Take-Home Final Exam

Statistical Computing, 36-350

Due December 7, 2015 at 11:59pm

Background. The American Community Survey (ACS) is a yearly survey given to a sample of the U.S. population, and collects information regarding demographics, occupations, educational attainment, veterans, whether people own or rent their home, and a variety of other attributes. The data set you will use in this exam is aggregated data between 2009 and 2013 for each of the 74,020 tracts—small Census Bureau statistical areas, nested within counties—in the United States. (Thanks to Jerzy Wieczorek for help assembling and cleaning this data.) In this exam, you are asked to manage, manipulate and analyse this data using R.

Part 0: Instructions

Make sure you read these.

- Unless you have opted out of the pairing process, you will have been assigned a random partner for this final exam. One of you or your partner must upload a file called `AndrewID1-AndrewID2-final.Rmd`, where `AndrewID1` and `AndrewID2` are your and your partner's Andrew IDs, in alphabetical order. If two reports are uploaded, then one of the two reports (between you and your partner's, chosen at random) will be graded, and both partners will be assigned that same grade. Thus you must work closely together.
- You should download the "final.Rmd" file from course webpage and start filling in your answers from there. Display all code that you write, and every output that you're asked to provide, but suppress unnecessary outputs (e.g., printing out of long data tables). Avoid the `echo=FALSE` option or the inline code style.
- As usual, a submission that does not knit as HTML (*including* stack space overflow) will get a zero.
- You only have to do **one of** Part 4a or Part 4b; you **do not have to do both**. Choose whichever looks more interesting to you, and submit a solution to just one of them. You will not get bonus points for doing both.
- There is an optional **bonus problem**, described in a separate file on the course website.
- **You may not work with anyone else than your assigned partner; you may only ask clarification questions on Piazza; you may not ask questions for help. This is a final exam, not a homework assignment.**

Part 1: Loading and cleaning [10 pts]

The following code loads the `plyr` library (*hint*: it will be very useful for this exam!), and reads in the `census` data frame from http://www.stat.cmu.edu/~ryantibs/statcomp/exams/census.RData. The result has 74020 rows and 31 columns. Each row represents a census tract, and each column a variable that has been measured.

```r
library(plyr)
load(url("http://www.stat.cmu.edu/~ryantibs/statcomp/exams/census.RData"))
dim(census)
```

```
## [1] 74020    31
```

1. How many states are represented among the 74020 census tracts? How many counties? [1 pt]

2. Columns 8 through 31 of the `census` data frame represent numeric variables, but columns 8 and 9 are not stored as such. These two are measured in US dollars: median household income (`Med_HHD_Inc_ACS_09_13`) and median house value (`Med_House_value_ACS_09_13`). What are the classes of these columns? [1 pt]

3. Convert these two columns into numbers (in whole US dollars). (*Hint*: it will be helpful to first convert into strings, then remove any non-numeric characters, then convert into numbers.) For example, `$63,030` should be converted into the integer `63030`. Check your answer by printing out the `summary()` of these two new columns. Make sure that empty entries (`""`) are properly converted to `NA`. [3 pts]

4. Several entries are missing in this data set, including the ones you discovered in the previous question. Compute the number of missing entries in each row, and save the vector as `num.na.row`. Then, obtain the indices of rows containing any missing values and save them in a vector named `contains.na`. What is the average number of missing values among the rows that contain at least one missing entry? [2 pts]

5. Are there any states with no missing values? If so, print out the names of all such states. [1 pt]

6. Redefine the `census` data frame by removing rows that have missing values, as per the `contains.na` vector computed in Part 1 Question 4. Check that the new `census` data frame has now 70877 rows. How many states and counties are represented in this new data frame? What states (if any) have been thrown out compared to the original data frame? [2 pts]

Part 2: Exploratory statistics [10 pts]

For this part, and the remainder of the exam, we will use the cleaned `census` data frame derived from Part 1 Question 6.

