University of California, Santa Cruz
Department of Applied Mathematics and Statistics
Baskin School of Engineering

AMS 206 (Applied Bayesian Statistics)

Take-Home Test 3 (final draft)

Drop-dead due date **25 Mar 2018**

Here are the ground rules: this test is open-book and open-notes, and consists of two problems (true/false and calculation); **each of the 7 true/false questions is worth 10 points, and the calculation problem is worth 204 total points, for a total of 274 points, plus the possibility of up to 28 extra-credit points**.

The right answer with no reasoning to support it, or incorrect reasoning, will get **half credit**, so try to make a serious effort on each part of each problem (this will ensure you at least half credit). In an AMS graduate class I taught in 2012, on a take-home test like this one there were 15 true/false questions, worth a total of 150 points; one student got a score of 92 out of 150 (61%, a D−, in a graduate class where B− is the lowest passing grade) on that part of the test, for repeatedly answering just "true" or "false" with no explanation. Don't let that happen to you.

On non-extra-credit problems, I mentally start everybody out at −0 (i.e., with a perfect score), and then you accumulate negative points for incorrect answers and/or reasoning, or parts of problems left blank. On extra-credit problems, the usual outcome is that you go forward (in the sense that your overall score goes up) or you at least stay level, but please note that it's also possible to go backwards on such problems (e.g., if you accumulate +3 for part of an extra-credit problem but −4 for the rest of it, for saying or doing something egregiously wrong).

This test is to be entirely your own efforts; do not collaborate with anyone or get help from anyone but me or our TA (Daniel Kirsner). The intent is that the course lecture notes and readings should be sufficient to provide you with all the guidance you need to solve the problems posed below, but you may use other written materials (e.g., the web, journal articles, and books other than those already mentioned in the readings), **provided that you cite your sources thoroughly and accurately**; you will lose (substantial) credit for, e.g., lifting blocks of text directly from `wikipedia` and inserting them into your solutions without full attribution.

If it's clear that (for example) two people have worked together on a part of a problem that's worth 20 points, and each answer would have earned 16 points if it had not arisen from a collaboration, then each person will receive 8 of the 16 points collectively earned (for a total score of 8 out of 20), and I reserve the right to impose additional penalties at my discretion. If you solve a problem on your own and then share your solution with anyone else (because people from your cultural background routinely do this, or out of pity, or kindness, or whatever motive you may believe you have; it doesn't matter), you're just as guilty of illegal collaboration as the person who took your solution from you, and both of you will receive the same penalty. This sort of thing is necessary on behalf of the many people who do not cheat, to ensure that their scores are meaningfully earned. In the AMS graduate class in 2012 mentioned above, five people failed the class because of illegal collaboration; don't let that happen to you.

In class I've demonstrated numerical work in `R`; you can (of course) make the calculations and plots requested in the problems below in any environment you prefer (e.g., `Matlab`, ...).

Please collect {all of the code you used in answering the questions below} into an Appendix at the end of your document, so that (if you do something wrong) the grader can better give you part credit. To avoid plagiarism, if you end up using any of the code I post on the course web page or generate during office hours, at the beginning of your Appendix you can say something like the following:

> *I used some of Professor Draper's* `R` *code in this assignment, adapting it as needed.*

Last ground rule: proof by `Maple` or some other symbolic computing package is not acceptable; when I ask You to show something, please do so by hand (You can check Your results with (e.g.) `Maple`, but You need to do the work Yourself).

1 True/False

[70 total points: 10 points each] For each statement below, say whether it's true or false; if true without further assumptions, briefly explain why it's true (and — *extra credit* — what its implications are for statistical inference); if it's sometimes true, give the extra conditions necessary to make it true; if it's false, briefly explain how to change it so that it's true and/or give an example of why it's false. If the statement consists of two or more sub-statements and two or more of them are false, you need to explicitly address all of the false sub-statements in your answer.

In answering these questions you may find it helpful to consult the following references, available on the course web page: DS (Degroot and Schervish (2012)) sections 3.10, 12.5, 12.6; Gelman et al. (2014) Chapter 11.

(A) If you can figure out how to do IID sampling from the posterior distribution of interest to you, this will often be more Monte-Carlo efficient than MCMC sampling from the same posterior.

(B) A (first-order) Markov chain is a particularly simple stochastic process: to simulate where the chain goes next, you only need to know (i) where it is now and (ii) where it was one iteration ago.

(C) The bootstrap is a frequentist simulation-based computational method that can be used to create approximate confidence intervals for population summaries even when the population distribution of the outcome variable y of interest is not known; for example, if all you know from problem context is that your observations $\boldsymbol{y} = (y_1, \ldots, y_n)$ are IID from *some* distribution with finite mean μ and finite SD σ, you can use the bootstrap to build an approximate confidence interval for μ even though you don't know what the population distribution is.

(D) In MCMC sampling from a posterior distribution, you have to be really careful to use a monitoring period of just the right length, because if the monitoring goes on for too long the Markov chain may drift out of equilibrium.

