Final Exam

Statistics 506, Fall 2017

Tuesday December 19, 10:30-12:30

Name:

Uniqid:

Instructions

This is the final exam for Stats 506 in Fall 2017. The exam consists of two parts. You should answer **all** four questions from part one.

There are four questions in part two from which you should choose **exactly two** questions to answer.

Answer each question on a separate piece of paper. Please **print** your name and **uniqid** on each page submitted. The one exception is for question 4, part b where you should respond directly on the page. When finished, please staple the pages of your exam 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!

Part 1 [120 points]

Answer all of the following questions on a separate sheet of paper. Clearly indicate the question and part being answered and write your name on each page.

Question 1 [30 points]

This question uses the "Starwars" data distributed with dplyr. It contains information on characters from the movie franchise "Star Wars". No knowledge of the films are needed to answer this question.

Here we extract the character names "name", two continuous variables "height" and "mass", and two categorical variables "homeworld" and "species."

```
library(dplyr); library(data.table)
sw_dt = as.data.table(dplyr::starwars %>% dplyr::select(name, height, mass, homeworld, species))
sw_tbl = dplyr::starwars; class(sw_tbl)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

Answer each of the following two questions three times: once each using "dplyr", "data.table", and "SQL" syntax. Your solutions should not create any intermediate tables but instead should consist of a single pipe, chained expression, or SQL block. **Label your answers as follows: 1a-dplyr, 1a-data.table, 1a-SQL, 2a-dplyr, ...**

Your SQL solutions can be generic meaning you do not need to determine how the SQL statements are passed to the specific SQL engine.

a. [15 pts / 5 pts each] Find all levels of "homeworld" shared by two or more "species". Order your results in the most appropriate way.

b. [15 pts / 5 pts each] Among all "species" represented by two or more characters, which has the largest average ratio of mass to height?

Solution

The solutions below for part "b" take into account characters with missing height and mass. You did not need to do this.

dplyr

a.

```
sw_tbl %>% group_by(homeworld) %>%
  summarize(n_species=length(unique(species))) %>%
  filter(n_species > 1) %>%
  arrange(-n_species)
```

b.

```
sw_tbl %>%
  mutate(ratio=mass/height) %>%
  group_by(species) %>%
  summarize(n=sum(!is.na(ratio)), ratio=mean(ratio,na.rm=TRUE)) %>%
  ungroup() %>%
  filter(n > 1) %>%
  filter(ratio==max(ratio))
```

data.table

a.

```
sw_dt[,.(n_species=length(unique(species))),by=.(homeworld)
    ][n_species>1
     ][order(-n_species)]
```

b.

```
sw_dt[,`:=`(ratio=mass/height)
    ][,.(n=sum(!is.na(ratio)), ratio=mean(ratio,na.rm=TRUE)),by=species
    ][n>1
    ][ratio==max(ratio)]
```

SQL

a.

```
SELECT homeworld, COUNT( DISTINCT species) AS n_species
  FROM Starwars
 GROUP BY homeworld
HAVING n_species > 1
ORDER BY -n_species
```

Without using "DISTINCT" ...

```
SELECT homeworld, COUNT(species) AS n_species
  FROM (SELECT homeworld, species
          FROM Starwars
        GROUP BY homeworld, species
        )
GROUP BY homeworld
HAVING n_species > 1
ORDER BY -n_species
```

b.

```
SELECT species, mean(mass/height) AS ratio, COUNT(name) AS n
  FROM Starwars
 GROUP BY species
 WHERE mean IS NOT NULL AND height IS NOT NULL
HAVING ratio = max(ratio) AND n > 1
```

Question 2 [30 points]

In this question you will translate data manipulations shown in dplyr syntax into data.table syntax.

This question uses the Lahman baseball data used in class to demonstrate SQL concepts. The Lahman baseball data contains historical statistics from the most prominent US professional baseball leagues. There is a table "Pitching" containing the performance of a particular class of players known as "pitchers". One measure of a pitcher's performance is given by a statistic "WHIP" computed as `3*(H + BB) / IPouts`; lower is better for this measure. Another performance measure "K per 9" is computed as `(SO / IPouts)*27`; for this measure higher is better.

For each of the questions below, you may assume the `Pitching` table is accessible by name in R. You should also assume that a copy of the `Pitching` table coerced to a `data.table` object is stored in `pitch` as shown below.

