sqrt( mean( {r_stucco - p_stucco}^2 ) )
                           ) , .(division)]

# Compute confidence bounds: --------------------------------------------
p_stucco[ , `:=`( lwr = pmax(p_stucco - m*se_stucco, 0),
                  upr = p_stucco + m*se_stucco
                 ) ]
```

The first line of your script should be:

```
libname mlib './';
```

Solution: A large number of you chose this question, but then used `proc summary` or `proc sql` as if they worked just like data.table. As you'll see in the solution below, we need to use remerging to accomplish this because the aggregation functions in `proc summary` and `proc sql` work on *existing variables* and do not work with expressions to compute new variables on the fly.

```
/* estimate the proportion of US homes in each
 * census division with wood shingle roofing
 */

/* library */

`libname mlib './'; `

/***********************************************
 * compute point estimates by divsion follows
 * 1: Compute derived variables
 * 2: Sum up by division
 ***********************************************
 */

/* Derived variables */
data recs;
 set mlib.recs;
 stucco=0;
 if walltype=4 then stucco=1;
 nw4=nweight*stucco;
 keep doeid nweight division walltype nw4;
run;

/* Point-estimates by division */
proc sql;

 create table p_stucco as
 select division, sum(nw4) / sum(nweight) as p_stucco
 from work.recs
 group by division;

quit;

/*******************************************
 * Compute replicate estimates as follows:
 * 1. join needed cols to long-form weights
 * 2. sum by by divsion/replicate weight
 *******************************************
 */

 proc sql;

 /* merge */
 create table p_stucco_r as
 select division, w, stucco*w as nw4, repl
 from mlib.weights_long wl
 left join work.recs r
 on wl.doeid = r.doeid;

 /* sum by division and replicate */
 create table p_stucco_r as
 select division, sum(nw4)/sum(w) as r_stucco
 from p_stucco_r
 group by divsion, repl;

 /* merge with p_stucco for std errors */
```

```
 create table p_stucco_r as
 select *
 from p_stucco_r r
 left join p_stucco p
 on r.division = p.division;

 quit;

/*************************************
 * Compute standard errors by:
 * 1. Merge with point estimates (done above)
 * 2. Compute residual and then replicate variance
 * 3.
 *************************************
 */

/* merge and compute "residual"*/
data p_stucco_r;
 residual = r_stucco-p_stucco;
 residual = residual*residual;

/* compute standard errors */
proc sql;
 create table p_stucco as
 select min(p_stucco) as p_stucco,
        mean(residual) as  v_stucco
 from p_stucco_r
 group by division;
quit;

/* compute confidence bounds */
data p_stucco;
 set p_stucco;
 se_stucco = 2*sqrt(v_stucco);
 lwr = p_stucco - 1.96*se_stucco;
 if lwr lt 0 then lwr=0;
 upr = p_stucco + 1.96*se_stucco;
```

9. data.table and dplyr

This is the same question as question 7, with the exception that you are asked to provide a dplyr translation.

Below is a portion of an R script from problem set 3, question 1 in which we use data.table and the RECS data set to estimate the proportion of US homes in each census division with wood shingle roofing and compute confidence bounds using the replicate weights method.

Repeat this analysis using dplyr syntax (note you originally did this as PS1 Q3).

Solution: See PS1 Q3 for a solution.

```
# libraries: ------------------------------------------------------------
library(tidyverse); library(data.table)

# Multiplier for confidence level: --------------------------------------
m = qnorm(.975)

# data: -----------------------------------------------------------------
file = './recs2015_public_v3.csv'
recs = fread(file)

# weights: --------------------------------------------------------------
brrwt_cols =  paste0('BRRWT',1:96)
weights =  recs[, c('DOEID', brrwt_cols), with = FALSE]
weights_long = melt(weights, id.vars = 'DOEID', measure.vars = brrwt_cols,
                    variable.name = 'repl', value.name = 'w')

# Point estimate: -------------------------------------------------------
p_stucco = recs[ , .(p_stucco = sum( NWEIGHT*{WALLTYPE == 4} ) / sum(NWEIGHT) ),
                 division]

# Estimates from replicate weights: -------------------------------------
p_stucco_r =
  weights_long[recs, .(w, repl, WALLTYPE, division), on = 'DOEID'] %>%
  .[ , .( r_stucco =  sum( w*{WALLTYPE == 4} ) / sum(w) ), .(division, repl)]

# Compute standard errors: ----------------------------------------------
p_stucco_r = merge(p_stucco_r, p_stucco, by = 'division')

p_stucco = p_stucco_r[ , .(p_stucco = p_stucco[1],
                           se_stucco = 2*sqrt( mean( {r_stucco - p_stucco}^2 ) )
                           ) , .(division)]

# Compute confidence bounds: --------------------------------------------
p_stucco[ , `:=`( lwr = pmax(p_stucco - m*se_stucco, 0),
                  upr = p_stucco + m*se_stucco
                ) ]
```