(E) Simulation-based computational methods are needed in Bayesian data science (inference, prediction and decision-making) because conjugate priors don't always exist and high-dimensional probability distributions are difficult to summarize algebraically.

(F) In MCMC sampling from a posterior distribution, you have to be really careful to use a burn-in period of just the right length, because if the burn-in goes on for too long the Markov chain will have missed its chance to find the equilibrium distribution.

(G) You're MCMC sampling from a posterior distribution for a vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$. During the monitoring period, the column in the MCMC data set for a component of θ (θ_j, say) behaves like an autoregressive time series of order 1 (AR_1) with estimated first-order autocorrelation $\hat{\rho}_j = 0.992$. This means that You'll need to monitor this component of θ for about 16 times more iterations than would have been necessary if You could have figured out how to do IID sampling on θ_j.

2 Calculation

(A) *[126 total points for this problem, plus up to 8 extra credit points]* The data set

`U.S.-family-income-2009.txt`

on the course web site contains family income values $\boldsymbol{y} = (y_1, \ldots, y_n)$ from the year 2009 (in units of \$1,000, and sorted from smallest to largest) for a random sample of $n = 842$ American households with incomes under \$1 million, obtained from the Current Population Survey conducted by the *U.S. Census Bureau*. Download this text file to a directory of Your choice, start an `R` session, and change directory inside `R` to the place You put the raw data file. Once You've done that, You can read the data file in and put it into a vector called `y` (say) with the command

```
y <- scan( 'U.S.-family-income-2009.txt' )
```

`R` will say `Read 842 items`, verifying that the entire data vector was read in correctly.

The point of this problem is to draw inferences about the mean family income θ in the population $\mathcal{P}$ of all U.S. households (with incomes under \$1 million) in 2009. In estimating θ we'll look at two frequentist-and-Bayesian parametric methods and two frequentist nonparametric approaches (one of which [the bootstrap] is also a Bayesian nonparametric method, as discussed in class); the meta-goal of this problem is to experience statistical model uncertainty and to begin to learn how to cope with it.

(a) *[12 total points for this sub-problem]* Make a histogram of these income values (on the density scale) with a large number of bars (e.g., 100); briefly comment on its shape, and visually estimate the mode *[6 points]*. Compute the sample mean, median, SD, and smallest and largest values, and compare the mean, median and mode; are they related to each other in the way you would expect them to be, given the distributional shape? Explain briefly. *[6 points]*

Two common parametric sampling models for long-right-hand-tailed variables such as family income are based on the Lognormal and Gamma distributions. Consider first the model

$$\left\{ \begin{array}{c} (Y_i \,|\, \mu\, \sigma^2\, \mathcal{L}\, \mathcal{B}) \stackrel{\text{IID}}{\sim} \text{Lognormal}(\mu, \sigma^2) \\ (i = 1, \ldots, n) \end{array} \right\} ; \tag{1}$$

here $Y_i \sim \text{Lognormal}(\mu, \sigma^2)$ simply means that $W_i \triangleq \log Y_i \sim N(\mu, \sigma^2)$ and $\mathcal{L}$ denotes the Lognormal sampling distribution assumption, which is not a part of $\mathcal{B}$. (You can see that Lognormal is a bad name for this distribution: if $W_i = \log Y_i$ is Normal then $Y_i = e^{W_i}$, so it should really be called the Exponential Normal model, but this name has been in use for more than 100 years and we're stuck with it.)

(b) *[8 total points for this sub-problem]* Create a new vector in `R` called `w` (say) that contains the $\log(y)$ values; calculate the mean and variance of `w`, and make a histogram of `w` (on the density scale) with a large number of bars (e.g., 100). If the Lognormal sampling model is reasonable for this data set, Your histogram should (of course) look a lot like the Normal curve; does it? *[4 points]* You can pin this down even more precisely by making a Normal qqplot of `w` with the `R` commands

```
qqnorm( w ) ; abline( mean( w ), sd( w ) )
```

This plot is supposed to look like a straight line if the data values do indeed behave like a random sample from a Normal distribution; does it? Explain briefly *[4 points]*

(c) *[10 total points for this sub-problem, plus up to 8 extra credit points]* It can be shown *(Extra credit [8 points]: show these two facts)* that in this model

(i) the repeated-sampling density of Y_i is

$$p(y_i \,|\, \mu\, \sigma^2\, \mathcal{L}\, \mathcal{B}) = \frac{1}{\sigma\, y_i \sqrt{2\,\pi}} \exp\left[-\frac{(\log y_i - \mu)^2}{2\,\sigma^2} \right] \tag{2}$$

for $y_i > 0$ (and 0 otherwise) and

(ii) $E_{RS}(Y_i \,|\, \mu\, \sigma^2\, \mathcal{L}\, \mathcal{B}) = \exp\left(\mu + \frac{\sigma^2}{2}\right)$, meaning that if we adopt sampling model (1) then $\theta = \exp\left(\mu + \frac{\sigma^2}{2}\right)$.