```
library(dplyr); library(Lahman); library(data.table)
pitch = as.data.table(Pitching)
```

a. Some players (`playerID`) may play for multiple teams within a single year (`yearID`) and league (`lgID`). First we aggregate rows so that there is one row for each unique combination of player, year (since 1960), and league. Then we remove rows with `IPouts` less than 3×162 to remove players who may not have played an entire season.

```
sum_Pitching = Pitching %>%
  filter(yearID >= 1960) %>%
  group_by(playerID, yearID, lgID) %>%
  summarize(H = sum(H), BB=sum(BB), SO=sum(SO), IPouts=sum(IPouts)) %>%
  filter(IPouts >= 3*162)
```

Translate the code above into data.table syntax using the data.table ***`pitch`****.* [10 points]

Solution:

```
# Here is one approach
sum_pitch =
  pitch[yearID >= 1960, .(H=sum(H), BB=sum(BB), SO=sum(SO), IPouts=sum(IPouts)),
        by = .(playerID, yearID, lgID)][IPouts >= 3*162]

## Here is an alternate approach using .SD
sum_pitch_a =
  pitch[yearID >= 1960, lapply(.SD,sum),  by = .(playerID, yearID, lgID),
        .SDcols=c("H","BB","SO","IPouts")][IPouts >= 3*162]
```

b. *Using the data.table you created in part a, add variables* ***`WHIP`*** *and* ***`K_per_9`*** *using the formulas given above. You should use reference semantics to modify the table in place.* [10 points]

To illustrate, here is a dplyr version that does this without updating by reference.

```
sum_Pitching = sum_Pitching %>%
  mutate(WHIP=3*{H+BB}/IPouts, K_per_9 = 27*SO/IPouts)
```

Solution:

```
sum_pitch[,`:=`(WHIP=3*{H+BB}/IPouts, K_per_9=27*SO/IPouts)]

## or

sum_pitch[,c("WHIP","K_per_9"):=.(3*{H+BB}/IPouts, 27*SO/IPouts)]
```

c. *Using the table created above, write code to find the player with the* ***lowest*** *WHIP in each league and year. Include only the years 2000-2016. Order the final table first by year, then alphabetically by league.* [5 pts]

Here is a dplyr version to help you understand what is going on.

```
whip_lead = sum_Pitching %>%
  filter(yearID >= 2000 & yearID <= 2016) %>%
  group_by(yearID, lgID) %>%
  arrange(WHIP) %>%
  filter(c(TRUE,rep(FALSE,n()-1))) %>%
  select(yearID, lgID, playerID, WHIP) %>%
  arrange(desc(yearID), lgID)
```

Solution:

```
whip_lead =
  sum_pitch[, .SD[which.min(WHIP)], by=.(yearID,lgID),
          .SDcols=c('playerID','WHIP')][order(-yearID,lgID)]
```

d. *Finally find the player with the* ***highest*** *"K per 9" in each league and year and produce a table, as in part (c), containing just* `yearID`*,* `lgID`*,* `playerID`*, and* `K_per_9`*.* [5 pts]

Solution:

```
k_lead =
  sum_pitch[, .SD[which.max(K_per_9)], by=.(yearID,lgID),
          .SDcols=c('playerID','K_per_9')][order(-yearID,lgID)]
```

Question 3 [35 points]

In part "a" of this question you will write a short SAS script using `PROC SQL` to repeat the analyses question two above. In part "b" you will demonstrate your understanding of SQL joins by extending the analysis. You should assume copies of the relevant tables are stored as `.sas7bdat` files in a directory `~/Lahman/`.

a. *Write a SAS script that uses `PROC SQL` to repeat the analyses in parts a-d of question two. Hint: you can easily combine parts (a) and (b) into a single block of code.* [20 pts]

b. Use a join to produce a table showing only those years when the same player led their respective league in both WHIP and K per 9. [5 pts]

c. *The table `Master` contains columns `playerID`, `nameFirst`, and `nameLast` along with other player information. It can be linked to the Pitching table by `playerID`. Use a join within `PROC SQL` to replace `playerID` with columns `First` and `Last` giving the player names from the `Master` table.* [5 pts]

Solution: The solutions to 3a-3c are combined into a single script.

