Prof. David Draper
Department of Statistics
Baskin School of Engineering
University of California, Santa Cruz
Winter 2024

STAT 206 (Applied Bayesian Statistics)

Take-Home Test 3: Part 2 (Required: 220 Total Points)

Absolute due date:
Uploaded to `canvas.ucsc.edu` by 11.59pm on Fri 22 Mar 2024

Here are the ground rules: this test is open-book and open-notes, and has two parts: Part 1 consists of 7 true/false questions, for a total of 70 points, and is ***Required***). Part 2 consists of a series of calculation questions for you to answer, with a total point value of 220, and is also ***Required***). ***All of Part 3 will be extra-credit.***

Note that this differs from what I've been saying recently in OH and optional LDS meetings, and represents the FINAL definition of Take-Home Test 3.

Name: Devanathan Nallur Gandamani (2086936)

Students who wish to gain full mastery of all of the material presented this quarter are strongly encouraged to participate in office hour sessions from now through Sun 24 Mar 2024.

Some advice on style as you write up your solutions: pretend that you're sitting next to the grader, having a conversation about problem (x) part (y). You say, "The answer is z," and the grader says, "Why?" You then give your explanation, as succinctly as possible to get your idea across. 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, the graders and 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.

This test is to be entirely your own efforts; do not collaborate with anyone or get help from anyone but me or our TAs. 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, 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.

> ***Under UCSC policies, submission of your solutions constitutes acceptance of the ethical rules stated above.***

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., `python`, `Matlab`, ...). 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 (see below) you can say something like the following:

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

As for the Appendix:

> ***Please collect {all of the code you used in answering the questions below that is either original to you or that represents a significant modification of my code} into an Appendix at the end of your document, so that (if you do something wrong) the graders can more accurately give you part credit. <u>NB</u> This means that you <u>don't</u> need to include in your Appendix any code that you've simply copied from my code.***

In what follows I've not left blank spaces for your solutions. Those of You who are using `LaTeX` or some other word-processing environment to prepare Your solutions can stick quote blocks below each question, into which You can type Your answers (I suggest that You use bold or italic font to distinguish Your solutions from the questions). If You're submitting Your answers in longhand, which is perfectly acceptable, You can just write them out on separate sheets of paper, making sure that the grader can easily figure out which chunk of text is the solution to which part of which problem.

Part 2 of this test is similar to problem 2(B) in Take-Home Test 2, in that it looks long but doesn't actually have that much for You to do: just read the problem carefully, run my code, and figure how to interpret the output.

Part 2: Calculation

[220 total points for this problem] 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 available in the `Pages` tab of the course `Canvas` 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 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 modeling in step (2) is accurate enough to release the app to the general public without poisoning a lot of people in the process.

We're going to do a maximum-likelihood analysis of this data set, because (I'm assuming that) You and I know so little about 'gilled mushrooms in the *Agaricus* and *Lepiota* Family' that a Bayesian analysis here would just reproduce the likelihood story. I'm also doing something a bit unusual in having You fit *linear* regression models to a binary outcome variable, but the more usual choice of *logistic* regression models (like those in part 3 of this test) would give essentially the same results.

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 standard English 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; there is such a variable, so early on in the analysis we'll drop it (`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 (that would be an undesirable action that's not needed with factor predictors). If those 2,480 values are *Missing Completely At Random* (MCAR; see Quiz 1), this will just make `stalk.root` noisier as a predictor; if they're not MCAR, we can use the fact of their missingness in the prediction process[1].

[1] Ideally we should do a sensitivity analysis in which the 2,480 rows are temporarily omitted from the data set, to see if we get the same results as those that arise when those rows are included; I've decided not to ask You to do that here, because the problem is already fairly long.

As discussed in the DD OH on 17 Mar 2024, the basic frequentist (multiple) linear regression model is of the form (for $i = 1, \ldots, n$)

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

in which the $(e_i \,|\, \sigma\, [SM\colon \mathbb{N}]\, \mathcal{B})$ are IID $N(0, \sigma^2)$; here $[SM\colon \mathbb{N}]$ denotes the Normality assumption for the sampling model (SM), which is not part of problem context. In the previously mentioned OH we also saw that this model can be written in matrix form as

$$\boldsymbol{y} = X\, \boldsymbol{\beta} + \boldsymbol{e} \,, \tag{2}$$

where $\boldsymbol{y}$ is the $(n \times 1)$ column 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 $[(k+1) \times 1]$ column vector whose transpose is $(\beta_0, \beta_1, \ldots, \beta_k)$ and $\boldsymbol{e}$ is the $(n \times 1)$ column 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 the 18th predictor in the data set `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 $x_{\{18\}} =$ `ring.number` as an example (here $x_{\{18\}}$ means the 18th predictor variable), having $\ell = 3$ levels, You create a new variable z_1 that's 1 if $x_{\{18\}} =$ "n" and 0 otherwise, and another new variable z_2 that's 1 if $x_{\{18\}} =$ "o" and 0 otherwise, and a third new variable z_3 that's 1 if $x_{\{18\}} =$ "t" and 0 otherwise.

If You now include all $\ell = 3$ of the z_j in the set of predictors, in place of $x_{\{18\}}$, 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 (2) 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

$$\begin{aligned} y_i \;\; = \;\; & \beta_0 + [\beta_1\, z_{i1} + \cdots + \beta_{\ell_1 - 1}\, z_{i,\ell_1 - 1}] + [\beta_{\ell_1}\, z_{i2} + \cdots + \beta_{\ell_1 + \ell_2 - 2}\, z_{i,\ell_1 + \ell_2 - 2}] \\ & + \cdots + \left[\beta_{L-K-(\ell_k - 2)}\, z_{i,L-K-(\ell_k-2)} + \cdots + \beta_{L-k}\, z_{i,L-k}\right] + e_i \,. \end{aligned} \tag{3}$$

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) ***[10 total points for this sub-problem]*** Create a new directory for this case study and download into this directory all of the files in the Pages tab of the course Canvas page that have 'mushroom' in their names. I've written some `R` code for You, to start You on the analysis of this data set; it's in the file you just downloaded called