The definition of the Lognormal distribution — $W_i \triangleq \log Y_i \sim N(\mu, \sigma^2)$ — and fact (i) suggest two different ways to obtain the MLEs $(\hat{\mu}_{MLE}, \hat{\sigma}^2_{MLE})$ for μ and σ^2 in model (1). Describe both of these ways to get the ML estimators; identify which method is easier, and briefly explain why; and use the easier method to get the MLEs *[6 points]*. Plot your histogram of the data set on the density scale from part (a) again, and superimpose a density trace of the Lognormal$(\hat{\mu}_{MLE}, \hat{\sigma}^2_{MLE})$ distribution; does this fit look acceptable? Explain briefly *[4 points]*. (*Computing Hint:* You can either write your own Lognormal density function (from equation (2)) or gain access to the Lognormal density function `dlnorm` in `R` (and the information about it available with the `help` command) by first issuing the command `library( stats )`.)

(d) *[22 total points for this sub-problem]* Use `RJAGS` to fit, via MCMC, a Bayesian version of the Lognormal model to the income data with a low-information-content prior; the code for doing this is on the course web page in a file called

```
ams-206-income-lognormal-rjags-analysis-R.txt
```

(use Your own personal random number seed when You run this code) and the model that's called by the `RJAGS` code is also on the course web page, in a file called

```
ams-206-income-lognormal-analysis-rjags-model.txt
```

[4 points]. It turns out that to get DIC values for Bayesian models in `RJAGS`, you need to run $A = 2$ or more parallel Markov chains with different random number streams, so when you ask for m monitoring iterations in `RJAGS` with $A = 2$ you're actually getting $2\,m$ iterations, which will take a bit longer; I recommend a burn-in of 1,000 iterations from the starting values I've given you, followed by a monitoring run of $m = 100{,}000$ iterations (on my desktop at home this only took about 3 minutes). Compare the posterior mean values for μ and σ^2 with their corresponding MLEs, and comment briefly on whether the comparison came out as you expected it to *[4 points]*. Summarize the posterior for θ by extracting its posterior mean, SD and 95% interval from the MCMC output *[4 points]*. Compare the mean and SD

and several quantiles of the predictive distribution for a new y value with the corresponding quantities from the income data set, and make a *qqplot* of your random draws from the predictive distribution against the data set; does the model seem to make good predictions? Explain briefly *[8 points]*. Identify the DIC value for the Lognormal model in Your output *[2 points]*.

Now instead consider the Gamma sampling model (for $\alpha > 0, \beta > 0$)

$$\left\{ \begin{array}{c} (Y_i \,|\, \alpha\, \beta\, \mathbf{\Gamma}\, \mathcal{B}) \overset{\text{IID}}{\sim} \Gamma(\alpha, \beta) \\ (i = 1, \ldots, n) \end{array} \right\} \quad \longleftrightarrow \quad p(y_i \,|\, \alpha\, \beta\, \mathbf{\Gamma}\, \mathcal{B}) = \left\{ \begin{array}{cc} \frac{\beta^\alpha}{\Gamma(\alpha)}\, y_i^{\alpha-1}\, e^{-\beta\, y_i} & \text{for } y_i > 0 \\ 0 & \text{otherwise} \end{array} \right\}, \quad (3)$$

using the parameterization in which $E_{RS}(Y_i \,|\, \alpha\, \beta\, \mathbf{\Gamma}\, \mathcal{B}) = \frac{\alpha}{\beta}$, meaning that if we adopt sampling model (3) then $\theta = \frac{\alpha}{\beta}$; here $\mathbf{\Gamma}$ denotes the Gamma sampling distribution assumption, which is not part of $\mathcal{B}$. As usual let $\boldsymbol{y} = (y_1, \ldots, y_n)$ and $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$; also set $\overline{\log y} \triangleq \frac{1}{n} \sum_{i=1}^n \log y_i$.

(e) *[18 total points for this sub-problem]* Show that the log-likelihood function in this model is

$$\ell\ell(\alpha\, \beta \,|\, \boldsymbol{y}\, \mathbf{\Gamma}\, \mathcal{B}) = n\, \alpha\, \log \beta - n\, \log \Gamma(\alpha) + (\alpha - 1) \sum_{i=1}^n \log y_i - \beta \sum_{i=1}^n y_i\,; \quad (4)$$

conclude therefore that $\left(\bar{y}, \overline{\log y}\right)$ is a 2–dimensional set of sufficient statistics for (α, β) and that the MLEs have to satisfy the relation $\frac{\hat{\alpha}}{\hat{\beta}} = \bar{y}$ *[8 points]*. Even so, briefly explain why equation (4) means that no closed-form (algebraic) expressions for the MLEs for α and β are possible in this model; numerical methods are needed to maximize (4) *[4 points]*. Using the `optim` function exactly as in Take-Home Test 2, it turns out with this data set (You don't need to show this) that $(\hat{\alpha}_{MLE}, \hat{\beta}_{MLE}) \doteq (1.305, 0.01575)$. Plot your histogram of the data set on the density scale from part (a) yet again, and superimpose a density trace of the $\Gamma(\hat{\alpha}_{MLE}, \hat{\beta}_{MLE})$ distribution; does *this* fit look acceptable? Explain briefly *[6 points]*.