```
libname mylib '~/Lahman';

proc sql;
  /* 2a: I also add the filter from c. */
  create table sum_pitching as
  select sum(H) as H, sum(BB) as BB, sum(SO) as K, sum(IPouts) as IPouts
    from mylib.Pitching
    where yearID ge 2000 and yearID le 2016
   group by playerID, yearID, lgID
   having IPouts ge 3*162;

  /* 2b */
  create table sum_pitch as
  select playerID, yearID, lgID, 3*(H+BB)/IPouts as WHIP, 27*K/IPouts as Kper9
    from sum_pitching;

  /* 2c */
  create table whip_lead as
  select playerID, yearID, lgID
    from sum_pitch
    where WHIP=min(WHIP)
  group by yearID, lgID
  order by -yearID, lgID;

  /* 2d */
  create table k_lead as
  select playerID, yearID, lgID
    from sum_pitch
    where Kper9=max(Kper9)
  gorup by yearID, lgID
  order by -yearID, lgID;

  /* 3b */
  create table leaders as
  select w.yearID as year, w.playerID as playerID, WHIP, Kper9
    from WHIP_lead w
  inner join k_lead k
          on w.yearID=k.yearID and w.playerID=k.playerID;
```

```
 /* 3c: I join to the leaders table, but you can join to any of these.*/
 create table coleaders as
 select First, Last, year, WHIP, Kper9
   from leaders l
 left join mylib.Master m
        on l.playerID = m.playerID;

quit;

/* Save some of the created tables for later use.
 * This isn't needed for points on the exam.
 */
data mylib.WHIPleaders_2000to2016;
 set WHIP_lead;

data mylib.Kper9leaders_2000to2016;
  set k_lead;

run;
```

d. *What type or types of join did you use for each of parts (b) and (c)? Under what circumstances would it be necessary to use a left join for part (c)?* [5 pts]

Solution: An inner join wsa needed for part **b** as we wanted only those years when the same player led both categories. For part **c** I used a left join to guard against the event that a `playerID` was missing from the `Master` table. In this case the `Master` table is complete and a left or inner join would work equally well.

Question 4 [25 points]

This question concerns R's C interface in part a and the Rcpp package in part b.

a. Consider the two C functions below defined in a file `funcs.c`. Then answer questions (i)-(iv).

Here are the contents of the file `funcs.c`:

```
#include <R.h>
void C_one_pass(std::vector<double>* x, int* len_x, double* result)
{
  double m=0, s=0;
  double n = (double) len_x[0];

  for(int i=0; i < len_x[0]; i++){
     m += x[i]/n;
     s += x[i]*x[i]/n;
  }

  result[0] = n / (n - 1) * (s - m*m);
}

void C_two_pass(std::vector<double>* x, int* len_x, double* result)
{
  double m=0, s=0;
  /* cast length of x to double */
  double n = (double) len_x[0];
```

```
  for(int i=0; i < len_x[0]; i++){
     m += x[i]/n;
  }

  for(int i=0; i < len_x[0]; i++){
     s += (x[i] - m)**2;
  }

  //replace: result[0] = n / (n - 1) * s;
  // with
  result[0] = s / (n-1);
}
```

(i) Write mathematical formulas to represent what is being computed by the functions ***one_pass*** *and* ***two_pass****. What is the name of the quantity computed by both of these functions?* [10 pts]

Solution:

These are two methods for computing the varaiance. The former has the advantage of needing to loop through the data just once, but is less numerically stable than the latter.

Assuming a vector $x \in \mathbb{R}^n$, `one_pass` computes

$$\frac{n}{n-1}\left(\sum_{i=1}^{n} x_i^2/n - \left(\sum_{i=1}^{n} x_i/n\right)^2\right)$$

while `two_pass` first computes

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

and then computes the variance as

$$\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

(ii) Before we can call these functions from within R, they need to be compiled into a shared library object (`.so`*). How can this be done from the command line?* [3 pts]

Solution: `R CMD SHLIB funcs.c`

(iii) After executing the command in (ii) how do we link against it from within an interactive R session? [2pts]

Solution: `dyn.load("funcs.so")`

(iv) The skeleton of an R wrapper for calling either of these two functions is given below. Complete the wrapper with calls to the underlying C functions using `.C`*.* [5 pts]

```
compute_c = function(x,  method=c('one_pass','two_pass') ){

  n = length(x)
  result = 0

  method = match.call(method)
  if(method=='one_pass'){
```

```
    # write one line of code below
    result = .C("one_pass", as.numeric(x), as.integer(n), as.numeric(result))[[3]]