`stat-206-mushroom-data-analysis.txt`

Table 1: *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 ***
```

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?). ***[10 points]***

The **tab.sum** function is a valuable statistical tool for investigating potential associations between categorical variables. In the context of mycology, it can be utilized to assess the likelihood of toxicity given certain mushroom characteristics. This function computes conditional probabilities, offering insights into the extent of the relationship between specific attributes and the toxicity of mushrooms. By comparing the observed frequencies to expected frequencies of toxic mushrooms within particular categories, we can determine if the presence of certain traits significantly deviates from what would be anticipated under the assumption of independence. In essence, this enables us to discern whether specific features are strong predictors of mushroom toxicity.

(b) ***[20 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 (3) are specialized to this regression, they looks like

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 \, z_{i1} + \cdots + \hat{\beta}_5 \, z_{i5} \, , \tag{4}$$

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 1. Explicitly identify $(\hat{\beta}_0, \ldots, \hat{\beta}_5)$ in the output in the table ***[10 points]***, and – by thinking about the form of equation (4) for each of the levels of `cap.shape` – explicitly relate the numbers in the `mean` column of the `tab.sum` output in Table 1 to the $\hat{\beta}_j$. ***[10 points]***

$\hat{\beta}_0 = 0.10619$ There is no value to find a difference with. So this is just equal to the mean

$\hat{\beta}_1 = 0.89381$ This is the difference in mean of capshape 2 and capshape 1 ie $1 - 0.10619 = 0.89381$

$\hat{\beta}_2 = 0.38746$ This is the difference in mean of capshape 3 and capshape 1 ie $0.49365 - 0.10619 = 0.38746$

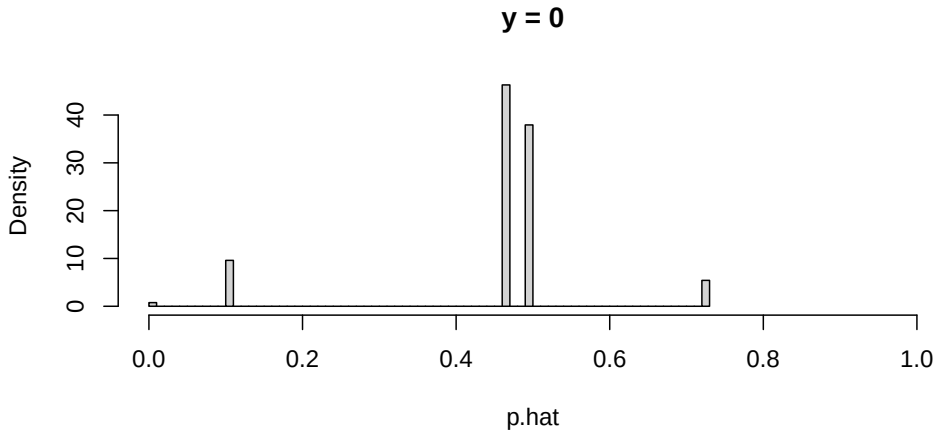
$\hat{\beta}_3 = 0.61844$ This is the difference in mean of capshape 4 and capshape 1 ie $0.72463 - 0.10619 = 0.61844$

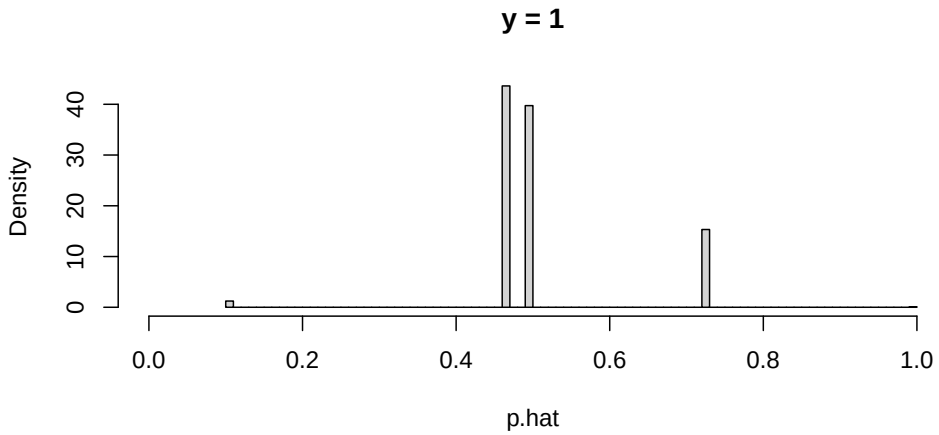
$\hat{\beta}_4 = -0.10619$ This is the difference in mean of capshape 5 and capshape 1 ie $0 - 0.10619 = -0.10619$

$\hat{\beta}_5 = 0.36098$ This is the difference in mean of capshape 6 and capshape 1 ie $0.46717 - 0.10619 = 0.36098$

(c) ***[40 total points for this sub-problem]*** Toward the end of the second code block, the code computes predicted $\hat{p} = P(y = 1 \,|\, x_1 \, [SM : \mathbb{L}] \, \mathcal{B})$ values (in which $[SM : \mathbb{L}]$ 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) ***[20 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 offers perfect information for predicting a dichotomous outcome y? Explain briefly ***[10 points]***. Do the histograms achieve that goal with the predictor `cap.shape`? Explain briefly ***[10 points]***.





The optimal distribution for these histograms, reflecting a model's predictive power, would manifest as pronounced peaks near the extremities of 0 or 1. Specifically, for observations where y = 0, an ideal p.hat would concentrate at 0 with a density of 100, and similarly, for y = 1, p.hat would concentrate at 1 with the same density, indicating highly accurate predictions. The current histograms approximate this desired outcome to a degree. Absent any predictive relationship, we would expect to observe a concentration around 0.48, indicative of no predictive power beyond random chance. While the current histograms exhibit a distribution that surpasses this baseline, they do not exhibit the pronounced skewness towards the extremes that would signify a robust predictive relationship. In the case of cap shape, the overlap in density across both histograms suggests that cap shape alone provides minimal predictive leverage regarding the toxicity of mushrooms.

(ii) ***[20 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:

$$PSI(x) = [\text{mean } (\hat{p} \text{ given } x) \text{ when } y = 1] - [\text{mean } (\hat{p} \text{ given } x) \text{ when } y = 0]\,. \tag{5}$$

What's the ideal value of the PSI, if a factor x is perfectly predictive of y? Explain briefly ***[10 points]***. Does the PSI come close to achieving that goal with `cap.shape`? Explain briefly ***[10 points]***.

In an ideal scenario, the Population Stability Index (PSI) would attain a value of 1 for a variable x that demonstrates perfect predictiveness for an outcome y. This would be reflected in a scenario where $\hat{p}$ equals 1 whenever y equals 1, and $\hat{p}$ equals 0 whenever y equals 0, signifying flawless prediction accuracy. A PSI of 0, conversely, would indicate a complete absence of predictive relationship. In the context of cap shape as a predictive factor, a PSI value of approximately 0.0603 suggests a negligible predictive power, as this value lies significantly closer to 0 than the ideal of 1, indicating minimal efficacy of cap shape in predicting mushroom toxicity.

(d) ***[30 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 2 (sort your entries from highest PSI down to lowest); I suggest that You use the phrases *extremely strong*, *strong*, *moderate*, *weak*, and *almost none* to describe the predictive power of each x variable (you can choose your own cutpoints defining those categories; there are no unique right answers; just be reasonable in your choices) ***[20 points]***. If You were going to base the app on only one or two predictors, which ones look like the best candidates? Explain briefly ***[10 points]***.

Refer to the filled table 2 in previous page

Selecting only a pair of predictors, both odor and spore print color emerge with the highest Predictive Score Index (PSI), suggesting that they are particularly efficacious in forecasting mushroom toxicity. Particularly, odor and spore print color outperform other potential predictors for our application. However, one issue with relying on odor is the subjective nature of olfactory perception, which can vary among individuals, potentially introducing variability in the predictions. Consequently, spore print color, alongside gill color, may offer more consistency and reduce the likelihood of human error when utilized in our application.

Table 2: *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
`odor`	0.9429	Extremely strong
`spore.print.color`	0.5665	Extremely strong
`gill.color`	0.4635	Strong
`ring.type`	0.3639	Strong
`stalk.surface.above.ring`	0.3457	Strong
`stalk.surface.below.ring`	0.3304	Strong
`gill.size`	0.2916	Moderate
`stalk.color.above.ring`	0.2755	Moderate
`stalk.color.below.ring`	0.2649	Moderate
`bruises`	0.2515	Moderate
`population`	0.2375	Moderate
`habitat`	0.1937	Moderate
`stalk.root`	0.1655	Moderate
`gill.spacing`	0.1214	Moderate
`cap.shape`	0.0603	weak
`cap.color`	0.0477	weak
`ring.number`	0.0461	weak
`cap.surface`	0.0388	weak
`veil.color`	0.0235	weak
`gill.attachment`	0.0167	Almost none
`stalk.shape`	0.0104	Almost none

(e) ***[30 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 in the Pages tab of the course Canvas page; based on Your reading of these pages, briefly explain what the number in the regression output is trying to measure ***[10 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.) ***[10 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? ***[10 points]***

The value **0.03878**, which corresponds to the Multiple R-Squared statistic, is designed to quantify the degree of correlation between the predicted values and the actual observations by squaring the correlation coefficient. It is logical to observe a congruence between the PSI and the Multiple R-Squared as both metrics aim to capture the effectiveness of the predictive model. However, the equivalence of the PSI and Multiple R-Squared values for binary outcomes is not a universally observed phenomenon; rather, they tend to converge under specific conditions aligned with the nature of the predictive model used.