(f) *[22 total points for this sub-problem]* Now use `RJAGS` to fit, via MCMC, a Bayesian version of the Gamma model to the income data with a low-information-content prior; the code for doing this is on the course web page in a file called

`ams-206-income-gamma-rjags-analysis-R.txt`

(use Your own personal random number seed when You run this code) and the model that's called by the `RJAGS` code is also on the course web page, in a file called

`ams-206-income-gamma-analysis-rjags-model.txt`

[4 points]. This model takes longer to fit (this fact is related to the difficulty you saw in part (d) with maximum-likelihood fitting), so you may wish to settle for a smaller value of m than you did with the Lognormal model (on my desktop at home, 100,000 monitoring iterations took about 12 minutes; you can adjust your value of m based on the speed of your machine and your patience, but keep it at or above 20,000). Compare the posterior mean values for α and β with their corresponding MLEs, and comment briefly on whether the comparison came out as you expected it to *[4 points]*. As you did in the Lognormal case, summarize the posterior for θ by extracting its posterior mean, SD and 95% interval from the MCMC output *[4 points]*. Compare the mean and SD and several quantiles of the predictive distribution for a new y value with the corresponding quantities from the income data set, and make a qqplot of your random draws from the predictive distribution against the data set; does *this* model seem to make good predictions? Explain briefly *[8 points]*. Identify the DIC value for the Gamma model in Your output *[2 points]*.

(g) *[4 total points for this sub-problem]* According to DIC, which of the Lognormal or Gamma models is better? *[2 points]* Does the fact that DIC says model M_1 is better than model M_2 mean that model M_1 is good enough to stop looking for a better model? Explain briefly *[2 points]*.

(h) *[4 total points for this sub-problem]* Since interest focuses on the population mean θ, one possible estimator of θ is the sample mean $\bar{Y}$. Use the Central Limit Theorem (CLT) to construct an approximate 95% confidence interval for θ based on $\bar{y}$ *[4 points]*.

Notice that the interval you created in part (h) is based on a frequentist *nonparametric* method: in building your interval you made no assumptions about the form of the sampling distribution (apart from the assumption built into the CLT that the population SD is finite, which is certainly true for income here). **Question:** How well calibrated is the CLT interval? By this I mean the usual notion of validity/calibration of 95% intervals: if the process leading to your interval in part (h) is repeated many times, do the resulting intervals include the true parameter value θ approximately 95% of the time?

In answering this question it looks like we need to know what the truth is about θ. Since we don't know θ, it turns out that the best we can do — in estimating the calibration of the CLT interval process — is to use the bootstrap idea we talked about in class: we can take random samples from our *sample* and use these as proxies for what we would get if we were able to take a new random sample of $n = 842$ households from the population $\mathcal{P}$. In this bootstrapping, we'll pretend that the true θ is the sample mean $\bar{y} \doteq 82.88$ you calculated in part (a).

I've written `R` code for You, posted on the course web page as

`ams-206-income-bootstrapping-sample-mean.txt`

This code repeatedly (m = 100,000 times, say) draws samples of size $n = 842$ at random with replacement from the data set `U.S.-family-income-2009.txt`, computes the CLT-based 95% interval each time, and works out the proportion of these intervals that include the actual sample mean. To get ready for parts (i) and (j), the code also keeps track of the m simulated sample means $\bar{y}^*$ that were calculated in assessing calibration of the CLT interval-building process. **Run this program for Yourself, and have the results ready in answering questions (i) and (j).**

(i) *[8 total points for this sub-problem]* The CLT method assumes that $n = 842$ is large enough to produce a Normal sampling distribution for $\bar{Y}$. Check this assumption by making a normal qqplot of the $\bar{y}^*$ values you generated in part (g) — has the CLT done its magic with $n = 842$, even though the population distribution had a quite long right-hand tail? Explain briefly *[4 points]*. When You ran my code, what was the *actual* coverage of the allegedly 95% CLT confidence intervals? Do You regard this as close enough to 95% to describe the frequentist nonparametric CLT approach as well calibrated in this problem? Explain briefly *[4 points]*.