  } else {
    # write a second line of code here
    result = .C("two_pass", as.numeric(x), as.integer(n), as.numeric(result))[[3]]
  }

  return(result)
}
```

b. *The functions above have been translated into C++ below and are saved in a file:* `Cpp_funcs.cpp`*. Add the appropriate tags to ensure they are callable from R after calling* `Rcpp::sourceCpp('./Cpp_funcs.cpp')`*.* [5 pts] Write directly on the exam for this question.

Contents of the file `Cpp_funcs.cpp`:

```
#include <Rcpp.h>

// [[Rcpp::export]]
double one_pass(std::vector<double> x)
{
  double m=0, s=0;
  double n = (double) x.size();

  for(int i=0; i < x.size(); i++){
     m += x[i]/n;
     s += x[i]*x[i]/n;
  }

  return = n / (n - 1) * (s - m*m);
}

// [[Rcpp::export]]
double two_pass(std::vector<double> x)
{
  double m=0, s=0;
  double n = (double) x.size();

  for(int i=0; i < x.size(); i++){
     m += x[i]/n;
  }

  for(int i=0; i < x.size(); i++){
     s += pow((x[i] - m),2);
  }

  return s / (n-1);
}
```

Part 2 [80 points]

Choose precisely two questions from this section to answer. Clearly indicate which two questions you would like graded by *circling* the question numbers below. Do not circle more than two.

Please grade the two circled questions:
Question 5, Question 6, Question 7, Question 8.

Each question in part two is worth 40 points.

Question 5

This question concerns memory in R. The first part asks you to provide some basic facts. Then you are asked to apply this knowledge to determine the more efficient storage structure for two example vectors.

a. [20 pts / 5 each] *In R, how many bytes are needed to store each of the vectors whose sizes are stored in i-iv created below?*

```
library(pryr)
## An empty vector
i = object_size(vector(mode='numeric',length=0))
i
```

```
## 40 B
```

```
## An integer vector of length 2
ii = object_size(1L:2L)
ii
```

```
## 48 B
```

```
## A numeric vector of length 2
iii = object_size(c(pi,2*pi))
iii
```

```
## 56 B
```

```
## A character vector of length 2
## Hint: Use the output below and then consider how much memory is needed
#        for each pointer to the global string pool.
object_size('a')
```

```
## 96 B
```

```
iv = object_size(rep('a',2))
iv
```

```
## 104 B
```

b. [10 pts] In this part you are asked to compare the size of two factor variables. Note that you do not need to know the actual size of each factor to answer the question. Also keep in mind that because of the global string pool, the actual values of the strings does not matter.

Will the Boolean commented comparison below return `TRUE` *or* `FALSE`*? Briefly explain your answer.*

```
b1 = c('a', 'b', 'c')[sample(1:3,1e4,replace=TRUE)]
## TRUE or FALSE?
object_size(b1) > object_size(as.factor(b1))
```

```
## [1] TRUE
```

```
object_size(b1); object_size(as.factor(b1))
```

```
## 80.2 kB
```

```
## 40.6 kB
```

Solution: Here there are very few levels relative to the length of the vector so the memory for the factor is dominated by storing 1e4 integers at 4B each. In contrast, the character vector needs 1e4 8B pointers to the global string pool.

Will the Boolean comparison commented below generally return ***TRUE*** *or* ***FALSE****? Briefly explain your answer.*

```
sample_word = function(n) paste(sample(letters,n,replace=TRUE), collapse='')
sample_word(5)
```

```
## [1] "kntni"
```

```
b2 = sapply(1:1e4,function(i) sample_word(5))
## TRUE or FALSE?
object_size(b2) > object_size(as.factor(b2))
```

```
## [1] FALSE
```

```
object_size(b2); object_size(as.factor(b2))
```

```
## 560 kB
```

```
## 600 kB
```

Solution: In this case the strings comprising `b2` will be all or nearly all unique with high probability. Since there are (almost) as many levels as as values, the integers make `as.factor(b2)` about 1.5 times more expensive than the pointers alone.

c. [10 pts] Recall that R has copy-on-modify semantics and that vectors must have all elements of the same type. *Use your knowledge of these two facts to determine the values, including units, held in* `c_i` *and* `c_ii` *below. Briefly explain each of your answers.*