1. There are several variables which count the percentage of respondents in various age categories (they all start with `pct_Pop_`). Use these to determine, for each census tract, the percentages of the population that are less than 5 years old, and append these values in a new column called `pct_Pop_0_4_ACS_09_13` to the `census` data frame. Then, use this new column, along with the existing `Tot_Population_ACS_09_13` column, to determine the number of 0-4 year olds in each state. Which state has the highest number, and what is its value? [3 pts]

2. Using your answer from the last question, determine the percentage of 0-4 year olds in each state. Which state has the highest percentage, and what is its value? [1 pt]

3. Calculate the correlation between each of the numeric variables, columns 8 through 31 of the `census` data frame. Which two variables have the highest positive correlation? Which two variables have the lowest negative correlation? Do these relationships make sense? [3 pts]

4. Plot a histogram of `Med_House_value_ACS_09_13`, and label the axes appropriately. See that small bump at the value 1000001? This is a due to a common practice in public data sets of "top-coding", i.e., censoring unusually large values to protect the anonymity of survey respondents. [1 pt]

5. It is possible that the tracts that have been top-coded differ significantly from the other tracts, in other ways than the median house value. The following code computes a t-test between two groups of census tracts, on any given variable. The two groups being compared are: all tracts with median house value equal to the max (1000001) and all tracts with median house value less than the max (<1000001). It then returns a p-value for the test. Note that a lower p-value means a more significant difference between top-coded and non-top-coded tracts, according to the variable in question.

```r
my.test = function(var) {
  group = census$Med_House_value_ACS_09_13 == 1000001
  p.val = t.test(var[group], var[!group])$p.value
  return(p.val)
}
```

Apply the function `my.test()` to the variables in columns 10 through 31 of the `census` data frame. What are the 2 smallest p-values, and which variables do these correspond to? Does this make sense? [2 pts]

Part 3: Sampling and plotting [20 pts]

Plotting data of this size is tricky just because the sheer number of datapoints. For instance, you can try on your own to `plot(census)` to see your computer wheeze, cough, crash and burn. In this problem, we will write a suite of functions that allow you to more efficiently plot relationships between pairs of variables in your dataset.

1. Write a function `plot.sample()` that takes in five arguments: `x`, `y`, `nsample`, `xlab`, and `ylab`. The first two arguments are variables to be plotted. The third is the number of points to be randomly sampled, with a default of 500. (Hence, if `x` and `y` are vectors of length, say, 5000, then a random 500 of the total 5000 x-y pairs will be plotted, by default.) The last two are the x and y labels, both with defaults of `""` (the empty string). A few notes:

- check that `x,y` have the same length, and if not, throw an error (using `stop()` or `stopifnot()`);
- check that the number of requested samples does not exceed the total number of data points, otherwise throw an error;
- the plot should not have a title.

After writing this function, you can try it out by uncommenting the following code below. [5 pts]

```r
#plot.sample(census$Med_HHD_Inc_ACS_09_13, census$Med_House_value_ACS_09_13,
#             xlab="Median HHD income", ylab="Median house value")
```

2. Next, write a function `add.lineartrend.info()`, which is designed to be called after `plot.sample()`. This takes as input `x`, `y` for variables that already appear on the x- and y-axes, and does two things:

- adds the best fit line (produced `lm()`) function in red, and with twice the default width (using the `lwd=2` option);
- adds a title to the plot with the numeric correlation of the two variables, using 3 significant digits.

To reiterate, this function assumes there is a plot already in place; it simply adds the line and the title. Again, after writing this function, you can try it out by uncommenting the following code below. [5 pts]

```
#plot.sample(census$Med_HHD_Inc_ACS_09_13, census$Med_House_value_ACS_09_13,
#            xlab="Median HHD income", ylab="Median house value")
#add.lineartrend.info(census$Med_HHD_Inc_ACS_09_13, census$Med_House_value_ACS_09_13)
```

3. Lastly, write a function `plot.all()` which takes as input `dataset` and `nsample`, a data frame, and the number of points to be sampled. This function will mimick the behavior of `plot()` when applied to data frames. In other words, if `dataset` has `p` columns, then `plot.all()` should produce a `p` by `p` retangular grid of plots, where the plot in entry `i`, `j` of the grid uses column `i` for the x-axis data, and column `j` for the y-axis data. Some notes:

- in each grid entry, the plot should be produced by your previous function, `plot.sample()`;
- in each grid entry, a best fit line should be added and the correlation should be displayed in the title, using your previous function, `add.lineartrend.info()`;
- in the diagonal grid entries, instead of plotting the data (and adding linear trend info), simply use `text()` to write the variable names in the middle of otherwise empty plots.