Run the fourth code block, in which a linear regression model is fit with all available predictors (let's call this the *full (sampling) model* [SM:$\mathbb{F}$]), 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 $P(y_i = 1 \,|\, \boldsymbol{x}_i\, [SM{:}\,\mathbb{L}\,\mathbb{F}]\, \mathcal{B}) = 1$, and all of the truly edible mushrooms have estimated $P(y_i = 1 \,|\, \boldsymbol{x}_i\, [SM{:}\,\mathbb{L}\,\mathbb{F}]\, \mathcal{B}) = 0$ (here $\boldsymbol{x}_i$ is the vector of predictor variables for mushroom i).

However, this evaluation of the predictive quality of $[SM{:}\,\mathbb{L}\,\mathbb{F}]$ may overstate its accuracy, because we used the same (entire) data set both to fit $[SM{:}\,\mathbb{L}\,\mathbb{F}]$ and then to see how good $[SM{:}\,\mathbb{L}\,\mathbb{F}]$ 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) ***[20 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 $[SM{:}\,\mathbb{L}\,\mathbb{F}]$ are perfect, even when cross-validated? Explain briefly ***[10 points]***. Does the cross-validated $RMSE$ value also support this idea? Explain briefly ***[10 points]***.

- The depicted graph indicates a flawless predictive performance, as evidenced by the exclusive clustering of points at the coordinates (0,0) and (1,1), denoting absolute concordance between predicted and observed values.
- The Root Mean Square Error (RMSE) metric corroborates this observation, with a value of 2.123322×10^{-13}, which is approximately zero, reflecting minimal deviation from the perfect prediction.

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 in this problem 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 $[SM{:}\,\mathbb{L}\,\mathbb{F}]$.

(g) ***[10 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:

`stat-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

Table 3: *Cross-tabulation of truth against what the app says for decision rules based on* `odor` *with two* $\hat{p}$ *cutoffs, 0.05 (left) and 0.01 (right).*

0.05 Cutoff

		Truth		
		Poisonous	Edible	Total
App Says	Poisonous	3796	0	3796
	Edible	120	4208	4328
	Total	3916	4208	8124

0.01 Cutoff

		Truth		
		Poisonous	Edible	Total
App Says	Poisonous	3916	3408	7324
	Edible	0	800	800
	Total	3916	4208	8124

yet in the model and adds it. For the mushroom data, the algorithm goes 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 (note that 3 answers are needed here) ***[10 points]***.

During the initial iterations, our algorithm systematically selects variables or their combinations for the application's foundation, guided by their efficacy in minimizing the Bayesian Information Criterion (BIC):

- Odor
- Odor and Spore Print Color
- Odor, Spore Print Color, and Stalk Color Below Ring

This selection strategy emphasizes reducing the BIC value over directly choosing variables based on the highest Predictive Score Index (PSI). Although the objectives of minimizing BIC and maximizing PSI are not entirely congruent, both approaches are interconnected in their aim to refine model accuracy and simplicity. By limiting the selection to one, two, or three variables, the algorithm effectively balances model complexity with predictive capability.

(h) ***[30 total points for this sub-problem]*** 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 3 above by **carefully** re-arranging the output of the final two `table` commands in the code block ***[10 points]***.

Refer to the filled table on top of this page

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 3 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 Take-Home Test 1) ***[10 points]***.

False Positive Rate (or) False Discovery Rate (FDR):

$$FDR = \frac{FP}{TP \ + \ FP}$$

$For\ 0.05\ cutoff, FDR = \ \frac{0}{3796+0} \ = \ 0\%$

$For\ 0.01\ cutoff, FDR = \ \frac{0}{3796+0} \ = \ 46.53\%$

False Negative Rate (or) False Omission Rate (FDR):

$$FOR = \frac{FN}{TN \ + \ FN}$$

$For\ 0.05\ cutoff, FOR = \ \frac{120}{120+4208} \ = \ 2.77\%$

$For\ 0.01\ cutoff, FOR = \ \frac{0}{0+800} \ = \ 0\%$

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 ***[10 points]***.

- **0.05 acceptable or not:** I'm not sure the 0.05 cutoff is acceptable. If the false negatives resulting in somebody dying, it would be on my conscience because somebody died after listening to the app. Morever, people get sued for this.
- **0.01 acceptable or not:** While the 0.01 cutoff removes our false negatives, it also cuts down on the amount of edible mushrooms. Better safe than sorry. So, I would say 0.01 cutoff is more acceptable than the 0.05 one.

(i) ***[10 total points for this sub-problem]*** Repeat part (h) (modifying code block 7 appropriately) with the app based on the best *two* predictor variables (instead of just `odor`), and exploring new cutoff values c. 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 points]***.

Lets take the top two variables ie odor and spore.print.color

False Positive Rate (or) False Discovery Rate (FDR):

$For\ 0.05\ cutoff, FDR = \ \frac{0}{3584+0} \ = \ 0\%$

$For\ 0.07\ cutoff, FDR = \ \frac{0}{3632+0} \ = \ 0\%$

$For\ 0.08\ cutoff, FDR = \ \frac{48}{4208+48} \ = \ 1.12\%$

False Negative Rate (or) False Omission Rate (FDR):

$For\ 0.05\ cutoff, FOR = \ \frac{624}{624+3916} \ = \ 13.74\%$

$For\ 0.07\ cutoff, FOR = \ \frac{576}{576+3916} \ = \ 12.82\%$

$For\ 0.08\ cutoff, FOR = \ \frac{0}{0+3868} \ = \ 0\%$

The optimal configuration for predictive accuracy appears to be achieved utilizing the variables of odor and spore print color with a threshold of 0.08. This setup yields a notably low False Positive Rate of 1.12

The primary rationale for selecting this combination is its False Negative Rate, which stands at 0

Comparatively, configurations with cutoffs at 0.05 and 0.07 exhibit no False Positives, suggesting an elimination of unnecessary resource utilization. However, they suffer from elevated False Negative Rates, posing a significant risk of failing to identify poisonous mushrooms and potentially leading to harmful exposures.

(j) ***[20 total points for this sub-problem]*** A student (Burleigh Charlton) in an earlier incarnation of this course raised the following issue about our app, which is referred to in the statistical data science literature as the problem of *errors-in-variables* or *measurement error in the predictors*: Yes, `odor` is a great predictor, but what happens to our predictive accuracy if a non-expert user of the app cannot precisely distinguish among the odors {almond, anise, creosote, fishy, foul, musty, none, pungent, spicy}? Discuss briefly ***[10 points]***.

For users to effectively utilize this application, a keen olfactory ability is crucial, particularly in distinguishing various scents accurately. A deficiency in this skill can significantly compromise the app's efficacy, making it challenging for non-expert users to verify the poisonous nature of mushrooms, notwithstanding the model's precise predictions. Regrettably, the realm of computational olfaction does not parallel the advancements seen in computer vision, limiting the app's utility for individuals lacking expertise in mushroom identification. In essence, the application's benefits may not fully extend to those without specialized knowledge.

How do You feel now about releasing the app to the general public? Explain briefly ***[10 points]***.

Before making this application available to the wider public, it would be prudent to include a cautionary note advising its optimal use in the presence of an individual possessing proficient olfactory skills and mushroom identification experience. This precaution aims to mitigate potential legal repercussions arising from users misapplying the app, leading to adverse outcomes and possible litigation.

Appendix

Here's all of the `R` code that I used to complete this test.

Note: I used some of Prof. Draper's `R` code in this assignment, adapting it as needed.