(j) *[8 total points for this sub-problem]* In producing the bootstrap distribution of the sample means $\bar{y}^*$ in part (i), you've also done all of the necessary work to calculate the bootstrap estimate of θ, its bootstrap standard error, and the 95% bootstrap confidence interval for θ; compute these quantities. How does this bootstrap CI compare with the CLT-based CI? Explain briefly. *[8 points]*

(k) *[10 total points for this sub-problem]* Summarize all of this by completing Table 1; a dash (—) in an entry means that You don't need to fill it in *[6 points]*. Given Your conclusions

Table 1: *Summary of inference about θ across six models/methods; ML = maximum likelihood.*

	θ		
Method/Model	Mean/ Estimate	SD/ SE	95% Interval
Lognormal ML	·	—	—
Lognormal Bayes	·	·	(· , ·)
Gamma ML	·	—	—
Gamma Bayes	·	·	(· , ·)
CLT	·	·	(· , ·)
Bootstrap	·	·	(· , ·)

about model fit and validity of method, which numbers in Table 1 seem most reliable to you? What do you conclude, both about mean income in the U.S. in 2009 and about the challenge of model uncertainty? *[4 points]*

(l) *(Extra Credit [10 points])* It's also possible, although more time consuming, to keep track of the calibration of the Lognormal and Gamma posterior intervals for θ. I've written `R` code for You, posted on the course web page as

`ams-206-income-bootstrapping-Lognormal-and-Gamma-intervals.txt`

This code repeatedly ($m = 100$ to 1,000 times, say, depending on Your patience in waiting for the results) draws bootstrap samples of size $n = 842$ at random with replacement from the data set `U.S.-family-income-2009.txt`, uses `RJAGS` to compute the Lognormal and Gamma intervals for θ with the bootstrap data sets, and keeps track of how often each of those intervals includes the actual sample mean of the original data set. Run my code **with Your own random number seeds** and report the actual coverage of the allegedly 95% Bayesian intervals, commenting on the extent to which parametric modeling with sampling distributions that don't fit the data well is capable of producing well-calibrated inferences.

(B) *[78 total points for this problem, plus up to 20 extra credit points]* As I'm sure You know, if You encounter a wild mushroom in a forest there's no guarantee that it's edible; every year several people die in the U.S. from wild mushroom poisoning. Two questions come to mind, in this age of cell phone apps: (1) Can the edible/poisonous status of a wild mushroom be accurately predicted from characteristics such as its appearance and odor? and (2) If You were building an app to give people advice about whether a wild mushroom they've found is edible, (to make the app easy to use) what's the minimum number of variables necessary to get highly accurate predictions?

The *U.C. Irvine Machine Learning Repository* has a data set – a copy of which is now on the AMS 206 web page, along with a text file containing important contextual information – consisting of $n = 8{,}124$

> hypothetical samples corresponding to 23 species of gilled mushrooms in the *Agaricus* and *Lepiota* Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The *Audubon Society Field Guide to North American Mushrooms* (1981) clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like "leaflets three, let it be" for Poisonous Oak and Ivy.

As You'll see when You begin looking at the data set, there are $k = 22$ predictor variables $(x_1, \ldots, x_k)$ available, ranging from aspects of the mushroom's cap to its habitat, and the outcome

variable y is coded 1 for poisonous and 0 for edible. The goals of this problem, corresponding to the two questions above, are (1) to build linear and logistic regression models, using these predictors, to produce estimated probabilities $\hat{p}$ that $(y = 1)$ as a function of a given mushroom's characteristics, (2) to identify the smallest subset of the x_j (for inclusion in the app) that still produces highly accurate $\hat{p}$ values, and (3) to decide whether the predictive system is accurate enough to release the app to the general public without poisoning a lot of people in the process.

When You examine the set of predictor variables, You'll see that they're all categorical (`R` calls such variables *factors*), taking on a number of possible values (*levels*) ranging from 1 to 12. **Important: all of the levels of all of the predictors in the data set have been abbreviated to a single letter in the Roman alphabet; the context file contains a dictionary that translates those abbreviations to their actual meanings.** Obviously any predictor variable that takes on only 1 possible value is useless for predicting anything, so early on in the analysis we'll drop this variable (`veil.type`) and reset k to 21. One variable – `stalk.root` – has a lot of missing values (2,480 out of 8,124), but one nice thing about categorical predictors is that *missingness can be treated as just another level of the factor*, so that no cases are lost by having to omit rows in the data set with missing values in them (an undesirable action that's not needed with factor predictors).

As discussed in class, the basic frequentist linear regression model is of the form (for $i = 1, \ldots, n$)

$$y_i = \beta_0 + \sum_{j=1}^{k} x_{ij}\, \beta_j + e_i\,, \tag{5}$$

in which the $(e_i \,|\, \sigma)$ are IID $N(0, \sigma^2)$; in class we also saw that this model can be written in matrix form as

$$\boldsymbol{y} = X\, \boldsymbol{\beta} + \boldsymbol{e}\,, \tag{6}$$

where $\boldsymbol{y}$ is the $(n \times 1)$ vector whose transpose is $(y_1, \ldots, y_n)$, X is the $[n \times (k+1)]$ matrix whose first column is a vector of 1s (to account for the intercept term β_0) and whose ith row is $(1, x_{i1}, \ldots, x_{ik})$, $\boldsymbol{\beta}$ is the $[(k+1) \times 1]$ vector whose transpose is $(\beta_0, \beta_1, \ldots, \beta_k)$ and $\boldsymbol{e}$ is the $(n \times 1)$ vector whose transpose is $(e_1, \ldots, e_n)$.