A code template for `plot.all()` is found below. (*Hint*: using a `for()` loop for the grid entries is perfectly fine.) Fill it out, and then uncomment the code that follows to plot the pairwise relationships between the 4 census variables `Med_HHD_Inc_ACS_09_13`, `Med_House_value_ACS_09_13`, `pct_College_ACS_09_13`, and `Mail_Return_Rate_CEN_2010`. Comment on the results; do the relationships make sense to you? [10 pts]

```
plot.all = function(dataset, nsample=500) {
  p = ncol(dataset)
  orig.par = par()

  # Set the margins, and plotting grid
  par(mar=c(2,2,2,2))
  par(mfrow=c(p,p))

  # TODO: your plotting code goes here

  # Reset the margins, and plotting grid
  par(mar=orig.par$mar)
  par(mfrow=orig.par$mfrow)
}
```

```
#mydat = census[,c("Med_HHD_Inc_ACS_09_13", "Med_House_value_ACS_09_13",
#                  "pct_College_ACS_09_13","Mail_Return_Rate_CEN_2010")]
#plot.all(mydat)
```

Part 4: Option A, permute away! [30 pts]

Here, we will use what is called *permutation test* to answer the following question: are the linear regression coefficients between two different states substantially different? Note: a permutation test is a fairly advanced statistical technique, but **not much statistical knowledge is required** to answer this question. For a bit more (optional) discussion on the permutation test, see the end of this problem.

1. From the `census` data frame, extract all rows that correspond to tracts in Virginia, or West Virginia, and call the resulting data frame `census.vw`. Verify that this has 2321 rows. [1 pt]

2. Now perform two separate linear regressions using `census.vw`. The first is a linear regression of `Mail_Return_Rate_CEN_2010` (as the response) on `pct_NH_White_alone_ACS_09_13` and `pct_Renter_Occp_HU_ACS_09_13` (as the predictors), using only the tracts in Virginia; the second is again a linear regression of `Mail_Return_Rate_CEN_2010` (as the response) on `pct_NH_White_alone_ACS_09_13` and `pct_Renter_Occp_HU_ACS_09_13` (as the predictors), but now using only the tracts in West Virginia. Ignoring the intercept terms, each linear regression model here gives you two coefficients (one for `pct_NH_White_alone_ACS_09_13` and one for `pct_Renter_Occp_HU_ACS_09_13`). Compare the two sets of coefficients. Are they the same? [4 pts]

3. Even though the two coefficients for Virginia and West Virginia may be different, it is hard to tell just *how different* they are. To help answer this question, we will look at the linear regression coefficients if we were to randomly scramble census tracts between Virginia and West Virginia, then refit the linear regression models. This is the idea behind a *permutation test*. Vaguely speaking, if the two sets of coefficients were truly the same, then shuffling the labels should have no real effect on our estimated coefficients. First, make a copy of `census.vw` called `census.vw.perm`. Then, randomly permute the entries in `census.vw.perm$State_name`. (*Hint*: recall the function `sample()`.) Once this is done, rerun the two regression models from Part 4A Question 1 using `census.vw.perm`, and report the coefficients for Virginia, and for West Virginia. What are their values now? Are the differences between coefficients for Virginia and West Virginia much smaller than they were in Part 4A Question 1? [3 pts]

4. You will write a function to encapsulate the tasks you performed in the previous questions. The function will be called `reg.coef()`, and it will take in five arguments:

- `census.subset`, a data frame, whose rows are a subset of the full `census` data frame;
- `x1`, `x2`, two strings that match column names in `census.subset`, representing predictor variables for the regression;
- `y`, another string that matches a column name in `census.subset`, representing a response variable for the regression;
- `shuffle`, a Boolean variable, whose default value is `FALSE`.