```
################################################################################
################################################################################

# stat 206, winter 2024

# R code for a partial analysis of the mushroom data set in THT 3 part 2

# dd 17 mar 2024

# first thing, (outside of R) create a directory where
```

```
# you wish to store all of the files relevant to this data analysis

# on my home desktop this new directory is set as follows:

#  setwd( 'C:/DD/Teaching/AMS-STAT-206/Winter-2024/Take-Home-Test-3/Part-2' )

# now download the following files from the Pages tab in the course
# Canvas page into that directory:

#   stat-206-mushroom-data-context.txt
#   stat-206-mushroom-data.txt

# open the file 'stat-206-mushroom-data-context.txt' and read it carefully;
# as i've emphasized many times in this class, we *cannot* do a
# meaningful analysis of a data set without understanding its context

# then open the file 'stat-206-mushroom-data.txt' and look at
# the raw data values; what interesting things do you notice?

###################### the first block of code starts here ####################

# unbuffer the output, if relevant in your R environment

# launch a new copy of R and start with an empty R workspace

rm( list = ls( ) )

# change directory inside R to the location to which you downloaded
# the file 'stat-206-mushroom-data.txt'

# for me on my home desktop this is done with the function call

#   setwd( 'C:/DD/Teaching/AMS-STAT-206/Winter-2024/Take-Home-Test-3/Part-2' )

# read in the raw data set

raw.mushroom.data <- read.csv( 'stat-206-mushroom-data.txt', header = T,
  stringsAsFactors = T )


print(

  n <- dim( raw.mushroom.data )[ 1 ]

)

# [1] 8124
```

```
# this is the sample size in this data set

# the next function call converts the categorical variable 'poisonous'
# into a quantitative dichotomous binary variable, in which 1 = poisonous
# and 0 = edible

raw.mushroom.data$poisonous <- ifelse( raw.mushroom.data$poisonous == 'p',
  1, 0 )

# in the table arising from the str( ) function call above, notice that
# the variable 'veil.type' is constant (it's a factor with only 1 level);
# such variables obviously have no predictive power, so the next function call
# drops this variable from the data set

raw.mushroom.data <- subset( raw.mushroom.data, select = - veil.type )

with( raw.mushroom.data, mean( poisonous ) )

# [1] 0.4820286

# the built-in function call 'with( )' allows us to refer to
# the variables in the data frame 'raw.mushroom.data' directly
# by name instead of having to type (e.g.) 'raw.mushroom.data$poisonous'

# the proportion of poisonous mushrooms in the data set is about 48%;
# prediction of a dichotomous (binary) outcome is easiest when the data set
# contains 50% yes/1/poisonous and 50% no/0/edible values, so this
# is close to optimal

##### UNDER CONSTRUCTION: everything below this line has not yet been edited

# the next function enables us to examine the relationship
# between each of the predictor variables (one at a time)
# and the dichotomous outcome

tab.sum <- function( x.1, y ) {

  # written by Daniel Helkey, edited by David Draper

  # summarize outcome variable y by levels of the categorical variable x1

  # make sure x and y are same length, etc...

  stopifnot( length( x.1 ) == length( y ) )

  # two helper functions

  summary.function <- function( x ) {
```

```
    return( list( n = length( x ), mean = mean( x, na.rm = TRUE ),
      sd = sd( x, na.rm = TRUE ) ) )

  }

  map.function <- function( level ) {

    indices <- ( x.1 == level )

    summary.function( y[ indices ] )

  }

  levels <- sort( unique( x.1 ) )

  out.matrix <- do.call( rbind, Map( map.function, levels ) )

  out.matrix <- cbind( levels, out.matrix )

  out.matrix <- rbind( out.matrix, c( 'Total', summary.function( y ) ) )

  colnames( out.matrix )[ 1 ] <- as.character( substitute( x.1 ) )

  return( out.matrix )

}

# the next function call computes the relative frequency distribution
# of the variable 'cap.shape'

with( raw.mushroom.data,
  signif( table( cap.shape, useNA = 'always' ) / n, 4 ) )

# cap.shape
#         b         c         f         k         s         x      <NA>
# 0.0556400 0.0004924 0.3880000 0.1019000 0.0039390 0.4500000 0.0000000

# consulting the text file containing contextual information
# about this data set, you can see that most of the mushrooms
# have a cap shape that's convex (category 'x'), followed by
# flat ('f'), with the other categories less abundant

# here, 'convex' probably actually means concave (bowl-shaped-down)

# you can also see that this variable has no missing values ('NA')

with( raw.mushroom.data, tab.sum( cap.shape, poisonous ) )
```

```
#      cap.shape n    mean      sd

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

# [7,] "Total"   8124 0.4820286 0.4997077

# the table above this line summarizes the relative frequency
# of poisonous mushrooms for each cap shape

# the numbering of the cap.shape categorical values in
# this table (1-6) corresponds to the alphabetical order
# of the categories listed in the table above:

# cap.shape
#         b         c         f         k         s         x      <NA>
# 0.0556400 0.0004924 0.3880000 0.1019000 0.0039390 0.4500000 0.0000000

# if all of the cap shape categories had the same proportion
# of poisonous mushrooms (namely, the overall poisonous rate of 48%),
# that would tell us that cap.shape has *no* predictive usefulness

# instead we see that there's a bit of signal here: for example,
# 828 of the mushrooms (the ones with cap.shape category 4,
# which the relative-frequency table above shows is 'k' and
# the context file says is knobbed) have a poisonous rate of about
# 72%, which is far above average, and 452 of the mushrooms
# category 1 ('b': bell) have a poisonous rate of only 10.6%,
# which is well below average