```
x <- y <- z <- 1L:1e4L
object_size(x)
```

```
## 40 kB
```

```
object_size(x,y,z)
```

```
## 40 kB
```

```
y[8] = 3.14
c_i = object_size(y)
c_i
```

```
## 80 kB
```

```
z[8] = 4L
c_ii = object_size(x,z)
c_ii
```

```
## 80.1 kB
```

Solution For `c_i` we are promting integers to vectors, doubling the cost: 40B + 1e4*8B = 80Kb. After modifying `z`, `x` and `z` no longer point to the same data. Hence we have `c_ii` $= 2(40B + 1e4 \times 4B) = 80.08Kb$.

Question 6

A researcher studying the SAT scores of incoming University of Michigan freshman has wide-form data in comma delimited format from the past 20 years. Owing to a change if SAT format, some rows have two scores per student (Reading and Math) while some students have three (Reading, Math, and Writing).

Before fitting mixed models to answer the substantive question, the researcher needs to transform this data from wide to long format. After transformation she wishes to drop rows with missing writing score. The first 4 rows of the data file "scores.csv" are shown below.

```
id,high_school,gender,reading,math,writing
7775,Okemos,M,750,640,650
7779,Clawson,F,750,800,
7881,Kalamazoo Central,F,600,650,750
8772,Pioneer,M,740,730,
```

a. [10 points] *Show the first 10 rows of the resulting long-form data.*

```
id,high_school,gender,test,score
7775,Okemos,M,reading,750
7775,Okemos,M,math,640
7775,Okemos,M,writing,640
7779,Clawson,F,reading,750
7779,Clawson,F,math,800
7881,Kalamazoo Central,F,reading,600
7881,Kalamazoo Central,F,math,650
7881,Kalamazoo Central,F,writing,750
8772,Pioneer,M,reading,740
...
```

b. [30 points] *Write a complete program in the language of your choice to read this data, convert to long format, and drop rows with missing writing score. At the end of your script you should either save the long-form data in a native format or export again to csv.*

Here is a solution using "tidyr::gather" within R.

```
## Convert to SAT scores to long form for research XX
## Date: Dec 19, 2017

# prep workspace
rm(list=ls())
library(tidyr); library(dplyr)

# read data
scores = read.csv('./scores.csv')

# reshape to long format and filter missing
scores_long =
 tidyr::gather(scores,test,score,reading:writing) %>%
 arrange(id,test) %>%
 filter(!is.na(score))

# write output
write.csv(scores_long, file='./scores_long.csv')
```

A somewhat common mistake was to filter before conversion; throwing away all data from ids without a writing score.

Question 7

This question concerns parallel computing and S3 methods in R.

Consider the following problem. Suppose you have flight data for the entire US from 2010-2016 similar to the NYC flights data used in class. Recall that this data contains information such as the carrier, origin and destination airport, distance, along with arrival and departure delays.

You are interested in developing a tool that predicts the expected arrival delay (`arr_delay`) for all flights on the following day using a model trained on the previous 30 days of data. You plan to evaluate a number of predictive models, so define an S3 class `ml_model` which is a list with the following elements:

`data` - a data.table containing relevant flight data `fit` - a function taking a subset of `data` as argument and returning a fitted model object with its own predict method `loss` - the loss function used to compare predictions to observed values.

This `ml_model` object can be used to predict flight delays on January 31, 2014 as follows:

```
# Express today as a date object
today = lubridate::ymd('2014-01-31')

ml_fit = with(ml_model,
  fit(data[date < today & date >= (today-30)])
)

pred = predict(ml_fit, ml_model$data[date==today])

ml_loss = with(ml_model, loss(data[date==today]$arr_delay, pred))
```

a. [10 points] Assume an `ml_model` object with data for all of 2014 is available in R. *Write a function* ***`day_loss`*** *to compute the loss for an abritray day* ***`day`***.

Solution:

Here is a minimal solution. A complete solution would include checking the class of inputs and ensure `(day-30) > 0` so the model is always fit with 30 days of training data.

```
day_loss = function(day,ml_model=ml_model) {
  ml_fit = with(ml_model, fit(data[date < day & date >= (day-30)]))
  pred = predict(ml_fit, ml_model$data[date==today])
  with(ml_model, loss(data[date==today]$arr_delay, pred))
}
```

b. [10 points] *Write a* ***`for`*** *loop to compute the average one-day-ahead loss from January 31, 2014 to December 31, 2014 using your function from part "a".*

Solution: This solution assumes an integer can be added to the date class used to produce a new date class. This is true for the class used here.