Your function should perform the following. If `shuffle` is `TRUE`, then the entries in `census.subset$State_name` are randomly permuted. If `shuffle` is `FALSE`, then this step is not performed (and `census.subset$State_name` is left alone). Next, for each state in the `census.subset` data frame, run a regression of the response, represented by `y`, on predictor variables, represented by `x1` and `x2`. A matrix with 2 columns is returned, and one row for each state in `census.subset`. Each row gives the coefficients for `x1` and `x2` in the linear regression model computed for that particular state (though they are included in the regressions, the intercept terms here are ignored). (*Hint*: for running regressions at a state-by-state level, use `daply()`.) Recreate the results of Part 4A Question 2 with your function, to check that it gives the same answers. Be sure to use `set.seed()` function to enforce reproducibility. [8 pts]

5. Lastly, we will finally implement our permutation test. At a high-level, the idea is to judge differences in coefficients, from (say) Part 4A Question 2 between Virginia and West Virginia, to those in Part 4A Question 3 computed using scrambled data. To make our comparisons to more meaningful, we will repeat the scrambling of state labels (the permutations) multiple times. Write a function `permutation.test()` that takes in the following seven arguments:

- `census`, the census data frame;
- `state1` and `state2`, two strings that represent state names;
- `x1`, `x2`, `y`, as in `reg.coef()`;
- `num.perm`, an integer with a default value of 100.

This function will carry out the following tasks, in order:

- ensure that the columns represented by `x1`, `x2` and `y` are distinct numeric columns of `census`; also make sure that `state1` and `state2` are valid states;
- extract the rows from `census` where each row contains only observations with `State_name` being equal to `state1` or `state2`, and call this smaller data frame `census.subset`;
- run `reg.coef()` from above with arguments `census.subset`, `x1`, `x2` and `y`, and set `shuffle` to `FALSE`; store your results in `obs.coef`;
- run `reg.coef()` as in the last step, but now with `shuffle` set to `TRUE`, and repeat for a total of `num.perm` times; stack the results into one big matrix has 2 columns, and `2*num.perm` rows, called `perm.coef`;
- plot the first column of `perm.coef` versus the second column; there should be `2*num.perm` points on this plot; in addition, draw on top the first column of `obs.coef` versus the second column; this should give you 2 points, and color them in red; also, label the axes according to the names of the variables `x1` and `x2`;
- lastly, return a list containing `obs.coef` and `perm.coef`.

Once written, uncomment the below to run `permutation.test()` on the different scenarios, and comment on the results. In each case, do the red dots lie roughly in the point cloud of black dots? If so, that points to the differences between states being insignificant; and if not, that points to the differences between states being significant. [12 pts]

```
#set.seed(10)
#out1 = permutation.test(census, "Virginia", "West Virginia", "pct_Males_ACS_09_13",
#                        "pct_Pop_5_17_ACS_09_13", "Tot_Population_ACS_09_13")
#out2 = permutation.test(census, "Wisconsin", "Connecticut", "pct_College_ACS_09_13",
#                        "pct_Renter_Occp_HU_ACS_09_13", "Mail_Return_Rate_CEN_2010")
```

Statistical background. For the statistically curious. Note that the permutation test we described is one of many ways to determine how substantially different regression coefficients are. Students who have taken other statistical courses might know that using standard errors is another way. The permutation test is in some sense more robust because it assumes less about the data-generating distribution.

Part 4: Option B, piecewise linear trends [30 pts]

In this problem, we will investigate the relationship between population density and the housing vacancy rate (as measured in `pct_Vacant_Units_ACS_09_13`). On general grounds, we might expect housing markets to operate more efficiently as the demand for housing increases, but we have no reason to expect this relationship to be linear.