# so cap.shape looks like a decent candidate to put into
# our predictive models

################### the first block of code ends here ##################

################ the second block of code starts here ##################

# the next function call fits a linear regression model
# in which the binary outcome variable 'poisonous'
# is regressed on the categorical predictor 'cap.shape',
# and then summarizes the fit

summary( linear.model.1 <- lm( poisonous ~ cap.shape,
```

```
  data = raw.mushroom.data ) )

# Call:
# lm(formula = poisonous ~ cap.shape, data = raw.mushroom.data)

# Residuals:
#     Min      1Q  Median      3Q     Max
# -0.7246 -0.4672 -0.1062  0.5063  0.8938

# 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 ***
# cap.shapef   0.38746    0.02437  15.898  < 2e-16 ***
# cap.shapek   0.61844    0.02834  21.824  < 2e-16 ***
# cap.shapes  -0.10619    0.08864  -1.198 0.230926
# cap.shapex   0.36098    0.02416  14.942  < 2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

#      cap.shape n    mean      sd

# [1,] b         452  0.1061947 0.308428
# [2,] c         4    1         0
# [3,] f         3152 0.4936548 0.5000391
# [4,] k         828  0.7246377 0.4469667
# [5,] s         32   0         0
# [6,] x         3656 0.4671772 0.4989898

# [7,] "Total"   8124 0.4820286 0.4997077

# cap.shape
#         b         c         f         k         s         x      <NA>
# 0.0556400 0.0004924 0.3880000 0.1019000 0.0039390 0.4500000 0.0000000

# Residual standard error: 0.4846 on 8118 degrees of freedom
# Multiple R-squared:  0.06031,   Adjusted R-squared:  0.05973
# F-statistic: 104.2 on 5 and 8118 DF,  p-value: < 2.2e-16

# recall that a *dummy* variable is just an indicator,
# taking the value 1 if some proposition is true and 0 if it's false

# this factor (x.1 = cap.shape) has k.1 = 6 levels; R has created
# ( k.1 - 1 ) = 5 temporary dummy variables called cap.shapec,
# cap.shapef, ..., cap.shapex corresponding to the levels 'c',
# 'f', ..., 'x' of x.1 and included these dummies as predictors

# this means that R has chosen the omitted group to be
```

```
# the first (in alphabetical order) level 'b' of x.1 = cap.shape;
# R always does this (i.e., it always omits the dummy variable
# corresponding to the alphabetically first level of a factor)

# the next function call creates a vector of p.hat values from linear.model.1

p.hat.linear.model.1 <- predict( linear.model.1, type = 'response' )

# the next four function calls make a diagnostic plot from the p.hat values;

par( mfrow = c( 2, 1 ) )

hist( p.hat.linear.model.1[ raw.mushroom.data$poisonous == 0 ],
  nclass = 100, main = 'y = 0', xlab = 'p.hat', prob = T,
  xlim = c( 0, 1 ) )

hist( p.hat.linear.model.1[ raw.mushroom.data$poisonous == 1 ],
  nclass = 100, main = 'y = 1', xlab = 'p.hat', prob = T,
  xlim = c( 0, 1 ) )

par( mfrow = c( 1, 1 ) )


pdf( 'stat-206-take-home-test-3-part-2-solutions-figure-1.pdf' )

par( mfrow = c( 2, 1 ) )

hist( p.hat.linear.model.1[ raw.mushroom.data$poisonous == 0 ],
  nclass = 100, main = 'y = 0', xlab = 'p.hat', prob = T,
  xlim = c( 0, 1 ) )

hist( p.hat.linear.model.1[ raw.mushroom.data$poisonous == 1 ],
  nclass = 100, main = 'y = 1', xlab = 'p.hat', prob = T,
  xlim = c( 0, 1 ) )

par( mfrow = c( 1, 1 ) )

dev.off( )

# the next function call computes a diagnostic i call the
# *predictive separation index* (psi), the last number
# in the vector of three numbers

c( mean( p.hat.linear.model.1[ raw.mushroom.data$poisonous == 0 ] ),
  mean( p.hat.linear.model.1[ raw.mushroom.data$poisonous == 1 ] ),
  mean( p.hat.linear.model.1[ raw.mushroom.data$poisonous == 1 ] ) -
  mean( p.hat.linear.model.1[ raw.mushroom.data$poisonous == 0 ] ) )
```

```
# [1] 0.45295970 0.51326496 0.06030526

################ the second block of code ends here #################
################ the third block of code begins here ################

# here i define a function that helps us to see
# the marginal predictive quality of a predictor variable

# coding note: i'm using a bad coding practice in this function by
# hardwiring the name of the data frame into the body of the function
# instead of passing it in as an argument (input) to the function;
# i've found it hard to get R to do the right thing,
# because R is tricky in the way it distinguishes between
# an object and the name of the object

univariate.exploration <- function( new.variable ) {

  print( with( raw.mushroom.data,
    signif( table( new.variable, useNA = 'always' ) / n, 4 ) ) )

  print( with( raw.mushroom.data, tab.sum( new.variable, poisonous ) ) )

  print( summary( temp.linear.model <- lm( poisonous ~ new.variable,
    data = raw.mushroom.data ) ) )

  p.hat.temp.linear.model <- predict( temp.linear.model, type = 'response' )

  par( mfrow = c( 2, 1 ) )

  hist( p.hat.temp.linear.model[ raw.mushroom.data$poisonous == 0 ],
    nclass = 100, main = 'y = 0', xlab = 'p.hat', prob = T,
    xlim = c( 0, 1 ) )

  hist( p.hat.temp.linear.model[ raw.mushroom.data$poisonous == 1 ],
    nclass = 100, main = 'y = 1', xlab = 'p.hat', prob = T,
    xlim = c( 0, 1 ) )

  par( mfrow = c( 1, 1 ) )

  print( c(
    mean( p.hat.temp.linear.model[ raw.mushroom.data$poisonous == 0 ] ),
    mean( p.hat.temp.linear.model[ raw.mushroom.data$poisonous == 1 ] ),
    mean( p.hat.temp.linear.model[ raw.mushroom.data$poisonous == 1 ] ) -
    mean( p.hat.temp.linear.model[ raw.mushroom.data$poisonous == 0 ] ) ) )

}

with( raw.mushroom.data, univariate.exploration( cap.surface ) )
```

```
# new.variable
#         f         g         s         y      <NA>
# 0.2856000 0.0004924 0.3146000 0.3993000 0.0000000
#      new.variable n    mean      sd
# [1,] 1            2320 0.3275862 0.4694342
# [2,] 2            4    1         0
# [3,] 3            2556 0.5524257 0.4973413
# [4,] 4            3244 0.5363748 0.498752
# [5,] "Total"      8124 0.4820286 0.4997077