```
loss = 0
day0 = lubridate::ymd('2014-01-31')
n_days = lubridate::ymd('2014-12-31') - day0
for(i in 0:n_days) {
 loss = loss + day_loss(day0 + i)
}
avg_loss = loss / {n_days + 1}
```

c. [10 points] *Parallelize the* ***`for`*** *loop you wrote in part b.*

Solution: There are several ways to do this including the simple option shown below.

```r
# set up
library(doParallel)
ncores = 8 # Set num cores here

# Parallel backend
cl = makeCuster(ncores)
registerDoParallel(cl)

day0 = lubridate::ymd('2014-01-31')
n_days = lubridate::ymd('2014-12-31') - day0

loss = foreach(n=0:n_days, .packages='lubridate', .combine='+') %dopar% {
  day_loss(n, ml_model)
}

# clean up
stopCluster(cl)
```

d. [10 points] *In part c, are you assuming the entire data set is communicated to or already available on each node of your cluster?* Suppose the data is quite large, such that any given 31 day subset is ~10GB in RAM. *Describe in words (but do* ***not*** *implement) a strategy for computing the average one-day-ahead loss without ever holding the entire data set in working memory. Briefly describe the trade-offs your plan considers.*

Solution: The solution I wrote in part "c" copies the entire data set to each worker in our cluster. For large data as described, it would be better to "chunk" the data into subsets needed to compute the loss. We want to avoid passing the same day-long subsets repeatedly when that day is, say, the first or last day of the training data.

For instance, in a preliminary step we might break the data into month-long chunks written in a compressed format on disk. To evaluate the loss for all days in any given month, we need only read the data for that and the preceding month into memory for the "worker" instance responsible for the days in that month. While we sould still read most months (February-November) twice, we've substantially reduced the communcation burden while maintaing an easy to follow chunking scheme.

Question 8

This question concerns matrix decomposition and ridge regression. In this question you will use the singular value decomposition to compute coefficient estimates for ridge regression. Ridge regression is used when in place of linear regression when our model matrix X is poorly conditioned due to colinearity between its columns or small n (number of rows) relative to p (number of columns).

a. [10 pts] The singular value decomposition (SVD) expresses a matrix X as $X = UDV'$. *Briefly describe the features of U, D, and V. Which of these matrices contains the "singular values" of X?*

Solution: D is a diagonal matrix of the singular values. U and V are orthonormal, meaning $U'U = I$ and $V'V = I$.

b. [10 pts] *Use the SVD of a model matrix X to express the coefficients estimates $\hat{\beta}$ in the least squares problem where $Y \in \mathbb{R}^n$, $X \in \mathbb{R}^{n \times p}$:*

$$\hat{\beta}_{OLS} = \arg\min_{\beta} ||Y - X\beta||^2.$$

Recall that this is done by solving the normal equations:

$$X'X\hat{\beta} = X'Y.$$

Solution: $\hat{\beta} = VD^{-1}U'Y$ with $D^{-1} = \text{diag}(D_{ii}^{-1})$.

c. [10 points] *Write an R function* `lm_svd` *that solves the least squares problem using the SVD. Your function should be parameterized by an $n \times 1$ matrix* `Y` *and an $n \times p$ model matrix* `X`*. It only needs to return the coefficient estimates $\hat{\beta}_{OLS}$.*

Solution:

```
lm_svd = function(Y, X){
 with(svd(X), v %*% diag(1/d) %*% t(u) %*% Y)
}
```

d. [10 points] In ridge regression we add a regularization term to the least squares criterion in order to shrink the coefficients toward zero:

$$\hat{\beta}_{ridge} = \arg\min_{\beta} ||Y - X\beta||^2 + \lambda||\beta||^2.$$

For a given λ the solution is given by the following formula:

$$\hat{\beta}_{ridge} = VD_\lambda U'Y$$

where D_λ is a diagonal matrix with $[D_\lambda]_{ii} = \frac{D_{ii}}{D_{ii}^2 + \lambda^2}$.

Using your answer to part c as a model, write a function `ridge_svd` *to output the ridge regression coefficients for a given λ.*

Solution:

```
ridge_svd = function(Y, X, lambda=0){
 with(svd(X), v %*% diag(d/{d^2 + lambda^2}) %*% t(u) %*% Y)
}
```