In applying this model to the mushroom data, a new question immediately arises: how can You bring a categorical predictor – such as `ring.number`, with the 3 levels "n" (none), "o" (one) and "t" (two) – into a regression model? The answer is with a set of *indicator*, also known as *dummy*, variables: with x_{16} = `ring.number` as an example, having $\ell = 3$ levels, You create a new variable z_1 that's 1 if x_{16} = "n" and 0 otherwise, and another new variable z_2 that's 1 if x_{16} = "o" and 0 otherwise, and a third new variable z_3 that's 1 if x_{16} = "t" and 0 otherwise. If You now include all $\ell = 3$ of the z_j in the set of predictors, in place of x_{16}, You will have created what's called a *collinearity* problem: by the nature of how the z_j were defined, for every mushroom i in the data set it's a fact that $z_{i1} + z_{i2} + z_{i3} = 1$. This makes the X matrix in equation (6) non-invertible, meaning that the computation of the maximum-likelihood estimate of $\boldsymbol{\beta}$, namely $\hat{\boldsymbol{\beta}} = (X^T X)^{-1}\, X^T\, \boldsymbol{y}$, would be more difficult to carry out. The (simple) solution is to omit one of the z variables in the set of z_j You include in the modeling: after all, in the `ring.number` example, if You knew z_{i1} and z_{i2}, $z_{i3} = (1 - z_{i1} - z_{i2})$ would be redundant (in the jargon of regression modeling, the category whose z dummy has been left out is called the *omitted group*). Letting ℓ_j be the number of levels of categorical predictor x_j and setting $L = \sum_{j=1}^{k} \ell_j$, the new linear regression model, expressed in terms of the dummy variables z, is

$$\begin{aligned} y_i \;=\; & \beta_0 + [\beta_1\, z_{i1} + \cdots + \beta_{\ell_1 - 1}\, z_{i,\ell_1 - 1}] + [\beta_{\ell_1}\, z_{i2} + \cdots + \beta_{\ell_1 + \ell_2 - 2}\, z_{i,\ell_1 + \ell_2 - 2}] \\ & + \cdots + \left[\beta_{L-K-(\ell_k - 2)}\, z_{i,L-K-(\ell_k-2)} + \cdots + \beta_{L-k}\, z_{i,L-k}\right] + e_i\,. \end{aligned} \tag{7}$$

This looks nasty but isn't: original categorical variable (factor) x_1 is replaced by $(\ell_1 - 1)$ dummy variables, original factor x_2 is replaced by $(\ell_2 - 1)$ dummies, and so on up to original factor x_k being replaced by $(\ell_k - 1)$ dummies, for a total of $k^* = (L - k)$ dummy variables replacing the original k factors. In the mushroom data set there are $k = 21$ non-trivial factors as predictor variables, and the total number of dummies needed to carry this information is $[(6+4+10+2+9+2+2+2+12+2+5+4+4+9+9+4+3+5+9+6+7) - 21] = 95$. (Where did I get the numbers $(6 + 4 + \cdots + 7)$?)

(a) *[4 total points for this sub-problem]* I've written some `R` code for You, to start You on the analysis of this data set; it's in a file on the course web page called

`ams-206-mushroom-partial-data-analysis.txt`

There's a block of code at the top of the file that begins `'the first block of code starts here'` and ends `'the first block of code ends here'`; run this code block and study the output. The function `tab.sum` in this code block provides diagnostic information on whether a factor x will turn out to be predictively useful in the modeling; briefly explain in what sense `tab.sum` provides such information (*Hint:* the function estimates the conditional mean (and SD, not useful here) of what variable given what other variable?) *[4 points]*.

(b) *[8 total points for this sub-problem]* Run the second code block, in which a linear regression model is fit with the dichotomous outcome `poisonous` regressed on the factor `cap.shape`, and study the output. When the predictions $\hat{y}$ from equation (7) are specialized to this regression, they looks like

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 \, z_{i1} + \cdots + \hat{\beta}_5 \, z_{i5} \, , \tag{8}$$

in which the $\hat{\beta}_j$ are the maximum-likelihood estimates of the regression coefficients and where $\{z_{i1} = 1$ if `cap.shape` = 'c' and 0 otherwise$\}$, $\{z_{i2} = 1$ if `cap.shape` = 'f' and 0 otherwise$\}$, and so on down to $\{z_{i5} = 1$ if `cap.shape` = 'x' and 0 otherwise$\}$. Now examine the extracts from the `tab.sum` and regression fitting in Table 2 below. Explicitly identify $(\hat{\beta}_0, \ldots, \hat{\beta}_5)$ in the output in the table *[4 points]*, and – by thinking about the form of equation (8) for each of the levels of `cap.shape` – explicitly relate the numbers in the `mean` column of the `tab.sum` output in Table 2 to the $\hat{\beta}_j$ *[4 points]*.