# Call:
# lm(formula = poisonous ~ new.variable, data = raw.mushroom.data)

# Residuals:
#     Min      1Q  Median      3Q     Max
# -0.5524 -0.5364 -0.3276  0.4636  0.6724

# Coefficients:
#               Estimate Std. Error t value Pr(>|t|)
# (Intercept)    0.32759    0.01017  32.200  < 2e-16 ***
# new.variableg  0.67241    0.24522   2.742  0.00612 **
# new.variables  0.22484    0.01405  16.001  < 2e-16 ***
# new.variabley  0.20879    0.01332  15.671  < 2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Residual standard error: 0.49 on 8120 degrees of freedom
# Multiple R-squared:  0.03878,   Adjusted R-squared:  0.03842
# F-statistic: 109.2 on 3 and 8120 DF,  p-value: < 2.2e-16

# [1] 0.46333584 0.50211511 0.03877928

# to call this function, just replace

#   cap.surface

# in

#   with( raw.mushroom.data, univariate.exploration( cap.surface ) )

# with the name of the predictor variable you want to explore
# (e.g., bruises, ...)

# added after test was assigned in 2022

# written by nate larsen
```

```
install.packages( 'R.utils' )

require( R.utils )

predictors <- colnames( raw.mushroom.data )[ 2:22 ]

for ( predictor in predictors) {

  printf( "\n\n==============  %s  ==============\n", predictor )

  with( raw.mushroom.data,
    univariate.exploration( eval( parse( text = predictor ) ) ) )

}

################ the third block of code ends here #################

############### the fourth block of code begins here ################

# the next function call fits a linear model using all predictors

summary( linear.model.all.predictors <- lm( poisonous ~ .,
  data = raw.mushroom.data ) )

# Call:
# lm(formula = poisonous ~ ., data = raw.mushroom.data)

# Residuals:
#        Min         1Q     Median         3Q        Max
# -1.863e-12 -9.900e-15 -7.000e-16  7.800e-15  2.636e-11

# Coefficients: (10 not defined because of singularities)
#                            Estimate Std. Error    t value Pr(>|t|)
# (Intercept)                4.509e-13  2.843e-13  1.586e+00 0.112793
# cap.shapec                 4.351e-13  1.898e-13  2.293e+00 0.021893 *
# cap.shapef                 3.121e-14  2.157e-14  1.446e+00 0.148077
# cap.shapek                 3.524e-13  2.345e-14  1.503e+01  < 2e-16 ***
# cap.shapes                -1.149e-14  7.394e-14 -1.550e-01 0.876466
# cap.shapex                 2.952e-13  2.067e-14  1.428e+01  < 2e-16 ***
# cap.surfaceg              -3.231e-13  2.320e-13 -1.393e+00 0.163785
# cap.surfaces              -2.751e-14  1.235e-14 -2.228e+00 0.025937 *
# cap.surfacey              -3.800e-14  1.035e-14 -3.672e+00 0.000243 ***
# cap.colorc                 5.543e-14  7.168e-14  7.730e-01 0.439323
# cap.colore                -1.709e-15  3.337e-14 -5.100e-02 0.959145
# cap.colorg                 4.570e-14  3.197e-14  1.429e+00 0.152914
# cap.colorn                 5.296e-15  3.262e-14  1.620e-01 0.871038
# cap.colorp                -3.893e-14  4.105e-14 -9.480e-01 0.343043
# cap.colorr                 1.660e-13  1.203e-13  1.380e+00 0.167615
```

```
# cap.coloru                 1.293e-13  1.203e-13  1.075e+00 0.282396
# cap.colorw                 4.347e-14  3.217e-14  1.351e+00 0.176600
# cap.colory                 3.094e-14  3.423e-14  9.040e-01 0.366158
# bruisest                   5.000e-01  1.102e-13  4.538e+12  < 2e-16 ***
# odorc                      5.000e-01  1.318e-13  3.792e+12  < 2e-16 ***
# odorf                     -5.000e-01  3.300e-13 -1.515e+12  < 2e-16 ***
# odorl                     -2.673e-15  2.318e-14 -1.150e-01 0.908213
# odorm                      2.500e+00  5.191e-13  4.816e+12  < 2e-16 ***
# odorn                     -1.500e+00  3.184e-13 -4.712e+12  < 2e-16 ***
# odorp                     -1.000e+00  3.964e-13 -2.523e+12  < 2e-16 ***
# odors                     -5.000e-01  3.306e-13 -1.512e+12  < 2e-16 ***
# odory                     -5.000e-01  3.306e-13 -1.512e+12  < 2e-16 ***
# gill.attachmentf           5.811e-14  1.093e-13  5.320e-01 0.594898
# gill.spacingw             -3.494e-14  4.732e-14 -7.380e-01 0.460285
# gill.sizen                -1.500e+00  2.984e-13 -5.027e+12  < 2e-16 ***
# gill.colore               -5.000e-01  1.976e-13 -2.530e+12  < 2e-16 ***
# gill.colorg               -5.000e-01  1.957e-13 -2.555e+12  < 2e-16 ***
# gill.colorh               -5.000e-01  1.953e-13 -2.560e+12  < 2e-16 ***
# gill.colork               -5.000e-01  1.960e-13 -2.552e+12  < 2e-16 ***
# gill.colorn               -5.000e-01  1.956e-13 -2.557e+12  < 2e-16 ***
# gill.coloro               -5.000e-01  2.033e-13 -2.460e+12  < 2e-16 ***
# gill.colorp               -5.000e-01  1.952e-13 -2.562e+12  < 2e-16 ***
# gill.colorr               -5.000e-01  2.115e-13 -2.364e+12  < 2e-16 ***
# gill.coloru               -5.000e-01  1.961e-13 -2.549e+12  < 2e-16 ***
# gill.colorw               -5.000e-01  1.948e-13 -2.567e+12  < 2e-16 ***
# gill.colory               -5.000e-01  2.015e-13 -2.481e+12  < 2e-16 ***
# stalk.shapet              -1.000e+00  1.944e-13 -5.145e+12  < 2e-16 ***
# stalk.rootb               -5.000e-01  1.287e-13 -3.886e+12  < 2e-16 ***
# stalk.rootc               -3.000e+00  6.028e-13 -4.977e+12  < 2e-16 ***
# stalk.roote                5.000e-01  1.184e-13  4.222e+12  < 2e-16 ***
# stalk.rootr               -3.500e+00  5.190e-13 -6.744e+12  < 2e-16 ***
# stalk.surface.above.ringk  5.733e-15  2.455e-14  2.340e-01 0.815332
# stalk.surface.above.rings  6.957e-15  1.973e-14  3.530e-01 0.724415
# stalk.surface.above.ringy -7.417e-15  2.696e-13 -2.800e-02 0.978056
# stalk.surface.below.ringk  3.675e-15  2.455e-14  1.500e-01 0.880992
# stalk.surface.below.rings  3.989e-15  1.973e-14  2.020e-01 0.839786
# stalk.surface.below.ringy  5.000e-01  1.647e-13  3.035e+12  < 2e-16 ***
# stalk.color.above.ringc           NA         NA         NA       NA
# stalk.color.above.ringe   -9.794e-15  5.372e-14 -1.820e-01 0.855337
# stalk.color.above.ringg   -7.479e-16  2.854e-14 -2.600e-02 0.979098
# stalk.color.above.ringn    2.426e-16  2.231e-14  1.100e-02 0.991321
# stalk.color.above.ringo   -1.000e+00  2.487e-13 -4.021e+12  < 2e-16 ***
# stalk.color.above.ringp   -7.090e-16  2.231e-14 -3.200e-02 0.974644
# stalk.color.above.ringw   -6.699e-16  2.543e-14 -2.600e-02 0.978987
# stalk.color.above.ringy    3.000e+00  4.990e-13  6.013e+12  < 2e-16 ***
# stalk.color.below.ringc           NA         NA         NA       NA
# stalk.color.below.ringe   -3.626e-15  5.372e-14 -6.800e-02 0.946181
# stalk.color.below.ringg    8.068e-16  2.854e-14  2.800e-02 0.977452
```