1. Construct two data vectors, `PopDensity` and `Vacancy`. From the `census` data frame available to us, `PopDensity` is simply `Tot_Population_ACS_09_13/LAND_AREA`. `Vacancy` is just `pct_Vacant_Units_ACS_09_13`. Exclude rows with zero values in `PopDensity` or `Vacancy` (you should be left with 69336 tracts). [2 pts]

2. You should find a wide range of values for `PopDensity`. What is the ratio of the maximum value to the minimum value (*hint*: it's big). To get a better sense of your data, plot `log(Vacancy)` versus `log(PopDensity)`, using the `plot.sample()` function from Part 3 Question 3, with `nsample=5000`. Make sure the x- and y-axes are labelled appropriately. [2 pts]

3. We will model the relationship between `log(Vacancy)` and `log(PopDensity)` as piecewise linear. Specifically,

$$\log(\text{Vacancy}) \approx \begin{cases} m_1 * [\log(\text{PopDensity}) - \text{BreakPoint}] + b, & \log(\text{PopDensity}) < \text{BreakPoint} \\ m_2 * [\log(\text{PopDensity}) - \text{BreakPoint}] + b, & \log(\text{PopDensity}) \geq \text{BreakPoint} \end{cases}$$

The model has four parameters: the break point itself, the value of the slope on either side of the break (m_1 and m_2), and the value of the model at the break (b). Note that we have required that the model be continuous at the break point.

Write a function `break.model()` that implements the piecewise linear transformation above, given five arguments:

- `x`, a vector of values to be transformed (we have in mind `log(PopDensity)`);
- `break.point`, the break point;
- `m1`, `m2`, and `b`, the three regression parameters. The function should return a vector of values, following the piecewise linear transformation described above. [4 pts]

4. Now your goal is to fit the model from the last question to the `log(Vacancy)`, `log(PopDensity)` data. Because it is not a simple linear model, we will not use `lm()`. Instead, write a loss function that returns the sum of squares of the differences between the predictions from the model (for given values of the break point, m_1, m_2, and b) and the values observed in the data. This function should be named `break.loss()`, and it should take three arguments:

- `params`, a vector of length 4, containing the break point, m_1, m_2, and b, in that order;
- A vector `x.data` with default value `log(PopDensity)`;
- A vector `y.data` with default value `log(Vacancy)`. Evaluate `break.loss` for `params` with the break point at 5.5, $m_1 = -0.25$, $m_2 = 0$, and $b = 2$. You should get approximately 42500. [4 pts]

5. Next, use `optim()` to minimize `break.loss()`, over the `log(Vacancy)`, `log(PopDensity)` data. Start at the initial parameter value `par=c(5.5,-0.25,0,2)`. What are the best-fit values for the break point, m_1, m_2, and b? How many function evaluations did `optim()` use before convergence? Plot your data as in Part 4B Question 2, then plot your best-fit model—note this is a piecewise linear function—over the top in red, with double the usual line width. Does the model look plausible? How would you describe the behavior of the model below and above the break point? What is the value of the population density at your break point? What kind of community (rural, suburban, etc.) corresponds to this population density? [8 pts]

6. For a variety of reasons we would like to know how well-determined the break point and other parameters are. For example, is the break point well constrained? Are we confident that the slope actually changes at the break point? Is the slope above the break point consistent with zero? We will use the jackknife method to estimate the uncertainty in the break point and the other parameters.

In fact, your instructors are so eager to know this information that they have provided code to make a jackknife estimate of the standard errors in the break point and other coefficients. They found it impractical to run `optim()` 69336 times, and produced their jackknife estimates by leaving out a subset of the observations, rather than leaving out each one in turn. However, in their enthusiasm they produced code with a number of errors. Debug the following, use it to estimate the uncertainty in the break point and other model coefficients, and discuss the results. (Note: we have set `eval=FALSE` because this code block, even once properly debugged, takes a long time to run, several minutes ... as you work to debug it, you may want to temporarily set `n.leaveout` to be a much smaller number ...) [10 pts]

```r
n.leaveout = 100
set.seed(0)
to.leaveout = sample(1:length(Vacancy),n.leaveout,replace=FALSE)
par.mat = matrix(rep(0,4*n.leaveout),nrow=3,ncol=n.leaveout)
for( i in 1:n.leaveout)
  j = to.leaveout[i]
  this.opt = optim(c(5.5,-0.25,0,2),break.loss,x.data=log(PopDensity)[-i],
                   y.data=Vacancy[-i])
  par.mat[i] = this.opt$par
}
jack.sd = apply(par.mat,2,sd) * sqrt(length(Vacancy))
jack.sd
```