(c) *[8 total points for this sub-problem]* Toward the end of the second code block, the code computes predicted $\hat{p} = P(y = 1 \,|\, \mathbb{L} \, x_1 \, \mathcal{B})$ values (in which $\mathbb{L}$ signifies the linear regression modeling assumption, which is not part of $\mathcal{B}$), makes a diagnostic plot and computes a numerical diagnostic – the *Predictive Separation Index (PSI)* – measuring the predictive strength of the factor `cap.shape`.

(i) *[4 total points for this sub-problem]* The diagnostic plot is in two parts: the top panel is a histogram of the $\hat{p}$ values for the mushrooms for which $y = 0$, and the bottom panel shows the same thing except for the cases in which $y = 1$. What would the ideal shape of these histograms be, if a factor x has extremely strong information for predicting a dichotomous outcome y? Explain briefly *[2 points]*. Do the histograms achieve that goal with the predictor `cap.shape`? Explain briefly *[2 points]*.

(ii) *[4 total points for this sub-problem]* The PSI, which is a numerical index that goes hand-in-hand with the diagnostic plot, is defined as follows:

$$PSI(x) = [\text{mean } (\hat{p} \text{ given } x) \text{ when } y = 1] - [\text{mean } (\hat{p} \text{ given } x) \text{ when } y = 0] \, . \tag{9}$$

What's the ideal value of the PSI, if a factor x is highly predictive of y? Explain briefly *[2 points]*. Does the PSI come close to achieving that goal with `cap.shape`? Explain briefly *[2 points]*.

Table 2: *Extracts from the output of the second code block.*

```
#      cap.shape n    mean      sd

# [1,] 1         452  0.1061947 0.308428
# [2,] 2         4    1         0
# [3,] 3         3152 0.4936548 0.5000391      output of tab.sum
# [4,] 4         828  0.7246377 0.4469667
# [5,] 5         32   0         0
# [6,] 6         3656 0.4671772 0.4989898

# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)  0.10619    0.02279   4.659 3.22e-06 ***
# cap.shapec   0.89381    0.24335   3.673 0.000241 ***   output of
# cap.shapef   0.38746    0.02437  15.898  < 2e-16 ***   linear
# cap.shapek   0.61844    0.02834  21.824  < 2e-16 ***   regression
# cap.shapes  -0.10619    0.08864  -1.198 0.230926
# cap.shapex   0.36098    0.02416  14.942  < 2e-16 ***
```

Table 3: *Predictive accuracy of each of the factors x in the mushroom data set, with PSI sorted from largest to smallest.*

Factor (x)	PSI	Predictive Power
⋮	⋮	⋮
`cap.shape`	0.0603	weak
`cap.surface`	0.0388	weak
⋮	⋮	⋮

(d) *[18 total points for this sub-problem]* Run the third code block and study the output. I've written a function called `univariate.exploration` that automates the process of repeating the first and second code blocks; run this function with each of the other 20 categorical predictors (save `odor` for last, for reasons that will become clear); in each case, pay particular attention to the table created by `tab.sum`, the diagnostic plot and the PSI value. Summarize Your findings by completing Table 3; I suggest that You use the phrases *extremely strong*, *strong*, *moderate*, *weak*, and *almost none* to describe the predictive power of each x variable *[16 points]*. If You were going to base the app on only one or two predictors, which ones look like the best candidates? Explain briefly *[2 points]*.

(e) *[6 total points for this sub-problem]* In the output from code block 3, the PSI for `cap.surface` came out 0.03877928, which we could round to 0.03878. That number appears somewhere else in the regression output; where? Read pages 748–749 in DeGroot and Schervish (2012), available on the course web page; based on Your reading of these pages, briefly explain what the number in the regression output is trying to measure *[2 points]*. Does it make sense that the PSI and this number are closely related? (This relation only holds for regressions with a dichotomous outcome; if y is continuous, the PSI doesn't make sense.) *[2 points]* Check several other sets of output from the `univariate.exploration` function with different

predictors; is the relation between the PSI and the number in the regression output always the same? *[2 points]*

Run the fourth code block, in which a linear regression model is fit with all available predictors (let's call this the *full model* $\mathbb{F}$), and study the output. You can see that `R` has a convenient way (`poisonous` $\sim$ `.`) to specify all of the predictors without having to name all of them. You can further see that prediction of the poisonous status of all $n = 8{,}124$ mushrooms using the full model is perfect: all of the truly poisonous mushrooms have estimated $P(y = 1 \,|\, \mathbb{L}\,\mathbb{F}\,\mathcal{B}) = 1$, and all of the truly edible mushrooms have estimated $P(y = 1 \,|\, \mathbb{L}\,\mathbb{F}\,\mathcal{B}) = 0$.