```
# stalk.color.below.ringn    6.493e-16  2.231e-14  2.900e-02 0.976776
# stalk.color.below.ringo           NA         NA         NA       NA
# stalk.color.below.ringp    6.740e-16  2.231e-14  3.000e-02 0.975895
# stalk.color.below.ringw    6.205e-16  2.543e-14  2.400e-02 0.980537
# stalk.color.below.ringy    1.474e-16  1.180e-13  1.000e-03 0.999004
# veil.coloro                1.224e-16  4.732e-14  3.000e-03 0.997937
# veil.colorw                       NA         NA         NA       NA
# veil.colory                       NA         NA         NA       NA
# ring.numbero               2.500e+00  4.776e-13  5.235e+12  < 2e-16 ***
# ring.numbert                      NA         NA         NA       NA
# ring.typef                 1.000e+00  2.390e-13  4.185e+12  < 2e-16 ***
# ring.typel                        NA         NA         NA       NA
# ring.typen                        NA         NA         NA       NA
# ring.typep                 5.000e-01  1.002e-13  4.989e+12  < 2e-16 ***
# spore.print.colorh                NA         NA         NA       NA
# spore.print.colork         4.758e-15  6.777e-14  7.000e-02 0.944034
# spore.print.colorn        -1.446e-17  6.692e-14  0.000e+00 0.999828
# spore.print.coloro        -1.100e-16  6.692e-14 -2.000e-03 0.998688
# spore.print.colorr         2.500e+00  3.196e-13  7.822e+12  < 2e-16 ***
# spore.print.coloru        -8.504e-15  9.463e-14 -9.000e-02 0.928401
# spore.print.colorw         1.500e+00  3.066e-13  4.893e+12  < 2e-16 ***
# spore.print.colory         2.863e-24  6.692e-14  0.000e+00 1.000000
# populationc               -6.210e-14  5.716e-14 -1.086e+00 0.277296
# populationn                3.537e-14  3.312e-14  1.068e+00 0.285547
# populations               -8.241e-15  2.366e-14 -3.480e-01 0.727585
# populationv               -6.210e-14  3.207e-14 -1.937e+00 0.052833 .
# populationy               -5.922e-14  3.322e-14 -1.783e+00 0.074703 .
# habitatg                   3.339e-16  1.969e-14  1.700e-02 0.986468
# habitatl                  -5.514e-17  1.821e-14 -3.000e-03 0.997584
# habitatm                   7.764e-16  3.352e-14  2.300e-02 0.981521
# habitatp                  -1.103e-16  1.440e-14 -8.000e-03 0.993889
# habitatu                   9.875e-14  3.432e-14  2.878e+00 0.004019 **
# habitatw                          NA         NA         NA       NA
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Residual standard error: 3.278e-13 on 8038 degrees of freedom
# Multiple R-squared:      1,     Adjusted R-squared:      1
# F-statistic: 2.221e+26 on 85 and 8038 DF,  p-value: < 2.2e-16

# the NA rows are telling us that, because of the way the mushrooms
# are distributed, some of the dummy variables are redundant
# in the presence of the other dummies; this turns out not to present
# a problem in the prediction process

p.hat.linear.model.all.predictors <-
  predict( linear.model.all.predictors, type = 'response' )
```

```
par( mfrow = c( 2, 1 ) )

hist( p.hat.linear.model.all.predictors[ raw.mushroom.data$poisonous == 0 ],
  nclass = 100, main = 'y = 0', xlab = 'p.hat', prob = T,
  xlim = c( 0, 1 ) )

hist( p.hat.linear.model.all.predictors[ raw.mushroom.data$poisonous == 1 ],
  nclass = 100, main = 'y = 1', xlab = 'p.hat', prob = T,
  xlim = c( 0, 1 ) )

par( mfrow = c( 1, 1 ) )

c(

 mean( p.hat.linear.model.all.predictors[ raw.mushroom.data$poisonous == 0 ] ),
 mean( p.hat.linear.model.all.predictors[ raw.mushroom.data$poisonous == 1 ] ),
 mean( p.hat.linear.model.all.predictors[ raw.mushroom.data$poisonous == 1 ] ) -
 mean( p.hat.linear.model.all.predictors[ raw.mushroom.data$poisonous == 0 ] )

 )

# [1] 2.981711e-13 1.000000e+00 1.000000e+00

############### the fourth block of code ends here #################

############### the fifth block of code begins here ################

# the next three function calls (a) load a CRAN package we need called 'DAAG'
# and (b) run one of the 'DAAG' functions

dynamic.require <- function( package ) {

  if ( eval( parse( text = paste( 'require(', package, ')' ) ) ) ) {

    return( 'done' )

  }

  install.packages( package )

  return( eval( parse( text = paste( 'require(', package, ')' ) ) ) )

}

dynamic.require( 'DAAG' )

cross.validation.all.predictors.10.fold <- cv.lm(
  data = raw.mushroom.data, form.lm = formula( poisonous ~ . ),
```

```
  m = 10, printit = F )

warnings( )

# Warning messages:
# 1: In predict.lm(subs.lm, newdata = data[rows.out, ]) :
#   prediction from a rank-deficient fit may be misleading
# 2: In predict.lm(subs.lm, newdata = data[rows.out, ]) :
#   prediction from a rank-deficient fit may be misleading
# 3: In predict.lm(subs.lm, newdata = data[rows.out, ]) :
#   prediction from a rank-deficient fit may be misleading
# 4: In predict.lm(subs.lm, newdata = data[rows.out, ]) :
#   prediction from a rank-deficient fit may be misleading
# 5: In predict.lm(subs.lm, newdata = data[rows.out, ]) :
#   prediction from a rank-deficient fit may be misleading
# 6: In predict.lm(subs.lm, newdata = data[rows.out, ]) :
#   prediction from a rank-deficient fit may be misleading
# 7: In predict.lm(subs.lm, newdata = data[rows.out, ]) :
#   prediction from a rank-deficient fit may be misleading
# 8: In predict.lm(subs.lm, newdata = data[rows.out, ]) :
#   prediction from a rank-deficient fit may be misleading
# 9: In predict.lm(subs.lm, newdata = data[rows.out, ]) :
#   prediction from a rank-deficient fit may be misleading
# 10: In predict.lm(subs.lm, newdata = data[rows.out, ]) :
#   prediction from a rank-deficient fit may be misleading
# 11: In cv.lm(data = raw.mushroom.data, form.lm = formula(poisonous ~  ... :

#  As there is >1 explanatory variable, cross-validation
#  predicted values for a fold are not a linear function
#  of corresponding overall predicted values.  Lines that
#  are shown for the different folds are approximate

# the next function call computes the cross-validated
# root-mean-squared error of the regression predictions,
# which is essentially the same as the value
# from the regression with the full data set

sqrt( attr( cross.validation.all.predictors.10.fold, 'ms' ) )

# [1] 2.199488e-13

############### the fifth block of code ends here ###################

############### the sixth block of code begins here #################

# the next function call investigates which predictors can be dropped
# from the model with little or no predictive accuracy loss
```

```
null.model <- lm( poisonous ~ 1, data = raw.mushroom.data )

full.model <- lm( poisonous ~ ., data = raw.mushroom.data )

variable.selection.with.bic <- step( null.model,
  scope = list( lower = null.model, upper = full.model ),
  direction = 'forward', trace = 1, steps = 1000, k = log( n ) )

# the output of this function call is in the file

#   stat-206-mushroom-analysis-variable-selection-with-bic.txt

# open this file and study its contents

############### the sixth block of code ends here ###################

############# the seventh block of code begins here ################

summary( linear.model.just.odor <-
  lm( poisonous ~ odor, data = raw.mushroom.data ) )

# Call:
# lm(formula = poisonous ~ odor, data = raw.mushroom.data)

# Residuals:
#    Min     1Q Median     3Q    Max
# -0.034 -0.034  0.000  0.000  0.966

# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept) 1.37e-14   5.98e-03    0.00        1
# odorc       1.00e+00   1.05e-02   95.30  < 2e-16 ***
# odorf       1.00e+00   6.51e-03  153.71  < 2e-16 ***
# odorl       9.37e-14   8.45e-03    0.00        1
# odorm       1.00e+00   2.08e-02   48.08  < 2e-16 ***
# odorn       3.40e-02   6.31e-03    5.39  7.1e-08 ***
# odorp       1.00e+00   9.57e-03  104.54  < 2e-16 ***
# odors       1.00e+00   7.78e-03  128.55  < 2e-16 ***
# odory       1.00e+00   7.78e-03  128.55  < 2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Residual standard error: 0.12 on 8115 degrees of freedom
# Multiple R-squared:  0.943,     Adjusted R-squared:  0.943
# F-statistic: 1.67e+04 on 8 and 8115 DF,  p-value: <2e-16

p.hat.linear.model.just.odor <-
  predict( linear.model.just.odor, type = 'response' )
```

```
par( mfrow = c( 2, 1 ) )

hist( p.hat.linear.model.just.odor[ raw.mushroom.data$poisonous == 0 ],
  nclass = 100, main = 'y = 0', xlab = 'p.hat', prob = T,
  xlim = c( 0, 1 ) )

hist( p.hat.linear.model.just.odor[ raw.mushroom.data$poisonous == 1 ],
  nclass = 100, main = 'y = 1', xlab = 'p.hat', prob = T,
  xlim = c( 0, 1 ) )

par( mfrow = c( 1, 1 ) )

table( p.hat.linear.model.just.odor, useNA = 'always' )

# p.hat.linear.model.just.odor
# 1.36725100485403e-14 1.07412118907621e-13   0.0340136054422548    0.999999999999688
#                  400                  400                 3528                  192
#    0.999999999999851     1.00000000000006     1.00000000000009     1.00000000000016
#                 2160                   36                  576                  576
#     1.00000000000018                 <NA>
#                  256                    0

app.says.poisonous.0.05 <- ifelse( p.hat.linear.model.just.odor > 0.05, 1, 0 )

with( raw.mushroom.data, table( app.says.poisonous.0.05, poisonous ) )

#                       poisonous
# app.says.poisonous.0.05    0    1
#                       0 4208  120
#                       1    0 3796

app.says.poisonous.0.01 <- ifelse( p.hat.linear.model.just.odor > 0.01, 1, 0 )

with( raw.mushroom.data, table( app.says.poisonous.0.01, poisonous ) )

#                       poisonous
# app.says.poisonous.0.01    0    1
#                       0  800    0
#                       1 3408 3916

# looking at the best decision rules based on odor and spore print color:

summary( linear.model.odor.spore.print.color <-
  lm( poisonous ~ odor + spore.print.color, data = raw.mushroom.data ) )

p.hat.linear.model.odor.spore.print.color <-
  predict( linear.model.odor.spore.print.color, type = 'response' )
```

```
par( mfrow = c( 2, 1 ) )

hist( p.hat.linear.model.odor.spore.print.color[ raw.mushroom.data$poisonous == 0 ],
  nclass = 100, main = 'y = 0', xlab = 'p.hat', prob = T,
  xlim = c( 0, 1 ) )

hist( p.hat.linear.model.odor.spore.print.color[ raw.mushroom.data$poisonous == 1 ],
  nclass = 100, main = 'y = 1', xlab = 'p.hat', prob = T,
  xlim = c( 0, 1 ) )

par( mfrow = c( 1, 1 ) )

table( p.hat.linear.model.odor.spore.print.color, useNA = 'always' )

# p.hat.linear.model.odor.spore.print.color
# -2.38704001323558e-14  6.99159606210441e-15  1.71080299696395e-14
#                    24                   176                   200
#  6.71763024013884e-14  6.85040678851202e-14  7.00592314937111e-14
#                    24                    48                    48
#  7.00838615098795e-14  7.14192263879391e-14  8.15356602954742e-14
#                    48                  1296                  1344
#  9.80382985958486e-14  1.08154732503384e-13    0.0646108663730507
#                   176                   200                    48
#    0.0719530102790713     0.998042094958255     0.999999999999684
#                   624                  1584                    96
#     0.999999999999694      1.00000000000006      1.00000000000007
#                    96                    36                    72
#      1.00000000000009      1.00000000000016      1.00000000000018
#                   576                   576                   128
#      1.00000000000019      1.00538423886428                  <NA>
#                   128                   576                     0

odor.spore.print.color.app.says.poisonous.0.05 <-
  ifelse( p.hat.linear.model.odor.spore.print.color > 0.05, 1, 0 )

with( raw.mushroom.data, table( odor.spore.print.color.app.says.poisonous.0.05,
  poisonous ) )

#                                              poisonous
# odor.spore.print.color.app.says.poisonous.0.05    0    1
#                                              0 3584    0
#                                              1  624 3916

odor.spore.print.color.app.says.poisonous.0.07 <-
  ifelse( p.hat.linear.model.odor.spore.print.color > 0.07, 1, 0 )

with( raw.mushroom.data, table( odor.spore.print.color.app.says.poisonous.0.07,
```

```
  poisonous ) )

#                                                 poisonous
# odor.spore.print.color.app.says.poisonous.0.07    0    1
#                                                 0 3632    0
#                                                 1  576 3916

odor.spore.print.color.app.says.poisonous.0.08 <-
  ifelse( p.hat.linear.model.odor.spore.print.color > 0.08, 1, 0 )

with( raw.mushroom.data, table( odor.spore.print.color.app.says.poisonous.0.08,
  poisonous ) )

#                                                 poisonous
# odor.spore.print.color.app.says.poisonous.0.08    0    1
#                                                 0 4208   48
#                                                 1    0 3868

############### the seventh block of code ends here #################
```