However, this evaluation of the predictive quality of $\mathbb{F}$ may overstate its accuracy, because we used the same (entire) data set both to fit $\mathbb{F}$ and then to see how good $\mathbb{F}$ is. As mentioned in class, *cross-validation (CV)* is a good way to check on the extent of any over-fitting: You partition the data set at random into non-overlapping subsets, fit the model on one subset, and evaluate the quality of the fit on another. A well-established CV method is called *s–fold cross-validation*: randomly partition the entire data set into s non-overlapping exhaustive subsets $\{S_1, \ldots, S_s\}$, and loop as j (say) goes from 1 to s: set aside subset S_j, fit the model M_j on the union of all of the other subsets, and evaluate the quality of the fit on S_j, by using M_j to predict all of the y values in S_j; when the loop is finished, average the resulting s quality estimates to get an overall evaluation of the model's predictive accuracy that avoids over-fitting.

(f) *[8 total points for this sub-problem]* Run the fifth code block, which implements s–fold CV with $s = 10$ (this choice has been shown to be reliable), and study the output, which is summarized in a graph and a number: the graph plots the cross-validated predictions against the true values of the outcome variable `poisonous`, and the number is the CV estimate of what's called the *root-mean-squared error (RMSE)* $\hat{\sigma}$ of the regression predictions, namely $\hat{\sigma} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$ (here $\hat{y}_i$ is the predicted value of y_i; what the code actually does is (a) compute the $s = 10$ separate estimates $\hat{\sigma}_j$ of the $RMSE$ arising from the cross-validation process and then (b) combine the $\hat{\sigma}_j$ values optimally with $\hat{\sigma} = \sqrt{\frac{1}{n}\sum_{j=1}^{s}\hat{\sigma}_j^2}$). Does the graph arising from the CV process support the idea that the predictions from the full model $\mathbb{F}$ are perfect, even when cross-validated? Explain briefly *[4 points]*. Does the cross-validated $RMSE$ value also support this idea? Explain briefly *[4 points]*.

Now that we've achieved the rare feat of perfect prediction, let's think about the app we're designing: do we really want to make users supply multiple-choice answers to 21 different questions about the mushroom they're thinking of eating? The next (and nearly final) task is to see if a subset of the full set of 21 predictors can do as well, or nearly as well, in predictive accuracy as the full model $\mathbb{F}$.

(g) *[6 total points for this sub-problem]* Run the sixth code block, which implements a method called *step-wise variable selection*, using the *Bayesian Information Criterion (BIC)* we talked about in class; recall that lower BIC values correspond to better models. The output of this code block is sufficiently voluminous that I put it into another `.txt` file, also available on the course web page:

`ams-206-mushroom-analysis-variable-selection-with-bic.txt`

Study the output from this code block. This implementation of the `R` function `step` starts with the *null model* consisting of just an intercept term, and then sequentially chooses the best variable not yet in the model and adds it. For the mushroom data, the algorithm goes

Table 4: *Cross-tabulation of truth against what the app says for decision rules with two $\hat{p}$ cutoffs, 0.05 (left) and 0.01 (right).*

0.05 Cutoff

		Truth		
		Poisonous	Edible	Total
App Says	Poisonous			
	Edible			
	Total			

0.01 Cutoff

		Truth		
		Poisonous	Edible	Total
App Says	Poisonous			
	Edible			
	Total			

through 11 iterations of this method, until it discovers that the model with 10 sequentially-best predictors yields perfect predictions, at which point it stops with an excellent and snarky warning message. By thinking about what the output is saying about the best subset of x variables, and in what order, answer the following three questions, as k goes from 1 to 3:

> If the app were going to be based on only k variable(s), which {one is}/{ones are} best?

Explain briefly *[6 points]*.

(h) *[20 total points for this sub-problem]* Finally, suppose that we tentatively decide to base the app only on the mushroom's `odor`. We would then still have to specify a decision rule to implement in the app, as a function of the $\hat{p}$ value it produces for a new mushroom. It's easy to show (You're not asked to show this) that the optimal decision rule is of the form

> If $\hat{p} \geq c$, declare the mushroom poisonous; otherwise declare it edible

for some $0 \leq c \leq 1$. Run the seventh code block, which summarizes the quality of two `odor`–based decision rules, one with $c = 0.05$ and the other with $c = 0.01$. Fill out Table 4 above by **carefully** re-arranging the output of the final two `table` commands in the code block *[8 points]*. Letting (as usual with classification rules) {App says poisonous} be a positive (+) finding and {App says edible} be a negative (–) result, use your filled-out Table 4 to estimate the false-positive and false-negative rates for each of the 0.05– and 0.01–cutoff rules (You may wish to refer back to the *ELISA* case study in class and/or one of the quizzes) *[8 points]*. Considering the real-world implications of a false-positive error, and repeating for false-negative mistakes, is the 0.05–cutoff rule acceptable as a basis for our app? What about the 0.01–cutoff rule? Explain briefly in both cases *[4 points]*.

(i) *Extra credit [up to 20 extra points]:* Repeat part (h) (modifying code block 7 appropriately) with the app based on the best *two* predictor variables (instead of just `odor`). Is there now, with the two best predictors instead of one, an optimal cutoff that You regard as an acceptable trade-off between false-positive and false-negative mistakes, if You were going to sell the resulting app to wild-mushroom hunters? Explain briefly (*10 more extra-credit points* for repeating (h) with the best *three* predictors).